QUALIFYING EXAMINATIONS IN COMPUTER SCIENCE, 19654978

edited by

Frank M. Liang

STAN-CS-79-730
April 1979

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

QUALIFYING EXAMINATIONS IN COMPUTER SCIENCE 1965 - 1978

by
the Stanford University
Computer Science Department

edited by
Frank M. Liang

Abstract

Since 1965, the Stanford Computer Science Department has periodically given "qualifying examinations" as one of the requirements of its graduate program. These examinations are given in each of six subareas of computer science: Programming Languages and Systems, Artificial Intelligence, Numerical Analysis, Computer Design, Theory of Computation, and Analysis of Algorithms. This report presents the questions from these examinations, and also the associated reading lists.

The preparation of this report has been supported in part by NSF grant MCS 77-23738 and in part by IBM Corporation.

Foreword

This report complements the collection of "Comprehensive Examinations in Computer Science, 1972-1978" published last fall as Stanford Computer Science Report CS-677; it contains most of our department's qualifying examinations since they were first given in 1965.

Originally each student was required to pass the "Systems Qual" plus two other area quals of his or her choice. These quals were usually written exams that lasted 3 or 4 hours. Since 1972 we have changed the policy: now the requirement is to pass a "Comprehensive Exam" plus only one of the area qualifying exams. Because of the fewer number of students taking each qual, they are now often given orally; some of these exams are not included in this report.

Since these examinations go back to the earliest days of computer science education, they have considerable historical value. Can the computer scientists of 1979 solve the problems of 1965 more easily or less easily than the students of 19657

But besides this obvious historical value, the questions in many cases still have considerable relevance and interest; in fact, a lot of nice results appear on these pages, heretofore unpublished. (For example, see the 1970 Systems Qual, question 4, or the 1971 Systems Qual, question 2.) I believe every computer scientist will gain much from browsing in this book.

Unfortunately we do not have written answers to most of these exams, so the reader is on his own. We have, however, included answers to the four take-home qualifying examinations in Analysis of Algorithms; if I may take the liberty to say so, these examinations and answers have particular interest, since they represent subject matter that is taught somewhat differently at Stanford than at most other departments of computer science.

The scope of the exams was roughly defined by reading lists that were given out periodically in each area. The references from these lists are included at the end of the report, along with the dates of the reading lists in which they appeared.

As with report CS-677, Frank Liang deserves enormous praise for his labors in collecting and editing this material.

Good reading!

D. E. Knuth
March 1979

Table of Contents

May 1965 Systems Qualifying Exam

1. Programming.

Construct an algorithm to determine the shortest path between any two points of a network, under the constraint that this path consists of no more than a given number of legs, where a leg is the direct path between two points. Note that the direct path between two points need not be the shortest path.

Write the algorithm as an ALGOL procedure for the B5500 computer, using the following procedure heading:

```
PROCEDURE SHORTESTPATH (A, B, N, L);
VALUE N, L; INTEGER N, L; ARRAY A, B [1,1];
```

N denotes the number of points in the network. A is the N×N matrix such that A[I, J] denotes the distance of the direct path between points with indices I and J. The procedure must not alter A. B is to represent the resulting N×N connection matrix whose elements B[I, J] are the shortest paths between points I and J with a number of legs $\leq$ L.

Note: Write your program on the coding paper supplied. It will be keypunched and tested on the computer.

2. Algebraic linguistics.

Let G $\langle V, T, P, S\rangle$ be a simple phrase structure grammar, where V is the vocabulary, T the terminal vocabulary, P the set of productions, and $S \in V$. Suppose all productions have the form

$U \rightarrow x$

where $U \in$ V-T, $x \in V^*$, and $U \neq x$. (V^* is the set of all strings composed of symbols of V, including the empty string ϵ.

It is always possible to construct an equivalent grammar $G'\langle V, T, P', S\rangle$, such that

$L(G') = L(G) - \{\epsilon\}$

and such that there is no production in P' of the form

$U \rightarrow \epsilon$.

(a) For the following two examples find equivalent grammars with the properties described above.

(1) $S \rightarrow A$
$A \rightarrow aA$
$A \rightarrow Ab$
$A \rightarrow \epsilon$

(2) <procedure heading> ::=
    <procedure identifier> <formal parameter part>;
    <value part> <specification part>
  <formal parameter part> ::=
    <empty> | (<formal parameter list>)
  <value part>::=
    <empty> | **value** <identifier list>;
  <specification part> ::=
    <empty> | <specifier> <identifier list>; |
    <specification part> <specifier> <identifier list>

(b) Why should a compiler designer be interested in finding such equivalent grammars?

(c) Describe a general method for constructing the discussed equivalent grammar.

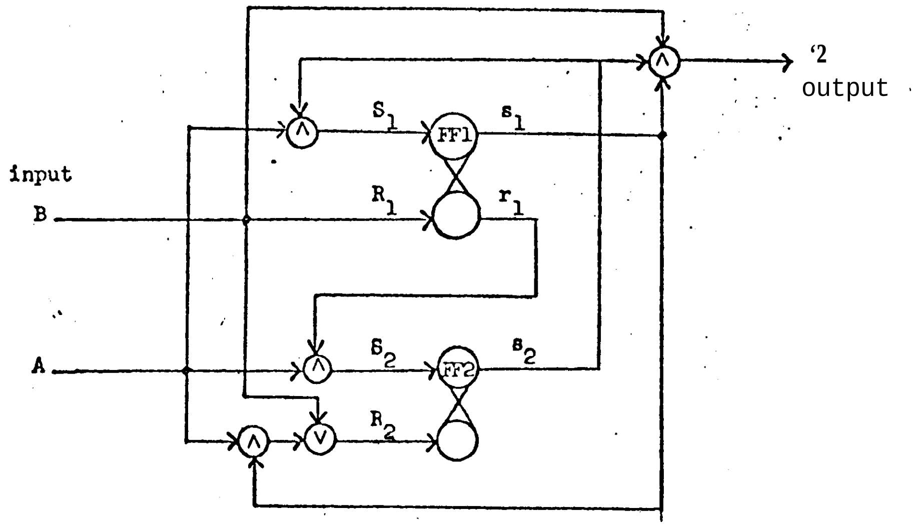
3. <u>Programming systems.</u>

Construct an algorithm in ALGOL 60 to translate an arithmetic expression of ALGOL 60 into an equivalent reverse-polish string of operands and operators. For input and output use the primitive procedures “insymbol” and “outsymbol”, and assume that identifiers and numbers are represented by the symbol λ. Do not consider the occurrence of conditional expressions and function designators.

The ALGOL report and the Report on Input-Output procedures “insymbol” and “outsymbol” are available for consultation.

4. <u>Switching circuit theory.</u>

A diagram is given below of a simple sequential circuit. The input variables *A* and *B* are “pulse” variables, that is, “no pulse” means *false* and “pulse” means true. Input pulses appear on only one lead at a time and are separated by a time interval greater than the resolution time of the circuit. The flip flops are set-reset. (A pulse on *R* sets *r* to true and *s* to *false*, and a pulse on *S* sets *s* to true and *r* to *false*.



(a) Draw a state diagram (transition graph) which represents the above circuitry, showing the relationship between input pulses and state transitions.

(b) Give a short description of the relationship between input pulses and output pulses.

5. Storage and storage access.

(a) It is possible to construct a random access memory out of a variety of materials and components. The majority of basic organizational principles and problems of random access memories are common to this wide range of technologies.

Using diagrams of memory elements and structures (including wiring) constructed from a technology with which you are familiar, discuss the following:

(1) The principles of operation of the memory element you have chosen.
(2) The basic organizational principles and problems of a random access memory organized at the bit level.
(3) The basic organizational principles and problems of a random access memory organized at the multiple-bit (word) level.
(4) The relative merits and demerits of the above two organizations.
(5) The major factors limiting memory speed.
(6) Other memory elements you know of (just list, you need not explain).

(b) There are a growing number of situations where it may not be possible or desirable to maintain a user's entire program in the "fast" main store of the computer and therefore it becomes essential to make efficient use of a fast main store-secondary storage device combination, such that machine speed is not degraded unnecessarily and the user can program as if he had direct access to a single-storage device. Schemes to accomplish the above go under various names, such as one-level-addressing, paging, and program segmentation. Several such schemes have been proposed; some which you may be familiar with are the techniques adopted by the designers of the Atlas, IBM 360, and B5000. Using appropriate block diagrams of one of the schemes above or any other you are familiar with, discuss the following:

(1) The basic features and problems of such a memory organization.
(2) A method of protecting the memory so that one user cannot enter another's program.

(c) Access to the contents of a memory location may be gained from the information contained in a data word either by special hardware or by program. If the access is achieved by a search technique by hardware, the term "content-addressable memory" is often used. Discuss the organization of such a memory.

If the address is generated by the program as a function of the data, the term "hash coding" is sometimes used. Discuss such a scheme and the type of situation in which it is useful.

(d) Machines such as the KDF9 and B5000 make extensive use of what is often called implicit or zero-addressing (although all other machines also use such a scheme to some extent). Discuss briefly what is meant by such an addressing scheme and the storage device organization of a machine such as the B5000 or KDF9 which allow its extensive use.

6. Machine organization.

Although there are many differences in the way in which different computers are organized, there are, nonetheless, certain common features. In particular the central processor of a machine must contain or have access to certain registers for holding instructions and data while carrying out its functions. This question deals with the dynamics of moving data into and out of the registers of a machine, called MACHINE, described below.

Although the MACHINE is not completely specified, the parts essential to the question are described. In fact, some of the descriptive material is not needed for this question.

It is possible to carry out the following fixed-point arithmetic, fetch, and store operations with two machine instructions:

(1) $Ac \leftarrow C$
(2) $B \leftarrow (C + B) \times A$

where initially C is located at the symbolic address M(C), B is at M(6), and A is at M(A); Ac is one of the arithmetic registers described below, In executing these two instructions the MACHINE must go through a sequence of events involving register clears and memory cycles. There are four types of cycles through which the machine may go; these are described below.

Problem: Fill in the table on the next page showing the contents of each of the three control registers, the three arithmetic registers, and the memory register at each stage of execution of the operations indicated by (1) and (2) above. The table provides for all machine cycles whether called for or not and leaves space for memory fetches. Draw a line through any of the cycles not called for.

Assume that the first instruction resides in the actual memory location $(1010)_{10}$ and has been fetched to the Function Register to start the operations. When filling in the address portions of the Function Register you may use the symbolic addresses M(C), M(B), and M(A). You may use decimal addresses where actual addresses are called for. Assume A, B, and C are full register fixed point numbers. You may use $C(XXXX)_{10}$ to indicate content of memory location at decimal XXXX. Mark irrelevant contents by a dash. Let $F_{20} = 8$ throughout.

Control Registers:	FR, DR, CC
Arithmetic Registers:	Ac, Qt, P
Memory Register:	MR

	MR					FR						DR	CC	Ac	Qt	P
		S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9					
Initial	$C(1010)_{10}$															
P cl Preliminary Cycle																
Operand Cycle																
Result Cycle																
Fetch Cycle																
P cl Preliminary Cycle																
Operand Cycle																
Result Cycle																
Fetch Cycle																

The MACHINE

BITS	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31	32 35	36 39
	B-ADDRESS			ORDER		TAG		A-ADDRESS		
SIGITS	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S

SIGIT = SEXADECIMAL DIGIT (base 16)

Word structure for instructions.

When a MACHINE word *is used* as an *instruction,* it consists of two 3-sigit addresses, called the *B-address* (bits 0-11) and the *A-address* (bits 28-39), a P-sigit order portion (bits 12-19), and a 2-sigit *tag* portion (bits 20-27). The order and tag portions specify precisely what operations are to be performed when this instruction is obeyed. In this sense, the order-tag portion is an operation code. Each instruction is brought into the Function Register (FR) prior to being executed. Consequently, the bits of an instruction are referred to as F_0 through F_{39}.

Among the common uses of the A-address are:

(1) Specifying the memory location of the multiplicand in multiplication, the divisor in division, or of the subtrahend in subtraction.

(2) Specifying the memory location of the addend in one type of add operation.

(3) Specifying the number of shifts in a right or left shift operation.

(4) Specifying the memory location into which results are to be stored in some types of store operations.

(5) Specifying the memory location from which the next instruction is to be obtained in some jump operations.

(6) Specifying the first memory location involved in the transfer of data in input-output operations.

Among the common uses of the B-address are:

(1) Specifying the memory location of the addend in one type of add operation (preliminary add).

(2) Specifying the memory location from which the next instruction is to be obtained in one type of jump operation (jump to 8).

(3) Specifying the memory location to be used as an index register (B-box).

(4) Specifying the memory location into which results are to be stored in one type of store operation (results to B).

MAY 1965

The Control Registers

Function Register (FR)

The 40-bit Function Register contains the current instruction being executed. The FR is treated as four registers, FRB, FRO, FRT, and FRA, holding the effective B-address, the order, the tag, and the unindexed A-address, respectively.

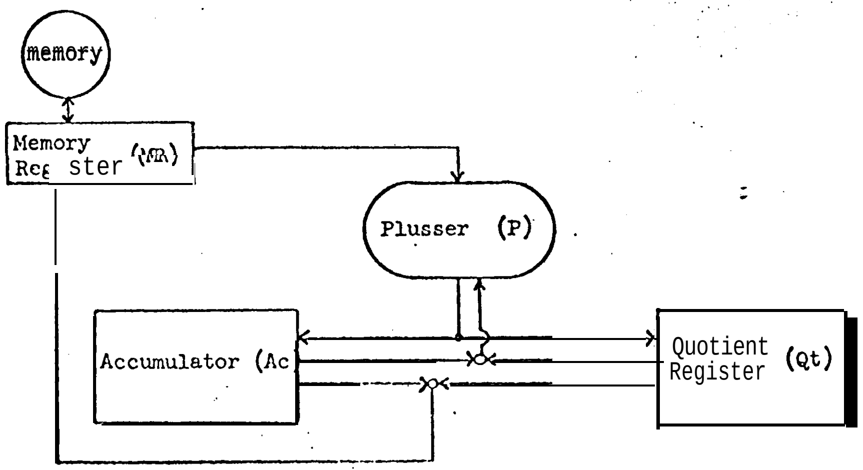
Dispatch Register (DR)

The Dispatch Register is a 12-bit register used to hold the address of the most recent word consulted in any memory cycle of an instruction.

Control Counter (CC)

The Control Counter, made up of 12 bits, holds 1 + the address of the instruction currently in FR. It is normally increased by 1 at the end of the last cycle of an instruction, i.e. at the end of the Fetch Cycle which brings in the next instruction. Jumps are treated specially but need not be considered herein.

REGISTERS OF IMPORTANCE TO PROGRAMMER IN UNDERSTANDING MACHINE ARITHMETIC OPERATIONS



Simplified Data-flow Diagram for MACHINE Arithmetic

Memory Register (MR)

The Memory Register is a 40-bit register which acts as a buffer unit, or transfer register, between the magnetic core storage and the arithmetic and control units.

It is always the contents of MR that are sent to the memory when a Store or Input instruction is executed, with the word(s) to be stored always initially sent to MR (one word at a time), then to the memory. Likewise, each word recalled from the memory for use by the control, arithmetic, or output unit is always sent first to MR, and from there to the unit or register. This word is not addressable. However, several instructions are available which make use of the current contents of MR.

Accumulator and Quotient Registers (Ac and Qt)

All arithmetic operations and most store orders on the MACHINE require use of either the Accumulator, the Quotient Register, or both. These registers are symmetric in all arithmetic operations except multiplications and division, as well as in all store orders except β orders (not considered herein). In general, one or two bits in the instruction determine which register will be used.

Ac and Qt are 41-bit registers ($a_x a_0 a_1 \ldots a_{39}$ and $q_x q_0 \ldots q_{39}$, respectively). As arithmetic registers, either can be used to hold a summand and/or the sum. In multiplication, Qt holds the multiplier, MR the multiplicand, and the product, a 79-bit number, is left in Ac Qt, with the 40 most significant bits in Ac ($a_0 \ldots a_{39}$). Also, 2^{-39} times the original contents of Ac is added to the product. In division, Ac and Qt together form a 79-bit dividend ($a_0 \ldots a_{39} q_0 \ldots q_{38}$), MR holds the divisor, Qt holds the quotient and Ac the remainder. Ac or Qt may be used as a transfer register (to move a word from one memory location to another) by adding the word to a cleared register (via the plusser) and storing the result.

Ptusser (P)

In all MACHINE additions, the contents of MR and the contents of either Ac or Qt are added in the Plusser and the sum returned to either Ac or Qt. The selection of Ac or Qt is determined by the particular instruction being executed.

The Plusser is neither addressable nor usable at the programmer's discretion, but is used by the MACHINE in performing additions.

Execution of the MACHINE's instructions

The execution of any MACHINE order requires from one to four memory cycles:

(1) Preliminary Cycle (possibly preceded by preliminary clear)
(2) Operand Cycle
(3) Result Cycle
(4) Fetch Cycle

Every instruction involves a Fetch Cycle. The other three cycles occur when needed to implement the particular instruction.

Preliminary Clear (PC 1)

Bit 23 in the Tag designates either the accumulator (bit 23 = 0) or the Quotient Register (bit 23 = 1) as R1 (the register being used). At the beginning of an instruction, it is possible to clear R1 to zero, to 2^{-11}, or to 2^{-39} if desired. (The clears to 2^{-11} and 2^{-39} are useful when it is desired to increment by one the B or A portion of some memory location.) Which clear, if any, is to be used is governed by bits 21-22 of the Tag:

F_{21}	F_{22}	
8	8	do not clear R1
8	1	clear R1 to zero
1	8	clear R1 to 2^{-11}
1	1	clear R1 to 2^{-39}

The above table with the designated R1 may be written as follows:

F_{21}	F_{22}	F_{23}	R1	
8	8	8	Ac	no preliminary clear
8	8	1	Qt	no preliminary clear
8	1	8	Ac	clear Ac to zero
8	1	1	Qt	clear Qt to zero
1	8	8	Ac	clear Ac to 2” '
1	8	1	Qt	clear Qt to 2^{-11}
1	1	8	Ac	clear Ac to 2^{-39}
1	1	1	Qt	clear Qt to 2^{-39}

Preliminary Cycle (PC)

A preliminary cycle occurs whenever either of the following is requested:

(1) A Preliminary Add ($F_{25} = 1$)
(2) B-boxing ($F_{26} = 1$) (not needed in this discussion)

In either case, the following 40-bit number is brought into MR: the contents of memory location B, where B is the three sigit B-address portion of the instruction.

Preliminary Add: In the case of the preliminary add, the number which has just been brought into MR from memory is added to the contents of R1 (Ac if $F_{23} = 8$, Qt if $F_{23} = 1$), and the sum is placed back in R1.

F_{25}	
8	no preliminary add
1	C (R 1) + C(xyz) to Rq, where xyz is the B address of the instruction

The table for these operations may be written as follows:

F_{25}	F_{23}	R1	
8	8	Ac	no preliminary add
8	1	Qt	no preliminary add
1	8	Ac	C(Ac) + C(xyz) to Ac
1	1	Qt	C(Qt) + C(xyz) to Qt

Exceptions: Preliminary add with multiply: Qt ← C(R1) + C(xyz); with divide: Ac ← C(R1) + C(xyz).

Operand Cycle (OC) (additions page 12 and multiplications page 13)

The execution of the fundamental operations of most instructions occurs during the operand cycle. Only the Stop orders, the various Store orders, and the Jump orders omit the operand cycle, since these operations by their nature must occur during another cycle (e.g. store order — result cycle, jump order — fetch cycle). If the operand cycle uses the memory, the A address is consulted (or the A-effective address, if B-boxing is indicated).

Result Cycle (RC)

A result cycle writes the contents of Ac, Qt, or MR into the memory. If the contents of Ac or Qt are being stored, they are first sent to MR; then MR (all or part of it) is stored into the memory as prescribed in the instruction.

A result cycle may be obtained in one of two ways: (1) The particular order specified by the order portion of the instruction may involve a store into memory; or (2) the *results to B* option may be invoked by bit 24 in the Tag portion of the instruction. The former case is described under the individual orders involved.

Result to B: The F_{24} bit in the tag makes it possible to store the contents of one of the arithmetic registers, Ac or Qt, into the memory without using a separate instruction for this purpose.

Normally, $F_{24} = 1$ causes the contents of the register containing the result of the operation performed to be written into the memory at the location specified by the B-address portion of the instruction (or by the B-portion of the B-box if $F_{26} = 1$). F_{19} in the order indicates where the results is to be found, i.e. $F_{19} = 0$ implies that the result is in Ac; $F_{19} = 1$ implies that the result is in Qt.

Fetch Cycle (FC)

During the Fetch Cycle, the next instruction is brought into FR, ready to be executed. Thus, FC always uses the memory and must occur in each instruction. if no jump is involved, the address consulted comes from the control counter (CC) and hence is the address of the instruction located in the memory immediately after the current instruction.

Σ orders (Add, Subtract) $S_3 = 1$

S_3S_4	
18	R1 + a → Ac
11	R1 + a → Qt
12	R1 - a → Ac
13	R1 - a → Qt
14	R1 + |a| → Ac
15	R1 + |a| → Qt
16	R1 - |a| → Ac
17	R1 - |a| → Qt
18	RI + a → Ac,a
19	R1 + a → Qt,a
1A	R1 - a → Ac,a
1B	R1 - a → Qt,a
1C	R1 + |a| → Ac,a
1D	R1 + |a| → Qt,a
1E	R1 - |a| → Ac,a
1F	R1 - |a| → Qt,a

F_{23}	R1	F_{19}	R2
8	Ac	8	Ac
1	Qt	1	Qt

F_{21}	F_{22}	Prelim Clear
8	8	Hold RI
8	1	8 → R1
1	8	2^{-11} → R1
1	1	2^{-39} → R1

F_{25}	Prelim Add
8	none
1	R1+b → R1

F_{16}	F_{24}	Main Operation
8	8	R1 Op a → R2
8	1	RI Op a → R2,b
1	8	R1 Op a → R2,a
1	1	R1 Op a → R2,a

π orders $S_3 = 2$

Result to A		
no	yes	
S_3S_4	S_3S_4	
28	28	Full-precision unrounded multiply
21	29	Full-precision unrounded divide
22	2A	Full-precision rounded multiply
23	2B	Full-precision rounded divide
24	2C	Half-precision unrounded multiply
25	2D	Half-precision unrounded divide
26	2E	Half-precision rounded multiply
27	2F	Half-precision rounded divide

In full-precision multiplication, the multiplier (m) is assumed to be in Qt, the multiplicand (q) at the A-effective address. The result in Ac Qt after multiplication is $mq + 2^{-39}p + 2^{-79}q_0$ where p is the contents of Ac before multiplication and q_0 is the O-bit of the multiplicand. Note that the result is in $(a_0a_1 \ldots a_{39}q_0 \ldots q_{38})$ with q_{39} holding the sign-bit of the multiplier.

In all divide orders, the dividend is assumed to be the 79-place number in Ac Qt (= $a_0a_1 \ldots a_{39}q_0 \ldots q_{38}$). The divisor is at the A-effective address. After the completion of a full-precision division order, Qt holds the 40-bit quotient, and Ac holds the true remainder.

Preliminary addition with π orders: Normally, the result of a preliminary addition is sent to R1. Multiplication and division, however, deviate from this rule: In multiplication, the preliminary add is c(R1) + c(B) to Qt. In division, it is c(R1) + c(B) to Ac.

Result to A: On 28, 2A, 2C, and 2E orders, the most significant part of the product (Ac) is stored into memory at the A-effective address, replacing the multiplicand. On 29, 2B, 2D, and 2F orders, the quotient is stored into memory at the A-effective address, replacing the divisor.

DECODING TABLE

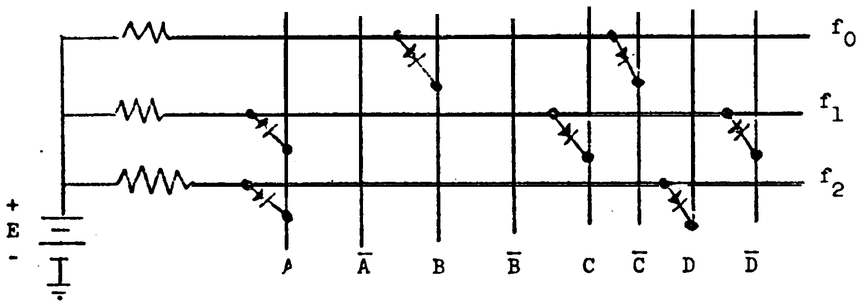
		ORDER					TAG							
		F_{12}-F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}	F_{26}	F_{27}
$^1/_0$	$\sum$	1	Result to A / Not	Mag / #	- / +	To Qt / To Ac	X	Pcl	Pcl	Cl or Q+M / Cl Ac Ac+M	Result to B / Not	P + / Not	B # / Not	J to B / Not
$^1/_0$	$\prod$	2	Result to A / Not	19 / 39	ROUND / Not	÷ / ×	X	Pcl	Pcl	cl Qt / cl Ac	Result to B / Not	P + / Not	B # / Not	J to B / Not
		S_3	S_4				:	S_5			S_6			

Pcl Lo:	F_{21}	F_{22}
Hold	0	0
0	0	1
2^{-11}	1	0
2^{-39}	1	1

March 1966 Systems Qualifying Exam

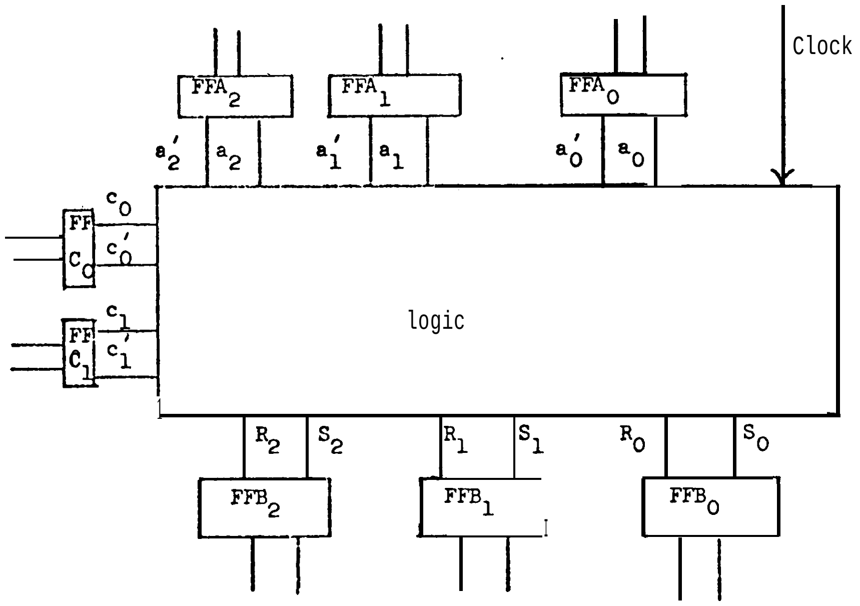
1. Logic circuits and switching theory.

(a)



Using the convention that +E = 1 and GROUND = 0, what functions are realized at f_0, f_1, and f_2?

(b)



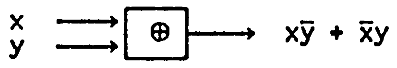
The box labeled "logic" is logic which will, depending on the state of flip-flops FFC_1 and FFC_0, transfer the contents of the 3-bit flip-flop register A into the 3-bit flip-flop register B either unchanged, complemented, left-shifted one place, or right-shifted one place when a signal appears on clock. The transfer is controlled by flip-flops FFC_0 and FFC_1 as given in the following table.

c_1	c_0	transfer
0	0	unchanged
8	1	complemented
1	0	left-shifted one bit
1	1	right-shifted one bit

The flip-flops of register B cannot be assumed cleared at the time of transfer and are set-reset flip-flops. On a left shift, flip-flop FFB_0 is reset; and on a right shift, flip-flop FFB_2 is reset.

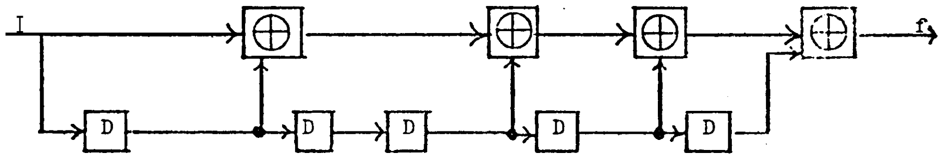
Write reasonably minimal logic equations for R_2, S_2, R_1 S_1, R_0, S_0 in terms of the input variables.

(c) The symbol below produces the exclusive-or function:



The symbol below indicates a unit delay

Given an input train of pulses on line I as shown in sequence S_1, what sequence is produced at f relative to the pulses on S_1? Assume no delay in the exclusive-or gates.



```
                   t1   t2   3   t4  t5   t6  t7  t8    t9
S1   0    0    1    0    0    0    0    0    0    0    0
```

2. <u>Machine organization.</u>

An algorithm for rapid binary multiplication is given below. You are not expected to have seen it before. Do not spend time figuring out why it works. Draw a block diagram for a multiplication unit which would perform this algorithm. The unit must have some correspondence with a unit that might actually be built. Show blocks for the storage registers, execution control logic. Show lines for the data flow.

(a) Describe the function each block performs. In the case of registers, specify their length, and in the case of blocks performing control operations, describe the control signals generated.

(b) Rewrite the algorithm in terms of the micro-operations of your unit.

An algorithm for binary multiplication of positive numbers using uniform shifts of two.

(1) Assume the multiplier is divided into two-bit groups, an extra zero being added to the high order end if necessary co produce an even number of bits.

(2) The multiplier will be scanned from the low order Co high order end.

(3) One addition or subtraction to or from the partial product will be made on each iteration for each group and, using the low order bit in the group as reference, this addition or subtraction will consist of either two or four times the multiplicand. These multiples of the multiplicand may be obtained by shifting the position of entry of the multiplicand one or two bits left of the reference position.

(4) The first cycle may require special handling as described below in (S).

(5) The general rule is that, following any addition or subtraction, the resulting partial product will be either correct or smaller than it should be.

(6) The multiple of the multiplicand to add or subtract from the partial product at a given time is determined by decoding the two bits of the current group and the low order bit of the next higher order group. The multiple IS given in the following table.

multiplier			operation	multiplier			operation
low order bit next high order group	current group		n xmultiplicand	low order bit next high order group	current group		n xmultiplicand
0	0	0	0	1	0	0	-4
0	0	1	+2	1	0	1	-2
0	1	0	+2	1	1	0	-2
0	1	1	+4	1	1	1	0

(7) Following the addition or subtraction, the partial product is shifted right two positions. This shift is a 2's complement right shift, assuming subtraction is accomplished by adding the 2's complement.

(8) On the first cycle, if the low order bit is a 1 enter the 2's complement of one times the multiplicand into the adder as well as the multiple determined by the table above.

The above can present a design problem if the 2's complement is formed by bringing the l's complement of the operand into the adder and adding a carry CO the low order bit. It is thus possible in this first cycle to require two low order carries. To avoid this problem the following dodge on the first cycle can be used. Always decode the low order bit as a zero and add the correct multiple from the above table and then use the true value of the low order bit co determine whether or not co add the true value of the multiplicand also.

Add if the low order bit is a 1. Do not add it if the low order bit is a. 0.

3. Supervisory programs.

(a) Describe in a few sentences the chief operating characteristics of (1) job-shop monitors, (2) time-sharing monitors, and (3) real-time monitors.

(b) (1) Diagram a transfer table co handle branching between program segments all of which are located in main storage. Use three segments as an example. Show how the transfer table is changed if one segment is relocated in core. Explain what must be done if some segments are stored on auxiliary file memory.

(2) Using diagrams, describe the paging and dynamic relocation scheme for some particular computer such as the ATLAS, SDS 940, GE 645, IBM 360/67, or CDC 3500.

(c) Consider a routine for scheduling tasks for processing by three resident programs, each at a different priority level. Let HP be the program with highest priority, for example a real-time control program. Let LP be a program with second highest priority, for example and on-line data analysis program. Let BG be the program with lowest priority, for example a background program. A program is permitted co complete each tasks and then it goes into a waiting loop until the scheduler provides a new message (or data) co initiate a new task. Construct a flow chart to show a scheduling routine for processing task sequences through these three programs.

4. Data channels.

(a) . What is the function of a data channel?

(b) Early computers did not have data channels. How were the functions now performed by data channels handled on early machines?

(c) Show schematically how a data channel is related Co other functional units of a computer. Show data paths by solid lines and control paths by dotted lines.

(d) What are some of the scheduling problems that arise in the use of a data channel (both ends)?

(e) What are the principal hardware features that are found in a data channel?

(f) What is a multiplexor channel? How does it differ from other channels?

. 5 . Procedure parameters in programming: languages.

Describe (1) the meaning and (2) the mechanism used to implement

(a) the FORTRAN subroutine parameter
(b) the ALGOL name parameter
(c) the ALGOL value parameter.

Compare and evaluate the three kinds of parameters in the light of their usefulness and the complexity of the required mechanism.

6. Syntax, ambiguity, and precedence relations.

(a) Give a formal definition of each of the following terms: (1) language, (2) phrase structure system, and (3) ambiguity.

(b) Consider the syntax with the following production rules:

$S ::= A$
$A ::= B \mid AXB \mid CXA$
$B ::= D \mid DYB$
$C ::= CYD \mid D$
$D ::= UAV \mid Z$

What language is generated? Is the system ambiguous? Give all possible parses of the following string:

$ZYZXZYZXZYZ$

(c) Define a phrase structure system which generates the same language and which is unambiguous.

(d) Define your phrase structure system in part (c) such that it is a simple precedence syntax. Indicate the precedence relations between the symbol pairs in the form of a matrix.

7. "Critical reading of publications".

In a recent issue of the Communications of the ACM the following "ALGOL" program was published:

```
Boolean array b(0;1) integer k,i,j
comment This is the program for computer i, which may be
        either 0 or 1, computer j ≠ i is the other one;
C0: b(i) := false;
C1:     if k ≠ i then begin
c2:     if not b(j) then go to C2;
        else k := i; go to C 1 end;
        else critical section
        b(i) := true;
        remainder of program;
        go to co;
        end
```

(a) Find and correct the formal errors.

(b) It is claimed that if two computers execute this program concurrently, it is impossible for both of them to enter the procedure "critical section" at the same time. Verify or disprove this claim.

Note: The computers operate independently. It is assumed that in the procedure "remainder of program" no reference is made to the variables k and b, nor that any other "shared" variables exist.

October 1366 Systems Qualifying Exam

1. Programming languages.

(a) Characterize the principal differences between so-called list processing languages and algebraic languages like ALGOL and FORTRAN.

(b) Describe the LISP scheme in terms of an ALGOL *program.* In particular explain the basic functions CAR, CDR, and CONS in the form of ALGOL procedures,

2. Syntax.

(a) Give a precise formal definition of a context-free phrase structure grammar. Explain the meaning of the attribute "context-free". Which part of your definition reflects this attribute?

(b) In connection with analysis of sentences (parsing), the term "bounded context" is often used. What does it mean, and how is this bound usually specified? What is the bound on the following two grammars?

(1) $A ::= B \mid C$ $\quad D ::= xy$
$B ::= DE$ $\quad E ::= zw$
$C ::= xFu$ $\quad F ::= yz$

(2) $A ::= B \mid yB \mid xC \mid C$ $\quad D ::= xy$
$B ::= D \mid BD$ $\quad E ::= yx$
$C ::= E \mid CE$

(c) Construct an alternative grammar for the language defined by the second example. This grammar must be unambiguous and have the property that it can be analyzed from left to right without ever looking ahead more than one symbol.

3. Machine organization.

(a) Outline the essential characteristics of a "Von Neumann machine".

(b) Briefly explain the following statement: "The fundamental design of a Von Neumann machine was dictated more by technological necessity than by logical necessity."

Outline briefly (1) the areas in which these technological constraints have been relaxed since 1946, and (2) the variations in organization which are plausible as a consequence.

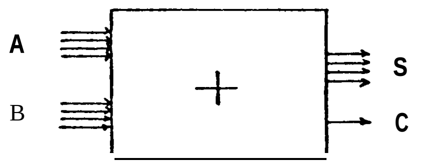
(c) Since Von Neumann (with Burks and Goldstine) published the description of the proposed Institute for Advanced Study computer in 1946, our concepts of logical organization for the central computer have slowly evolved. This evolution is illustrated by the IBM 701-7090 series, since the 701 was very close to Von Neumann's 1946 ideas. List at least three significant aspects of the 7090 which did not appear in Von Neumann's original conception (or in the 701).

4. Logical design..

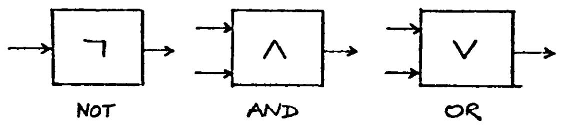
(a) Define each of the following representations for decimal digits. Each has been used in one or more automatic digital computers.

(1) binary coded decimal (l-2-4-8)
(2) l-2-2-4
(3) biquinary
(4) excess-3
(5) 2-out-of-5

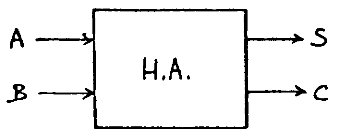
(b) Design a one-decimal-digit adder for the l-2-4-8 representation. This adder should accept two input digits A and B, and output a sum digit S and a carry digit C.



Present your solution as a block diagram using any or ail of the-following elementary logical box es:



We suggest that you introduce a binary half-adder:



$a + b = 2C + S$

as an intermediate step. Be sure to give the logical design of any such building-block circuits you use.

5. Allocation and scheduling.

Discuss in quantitative detail one of the two following questions.

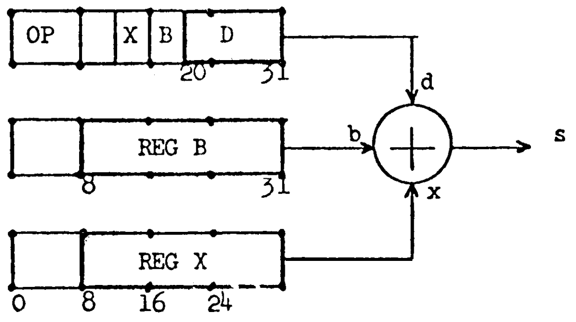
(a) Give the algorithm for dynamic storage allocation in some system which you know (for example, the B5500, 360/67, 360/91, or ATLAS). Explain what hardware features (e.g. stacks, registers, and tables) are utilized to implement this algorithm and how they are utilized. Also explain what special features in the software are utilized to implement this algorithm.

(b) Discuss the problem of optimization of register allocation (either index or base registers). Describe an algorithm for optimization of register allocation at compile time. How does this method compare with any other method you might know?

March 1967 Systems Qualifying Exam

1. Logical circuit design.

Certain instructions of the IBM 360 computer contain an address with two index fields. In order to form the effective address, the contents of two general registers are added to the instruction address part (called displacement).



$s = d + x + b$

Problem: Design a parallel adder with three inputs yielding the required effective address.

Step 1. Consider bit positions 20-3 1. Design the adder as a series of one-bit-position adders. Which are the inputs and which are the outputs of this circuit? Derive the Boolean expressions defining the outputs in terms of the inputs. Explain clearly your method of derivation.

Step -2. Note that the instruction address only contributes to positions 20-31. No triple adder is needed for positions 8-19. Repeat Step 1 for the design of a one-bit-position circuit to be used in positions 8- 19.

Step 3. Draw a diagram showing the interconnections of the various one-bit-position circuits.

2. Languages and parsing.

(a) Give a concise formal definition of each of the following terms:

(1) phrase structure language
(2) context free language
(3) regular (or left or right linear) language
(4) ambiguity

Define precisely the symbols used in your notation, and all intermediate notions you introduce.

(b) Define an unambiguous syntax which generates expressions composed of the symbols (denoting operands), ⊗, ⊕, and & denoting binary operators. Use BNF notation. Allow for parenthetical grouping of subexpressions using the symbols (and). Observe the following rules:

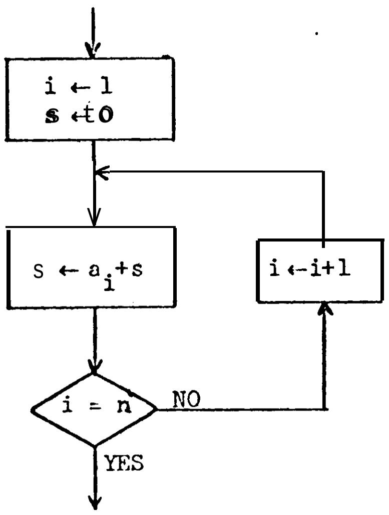
⊗ has precedence over ⊕ and &
⊕ has precedence over &

(c) Describe a parsing method applicable to your grammar defined in part (b). Either write a parsing algorithm, or, if your method depends on tables, describe the method and list the table values to be used for your grammar.

(d) Is it possible to define the expressions in part (b) in terms of a linear grammar? Explain briefly.

3. Programming.

It is claimed that the following program computes $s = \sum_{i=1}^{n} a_i$, if initially n is a positive integer and $a_1, \ldots, a_n$ are real numbers.



Give a convincing proof of this claim.

4. Programming systems.

Describe the principles of a storage allocation scheme used to implement Algol. How are variables addressed and accessed? Indicate the motivations and reasons for the methods you describe. Explain which ones are essential, and which ones are merely helpful in increasing efficiency.

January 1968 Systems Qualifying Exam

1. Syntax. (35 *points)*

Read the whole problem before answering any part of it.

(a) Describe a parsing algorithm which will parse sentences of a subset of context-free grammars using one of the following techniques: precedence, operator precedence, extended precedence, bounded context, transition matrices, production language, direct reduction, LR(k).

(b) Consider the following grammar:

```
<P>   ::= ⊥ <S> ⊥
<IC>  ::= IF EXP THEN
<S>   ::= <IC> <S>
<S>   ::= <IC> S1 ELSE <S>
<S>   ::= <IC> S1 ELSE S1 UNLESS EXP
<S>   ::= S1
```

Terminals	⊥, IF, EXP, THEN, ELSE, UNLESS, S1
Nonterminals	<P>, <IC>, <S>
Start symbol	<P>

Is this grammar acceptable to the parsing algorithm you described in part (a)? If not, change it so that it is acceptable, without altering the language and with minimal changes in the structure of the sentences. Then build the necessary tables and subroutines from the grammar for your parsing algorithm.

Answer one of parts (c) and (d).

(c) Consider the problem of compiling code for the language described in part (b). The meaning of all constructs except "UNLESS" should be clear; the conditional statement

<IC> $S1_1$ ELSE $S1_2$ UNLESS EXP

is equivalent to

<IC> $S1_1$ ELSE IF ¬EXP THEN $S1_2$

Augment your parser with "semantic routines" to generate the test and transfer instructions for the language. (You may use the grammar produced in part (b).) Specify precisely where each routine would be called and describe carefully what it would do. Generate only the following instructions:

JMP A	(Jump to label A)
CJMP A,EXP	(Jump to label A if EXP is true)
A:	(the label A itself)
S1	(statement S 1)

(d) (1) Give the formal definition for the class of grammars whose sentences can be parsed by the parser described in part (a).

(2) Define ambiguity for a grammar and for a language.

(3) Exhibit a nonambiguous context-free grammar which is not in the class defined in question (1) above.

(4) Exhibit a grammar in the class which is not a finite-state grammar.

2. Operating systems. (25 *points)*

(a) Consider the three stages of development of operating systems,

(1) What did the first primitive operating systems (e.g. FORTRAN MONITOR SYSTEM) do for the user?

(2) What characterizes the functions of the next phase of operating systems (e.g. IBSYS)?

(3) What are the key features of a multiprogramming operating system (e.g. OS/360)?

Answer one of parts (b) and (c).

(b) Describe a functioning operating system, using diagrams where appropriate. Include a discussion of both function and implementation.

(c) What analytical tools are being used to attack the problems of multiprogramming and time-sharing operating systems? Discuss at least one such analytical study in terms of model, technique, and conclusions of the study. You may choose one of the papers listed below or some other study with which you are familiar.

Denning, P. J., "Memory Allocation in Multiprogrammed Computers", MIT Project MAC Computation Structures Group Memo 24, March 1966.

Gaver, D. P., Jr., "Probability Models for Multiprogramming Computer Systems", *JACM 14*, July 1967, p. 423.

Horwitt, L. P., Karp, R. M., Miller, R. E., and Winograd, S., "Index Register Allocation", *JACM* 13, January 1966, p. 43.

Karp, Richard M. and Miller, Raymond E., "Properties of a Model for Parallel Computations", *SIAM Journal on Applied Mathematics* 14, November 1966, p. 1390.

Kleinrock, Leonard, "Time-shared Systems: A Theoretical Treatment", *JACM* 14, April 1967, p. 242.

Krishnamoorthi, B. and Wood, Rodger C., "Time-shared Computer Operations with both Interarrival and Service Times Exponential", *JACM 13,* July 1966, p. 317.

Nielsen, Norman R., “A Simulation of Time-Sharing Systems”, *CACM* 10, July 1967, p. 397.

Ramamoarthy, C. V., “The Analytical Design of a Dynamic Look-Ahead and Program-Segmenting System for Multiprogrammed Computers”, *Proceedings 21st National Conference of the ACM, 1966.*

3. Logical design. *(15 points)*

Answer one of the following two parts.

(a) The three most common fixed-point number representations are one’s complement, two’s complement, and sign-magnitude.

(1) Give the bit pattern in each representation of the quantities +25 and -62, assuming a word length of 12 bits.

(2) Give a flow chart or other brief description of addition in each representation, noting how overflow is detected and how the sign of the result is determined.

(3) State briefly the advantages and disadvantages of each representation, both from the viewpoints of hardware design and ease of programming use.

(b) Draw a circuit diagram for a time pulse distributer. The circuit has one input which receives a series of clock pulses. Whenever a pulse appears at the input, a pulse is to occur on one of four output leads in the following fashion: output lead 1 should have an output pulse with the 1 st, 5 th, 9 th, . . . input pulses; output lead 2 should have an output pulse with the 2nd, 6th, 10th, . . . input pulses, and so on.

4. Stack administration. (25 *points)*

On a stack oriented machine (or interpreter) the stack may used as a repository for various kinds of program information, such as those listed below.

(1) Storage for simple variables local to a scope (an Algol begin-end block, for instance).
(2) Return address for procedure or subroutine calls.
(3) Intermediate values obtained during expression evaluation.
(4) Pointers for locating variables local to enclosing scopes.

Describe a single stack structure containing all of this information in a sufficiently general form to be usable for an implementation of Algol 60.

Describe the action of your stack on procedure entry and procedure exit. One way to do this is to draw before-and-after diagrams.

May 1968 Systems Qualifying Exam

1. Syntax. (35 *points)*

Read the whole problem before answering any part of it.

(a) Describe a parsing algorithm which will parse sentences of a subset of context-free grammars using *one* of the following techniques: precedence, operator precedence, extended precedence, bounded context, transition matrices, production language, direct reduction, LR(k), or some other similarly powerful method.

(b) Consider the following grammar:

$\langle p \rangle \quad ::= \perp \langle be \rangle \perp$
$\langle be \rangle \quad ::= b \leftarrow \langle be \rangle \mid \langle ae \rangle = \langle ae \rangle$
$\langle ae \rangle \quad ::= a \leftarrow \langle ae \rangle \mid \langle at \rangle$
$\langle at \rangle \quad ::= \langle at \rangle - \langle ap \rangle \mid \langle ap \rangle$
$\langle ap \rangle \quad ::= a \mid (\langle ae \rangle) \mid *(\langle be \rangle)$

where

$\langle be \rangle$ is a Boolean expression,
$\langle ae \rangle$ is an arithmetic expression,
$\langle at \rangle$ is an arithmetic term,
$\langle ap \rangle$ is an arithmetic primary,
b is a Boolean identifier,
a is an arithmetic identifier.

To evaluate a Boolean expression $b \leftarrow \langle be \rangle$, one evaluates $\langle be \rangle$ and assigns the result to b. The value of the Boolean expression is then the value of b. Similarly for the arithmetic expression a $\leftarrow \langle ae \rangle$. The value of $*(\langle be \rangle)$ is 1 if $\langle be \rangle$ is true, 0 otherwise.

Is this grammar acceptable to the parsing algorithm you described in part (a)? If not, change it so that it is acceptable, without altering the language and with minimal changes in the structure of the sentences.

(c) Construct the necessary tables, subroutines, etc. for the parsing algorithm and grammar described in parts (a) and (b). Assume you are writing a compiler for student use and that therefore the syntactic error detection and recovery facilities are of first importance.

(d) Write the best error message which your scheme could produce for the following invalid programs.

(1) $\perp (a) \perp$
(2) $\perp$ a $- *(b)$ 1
(3) $\perp := := b := \perp$
(4) $\perp *(b) := a \perp$

(e) In parts (a) through (d) we assumed that there were distinct terminal symbols for arithmetic and Boolean identifier (a and *b*). Suppose now that the language includes *block structure* **and** *declarations* and that the following two productions were added.

b ::= <identifier>
a ::= <identifier>

How would you incorporate data-type handling thereby implied in the formal system used in parts (a)-(d)? What general property of formal parsing schemes does this point out?

2. Symbol table. (20 *points)*

Considering a symbol table to be a place where a translator stores identifiers and their associated attributes, present three alternative methods of organizing such a table. For each method, answer the following **questions.**

(a) Detail the limitations each method imposes on the language translated (does it handle block structure, is there an upper bound on the length of identifiers, etc.?)

(b) Discuss the method of searching, inserting, and deleting entries in the table.

(c) Give space and time estimates for each kind of table manipulation in terms of **the actual** number of identifiers in **the** table.

(d) What is the behavior of the table as it becomes full?

(e) Using **one** of the **tables** discussed above, give an explicit algorithm to discover, record, and recover the address couple (nesting level and order number) for a block structured language.

3. Compiling. (25 *points)*

Consider **a computer** with a memory M and a set of general registers R. M[i] is the ith word of memory and R[i] is the ith general register. Some of the instructions are defined **below:**

L	R1, R2, A	if R2 = 0 then R[R1] ← M[A] else R[R1]←M[R[R2]+A]
S	R1, R2, A	if R2 = 0 then M[A] ← R[R1] else M[R[R2]+A] ← R[R1]
A	R1, R2	R[R1] ← R[R1] + R[R2]
D	R1, R2	R[R1] ← R[R1] / R[R2]
SKIPE	R1, R2	iff R[R1] = R[R2] then skip one instruction
SKIPG	R1, R2	iff R[R1] > R[R2] then skip one instruction
B	A	branch unconditionally to the statement labelled A

(a) Write an assembly language program to execute statements 3-5 of the following PL/I program fragment:

```
DECLARE (I, N, (A, B) (100)) FIXED;            /* 1 */
GET LIST (N, A, B);                            /* 2 */
DO I = N/2 TO N;                               /* 3 */
     A(I) = B(1) + I;                          /* 4 */
END ;                                          /* 5 */
```

Put an appropriate comment on each line of code. You may use any reasonable format for your assembly language.

(b) Assume the computer has one more instruction:

```
SKIPC    R1, R2        R[R1] ← R[R1] + 1 ,
                       iff R[R1] ≤ R[R2] then skip one instruction
```

Assuming all instruction times are equal, write the fastest program you can to execute statements 3-5. Give the run time as a function of N.

(c) Outline a code emitting algorithm which approaches your result in part (b). Point out where it will fall short, and why.

4. Operating systems. (20 *points)*

Answer one of the following two questions.

(a) Describe and illustrate the important features of some time-sharing system, including such features as

(1) dynamic relocation of programs
(2) protection
(3) program switching
(4) interrupt structure and status preservation
(5) the algorithm for memory and processor scheduling

(b) Develop a synchronization mechanism which will (1) avoid mutual exclusion and (2) avoid simultaneous execution of the *critical sections* of two otherwise parallel processes. Show a flow chart or pseudo-Algol code which will carry out the synchronization. If you utilize any unusual meta-instructions, describe their implementation. The synchronization is to be independent of the speeds of the processors carrying out the two parallel processes.

May 1969 Systems Qualifying Exam

1. Programming languages. *(60 minutes)*

Do one of parts (a) and (b).

(a) A number of languages have facilities for defining structures or records, that is, sequences of components which, in some cases, may themselves have components. For example, ALGOL W, COBOL, PL/1, the language defined in "A contribution to the development of ALGOL" by Wirth and Hoare, and the language defined in Standish's thesis. Discuss and compare these facilities in three different languages. Some points you may want to consider are:

(1) how the structure is defined
(2) how components are referenced
(3) how pointer variables affect the implementation
(4) how much work must be done at runtime to implement the structures.

(b) Consider PL/l-like structures with the following syntax for the declaration:

```
<structure dec> ::= DECLARE <component>;
<component> ::= <level number> <identifier> <attribute> |
                <level number> <identifier>, <component list>
<component list> ::= <component> | <component list>, <component>
<level number> ::= integer
<attribute> ::= FIXED | FLOAT | CHARACTER
```

. Thus each component is a value with a simple type, or it consists of a sequence of subcomponents. We also require the main component to have **level** number 1; and if a component **has** level number i all its subcomponents must have level number $i+1$.

Example:
```
DECLARE 1 BOOK
          2 AUTHOR
             3 FIRSTNAME CHARACTER,
             3 LASTNAME CHARACTER,
          2 TITLE CHARACTER,
          2 CALLNUMBER FIXED;
```

One references a component of a structure by the complete sequence of identifiers, **leading** from the structure name (level 1) down to the desired component.

Example: BOOK. AUTHOR. LASTNAME or BOOK. TITLE.

Your problem is to discuss how a compiler would handle such structures.

(1) Indicate, with diagrams, the format of each symbol table element, in general, **and** the symbol table for the above example.

(2) Explain what the semantic routines for the above productions (or other valid ones) would do in order to parse a structure declaration and make up the **symbol table** elements for it. Note: We are not interested in the parsing scheme, but in semantics.

(3) Give the algorithm for finding a symbol table element for a component reference (like BOOK. AUTHOR. NAME).

2. Syntax. (40 *minutes)*

The simple precedence relations are defined for a BNF grammar as follows:

$R = S$ if there is a production $U ::= \ldots RS \ldots$
$R < S$ if there is a production $U ::= \ldots RW \ldots$ where $W \Rightarrow^{+} S \ldots$
$R > S$ if (1) there is a production $U ::= \ldots VS \ldots$ where $V \Rightarrow^{+} R \ldots$, or
(2) there is a production $U ::= \ldots VW \ldots$ where $V \Rightarrow^{+} R \ldots$ and $W \Rightarrow^{+} S \ldots$

A grammar G is a simple precedence grammar if (1) At most one relation holds between any pair of ordered symbols *(R, S)* of G; and (2) No two productions have the same right part.

(a) How does condition (1) help us parse sentences of a simple precedence grammar?

(b) How does condition (2) help us parse sentences of a simple precedence grammar?

(c) Suppose we wished to parse in a right-to-left manner instead of left-to-right. Redefine the relations so that the *rightmost* simple phrase will be detected at each step as we scan the symbols from the end to the beginning.

3. Linking and loading. *(50 minutes)*

An important part of any system is the linkage editor and loader, which accepts object modules (binary decks) from compilers and/or assemblers, links them with necessary subprograms, and loads the complete program for execution. Discuss the form and content of the object module of some fairly sophisticated system. Include in your discussion: entry points, external references, relocation. Specify briefly how the linkage editor goes about its business. Give diagrams and be concise.

4. Memory management. (30 *minutes)*

(a) Describe in no more than two pages the memory mapping scheme for one of the following: ATLAS supervisor, SDS 940 Timesharing Monitor, IBM 360/67 TSS, or some other timesharing machine. Include a diagram.

(b) What are the main categories of memory protection? Describe a system for providing memory protection to physical addresses. Describe a system for applying protections to the logical address space.

Do one of parts (c) and (d).

(c) Briefly describe (with diagram) the cache memory organization on the IBM 360/85. What is the key feature of programs that is essential to the success of a cache?

(d) Describe two of the following schemes for handling collisions in hash coding: random probing, linear probing, or direct chaining. What is the expected number of probes for schemes you picked?

5. Concurrency, parallelism, and scheduling. *(20 minutes)*

Do one of parts (a) and (b).

(a) Describe accurately but briefly the Multi-level Scheduling Algorithm used in CTSS (MIT). How did it function?

(b) Describe with diagram one of the flow graph models of computations listed below.

Karp-Miller computation graph
Estrin-Martin-Turn directed acyclic graphs
Adams data flow graphs

What are the principal elementary components? What are the principal functional properties? What properties of computation have been studied by the model you pick? Could you derive any of the mathematical properties of the model in 20 minutes if you were asked to do so?

May 1970 Systems Qualifying Exam

1. General programming. (60 *minutes)*

(a) Consider the block labelled "MYSTERY" in the following undocumented fragment of an ALGOL W program. What effect does the execution of this block have on the global array *A*? Do the elements A(1), A(2), . . . , *A(n)* satisfy any interesting relations when we reach FIN?

```
begin integer array A(1::10000); integer n;
   . . .
MYSTERY:
   begin integer i, j, k;
      i := 1; j := n;
      while true do
      begin
         while (i < j) and (A(i) < 0) do i := i+ 1;
         if i ≥ j then go to FIN;
         while (i < j) and (A(j) ≥ 0) do j := j- 1;
         if i ≥ j then go to FIN;
      XCH: k := A(j); A(j) := A(i); A(i) := k;
         i := i+1; j := j-1;
      end;
   end MYSTERY;
FIN: . . .
end.
```

(b) Using the following instructions for a hypothetical computer, translate the MYSTERY block into the "best possible" machine code program. Write your program in an appropriate symbolic assembly language, assuming that an appropriate environment for the MYSTERY block has been set up. Give comments explaining the effect of each line of your program. Assuming that each instruction takes one unit of time, give an approximate formula for the running time of your program, as a function of n and the number of times the statement XCH is executed. (Try to produce a short program with minimum execution time.)

The hypothetical computer has memory locations M[0],M[1],M[2],. . . and integer registers R[0],R[1],R[2], . . ., such that R[0] always contains 0. The computer's instructions are given below, where i, j, and k are nonnegative integer constants.

SET	i,j,k	R[i] := R[j]+k
LOAD	i,j,k	R[i] := M[R[j]+k]
STORE	i,j,k	M[R[j]+k] := R[i]
ADD	i,j,k	R[i] := R[i] + (R[j]+k)
SUB	i,j,k	R[i] := R[i] - (R[j]+k)
GEQ	i,j,k	if R[i] ≥ 0 then go to location R[j]+k
LSS	i,j,k	if R[i] < 0 then go to location R[j]+k

Instructions are executed sequentially unless GEQ or LSS causes a transfer of control. **Example:** GEQ 0,0,0 always transfers control to location 0.

(c) Describe briefly the types of code economization a compiler would have to perform if it were clever enough to produce your program of part (b) from the source code in part (a).

2. Language design. (30 *minutes)*

(a) What is meant by "binding time" for symbols?

(b) What are the advantages and disadvantages of early and late binding?

(c) Give examples to illustrate the possible binding times for symbols used in some programming system with which you are familiar.

(d) What are the implications for a programming system of early and late binding?

3. Operating systems. (30 *minutes)*

(a) Discuss briefly the scheduling algorithm used on a time-sharing system such as CTSS, SDS 940, or IBM/360 TSS.

(b) One way to conduct research on operating systems (or many other areas!) is the following:

(1) Give a precise definition of a problem.
(2) Solve the problem.
(3) Prove that the problem has been solved.

Some studies of operating systems have been performed using this approach. Discuss one such study. State carefully the problem of interest and any fundamental assumptions. How reasonable are those assumptions? Give the essential ideas of how the problem was solved, including a discussion of any algorithms which have been developed. What motivated the discovery of this solution? How was it shown that the solution was correct?

4. Syntax analysis. (90 *minutes)*

One of the few notational conventions of mathematical logic which has not yet been widely adapted to computer science is Peano's parenthesis-free notation based on dots for grouping. This problem considers the possibility of exploiting Peano's notation.

We will define a language, called *Ldot*, of expressions in a single left-associative operator. The operands are single lower-case letters and the operators are represented by concatenation; for punctuation we use dots. We will be interested in translating expressions in *Ldot* into equivalent expressions in another parenthesis-free language *Lpost*, which is Polish postfix notation with the operators represented by "*".

The punctuating effect of dots is as follows: The largest number of consecutive dots in any expression divides the expression into its two principal subexpressions. Each subexpression is evaluated using the same rule. Any "ties" are broken by associating to the left. For example:

Ldot	*Lpost*
abc	*ab*c**
a.bc	*abc***
a.b.c..d.ef	*ab*c*def****
a. .b. .c	*ab*c**

(a) Write a program which translates expressions from *Ldot* to *Lpost*, by simulating a machine with one pushdown stack. Use a dialect of Algol (specify which dialect you are using). You may assume, if you like, that the input and output strings are represented as arrays of integers.

(b) Can you write a BNF description of *Ldot* which could be used by a syntax-directed compiler to carry out this translation?

(c) It is well known that BNF is equivalent to nondeterministic pushdown-store automata (NDPA). How do you reconcile this fact with the results of parts (a) and (b)?

(d) What are some interesting problems related to dot notation which you think would be worth exploring if you had time to do so?

May 1971 Systems Qualifying Exam

(Take home exam — return in 5 days.)

Problem 1.

Consider the implementation of an incremental compiler for ALGOL 60 or a similar ALGOL-like language. Assume the programmer is able to converse with the incremental compiler, and is able to insert, delete, or alter statements in his program dynamically. Also assume that he may initiate program execution. Similarly, assume that he may make use of a "PAUSE" statement in his program, which causes execution to be suspended when it is reached. Program execution can be resumed from the point of suspension under programmer control. Thus, a programmer can create a program, start it in execution, suspend it at any arbitrary point, modify the program, then resume execution of the modified program at the point of suspension.

The design of the incremental compiler is such that the compiled program must be primarily, if not entirely, executable machine code, rather than pseudo-code that is interpreted by a program. Moreover, incremental changes in the source program must cause only incremental changes in the compiled program.

(a) For each of the facets of a compiler mentioned below, discuss the problems of implementation given the constraints above. For each facet listed, discuss at least one implementation technique which in your judgment best overcomes the problems your raise. Where pertinent, indicate one or more alternative implementations which you believe are worthy of mention.

(1) Assume that all statements are numbered by the programmer as they are entered. To insert a statement between statement N and M, $N < M$, the new statement is given any number in the interval $N < x < M$. To replace statement N, the new statement is numbered N. To delete statement N, the empty statement is entered with a number N. Consider the problem of maintaining a representation of the current state of the source program.

(2) Consider the effects of allowing declarations to be modified dynamically.

(3) How should the ALGOL nested scope feature be treated in an incremental environment?

(4) How should memory allocation of program variables be done? Consider the effects of arbitrary modification of array bounds after array allocation has occurred.

(b) Find some other aspect of the incremental compiler which is nontrivial to implement, and discuss it as in part (a) above.

Problem 2.

(a) The following algorithm describes a buffering scheme for an n-way merge of ordered files, $n \geq 2$. Your **job** is to give an informal but convincing proof that the algorithm works. (Alternatively, if the algorithm is invalid, you should present an example or examples which make it fail, suggest changes which fix things up, and prove the validity of the corrected algorithm.)

Each of the input files is assumed to be stored on external memory, in blocks containing m records **each.** Each record contains a numerical "key" value. The value of the j th key in a block is **less** than or equal to the value of the $(j+1)$st key, for $1 \leq j \leq m$, and the value of the m th key is less than or equal to the value of the first key in the next block of the file. The final block of the file has been filled with one or more "dummy" records; the key of a dummy record is "∞", **a** value which is greater than all of the non-dummy keys. The output file should adhere to the same conventions as the input files, and should contain **all the** non-dummy records of the input files.

The computer is able to read, write, and compute simultaneously, but it can read at **most one** block at a time and it can write at most one block at a time. The principal virtue of the following algorithm is that, once it gets started, it maintains continuous reading and writing, and essentially issues the read and write commands simultaneously.

The algorithm makes use of 2n input buffer areas, each one block long, denoted by $I[1]$, $I[2]$, ..., $I[2n]$. There are two output buffer areas, denoted by $O[0]$ and $O[1]$. We write

key $[i,j]$

for the value of the jth key in $I[i]$, and

$O[k,l] \leftarrow I[i,j]$

for the operation of moving the jth record of $I[i]$ to the lth position of $O[k]$.

The following auxiliary tables are used:

Name of table	Range of values	Intended significance
$A[i]$, $1 \leq i \leq 2n$	0 or 1	0 if $I[i]$ is available for input, 1 otherwise.
$B[i]$, $1 \leq i \leq n$	1, ... n	Buffer containing the last block read so far from file i.
$C[i]$, $1 \leq i \leq n$	1, ... **n**	Buffer currently being used for the input to the merge from file i.
$L[i]$, $1 \leq i \leq n$	*key*	Value of last key read so far from file i.
$P[i]$, $1 \leq i \leq n$	1, . . ., m+l	The number of the record currently being scanned in buffer $I[C[i]]$.
$S[i]$, $1 \leq i \leq 2n$	1, . . ., n	The buffer to use when $I[i]$ **has been** completely scanned,

Step 1. (the initialization)

1.1 Do the following for $1 \le i \le n$:
1.1.1 Initiate reading the first block of file i into buffer $I[i]$.
1.1.2 Set $A[i] \leftarrow$ **1,** $A[i+n] \leftarrow 0$, $B[i] \leftarrow i$, $C[i] \leftarrow$ i, $P[i] \leftarrow$ **1.**
1.1.3 Wait for the input operation to be complete, then set $L[i] \leftarrow key\ [i, m]$.
1.2 Find q such that $L[q] = \min \{L[1], L[2], \ldots, L[n]\}$.
1.3 Set $t \leftarrow 0$, $k \leftarrow$ nt 1. (t represents the current output buffer,
k the current buffer for input)
1.4 Initiate reading the next block from file q into buffer $I[k]$.

Step 2. (the merging)

2.1 do the following for $l = 1$ *to n:*
2.1.1 If $P[i]$ = mt 1 for some i, set $P[i] \leftarrow 1$ and $C[i] \leftarrow S[C[i]]$ and $A[C[i]] \leftarrow 0$.
2.1.2 Find r such that key $[C[r], P[r]] = \min \{key[C[1], P[1]], \ldots, key\ [C[n], P[n]]\}$.
2.1.3 Set $O[t, l] \leftarrow I[C[r], P[r]]$.
21.4 Set $P[r] \leftarrow P[r]$ t 1.

Step 3. (wait for input/output completion)

3.1 Wait if necessary until the previously initiated input and/or output is complete.
3.2 Set $A[k] \leftarrow 1, S[B[q]] \leftarrow k$, $B[q] \leftarrow k$.
3.3 If $L[q] \neq \infty$, set $L[q] \leftarrow key\ [k,$ ml.

Step 4. (the next input/output)

4.1 Find q such that $L[q] = \min \{L[1], L[2], \ldots, L[n]\}$.
4.2 Find k such that $A[k] = 0$ (such a k will exist).
4.3 Initiate writing from buffer $O[t]$ to the output file.
4.4 If $L[q] \neq \infty$, initiate reading the next block from file q into buffer $I[k]$.
4.5 If the final key in buffer $O[t]$ is ∞, stop; otherwise set $t \leftarrow$ l-t and return to step 2.

(b) Assume now that the computer can do two reads and two writes simultaneously. **Design a** similar algorithm which essentially doubles the rate of input/output of the method in part **(a),** by having two reads and two writes in progress most of the time (assuming very fast processor speed and very large files). Your algorithm should not use more buffers than necessary to ensure continuous operation; for example, the algorithm in part (a) would fail if there were only $2n-1$ buffers, so $2n$ are necessary in that case.

One solution to this problem would be simply to have $4n$ input buffers and 4 output buffers, grouped in pairs, and to carry out the algorithm of part (a) almost as if m were $2m$. But **this** is not satisfactory ! Arrange instead to have only three output buffers; at a typical instant **the** algorithm will be storing into one output buffer, doing the second half of **a** write **from** another, and doing the first half of a write from the third. A similar technique should **be** used for the input files, initiating each read when the previously initiated read is half finished.

Give an informal but convincing proof that your algorithm is correct, and that it doesn't require too many buffers.

Problem 3.

You are to design a system procedure, called the OPEN procedure, to initiate the processing of a predefined user file. A *file* is a collection of records which are sequences of words. Assume a multi-programmed environment with a single central processing unit and movable head disk storage. When not being used, a file resides on disk storage. Files are not shared. At most one user may be accessing a file at any instant. In this system, when a routine initiates a transfer to or from the disk, the routine loses control of the central central processing unit until the disk transfer is complete.

The following information must be maintained for each user file.

(1) File identifier – a unique identifier is associated with each file.

(2) File access control list – a list of all authorized users of the file. Each entry consists of (a) the user identifier and (b) the access type. Examples of access types are read only, read/write only, append only, delete only, execute only, or perhaps a combination of these such as "read and append" access.

(3) File access history – a list of all successful and unsuccessful file access requests. For each access request the user identifier, access type, file identifier, time of request, and success or failure must be recorded.

(4) File status and location – an indication of whether the file is *active*, i.e. some user has successfully called the OPEN procedure and the file is either wholly or partially in main memory; or inactive, i.e. the file is disk resident and not currently in use.

As stated below, part of this problem is to decide where (disk or main memory) and how this information should be stored. The file system maintains a *file directory*, which is at the minimum a record of all files currently known to the system and an indication of the file location and status. The file directory always resides in main memory. The file system maintains a *file header* as the first record of each file on disk storage. The file header contents will be specified by you as part of the solution of this problem. For each active file, a *file control block* is maintained. This control block contains all information necessary to use the file data (location of file buffers, numbers of records currently in main memory, etc.).

(a) Give a pseudo-ALGOL description of the OPEN procedure. This procedure has as input parameters the file identifier, user identifier, and user access type. Assume all parameters are of type INTEGER. The OPEN procedure checks the validity of the access request and if the request is valid defines a file control block for the file. The OPEN procedure is responsible for the maintenance of the file access history. In designing this procedure, keep in mind the fact that disk transfers are slow and it is undesirable to require many transfers to open a file. Also, use of main memory by the file system should not be excessive. Assume all disk input/output is performed by system routines READDISK and WRITEDISK which return when the transfer is completed. Assume all disk transfers are error-free.

(b) Specify the contents of the file directory and file header required to implement the OPEN procedure described above and describe the data structures you have selected. Justify briefly your reasons for the selection of each data structure. Assume that individual entries may be added and deleted from the access control list and that individual entries are added to but never deleted from the access history.

(c) Give formulas for

(1) Main memory required for each file directory entry.
(2) Disk memory required for each file header.
(3) Any additional storage requirements (disk or main).
(4) Execution time, ignoring disk transfer time, to open a file.

Show how (1)–(4) depend on the size of the access control list and the access *history*.

(d) Consider the following two file access methods.

(1) Sequential access method – file records must be accessed one after the other.
(2) Random access method – file records may be accessed in any order.

What, if any, would be the effect *on* your answers to parts (a) through (c) if all accesses were sequential? Random?

ffiay 1972 Systems Qualifying Exam

Problem 1. (50 *minutes)*

Let G be a context-free grammar with initial symbol S, nonterminal alphabet $\{S, A, B\}$, terminal alphabet $\{\vdash, a, 6, \dashv\}$, and the production rules shown below:

$$S \rightarrow \vdash A \dashv$$
$$A \rightarrow a$$
$$A \rightarrow aAB$$
$$B \rightarrow b$$
$$B \rightarrow bA$$

(a) Give a parse tree for the string *I-aabaabi.*

(b) For what values of n, $m \geq 0$ is the string $\vdash a^{n+1}ba^{m+1}b^{n+m-1}\dashv$ in L(G)?

(c) Is this grammar ambiguous? Justify your answer.

(d) Is this grammar LR(1)? Why *or* why not?

Problem 2. *(25 minutes)*

Consider a hypothetical computer with the following characteristics.

1. The computer can support multiprogramming and shared re-entrant programs.
2. The computer has a relocation register and a program length register. The contents of the relocation register are added to every memory address generated by the program, and every valid address must be in the region delimited by the relocation register and the program length register.
3. Programs other than the operation system operate in a "user" state in which the memory protection is delimited by the relocation and bounds registers. Thus such programs must occupy contiguous regions of memory, and their apparent address space runs from location 0 to location L-l where L is the program length. Re-entrant, shared programs must operate in user state.

(a) Discuss problems concerning the use of shared programs in the computer environment described above. Your answer should focus on the problems of communication between programs and shared programs within the constraints placed on their address spaces and the memory protection mechanism.

(b) Invent some reasonable hardware mechanism to facilitate the use of shared programs on this computer system. Your invention should not be a paging mechanism.

Problem 3. (50 *minutes)*

Discuss the problem of writing an optimizing compiler for a paging machine. The source language can be FORTRAN (easier) or ALGOL. The main issue to be addressed is the minimization of page faults.

Problem 4. *(50 minutes)*

Below are two similar procedure bodies for performing a binary search of array A, from element A[1] to element A[N], for ITEM.

```
      LOW := 0 ;
      HIGH := N+1 ;
      FOUND := false;
loop:  if LOW + 1 ≥ HIGH then
         go to notfound;
      PLACE := (LOW+HIGH) ÷2;
      if ITEM = A[PLACE] then
        go to foundit;
      if ITEM < A[PLACE] then
        HIGH := PLACE
      else LOW := PLACE;
      go to loop;
foundit: FOUND := true;

notfound:
      return;
```

PROCEDURE 1

```
      LOW := 0;
      HIGH := N+1;

loop:  if LOW + 1 ≥ HIGH then
         go to exit;
      PLACE := (LOW+HIGH) ÷2 ;


      if ITEM < A[PLACE] then
        HIGH := PLACE
      else LOW := PLACE;
      go to loop;
exit:  FOUND := ITEM = A[LOW] ;
      PLACE := LOW;

      return;
```

PROCEDURE 2

Note that Procedure 1 has an equality check in the loop and that Procedure 2 does not.

Under the assumption that both procedures are correct determine if one procedure is preferable to the other, and under what conditions. You may assume for convenience that N is of the form 2^m-1.

To determine approximate timings for the procedures, assume that the language is compiled and run on an IBM System 360 type of machine, and that the inner loops of the two procedures consist of instructions selected from the set below, Each instruction takes one time unit.

Operation		Description
L	LOAD	Load an accumulator from memory (may be indexed).
LR	LOAD REGISTER	Load register.
A	ADD	Add memory to an accumulator.
AR	ADD REGISTER	Add an accumulator to an accumulator.
ST	STORE	Store an accumulator into memory (may be indexed).
C	COMPARE	Compare an accumulator with memory, set the condition code (may be indexed).
CR	COMPARE REGISTER	Compare an accumulator with another accumulator, set the condition code.
B	BRANCH	Unconditional branch.
BE	BRANCH IF EQUAL	Branch if the condition code indicates that the last comparison was an equality.
BL	BRANCH IF LOW	Branch if the condition code indicates that the first operand was less than the second for the last comparison.
BH, BNL, BNH, BNE, . . .		All other possible variants of conditional branches.
SRA	SHIFT RIGHT	Arithmetic right shift of a specified accumulator.

TAKE HOME PROBLEMS

Problem 1.

Consider the problem of adding matrix arithmetic to ALGOL. The extensions should include the matrix operations of addition, multiplication, equality comparison, inversion, and transposition.

(a) You might want to add a new data type *matrix*. Discuss the advantages and disadvantages of doing so. Write the additional BNF syntax needed for your solution.

(b) You should be able to generate better code for matrix operations in your language than would be possible with ordinary ALGOL. Discuss how to do it.

(c) How would the difficulty of doing this addition depend on the original implementation of ALGOL? Choose some extensible compiler or other translator writing system and describe how you would carry out the extension in its framework.

Problem 2.

Read Chapter 6 of the *Processor Handbook* for the PDP-I 1/45 computer made by Digital Equipment Corporation. The chapter describes the Memory Segmentation Unit available on the 11/45. A partial quote from Chapter 1 is given below.

> The PDP-1 1/45 is a powerful 16-bit computer representing the large computer end of the PDP-I 1 family of computers. It is designed as a powerful computational tool for ... large multi-user, multi-task applications requiring up to 124K words of addressable memory space.

Describe a memory management policy that an operating system might provide for a general purpose multiprogrammed timesharing system in which a very large high-speed drum is available as a backing store. Give attention to the main memory fragmentation problem and the management of shared segments.

Problem 3.

We wish to process a given sequence of integers. Processing consists of the application of one or both of subroutines A and B iteratively. The order in which we apply subroutines A and B and their parameters for each call are unknown ahead of time.

Subroutine A: Given an integer *i*, find the largest element of the first i elements in the sequence.

Subroutine B: Given a integer *i*, and a datum *n*, insert *n as* the *i* th element of the sequence, moving the previous *i* th element to position i+ 1, the *i*+ 1 st element to position *i*+2, and so on.

You must retain the values of the entire sequence in your data structure.

It is possible to design a data structure for which both subroutines A and B require a time proportional to $\log_2 N$ in the worst case, where N is the length of the sequence at the time of the subroutine call.

Example:

position	1	2	3	4	5	6	7	8
element value	4	6	3	7	1	2	5	8

Subroutine A returns:

i	1	2	3	4	5	6	7	8
returned value	4	6	6	7	7	7	7	8

Call Subroutine B with $i = 5$, $n = 9$. New sequence:

position	1	2	3	4	5	6	7	8	9
element value	4	6	3	7	9	1	2	5	8

Subroutine A returns:

i	1	2	3	4	5	6	7	8	9
returned value	4	6	6	7	9	9	9	9	9

Your answer should be based on the algorithm in Example 11.9 of Stone, *Introduction to Computer Organization and Data Structures).* Describe precisely what modifications to the algorithm are necessary to perform the required task. You should indicate which lines of code have to be changed, what additions are required to the data structure, and what additional Algol code is required. Also, describe in English how your modified algorithm works.

June 1974 Systems Qualifying Exam

(This exam was given orally. The following is a simplified version of the questions asked.)

1. <u>Compilers.</u>

(a) Invent a small simple precedence language and show how to derive precedence tables for it. What are the disadvantages of precedence languages?

(b) What are precedence functions? Do they always exist? Are they information lossy? How do they hinder error protection?

(c) What problems face a language designer who must implement a one-pass compiler? What language features cause trouble? How can some of these problems be solved?

(d) Discuss code optimization for one-pass or multi-pass compilers. What kinds of optimization can be done?

2. Build a $SWAP(A, B)$ procedure in ALGOL 60 (using NAME parameters) which always **works**.

3. . <u>Operating systems.</u>

(a) Describe some CPU scheduling methods and explain what each method is intended to optimize.

(b) Sketch response time versus CPU service time for first-come first-served and round-robin scheduling.

4. What data structure would you use to maintain a queue of priority tasks in which each task has a unique priority? You must be able to select the highest priority task for running, and also be able to add and delete tasks quickly. Try to minimize the use of space.

5. Describe several methods of building hash tables, and especially discuss ways of resolving collisions.

6. Describe a virtual machine and tell how it is implemented. What special hardware features are needed?

April 1975 Systems Qualifying Exam

1. Optimization.

(a) What is interval analysis?

(b) What is dead variable analysis?

(c) Are recursion and iteration interchangeable?

(d) How do you do register allocation?

(e) Describe some methods of loop optimization.

(f) How could expression evaluation be optimized?

2. Language design.

(a) What control structures are needed for goto-free programming?

(b) What are the differences between typed and typeless languages? Advantages and disadvantages?

(c) What features do programming languages need for systems programming?

(d) What is the difference between static and dynamic execution environment in block structured languages?

3. Formalisms.

(a) Show that $\{w\bar{w} \mid w \in \Sigma^*\}$ is not a finite state language.

(b) Give a brief description of Hoare's proof of correctness method.

4. Hashing / Searching / Sorting.

(a) What are the worst case times for Quicksort? Heapsort?

(b) What are twin primes used for in hashing methods?

(c) What is the average insertion time in random trees? Deletion time?

5. Data structures / Storage allocation.

(a) What is a B-tree and what is it good for? How is it used?

(b) What is a good data structure for a priority queue?

(c) How do you do minimal space garbage collection for LISP-like environments?

6. Operating systems.

(a) What are the advantages and disadvantages of virtual machines?

(b) What is dynamic linking in MULTICS?

(c) Why do terminal users want multiple processes? Multiple cooperating processes?

(d) What is the difference between *lock-unlock* and P-V?

(e) What is a good job scheduler for an interactive time sharing system? For a batch system? Give an example of a very *bad* time sharing job scheduling algorithm.

(f) How do you fit dismountable disk packs into a hierarchical file system?

(g) What fundamental concepts must your operating system implement for a multiprocessor system like *C.mmp*?

(This exam was given orally.)

May 1976 Systems Qualifying Exam

(The following is an outline of the topics covered in the oral exam.)

1. Compilers.

(a) *uvwxy* theorem.

(b) Give an example of an operator precedence grammar for parenthesized arithmetic expressions.

(c) Compare recursive descent with $LR(k)$.

(d) Explain heap sort.

(e) What is the basic abstract data structure needed for code optimization? What do you do with it?

(f) Give an example of a loop invariant. Explain the notion of weakest pre-condition.

2. Data structures.

3. Sorting and searching.

4. Program design and analysis.

5. Basic operating system principles.

(a) Select a good systems programming language. What features should the language have?

(b) Discuss the pros and cons of various synchronization methods.

(c) Compare interrupts vs. polling as a scheduling discipline.

(d) How do you allow users to share files? In a hierarchical file system? With dismountable packs?

(e) File mapping issues.

(f) What makes a machine "virtualizable"?

(g) Compare directly addressable scratch pad vs. cache memory.

May 1977 Systems Qualifying Exam

Operating Systems.

1. Describe the concept of capabilities in an OS.

2. What are criteria to establish the upper and lower bounds for time slices In a time sharing system?

3. Describe the working set algorithm for paging. When is a program's working set the largest?

4. What are the possible states of processes outside of the set of processes which are actively using memory (the balance set).

5. When processing requests from a disk, a one-way scan has been advocated (SCAN, Teory). Why? Sketch the queue behavior. Estimate the required ratio of arrival rate (λ) to service rate (μ) to avoid long term queue buildup.

6. Describe the principal components of an indexed-sequential file.

Program Design and Analysis.

1. Compare the advantages of macro vs. procedure call in organizing a program.

2. Are there programs that can't be written without go to statements? How can you transform an arbitrary program to eliminate go to's?

3. Is the PL/I ON condition structured? Suggest a structured way of handling exceptions.

4. Define the concept of weakest pre-condition. What is $wp(x:=2*x, x=y)$? What is $wp(\text{if } B \text{ then } S, P)$?

5. What are desirable features in a high-level language for synchronization/communication between parallel programs?

6. What are coroutines? Describe a problem where they would be useful.

Data Structures and Alnorithms.

1. (a) What are some of the different methods for organizing a table, if the operations performed are (1) lookups and (2) insertions? Compare the time and space requirements of these methods.

   (b) Describe methods for resolving collisions in hashing.

2. How does the following program work, and what is its asymptotic running time?

```
/* Given input N, list all the prime numbers ≤ N.*/
integer array A(1:N); integer I, J, N;
read (N);
for I:=1 until N do A(I):=1;
for I:=2 until SQUAREROOT (N) do begin
    J:=2;
    while I*J ≤ N do begin
        A(I*J):=0;
        J:=J+1
    end
end;
for I:=1 until N do
    if A(1) = 1 then write (I);
```

3. Describe a method of external sorting. Estimate its speed.

<u>Compilers</u>.

1. Discuss the allocation of storage for arrays. What are two common methods? What are the advantages and disadvantages of each method?

2. What is intermediate code (or internal form)? Why is it used? What types are there? What are the advantages and disadvantages of each type?

3. (a) What is a left parsable (context free) grammar? Informally describe what a $LL(k)$ grammar is.

   (b) Consider $S \to Sa \mid b$. Is it $LL(k)$? If so, what is k and why? Is it left parsable? Why?

   (c) Answer part (b) for the grammar $S \to bAba$, $A \to b \mid e$.

   (d) Are all $LL(k)$ grammars left parsable? And vice versa?

4. Describe the organization of a compiler-compiler. Describe how you would use one to produce a compiler. State some of the advantages or disadvantages of using a compiler-compiler system as compared to writing the compiler directly.

5. Most programming languages are of what type? Is there an algorithm for determining if the language generated by a context-free grammar is empty? How and why does it work?

(This exam was given orally.)

April 1978 Systems Qualifying Exam

COMPILERS AND PROGRAMMING LANGUAGES

1. <u>Language design.</u>

(a) Often language features are related. What language feature should be included **in a** programming language if OWN variables are part of the design?

(b) What does the term “dangling reference” refer to?

2. <u>Optimization</u>.

For integers A, **B**, C, 0, E, **F**, and **TEMP**, when is the following transformation not valid for optimization?

```
A := B*C+D;        ⇒        TEMP := B*C;
E := F+B*C;                 A := TEMP +D;
                            E := F + TEMP;
```

3. <u>Parsing</u>.

(a) Compare the power of simple precedence, *SLR*(1), *LALR*(I), and *LR*(1) parsers in terms of languages and grammars.

(b) Qualitatively, how does the construction of *SLR*(1), *LALR*(1), and *LR*(1) parsers differ?

4. <u>Garbage collection.</u>

(a) What is “compactifying garbage collection”? What is the **major** implementation problem involved in writing such a garbage collector?

(b) Explain, using pictures, how to implement a compactifying garbage collector.

5. <u>Formal language theory.</u>

Are the following problems decidable or undecidable? Give some argument to **justify your answer** (not necessarily a formal proof).

(a) Does a PDA *M* accept the null string? ($\epsilon \in$ L(M))
(b) Does a PDA *M* accept the empty language? (L(M) $= \emptyset$)
(c) Does a PDA *M* accept an infinite language? (IL(M) $| = \infty$)
(d) Does a PDA *M* accept a regular language? (3R L(M) = R)
(e) Does a DPDA *M* accept a regular language? (3R *L*(*M*) = R)

PROGRAMMING METHODOLOGY AND LANGUAGES

1. Explain the view of data types as demonstrated by Hoare in his axioms for data types (in Pascal or *Notes on Data Structuring).*

2. (a) Explain the concept of data abstraction.
   (b) How are the ideas of an object and a variable used to define the CLU abstraction mechanism?

3. What is a generator? Give an example of its use. (A generator is a data abstraction which computes sequences of values for an abstract type, It provides an initialization procedure and a procedure to obtain the next value in the sequence.)

4. Explain the concepts of proving backwards (Hoare formalism) vs. proving forwards (Floyd formalism).

5. What are guarded commands? What are they intended for? What type of formal rules are applied to them?

EJG SYSTEMS QUAL QUESTIONS

1. The language Pascal requires all arrays to have constant dimensions. What simplifications does this imply in the run time environment implementation? Pascal character strings are simply linear arrays of single characters. Discuss restrictions this place on the programmer which would be removed if arrays were dynamic. What desirable string operations would still not be available in a straightforward implementation of dynamic arrays?

2. What is a "pipe" as implemented in UNIX? Discuss some advantages and disadvantages of having this as the only communication mechanism.

3. Discuss issues in machine independence of modern programming languages. What problems are there in seeking true independence? What are some implications of machine independence on language design? In particular, can you think of any impacts of machine independence on design of an intermediate language?

4. Consider a simple binary search. Write an abstract algorithm on the blackboard (in English or high level pseudo-Algol). What is a useful invariant of the main loop?

5. What are some means for avoiding deadlock? From a practical standpoint, discuss deadlock recovery.

DATA STRUCTURES AND ALGORITHMS

1. Describe a tree search technique for searching a fixed set of data. When can you produce an optimal search tree? What is it? What techniques are needed for a search tree organization over varying data?

2. Describe a data structure for implementing a **multiset** with two operations: (1) add an element; and (2) choose some element, remove it, and return its value. Is your solution best suited for large or small data items? Give **a solution** for the other case.

3. Given an algorithm for computing transitive closure, how can it be used to get an algorithm for finding path lengths in a graph?

4. Analyze the order of execution time for the following recursive function in terms of n, m

```
f(m,n) = f(m,n,2)

f(m,n,i) = if m ≤ i or n ≤ i then n*m
           else if n mod i = 0 and m mod i = 0
                   then i*g(m/i,n/i,i)
                   else g(m,n,i+1)
```

5. Give an efficient method for finding a mod b without using division (a, b are not necessarily integers).

OPERATING SYSTEMS

1. Locks.

What is the objective of lock primitives that lock multiple resources with one invocation?

2. Queues.

(a) When is FIFO not appropriate for a timesharing scheduler?

(b) Characterize the behavior of a queue when a transient load exceeds the maximum service rate.

3. Confinement.

What strategy is used by a capability-based system to confine processes? What is the critical mechanism? Are the remaining problems in confinement?

4. Virtual memory.

Why is segmentation desirable in addition to paging?

5. Network.

What services are required to let processes in distinct machines interact with each other?

(This exam was given orally.)

May 1965 Artificial Intelligence Qualifying Exam

Problem 1. (50 *minutes)*

Suppose that the *Communications* of *the* ACM were running a series of short expository articles on various fields in the computer sciences. Each article contains a survey of: the history of the field; genera1 approach, theory, and rationale underlying the work; major projects attempted, successes and failures, and how these fitted into the stream of the research effort; major problems exposed and as yet unsolved; and indicated paths for further research, with specific suggestions.

You are now asked to produce a resonably detailed outline of an article for each of the following fields:

(a) Heuristic programming

(b) Mathematical theory of computation

(For problems 2-5, do one of parts (a) and (b).)

Problem 2. *(40 minutes)*

(a) A criticism that has often been levied against programs in advanced nonnumeric applications areas (such as machine translation of languages, information retrieval, or problem solvers) is that the programs behave "without understanding", that is, without a sense of the "meaning" of the problem. Not infrequently, this criticism is made by sensitive and astute observers of the scene. What do you make of the above criticism? What does it mean? In particular, what types of research efforts are indicated to overcome the criticism? In carrying out this research, what previous work is relevant as a foundation and basis for progress, and how so? Try to be as concrete as you can in your analysis and references.

(b) Artificial intelligence researchers have devoted a considerable amount of energy to the problem of modeling and programming processes of *deductive* inference (e.g. theorem proving programs, game playing programs). But what efforts have been made to study processes of *inductive* inference for computers? If you know of any, name them, and for each describe the task studied, the general scheme of the program, and the results. In your answer, please state what you mean by inductive inference (it is *not* the same as mathematical induction).

Problem 3. (30 *minutes)*

(a) One major issue in the design of list processing languages and translating systems is the erasure problem. Different list processing systems treat the problem in different ways. What is the erasure problem? How is it handled in LISP? SLIP? IPL-V?

(b) Let a directed graph be given by a list of lists. Each sublist gives first a vertex and then the vertices that can be reached from it in one step. For example, the graph

$$\begin{tikzcd}
 & B \arrow[r, bend left] \arrow[dl, bend right] & D \arrow[l] \\
A \arrow[ur, bend left] \arrow[rr] & & C \arrow[ul]
\end{tikzcd}$$

is described by the list ((A *B* C) (*B A D*) (C *B*) (*D* B)).

Write M-expressions for a function *distance*[*x*; *y*; *g*] which gives the number of steps required to go from vertex *x* to vertex *y* in the graph g. It should return the atom *infinity* if there is no path from *x* to *y* in g.

Problem 4. *(30 minutes)*

(a) Discuss the problem of proving the correctness of an ALGOL translator. What relevant concepts and techniques are available? What remains to be developed? Include a discussion of the concept of “correctness” you are using in your answer.

(b) The reverse of a list may be computed by the function

reverse[*x*] = *rev*[*x*; *NIL*]

where

rev[*x*; *y*] = **if** *null*[*x*] then *y* else *rev*[*cdr*[*x*]; *cons*[*car*[*x*]; *y*]]

. For example, *reverse*[(*A B* C *D*)] = (*D* C *B* R).

Prove by recursion induction that for any list *x*,

reverse[*reverse*[*x*]] = *x*.

Problem 5. *(30 minutes)*

(a) Potentially one of the most powerful heuristic methods for problem solving programs is *planning.* Illustrate briefly by example, in an abstract way, how planning can be a powerful heuristic device. Under what conditions is planning not a useful heuristic? Describe how the planning method of the *General Problem Solver* works.

(b) (1) In Feigenbaum’s article “The Simulation of Verbal Learning Behavior” (in *Computers and Thought)* he reports that two types of forgetting (so-called oscillation and retroactive inhibition) are observed in the behavior of the EPAM model in simulated verbal learning experiments even though there is no information decay or destruction postulated in the model. What gives rise to the forgetting? (Explain in some detail.)

(2) What are the major similarities and differences between EPAM and Selfridge’s **PANDEMONIUM** as models of pattern recognition decision processes?

March 1966 Artificial Intelligence Qualifying Exam

Problem 1.

(a) You have been given two papers. Both address themselves to present-day limitations and inadequacies in the concepts and techniques used in heuristic programming.

What are the main issues being treated? What is your analysis and opinion of the contentions of the authors?

(b) Newell explicitly brings up the issue of representation. What is the problem of representation? Why is it important? Concrete examples will help your argument. What is being done to attack the problem of representation and how do you think this attack should proceed?

Problem 2.

Write the simplest program in LISP M-expressions that you can for determining whether 3-dimensional tic-tat-toe (in a 4 x 4 x 4 cube) is a win, draw, or loss for the first player. Explain carefully the role of each auxiliary function, giving examples if necessary.

Problem 3.

What is meant by an *unsolvable class of problems?* Give an example of such a class, and outline a proof that it is unsolvable.

Problem 4.

Write the statements that 3-dimensional tic-tat-toe is a win, loss, or draw for the first player as sentences of first-order logic. Give the intuitive meaning of any predicate or function letters that you use.

Problem 5.

(a) What is "hash addressing"?

(b) How might it be used in the implementation of a list processing language?

(c) What kinds of facilities for using hash addressing capabilities at the source language level might be made available in a symbolic computation language?

Problem 6.

Let the concatenation $x*y$ of two lists x and y be defined by

$x*y =$ if n x then y else ax. $[\mathrm{d}x*y]$

Let the length of the list x be defined by

$l[x] =$ if n x then 0 else $[l[\mathrm{d}x]]'$

Let addition be defined by

$m + n =$ if $n = 0$ then m else $m' + n^-$

Prove that

$l[x*y] = l[x] + l[y]$

Hint: The following alternative form of the recursive formula for addition may be useful.

$plus\,[m; n] =$ if $m = 0$ then n else $[plus\,[m'; n]]'$

If you use this formula, prove it equivalent to the original definition.

Problem 7.

What has been achieved so far in the heuristic program area in making programs learn from their experience? What are the limitations of the methods used? What do you think should be done next? What will be the limitations, if any, of the programs you advocate?

Problem 8.

(a) Each of the following examples has been discussed in connection with more than one computer program. Discuss at least two approaches to theorem-proving by computer, using in your discussion at least one of the examples.

(1) $\sim(P \vee Q) \supset \sim P$

(2) $(\exists x)(\exists y)(\forall x)(Fxy \supset FyzFzz)(FxyGxy \supset GxzGzz)$

(3) In a group, the existence of a right inverse follows from the other axioms.

(b) State (1) the completeness theorem for first-order predicate calculus and (2) Church's theorem. Discuss their relevance to theorem-proving by computer.

October 1966 Artificial Intelligence Qualifying Exam

Problem 1. (60 *minutes)*

Answer 6 of the following 8 parts.

(1) With suitable training, EPAM has no trouble recognizing TAE CAT as THE CAT and ***not*** TAE CHT. Why?

(2) What does "static evaluation" mean as used in the literature of game-playing programs?

(3) Show by example how the alpha-beta heuristic works.

(4) What does "linear separability" mean, in the pattern recognition literature?

(5) What steps does **Bobrow's STUDENT** program go through in deriving an answer to an algebra word problem?

(6) What kinds of operators does the Uhr-Vossler pattern recognition program use? **In** what ways does it get its operators?

(7) In the mechanical translation game, what in general do parsing programs do and what is their interaction with dictionaries? What is a context-free phrase structure grammar?

(8) In CPS, what is the fundamental problem with applying operators that led the researchers to organize into a distinct and separate goal the apparently straightforward job of applying a selected operator?

Problem 2. *(60 minutes)*

(1) Formal logic.

(a) Transform the following schema into an equivalent one which has disjunction and negation as its only truth-functional connectives:

$$\sim(p \equiv (qr)) \supset r$$

(b) Determine which of the three schemata

$$(\exists x)(Fx \supset Gx)\,;\quad (\exists x)(\exists y)(Fx \supset Gy)\,;\quad (\exists x)\,Fx \supset (\exists y)\,Gy$$

if any, are equivalent to each other. Show your reasoning.

(c) Would it be possible to write a computer program to solve problems like part (a)? Like part (b)? Explain briefly why or why not.

(2) The *printing* ***problem*** is the general problem of deciding for an arbitrary given Turing machine and input tape whether or not the machine ever prints the symbol S_1. Is the printing problem solvable? Prove your answer.

Problem 3. (60 *minutes)*

(1) It is alleged that list processing, a basic tool in artificial intelligence research, is hideously inefficient on a "paged" virtual memory machine, such as the IBM 360/67. Why is this alleged? Describe a method or methods for getting around the basic problem, and defend your scheme.

(2) Contrast a list processing language such as LISP with a language not having specific list handling capabilities for use in: language translation; character recognition; or compiling.

Problem 4. *(60 minutes)*

(1) Do you believe that heuristics discovered in one field of intelligent activity, such as theorem proving, can help us in some other field, for example language translation? Justify your belief by specific reference to cases, choosing any two fields with which you may be familiar.

(2) In H. L. Dreyfus' recent critique of artificial intelligence research, he asserts that there are three fundamental forms of human information processing (fringe consciousness, essence/accident discrimination, and ambiguity tolerance) which he maintains are systematically excluded from all attempts to analyze intelligent behavior in digital form. Give evidence either to support or to deny this assertion. Pick one of these processes and outline a method which you believe might work in order to program this form of human activity for a computer.

March 1967 Artificial Intelligence Qualifying Exam

Problem 1. *(60 minutes)*

Answer 6 of the following 9 questions.

(1) Which of the six clauses of the following formula can be dropped as redundant?

$$p\bar{q} \vee \bar{q}r \vee p\bar{r} \vee \bar{p}q \vee \bar{p}r \vee q\bar{r}$$

(2) Is the following formula valid? Prove your answer.

$$(x)(Ey)Fxy \vee (x)(Ey){\sim}Fyx$$

(3) Last January, Dr. Nilsson of SRI described the "robot" project of his Artificial Intelligence Group at SRI. In designing the robot they have done a great deal of computer simulation work. They have simulated the robot, simulated the environment, and made the simulated robot behave in the simulated environment. Those of you who have seen the movie of the CRT displays of this know that these simulation results are quite impressive-so impressive that they raise the question, "If you can prove out your basic ideas so handily by simulation techniques, why bother to go through the annoying and tedious engineering work to actually build an electromechanical robot?" From the point of view of the advancement of the art and science of artificial intelligence, in your opinion what is the most cogent and plausible answer that Dr. Nilsson could give to this question?

(4) The following are some characterizations of problem solving strategies. For each, describe briefly what it is and illustrate by citing an example.

(a) breadth-first
(b) depth-first
(c) progressive deepening

(5) This question relates to the Uhr-Vossler pattern recognition program described in *Computers and Thought.*

(a) The program processes the input "retina" with "5 x 5" operators (i.e. 25 cells). Discuss briefly the ways in which it obtains these operators.

(b) Does the Uhr-Vossler scheme represent a significant departure from or advance over previous efforts? Answer "yes", "no", or "yes and no", and justify your answer.

(6) Recall that in the experiments done by Paige and Simon on human problem solving behavior in algebra word problem tasks (experiments undertaken in the light of Bobrow's STUDENT program), the experimenters designed certain "impossible" problems, for example in which the answers were negative speeds or negative lengths. How would the STUDENT program behave if presented with these algebra problems to solve? Justify your answer briefly.

(7) Characterize in a few sentences the major similarities and differences between the following two stimulus-classifying systems:

EPAM (Feigenbaum and Simon)
Pandemonium (Selfridge)

(8) In the machine learning area, what is the credit assignment problem? How is credit assignment handled in the learning procedures of Samuel's checker playing program?

(9) In heuristic programming, what is meant by the representation problem? How does McCarthy propose to handle the representation problem for an Advice Taker?

Problem 2. *(60 minutes)*

Answer 2 of the following 5 questions.

(1) What do you think of the prospects of proving programs correct and checking the proofs by computer? State relevant results and discuss what remains to be done.

(2) Give an example of a theorem and its proof by recursion induction. Make sure that there are no gaps in the statement of the theorem or in the list of hypotheses used.

(3) Take as a domain the set of all humans that ever were or will be and suppose them all descended from Adam and Eve. Let Dxy mean x is the daughter of y and Sxy mean x is the son of y. Write a complete set of axioms for these relationships, that is, such that any domain and predicates D and S satisfying the axioms could be the set of all humans and such that the set of all humans is guaranteed to satisfy the axioms.

(4) Let L be a context-free language. Let x be a terminal symbol and a be a variable over strings, and let

$$D_x = \{ \alpha \mid x\alpha \in L \}.$$

Is D_x context-free? Prove your answer.

(5) Prove that a set of positive integers is recursive if and only if it is recursively enumerable in increasing order without repetitions.

Problem 3. *(60 minutes)*

Consider the problem of inferring, from a set of strings in a language, a grammar for that language. For example, the strings

car, cdr , caar, cadr , caddr, cddar

could be part of the language generated by the following grammar:

<string> ::= *c* <middle> *r*
<middle> ::= *a* | *d* | *a* <middle> | *d* <middle>

In trying to design a program to carry out such an inference, one would encounter many of the central questions in Artificial Intelligence. Discuss one of the following aspects of the grammatical induction problem. Be as precise and specific as possible. You may also include a set of random comments that occur to you while answering this question.

(a) *Representation.* How would you represent the strings, the grammar, and any auxiliary information? You have your choice of existing languages. Is the representation likely to be a crucial factor in this problem?

(b) *Generalization.* A grammar may be regarded as a generalization of the set of strings. What measures of generality would be appropriate here? Do the measures generalize?

(c) *Modes of inference.* To what extent do the terms

deductive, inductive, abductive, hypothetico-deductive

apply to the problem of inferring a grammar?

(d) *Learning.* How would you design a program which would learn a grammar through interaction with the environment? Would you expect such a program to perform **better than** a non-learning program requiring the same amount of effort to produce?

January 1968 Artificial Intelligence Qualifying Exam

Problem 1. (45 *minutes)*

Answer **one** of parts (a) and (b).

(a) Suppose you are writing a literature review article entitled "Artificial Intelligence: Themes in the Second Decade".

Choose one "theme" out of the many that can be identified in the artificial intelligence research area. Cite the key papers of the past six years bearing on the theme, which taken together constitute evidence that such a theme in fact exists. In approximately 500 words of clear exposition, survey your selected theme, in the style of a literature survey article such as might be acceptable for the *CACM*. For purposes of this question, only post-1962 articles are relevant.

(b) Discuss in detail the state of accomplishment of one of the following programs and how further progress might be made in that area.

(1) the Creenblatt chess program
(2) the Samuel checker program
(3) the Wos-Robinson work on theorem proving **by resolution**
(4) the Stanford AI Project block stacker
(5) **the** general area of question answering programs.

Problem 2. *(15 minutes)*

What are **the** prospects for using a general concept of similarity in heuristic programming (for example: similarity of chess positions, pictures of characters, or pictures of faces)? Give examples of your proposed **uses.** In this connection you may discuss Dreyfus' complaint that machines cannot have "ambiguity tolerance".

Problem 3. *(30 minutes)*

Answer **one** of parts (a) and **(b).**

(a) Give a first-order logic axiomatization (using predicates, functions, and equality) of the following problem:

> A farmer **has** to cross a river with a wolf, a goat, and a cabbage in a boat that can hold only one of them at a time besides himself. Leaving the wolf alone with the goat or the goat alone with the cabbage is disastrous. How should he cross?

The axiomatization should be such that the fact that the farmer can get across with his charges is a theorem. Explain any predicates **and functions used.**

(b) The "representation question" is a central one in artificial intelligence. Describe as many as you can of the alternative representations of the game tic-tat-toe. For each representation, describe the problem solving technique which seems most applicable.

Problem 4. (30 *minutes)*

Using abstract syntactic and semantic definition of suitable source and object languages, define a compiler for conditional arithmetic expressions. State formally a definition of its correctness. What are the main ideas of the proof?

Problem 5. (15 *minutes)*

Answer one of parts (a) and (b).

(a) Give a grammar for the language consisting of all well-formed formulas of the first-order predicate calculus.

(b) (1) Does each of $(Ex)(Fx \equiv Gx)$ and $(Ex)Fx \equiv (Ex)Gx$ imply the other?

(2) Are the formulas $(x)(y)(Fxy \supset \sim Fyx)$ and $(x)(y)(\sim Fxy \supset Fyx)$ compatible?

In each part, show how you arrive at your answer.

Problem 6. (30 *minutes)*

Answer one of parts (a), (b), and (c).

(a) Let A be the set of all Gödel numbers of Turing machines. Let K be the set of Gödel numbers of Turing machines which halt when given their own number as input. For each of the three sets A, K, and $A-K$ answer the following questions and give brief proofs.

Is the set recursive?
Is the set recursively enumerable?

(b) Let G_1 and G_2 be arbitrary context-free phrase structure grammars. Is $L(G_1) \cap L(G_2) = \emptyset$ decidable? Prove your answer,

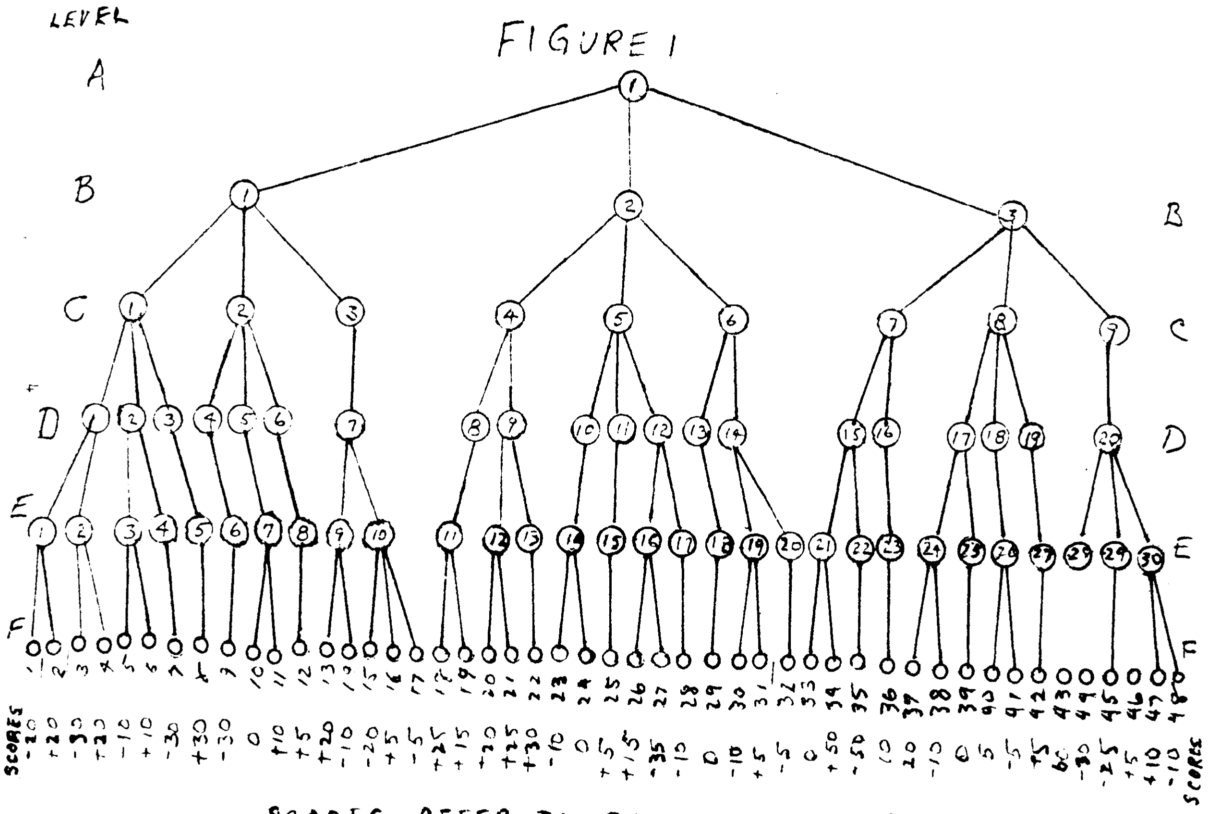
(c) Give two basically different formulations of the intuitive notion of effective process. Outline a proof of their equivalence.

January 1969 Artificial Intelligence Qualifying Exam

Problem 1. (30 *minutes)*

(a) The move tree in Figure 1 is to be scanned from left to right on a depth basis using alpha-beta pruning. Whenever a node can be pruned, list the node at which the comparison leading to the prune is made and the highest level node affected. For example, if a decision at node **B3** leads to the pruning of node **C9**, list **B3-C9** and do not list node 020, nodes **E28, E29**, E30 or nodes *F43* through *F48.*

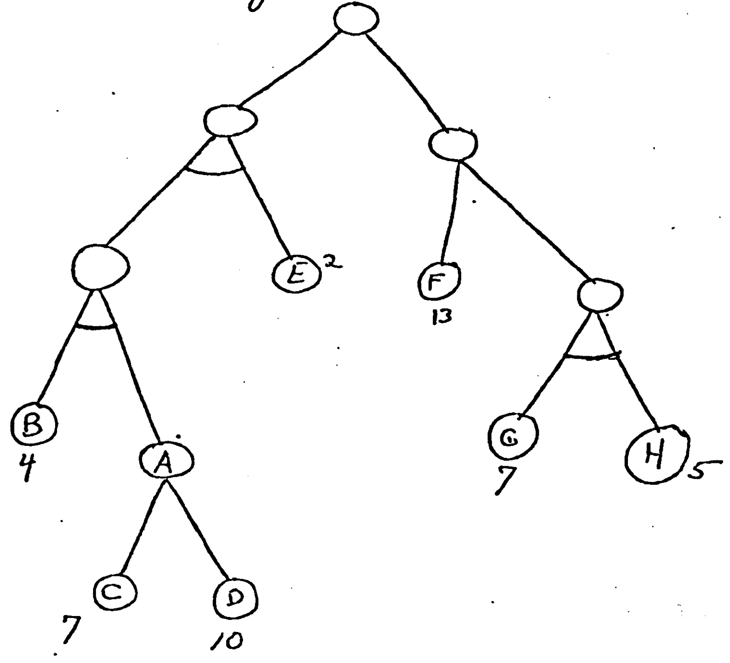
(b) The and/or tree (or sub-goal tree) illustrated in Figure **2** has been generated by a problem solving process. The numbers adjacent to the trimmed nodes represent an estimate of the amount of effort required to solve the problems represented by these nodes. The problem represented by the node marked A has just been converted into two alternative sub-problems. In order to finish the remaining part of the problem with least amount of additional effort, which sub-problem should be processed next and why?



SCORES REFER TO TERMINATING BOARDS AS VIEWED BY SIDE TO PLAY THIS TERMINATING BOARD.

1-7-67

Figure 2



Problem 2. (90 *minutes)*

There has been much talk in artificial intelligence about the "representation **problem".**

(a) Briefly discuss one paper on the subject.

(b) Give a specific example from the above paper to illustrate **the importance of representation in a problem solving** system.

Problem 3. (40 *minutes)*

(a) **The paper** on "State of the Art of Pattern Recognition" by Nagy purports to cover all that **is important** about pattern recognition. Most of **these** are inadequate **in** dealing **with machine perception of vision and speech. Why?**

(b) Discuss the **merits** and demerits of syntax-directed recognition **schemes in vision and speech.**

Problem 4. (40 *minutes)*

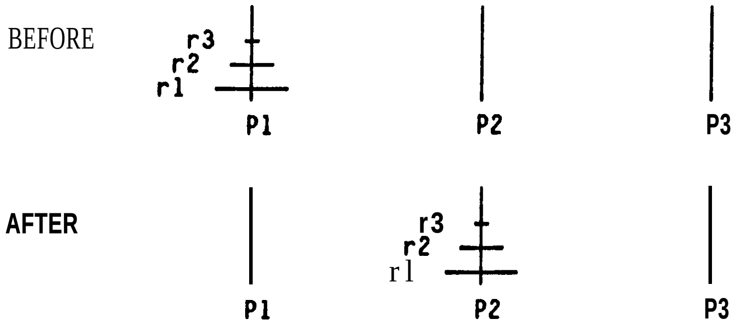
Compare the models used **by** Colby, Green and Raphael, and **Quillian in dealing with natural** language.

Problem 5. (50 *minutes)*

Answer one of parts (a) and **(b).**

(a) In his paper "Heuristic Programming: Ill-structured Problems", Newell has presented a **comprehensive** study of heuristic programming. Describe the current state of **knowledge on** the subject, giving specific examples to illustrate as many **points as possible.**

(b) The "Towers of Hanoi" problem is to be axiomatized. Assume there are three pegs, P1, P2, and P3, and three rings of decreasing size, rl, r2, and r3. The three rings are originally stacked **in** decreasing order of size on P1 and we want to place them in decreasing order on P2.



A legal **move** requires **moving** exactly **one ring to another peg, with the** restriction that the ring **may not be** placed **on a** smaller ring.

Write a set of first-order logic axioms for this problem such that the following **can be** proved:

$$holds(on(P1, list(r3, r2, r1)), on(P2, null), on(P3, null), S_0) \supset$$
$$\exists x\ holds(on(P1, null), on(P2, list(r3, r2, r1)), on(P3, null), S)$$

The fourth position of the predicate *"holds"* is a state variable. You might consider **axioms** involving state-valued functions.

Problem 6. *(30 minutes)*

Extra question for **students planning** to do a thesis in the area of Artificial Intelligence:

What **do you think are the** "kernel ideas" in **the** field of Artificial Intelligence?

March 1970 Artificial Intelligence Qualifying Exam

(Time limit: 7 hours)

The Primacy of Search

The history of the study of problem solving in artificial intelligence is primarily a history of the study of search. From the earliest reports of problem solving programs in the mid-fifties to the most recent synthesis (Nilsson), the central focus has been on the generation of solution spaces and the heuristic control of search for solutions within these spaces. **Nilsson** reaffirms the "primacy of search". Feigenbaum (*IFIP* 68 paper) has referred to heuristic search as the central paradigm of artificial intelligence research.

Problems 1 through 4 all relate, in one way or another, to this paradigm.

Problem 1.

Heuristic search has been characterized as follows:

> "A tree of 'tries' (also called subproblems, reductions, candidates, solution attempts, or alternatives-and-consequences) is sprouted by a generator. Solutions exist at particular depths along particular paths. To find one is a 'problem'. For any task regarded as nontrivial, the search space is very large. Rules and procedures called heuristics are applied to direct the search, to limit the search, or to constrain the sprouting of the tree."

Comment briefly on this characterization.

Problem 2.

List and briefly characterize particular techniques and methods for heuristic control of search that have been studied to date. As a start, here is a **list** that should trigger appropriate associations: Logic Theorist, problem reduction, game-playing programs, MUTIPLE, evaluation, minimum-cost analysis, bi-directional search, planning, DENDRAL.

Problem 3.

Consider the role of task-specific information, as revealed by the following dialectic:

Thesis: The primacy of heuristic search,

Antithesis; The power of a particular problem solving program is a function of the quality and quantity of task-specific knowledge and theory employed; heuristic search is an inefficient and risky procedure employed by the ignorant.

You are to supply the synthesis in this dialectic.

Examples: In the most recent DENDRAL work (on amines, unpublished), the spectral theory employed in the Preliminary Inference Process is so good (i.e. powerful) that there is no work remaining for the Structure Generator to do, that is, only *one* structural hypothesis is implied.

The Moses integration program differs from Slagle's program in that the former does almost no search-it has the right method for every problem.

Problem 4.

If one considers each set of clauses to be the node of a tree, the resolution principle is a "legal move generator" for the space of possible proofs-by-contradiction for theorems in the first-order predicate calculus. By itself, it is of course an inefficient theorem prover in the same sense that a legal move generator is, by itself, an inefficient chess or checker player. The quest for efficiency (and thereby effectiveness in particular problems of interest) involves search strategies. Some simple ones have been devised.

Name three such search strategies, and illustrate how they prune the tree and guide the search for a contradiction.

What is *not* simple is to devise heuristic search strategies for exploiting task-specific information in the domain of a problem that has been given a representation in first-order predicate calculus.

Present one such heuristic search strategy for some problem domain in which a resolution theorem prover is to be used as a "general problem solver", and illustrate how it would work to control search.

Problem 5.

(a) Give an axiomatitation of the "Missionaries and Cannibals" problem in the formalism of McCarthy and Hayes such that it follows from the axiomatitation that the missionaries can' get safely across the river with the cannibals.

(b) What is the relation between this formalization of the M & C problem and that of GPS?

Problem 6.

A single monocular view of a scene is not, in general, sufficient to determine the position of objects in the scene.

(a) State as precisely as possible why this is the case.

(b) Describe briefly at least 6 ways in which people get complete scene descriptions.

(c) Pick one of the above methods and describe in detail how you would implement it on a computer.

Problem 7.

The following set of strings is part of a finite-state language. Find the best finite state grammar for this language. Explain, as precisely as possible, why your choice is the best one.

a	*bbaa*
ba	*abba*
bb	*baaa*
aba	*aabb*
aaba	*baba*
abbb	*bbbba*
bbba	*abaaa*

Hint: There is a non-deterministic finite state grammar with three non-terminals which generates this set of strings.

Problem 8.

Discuss the Frame problem.

March 1971 Artificial Intelligence Qualifying Exam

(Time limit: 7 hours)

Problem 1.

Let *successors*(p) be a function that gives a list of the immediate successor positions to a position p in a tree. Let the predicate *iswin*(v) be true if v is a won position. Write programs in LISP or some other well-known language for

(a) depth-first search to a maximum depth n
(b) breadth-first search.

The output of the program should be a path through the tree from an initial vertex S to a vertex that satisfies the predicate *iswin.*

Problem 2.

Two blocks are on a table in front of a robot hand. The hand can move to any point in the vicinity, and its fingers can close or open (to grasp or drop objects, respectively).

Given the following predicates:

$At(x, p, s)$	object x is at position p in state s
$Hand(p, s)$	the hand is at position p in state s
$Closed(s)$	the fingers are closed in state s
$Open(s)$	the fingers are open in state s
$Same(p, q)$	positions p and q are identical

and the following functions:

$grasp(s)$	the state gotten to from state s by closing the fingers
$drop(s)$	the state gotten to from state s by opening the fingers
$move(p, s)$	the state gotten to from state s by moving the hand to position p (from **any** other position)
$on(p)$	a position in space just above position p

We can describe these circumstances in first-order predicate **calculus** as follows:

The initial state:

1. $At(B1, P1, S0)$
2. $At(B2, P2, S0)$
3. $Hand(P3, S0)$
4. $\sim Same(P1, P2) \wedge \sim Same(P2, P3) \wedge Game(P1, P3)$

The effects of actions:

5. $(\forall p, s)\ Hand(p, move(p, s))$
6. $(\forall s)\ Closed(grasp(s))$
7. $(\forall s)\ Open(drop(s))$
8. $(\forall x, p, q, s)(At(x, p, s) \wedge Hand(p, s) \wedge Closed(s) \supset At(x, q, move(q, s))$

Predicates unaffected by actions ("frame" axioms):

9. $(\forall x, p, q, r, s)\,(At(x, p, s) \wedge Hand(r, s) \wedge \sim Same(p, r) \supset At(x, p, move(q, s)))$
10. $(\forall x, p, s)\,(At(x, p, s) \supset At(x, p, grasp(s)) \wedge At(x, p, drop(s)))$
11. $(\forall p, s)\,(Hand(p, s) \supset Hand(p, grasp(s)) \wedge Hand(p, drop(s)))$

From the above axioms, it is possible to prove that the hand can stack the blocks and then be free for further use.

(a) Construct a proof by resolution of the following theorem:

$$(\exists s, p)\,(At\,(B1, p, s) \wedge At\,(B2, on(p), s) \wedge Open(s))$$

(b) Does your proof use any of the principal resolution strategies such as unit preference, set of support, subsumption, or linear format? Discuss in general the role of such strategies, and why they were or were not useful in this problem.

(c) Axiom 10 contains an obvious oversimplification. Describe the situation that is not appropriately handled, and propose a substitute axiomatization that corrects this deficiency.

Problem 3.

One of the key problems of Artificial Intelligence is how to represent common knowledge about the physical world by data structures in a computer's memory. For example, consider the following information about a well-known object:

"A chair is a seat with four legs and a back, for one person to sit upon. It is an article of furniture, frequently used in front of a desk, and is usually made of wood or metal."

(a) Show in detail how you could represent this information in each of the following ways:

(1) By a "semantic net" of word associations (as in the work of Quillian, Schank, or Winston).

(2) By predicate calculus (cf. McCarthy, Green, Sandewall).

(3) By any other representation that differs from both (1) and (2) in some essential way (e.g. PLANNER).

(b) For each of the above two representations of the concept "chair", discuss one use for which that representation is clearly better suited than the other.

Problem 4.

Discuss as concretely as possible what a reasoning program needs to know about what it and others (e.g. travel agents) know in order to plan a trip to Timbuktoo. How do you propose to represent this information? What rules of inference and axioms will you provide? In this problem, you are essentially on your own; there is hardly any relevant literature.

Problem 5.

(a) Compare the search procedures, generality of techniques, task domains and goals of GPS and Heuristic DENDRAL. What are the most severe limitations of each program?

(b) Tower of Hanoi problem.

In a temple in Hanoi, there are three diamond needles, and in the beginning, 64 gold disks of 64 different diameters were placed on one of the needles in such a way that no disk is on top of a disk of smaller diameter. The monks must move the disks from one needle to another using all three needles and never placing a disk on top of a smaller one. The Tower of Hanoi problem is to determine their strategy for moving the disks. When they finish, the world will come to an end.

(1) Outline in detail a CPS-like solution for the Tower of Hanoi problem (if there is none carefully explain why).

(2) Outline in detail a Heuristic-DENDRAL-like solution (if there is none carefully explain why).

Problem 6.

Suppose that an exploratory robot vehicle is to be landed on Mars in 1980 with the following characteristics:

(1) It has a suitably compact computer of approximately the performance of the AI project's PDP-10.

(2) It can transmit and receive 10^6 bits per second in communication with earth when it is facing earth.

(3) It is landed at a time when the round trip for signals is 15 minutes.

(4) It has television cameras.

(5) It can move either on wheels or 6 legs provided you can program it.

With what AI vision and motion capabilities would you equip it and how would they be programmed and used? Give as much detail as you can. What other equipment and programs would you provide?

April 1972 Artificial Intelligence Qualifying Exam

(Time limit: 6 hours)

Problem 1.

(a) In terms of the quality of solutions found and the efficiency with which solutions are produced, the Heuristic DENDRAL program is one of the most powerful heuristic programs in existence. What is the primary source of this problem-solving power?

(b) In his paper on Heuristic Programming, subtitled "Ill-Structured Problems", Newell introduces concepts and terminology intended to categorize and describe heuristic programs. Use Newell's concepts and terminology to describe Heuristic DENDRAL.

(c) What are the purposes of having a systematic generator (the DENDRAL algorithm) at the heart of Heuristic DENDRAL?

(d) We use heuristic processes to achieve search reduction in administering the search for a solution to a problem. How does the heuristic process known as the Planner in Heuristic DENDRAL contribute its heuristic power to search reduction? Illustrate by making reference to some of the results in the results tables of the DENDRAL paper you were asked to read. From a heuristic search point of view, how does "planning" in DENDRAL differ from "planning" as this method has been discussed elsewhere in the A. I. literature (e.g. the Planning Method of GPS, Hewitt's Planner, Robot Planning)?

(e) You were asked to read a paper by Amarel in which he discusses representation of knowledge and shift of representation. How has this problem been studied in the context of the task environment of Heuristic DENDRAL? What are the results?

Problem 2.

The unbounded unit-preference strategy is: "Compute the resolvents of all unit clauses with every clause before computing the resolvents of any pair of non-units."

The input clause strategy is: "Compute the resolvents of a pair of clauses only if one of them is a member of the initial set of clauses (i.e. an axiom or the negation of the theorem)."

(a) Give examples showing that both of these strategies are logically incomplete.

A replacement rule of inference for equality may be defined as follows.

Let E be the equality predicate and s, t, u be terms. Let A and B be clauses with no variables in common such that A contains a positive equality atom, either $A = E(s,t) \vee A'$ or $A = E(t,s) \vee A'$, and a term u occurs at least once in B. (Note: u may occur as a subterm,) Let s and u have a common substitution instance, and suppose that α is a most general unifier such that $s\alpha = u\alpha$. Let $B*$ be the result of replacing an occurrence of $u\alpha$ in $B\alpha$ by $t\alpha$. Let C be the clause $A'\alpha \vee B*$. Then C may be inferred by replacement from A into B. Denote the set of such inferences by $P(A, B)$.

If A and B have common variables, these must be eliminated by a change of variables before the rule is applied.

(b) For all A and B, is $P(A,B) = P(B, A)$? Prove your answer.

(c) Let A be $E(f(x, g(x)), e)$ and B be $E(f(x,x), e)$. Compute $P(A, B)$ aand $P(B, A)$.

(d) *Prove:* For any C that can be inferred by replacement from A and B there is a C' satisfying (1) C' implies C and (2) C' is obtained by a sequence of resolutions from the set consisting of A, B, and the axioms for equality.

Problem 3.

After reading the speech report for inspiration, you have accepted a consulting job with the linguistics department to predict the feasibility of a speech understanding system. You are given the following vocabulary and grammar. You want a computer to recognize semantically and syntactically legal sentences. The department did not specify the semantics but any reasonable assumptions will do.

The vocabulary is:

programs, monkeys, termites,
search, climb, eat
trees, bits, bananas

The grammar is:

S → *subject* | *verb* | *object*
subject → programs | monkeys | termites
verb → search | climb | eat
object → trees | bits | bananas

(a) First, assume a probability of correct recognition of one of the vocabulary words, when isolated, to be .7. Suppose the lexical segmentation scheme is perfect. Without use of the grammar what correct string recognition rate (all words correct) might be expected on 3-word strings?

(b) Based on the results of the speech report, how might the probability of word-confusion error depend on vocabulary size?

(c) Make a reasonable assumption (either your answer to part (b) or some other guess) of the effect of vocabulary size on recognition rate. State your assumption. Now, using the grammar, but still no semantics, what correct string recognition rate might be expected? (Rough calculations are sufficient.)

(d) Specify your assumed semantically meaningful strings. Show precisely why the recognition rate is better. For extra credit, calculate an expected correct S-word string recognition rate using both syntax and semantics.

Problem 4.

Consider the following variant of the missionary and cannibals problem:

> "Three missionaries and three cannibals come to a river that they wish to cross. They find a boat that holds two people and can be rowed by one or two. However, if one person rows by himself, he will be too tired to row by himself again. Besides that, if the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. How can they all safely cross the river?"

(a) Write a LISP program to find a solution.

(b) Write a micro-PLANNER program to find a solution.

(c) Write a situation-calculus description of the situation and the effects of actions from which it follows that there is a solution. The "result" formalism of McCarthy and Hayes is recommended.

(d) Discuss the problem of making a program that could go from the above English statement of the missionary and cannibals problem to a LISP program for doing the tree search. Would the PLANNER formalism or the McCarthy and Hayes formalism be suitable as intermediate steps? Why or why not? Try to divide the overall problem into sub-problems which might be solved independently.

Problem 5.

One of the central problems in the recognition of scenes involving plane-bounded objects is the segmentation problem. Falk, in his thesis, suggests improvements to Guzman's algorithm, Describe Guzman's and Falk's algorithms. Give an example different from those in Falk's thesis where Guzman's method fails and Falk's succeeds. Give an example where they both fail.

For extra credit:

(a) Extend Falk's algorithm to cover the case you presented above. If the new algorithm doesn't cover all cases, find a counterexample.

(b) Discuss the segmentation problem for curved objects.

May 1974 Artificial Intelligence Qualifying Exam

(Time limit: 6 hours)

Problem 1. (60 *minutes)*

(a) Briefly describe each of the following concepts and discuss how it would be useful in a system working in the blocks world. This question is intended to test your understanding of the mechanisms and not of the blocks world. If one or another of the mechanisms is not particularly suited to the blocks world, then you should discuss it in the context of another domain. Also, you may want to make note of how some of the concepts are interrelated.

(1) "procedural embedding" of knowledge
(2) declarative representations (e.g. assertions, predicates)
(3) automatic backtracking
(4) alternative "worlds" of data contexts
(5) "demons"
(6) pattern matching.

(b) Define each of the following problem-solving paradigms and give a short description of the sort of problem domain to which it is applicable. (For instance, "table lookup" might be defined as the selection of data elements from an information structure by processing keys associated with the data. It would be applicable in situations where the set of possible answers is explicit, there are appropriate keys and selection functions, and the number of elements is small enough to allow the storage and retrieval to be reasonably efficient.)

(1) alpha-beta pruning
(2) generate-and-test
(3) heuristic search (e.g. A* or branch-and-bound)
(4) hill climbing
(5) means-end analysis.

Problem 2. *(JO minutes)*

Give brief descriptions of the Heuristic DENDRAL and Meta-DENDRAL programs. Identify the major components and contrast the purposes of the two programs. Specifically tell how Meta-DENDRAL makes the "big switch" in Heuristic DENDRAL powerful.

Problem 3. *(JO minutes)*

Describe the representation of knowledge in the MYCIN system (Shortliffe, et. al.). Describe how the set of rules is organized to allow for flexibility in changing the rules. Contrast this representation with a decision tree representation.

Problem 4. (60 *minutes)*

Knowledge has been called the key to high performance of AI programs and has been suggested as a major topic for new research in the coming years. Among the many dimensions of this research are the representation, use, and acquisition of knowledge in AI programs.

(a) Discuss the representation of knowledge in the Carnegie-Mellon HEARSAY system. In particular, what is the organization of the system, what knowledge is represented, and how does the system deal with multiple representation of knowledge?

(b) Discuss the use of static and dynamic world knowledge in the STRIPS robot system, especially in planning. Be sure to mention dependency conditions in plans. How does STRIPS know if a plan if applicable once there is a change in the world?

(c) One way of acquiring new knowledge is by induction. Describe the methods of acquiring knowledge, and the types of knowledge acquired, in Meta-DENDRAL, STRIPS, and Winston's arch finder. (These are three programs which can be said to do induction.)

Problem 5. (45 *minutes)*

In natural language, words have several different aspects of meaning. Among these, we might include *reference* (their connection to objects in the real world), *inference* (their connections to other information implied by their use), *connotation* (the other associations they provide for the hearer, often attitudinal or emotional), and *appropriateness conditions* (the contextual conditions under which they can appropriately be used). For each of the four words 'dog", "throw", "angry", and "always", discuss the different aspects of meaning they convey, and the way they might be handled in different systems (particularly those of Schank, Colby, Wilks, and Winograd).

Your answer should include:

(1) A discussion of the use of primitives, as in Schank. Describe how they are applicable to some of the four words, and what problems there are in extending the idea to handle others. Describe how conceptual dependency might handle some of the connotative meaning of "angry".

(2) Different approaches to representing the "essential" meaning of "dog". What would it look like in a BLOCKS-world? What components of its meaning describe its preference semantics?

(3) The various inferences that can be drawn from a use of the word 'throw", and how they would be handled in the inference components of various systems.

(4) The problem of handling time adverbs like "always", and ways they might be integrated where not present now in various systems.

(5) The kinds of patterns which might be present in a Colby-like system containing these words, and how they would be used. What aspects of their meaning are brought out?

For extra reinforcement, consider any of the other possible combinations of word, meaning type, and system. Use the specific words as starting points to discuss which aspects of meaning are and are not handled by the various programs, and to compare the ways each program deals with the same aspect of meaning.

Problem 6. (30 *minutes)*

It has often been noted that there are many similarities between the process of understanding a spoken utterance and "understanding" a visual scene. Clearly, the acoustic wave form corresponds to the pattern of intensities on the retina, phonemes correspond to lines, vertices, etc., and the S-dimensional description of the scene is in some sense the "meaning".

The following statements are made in an (imaginary) proposal for a new vision system. For each statement, identify a particular natural language system which has made significant use of the same basic idea and also tell how the idea has been applied in actual vision systems. If the idea has not been used, suggest where it might.

(a) Many of the system's responses do not require analyzing the entire scene, but depend on reactions to particular features. If a furry beast is jumping at us through the air, we want to give the appropriate response without worrying about whether it is a lion or a tiger.

(b) We expect to see objects in familiar configurations. For example, we expect to see a handset, a body, and a dial associated in a particular relationship when we look at a telephone. The system uses simple heuristics to group visual features into prospective objects, then applies its vocabulary of templates for plausible objects to each of these seeing if a fit can be found. If there is more than one possible match, we can weight the preference of each one by checking whether the individual components are of the right type for the template.

(c) The system must make use of all levels of information in a flexible way. At times we may want to use the fact that a particular object is hypothesized to look more carefully for some line. At other times, a particular configuration of lines and vertices may suggest a possible object.

(d) In order to describe complex shapes, we need a small vocabulary of basic shape-constituents, such as "plane", "sphere", "rod", etc. We can then define more complex shapes as combinations of these linked with spatial relations like "above", "inside", etc.

(e) When we walk into a room we have a set of expectations for particular elements which need to be filled in, e.g. walls, floor, ceiling. This "frame" determines the way in which we will interpret the elements we see.

Problem 7. *(30 minutes)*

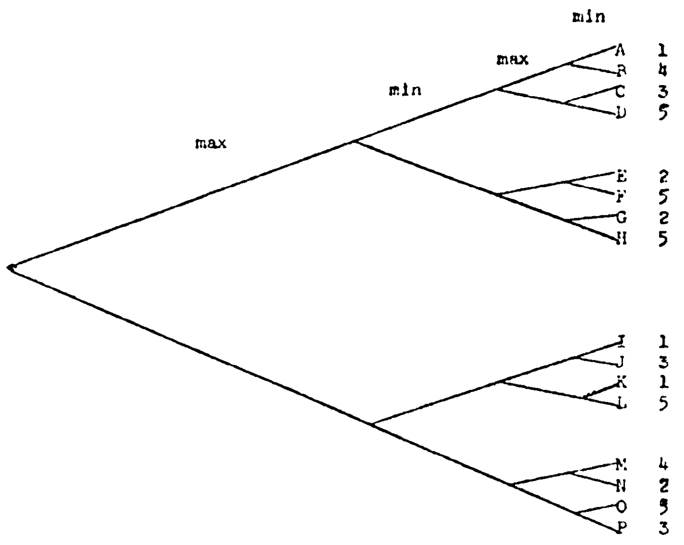
Consider the game tree shown below, and let moves from any vertex be generated from top to bottom. Subtrees are to be named by giving the letters attached to their terminal vertices. We will use the α-β heuristic.

(a) What subtree is visited if initially a $= -\infty$ and $\beta = \infty$? What does it tell us about the value of the game?

(b) What subtree is visited if initially a $=$ 1.5 and $\beta =$ 2.51

(c) What subtree is visited if initially a $=$ 3 and $\beta = \infty$? What does it tell us about the value of the game?

(d) Re-order the subnodes so that by using an α-β depth-first, top-to-bottom search, the smallest number of terminal vertices is visited. There may be many such re-orderings; you need find only one.



(Do any two of problems 8, 9, and 10.)

Problem 8. *(60 minutes)*

Consider the interactive Program-Writing System in Floyd's 1971 *IFIP* paper Toward Interactive Design of Correct Programs". Recall that there are two parts of the system, which are referred to respectively as "man" and "computer" in the imaginary dialogue. In the following problems, you are to show in some detail what "mechanizable" reasoning along with what necessary facts would lead to each of the desired results. You may use any plausible formalism or pseudo-formalism such as Floyd's, first-order logic, or micro-planner. As a suggestion, first sketch out the reasoning in any language you wish, such as conventional mathematical language and English. Then reduce it to a formalism. You will receive most of the credit if your reasoning is complete even if you can't express it in a known formalism.

(a) On the second page of the paper, "computer" states that the antecedent of P_2 does not follow from the antecedent of P_1 and the iteration-non-terminated condition. "Computer" also implies that the iteration is not initialized. Show how these two conclusions can be reached.

(b) In the next interchange, "computer" states that the consequent of P_2, with the iteration termination condition, does not "seem" to imply the consequent of P_1. Sketch out the reasoning and information necessary to reach this conclusion. What added facts or inferences would be necessary to "prove" that the consequent of P_1 is not implied by the consequent of P_2?

(c) Just before the end of the dialogue, the computer asks the man to design the initialization of the sub-program P_2. Suppose we want "computer" to design the initialization itself. Show how "computer" might deduce the correct initialization.

Problem 9. (60 *minutes)*

Read the paper "A Semantics-Based Decision Theory Region Analyzer" by Yakimovsky and Feldman (3rd IJCAI, 1973). Consider the problem of applying their region analyzer to the blocks world.

(a) What would you do to try it?

(b) What results would you expect?

(c) Now think about adding the vertex classification system of Waltz. What changes in the abstract description of the Y & F system and what changes in the program organization would be needed? How well would this system work?

(d) What does your answer suggest in general about vision research?

Problem 10. (60 *minutes)*

For a position u in tic-tat-toe, *winnable(u)* asserts that the player whose turn it is to move can force a win. Write sentences of predicate calculus axiomatizing *winnable(u)*. Assume the standard interpretation of the usual set theoretic and arithmetic predicates and operations so these don't have to be axiomatized. Use the following initial definitions.

$$\text{Board} = \{1, 2, 3\} \times \{1, 2, 3\}$$

$$\forall u\ u \in \mathit{Positions} \equiv \exists xyz\ u = \langle x, \langle y, z\rangle\rangle \wedge$$
$$x \in \{\text{"x"}, \text{"o"}\} \wedge y \in \mathit{Powerset}(\mathit{Board}) \wedge z \in \mathit{Powerset}(\mathit{Board}) \wedge y \cap z = \{\}$$

In this formula, x is the player whose turn it is, y is the set of squares occupied by "x", and z is the set of squares occupied by "o".

Use car and cdr to extract elements of ordered pairs so that

$$\forall xy\ \mathrm{car}(\langle x, y\rangle) = x \wedge \mathrm{cdr}(\langle x, y\rangle) = y$$

Hint: Use a predicate *won(u)* that is true if the player that last moved has three in a row.

May 1975 Artificial Intelligence Qualifying Exam

Select *one* of the following problems. It should *not* be in your area of **A.** I. specialization. Spend a few days on the problem (about 10 hours altogether), and turn in a copy of your work. Your written work will serve as the basis for the discussion at the oral exam.

All papers referred to below are from the Proceedings of the Third International Joint Conference on Artificial Intelligence **(31 JCAI).**

Problem 1.

Read Pohl (p. 12) and Harris (p. 23).

(a) Pohl doesn't distinguish between time exhaustion and space exhaustion kinds of Type-l catastrophes, but hints that there might be some things to say about them. Can you think of anything? How can their frequency be minimized by more careful planning while a system is being designed7 What are some ways one might detect an imminent collapse? Once detected, how could its impact be minimized?

(b) Propose a simple problem which might lead to an unending search unless **Pohl's** dynamic weighting scheme is used. Could Harris' bandwidth constraint solve this problem as well?

(c) Harris and Pohl both use the Travelling Salesman problem to illustrate their techniques. Why is this an apt choice? Does it facilitate comparing the two techniques? (Warning: If it does, then compare them!)

(c) Given a resolution theorem-prover, might these techniques be **applied** to advantage?

Problem 2.

Read Bledsoe (p. 56). The system described is dealing with a domain (topology) but does not seem to possess much knowledge **in a** format suited to that **domain**. For example, humans rely on visual intuitions about space and continuity quite frequently while attempting proofs in this field. **How** might analogical models like Gelernter's or Bundy's (p. 130) be employed by the system?

In 1973, Bledsoe seemed to expect his system to prove new topology theorems any day. If today, two years later, this isn't so, how can you account for this? That is, how could **it** be that a system was able to prove hard but known theorems, yet not **a** single "new" interesting one?

Problem 3.

Read Woods (p. ZOO) and **a** HEARSAY article. Describe how incremental simulation might have been effectively employed for some non-speech projects (e.g. the Dendral task or a large theorem-prover). Why are so many different experts used in Woods' project? How do these experts correspond to the modules in HEARSAY (Reddy et al, p. 185, **p. 194)? How** might you organize an incremental simulation of a system to maintain and draw inferences from Conceptual Dependency nets **(Schank, p. 255)?** To what extent has the basic idea permeated AI research? Consider, for example, the “pre-processed” format for input assumed by Winston’s ARCH-learning system. Is incremental simulation meaningful for an individual doing research? Is there a “flaw” in the idea; can you think of a situation where it might be detrimental to a final project?

Problem 4.

Read Darlington and Burstall (p. 479) and Boyer and Moore (p. 486). Could Boyer and Moore’s system be programmed using the transformation schemata concept of Darlington and Burstall? Write one schema for a particular induction problem (e.g. proving that *append* is associative), and indicate how it would be applied. Is this feasable for the entire range of Boyer and Moore’s system’s abilities? Consider how one might automate the acquisition of new program transformation schemata. For example, consider how META-Dendral automates rule acquisition for Dendral (see Buchanan, p. 67, or Buchanan et al in the references list).

Problem 5.

Read about Waltz’s work; also, look at Stefanuk (p. 612). Stefanuk and others seem to argue for the use of local processing to solve problems. Give a few examples where this is essential, and a few where only a global attack can get anywhere easily. How might one characterize the class of problems which need a particular level of scrutiny? How does this mirror the problem of using semantic vs. syntactic knowledge?

Draw an analogy between local/global knowledge and the use of phonetic/semantic knowledge in speech systems. Consider how a successful speech system uses the different levels of knowledge synergetically (e.g. HEARSAY). Using your analogy, how might one intermix local and global kinds of knowledge of the kind used by Waltz? That is, use your analogy to propose a new design for Waltz’s system.

Problem 6.

Sketch out a program to play tic-tat-toe in one of the AI languages (micro-planner, conniver, QA4). This is *not* a programming problem. You are not expected to run the program. What is important is a discussion of the basic representations you use and the tradeoffs involved in the way the program is designed. In particular discuss how your design would differ if the program were for 3-D 4 x 4 tic-tat-toe.

Problem 7.

The first paper in 3IJCAI describes a method for searching “additive and/or graphs”. Find some real AI search problem (where “real” means in a specific problem domain like game playing, syntactic parsing, or scene analysis) to which this approach might be applied. Discuss why the algorithm would be useful, and the tradeoffs involved in the different choices (like top-down vs. bottom-up). If you can find two domains with different characteristics, all the better.

Problem 8.

Read Koffman & Blount, "Artificial Intelligence and Automatic Programming **in** CAI" (p. 86). Assume you wanted to write a program which would take the role of the STUDENT (i.e. the other participant in a dialog with their program). Discuss the basic issues you would need to handle, and the ways in which your result would be the same (or different) as a general automatic **programming** system.

Problem 9.

DENDRAL takes the results of a complex interactive event and tries to deduce what components went into it. Consider applying the same techniques to a "de-compiler" which takes machine code and tries to deduce the higher level language code which produced it. Pick your own favorite machine and higher level language, but assume that the system will have to handle the output of aribtrary compilers (all correct, but some involving optimitations and other such complications). Describe what the resulting system would look like, in particular pointing out which DENDRAL features seem useful and which don't. Reminder: This is a thought problem, *not* a programming problem – don't try to build the **whole** thing, but spend your time figuring out what the significant issues are.

Problem IO.

John Seeley Brown has a paper on "Steps toward automatic theory formation" (p. 121). It discusses a task involving learning names for kinship relations. Describe how Winston's learning program would have to be modified to handle the learning of kinship relation names. Discuss the problems in designing an appropriate training sequence. Discuss the relationship between these and other "concept formation" programs which might be set the same task.

Problem I l.

Harry Pople ("On the mechanization of abductive logic", p.147) describes a formalism for abductive reasoning in the context of medical diagnosis. Describe a production system-based method for doing the reasoning. How could it be fit into MYCIN?

Problem 12.

Three papers in 3IJCAI disucss problem solvers in the domain of moving objects from room to room in a simple flwrplan (Siklossy & Roach, p. 383; Sacerdoti, p. 412; and Siklossy & Dreussi, p. 423). Describe the relative advantages and disadvantages of each (in particular look for things one system advertises doing which would be difficult or impossible for the others, and **analyze why).**

May 1977 Artificial Intelligence Qualifying Exam

Be prepared to discuss the following questions orally.

One computer program says about another, "It knows I want to use the telephone line to Boston, and it is deliberately holding on to it in order to prevent my using it."

1. English aside, how would you represent such an assertion as a LISP, PLANNER, or first-order logic data structure?

2. What semantics would you give this assertion, that is, in what states of the world would you regard it as true?

3. How would you axiomatize the concepts involved, what rules of inference would you use, and/or what MICROPLANNER "theorems" involving them would you give a program that must generate such a statement and use it?

4. From what evidence might the computer deduce such a statement? E.g. from what external observations?

5. When if ever would it be important for a computer program to be able to use such assertions?

May 1965 Numerical Analysis Qualifying Exam

Problem 1.

In five minutes or less, state how you would go about locating the **most** accurate published decimal approximation of $1/\pi$, where $\pi = 3.14\,159\ldots$

Problem 2.

Suppose we want to numerically evaluate $\int_0^1 e^{x^2}\,dx$ with an error provable to be less then 10^{-10}.

We are able to generate values of e^{x^2} for any x in $[0,1]$ to any desired accuracy. We decide to use the trapezoidal method (with no acceleration) with n t 1 abscissas. Assume ordinary rounded floating-decimal arithmetic, with s significant decimal digits for the mantissas. We may use any integer values of n and s that we need, but we must not waste resources with values that are much too large.

(a) Approximately what values of n and s are large enough to do the job?

(b) For some reasonable pair of values n, s, show that the error is indeed less than 10^{-10}.

(c) What error would you expect to actually occur in a computation with this n and s?

Problem 3.

Suppose that the equation $x^2 + a_1x + a_2 = 0$ with real coefficients possesses real roots a, β. Show that, if x_0 is chosen sufficiently close to a, the iteration

$$x_{k+1} = -\frac{a_1x_k + a_2}{x_k}$$

converges to a if $|a| > |\beta|$; the iteration

$$x_{k+1} = -\frac{a_2}{x_k + a_1}$$

converges to a if la $| < |\beta|$; and the iteration

$$x_{k+1} \;\text{c}\; \frac{x_k^2 + a_2}{a_1}$$

converges to a if $2|\alpha| < |\alpha + \beta|$.

Problem 4.

Let $\lambda(h)$ be a function of h. Assume that

$$\lambda(h) = \lambda(0) + ah^{4/3} + bh^2 + o(h^2) \quad \text{as } h \to 0,$$

where a and b are unknown constants. Assume further that the values of $\lambda(h)$ are known for $h = 1/10, 1/20, \ldots, 1/90, 1/100$. Describe an extrapolation to the limit algorithm which makes use of the assumed information to estimate $\lambda(0)$.

Problem 5.

Let A be a real $n \times n$ matrix with eigenvalues $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. Let $Az_1 = \lambda_1 z_1$. Consider the algorithm which generates the following sequence of vectors:

$$y^{(i+1)} = Ax^{(i)}$$
$$x^{(i+1)} = \frac{1}{\|y^{(i+1)}\|} y^{(i+1)}$$

where $x^{(0)}$ is an arbitrary real vector with unit norm (we use the Euclidean norm).

(a) Under what conditions will $x^{(i)}$ converge to z_1?

(b) If $\lambda_1 = \rho e^{i\theta}$, $\lambda_2 = \rho e^{-i\theta}$, and $|\lambda_2| > |\lambda_3|$, describe a method for computing λ_1 and λ_2, using only real arithmetic.

(c) Describe ways of improving the rate of convergence of the above algorithm when $|\lambda_1|$ and $|\lambda_2|$ are close.

(d) Briefly mention the advantages and disadvantages of the power method over other methods.

Problem 6.

The following algorithm has been used to generate circles on the CRT display of the PDP-1. Given x_0, y_0, and h, let

$$x_{n+1} = x_n + hy_n$$
$$y_{n+1} = y_n - hx_{n+1}.$$

(Note particularly that the "new" value of x is used to obtain the new value of y.) The points (x_n, y_n) are displayed as they are calculated and appear to have an almost constant distance from the center of the screen. In practice, h is of the form 2^{-l}, but this is irrelevant here.

Express the algorithm in the form

$$z_{n+1} = A(h)\, z_n, \quad \text{where } z_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix},$$

for some matrix A(h). Use this to show that the algorithm works by proving that there is a constant c so that for all n,

$$\frac{1}{c} \|z_0\| \le \|z_n\| \le c\, \|z_0\|,$$

where $\|z\|$ is the Euclidean vector norm. You may ignore roundoff errors.

If the original algorithm is changed to

$$\begin{aligned} x_{n+1} &= x_n + hy_n \\ y_{n+1} &= y_n - hx_n, \end{aligned}$$

then we obtain an expanding spiral instead of a circle. Why?

Make a few brief comments relating the above observations to the numerical solution of ordinary differential equations.

Problem 7.

Consider the ordinary differential equation problem of determining y(x) so that

$$\frac{dy}{dx} = f(x, y), \quad y(0) = y_0. \tag{1}$$

In the Milne-Simpson method of approximately solving (1), we solve the difference equation

$$y_{n+1} = y_{n-1} + \frac{h}{3}\left(f(x_{n-1}, y_{n-1}) + 4f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right) \tag{2}$$

where $x_n = nh$ and y_n is an approximation to $y(x_n)$. The actual methods of getting y_1 and solving the implicit equation (2) for y_{n+1} are irrelevant here.

(a) In spite of the universal success enjoyed by the Milne-Simpson method with desk calculators, it has not been popular with automatic computers, Why?

(b) As explicitly as you can, find the solutions of (2) for the two cases $f(x, y) = y$ and $f(x, y) = -y$. Explain the relevance to part (a).

Problem 8.

In one hour, discuss in depth one of the following areas of numerical analysis:

(1) Numerical solution of ordinary differential equations, including discretization error and stability.

(2) Numerical solution of elliptic partial differential equations.

(3) Numerical solution of hyperbolic partial differential equations,

(4) Numerical solution of parabolic partial differential equations.

(5) Round-off error.

(6) Approximation of functions.

(7) Computational methods in linear algebra.

(8) Numerical integration.

(9) Monte Carlo methods.

Include in your discussions as many of the following subjects as seem to be appropriate.

(a) The important problems in the area.

(b) The most important results.

(c) General literature which would provide an introduction to the field for a person who wanted to learn about the area. Also some of the more recent literature in which important results are given.

(d) The pragmatics of solving problems in the field on an automatic digital computer. Also give sources of routines or algorithms for solving the common problems of the field;

(e) Some unsolved problems in the field.

March 1966 Numerical Analysis Qualifying Exam

Problem 1.

You are required to evaulate

$$\int_{10}^{\infty} \sqrt{\frac{1}{\ln(x+1)\ln(x-1)} - \frac{1}{(\ln x)^2}}\,\frac{dx}{x}.$$

You may use any quadrature programs you want, and up to two minutes of time on the B5500. Spend 20 minutes now, with closed book, to outline what you propose to do and why. Execute your proposal this afternoon and hand in your result together with supporting documentation and an explanation of any deviation from the plan you propose now.

Your grade will depend upon

(1) The accuracy you achieve.

(2) The strength of the evidence *or* argument which you supply to support your claims to accuracy.

(3) The total amount of computer time (debug+compile+execute) consumed. You will be penalized for computer time consumed in excess of two minutes.

Do two of Problems 2-6 (20 minutes each).

Problem 2.

Let $\{x_i\}$ be a sequence of real numbers converging to α. Let z_i be a sequence defined by

$$z_i = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i}.$$

(a) Calculate z_i if

$$x_{i+1} - \alpha = K(x_i - \alpha),$$

for some K, where $|K| < 1$.

(b) Let $x_{i+1} - \alpha = (K + \sigma_i)(x_i - \alpha)$ for some $|K| < 1$, and where $\sigma_i \to 0$. Prove that

$$\lim_{i\to\infty} \frac{z_i - \alpha}{x_i - \alpha} = 0.$$

(c) What hypotheses on σ_i would enable you to conclude that

$$\lim_{i\to\infty} \frac{z_i - \alpha}{(x_i - \alpha)^2}$$

exists? Prove your assertion.

Problem 3.

Describe what is meant by "instability" in relation to the numerical solution of differential equations.

Find the greatest step length h such that the initial value problem

$$y'' + ay' + by = 0, \text{ where } 0 < a < 2\sqrt{b}$$

may be solved without instability, using

$$y_{n+1} = y_n + h y'_n$$

to perform the integrations.

Problem 4.

Consider the following ALGOL procedure:

```
real procedure lg(x) value x; real x;
comment lg(x) = log_e(1+x) to within a few units in its last decimal place,
        provided the ALGOL function ln(y) = log_e(y) to within a unit or two
        in its last decimal place for all y > 0.
        The simple statement
                lg := ln(1+x)
        was rejected because it produces inaccurate results when x is near zero.;
begin
        real y;
        y := 1+x;
        if y = 1 then lg := x
        else lg := x * ln(y)/(y- 1)
end
```

Explain why the rejected statement produces inaccurate results and show that the procedure does work as accurately as claimed when run on a computer, like the 85500 or IBM 7090, which normalizes sums and differences of floating-point numbers before rounding.

Problem 5.

Consider the following ALGOL procedure:

```
real procedure rufsqrt (x, a, b); value x, a, b; real x, a, b;
begin
        real t;
        t := a*x + b;
        rufsqrt := (x/t + t) * 0.5
end
```

Let $E(x, a, b) = (s/\sqrt{x}) - 1$ be the relative error with which $s = rufsqrt(x, a, b)$ approximates $\sqrt{x}$. Let

$$\bar{E}(a, b) = \max_{1/4 \le x \le 4} |E(x, a, b)|.$$

Prove that

$$\bar{E}(a, b) = \bar{E}(b, a) \ge \bar{E}\left(\frac{a+b}{2}, \frac{a+b}{2}\right)$$

and hence that the values of a and b which minimize $\bar{E}(a, b)$ are equal.

Problem 6.

Read the paper "Note on the inversion of symmetric matrices by the Gauss-Jordan method,,, by R. DeMeersman and L. Schotsmans (see /CC Bull 3 (1964), pp. 152-155).

The authors claim that their algorithm will work for any symmetric non-singular matrix, but it won't. Supply a 2 x2 counter-example.

Do two of Problems 7-12 (60 minutes each).

Problem 7.

(a) Let

$$C(\alpha) = \begin{pmatrix} \alpha & -1 & 0 & & \cdots & 0 \\ -1 & \alpha & -1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & & \vdots \\ 0 & 0 & -1 & \alpha & -1 & 0 \\ 0 & \cdots & 0 & -1 & \alpha & -1 \\ 0 & \cdots & & 0 & -1 & a \end{pmatrix}$$

be a real $n \times n$ tridiagonal matrix.

Give a simple condition on a for C(a) to be (1) non-singular and (2) positive definite, citing reasons for your answer.

For parts (b)-(f), consider the real $n \times n$ tridiagonal matrix

$$A = \begin{pmatrix} \alpha_1 & -1 & 0 & & \cdots & 0 \\ -1 & \alpha_2 & -1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & & \vdots \\ 0 & 0 & -1 & \alpha_{n-2} & -1 & 0 \\ 0 & \cdots & 0 & -1 & \alpha_{n-1} & -1 \\ 0 & \cdots & & 0 & -1 & \alpha_n \end{pmatrix}$$

(b) Answer part (a) for the matrix A.

(c) If A is non-singular how would you propose to solve Rx = b? Give more than one method, giving reasons when each method is applicable.

(d) The inverse of A sometimes has only positive entries. When is this true for a non-singular A? Why is this knowledge useful?

(e) Show that if A^{-1} exists and has nonnegative entries then A has at least one nonnegative eigenvalue and eigenvector with nonnegative entries.

(f) Suppose that $\alpha_i \geq 2$ and $|b_i - c_i| \leq E$ for $i = 1, 2, \ldots, n$. Let x and y be the respective solutions of Rx = b and $Ay = c$. Find an estimate for max $|y_i - x_i|$.

(g) Consider the problem

$$(-p(x)\,\phi_x(x))_x + f(x)\,\phi(x) = g(x)$$
$$\phi(a) = \phi_a,\ \phi(b) = \phi_b$$

where $p(x) \geq$ a > 0, $f(x) \geq 0$, $a \leq x \leq b$.

Discuss the solution of this problem by finite-difference techniques.

Problem 8.

(a) The coefficients a_r and b_r are related by Horner's recurrence

β: $b_n = a_n$, and for $r =$ n-l, *n-2*, . . . , 1, 0,
I $b_r = x b_{r+1} + a_r$.

Show that the polynomials

$$A(z) \equiv \sum_{r-0}^{n} a_r z^r \text{ and } B(z) \equiv \sum_{r=1}^{n} b_r z^{r-1}$$

satisfy

$$A(x) \equiv (z-x)\ B(z) \text{ t } b_0.$$

(b) The recurrence β can be implemented as an ALGOL program which, given computer numbers n, x, and a,., generates the coefficients b_r. However, rounding errors will prevent the stored values b_r from satisfying recurrence β precisely. To appraise these errors, assume that the ALGOL statement

$s := u$ t v

when executed, produces a number s satisfying

$|u+v-s|/|s| \le \sigma$

and that

$p := u * v$

produces a number p satisfying

I uv-p $|/|uv$ I $\le \pi$,

where σ and π are small numbers (of the order of 10^{-11} on the B5500). Then show that the computed value of b_0 satisfies the following quite close bound:

$|A(x)-b_0| \le (\sigma+\pi)e_0 - |b_0|\pi$, where

ϵ: $e_n = |a_n|\pi/(\sigma+\pi)$, and for $r =$ n-l, *n-2*, ... 1, 0,
I $e_r = |x|e_{r+1} + |b_r|$.

If you cannot prove this, give another reasonably close bound for $|A(x) - b_0|$.

(c) Now think about an ALGOL program which uses, say Newton's iteration to compute a zero of the polynomial A(r), and which includes both recurrences β and ϵ, the latter to provide an error-bound for the former. Discuss the suitability of the following criterion for stopping iteration, in the light of practical considerations:

"If $|b_0| \le (\sigma+\pi)e_0 - |b_0|\pi$, then x is an acceptable approximation to a zero of *A(r).*"

Problem 9.

For the solution of systems of linear equations Ax = *b* by Gaussian elimination, discuss the motivation for and the consequences of various pivot-selection strategies most widely used. Explain how these strategies can be carried out in a program for solving linear equations. Discuss any other operations on the system which might be relevant in the pivot-selection strategies. What inferences, if any, can be drawn from the sizes of the pivots7

You may wish to illustrate the points of your discussion'by means of numerical examples either of your own choosing or from the following:

(i)

$$A_1 = \begin{bmatrix} 1 & -1 & -1 & \cdots & -1 \\ 0 & 1 & -1 & \cdots & -1 \\ 0 & 0 & 1 & \cdots & -1 \\ \vdots & & & \ddots & \vdots \\ \vdots & & & 1 & -1 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}_{60\times 60} \qquad \begin{aligned} a_{ii} &= 1 \;. \\ a_{ij} &= -1 \text{ if } i<j \;. \\ a_{ij} &= 0 \text{ if } i>j \;. \end{aligned}$$

(ii)

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & 0 \cdots & 0 & 1 \\ \vdots & & \ddots & \ddots & \vdots & \vdots \\ \vdots & & & 1 & 0 & 1 \\ -1 & -1 & \cdots & -1 & 1 & 1 \\ -1 & -1 & \cdots & -1 & -1 & 2 \end{bmatrix}_{60\times 60} \qquad \begin{aligned} &a_{ii} = 1 \text{ except} \\ &a_{60,60} = 2 \;. \\ &a_{ij} = -1 \text{ if } i > j \;. \\ &a_{ij} = 0 \text{ if } i<j \text{ except} \\ &a_{i,60} = 1 \text{ if } i<60 \;. \end{aligned}$$

(iii)

$$A_3 = \begin{bmatrix} 1/2 & & & & & \\ & 1/4 & & & \bigcirc & \\ & & 1/8 & & & \\ & & & \ddots & & \\ & & & & 2^{-58} & \\ & \bigcirc & & & 2^{-59} & \\ & & & & & 2^{-59} \end{bmatrix} \cdot \begin{bmatrix} 2 & \\ & \\ \bigcirc & 4\;8 \ldots 2^{0}\;2^{59}\;2^{58}\;2 \end{bmatrix}$$

(The last diagonal elements deviate from the geometric progressions.)

Problem 10.

Discuss the role that the concept of the *order* of an iteration function plays in the theory of root-finding. You might want to consider some of the following points in your discussion:

(1) Definition of order.

(2) Methods for generating iteration functions of arbitrary order,

(3) When order is integral and when it is not.

(4) Effects of multiple zeros on order.

(5) Derive the order of a number of iteration functions.

(6) The convergence properties of iteration functions of linear or superlinear order.

(7) The pros and cons of using high-order methods in practice.

(8) The relation between the order of an iteration function and the function evaluations it requires.

Problem 11.

Let $\phi(x,y)$ be a solution of

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = f(x,y)\phi - g(x,y)\,, \quad f \geq 0$$

for ϕ in an L-shaped region (a square with a quarter of the square removed), where the value of ϕ on the boundary is given. Assumef and g are smooth.

(a) Discuss the approximation of this problem by finite difference techniques.

(b) Discuss several schemes for solving the resulting approximation.

(c) What role can the maximum principle play in the analysis of this problem and the approximation problem?

(d) Explain how an estimate for the difference between the analytic and approximate solutions may be obtained.

Problem 12.

(a) Discuss the properties and tests relevant to a good random number generator.

(b) Propose a numerical problem for which Monte Carlo methods offer the only reasonable approach and describe how you would use them to solve **it.** How do you estimate the accuracy of your result?

October 1966 Numerical Analysis Qualifying Exam

Problem 1.

Let

$$Q_n(x) = x^n + a_{n-1}x^{n-1} + \dots + a_0$$

'be a polynomial of degree n with leading coefficient 1. A polynomial $P_{n-1}(x)$ of degree n-l is desired such that the maximum of the error $|Q_n(x) - P_{n-1}(x)|$ is minimized on the closed interval $[-1, 1]$. Let

$$E_n(x) = Q_n(x) - P_{n-1}(x).$$

Give a characterization of $E_n(x)$ and $P_{n-1}(x)$. What is the value of

$$\max_{-1 \le x \le 1} |E_n(x)| ?$$

Problem 2.

In many mathematical applications it is necessary to compute the eigenvalues of matrices. Describe an algorithm for each one of the following problems:

(1) Find all the eigenvalues of a real symmetric matrix.

(2) Find the largest three eigenvalues and associated eigenvectors of a sparse matrix of order 500.

(3) Find all the eigenvalues of an arbitrary real matrix.

Give reasons why the algorithm chosen is especially applicable to the corresponding problem. Briefly, what modifications would you recommend if an auxiliary storage is to be used?

Problem 3.

Let $\|\cdot\|$ be a vector norm defined in n-dimensional real vector space *(n* is finite). The least upper bound norm $\text{lub}(A)$ of a real n x n matrix A with respect to $\|\cdot\|$ is defined as

$$\text{lub}(A) = \max_{x \ne 0} \frac{\|Ax\|}{\|x\|} .$$

(a) For which matrices A is $\text{lub}(A) = 0$?

(b) What is $\text{lub}(A)$ with respect to $\|x\| = \max(|x_1|, \dots, |x_n|)$ for

$$A = \begin{pmatrix} 5 & 1 \\ 3 & -4 \end{pmatrix} ?$$

(c) We could define $\text{cond}(A)$ by

$$\text{cond}(A) = \begin{cases} \infty & \text{if A is singular} \\ \text{lub}(A)\ \text{lub}(A^{-1}) & \text{if A is nonsingular.} \end{cases}$$

What range of values can $\text{cond}(A)$ assume?

(d) Consider the system of linear equations $Ax = b$, where A is a known $n \times n$ real matrix, the real vector b is known, and the vector x is unknown. We want to check the sensitivity of the solution x to changes in b. If

$$A(x + Ax) = b + \Delta b,$$

derive an expression for an upper bound for the relative change $\| Ax \| / \| x \|$ in x (with respect to $\|\cdot\|$) in terms of $\text{cond}(A)$ and the relative change $\| A6 \| / \| b \|$ in b.

(e) What does the phrase "to scale the matrix A" mean? How may it help the problem of decreasing the sensitivity of the solution x to perturbations in b, as described in part (d)?

(f) Suppose we want to solve the system $Ax = b$ for x with a particular computer algorithm. The computer, however, works with the approximations $A + AA$ to A and b + Ab to b, that is, it attempts to solve the system $(A + \Delta A)x = b$ + Ab. In a few sentences, describe what one means by a *backward error analysis* of the algorithm in question and state when the algorithm will produce a suitable solution.

Problem 4.

Describe the role of orthogonal polynomials in the derivation of quadrature formulas of the Gaussian type. Determine by this method or any other method the weights and points for the two-point formula

$$\int_0^1 x^{1/3} f(x)\,dx \sim w_1 f(x_1) + w_2 f(x_2)$$

which is exact for cubic polynomials.

Problem 5.

One of the calculations which arises frequently in statistics is

$$S = \sum_{i=1}^{n} (x_i - \bar{x})^2 , \quad \text{where } \bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}.$$

A short manipulation shows that

$$S = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2.$$

Give advantages and disadvantages of these formulas for computing S, with emphasis on data handling, number of operations, and numerical accuracy.

Suppose that the relative error in addition, subtraction, multiplication, and division is bounded by a positive number ϵ, e.g. $fl(a+b) = (a+b)(1+\epsilon_+), |\epsilon_+| \le \epsilon$, where $fl(a+b)$ indicates the floating-point sum of a and b. Determine a bound for the relative error in the computed mean $fl(\bar{x})$ when $x_i \ge 0$.

Problem 6.

(a) Explain what is meant by the *order* of a convergent iteration method.

(b) Give conditions sufficient for the convergence of the iteration

$$x_{i+1} = f(x_i)$$

to a real root of the equation $x - f(x) = 0$, given x_0.

(c) It is proposed to use the iteration sequence

$$x_{i+1} = \alpha x_i + (1-\alpha)f(x_i)$$

to determine a root of the above equation, where it is known only that the derivative of f lies in the range

$$-G \le f'(x) \le 0, \ (G > 0).$$

Show that a safe choice of α, considering the most unfavorable values that $f'(x)$ can attain, minimizes the maximum value of

$$|\alpha + (1-\alpha)f'(x)|$$

and the minimum is achieved when

$$\alpha = \frac{G}{G+2}.$$

Problem 7.

Consider the differential equation

$$y' = f(x, y)$$

with $y(a) = \alpha$. Euler's method for the numerical solution is defined by

$$y_{n+1} = y_n + hf(x_n, y_n)$$
$$x_{n+1} = x_n + h,$$

for $n = 0, 1, 2, \ldots$ with $y_0 = \alpha$, $x_0 = a$.

(a) Explain briefly how this algorithm is obtained.

(b) Assume

(1) $y''(x)$ is continuous for $a \le x \le b$

(2) $|f(x, y) - f(x, y^*)| \le L|y - y^*|$ for any $x \in [a, b]$.

Let $h = (b-a)/N$. Show that $y(b) - y_N \to 0$ as $N \to \infty$.

Problem 8.

(This problem is the same as Problem 8 of the May 1965 Exam.)

March 1967 Numerical Analysis Qualifying Exam

Problem 1.

(a) Explain the fundamental ideas behind the Romberg integration method.

(b) If one tries to apply the Romberg method to the evaluation of the integral

$$\int_0^1 \sqrt{x} \cos x \, dx$$

one finds that the method converges slowly if at all. Explain why this difficulty arises. What can be done to eliminate this difficulty?

(c) The ideas of the Romberg integration can be applied to the approximation of the value at $h =$ 0 of a function of h which can be calculated for a set of values of $h > 0$. Suppose that

$$X(h) = X(0) + ah^{3/2} + bh^2 \text{ t } o(h^2) \text{ as } h \to 0$$

where a and b are unknown constants. Assume that values of X(h) have been calculated for $h = 1/10, 1/20, 1/40, 1/80$. Describe an algorithm based on the same ideas of the Romberg integration algorithm which makes use of the calculated information to estimate A(0).

Problem 2.

We wish to determine error bounds for some of the basic *complex* single precision floating-point operations, Assume

$$fl\ (a \pm b) = (a \pm b)(1 + \epsilon_+)$$
$$fl\ (a \times b) = ab\ (1 + \epsilon_\times)$$
$$fl\ (a \div b) = a/b\ (1 + \epsilon_\div)$$
$$fl\ (a^{1/2}) = a^{1/2}(1 + \epsilon_s)$$

where

$$|\epsilon_+|,\ |\epsilon_\times|,\ |\epsilon_\div|,\ |\epsilon_s| \le \epsilon.$$

Let $z_1 = x_1 + iy_1$ and $z_2 = x_2 + iy_2$ where $x_1,\ x_2, y_1, y_2$ are single precision floating-point numbers.

Determine a bound for each of the following quantities.

(a) $|fl(z_1 + z_2) - (z_1 + z_2)|$

(b) $|fl(z_1 \times z_2) - z_1 z_2|$

(c) $|fl(|z_1|) - |z_1||$

Problem 3.

In the Newton-Raphson method for finding a root a of an equation $f(x) = 0$ we start with z_0 and calculate the sequence $\{z_k\}$ using the recurrence relation

$$z_{k+1} = z_k - \frac{f(z_k)}{f'(z_k)}.$$

If $z_k \to a$, the convergence is known to be of second order.

(a) In order to save the labor of calculating the derivative at each step, it has been proposed to replace $f'(z_k)$ by $f'(z_0)$ in all steps. In this method the recurrence relation is

$$z_{k+1} = z_k - \frac{f(z_k)}{f'(z_0)}.$$

Show that if $z_k \to$ a, this method has convergence of first order. Obtain a condition, involving derivatives of f, which is necessary for convergence.

(b) In order to obtain more rapid convergence than in part (a) and still reduce the labor of calculating the derivatives, it has been proposed that the derivative be calculated every other step. This method is described by the recurrence relations

$$z_{2k+1} = z_{2k} - \frac{f(z_{2k})}{f'(z_{2k})}$$
$$z_{2k+2} = z_{2k+1} - \frac{f(z_{2k+1})}{f'(z_{2k})}.$$

Assume that $z_k \to$ a and find the order of convergence of this method.

Problem 4.

Consider the following numerical integration methods for solving an ordinary differential equation of the form $y' = f(x, y)$.

(1) $y_{n+1} = \frac{1}{2}y_n - \frac{1}{4}y_{n-1} + \frac{h}{3}(2y'_n + y'_{n-1})$

(2) $y_{n+1} = y_n + h(2y'_n - y'_{n-1})$ (predictor)

$y_{n+1} = y_n + \frac{h}{2}(y'_{n+1} + y'_n)$ (corrector)

(3) Same as (2) but using the corrector only once for each step.

(a) What is meant by the condition of consistency for a numerical integration formula? Which of the above formulas satisfy this condition?

(b) Suppose we wish to solve the differential equation problem

$$y' + Ky = 0, \quad K > 0$$
$$y(0) = 1$$

by a numerical integration method. Explain what is meant by the stability of such a method. For what values of K, if any, are the methods (1), (2), (3) above stable?

Problem 5.

Let $f(x)$ be a continuous function on the closed interval $[a,b]$. Let $p_n^*(x)$ be the unique polynomial of degree n for which

$$\max_{a\le x\le b} |f(x) - p_n^*(x)| < \max_{a\le x\le b} |f(x) - p_n(x)|$$

where $p_n(x)$ is any polynomial of degree n. We call $p_n^*(x)$ the *Chebyshev approximation* of $f(x)$.

(a) Let $\epsilon(x,p) = f(x) - p_n(x)$. Characterize $\epsilon(x,p^*)$, that is, what conditions must $\epsilon(x,p)$ satisfy when $p = p^*$?

(b) Let $g(x) = f(x) + q_r(x)$ where $q_r(x)$ is a polynomial of degree r and $r \le n$. Given that $p_n^*(x)$ is the Chebyshev approximation to $f(x)$, determine the Chebyshev approximation to $g(x)$.

(c) Consider the function

$$f(x) = \frac{1}{x-\lambda} \text{ for } x \in [-1,1], \text{ where } \lambda > 1.$$

The coefficients of $p_n^*(x)$ can be calculated explicitly so that

$$p_n^* = \sum_{j=0}^{n} d_j x^j.$$

Let $g(x) = x^{k+1}/(x-\lambda)$ where $k \le n$. Then using parts (a) and (b), determine the Chebyshev approximation of degree n to $g(x)$ for $x \in [-1,1]$.

Problem 6.

Let A be a real m x m matrix of rank m. Consider the matrix iteration formula

$X_{n+1} = X_n(2I - AX_n)$, X_0 arbitrary.

This method can be used to compute A^{-1} (for an appropriate choice of X_0).

(a) Show that if $AX_0 = X_0A$, then $(*)$ $AX_i = X_iA$ for all $i \ge 0$.

In parts (b) and (c), assume that $(*)$ holds.

(b) Let $E_i = A^{-1} - X_i$. Show that $E_{i+1} = AE_i^2$ for $i = 0, 1, \ldots,$ and thus $E_i = A^pE_0^q$. (YOU must determine p and q as functions of i.)

(c) Assume A is a real symmetric positive definite matrix with $0 < \alpha \le \lambda_i(A) \le \beta$. Furthermore, let $X_0 = cI$, c a scalar.

(1) For what range of values of c will the iteration converge?

(2) What choice of c will minimize the spectral norm of AE_0?

Problem 7.

Let $f(x)$ be a real-valued continuous function on the closed interval [a, b] and suppose $f(a)$ and $f(b)$ are of opposite signs.

(a) Assuming an algorithm is known for evaluating $f(x)$ show how the method of bisection can be used to locate a zero in the interval [a, b]. Is it necessary for $f(x)$ to be differentiable? Does the algorithm work if $f(x)$ has a number of roots in [a, b]?

(b) The following iterative algorithm (referred to as "successive interpolation") is proposed for finding a zero of $f(x)$:

$$x_{r+1} = (x_r f_{r-1} - x_{r-1} f_r)/(f_{r-1} - f_r) \text{ where } f_r = f(x_r).$$

Assuming $f(x)$ has a continuous third derivative and that x_r does indeed converge to a single zero $x = a$, discuss the asymptotic behavior of $x_r - a$.

Show by means of an example that even when f_1 and f_2 have opposite signs, the algorithm is not necessarily satisfactory for finding a root between x_1 and x_2.

(c) The following algorithm *root* represents a combination of the bisection algorithm and successive interpolation. Describe in general terms how it works; use diagrams if needed. What is the purpose of the conditional statement labeled *"iteration"?*

The procedure *root* returns a root of $fx = 0$ between a and b, where fx is a real function of x taking different signs at a and b. Iteration continues until a zero has been found with a tolerance $\le abs(x*e1) + e2$. Here $e1$ and e2 are prescribed relative and absolute errors. Rounding errors are not considered.

```
real procedure root(x, a, b, fx, e1, e2);
value a, b; real x, a, b, fx, e1, e2;
begin real c, fa, fb, fc, tol;
    x := a; fa := fx; x := b; fb := fx; go to initial;
    iteration: if abs(a-b) ≤ tol then a := b+sign(c-b)*tol;
    comment decide whether to take interpolated point or bisection point;
    if sign(a-x) = sign(b-a) then x := a;
    a := b; fa := fb; b := x; fb := fx;
    comment make sure f(c) and f(b) have opposite signs;
    if sign(fc) = sign(fb) then
    initial: begin c := a; fc := fa end;
    comment make sure that |f(b)| ≤ |f(c)|;
    if abs(fb) > abs(fc) then
    begin a := b; fa := fb; b := c; fb := fc; c := a; fc := fa end;
    x := (b+c)/2;
    a := if fb-fa ≠ 0 then (a*fb-b*fa)/(fb-fa) else x;
    tol := abs(b*e1)+e2;
    if abs(x-b) > tol then go to iteration;
    root := x
end root;
```

January 1968 Numerical Analysis Qualifying Exam

Problem 1.

It is desired to evaluate $f(x) = \log(1+x)$ for $|x| \le 1/2$ in t-digit (where t is to be determined) rounded floating-decimal arithmetic with an error less than 10^{-15}, using a truncated power series

$$u_n(x) = \sum_{k=1}^{n} (-1)^{k+1} \frac{x^k}{k}.$$

(a) What is the smallest value of n' for which you can prove that the truncation error $|f(x) - u_n(x)| < 10^{-17}$?

(b) For the value of n found in part (a) and for any floating-decimal number x with $|x| \le 1/2$, describe some reasonable method of evaluating F,(x) $= fl(u_n(x))$.

(c) Give an expression for the round-off error F,(x)- $u_n(x)$ in terms of t.

(d) Give a reasonable bound for the total error | F,(x) $-f(x)|$ in terms of t.

(e) What is the least value of t which will guarantee that $|F_n(x) - f(x)| < 10^{-15}$?

. Problem 2.

The XYZ Corporation is designing a new line of digital computers. They seek your advice on the needs of numerical analysts, that is, persons who need to get good numerical results from mathematical algorithms easily, together with provable error bounds.

State in considerable detail what the operational characteristics of the arithmetic unit should be, including the number representation, the mathematical nature of the arithmetic instructions, and the high-speed registers. Point out which considerations are vital and which are debatable.

Problem 3.

A newly forming library of practical mathematical programs for the 360/67 needs algorithms. Suggest one broadly useful algorithm for each of the following applications, either naming it or describing it, or giving its source well enough to identify it roughly. If you can't do better, say where to look for a good algorithm, Justify your answer in a sentence or two.

(1) Solve a linear equation system with a dense, stored matrix.

(2) Find all the eigenvalues of a dense, stored, symmetric matrix.

(3) Find one eigenvalue largest in modulus of a dense, stored matrix.

(4) Solve a 1inear algebraic system with a very large, sparse matrix.

(5) Find all the zeroes of a polynomial with real coefficients.

(6) Solve an ordinary differential equation $dy/dx = f(x, y)$.

(7) Solve a system of ordinary differential equations.

(8) Solve a boundary-value problem for Laplace's equation in two dimensions, with the function values prescribed on the boundary.

(9) Integrate a smooth function of one variable over a finite real interval (a, b).

(10) Find a crude estimate of such integrals as

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 f(x_1, x_2, x_3, x_4, x_5)\, dx_1 dx_2 dx_3 dx_4 dx_5$$

where $0 \le f(x_1, \ldots, x_5) \le 1$.

(11) Interpolate values of a smooth function of one variable given at equally-spaced values of the independent variable.

(12) Find a local maximum of a smooth function of n real variables.

(13) Generate pseudo-random numbers uniformly distributed on the interval $(0,1)$.

(14) Minimize the linear functional $b_0{}^T x$, subject to inequalities of the form

$$\begin{aligned} x_i &\ge 0, & i &= 1, 2, \ldots . n; \\ b_i{}^T x &\ge \alpha_i & i &= 1, 2, \ldots, m. \end{aligned}$$

Here $b_0, \ldots,$ b,,, , x are column vectors in El,, with $n >> m$, and the α_i are scalars.

(15) Find a real solution of a system of 20 nonlinear equations in 20 real unknowns, where a reasonable estimate of the solution is given.

Do three of Problems 4-9.

Problem 4.

Consider the ordinary differential equation problem

$$\frac{dy}{dx} = f(x, y), \quad y(0) = y_0. \tag{1}$$

In Milne's method for the approximate solution of (1), the corrector formula

$$y_{n+1} = y_{n-1} + \frac{h}{3}[f(x_{n+1}, y_{n+1}) + 4f(x_n, y_n) + f(x_{n-1}, y_{n-1}], \tag{2}$$

where $n = 1, 2, \ldots .$ is used in conjunction with a suitable predictor formula. Here $x_n = nh$ and y_n is an approximation to $y(x_n)$. The corrector formula (2) is applied repeatedly until no further change in y_{n+1} occurs and so the particular predictor used is irrelevant.

(a) Derive a formula for the truncation error which arises in a single step from x_n to x_{n+1} when using this formula.

(b) Suppose that this corrector formula with suitable predictor is used to find an approximation to the solution of the problem p' = tp, $y(0) = 1$. Derive an approximate formula for the truncation error which arises in N steps. Assume that $y_0 = y(0) = 1$ and that $y_1 = y(h)$ has been calculated exactly by some other method (e.g. Taylor's series).

(c) Suppose that we use as a predictor for (2) the formula

$$y^*_{n+1} = -4y_n + 5y_{n-1} + 2h\,[2f(x_n, y_n)\ \text{t}\ f(x_{n-1}, y_{n-1})] \tag{3}$$

and that we use (2) only *once* (no iteration as in parts (a) and (b)). Show that for the differential equation problem

$$y' = -y, \quad y(0) = 1$$

this scheme is stable. Is the predictor (3) stable?

Problem 5.

Let A be a real m x n matrix with $m \geq n$. Then it is known that

$$A = U \Sigma V^T,$$

where U, V are orthogonal square matrices and Σ is the diagonal matrix of singular values $\sigma_i(A)$. It is well known that

$$\| A \|_2 = \sigma_{\max}(A).$$

We define

$$\text{cond}(A) = \begin{cases} \|A\|_2 \|A^+\|_2, & \text{if } A \neq 0 \\ 1, & \text{if } A = 0 \end{cases}$$

where $A+$ is the pseudo-inverse of A.

(a) Show that if $b \neq 0$, $x = A^+b$, and p = $A^+(b+\delta)$, and rank(A) = n with $\|\delta\|_2/\|b\|_2 \leq \epsilon$, then

$$\frac{\| x - p \|_2}{\| x \|_2} \leq \epsilon \ \text{cond}(A).$$

(b) Let $B = AH$ where H is an $n \times k$ matrix and $H^T H = I_k$. Show that $\sigma_{\max}(B) \leq \sigma_{\max}(A)$.

(c) Let $\bar{A}$ be a $m \times k$ matrix made up of any k columns of A. Using the result of part (b), show that $\sigma_{\max}(\bar{A}) \leq \sigma_{\max}(A)$.

(d) Extending the results of parts (b) and (c), show that $\text{cond}(\bar{A}) \leq \text{cond}(A)$.

Problem 6.

(a) How would you recognize that you have obtained the nth degree polynomial that is the minimax (Chebyshev) approximation to a function $f(x)$ in $C[a, b]$? ($C[a, b]$ is the class of functions continuous on $[a, b]$.)

(b) Use your answer to part (a) to find the straight line that is the best Chebyshev approximtion to the function $ax^2 + bx + c$ in $[-1,1]$.

(c) Find the answer to the problem in part (b) by expanding the function $ax^2 + bx + c$ in a series of Chebyshev polynomials $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$. State the general theorem you are using.

(d) Prove that

$$f(x) = (x - x_1)(x - x_2) \cdots (x - x_n)$$

where the x_i are at your disposal, is minimized in the maximum norm in $[-1,1]$ by choosing $x_r = \cos[(2r-1)\pi/2n]$.

Problem 7.

Suppose that one wishes to find a solution of the system

$$f(x, y) = x^2 + y^2 - 4x = 0$$
$$g(x, y) = y^2 + 2x - 2 = 0$$

by an iterative method. It is known that there is a solution close to $x = 0.5$, $p = 1$.

(a) It is proposed that the following iteration be used:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - \begin{pmatrix} -1/5 & 1/5 \\ 1/5 & 3/10 \end{pmatrix} \begin{pmatrix} x_n^2 + y_n^2 - 4x_n \\ y_n^2 + 2x_n - 2 \end{pmatrix}$$

starting with $x_0 = 0.5$, $y_0 = 1$. Prove that this sequence converges linearly to the solution of the system.

(b) Write down explicitly the formulas needed to solve this system by Newton's method. What is the order of convergence of this method? Give reasons for your answer.

(c) What is the relation of the method proposed in part (a) to Newton's method?

Problem 8.

Let $f(x)$ be a given function and $\{x_i\}_{i=1}^{r}$ be a sequence of points for which $f(x_i)$ is known. We assume that $x_1 < x_2 < \ldots < x_r$.

Let $P_s(x)$ be an sth-degree polynomial, and let

$$\mathcal{P} = \{P_s(x) \mid P_s(x_i) \geq f(x_i), \quad i = 1, 2, \ldots, r\}.$$

We wish to determine $\hat{P}_s(x)$ a $\mathcal{P}$ such that

$$P_s(x_i) - f(x_i) \geq \hat{P}_s(x_i) - f(x_i) \text{ for all } P_s(x) \in \mathcal{P}.$$

(a) Show that the coefficients of $\hat{P}_s(x)$ solve a linear programming problem.

(b) Give the dual form of the problem developed in part (a).

(c) What special computational devices may be used for solving the dual problem to take advantage of the special form of the matrix?

Problem 9.

The well-known Horner scheme for evaluating polynomials

$$x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$$
$$= (\ldots((x + a_{n-1})x + a_{n-2})x + \ldots)x + a_0$$

evaluates a normalized polynomial (leading coefficient = 1) in $n - 1$ multiplications and n additions, and any polynomial in n multiplications and n additions. It is also known that this is not the best possible method (counting operations) if one is willing to do some preprocessing on the coefficients. In this case, it is possible to reduce significantly the number of multiplications.

(a) Illustrate a reduction in number of multiplications on a general normalized polynomial of order 4. Devise a computation scheme that would require only 2 multiplications.

(b) Try to devise a scheme for computing a general normalized polynomial of order 5 using only 3 multiplications.

(c) Estimate the lowest number of operations required for computing a polynomial of order n. Give a plausible argument for your estimate.

(d) What else do you know on this subject? Are you familiar with any effective algorithms for carrying out this reduction in the general case?

April 1969 Numerical Analysis Qualifying Exam

Problem 1.

For a linear programming application it is desired to compute $\pi\rho$ where $\pi = \gamma B^{-1}$. The components of the column vector $p = (\rho_1, \rho_2, \ldots, \rho_m)$, row vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_m)$, and m x m non-singular matrix $B = [b_{ij}]$ are known without error. How accurately must the components of $B^{-1} = [\beta_{ij}]$ be computed to guarantee that the maximum error of computing $\pi\rho$ is $\leq 10^{-5}$?

Problem 2.

Consider five-point Lagrange interpolation of a function $f(x)$ based on equally spaced abscissas with spacing h. Show that if $h^5 |f^{(5)}(x)|$ does not exceed 32 units in the last place to be retained, then the truncation error cannot exceed one unit in that place, and also that $h^5 |f^{(5)}(x)|$ may be as large as 84 units if the interpolation is effected only between the second and fourth of the five successive abscissas.

Problem 3.

(a) How would you recognize that you have obtained the nth degree polynomial that is the minimax (Chebyshev) approximation to a function $f(x)$ in $C[a,b]$? ($C[a,b]$ is the class of functions continuous on [a, b].)

(b) Prove that if $f(x)$ is an even function of x, then the minimax approximation on any interval $[-a,a]$ is an even function of x.

(c) Determine the values of a, 6, c and d in the polynomial $P(x) = ax^3$ t bx^2 + cx + d which minimize

$$\max_{-1 \leq x \leq 1} |P(x) - |x||.$$

Problem 4.

The matrix

$$A = \begin{bmatrix} 1 & .01 & -.01 & .01 \\ .01 & 2 & .04 & .3 \\ -.02 & .01 & 3 & -.1 \\ .3 & -.2 & .1 & 4 \end{bmatrix}$$

has eigenvalues $\lambda_1 \approx 1, \lambda_2 \approx 2, \lambda_3 \approx 3$, and $\lambda_4 \approx 4$.

(a) Show that $0.97 \leq \lambda_1 \leq 1.03$.

(b) It is *very* easy to give much tighter bounds for λ_1. Give the best bound you reasonably can.

(c) Give a reasonably low upper bound for the spectral norm

$$\max_{\|x\| = 1} \| Ax \|$$

where $\|\cdot\|$ is the Euclidean length.

Problem 5.

Consider the system of differential equations

$$y' = z$$
$$z' = -69 - az$$

with $y(x_0) = y_0$ and $z(x_0) = z_0$, and where a and 6 are real.

(a) Give the analytic solution of this system of equations in exponential form.

(b) Assume $0 < a < 2\sqrt{b}$, $6 > 0$. Show that the solution of the system remains bounded for all y_0 and z_0.

(c) Give Euler's method for solving the system of equations.

(d) What is the largest step length h for which all solutions of the corresponding difference equation are bounded?

Problem 6.

A square matrix A, a column vector c, and a row vector r are all given. Let $B = A + cr$, and assume that B^{-1} exists.

(a) Prove that

$$(*) \quad B^{-1} = A\text{-'} \ t \ \gamma p,$$

where γ is a column vector and p is a row vector.

(b) Give expressions for γ and p.

(c) Suppose that a lower triangular matrix L and an upper triangular matrix U are also given such that $LU = A$, and that a column vector d is also given. Make use of $(*)$ to give an efficient algorithm to solve $Bx = d$.

Open Book Computing Problem.

A continuous curve p $= f(x)$ is defined for $x \geq 0$ by the differential equation

$$\frac{dy}{dx} = x + y^3$$

and the initial condition $x = 0$, p $= 1/2$.

(a) Show that the curve p $= f(x)$ has a vertical asymptote at $x = a$, for some finite $a > 0$. *(Hint:* Look at $dy/dx = y^3$.)

(b) Find $f(1)$ as accurately as you can.

(c) Find $f(1.107)$ as accurately as you can.

(d) Find 6 such that f(b) = 28 as accurately as you can.

(e) Find the abscissa *a* of the vertical aymptote as accurately as you can.

April 1970 Numerical Analysis Qualifying Exam

Open Book Computing Problem.

A chemist is studying some reactions and he knows that the concentrations of two components in his experiment obey the ordinary differential equations

$$\frac{dy}{dt} = -k_1 y \text{ t } k_2(b - 29 - z)z$$

$$\frac{dz}{dt} = -k_3 z + k_4(b - 29 - z)(a - p - z) - \frac{dy}{dt}$$

where the k_i are unknown positive constants, and a and 6 are known positive constants.

The following experiment was made:

1. At $t = 0$, the values of a, 6, p(O), and $z(0)$ were set to

   a = 1.0, 6 = 2.0, $y(0) = .25$, $z(0) = .50$.

2. The values of $p(t)$ and $z(t)$ were sampled at various times. The following data was gathered.

t	$y(t)$	$z(t)$
0	.250	.500
.333	,301	.403
.672	.324	.362
1.012	.335	.345
∞	.345	.332

By $t = \infty$ the chemist means a sufficiently long time so that the system has become stable (i.e. $dy/dt = dz/dt = 0$). In this case it was certainly stable by $t = 100$.

From physical considerations the chemist knows that the k_i should be close to one. What values of k_i can you calculate for him?

Your answer will be graded on the accuracy of the results, the reasonableness of the methods and the amount of computing used,

April 1971 Numerical Analysis Qualifying Exam

Computer Problem.

You have about six days to work on the following computer problem. You will be assigned an account number and expected to use only that number when working on the problem. The total charges accumulated will be taken into account in the grading.

You may use any computer language you wish. You are encouraged to use any appropriate subprograms available to you through various libraries, friends and relatives, or past projects of your own. Please identify the source of any such programs.

It is hoped that you will learn something while working on this problem.

Let

$N = 10,$

$M = 20,$

$r(\theta) = 1/\max(\cos\theta,\ \sin\theta),\ 0 \le \theta \le \pi/2,$

$\alpha_n = \begin{cases} 2n - 2/3, & n \text{ even}, \\ 2n - 4/3, & n \text{ [illegible]} \end{cases} \quad n = 1, \ldots, N,$

$\theta_m = \frac{m}{N}\frac{\pi}{2},\ m = 1, \ldots, M,$

$r_m = r(\theta_m),\ m = 1, \ldots, M,$

$J(\alpha, x) =$ the a-th order Bessel function of x, scaled so that

$J(\alpha, x) \approx \left(\frac{x}{2}\right)^{\alpha}$ for small x,

$a_{mn}(\lambda) = J(\alpha_n,\ \sqrt{\lambda}\, r_m)$ sin (a,&), $m = 1, \ldots, M,\ n = 1, \ldots,$ N,

$A(\lambda) =$ the M x N matrix with elements $a_{mn}(\lambda)$,

$\|\cdot\| =$ the Euclidean vector norm.

Your problem is to find a value of λ between 9 and 10 so that the columns of *A(h)* are nearly linearly dependent and find the coefficients in that dependence. Specifically, find a scalar λ and an N-vector c which give

$$\min_{9 \le \lambda \le 10}\ \min_{\|c\| = 1} \|A(\lambda)c\|$$

Note: This problem is derived from two papers: Fox, Henrici and Moler, SIAM *J. Numer. Anal. 4*, 1967, pp. 89-102; and Moler, Stanford Report No. CS 121, 1969. You may want to refer to these papers for background and hints, although it is not necessary to understand them in detail.

April 1972 Numerical Analysis Qualifying Exam

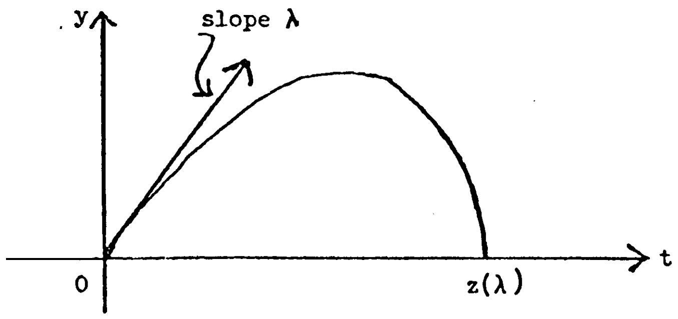
<u>**Computer** Problem.</u>

For **any** $\lambda > 0$, **let** $z(\lambda)$ **be the least** positive zero of the function $y(t)$ **that solves the** ordinary differential equation problem

$$y''(t) + I_0 + t/10 = 0$$
$$y(0) = 0, \quad y'(0) = \lambda$$

where $I_0(t)$ is **the** zeroth order modified Bessel function (see Abramowitz and Stegun, *Handbook of Mathematical Functions,* National Bureau of Standards, AMS 55, pp. 374-375).

The graph of the solution $y(t)$ is approximately as shown below.



(a) Find λ_{max}, the unique value of λ such that $z(\lambda)$ is a maximum.

(b) Find $z(\lambda_{max})$.

(c) Give **a** table of **values** of $y(t)$ for $t = 0(0.1)z(\lambda_{max})$, for the $y(t)$ which maximizes $z(\lambda)$, that is, for $y(t)$ satisfying $y'(0) = \lambda_{max}$.

(d) Give a discussion of **the** accuracy of your results.

July 1973 Numerical Analysis Qualifying Exam

Computer Problem.

Consider the Fredholm integral equation of the first kind,

$$\int_0^1 K_\gamma(t, s)\, x(s)\, ds = y(t), \quad 0 \le t \le 1, \tag{1}$$

where

$$K_\gamma(t, s) = \frac{1-\gamma^2}{1+\gamma^2 - 2\gamma \cos(2\pi(t+s))},$$

$$y(t) = \cos(2\pi t).$$

The object is to find a numerical approximation to $x(s)$, i.e., values for $x(s_j)$ for points $s_j \in [0, 1]$. We decide to use the following method.

Collocation: Let $t_i = (i - 1/2)/n$, $i = 1, 2, \ldots, n$, and replace (1) by the n equations

$$\int_0^1 K_\gamma(t_i, s)\, x(s)\, ds = y(t_i), \quad i = 1, 2, \ldots.\, n. \tag{2}$$

Quadrature: Replace (2) by

$$\sum_{I"}^{n} K_\gamma(t_i, s_j)\, x(s_j)\, w_j + \epsilon(t_i) = y(t_i), \quad i = 1, 2, \ldots, n, \tag{3}$$

where ϵ_i is the quadrature error, and w_j are quadrature weights at points $s_j, j = 1, 2, \ldots, n$.

This produces a matrix system

$$Ax + \epsilon = y, \tag{4}$$

where the vectors x, y, and ϵ are defined by $x_j = x(s_j)$, $y_j = y(t_j)$, and $\epsilon_j = \epsilon(t_j)$, and the matrix A is defined by $a_{ij} = K_\gamma(t_i, s_j)w_j$.

(a) Solve for x from (4) assuming $\epsilon = 0$ for the following twelve combinations of parameters:

$\gamma = (0.75, 0.25)$
$n = \{10, 20, 40)$

$$\text{quadrature} = \begin{cases} \text{forward rectangular } (s_j = (j-1)/n) \\ \text{midpoint } (s_j = t_j) \end{cases}$$

Given that the true solution is $x_T(s) = (1/\gamma)\cos(2\pi s)$, construct a table (with twelve entries) showing the actual error $\left\{\sum_{j=1}^{n} (x_j - x_T(s_j))^2\right\}^{1/2}$. Also construct a table showing the residual $\|Ax - y\|_2$ for your calculated solutions. Comment on your results.

(b) Suppose we are given an extra piece of information, namely that

$$\| x_T(s) \|_2 = \frac{1}{\gamma\sqrt{2}}$$

and that this is approximated by

$$x^T x = \| x \|_2 = \frac{\sqrt{n}}{\gamma\sqrt{2}}.$$

Construct an algorithm to solve the mathematical programming problem

$$\text{minimize } \| Ax - y \|_2 \text{ } \textit{over} \text{ all } x \text{ such that } \| x \|_2 = \frac{\sqrt{n}}{\gamma\sqrt{2}}. \tag{5}$$

Hint: Use Lagrange multipliers.

(c) Use your algorithm and available matrix subroutines to calculate solutions for the twelve parameter values of part (a). Construct tables for the error and the residual as in (a). Comment on these solutions.

(d) Find error estimates for your solutions to (5), that is, find an expression B(c) which is a function of the quadrature error, such that

$$\| x - x_T \|_2 \le B(\epsilon) ,$$

where x is the solution to (5) and x_T has elements with values being the true solution evaluated at $s_j, j = 1, 2, \ldots, n$.

May 1968 Computer Design Qualifying Exam

Problem 1.

You are to describe a system for incorporating a push-down stack store into a computer system by utilizing a portion of the main core memory, three registers, and some logic and flip flops as necessary. Specify the details of a "push" operation (inseting a new word in the top of the stack).

Problem 2.

Design a network having as inputs w, x, y, z and as outputsf and g, where

$$f(w, x, y, z) = \Sigma\,(3, 7, 11, 12, 13, 14, 15)$$
$$g(w, x, y, z) = \Sigma\,(1, 2, 3, 5, 6, 7, 9, 10, 11).$$

Use NAND gates only. Use as few gates as you can (6 are sufficient) assuming double-rail inputs, i.e. the complements of the inputs are available.

Problem 3.

Consider a computer system in which signed (algebraic) numbers are represented in memory as signed two's complements. Since obtaining the magnitude of a number in such a system is non-trivial, it is desirable to be able to carry out multiplication with numbers still in signed two's complement form. Describe a scheme for doing this.

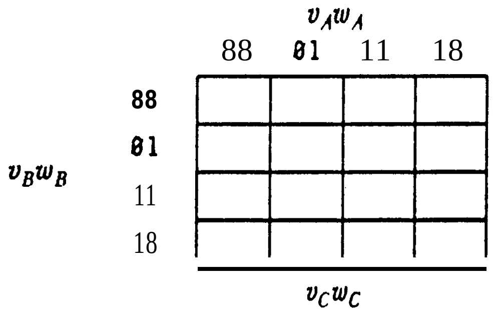
Problem 4.

Most digital systems have their sequencing of operations controlled by a central timing source or "clock". It is also possible to design systems in which the completion of an operation is explicitly detected and this information is then used to initiate the next operation. One scheme for doing this involves using two leads to represent each single variable and encoding the variable as follows.

Original variable	Encoded version	
A	v_A	w_A
0	0	1
1	1	0
variable not available yet	0	0
unused combination	1	1

Initially all signals are set to 0 and then the inputs are set to the desired encoded values; completion is detected when the outputs each change from 00 to either 10 or 01.

(a) Let A and B be inputs and let $C = A \cdot B$. You are to fill in (with O's, l's, and d's) the following map for a circuit to have encoded versions of A and B as inputs and have an encoded version of C as its outputs.



(b) Write expressions for v_C and w_C.

(c) Draw a circuit for v_C and w_C using AND gates and OR gates.

(d) Draw a circuit with the encoded version of A as input and the encoded version of A' as output.

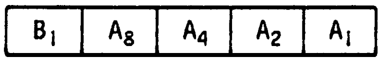
Problem 5.

An efficient technique for converting a binary integer to a BCD (8421) integer can be based on the fact that binary "1" equals BCD "1" and shifting a binary number left one position is equivalent to multiplying it by 2. The binary integer is shifted left bit by bit into a BCD register and the contents of the BCD register are doubled after each shift.

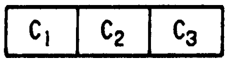
Example:

Binary number	,1010
Shift left (binary side)	1,010
Double (BCD side)	18,010
Shift left (binary side)	10,10
Double (BCD side)	100,10
Shift left (binary side)	101,0
Double (BCD side)	1,0000,0
Shift left (binary side)	1,0000,
	BCD , binary

Design the clocked sequential circuitry necessary to perform the binary to BCD conversion as indicated above.



BCD Register



Binary Register

Fill in the following table and complete the design to obtain flip flop excitation equations for JK flip flops. (That is, determine the necessary J and K inputs for the flip flops of the BCD register.)

	Time n	Time n+1	
Decimal	$A_8A_4A_2A_1$	$B_1, A_8A_4A_2A_1$	Decimal
0			0
1			2
2			4
3			6
4			8
5			10
6			12
7			14
8			16
9			18

Problem 6.

(a) Discuss briefly with the aid of sketches the characteristics (advantages and disadvantages) of different magnetic core memory organizations (3D, 2D, 2.5D).

(b) Describe the output signal which appears on a sense line when a "zero" is read and when a "one" is read from a core.

(c) How many drive and sense amplifiers are required for a memory of 4096 words of 16 bits per word for: (1) 3D organization and (2) 2D organization.

(d) Design a minimal complete decoding network to address 64 lines (using 2 or 3 input gates).

Problem 7.

One novel approach to the construction of error-correcting codes is through the use of geometric notions. The simplest of these is that described here: row and column parity checks in a rectangular array.

Suppose we define a particular binary group of $n =$ 16 bits (9 information plus 7 parity check bits) by insisting that any code word fits into the 4 x4 square shown below, where every row contains an even number of ones, and every column contains an even number of ones.

a	b	c	t
d	e	f	u
g	h	i	v
w	x	y	z

(a) Express the check digits $t, u,$ v, w, x, y, z in terms of the information digits a, b, c, d, e, f, g, h, i.

(b) What is the minimum Hamming distance between any two distinct code words? If an error pattern is undetectable with this code what is the smallest number of digits that could be in error? Give an example of such a pattern.

(c) Explain how to use the code for correction of single errors.

(d). Give an example of an error pattern containing four errors that is detectable. Are any error patterns with more than four errors detectable?

Problem 8.

This problem is about *unit-distance codes.* It is desired to encode the eight (analog) quantities: 0, 1, 2, 3, 4, 5, 6, 7 (mod 8) into binary code words of four bits each such that the following two properties are satisfied:

P1: Analog quantities that differ by ±1 (mod 8) are to be encoded into binary words differing in exactly 1 bit.

P2: If an error of any one bit occurs in the transmission of a four-bit code word, then the resulting error in the analog quantity must not exceed ±1 in magnitude, or must result in detection of the fact that an error has occurred.

(a) Does either of P1 or P2 imply the other? State which, if they are not independent. Are they equivalent?

(b) Find an encoding that satisfies these conditions. This is called a unit-distance error-checking code.

Problem 9.

It is desired to encode an alphabet of six symbols A, B, C, D, E, and F, for transmission over a noiseless binary channel. The code strings to be assigned to these symbols may be of different lengths. We want the average length of the string sent over the channel to be minimized, where the source symbols (A through F) are used with the following probabilities:

Symbol	Probability	Coding string
A	0.500	?
B	0.250	
C	0.125	
D	0.100	
E	0.015	
F	0.010	

(a) What is the lower bound to the average length, $\bar{L} = \sum_{i=A}^{F} L_i P(i)$, where L_i is the length of string used to encode the i-th source symbol?

(b) Find a variable-length encoding that gets as close to this lower bound as possible. What is the efficiency of this encoding?

(c) If we insist that this encoding be *uniquely decipher&e* (that is, it is assumed that the channel carries a continuous stream of binary digits without spaces or other demarcation between the strings representing separate consecutive symbols), what is the average length $\bar{L}$ attainable?

May 1969 Computer Design Qualifying Exam

Problem 1.

Design a circuit realizing

$$f(w,x,y,z) = \Sigma\,(1,2,4,7,8,11,13,14)$$

using threshold gates with positive and negative weights allowed, Use as few gates as you can.

Problem 2.

In a certain 4-digit mixed-radix number system, the radices (by position) of a number $d_4d_3d_2d_1$ are 5, 4, 3, and 2, respectively. A convention must be defined for representation of negative numbers in this system, without using an extra sign bit.

(a) Define a complementation algorithm for this number system based on $(B-1)$'s complement representation. What are the most positive and most negative numbers representable? What are their representations?

(b) For the representation scheme of part (a), how can a test for positive sign be implemented?

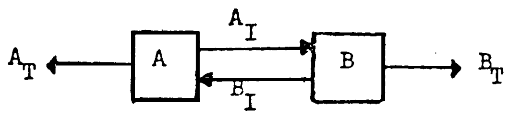
(c) Define a complementation algorithm based on B's complement representation. What are the most positive and most negative numbers representable7 What are their representations?

problem 3.

In addition to its normal capabilities, a sequential machine A has an input A_I which may be connected to test a copy of itself, machine B. Machine A also has a special output A_T which is "1" if the tested machine B is functioning properly and "0" if B has failed. If, however, machine A has failed, the output A_T is invalid and may be "1" or "0" regardless of the condition of B.

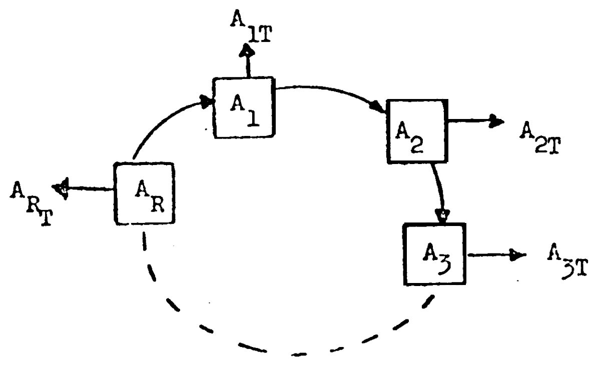
A failure or a set of failures is "detected" if one can determine with certainty that not all machines are functioning properly. A failure is "diagnosed" if one can determine that a *particular* machine has failed.

(a) Assume two machines A and B are testing each other, as shown below.



Can single failures (failures of machine A or B but not both) be detected by observing A_T and B_T? Can single failures be diagnosed? Can double failures be detected or diagnosed? Justify your answers.

(b) Assume we have a ring of *R* machines, each testing the next machine in the ring, as shown below.



Is there a value of *R* which will allow diagnosis of single failures (assuming multiple failures cannot occur)? If so, what is the minimum such value? Explain.

Problem 4.

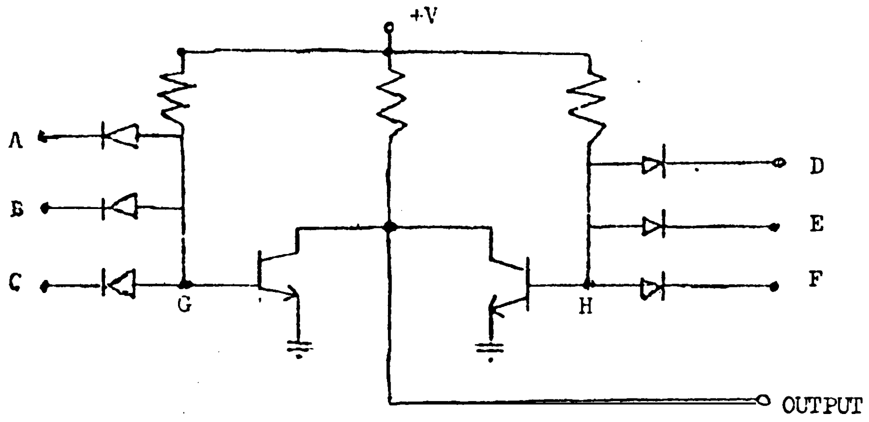
A mythical computer manufacturer produces systems with no "I/O wait" cycles due to their superb I/O system architecture. Careful statistical sampling of programs executed on this machine show that, on the average, the following distribution of memory cycles occurs for each executed instruction:

Memory cycles	Purpose
I	Instruction fetch
1/4	Indirect addressing
1/2	Operand fetch or operand store
2	Input-output
3 3/4	Total average cycles per executed instruction

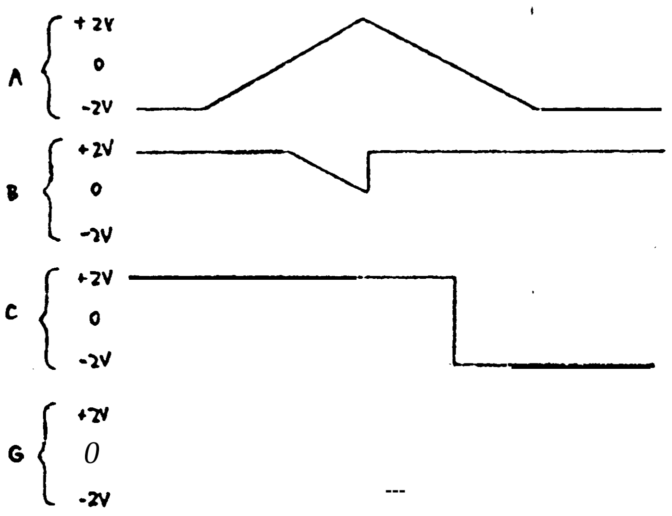
The memory architecture for this system employs a single bank of 64K words, each 16 bits, with a 1 μsec cycle time. The run time for a typical program would thus be estimated at 3.75 μsec per executed instruction. Cynics have noted that this memory is the bottleneck of the system, noting that a speed increase in memory will be matched by a proportional increase in the throughput of the system. A faster memory is, however, prohibitively expensive. Describe a scheme for substantially increasing the throughput of the system by changing the memory architecture, but not the memory speed. Estimate the average time per executed instruction in your scheme.

Problem 5.

The network shown below has 6 inputs A, B, C, D, E, and F. Two Internal points G and H are labelled for reference. Connections to a +V voltage source, ground, and the OUTPUT point are labelled.



(a) Draw the voltage at 6 as a function of the input voltages at A, B, and C shown below.



(b) Assume now that inputs A through F always carry either +V or GND voltage and assume a positive logic convention. What logic function is performed at G? At H? At the OUTPUT?

Problem 6.

(a) Give the mathematical definition of a regular expression.

(b) What is the relation of regular expressions to sequential machines?

(c) Show that the complement of a regular expression is a regular expression.

Problem 7.

An n-variable switching function is said to be *symmetric* if and only if the value of the function depends only on the number of arguments that have the value 1. 'We use the notation $S^{(n)}_{i,j,\ldots}$ to denote the n-variable function which takes the value 1 if *i* of its arguments are 1 or if *j* of its arguments are 1, etc.

(a) Fill in the truth tables for $S^{(3)}_{1}$, $S^{(3)}_{1,2}$, and $S^{(3)}_{1,3}$.

x_1	x_2	x_3	$S^{(3)}_{1}$	$S^{(3)}_{1,2}$	$S^{(3)}_{1,3}$
0	0	0			
0	0	I			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	I			

(b) Prove that, for any n, the n-variable symmetric switching functions form *a* Boolean algebra with the operations . and + (AND and OR).

(c) What are the atoms of this Boolean algebra?

Problem 8.

The IBM 1 130 uses the following code for disk recording. Each block of data consists of 20 bits, 16 information bits and 4 check bits. The encoder computes the check bits by using a rule that guarantees that the total number of l's in a block will be a multiple of 4. The decoder counts the number of I's per block and signals that an error has occurred if the count is not *a* multiple of 4.

(a) What is the minimum Hamming distance of this code? Prove your answer.

(b) What is the fewest number of check digits for 16 information digits that will yield a code of the same minimum distance? Prove.

(c) What is the fewest number of check digits for 16 information digits that will yield a code with minimum distance 3?

(d) Show how to construct the code of part (c) and describe a decoding algorithm that can be used for error correction.

Problem 9.

(a) Find a Huffman coding for the alphabet of six symbols given below with their respective probabilities of transmission.

Symbol	Probability
s_1	.20
s_2	.195
s_3	.175
s_4	.15
s_5	.14
s_6	.14

(b) Use your code to prove or disprove the following:

Given a Huffman code H, let H^* be the code formed by associating with each symbol the reversal of its code word in H. Hence if s_1 is coded by 1101 in H it will be coded by 1011 in H^*. The code words of H^* may not be uniquely decodable symbol by symbol, but every coded string of symbols can be decoded by a decoder with arbitrarily large memory if the end of the string is known to the decoder.

Problem 10.

An AN is a code that is used to check addition operations in computers. in this code the integer i is represented by the integer $A \cdot i$ where A is a fixed constant.

Assume that we are to use an AN code in a computer which performs addition modulo M.

(a) Prove that the set of coded integers forms a commutative group under addition if and only if M is a multiple of A.

(b) Let A4 $= A \cdot r$. Show that the addition of coded integers modulo M is equivalent to the addition of uncoded integers modulo r.

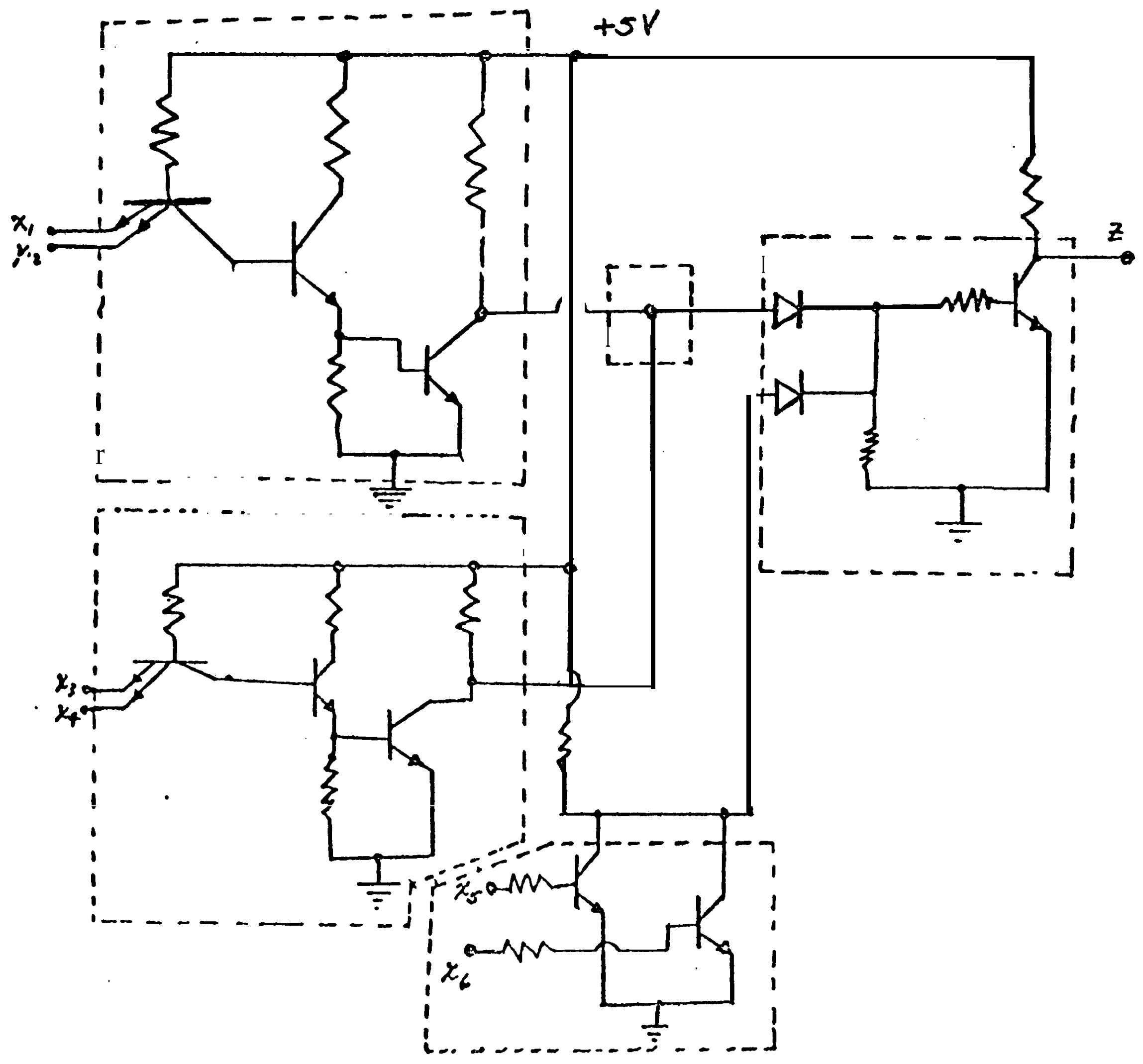
(c) The *arithmetic weight* of an integer is defined to be the number of non-zero coefficients in its binary representation. For example, $weight(16) = 1$ and $weight(7) = 3$.

Prove that an AN code has minimum arithmetic distance 3 or greater if and only if the residues of $\pm 2^j$ modulo A are distinct and non-zero for all j such that $2^j < M$, where M is the modulus of arithmetic.

May 1970 Computer Design Qualifying Exam

Problem 1.

(a) Draw a logic gate diagram for the circuit whose schematic diagram is shown below. Assume +5 V is logical 1 and 8 V is logical 8.



(b) Give a minimal product of sums Boolean expression for the output signal z as a function of the input signals $x_1, x_2, \ldots, x_6$.

Problem 2.

(a) Contrast the Quine-McCluskey method with the iterative consensus method for finding all of the prime implicants of switching functions. In particular, contrast the initial inputs to the algorithms and the strategies used in the computations.

(b) Give an estimate of the number of computational steps required by each algorithm in its worst case. The growth of this number with respect to some parameter is desired; constant coefficients are unimportant. You may wish to use the following parameters.

(1) $w =$ the number of true minterms (weight of the function)
(2) $p =$ the number of prime implicants
(3) $t =$ the number of terms in an initially specified algebraic expression for the function
(4) $n =$ the number of variables

(c) Which of the two methods is preferable from a computational point of view? State your assumptions.

Problem 3.

A non-zero element m of a Boolean algebra is called ***minimal*** iff for every element x of the algebra, if $x + m = m$ then $x = m$ or $x = 0$.

(a) Show that m is minimal iff $x.m = m$ or $xm = 0$ for every x in the algebra.

(b) The following statement is a theorem:

All finite Boolean algebras are isomorphic iff they have the same number of elements.

It is also well-known that the set of switching functions of n variables forms a Boolean algebra and the subsets of a set with n elements form a Boolean algebra. Are these algebras isomorphic?

(c) Consider the Boolean algebra of P-variable switching functions.

(1) How many elements does this algebra have?

(2) What are the minimal elements?

(3) What is the 0 of this Boolean algebra?
What is the 1 of this Boolean algebra?

Problem 4.

(a) Let t be the maximum number of errors correctable by a binary block code, and let d be the minimum Hamming distance between two code words, Derive the relation between t and d.

(b) A linear block code is defined as follows. Let G be a k x n matrix, $n \geq k$. If u is a k-tuple information vector, then it is encoded by the n-tuple v where

$$v = u[G]$$

Show that the minimum distance of a linear code is equal to the minimum Hamming weight of a code word.

(c) A *two-dimensional block code* is a linear code such that each code word is a &dimensional matrix as shown below.

Information	row checks
Column checks	checks on checks

Each row contains code vectors from a linear code and each column contains code vectors from a linear code, not necessarily the same as the row code.

Suppose the lengths, number of information bits, and minimum distances of the row and column codes are n_1, k_1, d_1 and n_2, k_2, d_2, respectively. Find a formula for the minimum distance of the two-dimensional code in terms of these parameters, and indicate the correctness of your formula.

Problem 5.

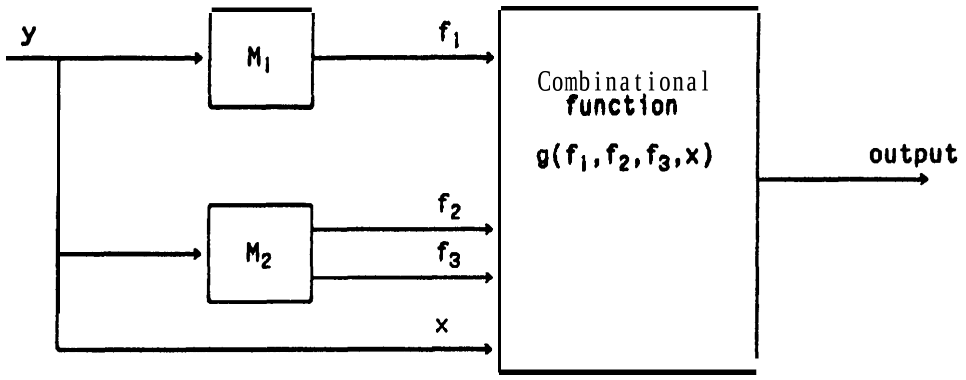
Let M be the machine given below. M is assumed to operate in clock pulse mode so that a single transition occurs in the presence of a clock.

	y = 0	y = 1
A	D,0	E,0
B	A,0	F,1
C	B,1	H,1
D	C,1	G,0
E	G,0	A,1
F	E,1	B,0
G	H,1	D,1
H	F,1	C,0

next state, output

Machine M

We wish to construct M from the parallel connection of two machines M_1 and M_2 with 2 and 4 states, respectively. The structure of the parallel connection is shown below.



The state table for M_1 is

	Y = 0	Y = 1
U	U,0	V,1
V	V,1	U,0

next state, f_1

Machine M_1

Fill in the tables below.

	y = 0	y = 1
W	, ,	, ,
X	, ,	, ,
Y	, ,	, ,
Z	, ,	, ,

next state, f_2, f_3

Machine M_2

y	f_1	f_2	f_3	output
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Problem 6.

High speed addition may be achieved by dividing an n-bit adder into *n/g* groups of g bits each. One such scheme employs groups whose sum outputs are valid at time $3g$, but whose carry output is valid 3 time units after the group inputs are valid. This is the "carry by-pass" approach.

(a) Assume all group inputs, except carry inputs from previous groups, are valid at time 0. When is the output of such an n-bit carry by-pass adder valid (assume *n* is a multiple of g)?

(b) Derive the value of g which produces the fastest addition for a 36-bit adder.

(c) How long does this fastest addition take?

Another scheme employs groups whose sum and carry outputs are both valid at time 3 after the group inputs are valid. This scheme is certainly more expensive than the carry by-pass scheme if the same size groups are used.

(d) If the groups are *h* bits each, how fast is an n-bit addition (assume *n* is a multiple of *h*)?

(e) What size groups must be used for a 36-bit adder as fast as in part (c)?

(f) Give a rough cost comparison of the costs of the adders of parts (c) and (e).

Problem 7.

A computation is to be performed on *n* independent sets of data. The computation is realized by the six instruction sequence: $I_1, I_2, I_3, I_4, I_1, I_2$. The four distinct instructions share no hardware in the machine. Operands generated by each instruction are used by the next. Each instruction takes 1 time unit.

(a) If parallel computation is possible but no hardware duplication is allowed, how quickly can the computation be performed? Sketch your strategy for achieving this result.

(b) Answer part (a) allowing two copies of each of the four instruction calculators.

October 1979 Computer Design Qualifying Exam

Problem 1.

A certain computer uses instruction opcodes that are encoded in fields of several different widths. Let $c_1, c_2, \ldots, c_n$ be the opcodes and assume that it is known that each opcode is used with probability p_i, $1 \le i \le n$. Let L_i be the number of bits in the code for c_i.

We wish to find a binary encoding of opcodes that satisfies the following criteria.

(1) The encoding of any opcode cannot be the prefix of the encoding of any other opcode.

(2) The encoding must minimize $\bar{L} = \Sigma_{i=1}^{n} p_i L_i$. (The minimization is over all encodings that satisfy (1).)

Prove that an encoding that satisfies properties (1) and (2) must also satisify the following properties.

(a) If $p_i > p_j$ then $L_i \le L_j$.

(b) Let $L_{\max}$ be the length of the longest opcode. Then there are at least two opcodes of length $L_{\max}$ that differ only in their rightmost bit.

Problem 2.

The IBM System/360 tape drives use an encoding scheme similar to the one shown in the diagram below. Each row in the record is an eight bit character with a parity check on all eight bis. Each column contains a column parity check at the end of the record. Information bits in the diagram are represented as b's and parity bits as p's.

```
b b b b b b b b p        first character
b b b b b b b b p        second character
        ...
b b b b b b b b p        last character
P P P P P P P P P        column parity checks
```

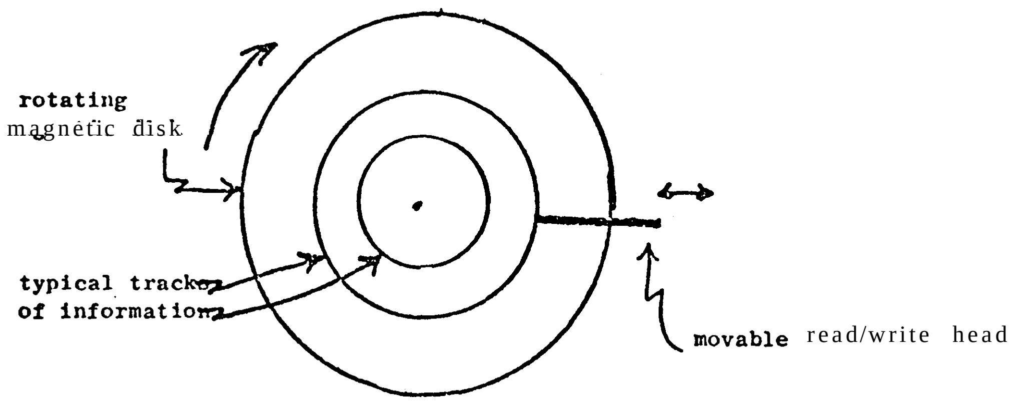
Recall that a *burst* error of *length L* is an error pattern *L* bits long that begins and ends with a 1. The decoding scheme for the code above is designed to correct burst errors that are contained entirely within one column.

(a) Consider an error which is contained entirely within column 1, starts at the first character, and is a burst of length *L* in that column. How many burst patterns are there of this type? How many of these patterns are correctable with the parity checks shown above?

(b) Assume that there exists a mechanism by which we can identify the column containing a burst error. Give a burst error decoding procedure that uses this mechanism. What is the longest burst that is correctable in this case?

Problem 3.

Consider a rotating magnetic disk storage device with a large number of concentric tracks of information and a read/write head which **must** be moved into position over a track in order to access the information which the track contains.



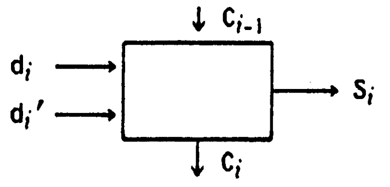
One of the major considerations in evaluating the performance of such a device is the expected distance which the head **must move between** successive track accesses.

(a) Derive an expression for this expected distance under the assumption that track accesses are independent and uniformly distributed. Let *L* be **the** distance the head **must move** to go from **the** outermost track to the innermost track.

(b) Give upper and lower bounds on the average head movement for the case in which head accesses are not distributed independently, and indicate track access distributions for which these bounds are achieved.

Problem 4.

Consider signed-digit number representations such that $d_n, d_{n-1} \ldots d_2 d_1 d_0$ represents the value $\Sigma_{i=0}^{n} d_i b^i$, where the base is $b \geq 3$ and the signed digits d_i satisfy $-b < d_i < +b$. Note that non-zero values do not in general have a unique representation in this system. The fth stage of an adder for signed-digit numbers can be viewed as having three inputs and two outputs:



where d_i and d_i' are the ith signed digits of the two numbers to be added, C_{i-1} is the signed carry from the previous stage of the adder, S_i is the ith signed digit of the sum of the two numbers, and C_i is the signed carry out of the i-th stage. Also $-1 \leq C_i \leq +1$ for $i \geq 0$, and $C_{-1} = 0$. It is possible to design such an adder so that the carry out C_i is independent of the value of the carry in C_{i-1}, thus eliminating much of the carry propagation time of the adder.

(a) For $b = 3$ show sum and carry functions for a stage of a signed-digit adder. Make C_i independent of C_{i-1}.

$d_i + d_i'$	C_i	S_i when $C_{i-1} = -1$	S_i when $C_{i-1} = 0$	S_i when $C_{i-1} = +1$
-4				
-3				
-2				
-1				
0				
+1				
+2				
+3				
+4				

(b) Give a general formula for the sum and carry functions of a signed-digit adder for arbitrary base b.

Problem 5.

The partial ordering relation “divides-evenly” defines a Boolean algebra on the set {1, 2, 3, 5, 6, 10, 15, 30}.

(a) What are the operations "+" and "•" in this algebra?

(b) Identify the complement of each element in the algebra.

(c) Consider the logical operator ⊙ defined by the Boolean function below. Prove or disprove the following: A set of logic gates realizing this function forms a complete set.

u v	u⊙v
8 8	1
81	1
18	8
t 11	1

Problem 6.

(a) What is a master-slave flip-flop?

(b) Why is such a flip-flop useful?

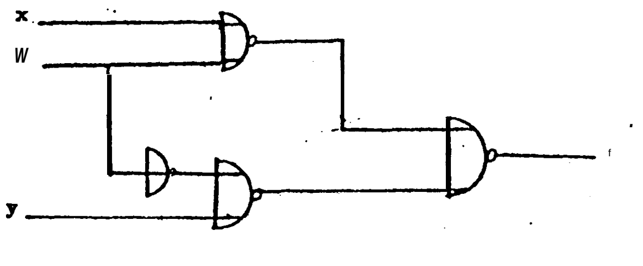
Problem 7.

(a) Define the following terms:

(1) static 0 hazard
(2) static 1 hazard

(b) State necessary and sufficient conditions for the existence of a static 1 hazard.

(c) Find the 0 sets and 1 sets of the following network.



(d) Specify all static hazards in this network.

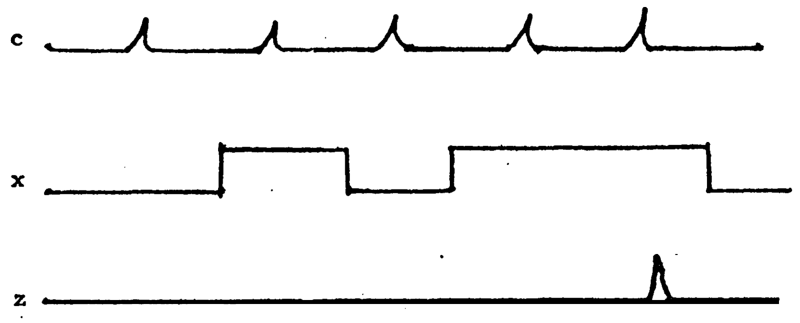
(e) Draw a NAND network for the function realized by the network in part (c) that does not contain any static hazards.

Problem 8.

(a) Give a state transition diagram, a flow table, and an excitation table for a clocked sequential circuit with clock input *c*, a level input *x*, and pulse output *z*. The circuit is operated in pulse mode and must produce a 1 output if and only if the input sequence recognized for the *x* input is contained in the following regular set:

$(01)^*11$

Example:



(b) Does the excitation table of part (a) contain any critical races? Explain briefly.

May 1971 Computer Design Qualifying Exam

Problem 1.

Define and give an example of each of the following techniques for speeding up binary multiplication in a parallel arithmetic unit.

(a) multiplier recoding

(b) carry save addition

2. Problem

(a) Define an *execute* instruction.

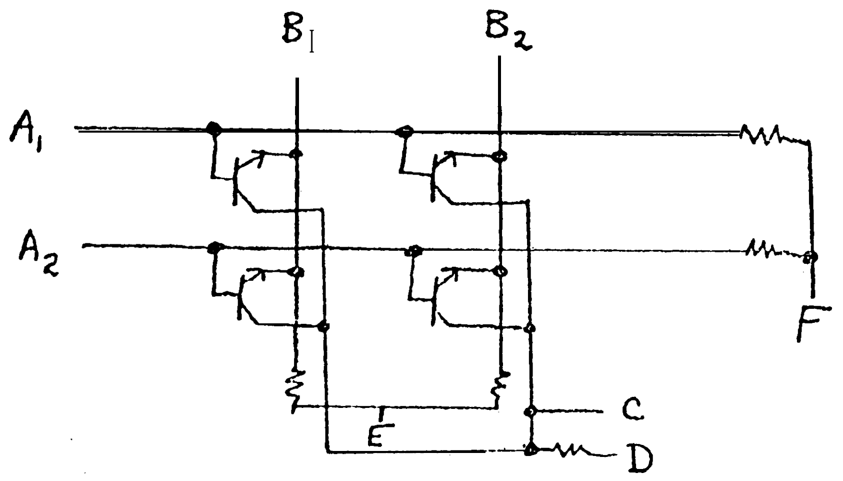
(b) What modifications would be necessary to add an execute instruction to the HP 2116 computer?

(c) Write an *execute* instruction microprogram for the HP 2116 with the modification of part (b).

(d) Now do parts (a)-(c) for a *repeat* instruction.

Problem 3.

Consider the following circuit:



(a) To make this circuit act like a read-only memory, what should be connected to each of the labelled points? Note: The connection to the C line may be present or missing at each cell.

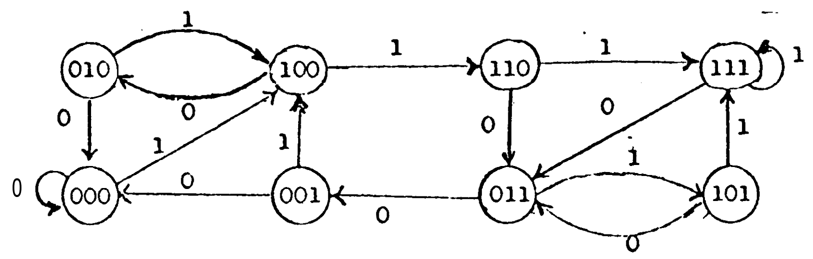
(b) Explain your addressing scheme. Discuss polarity of address lines and associate binary addresses with the memory cells shown.

(c) What voltage appears on the output line if the selected cell is connected? If the cell is not connected?

(d) How can this memory be expanded to a 16 location, 4 bit word memory?

Problem 4.

(a) For the machine M_1 shown below, how many components (at most) might be useful in a composite realization? How many states (at least) would each component have?



State transitions for M_1

(Outputs are, for the moment, unspecified, but it should be assumed that there are no equivalent states.)

(b) Draw a logic gate diagram realizing the above machine.

(c) Machine M_1 is to be used to correct the output of an unreliable single output machine M_2.

M_2 attempts to repeat each output symbol in its output string for four consecutive time periods. At most one of these 4 will be wrong. The output of M_2 drives the input to M_1. Assign outputs to the transitions of M_1 so that the M_1 output accompanying every fourth M_2 output will be correct.

(d) For the purpose that M_1 serves in part (c), can you design a better machine than M_1? If so, draw its logic gate diagram.

Problem 5.

A control timing unit for a navigation system computer is to be designed to generate the output sequences on lines A, *B*, and C. The sequences are to repeat after the 100 clock pulse times shown below.

Clock time	Output A	Output B	Output C
1, ..., 9	1	1	1
10, 11	0	0	1
12, ..., 21	1	0	1
22, 23	0	0	1
24, ..., 34		0	1
35, 36	0	0	0
37, ..., 48		0	0
49, 50	0	0	0
51, ... 61	1	1	0
62, 63	0	1	0
64, ... 72		1	0
73, 74	0	1	0
75, ..., 86		1	1
87, 88	0	1	1
89, ..., 98	1	1	1
99, 100	0	0	0

(a) Discuss the design of a clocked sequential network which will realize the desired performance using OR gates, AND gates, and RS flip-flops. How might the minimization of gates and flip-flops be accomplished?

(b) Obtain a solution using counters and read-only memories. Contrast the hardware complexity of this solution with that of part (a).

Problem 6.

In an N-valued logic (i.e. variables can take on the values 0, 1, 2, . . . , N-l), let

$m(x_1, x_2) =$ minimum of x_1 and x_2

$$C(x) = \begin{cases} x+1 & \text{for } x \neq N-1, \\ 0 & \text{for } x = N-1. \end{cases}$$

Prove that m and C form a functionally complete set of operations for $N \geq 2$.

Problem 7.

The following is a generating matrix of a linear code over GF(3).

$$\begin{bmatrix} 1 & 0 & 0 & 2 & 1 & 2 \\ 0 & 1 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{bmatrix}$$

(a) Give a check matrix for this code.

(b) How many words are in the code?

(c) What is the minimum distance of the code?

(d) If this code is transmitted through a ternary symmetric channel with channel matrix

$$\begin{bmatrix} 1-p & p/2 & p/2 \\ p/2 & 1-p & p/2 \\ p/2 & p/2 & 1-p \end{bmatrix}$$

what is the probability that the decoder will make a mistake?

Problem 8.

Specify a sequential state machine with 2 different S. P. partitions (preserved cover partitions) π_1 and π_2, such that $M_{\pi_1} \cong M_{\pi_2}$.

Note: The states of M_{π_1} correspond to the blocks of π_1.

April 1969 Theory of Computation Qualifying Exam

Problem 1.

Let S_t^x a denote the result of substituting the term t for ail free occurrences of the variable x in the wff *a*.

Definition: Let K be a first order theory and let E be a binary predicate of *K*. We say that *K is* a *first order theory with equality,* with E as equality predicate, if the following are **theorems of** *K:*

(1) $\forall x\, E(x, x)$

(2) Vx Vy $\{E(x, y) \supset [(S_x^z\,\alpha) \supset (S_y^z\,\alpha)]\}$, where a is any wff such that the individual variables x and y are free for the individual variable z in a (that is, no free occurrences of z in a lie within the scope of any quantifier (x) or (y)).

Your problem is to show that the equality relation in a first order theory is unique in an appropriate sense. That is, suppose there is a first order theory K with binary predicates E_1 and E_2 such that K is simultaneously a first order theory with equality with E_1 as equality predicate and with E_2 as equality predicate. Exhibit a wff or set of wffs which expresses within K the assertion that E_1 and E_2 are the same. Then show that this wff must be provable in *K*.

Problem 2.

Determine whether or not the following well formed formulas of first order logic are (1) satisfiable and (2) valid. Justify your answers by an appropriate rigorous argument.

(a) $\forall z\, \exists x\, \forall w\, \forall y\, \{[F(x, z) \wedge F(z, w)] \supset [F(y, z) \supset F(z, y)]\}$

(b) $\exists x\, \forall y\, \{F(x) \equiv [F(y) \vee F(x)]\}$

Problem 3.

Suppose R is a regular set of symbol strings over an alphabet Σ, and S is an arbitrary set of strings over Σ. Define T to be the set of strings x such that for some y $\in S$, xy belongs to *R;* that is,

$T = \{\, x \mid \exists y$ such that xy a $R\,\}$.

Prove that T is a regular set.

Problem 4.

State (without proof) the equivalence $(\equiv)$, inclusion(c), and non-inclusion $(\not\subset)$ relations you know of among classes of formal languages, and classes of languages defined in various ways (generation, accept ance, recognition) by various kinds of automata, finite or infinite, deterministic or non-deterministic. Draw a diagram using the above symbols to indicate relations. You need not show relations which follow by transitivity from other relations.

Problem 5.

Consider a progamming language L defined as follows:

```
<variable>    ::= <letter> | <variable> <digit>
<expression>  ::= 0 | <variable> | (<expression> + 1)
<statement>   ::= <variable> := <expression> |
                  for <expression> times do <statement> |
                  begin <statement>; <statement> end
<program>     ::= <statement>
```

The form **for** E times do s means that E is evaluated once, giving a numerical value e, and that S is then repeated e times.

(a) Show that for any primitive recursive function $f(x_1, x_2, \ldots, x_n)$, there is a program of L which sets $X0 = f(I_1, I_2, \ldots, I_N)$, where $I_1, \ldots, I_n$ are the initial values of $X1, \ldots, Xn$.

(b) Show that for every program in L, the final value of $X0$ is a primitive recursive function of the initial values of ail the variables.

Problem 6.

Let A, B, C, D, E be ground clauses and $R(A, B)$ denote the set of ground resoivents of A and B. Prove that if D is a clause in $R(R(A, B), C)$ then there is a clause E in $R(R(C, B), A) \cup R(R(C, A), B) \cup R(R(C, B), R(C, A))$ such that $E \subseteq D$. Give an example to show that it is not always true that there is an E such that $E = D$.

Problem 7.

Consider the general recursive definition

$$f(x) = \text{if } P(x) \text{ then } g(x) \text{ else } h(f(k(x))),$$

where the predicate P and the functions g, h, and k are understood to be primitives, and total. Express in terms of first order logic (including the ideas of validity, satisfiability, etc.) the proposition that for ail x such that $R(x)$ is true, $f(x)$ is defined and satisfies the relation $S(x, f(x))$.

June 1970 Theory of Computation Qualifying Exam

(Time limit: 4 1/2 hours)

Problem 1. (20 *points)*

Let $Q(x,y)$ be a binary predicate symbol. Consider the two wffs A and B given by

A: $\forall x\, \exists y\, Q(x,y)$
B: $[\forall x\, ((\exists y\, Q(x,y)) \supset Q(x,x))] \supset \forall x\, Q(x,x)$.

Which of the following is true?

(a) $\vdash A \supset B$
(b) I- $B \supset A$

If true, give a proof (using any standard derived rules of inference and metatheorems desired). Otherwise, give an interpretation as a counterexample.

Problem 2. (20 *points)*

For any language $L \subseteq \{0,1\}^*$, let

$$L' = \{\, yx \mid xy \in L,\ x \in (0,1\}^*,\ y \in (0,1\}^* \,\}$$

(a) Prove that if L is a regular language, then so is L'.
(b) Find a counterexample to show that the converse of (a) is not true.

Problem 3. *(JO points)*

Suppose P and Q are predicate symbols, S is a set of clauses, $\vdash_R$ denotes deducibility by resolution, and

$\{P(x)\} \cup S \vdash_R \square$
$\{Q(y)\} \cup S \vdash_R \square$.

Show *constructively* that

$\{P(x) \vee Q(y)\} \cup S \vdash_R \square$ ✋📬

Problem 4. *(15 points)*

Consider the following LISP functions operating on lists:

$x * y =$ if nx then y else $ax.[dx * y]$
$reverse[x] = rev[x; nil]$
$rev[x; y] =$ **if** nx **then** y **else** $rev[dx; ax.y]$

Prove that for any lists x and y,

$reverse[x * y] = reverse[y] * reverse[x]$.

Problem 5. *(10 points)*

The following program computes the remainder m mod n.

```
a:    if m < n then go to done;
      m := m − n;
      go to a;
done:
```

Write **a** wff of first order logic with one free predicate letter q whose truth for **all** interpretations of q is equivalent to the convergence of the above program.

Problem 6. (25 *points)*

The following ALGOL 60 program is intended to find the smallest positive integer expressible **as** the sum of two cubes of positive integers in two different **ways** given that there **is such a number less** than 8000.

```
begin integer array a[1:16000];
for i := 1 step 1 until 16000 do a[i] := 0;
for i := 1 step 1 until 20 do
    for j := 1 step 1 until i do
    a[i↑3 + j↑3] := a[i↑3 + j↑3] + 1;
for i := 1 step 1 until 8000 do
    if a[i] > 1 then go to done;
done:
end;
```

We want to use Floyd's formalism to prove this program correct (ignoring the question of termination) but first we must replace the **for** statements by ordinary **loops. The modified program is** shown on the next page.

```
      begin integer array a[1:16000];
      i := 1 ;
   6: if i > 16000 then go to c;
      a[i] := 0;
      i := i + 1;
      go to b;
   c: i := 1;
   g: if i > 20 then go to d;
      j := 1;
   f: if j > i then go to e;
      a[i↑3 + j↑3] := a[i↑3 + j↑3] + 1;
      j := j + 1;
      go to f;
   e: i := i + 1;
      go to g;
   d: i := 1;
   h: if i > 8000 then go to done;
      if a[i] > 1 then go to done;
      i := i + 1;
      go to h;
      done:
      end
```

In answering the following questions, attach labels $p1, p2$, etc. to appropriate points in the program.

(a) What assertion expresses the overall correctness of the program and to what point is it to be attached?

(b) How does the declaration **integer array** $a[1:16000]$ affect the assertion to be proved?

(c) Attach the appropriate Floyd assertions to the appropriate points in the program. Remember that the truth of each assertion must follow from those that immediately precede it in the flow of control.

October 1970 Theory of Computation Qualifying Exam

(Time limit: 7 hours)

Problem 1.

A professor of logic at Holy Smoke U. announces to the class one day that he has a new formal system for the first order predicate calculus. The wffs are the same, the axioms consist of all instances of tautologies, and the rules of inference are modus ponens and "p-constituent substitution" (see below). He enthusiastically begins to prove some lemma about the system when a student interrupts him and proves the professor's system is incomplete.

The professor rushes from the room but returns fifteen minutes later to state that he has now repaired the trouble. By adding a certain non-tautologous axiom he now has a complete logic system. Before he can even write down the axiom, another student jumps up and proves that now the system not only yields valid wffs as theorems but all wffs as theorems. (It is rumored the professor now teaches Sociology in a junior college somewhere in the deep South.)

What are the proofs the two students gave? Be precise.

Definition: An occurrence of a wff *B* in a wff *A* is a *p-constituent occurrence* of *B in A* if this occurrence of *B* is in the scope of no quantifier of A, and *B* is atomic or *B* consists of a quantifier and its scope.

P-constituent Substitution Rule: Let *A*, *B*, and C be wffs such that *B* is atomic or *B* consists of a quantifier and its scope. Let *D* be the result of replacing all p-constituent occurrences of *B* in *A* by occurrences of C. Then from *A* one may infer *D*.

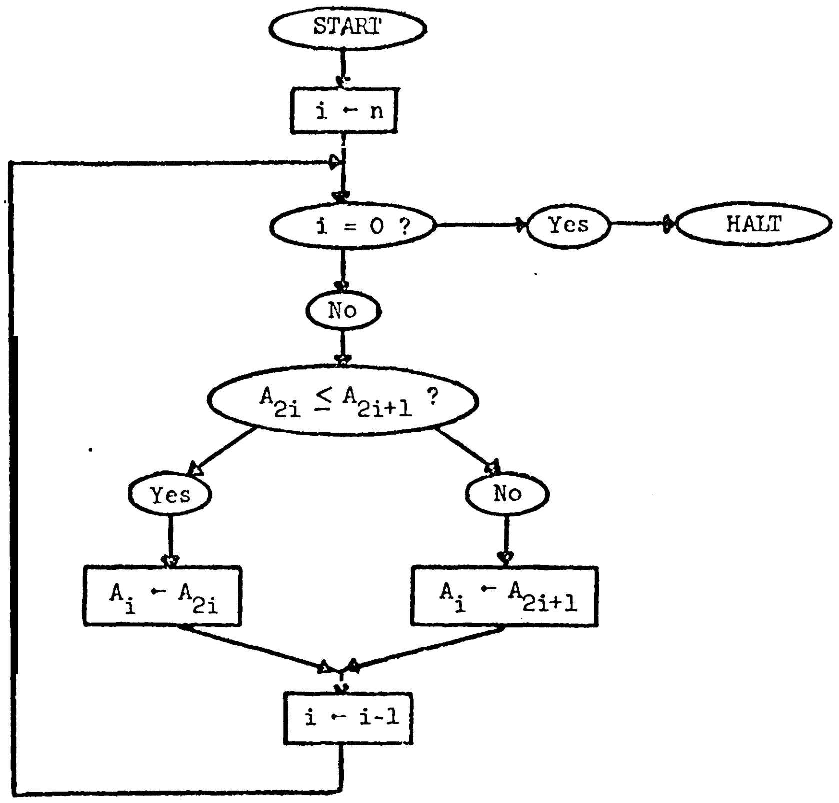
Problem 2.

Use the resolution method to determine whether the following wffs are valid.

(a) $\exists x\ \exists y\,[Mxy \wedge \forall r \sim Gxr] \vee \forall s\ \forall t\ \exists z\,[\sim Mst \vee Gtz \vee [Gsz \wedge \sim Mtz]]$

(b) $\exists x\ \exists y\,[Mxy \wedge \forall r \sim Gxr] \vee \forall s\ \forall t\ \exists z\,[\sim Mst \vee Gzt \vee [Gsz \wedge \sim Mtz]]$

Problem 3.



Prove that the above program computes the minimum of $A_{n+1}, A_{n+2}, \ldots, A_{2n+1}$ leaving the result in A_1, and leaving A_{n+1} through A_{2n+1} undisturbed. Use the method of assigning predicates to flowchart arcs.

Problem 4.

Define

$reverse[u] \leftarrow rev[u, nil]$

where

$rev[u, v] \leftarrow$ if nu then v else $rev[du, au.v]$.

Prove that for any list u,

$reverse[reverse[u]] = u.$

Note: The notations $nx, ax,$ dx, and $x.y$ stand for $null[x], car[x], cdr[x],$ and $cons[x, y]$, respectively.

Problem 5.

The programming languages L_0, L_1, L_2, L_3 all permit the use of integer variables and constants, and the operators $+, -, *$. They all permit assignments of expressions to variables, and combining commands into blocks. L_0 has no other operations. The other languages have the additional commands described below.

(1) L_1 allows the command

for E times do S

where E is an expression and S is a command. The expression is first evaluated, giving a value e, and then S is executed $|e|$ times.

(2) L_2 allows the command

call n

where n is a positive integer constant. If β is the block associated with the n-th **begin** in the program, the **call** command is executed by executing β.

(3) L_3 has both of the above commands.

For each pair (i, j) such that L_i can compute **a** function not computable by L_j, give an example of such a function. justify your answers.

Problem 6.

Let Σ be a finite set of symbols and let Σ^* denote the set of ail finite length strings of symbols from Σ including the string of zero length ϵ. Suppose we are given $A \subseteq \Sigma^*$ and $C \subseteq \Sigma^*$.

(a) Under what conditions does there exist a unique set X satisfying the equation

$$X = AX + C$$

where $AX = \{xy \mid x \in A, y \in X\}$ and $+$ is set union?

(b) Find all sd ut'ions to the equation in part (a).

Let A_{ij} and $C_i, 1 \leq i \leq n, 1 \leq j \leq n$ be subsets of Σ^*. Consider the simultaneous equations

$$X_i = \sum_{j=1}^{n} A_{ij}X_j + C_i.$$

(c) Give an algorithm for finding a solution.

(d) What can be said about the solution found in part (c) when the A_{ij}'s and C_i's are regular sets, context-free languages, recursive sets, or recursively enumerable sets? Justify your statements.

Problem 7.

Imagine that we write a "queue-processing" language over a set A of atoms. Think of a queue as a finite sequnce of atoms. The primitives are:

null(x),	which tests whether a queue x is the empty sequence;
$q(x, \alpha)$,	which adds the atom a at the right of the queue x;
$h(x)$,	which yields the leftmost atom of x; and
$t(x)$,	which yields the queue x with the leftmost atom removed.

Give a set of axioms for the queues which are sufficient for proofs about queue programs, in a manner analogous to LISP axioms (e.g. $cdr(cons(x, y)) = y$, $car(cons(x, y)) = x$). The only functions and predicates you should use are the queue primitives, set membership, and equality. Justify the appropriateness of your axioms.

April 1971 Theory of Computation Qualifying Exam

(Time limit: 7 hours)

Problem 0.

State briefly, without proofs:

(1) Kleene's recursion theorem (the fixed-point theorem of recursive function theory).

(2) Manuel Blum's "speedup" theorem concerning the computational complexity of recursive functions.

(3) The Krohn-Rhodes theorem on algebraic decomposition of finite automata.

(4) The completeness and decidability properties of second-order logic.

(5) The function of paramodulation and hyper-resolution in mechanical theorem proving.

(6) The meaning of "liberal" and "progressive,, as applied to schemata.

Problem 1. (20 *points)*

The language L is recognizable by an on-line Turing machine, that is, a Turing machine with initially blank tape, connected to an input unit from which it can bring in the string to be recognized, one character at a time, from left to right. The machine, on inputs of length n, never uses more than $f(n)$ squares of its tape, where $\lim_{n\to\infty} f(n)/\log(n) = 0$. Show that L is a regular (type 3) language. For definiteness, you may assume that the input operation writes the input character on the current tape square.

Problem 2. *(20 points)*

Suppose that the predicates P and Q satisfy the verification condition V_S of a command S, so that if we start with P true and execute S, we finish with Q true. In Hoare's notation, $P\{S\}Q$. Floyd asserts, in his paper "Assigning Meanings to Programs", that one can infer

(1) $(\forall x\ P)\ \{S\}\ (\forall x\ Q)$

(2) $(\exists x\ P)\ \{S\}\ (\exists x\ Q)$.

Scott pointed out an error in one of these. Give either proofs or counterexamples for (1) and (2).

Problem 3. *(15 points)*

In examples of resolution, one often sees the associative law stated by the two clauses

$$(1.1)\quad \overline{P}(u,v,x) \lor \overline{P}(v,w,y) \lor \overline{P}(u,y,z) \lor P(x,w,z)$$
$$(1.2)\quad \overline{P}(u,v,x) \lor \overline{P}(v,w,y) \lor \overline{P}(x,w,z) \lor P(u,y,z)$$

instead of

$$(2)\qquad f(f(u,v),w) = f(u,f(v,w))$$

in order to avoid the definition and use of the equality predicate. (In the above, $P(x,y,z)$ is intended to signify $f(x,y) = z$.)

We could first transform (2) into

$$(3)\qquad (f(u,v) = x \land f(v,w) = y) \supset (f(x,w) = z \equiv f(u,y) = z)$$

and thence to (1.1) and (1.2). In the following we concentrate on the second stage of **this** transformation.

Suppose a set of clauses contains the usual axioms for the equality predicate (regarding $x = y$ as an abbreviation for $E(x, y)$), and a function symbol f occurs only in iiterais of the form

$$f(e_1,e_2) = e_3 \text{ or } f(e_1,e_2) \neq e_3.$$

Systematically replace these **literals** by new iiterais

$$P_f(e_1, e_2, e_3) \text{ or } \overline{P}_f(e_1, e_2, e_3)$$

where P_f is a new predicate symbol. What is the relation between the satisfiabiiity of the original set of clauses and that of the transformed set? Justify your answer.

Problem 4. (20 *points)*

Define a counter-input Turing machine as a Turing machine, with an initially blank tape, to which is attached a counter C containing the input to the machine.

The possible values of the counter are the non-negative integers, and a special value called ∞. The only operations available on C are:

(1) Test whether C = 0.

(2) Decrease C by I: $C \leftarrow C \dot{-} 1$, where $0 \dot{-} 1 = 0$, $\infty \dot{-} 1 = \infty$, and otherwise $x \dot{-} 1 = x - 1$.

Define S_∞ as the set of counter-input machines which halt for ail possible initial values of C. Define S_j as the set of counter-input machines which halt for ail possible integer (not ∞) values of C.

Prove that (a) one of the above sets is recursively enumerable, and (b) that the other is not. (One can show recursive enumerability by showing that the set of **Gödel** numbers of machines **in** the given class is recursively enumerable, or by describing a program which prints the descriptions of the machines in the class.)

Problem 5. *(25 points)*

The parenthesis language *P* is defined by the context-free grammar $S \to \epsilon \mid (S) \mid SS$. It consists of strings of matched parentheses, such as

$$(\,(\,(\,)\,(\,)\,)\,(\,(\,(\,)\,)\,)\,(\,)\,).$$

Consider an arbitrary grammar G which generates a language $L_G \subseteq P$. For example, G might be

$$S \to (A \qquad A \to (()A) \qquad A \to)$$

Define the *weight* of a string of parentheses by

$$w(\text{"("}) = 1$$
$$w(\text{")"}) = -1$$
$$w(xy) = w(x) + w(y).$$

Define a *nest* of parentheses as a string x, such that for every decomposition $x = yz, w(y) = -w(z) \geq 0$. The *depth* of such a nest is the maximum $w(y)$ such that $x = yz$.

In a sentence (with accompanying derivation) of a context-free language, a *phrase* is a substring derived from a single non-terminal symbol. Show that there is a number $n = n(G)$ such that if uvw is a sentence of G, and v is a nest of parentheses of depth greater than *n*, then some substring of v contains a phrase of uvw.

To summarize the above in intuitive form: If a context-free language contains only properly nested parentheses, then every sufficiently deep nest of parentheses contains at least one phrase.

Problem 6. (21 *points)*

A Scott schema is a (Ianov schema-like) monadic functional schema which consists of one individual variable x, monadic function variables F_i, where F_0 is designated as the *root* function, monadic function constants g_i, and monadic predicate constants p_i.

For example,

```
F_0(x) ⇐ if p_1(x) then if p_2(x) then F_1(x)
                                  else F_1(g_1(x))
                   else x
F_1(x) ⇐ if p_3(x) then F_0(g_2(x)) else g_1(x)
```

We use F_∞ as the "divergence function", always defined by

$$F_\infty(x) \Leftarrow F_\infty(x).$$

An interpretation of a Scott schema is defined in the usual way, including assignment of an element of the domain as the initial value of x. The definitions of termination, divergence, freedom, and equivalence are extended in the natural way from flowchart schemas to Scott schemas.

Discuss the termination, divergence, freedom, and equivalence properties (decision problems) for Scott schemas. If you cannot find an answer for the general class of Scott schemas, define appropriate subclasses and discuss their decision problems. Justify your arguments.

October 1971 Theory of Computation Qualifying Exam

(Time limit: 6 hours)

Problem 1.

Consider the recursive schema P defined by

$F(x) \leftarrow$ if p(x) **then** x else $F(b(F(a(x))))$,

where p is a unary predicate symbol and a and b are unary function symbols.

It is an open problem whether this recursive schema can be translated to an equivalent flowchart schema. However, it can be translated if we add extra features to our class of flowchart schemas, such as counters, pushdown stacks, equality tests, or arrays.

Discuss clearly and in detail the translation of the recursive schema P to three such flowchart schemas.

Problem 2.

(a) Define modified algorithms for resolution and unification, incorporating an associative law for a particular function f of two arguments. Concretely, the new law of resolution must be such that for any set S of clauses, $\{\} \in$ R(S), i.e. the empty clause can be obtained from S by resolution, if and only if S is not satisfiable by any interpretation for which

$$(\forall x, y, z)\,[f(f(x, y), z) = f(x, f(y, z))].$$

(b) Sketch a proof that your algorithms achieve this goal.

Problem 3.

Ackermann's function $F(x, y)$ is defined recursively over the natural numbers as follows:

```
F(x, y) ← if x = 0 then y + 1
                    else if y = 0 then F(x-1, 1)
                                  else F(x-1, F(x, y-1)).
```

The following is an "Algol" program for computing $F(M, N)$ for any pair of natural numbers M **and** N. The **value of** $F(M, N)$ **is obtained in** $A[1]$ **where A is an infinitely long** integer **array.**

```
start: A[1] ← M;
       A[2] ← N;
       I ← 2;
a:     If I = 1 then go to halt;
       if A[I-1] = 0 then begin A[I-1] ← A[I]+1; I ← I-1; go to α end;
       if A[I] = 0 then begin A[I-1] ← A[I]-1; I ← 1; go to a end;
       A[I+1] ← A[I]-1;
       A[I] ← A[I-1];
       A[I-1] ← A[I-1]-1;
       I ← I+1;
       go to a;
halt:
```

(a) Attach an appropriate inductive assertion at point a **and show** partial correctness of the program.

(b) Prove that the program terminates for any pair M and N of natural **numbers.**

Problem 4.

Suppose that the function $c(P)$ is an effective measure of the computational cost of a parameterless program P depending monotonically on the size and running time of P. We require that c(P) be defined if P terminates, but it need not be defined if P does not terminate. Give the weakest conditions you can on the function c, such that **a** program of least cost having given output can be effectively constructed. Assume some fixed definite machine by which programs are executed.

Problem 5.

A lion and a Christian are released at points a and b respectively in an arena and move alternately, the Christian first. If it is the Christian's turn and she is at point x, she can go to her choice of points $c1(x)$, $c2(x)$, and $c3(x)$. If it is the lion's turn and it is at point x, it has a choice of $l1(x)$, $l2(x)$, and $l3(x)$.

Let E be a sentence of first order logic that axiomatizes whatever properties we wish to specify of a, b, the c's, and the l's.

Strategies for the Christian and the lion may be specified by giving predicates $pc1(x,y)$, $pc2(x,y)$ and $pl1(x,y)$, $pl2(x,y)$ that give the conditions for the Christian or lion to make move 1 or 2 when the Christian and lion are at positions x and y respectively.

If the positions of the Christian and lion are ever the same, the chase terminates with the lion eating the Christian.

Let F be a sentence of first order logic that axiomatizes whatever properties we wish to specify about the predicates pc1, $pc2$, $pl1$, and $pl2$. (It may include a, b, the c's, and the l's.)

(a) Write a sentence of first order logic that is provable if and only if the lion will catch the Christian in all interpretations of the constants, predicates, and functions satisfying E and F.

(b) Write a sentence of first order logic that is provable if and only if there is a strategy for the lion that will catch the Christian independently of what the Christian does in all interpretations of the constants, predicates, and functions satisfying E.

Problem 6.

Let PC stand for first order predicate calculus without equality, and let PCE stand for first order predicate calculus with equality. In PCE we allow atomic formulas of the form $t = u$, where t and u are any terms.

Herbrand's theorem for PC can be stated as follows: A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.

(a) Prove Herbrand's theorem for PC.

(b) Give a counterexample to show that the theorem does not hold for PCE.

(c) Suggest a modified Herbrand's theorem for PCE and prove it.

May 1974 Analysis of Algorithms Qualifying Exam*

Problem 1.

The following recurrence relation recently arose in connection with the analysis of a certain paging algorithm:

$$a_{n+1} = \frac{n^2-1}{n^2+1} a_n + \frac{n+1}{n^2+1}, \quad n \ge 1.$$

Find the asymptotic value of a_n as $n \to \infty$, to terms $O(n^{-2})$. (You need not relate constants appearing in your answer to "known" ones, but you should at least indicate how such constants could be computed.)

Problem 2.

When $n \le N$, the following procedure computes a random permutation $a[1] \ldots a[n]$ of a random combination of n things from $1, 2, \ldots, N$, i.e. a random one-to-one mapping from $\{1, \ldots, n\}$ into $\{1, \ldots, N\}$:

```
for $j \leftarrow 1$ step 1 until N do $M[j] \leftarrow j$;
for $j \leftarrow 1$ step 1 until $n$ do begin
    $r \leftarrow unif(j, N)$;
    $a[j] \leftarrow M[r]$; $M[r] \leftarrow M[j]$;
end.
```

Here $unif(a, b)$ is a procedure that computes a random integer in the closed interval $[a, b]$, each with probability $1/(b+1-a)$.

However, when N is large it is desirable to do the computation in $O(n)$ cells of memory. The two algorithms shown on the next page have been proposed for this problem.

*All of the Analysis of Algorithms quals were take-home exams, which students had about one week to work on.

Program A:

```
for j ← 1 step 1 until n do begin
    r[j] ← unif(j, N);
    k ← r[j];
    for i ← J-1 step -1 until 1 do
        if r[i] = k then k ← i;
    a[j] ← k;
end.
```

Program B:

```
for j ← 1 step 1 until n do b[j] ← j;
for j ← 1 step 1 until n do begin
    r[j] ← unif(j, N);
    if r ≤ n then begin a[j] • b[r]; b[r] ← b[j] end
    else begin
        a[j] ← r; k ← 1;
        while a[k J ≠ r do k ← k+1;
        if k ≠ j then begin a[j] ← b[k]; b[k] ← b[j] end
    end
end.
```

For example, suppose $N = 9$, $n = 7$ and suppose the *unif* procedure returns successively the values 3 9 7 9 6 7 9; then all three programs will set $a[1] \ldots a[7]$ to 3 9 7 2 6 1 4.

The purpose of this problem is to analyze Program A and Program B. On a particular computer it has been found that Program A takes $44n + V + 5W + 2$ units of time, while Program B takes $54n + 3X + 9Y + 42 + 3$ units, where

V = number of times "$k \leftarrow i$" is performed in Program A;
W = number of times "if $r[i] = k$" is tested in Program A;
X = number of times "$a[j] \leftarrow b[r]$" is performed in Program B;
Y = number of times "$a[j] \leftarrow b[k]$" is performed in Program B;
Z = number of times "$k \leftarrow k+1$" is performed in Program B.

Determine the minimum and maximum running time of each program, and also determine the average values of V, W, X, Y, and Z, as functions of n and N. Express the average values exactly, and also find the asymptotic behavior when n/N has a fixed value $a < 1$ as $n \to \infty$. Note: when determining the maximum and minimum running times you may assume that $N \geq 2n$.

Problem 3.

Let $\pi = \pi[1]\pi[2]\ldots\pi[n]$ be a permutation of $\{1, 2, \ldots, n\}$, and consider the following algorithm:

```
begin integer array $A[1:n]$; integer $k$;
    $(A[1], \ldots A[n]) \leftarrow (\pi[1], \ldots, \pi[n])$;
    loop: print $(A[1], \ldots, A[n])$;
    $k \leftarrow A[1]$;
    if $k = 1$ then go to finish;
    (All, $\ldots, A[k]) \leftarrow (A[k], \ldots, A[1])$;
    go to loop;
finish: end.
```

For example, when $n = 9$ and $\pi = 314592687$, the algorithm will print

```
3 1 4 5 9 2 6 8 7
4 1 3 5 9 2 6 8 7
5 3 1 4 9 2 6 8 7
9 4  1 3 5 2 6 8 7
7 8 6 2 5 3 1 4 9
1 3 5 2 6 8 7 4 9
```

and then it will stop.

Let $m = m(n)$ be the total number of permutations printed by the above algorithm. Prove that m never exceeds the Fibonacci number F_{n+1}. In particular, the algorithm always halts.

Extra credit problem.

Let $M_n = \max (m(n) \mid \pi$ a permutation of $(1, \ldots, n\}$. Find the best upper and lower bounds on M_n that you can.

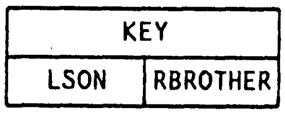
May 1975 Analysis of Algorithms Qualifying Exam

This entire exam is based on an interesting way to represent priority queues as a special kind of linked *forest*. Each node of the forest contains a KEY field, and when the forest is traversed in *preorder* $P_1 P_2 \ldots P_n$ we have

$$\text{KEY}(P_1) \le \text{KEY}(P_2) \le \ . \ . \ . \ \le \text{KEY}(P_n).$$

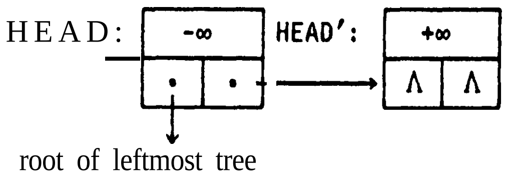
Furthermore, the rightmost member of every family will be **sonless.** In particular, if the forest is **nonempty** its rightmost tree will consist of a root alone.

The forest is represented in the "natural" way as a binary tree, so that each node has two link fields LSON (pointing to the leftmost son of this node, if any) and RBROTHER (pointing to the node's next brother to the right, if any).



For convenience in describing the algorithm, we shall assume that there are two header nodes, pointed to by HEAD and HEAD', where

KEY(HEAD) = $-\infty$, KEY(HEAD') = $+\infty$,
LSON (HEAD) = pointer to root of leftmost tree,
RBROTHER(HEAD) = HEAD',
LSON(HEAD') = RBROTHER(HEAD') = Λ.



If the priority queue is empty, LSON(HEAD) = A. Actually it turns out that the algorithm never looks at any of these fields except LSON(HEAD), so the other values (e.g. $-\infty$) need never be stored in memory, and the HEAD' node doesn't need to be present at all! However, it's easier to explain the algorithm (see top of next page) if we assume that these two artificial nodes exist.

We shall attempt to analyze the behavior of the algorithm when it is applied to the successive insertion of n distinct keys in random order. In particular, if $a_1 a_2 \ldots a_n$ is a permutation of $\{1, 2, \ldots, n\}$, we will consider the behavior of Algorithm I when a, is inserted, assuming that $a_1, \ \ldots, a_{n-1}$ have previously been inserted (in that order) into an initially empty forest.

Let A be the number of times step I2 is performed, and let B be the number of times step I4 is performed. Let (T_1, T_2, T_3) be $(1,0,0)$, $(0,1,0)$, or $(0,0,1)$ according as the algorithm terminates because of the respective conditions Q = A in I3, or X < KEY(Q) in 13, or RBROTHER(Q) = A in 14. Then the running time of the algorithm on most computers will be $\alpha A + \beta B + \tau_1 T_1 + \tau_2 T_2 + \tau_3 T_3 + \gamma$ for some constants a, $\beta, \gamma, \tau_1, \tau_2, \tau_3$.

Algorithm I *(Insert into priority queue)*. This algorithm inserts a new node, which will contain a given key X, into the forest. If there are other nodes with key X, the new node will precede them in preorder.

I1. [Create new node and initialize,] Set R + AVAIL, KEY(R) $\leftarrow$ X, LSON(R) $\leftarrow$ A. Also set P $\leftarrow$ HEAD.

I2. [Prepare to insert into family.] (At this point we have KEY(P) < X ≤ KEY(RBROTHER(P)), and we want to insert X among the descendants of node P preserving the preorder condition.) Set Q $\leftarrow$ LSON(P).

13. [Insert at left of family?] If Q = A or X ≤ KEY(Q), set LSON(P) + R, RBROTHER(R) $\leftarrow$ Q, and terminate the aigorithm.

14. [Loop on sons of P.] (Now KEY(Q) < X.) If RBROTHER(Q) = A, go to 16.

I5. [Correct son found?] If X ≤ KEY(RBROTHER(Q)), set P $\leftarrow$ Q and return to step 12. Otherwise set Q + RBROTHER(Q) and return to step 14.

16. [Insert at right of family.] Set RBROTHER(Q) + R, RBROTHER(R) $\leftarrow$ A, and terminate. ▮

<u>Problem 1</u>. (IO *points)* Find a simple relation between the forest obtained from the permutation $a_1 a_2 \ldots$ *a,* and the forest obtained from the permutation *(n*+ 1 -a_1)(n+1−a_2)... (n+ 1 -*a,).*

<u>Problem 2</u>. *(10 points)* Find the generating function g,,(z) in which the coefficient of z^k is the probability that the forest constructed from $a_1 a_2 \ldots$a, has exactly k trees.

<u>Problem 3</u>. (20 *points)* Let x, y be integers with $1 \le x < y \le n$. Find the probability that the forest constructed from $a_1 a_2 \ldots$ a,, contains the keys x and y on level 0 in the roots of *adjacent* trees. (For example, let $n = 4$, $x = 1$, $y = 3$. Then the permutations which make 1 and 3 appear in adjacent roots are 1324, 1342, 3124, 3142, 3412, 4312; and the probability is 1/4 in this case.)

<u>Problem 4</u>. *(30 points)* Determine the average values of A, B, T_1, T_2, and T_3 when a, is inserted. (Hints: Use the results of Problems 1, 2, and 3 to deduce appropriate recurrence relations. Try to find "closed form" expressions which solve the recurrences. It can be done.)

<u>Problem 5</u>. *(10 points)* The running time of Algorithm I on a certain computer is $7A + 6B + 2T_1 + 5T_2 + T_3 + 6$ units. What is the exact value of the maximum time it will take to insert a,, over all permutations $a_1 a_2 \ldots$ *a,,,* on this computer? (Give the answer as a function of *n,* for all $n \ge 1$. Note that as in problem 4 only the time to insert the one key a, is being considered here, *not* the total time to build the entire tree from $a_1 a_2 \ldots$a,.)

<u>Problem 6.</u> *(10 points)* Find a simple formula for the generating function

$$F(z) = \sum_{n \ge 0} f_n z^n = 1 + z + {}^2 + 2z^3 + 4z^4 + 9z^5 + \ldots,$$

where f_n is the number of forests on *n* elements having the "sterile rightmost" rightmost property.

<u>Problem 7.</u> *(10 points)* Continuing Problem 6, determine the asymptotic value of f^n, with relative error $1/\sqrt{n}$, that is, find an explicit function $\phi(n)$ such that $f_n = \phi(n) + O(\phi(n)/\sqrt{n})$.

May 1977 Analysis of Algorithms Qualifying Exam

Problem 1. (20 *points)*

Let $B^{(n)}$ be the set of all 2^n n-tuples of O's and l's. A k-cube or clause C over $B^{(n)}$ is a subset of 2^k n-tupies having specified values in *n-k* components; for example, if $n = 5$ and $k = 2$, the clause C = $0{*}10{*}$ is the set {00100, 00101, 01100, 01101}. Given a subset $V \subseteq B^{(n)}$, a *prime implicant* of V is a clause C $\subseteq V$ such that no clause properly containing C is contained in V. The purpose of this problem is to show that an algorithm which computes prime implicants of a given set V in time proportional to the number of clauses contained in V will have average running time almost, but not quite, linear in the length of its output, when V contains 2^{n-1} elements.

Suppose V is a randomly chosen subset of $B^{(n)}$ having m elements, where all such subsets are equally likely. Let $c(n, m)$ be the average number of clauses contained in V, and let $p(n, m)$ be the average number of prime implicants contained in V.

(a) Find exact expressions for $c(n, m)$ and $p(n, m)$. (These formulas need not be summed in "closed form".)

(b) Prove that $c(n, 2^{n-1})/p(n, 2^{n-1}) \to e$ when $n = 2^{2^v}$ and v is an integer, $v \to \infty$. *Hint:* Show that almost all of the clauses contributing to $c(n, 2^{n-1})$ and $p(n, 2^{n-1})$ are v-cubes.

(c) Prove that the ratio $c(n, 2^{n-1})/p(n, 2^{n-1})$ is $O\left(\frac{\log \log n}{\log \log \log n}\right)$.

(d) Prove that there exist integers n such that the ratio $c(n, 2^{n-1})/p(n, 2^{n-1})$ is arbitrarily large, in spite of the fact that the limit in (b) exists.

Problem 2. *(10 points)*

Let $G = (V, E)$ be a directed graph. G *is strongly connected* if, for every pair of vertices v and w, there is a path in G from v to w and a path in G from w to v. Suppose G is strongly connected. A *minimum equivalent digraph* G' is a subgraph of G which (i) contains all the vertices of G, (ii) is strongly connected, and (iii) contains a minimum number of edges among all graphs with properties (i) and (ii). Note that a minimum equivalent digraph need not be unique.'

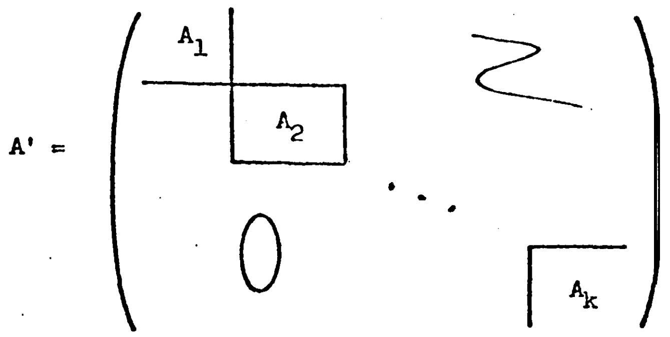
Show that the following problem is NP-complete:

Input: A strongly connected graph G and an integer k.

Question: Does G have a minimum equivalent digraph containing k or fewer edges?

Problem 3. (20 *points)*

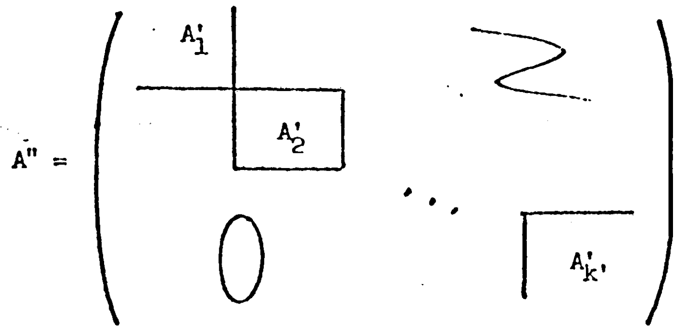
Let A = (a_{ij}) be an $n \times n$ non-singular real-valued matrix. That is, A has a non-zero determinant. We wish to permute the rows of A and *independently* permute the columns of A so that the permuted matrix A' has the form



where the submatrices $A_1, A_2, \ldots, A_k$ are square and lie on the main diagonal of A', all elements of A' below $A_1, \ldots, A_k$ are zero, and the elements above $A_1, \ldots, A_k$ can have any value.

Of course, the original matrix A has the desired form with $k = 1$, but we wish to make k maximum.

(a) Prove that if k is maximum, *A'* is unique in the following sense. If



is another rearrangement of *A*, then the set of rows in each submatrix A_i' consists of the union of the sets of rows in one or more submatrices $A_{j_1}, \ldots, A_{j_l}$, and the set of columns in A_i' consists of the union of the sets of columns in $A_{j_1}, \ldots, A_{j_l}$. Thus *A'* is unique up to permutations of rows and columns belonging to the same submatrix A_i.

(b) Describe an efficient algorithm for finding a permuted matrix A' with maximum value of *k*. What is the worst-case running time of your algorithm?

Problem 4. *(20 points)*

Consider n records with keys $K_1 < K_2 < \ldots < K_n$, and access frequencies $p_1, p_2, \ldots, p_n$, where $\sum_i p_i = 1$. We wish to store them in a data structure with fast average retrieval time. The problem calls for the study of the following scheme, which is a mixture of sequential files and search trees.

Consider any subset of $l-1$ keys $S = \{K_{i_1} < K_{i_2} < \ldots < K_{i_{l-1}}\}$. It divides the rest of the keys into l subfiles; the j-th subfile consists of keys between $K_{i_{j-1}}$ and K_{i_j}. As shown in Figure 1, a structure called l-file can be obtained by linking these l-1 keys and the l subfiles together, where each subfile is organized as a sequential file. Note that for $l > 2$, several distinct l-files can be formed even for the same S (see Figure 2). For an l-file L, the procedure to search for a key K in the file is to start at the root, comparing keys and branching accordingly down the tree until a subfile is encountered; then a sequential search is performed. The process halts whenever the key K is located. Let $C_L(K)$ be the number of keys examined before locating K. The *average cost* for L is then $\overline{C}_L = \sum_{1 \le i \le n} p_i \cdot C_L(K_i)$. In this problem, you are asked to design algorithms for computing $A_l(p_1, p_2, \ldots, p_n)$, the minimum average cost for any l-file, where l is fixed.

(a) Give an $O(n \log n)$-time algorithm for computing A_2.

(b) Give an $O(n^3)$-time algorithm for computing A_l.

(c) Let $p_i^{(n)} = 1/i \cdot H_n$, where $H_n = \sum_{1 \le i \le n} 1/i$ is the n-th harmonic number.

(1) Prove that, for each l, the limit

$$a_l = \lim_{n \to \infty} \frac{A_l(p_1^{(n)}, p_2^{(n)}, \ldots, p_n^{(n)})}{n/\ln n} \text{ exists.}$$

(2) Find a recurrence equation for determining a_l. Evaluate a_2, a_3, and a_4.

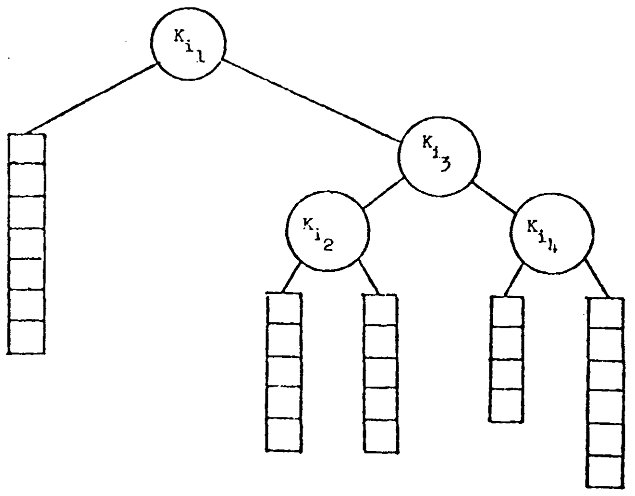


Figure 1. A 5-file.

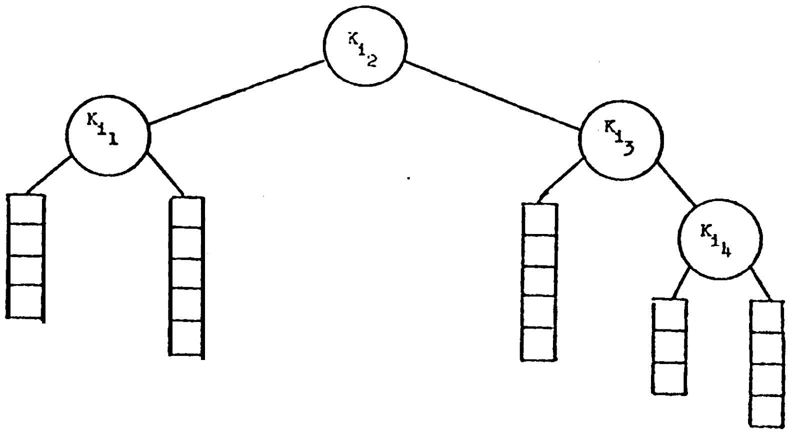


Figure 2

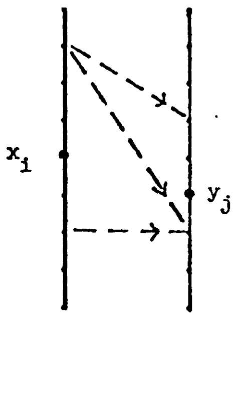
Problem 5. (20 *points)*

The Stanford boxing team is going to meet the USC boxing team next month. Let $x_1 > x_2 > \ldots > x_n$ be the linear ranking of the n members on the Stanford team among themselves, and $y_1 > y_2 > \ldots > y_n$ the ranking of members of the USC team. If the two teams had never met before, it would be fair to assume that all $\binom{2n}{n}$ rankings of the $2n$ members taken together are equally likely. It would then be straightforward to compute $P(i,j)$, the probability that x_i can beat y_j in the opening match this year. For example, $P(1,1) = 1/2$. *However,* since the two teams have met, and in all contests Stanford has won (the number of the contests, and the matches can be arbitrary), intuitively x_i would have a chance greater than or equal to $P(i,j)$ over y_j. Prove it.

Remark 1. A formula for $P(i,j)$ can be found on p. 191 of Knuth, Vol. 3, but it is not necessary to know this.

Remark 2. First try to prove it for $i = 1, j = 1$. Of course, any particular method may not generalize easily to other i and j. Partial credit will be given to solutions of special cases.

Remark 3. Assume boxers' skills satisfy transitivity, and do not improve or deteriorate with time.



May 1978 Analysis of Algorithms Qualifying Exam

Problem 1. *(20 points)*

Lt U_n be a fully balanced binary tree of height *n*, and S_n its set of internal nodes. For any integer $m > 0$, an *m-storage hashing scheme* is a function h from S_n into $\{1, 2, \ldots, m\}$; let $H(m,n)$ denote the set of all such h's. For any subset $F \subseteq S_n$, we can use h to store the elements of F by organizing all the elements that are hashed to the same location as a balanced tree. Assuming that each element of F is equally likely to be retrieved, we define the *retrieval cost* of h on F by

$$c(h, F) = \frac{1}{|F|} \sum_{1 \le j \le m} n_j \lg(n_j+1), \quad \text{where } n_j = |\{v \mid v \in F, h(v) = j\}|.$$

In this problem, we are interested in the efficient storage of a particular family of subsets of S_n. Let $\mathcal{T}_n$ be the family of n-node subtrees of U_n that contain the root (see Figure 1). The efficiency of h is measured by its worst-case retrieval cost for any $T \in \mathcal{T}_n$, i.e.,

$$f(h) = \max_{T \in \mathcal{T}_n} c(h, T).$$

(We have identified T with its set of nodes in the above definition.)

(a) *(5 points)* A hashing scheme $h_0 \in H(n, n)$ is defined as follows. For each $v \in S_n$, let $i(v)$ be the binary sequence of length $n-1$ or less associated with v as shown in Figure 2. Let $h_0(v) = |l-k| + 1$, where l and k are respectively the number of 0's and 1's in i(v). Determine the order of magnitude of $f(h_0)$ for large n.

(b) *(15 points)* Let $\epsilon > 0$ be any fixed number. Prove that, if n is sufficiently large, then there exists an $h \in H(n, n)$ such that $f(h) \le 4+\epsilon$. (*Hint:* Almost all h will do.)

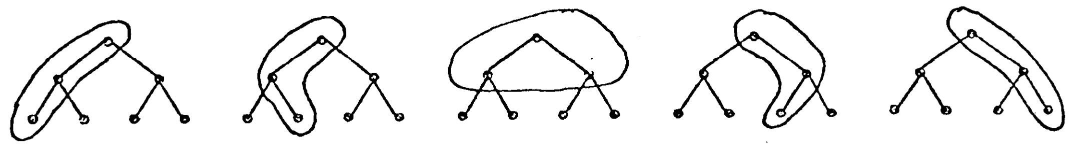


Figure 1. The family $\mathcal{T}_n$ for $n = 3$.

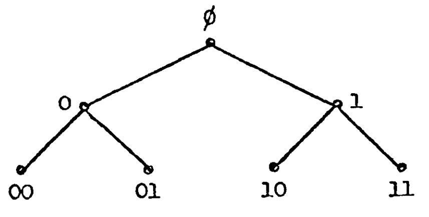


Figure 2. The tree U_n for $n = 3$. The sequence $i(v)$ is shown for each node v, with $i(\text{root})$ = empty string.

Problem 2. (20 *points)*

Consider sorting networks constructed from comparator modules as discussed in Section 5.3.4 of Knuth, where a comparator module can be represented functionally as in Figure 1(a). Let $\hat{S}(n)$ be the minimum number of comparators needed in any sorting network for n inputs.

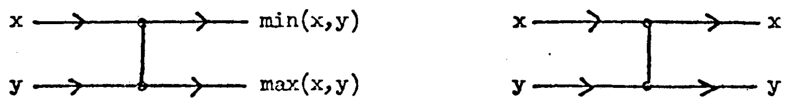
Suppose we are given a bunch of comparators, but among them one comparator may be faulty, in the sense that this comparator works as in Figure 1(b) instead of Figure 1(a). Our problem is to investigate methods of constructing valid sorting networks without locating the faulty comparator.

Formally, a ***1-fault** tolerant sorting network* is a network of comparators such that it remains a valid sorting network if exactly one of its comparators is faulty. Let $\hat{S}_1(n)$ be the minimum number of comparators needed for such a network with n inputs.

(a) *(5 points)* Determine $\hat{S}(3)$.

(b) *(15 points)* Show that $\hat{S}_1(n) \le \hat{S}(n) + n - 1$.

(Unfortunately, the intended proof for part (b) of this problem was incorrect. Perhaps the reader can supply a proof or disproof of this result.)



(a) a comparator (b) a faulty comparator

Figure1

Problem 3. (20 *points)*

Consider the following scheduling problem. We wish to carry out a number of tasks whose memory requirements are to be satisfied from a large block of storage consisting of m memory locations. Each task T_i requires a contiguous block of s_i storage locations out of the m.

We wish to schedule the tasks into the memory in an on-line fashion using the following rule: When a new task T_i requiring s_i memory locations is to be carried out, it is assigned to the *first* block of s_i contiguous storage locations which are currently empty.

Note that the first tasks to be scheduled will be packed densely into the first part of the memory, but that as tasks are completed the free memory will consist of "holes" separated by occupied areas.

(a) *(10 points)* Describe a data structure to implement this scheduling rule, assuming that the total amount of available memory is infinite $(m = \infty)$. Your data structure should allow each of the following operations to be executed in $O(\log n)$ time, where n is the number of tasks currently occupying the memory.

(i) Given a new task, assign it to the first block of memory into which it will fit, and modify the data structure accordingly.

(ii) Given a task which has just been completed, modify the data structure to represent the freeing of the storage previously occupied by the task.

(b) *(10 points)* Suppose the total amount of memory (m) is finite. We wish to modify the data structure to include a *waiting list* of tasks still to be scheduled but which will not currently fit into the memory. When storage becomes freed as a task is completed, the task which has been waiting for the longest time and which now fits into the memory is scheduled. Modify your data structure so that the following operations each take the indicated amount of time.

(i) Given a new task, assign it to the first block of memory into which it fits. If it will fit nowhere, add it to the waiting list. ($O(\log n)$ time.)

(ii) Given a completed task, free the storage it occupied previously and schedule the task which now fits and has been waiting the longest. Continue scheduling tasks from the waiting list until no more will fit. ($O(\log n)$ time plus $O(\log n)$ time per task scheduled,)

Problem 4. (20 *points)*

Suppose we are given a collection of rectangles which we wish to put in non-overlapping fashion into a rectangular bin which is open at the top. One side of each of the rectangles is parallel to the bottom of the bin and the rectangles may *not* be rotated. The *height* of a packing is defined to be the distance from the bottom of the bin to the top of the block which sticks up the farthest. We wish to minimize this height. See Figure 1 below.

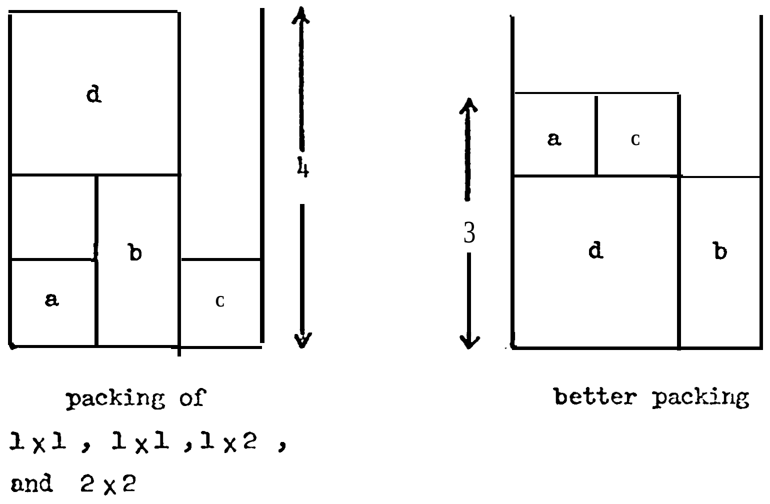


Figure 1

(a) Prove that it is NP-complete in general to determine whether a given collection of blocks can be packed into a given bin to satisfy a given upper bound on height.

The next three parts involve heuristics for packing the bin using the following fist *scheduling rule.* First, we construct a list of all the rectangles, ordered in some fashion. Next, we pack the rectangles one at a time from the list. To pack a rectangle, we place it into the position of lowest height in which it fits, breaking ties by placing the block as far left as possible. Note that Figure 1 gives packings which could have been obtained according to this rule from lists a, *b*, c, *d* and *d*, *b*, a, c, respectively.

(b) Give a class of examples to show that if the list can be in arbitrary order, the ratio of the height of the computed packing to that of a best packing can be arbitrarily large.

The *decreasing width heuristic* consists of arranging the blocks in decreasing order by width, breaking ties arbitrarily, and applying the list scheduling rule.

(c) Show that the height of any packing generated by the decreasing width heuristic is within a factor of 3 of the height of a best packing.

(d) Give a class of examples to show that for any ϵ the decreasing width heuristic can generate a packing with a height at least (3-c) times the height of a best packing.

Note: In parts (c) and (d), if you can't prove a factor of three, give the best upper and lower bounds you can on the worst-case ratio of the height of a decreasing width packing to the height of a best packing.

Problem 5. (20 *points)*

You are using a parallel computer. You have divided your problem into N subproblems, one for each processor. The running times for each are independent, with common mean M and variance V ($\sqrt{V} \ll M$), and are (very nearly) normally distributed. The program is complete when all N subproblems are done. Estimate the mean and variance of the running time, for large N. What does the distribution of running times look like?

Problem 6. (20 *points)*

Consider (I) binary trees and (2) LISP S-expressions, with N internal nodes, and with B bits of data stored at each external node. In each case, what is the information content of such a tree, given N, and assuming all N-node trees or S-expressions are equally likely? (Note: In an S-expression, a single copy of a sub-expression may be pointed to from several places within an expression.) Do conventional list representations approach optimal storage efficiency as $N \to \infty$? If not, suggest improvements.

Problem 7. *(20 points)*

N points are chosen uniformly from the unit cube. We say that point P dominates point Q if $P_x > Q_x$, $P_y > Q_y$, and $P_z > Q_z$. A point is maximal if no other point dominates it. Describe an efficient (say, polynomial time) algorithm to compute the expected number of maximal points as a function of N. Generalize to D dimensions,

Answers

SOLUTIONS TO ALGORITHMICS QUAL PROBLEMS -- 1974.

(These solutions were prepared by the professor in charge before the test was given and are presented unchanged from his sketched answers. Remarks in brackets [ ] are sane of the alternatives and refinements found by students taking the exam.)

Problem 1

After applying a summation factor we have $a_n = \frac{n}{n-1} \prod_{j \geq n} \left(1 + \frac{1}{j^2}\right) b_n$ for $n \geq 2$, where

$b_n = \sum_{1 \leq k < n} \frac{1}{k} \prod_{j \geq k} \left(1 + \frac{1}{j^2}\right)^{-1} = \sum_{1 \leq k < n} \frac{1}{k} f(k) + H_{n-1}$, where $f(k) = \left(\prod_{j \geq k} \left(1 + \frac{1}{j^2}\right)^{-1}\right) - 1 =$

$= \exp\left(- \sum_{j \geq k} \frac{1}{j^2} + O(k^{-3})\right) - 1 = \exp(-k^{-1} - \frac{1}{2} k^{-2} + O(k^{-3})) - 1 = -k^{-1} + O(k^{-3})$. (Note that

$\sum_{k > n} \frac{1}{k^a} = \zeta(a) - H_{n-1}^{(a)} = \frac{1}{a-1} n^{1-a} + \frac{1}{2} n^{-a} + \frac{a}{2!} B_2 n^{-a-1} + \ldots$, cf. ex. 6.1-8.) Therefore

$\sum_{1 \leq k < n} \frac{1}{k} f(k) = C - \sum_{k > n} \frac{1}{k} f(k) = C + n^{-1} + \frac{1}{2} n^{-2} + O(n^{-3})$ where $C = \sum_{k \geq 1} \frac{1}{k} f(k)$ is a constant;

$b_n = H_n + C + \frac{1}{2} n^{-2} + O(n^{-3})$. Now $\prod_{j \geq n} \left(1 + \frac{1}{j^2}\right) = \exp\left(\sum_{j \geq n} \frac{1}{j^2} + O(n^{-3})\right) = \exp(n^{-1} + \frac{1}{2} n^{-2} + O(n^{-3}))$

$= 1 + n^{-1} + n^{-2} + O(n^{-3})$, so $a_n = (1 + 2n^{-1} + 3n^{-2})(H_n + C + \frac{1}{2} n^{-2}) + O(n^{-3} \log n)$

$= H_n + C + 2n^{-1}H_n + 2Cn^{-1} + 3n^{-2}H_n + (3C + \frac{1}{2}) n^{-2} + O(n^{-3} \log n)$. [The constant C seems to be equal to $-.7665\ldots$.

Another approach is to rewrite the recurrence as $a_n = \sum_{1 \leq k < n} \frac{k+1}{k^2+1} - 2 \sum_{1 \leq k < n} \frac{a_k}{k^2+1}$ and to prove that a_n

has the form $\sum_{0 \leq s < t} (c_s \ln n + d_s) n^{-s} + O(n^{-t} \log n)$ by induction on t . Then the constants cs and ds

can be evaluated for $s = 1, 2, \ldots$ in terms of $c_0 = 1$ and $d_0 = C + \gamma$.]

Problem 2

Clearly $W = \binom{n}{2}$, so Program A depends only on V . The chance that $r[i] = k$ for each i and j is $1/(N-i+1)$, since k is always in the range $(i,N]$. Hence the average value of V is

$\sum_{1 \leq i < j \leq n} 1/(N-i+1) = n - (N+1-n)(H_N - H_{N-n})$. [Let $(k_{n-1}, \ldots, k_1)$ be the respective values of k which are compared to r[1] in Program A; it can be shown that each of the $(N-1)^{\underline{n-1}}$ possible (n-1) -permutations of $\{2,3,\ldots,N\}$ occurs equally often as $(k_{n-1},\ldots,k_1)$. Hence the generating function satisfies

$V_{n,N}(z) = V_{n-1,N-1}(z)\left(\frac{n-1}{N}(z-1)+1\right)$, for $n \geq 1$, and the variance of V comes to

$(N-n+1)(H_N - H_{N-n}) - (N-n+1)^2(H_N^{(2)} - H_{N-n}^{(2)})$.]

Similarly in Program B the probability that $r_j \leq n$ is $(n+1-j)/(N+1-j)$, hence the average value of X is $\sum_{1 \leq j \leq n} (n+1-j)/(N+1-j) = n - (N-n)(H_N - H_{N-n})$. Let Y_j be the average contribution to Y for a particular j ; then Y_j is the probability that $r_j \in \{a[1],\ldots,a[j-1]\}$ times the probability that $r_j > n$, since a[1] . . . a[j-1] is a random permutation (assuming that Program B is correct). Hence $Y_j = \frac{j-1}{N} \cdot \frac{N-n}{N+1-j}$,

$Y_{ave} = (N-n)(H_N - H_{N-n}) - n \cdot \frac{N-n}{N}$. Finally Z_j is $j-1$ if $X_j = Y_j = 0$; it is 0 if $X_j = 1$; otherwise Z_j is equally likely to be $0,1,\ldots,j-2$; hence

$$Z_j = (j-1)\left(\frac{N-n}{N+1-j} - Y_j \right) + \frac{0+1+\ldots+(j-2)}{j-1} Y_j = (N - \frac{1}{2} j)Y_j \quad ;$$

$$Z_{ave} = \frac{1}{4} \frac{N-n}{N} n(n-1) + \frac{1}{2} (N-1)Y_{ave} \; .$$

If $n = \alpha N$ then $H_N - H_{N-n} = \ln(\frac{1}{1-\alpha}) + O(n^{-1})$ and we have

$$V_{ave} = n\left(1 - \frac{1-\alpha}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)\right) + O(1) = n\alpha/2 + O(\alpha^2 n)$$

$$X_{ave} = n\left(1 - \frac{1-\alpha}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)\right) + O(1) = n\alpha/2 + O(\alpha^2 n)$$

$$Y_{ave} = n\left(a - 1 + \frac{1-\alpha}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)\right) + O(1) = n\alpha/2 + O(\alpha^2 n)$$

$$Z_{ave} = n^2 \left(\frac{1-\alpha}{4}\right) + \frac{n}{2\alpha} Y_{ave} + O(n) = \left(\frac{1}{2} - \frac{1}{6}\alpha\right)n^2 + O(\alpha^2 n^2)$$

as $n \to \infty$ and $\alpha \to 0$. [$X_{n,N}(z) = V_{n+1,N}(z)$.]

For the minimum running time of both programs it is easy to see that $r_1 \ldots r_n = N2 \ldots n$ yields $V,X,Y,Z = 0,n-1,0,0$ which has minimum time. But the maximum running time is not obvious in either program.

To maximize V we may note that Vj is the number of times the final value of a[j] was moved to the right in the M array of the original program. It can therefore be shown that if we were to continue the loop of Program A all the way to $j = N$, using r[j] =j for all $j > n$, the total value of V would be exactly equal to n minus the number of $j \leq n$ with r[j] = j . Consequently $V < n-1$; this value is achieved, e.g. when $r_1 \ldots r_n = N \ldots N$.

In Program B the contribution to the running time for fixed j is $3X_j + 9Y_j +_j 4Z$. . When $j = 1$ this is 3 if $X_1 = 1$, otherwise zero; when $j = 2$ it is 3 if $X_2 = 1$, or 4 if $Z_2 = 1$, or 9 if $Y_2 = 1$. When $j \geq 3$ there are several cases: (a) $Z_j = j-1$, $Y_j = X_j = 0$, total $4j-4$. (b) $Z_j = j-2$, $Y_j = 1$, $X_j = 0$, total $4j+1$; case (a) must have occurred for $j-1$. (c) $Z_j = j-3$, $Y_j = 1$, $X_j = 0$, total $4j-3$; case (a) must have occurred for $j-2$. (d) $Z_j < j-3$; we can always achieve a greater total by choosing r_j to be a 'new' value, without affecting other costs. A dynamic programming approach is now suggested: Let $c_k = \max\left\{ \sum_{1 \leq j \leq k} (X_j + Y_j + Z_j - 4j + 4) \mid Z_j = j-1 \right\}$. Then cl = 3, $c_2 = 3$, $c_3 = 5$, $c_4 = 8$, and for $k > 5$ we have

$$c_k = \max(0 + c_{k-1} , 5 + c_{k-2}, 1 + 5 + c_{k-3})$$

corresponding respectively to cases (a), (b), (c). The solution to this recurrence is easily seen to be $c_k = \lceil \frac{5}{2}(k-1) \rceil$, for $k \geq 2$. The maximum running time of Program B is $c_{n+1} + \sum_{1 \leq j \leq n} (4j-4) + 54n + 3$ $= 2n^2 + 54n + 3 + \lfloor \frac{1}{2} n \rfloor$, which is achieved by the sequence $n,n+1,n+1,\ldots,n+t,n+t$ for $n = 2t+1$ and by the sequence $n+1,n+1,\ldots,n+t,n+t$ for $n = 2t$.

Further notes: To prove that Program B works, let M^j be the contents of M after j iterations of the second loop of the original program. Then after j iterations of the second loop of Program B we have $b[k] = M^{j-1}[k]$ for $j \leq k \leq n$; and $b[k] = M^{j-1}[a[k]]$ for $1 \leq k < j$ if $a[k] > n$. (This 'invariant' is easily checked, once written down.)

Thus a different search method would convert Program B Into an n log n algorithm.

Similarly for Program A, the invariant is that $a[j] = M^i[k]$ during the loop on i .

Problem 3

If array element A[1] takes on k distinct values during the (possibly infinite) execution of the algorithm, we will show that $m \le F_{k+1}$ (hence m is finite). This is obvious for $k = 1$, since $k = 1$ can occur only when $\pi[1] = 1$.

If $k > 2$, let the distinct values assumed by A[1] be $d_1 < d_2 < . . . < d_k$. Suppose that $A[1] = d_k$ occurs first on the r-th permutation, and let $t = \pi[d_k]$. Then the (r+1) -st permutation will have $A[1] = t$ and $A[d_k] = d_k$. All subsequent permutations will also have $A[d_k] = d_k$ (they leave A[j] untouched for $j \ge d_k$), hence at most k-1 values are assumed by A[1] after the r-th permutation has been passed; By induction, $m-r \le F_k$, so m is finite and $d_1 = 1$.

Interchanging d_k with 1 in π produces a permutation π' such that $m(\pi') = r$, and for which the values d_k and t never appear in position A[1] unless $t = 1$. If $t = 1$ we have $r \le F_k$, since A[1] assumes at most k-1 values when processing π', hence $m = r+1 \le F_{k+1}$. If $t > 1$ we have $r \le F_{k-1}$ since A[1] assumes at most k-2 values when processing π' (note that $t = d_j$ for $j < k$) hence $m \le F_k+r \le F_{k+1}$.

Three hours of further concentration on this problem lead to the hypothesis that it is difficult either to prove or to disprove the conjecture $M_n = O(n)$; the upper bound F_{n+1} is exact only for $n < 5$.

[The upper bound applies more generally to any algorithm that sets $(A[1],\ldots,A[k]) \leftarrow (A[k],A[p_2],\ldots,A[p$
$(A[1],\ldots,A[k]) \leftarrow (A[k],A[p_2],\ldots,A[p_{k-1}],A[1])$ when $p_2 \cdots p_{k-1}$ is an arbitrary permutation of $\{2,\ldots,k-1\}$.

Computer calculations show that $M_6 = 11$, $M_7 = 17$, $M_8 = 23$, $M_9 = 31$, so $M_{n+1} - M_n$ maypossibly increase without limit. This search is speeded up slightly by restricting consideration to permutations without fixed points.

The long-winded permutations on 7, 8, 9 elements are 3146752 , 4762153 ; 61578324 ; 615972834 .

When $n \ge 3$ and $1 \le k \le 3$, exactly (n-1)! permutations π satisfy $m(n) = k$. It is conjectured that exactly (n-1)! permutations π will satisfy $A[1] = n$ at some stage.7

ANSWERS to ANALYSIS OF ALGORITHMS QUAL, June 1975 D. Knuth

[The subject matter for this examination is based on a forthcoming paper by Arne Jonsssen and Ole-Johan Dahl, who first proved most of the facts below, earlier this year. The following notes were written before grading the exams.]

A forest F is either empty or has the form

$$\begin{array}{ccc} x_1 & \ldots & x_m \\ | & & | \\ F_1 & & F_m \end{array}$$

for some $m \geq 1$ and some forests $F_1,\ldots,F_m$. The <u>sterile-rightmost forests</u> (SRFs) are characterized by the additional property that $F_1,\ldots,F_m$ are SRFs and F_m is empty. For convenience we shall identify nodes and keys, so that x_j denotes both the root of a forest and the key stored there. Since F_m is empty we can adopt the notation $(x_1F_1 \ldots F_{m-1}x_m)$ for nonempty SRFs.

Now the insertion algorithm on distinct keys can be described in the follouing recursive manner: Insertion of x into an empty SRF produces (x) . Insertion of x into $(x_1F_1\ldots F_{m-1}x_m)$ produces $(xx_1F_1\ldots F_{m-1}x_m)$ if $x < x_1$; or $(x_1\ldots x_i F_i' x_{i+1}\ldots x_m)$ if $x_i < x < x_{i+1}$ for some i , $1 \le i < m$, where F_i' is the result of inserting x into F_i ; or $(x_1F_1 \ldots F_{m-1}x_m x)$ if $x > x_m$. This recursive description of the algorithm guarantees that the keys of the SRF are always increasing in preorder; in particular $x_1 < \ldots < x_m$, and the value of i is uniquely defined above.

Let $\pi = a_1 \ldots a_n$ be a permutation of the distinct keys $\{a_1,\ldots,a_n\}$ and let F(n) denote the SRF obtained by successive insertion of $a_1\ldots a_n$. It follows easily from the above recursive definition that, when $n > 1$, we have $F(n) = (x_1F_1 \ldots F_{m-1}x_m)$ where $\{x_1, \ldots, x_m\}$ are the <u>left-to-right extrema</u> (LRXs) of the permutation π , i.e., those a_j such that $a_j = \min(a_1,\ldots,a_j)$ or $a_j = \max(a_1,\ldots,a_j)$. Furthermore $F_j = F(b_1 \ldots b_r)$ where $b_1 \ldots b_r$ is the permutation obtained from π by deleting all elements $\leq x_j$ and $\geq x_{j+1}$. Let us call this the permutation $\pi \cap (x_j,x_{j+1})$. (These facts are readily proved by induction on n).

Now we are ready to tackle the problems on the exam.

(1) If F is an SRF, let $\bar{F}$ be its "reflection" defined as follows: If F is empty, then $\bar{F}$ is empty, if $F = (x_1F_1 \ldots F_{m-1}x_m)$, then $\bar{F} = (\bar{x}_m\bar{F}_{m-1} \ldots \bar{F}_1\bar{x}_1)$. For example, if F = (a(b(cd)e(f)g)h) then $\bar{F} = (\bar{h}(\bar{g}(\bar{f})\bar{e}(\bar{d}\bar{c})\bar{b})\bar{a})$. If the nodes of F are $\{a_1,\ldots,a_n\}$ the nodes of $\bar{F}$ are $\{\bar{a}_1,\ldots,\bar{a}_n\}$.

Now let $\pi = a_1 \ldots a_n$ and $\bar{\pi} = \bar{a}_1 \ldots \bar{a}_n$ be permutations such that $a_i < a_j$ iff $\bar{a}_i > \bar{a}_j$. In particular if $a_1 \ldots a_n$ is a permutation of $\{1,\ldots,n\}$ we may let $\bar{a}_j = n+1-a_j$. Then $F(\bar{\pi}) = \overline{F(\pi)}$.

<u>Proof</u>: The LRXs of π are the LRXs of $\bar{\pi}$, and if $x_j < x_{j+1}$ are consecutive LRXs of π then $\bar{x}_{j+1} < \bar{x}_j$ are consecutive LRXs of $\bar{\pi}$ and $\bar{\pi}\cap(\bar{x}_{j+1},\bar{x}_j) = \overline{\pi\cap(x_j,x_{j+1})}$. The formula

$F(\bar{\pi}) = (\bar{x}_m\ F(\bar{\pi}\cap(\bar{x}_m,\bar{x}_{m-1}))\ldots F(\bar{\pi}\cap(\bar{x}_2,\bar{x}_1))\bar{x}_1) = (\bar{x}_m\ F(\overline{\pi\cap(x_{m-1},x_m)})\ldots F(\overline{\pi\cap(x_1,x_2)})\bar{x}_1) = \overline{F(\pi)}$ follows by induction on the length of π . ∎

(2) The number of LRXs for permutations of length n has the probability generating function defined by

$$g_1(z) = z\ , \quad g_n(z) = g_{n-1}(z)\left(\frac{2z+n-2}{n}\right) \qquad \text{for } n > 1\ ,$$

since a_1 is always an LRX, and a_n is an LRX with probability $\frac{2}{n}$ independently of the other a_j . Thus,

$$g_n(z) = z(2z)(2z+1)\ldots(2z+n-2)/n!\ ,$$

and the probnbility of m LRXs is the coefficient of z^m, namely

$$2^{m-1}\left[{n-1 \atop m-1}\right] / n!$$

by Eq. 1.2.9-27.

(3) Given consecutive LRXs $x < y$. If x was inserted after y, it must be a left-to-right minimum (not maximum), hence y being the smallest preceding element must also be a left-to-right minimum. Conversely, these conditions will make $x < y$ consecutive LRXs. Considering only the elements $\le y$, we are requiring y to appear first (probability $1/y$) and x to appear next (probability $1/(y-1)$), hence the probability of such permutations is $1/y(y-1)$. If x was inserted after y, a similar argument (or a dual one using the correspondence of question (1)) shows that the probability for this case is $1/(n+1-x)(n-x)$. The desired probability is the sum of these two.

(4) The average cost for inserting n elements is obtained by recurrences corresponding to the recursive formulation of the algorithm, namely the average cost on the first level plus the average cost incurred by involving the recursion. The latter is

$$\sum_{1\le x<y\le n-1} \frac{y-x}{n}\left(\frac{1}{y(y-1)} + \frac{1}{(n-x)(n-1-x)}\right) \text{(average cost of inserting y-x elements)}$$

since a_n lies between the x-th and y-th largest of $a_1 \ldots a_{n-1}$ with probability $(y-x)/n$, and they are consecutive LRXs with probability $1/y(y-1) + 1/(n-x)(n-1-x)$. The general form of recurrence that arises is therefore

$$C_n = f(n) + \sum_{0<x<y<n} \frac{y-x}{n}\left(\frac{1}{y(y-1)} + \frac{1}{(n-x)(n-x-1)}\right) C_{y-x}$$

$$= f(n) + \sum_{1\le d\le n-2} K_d\, C_d$$

where

$$K_d = \sum_{\substack{0<x<y<n\\ y-x=d}} \frac{y-x}{n}\left(\frac{1}{y(y-1)} + \frac{1}{(n-x)(n-x-1)}\right)$$

$$= 2 \sum_{\substack{0<x<y<n\\ y-x=d}} \frac{d}{n}\left(\frac{1}{y(y-1)}\right)$$

by symmetry (replacing (x,y) by $(n-y,n-x)$). Now

$$\sum_{\substack{0<x<y<n\\ y-x=d}} \left(\frac{1}{y-1} - \frac{1}{y}\right) = \frac{1}{d} - \frac{1}{n-1}, \quad \text{for} \quad 1 \le d \le n-2,$$

since it is a telescoping series, therefore the general recurrence takes the form

$$C_n = f(n) + \frac{2}{n} \sum_{1\le d\le n-1} \left(1 - \frac{d}{n-1}\right) C_d, \quad \text{for } n \ge 1 .$$

The first step is to eliminate the Σ from this recurrence:

$$n(n-1)C_n = n(n-1)f(n) + 2\sum_{1\le d\le n} (n-1-d)C_d + 2C_n$$

$$(n+1)\, nC_{n+1} = (n+1)\, n\, f(n+1) + 2 \sum_{1\le d\le n} (n-d)\, C_d$$

$$(n+2)(n+1)C_{n+2} = (n+2)(n+1)f(n+2) + 2 \sum_{1 \le d \le n} (n+1-d)C_d$$

$$\therefore n(n-1)C_n - 2(n+1)\,nC_{n+1} + (n+2)(n+1)C_{n+2} = g(n) + 2C_n \qquad \text{for } n > 1 ,$$

where $g(n) = n(n-1)f(n) - 2(n+1)\,n\,f(n+1) + (n+2)(n+1)f(n+2)$.

In generating function form, with $C(z) = \sum C_n z^n$ and $G(z) = \sum g(n)z^n$, this is

$$(1-z)^2 C''(z) = G(z) + 2C(z)$$

if we let $C_0 = 0$ and $g(0) = 2C_2$. [See exercises 5.2.2-28, 29,55 for a similar case.]

Let P be the differential operator defined by $Pf(z) = (1-z)f'(z)$. Then

$$P^2 f(z) = (1-z)(-f'(z) + (1-z)f''(z))$$

$$= (1-z)^2 f''(z) - (1-z)f'(z)$$

and our generating function satisfies the differential equation

$$G(z) = (P^2 + P - 2)C(z) = (P+2)(P-1)C(z)$$

We solve this in two steps, first solving

$$G(z) = (P+2)D(z)$$

for D , then solving

$$D(z) = (P-1)C(z)$$

for C . The coefficients of these equations satisfy respectively

$$g(n) = (n+1)D_{n+1} - nD_n + 2D_n$$

$$D_n = (n+1)C_{n+1} - nC_n - C_n \quad .$$

Thus $g(2) = 3D_3$, $g(3) = 4D_4 - D_3$, etc.; the values D_n for $n \ge 3$ are independent of $g(0)$ and $g(1)$. The recurrence for C_n In terms of D_n telescopes immediately.

$$C_n = C_3 + \frac{D_3}{4} + \cdots + \frac{D_{n-1}}{n} = C_3 + \sum_{3 \le k < n} \frac{D_k}{(k+1)}$$

To solve for D_n we have

$$(n+1)n(n-1)D_{n+1} = n(n-1)(n-2)D_n + n(n-1)g(n) ,$$

hence

$$D_n = \frac{1}{n(n-1)(n-2)} \sum_{0 < k < n} k(k-1)g(k) \qquad \text{for } n \ge 3 .$$

Now let us particularize the equations. For T1 we have $f(n) = \delta_{1n}$, hence $g(n) = 0$; $D_n = 0$ and $T1_n = T1_3$ for $n > 3$. For T2, we have $T2_n = T3_n$ by symmetry (problem 1), and $T1_n + T2_n + T3_n = 1$, hence $T2_n = T3_n = \frac{1}{2}(1 - T1_n)$. For A , we have $f(n) = 1$, hence $g(n) = 2$, and $D_n = \frac{2}{3}$, $A_n = A_3 + \frac{2}{3}(H_n - H_3)$ for $n \ge 3$.

Finally for B , we note that the first-level cost for $a_1 \ldots a_n$ plus the first-level cost for $\bar{a}_1 \ldots \bar{a}_n$ equals m , the number of trees in $F(a_1 \ldots a_m)$. By symmetry therefore, the average first-level cost f(n) for B is the average value of $\frac{1}{2}m$; and this is $H_n - \frac{1}{2}$, by the result of problem 2. In this case $g(n) \quad n(n-1)H_n - 2(n+1)\,nH_{n+1} + (n+2)(n+1)H_{n+2} - 1 = 2H_n + 2$; hence for $n \ge 3$

$$D_n = \frac{2}{3} + \frac{2}{n(n-1)(n-2)} \sum_{0 \le k < n} k(k-1)H_k = \frac{2}{3} H_n + \frac{4}{9}$$

by Eq. 1.2.7-?. The result of exercise 1.2.7-14 gives us the sum $\sum H_k/(k+1)$. hence we obtain the following formulas:

n	$T1_n$	$T2_n$	$T3_n$	A_n	B_n
1	1	3	0	1	0
	0	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$
	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{4}{3}$	1
4	$\frac{1}{3}$	$\frac{1}{3}$		$\frac{3}{2}$	$\frac{17}{12}$
≥ 3	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3} H_n + \frac{1}{9}$	$\frac{1}{3} H_n^2 + \frac{4}{9} H_n - \frac{1}{3} H_n^{(2)} - \frac{13}{27}$

The average total number of key comparisons to insert a_n is $A_n + B_n - T1_n - T3_n$; note that this is order $(\log n)^2$. [The formula for B_n is not the sort of thing one would guess easily by looking at a table of values; in fact it isn't even easy to prove it by induction from the recurrence relation!]

(5) Clearly the. maximum time M_n is nondecreasing in n . We use 'dynamic programming' (approximate for recursive algorithms): Let $f(A, B, T_1, T_2, T_3) = 7A + 6B + 2T_1 + 5T_2 + T_3 + 6$; then $M_1 = f(1,0,1,0,0) = 15$, and $M_2 = \max(f(1,0,0,1,0) , f(1,1,0,0,1)) = 20$. For $n \ge 3$ we have

$M_n = \max(f(1,0,0,1,0) , f(1,n-1,0,0,1) , f(1,n-2,0,0,0) - 6 + M_1 , f(1,n-3,0,0,0) - 6 + M_2 , \ldots, f(1,1,0,0,0) - 6 + M_{n-2})$;

. here the term $f(1,k,0,0,0) - 6 + M_{n-1-k}$ is the maximum time attainable when the first-level B value is k , for $1 \le k < n-2$. We find $M_3 = \max(20 , 26, 13 + M_1) = 28$, $M_4 = \max(20 , 32 , 19 + M_1 , 13 + M_2) = 34$, $M_5 = \max(20 , 38 , 25 + M_1 , 19 + M_2 , 13 + M_3) = 41$, and by induction

$$M_n = 6n + \lceil n/2 \rceil + 8 - \delta_{n2} .$$

(6) The number of SRFs with n nodes and m trees is $S_m = \sum \{f_{n_1} \ldots f_{n_{m-1}} \mid n_1 + \ldots + n_{m-1} = n-m\}$ = coef. of z^n in $z^m F(z)^{m-1}$. Hence $F(z) = 1 + \sum_{m \ge 1} z^m F(z)^{m-1} = 1 + z/(1-zF(z))$. The solution of this quadratic with F(0) finite is $F(z) = \frac{1}{2} + \frac{1}{2z} - \frac{1}{2z}\sqrt{1 - 2z - 3z^2}$.

(7) Now we write $1 - 2z - 3z^2 = (1-3z)(1+z)$ and work on the singularity nearest the origin, at $z = \frac{1}{3}$. Let $w = 3z$, we want to find the coefficient of w^n in

$\sqrt{(1-w)(1 + \frac{w}{3})} = \sqrt{(1-w)(1 + \frac{1}{3})} - \frac{1}{3} (1-w)^{3/2} (\sqrt{1 + \frac{w}{3}} + \sqrt{1 + \frac{1}{3}})$. Since $1/(\sqrt{1 + \frac{w}{3}} +$ $/q) = \sum c_n w^n$ converges for $|w| < 3$, we have $\limsup \sqrt[n]{|c_n|} = \frac{1}{3}$, and c_n is certainly $O(2^{-n})$. It follows that the coefficient of w^n in $(1-w)^{3/2} \sum c_n w^n$ is $O(\binom{3/2}{n}) = O(n^{-5/2})$. Thus the coefficient of w^n in $\sqrt{(1-w)(1 + \frac{w}{3})} = \sqrt{\frac{4}{3}} \binom{1/2}{n}(-1)^n + O(n^{-5/2})$. Now $\binom{1/2}{n} = \frac{1}{2} (-1)^{n-1} \Gamma(n + \frac{1}{2}) / \Gamma(\frac{1}{2}) \Gamma(n+1) =$ $(-1)^{n-1}/\sqrt{4\pi n^3} + O(n^{-5/2})$ by Stirling's formula. Going back to our original problem, we have

$$f_n = \left(\sqrt{\frac{3}{4\pi}}\, 3^n / n^{3/2}\right)(1 + O(1/n)) .$$

[Note: This exam seems to cover seven fundamental paradigms of algorithmic analysis.]

SOLUTIONS to Analysis of Algorithms Qual -- Spring 1977.

Problem 1.

(a) There are $\binom{n}{k}2^{n-k}$ k-cubes, each of which is contained in V with probability $\binom{2^n-2^k}{m-2^k}\Big/\binom{2^n}{m}$.

By Inclusion-exclusion, each k-cube is a prime implicant of V with probability

$$\left(\binom{2^n-2^k}{m-2^k}-\binom{n-k}{1}\binom{2^n-2\cdot 2^k}{m-2\cdot 2^k}+\binom{n-k}{2}\binom{2^n-3\cdot 2^k}{m-3\cdot 2^k}-\cdots\right)\Big/\binom{2^n}{m},$$ since $\binom{2^n-2^k-j2^k}{m-2^k-j2^k}$ is the number of

sets V containing a given k-cube together with a given set of j of its n-k "neighbors". For example, the neighbors of 0*10* are 1*10*, 0*00*, 0*11*; a clause contained in V is prime if and only if none of its neighbors is contained in V. Hence

$$c(n,m) = \sum_k \binom{n}{k}2^{n-k}\binom{2^n-2^k}{m-2^k}\Big/\binom{2^n}{m}\ ; \tag{1}$$

$$p(n,m) = \sum_k \binom{n}{k}2^{n-k}\sum_j \binom{n-k}{j}(-1)^j\binom{2^n-(j+1)2^k}{m-(j+1)2^k}\Big/\binom{2^n}{m}\ . \tag{2}$$

(b) First we make some estimates useful later. By Stirling's approximation

$$\ln\frac{X!}{(X-x)!} = x\ln X - \frac{x^2}{2X}+\frac{x}{2X}-\frac{x^3}{6X^2}+\frac{x^2}{4X^2}-\frac{x}{12X^2}+O\left(\frac{x^4}{X^3}\right)\ , \tag{3}$$

a formula useful when x grows more slowly than X. Now we can deduce that

$$\frac{\binom{2^n-2^k}{m-2^k}}{\binom{2^n}{m}} = \frac{m!}{(m-2^k)!}\ \frac{(2^n-2^k)!}{2^n!} = \left(\frac{m}{2^n}\right)^{2^k}\exp O\left(\frac{2^{2k}}{m}\right)\ . \tag{4}$$

There is a unique value of k, call it $k=\nu$, such that

$$n^{-4/3} \le \left(\frac{2^{n-1}}{2^n}\right)^{2^\nu} = 2^{-2^\nu} < n^{-2/3}\ . \tag{5}$$

It follows that

$$\lg\lg n+\lg\tfrac{2}{3} < \nu \le \lg\lg n+\lg\tfrac{1}{3} \tag{6}$$

We can now use (4) to observe that the terms of (1) increase until $k=\nu$ or $\nu+1$, then they decrease. Let $t_k(n,m)$ be the k-th term of the sum, and let $\alpha_n = 2^{-2^\nu}$, $m = 2^{n-1}$. Then

$$\sum_{k<\nu}\frac{t_k(n,m)}{t_\nu(n,m)} = O\left(\nu\,\frac{t_{\nu-1}(n,m)}{t_\nu(n,m)}\right) = O\left(\frac{\nu^2}{n\sqrt{\alpha_n}}\right) = O\left(\frac{(\log\log n)^2}{n^{1/3}}\right). \tag{7}$$

Furthermore

$$\frac{t_{\nu+1}(n,m)}{t_\nu(n,m)} = \frac{n-\nu}{2(\nu+1)}\alpha_n\left(1+O\left(\frac{2^{2\nu}}{m}\right)\right)\ ; \tag{8}$$

and for $k>0$ we have

$$\frac{t_{\nu+k}(n,m)}{t_\nu(n,m)} = O\left(n^k\,\alpha_n^{2^k-1}\right) = O\left(n^{k-\frac{2}{3}(2^k-1)}\right)$$

since

$$\frac{m!}{(m-2^{\nu+k})!}\ \frac{(2^n-2^{\nu+k})!}{2^n!} \leq \frac{m!}{(m-2^\nu)!}\ \frac{(2^n-2^\nu)!}{2^n!}\left(\frac{m}{2^n}\right)^{2^{\nu+k}-2^\nu}$$

Thus

$$\frac{c(n,m)}{t_\nu(n,m)} = 1 + \frac{n-\nu}{2(\nu+1)}\alpha_n + \frac{(n-\nu)(n-\nu-1)}{4(\nu+1)(\nu+2)}\alpha_n^3 + O\left(\frac{(\log\log n)^2}{n^{1/3}}\right) . \qquad (9)$$

Now let us consider (b). In this case $\nu = x$ and $a_n = n^{-1}$, hence

$$c(n,2^{n-1}) = \binom{n}{\nu}\frac{2^{n-\nu}}{n}\left(1 + O\left(\frac{1}{\log\log n}\right)\right) . \qquad (10)$$

Since almost all the contribution comes from $k = \nu$, we consider first which ν-cubes arc prime implicants. We have

$$\Sigma\binom{n-\nu}{j}(-1)^j\left(\frac{m}{2^n}\right)^{2^\nu(j+1)}\exp O\left(\frac{2^{2\nu}(j+1)^2}{m}\right)$$

$$= \sum_j\binom{n-\nu}{j}(-1)^j\left(\frac{m}{2^n}\right)^{2^\nu(j+1)} + O\left(\sum_j\binom{n-\nu}{j}\left(\frac{m}{2^n}\right)^{2^\nu(j+1)}\frac{2^{2\nu}(j+1)^2}{m}\right)$$

$$= \alpha_n(1-\alpha_n)^{n-\nu} + O\left(n^2\alpha^2(1+\alpha_n)^n\frac{2^{2\nu}}{2^{n-1}}\right)$$

and the remainder term is exponentially small. The contribution From $k \neq \nu$ is negligible since it is at most $c(n,2^{n-1}) - t_\nu(n,2^{n-1}) = O(c(n,2^{n-1})/\log\log n)$, hence

$$p(n,2^{n-1}) = \binom{n}{\nu}\frac{2^{n-\nu}}{ne}\left(1 + O\left(\frac{1}{\log\log n}\right)\right) \qquad (11)$$

. when $n = 2^{2^\nu}$ and ν is an integer.

(c) Similarly we have in general (cf. (9))

$$\frac{p(n,2^{n-1})}{t_\nu(n,2^{n-1})} = \exp(-n\alpha_n) + \frac{n-\nu}{2(\nu+1)}\alpha_n\exp(-n\alpha_n^2) + \frac{(n-\nu)(n-\nu-1)}{4(\nu+1)(\nu+2)}\alpha_n^3\exp(-n\alpha_n^4) + O\left(\frac{(\log\log n)^2}{n^{1/3}}\right)$$

$$= \frac{c(n,2^{n-1})}{t_\nu(n,2^{n-1})} + O(1) . \qquad (12)$$

Case 1, $n\alpha_n \leq \ln\nu - \ln\ln\nu$. Then consider only the $k = \nu$ term of p,

$$\frac{p(n,2^{n-1})}{t_\nu(n,2^{n-1})} \geq \frac{\ln\nu}{\nu}\left(1 + O\left(\frac{\nu\log\nu}{n}\right)\right) .$$

Case , $n\alpha_n \geq \ln\nu - \ln\ln\nu$. Then consider only the $k = \nu+1$ term,

$$\frac{p(n,2^{n-1})}{t_\nu(n,2^{n-1})} \geq \frac{n-\nu}{2(\nu+1)}\left(\frac{\ln\nu - \ln\ln\nu}{n}\right)\left(1 + O\left(\frac{\nu\ln\nu}{n}\right)\right) .$$

In both cases we have, by (15) and (9),

$$\frac{c(n,2^{n-1})}{p(n,2^{n-1})} = 1 + \left(\frac{c(n,2^{n-1})}{t_\nu(n,2^{n-1})} - \frac{p(n,2^{n-1})}{t_\nu(n,2^{n-1})}\right)\Bigg/\frac{}{t_\nu(n,2^{n-1})}$$

$$O\left(\frac{\nu}{\ln\nu}\right) . \qquad (13)$$

(d) Let $n = \lfloor (\ln \nu - \ln \ln \nu) 2^{2^\nu} \rfloor$. Then

$$2^{-2^\nu} = \frac{\ln \nu - \ln \ln \nu}{n + O(1)} = \alpha_n . \qquad (14)$$

We find, by (12) and (13),

$$\frac{c(n, 2^{n-1})}{t_\nu(n, 2^{n-1})} = 1 + O\left(\frac{\log \nu}{\nu}\right)$$

$$\frac{p(n, 2^{n-1})}{t_\nu(n, 2^{n-1})} = \frac{\ln \nu}{\nu} + \frac{\ln \nu}{2\nu} + O\left(\frac{\log \log \nu}{\nu}\right) ,$$

hence

$$\frac{c(n, 2^{n-1})}{t_\nu(n, 2^{n-1})} = \frac{2}{3} \frac{\nu}{\ln \nu} \left(1 + O\left(\frac{\log \log \nu}{\log \nu}\right)\right) . \qquad (15)$$

Problem 2.

Show that the following problem is NP-complete:

Input: A strongly connected graph G and an integer k .

Question: G have a minimum equivalent digraph containing k or fewer edges?

(a) The MED problem is in NP:

algorithm: (i) if $k \geq n$ answer yes. Otherwise,

(ii) guess a set of k edges in G ;

(iii) test whether this set of edges defines a strongly connected graph containing all vertices of G .

(b) . The directed Hamiltonian cycle problem (known to be NP-complete) is reducible to the MED problem:

A directed graph G with $n > 2$ vertices contains a Hamilton cycle iff the MED problem has answer "yes" for this graph with $k = n$.

Proof: (i) A Hamilton cycle defines a strongly connected spanning subgraph with $k = n$ edges.

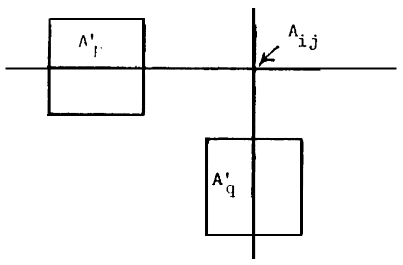
(ii) A strongly connected spanning subgraph contains at least one edge into and out of each vertex and thus contains at least n edges, if $n \geq 2$. A strongly connected spanning subgraph with n edges contains exactly one edge into and out of each vertex and thus consists of a set of cycles. To be strongly connected, such a subgraph must consist of exactly one cycle, which is a Hamilton cycle.

Problem

A transversal of A is a set of n non-zero entries, no two in the same row or the same column. Since A is non-singular, it must contain at least one transversal. Let $A'_1, \ldots, A'_k$ be the blocks of any decomposition.

(a) Thm: If A_{ij} is a non-zero element in some transversal, then row i and column j are in the same block A'_i .

Proof. Suppose the theorem is false. Pick a transversal containing a non-zero element which violates the theorem and let A_{ij} be the transversal element with minimum i violating the theorem. (I assume the numbering of rows and columns corresponds to a permutation defining the blocks $A'_1, \ldots, A'_k$.) Let row i be in block A'_p and column j be in block A'_q . Since $A_{ij} \neq 0$, $p < q$. The columns of A'_p contain exactly n elements of the transversal. Only n-1 of these can be inside block A'_p since each row in A'_p can contain only element of the transversal and the element in row i (A_{ij}) is outside the block A'_p . Thus some column of A'_p contains a transversal element outside A'_p . This transversal element cannot be below block A'_p since all elements below A'_p are zero, and it cannot be above A'_p since i was chosen to be minimum. This contradiction proves the theorem.



By the theorem, any transversal lies entirely within any set of blocks $A'_1,\ldots,A'_k$. By row and column permutations within the blocks $A'_1,\ldots,A'_k$, any such transversal can be placed on the main diagonal of A without affecting the block decomposition $A'_1,\ldots,A'_k$. It follows that if any transversal is permuted onto the main diagonal of A, all possible block decompositions are defined by subsequent simultaneous row and column permutations.

We can thus assume without loss of generality that the diagonal of A is non-zero and restrict our attention to simultaneous row and column permutations. Let G be the directed graph with vertex set $V = \{1,2,\ldots,n\}$ and edge set $E = \{(i,j) \mid i \neq j \text{ and } A_{ij} \neq 0\}$. The strongly connected components of G correspond to the blocks of a maximum-k block decomposition of G, and two rows whose vertices are in the same strongly connected component of G must be in the same block of any block decomposition. (The latter fact immediately implies the former.)

To prove this, suppose that $A'_1,\ldots,A'_k$ is any block decomposition. Since only zeros appear below the blocks, no edge in G leads from a vertex corresponding to a row in A'_p to a vertex corresponding to a row in A'_q, if $p > q$. It follows that no strong component of G can contain vertices corresponding to rows in two or more blocks of $A'_1,\ldots,A'_k$. Furthermore, if the strong components of G are sorted in topological order, they correspond to a block decomposition of A.

(b) The algorithm is:

(1) Find a transversal of A, using the bipartite matching algorithm of Hopcroft and Karp; put the transversal on the diagonal.

(a) Find the strong components of the graph G corresponding to the permutation of A, sort the components topologically, and determine the corresponding blocks.

Step (1) requires $O(n^{1/2} m)$ time if A is $n \times n$ and has m non-zeros. Step (2) requires $O(n+m)$ time. The total time required is $O(n^{1/2} m)$.

Problem 4.

Part 1. The usual sequential file will be considered as a 1-file. In an optimal 1-file, it is well known that the keys are arranged in the order of decreasing frequency. We shall denote by D(G) the average cost of an optimal 1-file G, then

$$D(G) = \sum_{1 \le m \le t} m p_{j_m} \qquad (1)$$

where $p_{j_1} \ge p_{j_2} \ge \ldots \ge p_{j_t}$ are the frequencies of keys in G.

A 2-file F is specified by a key K_d $(1 \le d \le n)$; the two subfiles L_d and R_d consist of keys less than K_d and greater than K_d, respectively. For F to be optimal, clearly L_d and R_d are optimal 1-files themselves; furthermore, in this case,

$$C_F = 1 + D(L_d) + D(R_d) . \qquad (2)$$

If we can compute the 2n numbers $D(L_d)$ and $D(R_d)$ for $1 \le d \le n$ in $O(n \log n)$ steps, then equation (2) enables us to compute C_F for n possible F in O(n) time. We then finally compute $A_2 = \min_F C_F$ in O(n) additional steps. This would solve the problem in $O(n \log n)$ steps. Below we give an algorithm computing $D(L_d)$ for $1 < d \le n$ in $O(n \log n)$ time; the computing of $D(R_d)$ is identical, and will not be repeated.

The obvious way to compute $D(L_d)$ is to sort L_d according to frequency and compute $D(L_d)$ by its definition. But this would take $O(n \log n)$ for each d, and $O(n^2 \log n)$ overall. To do better, we observe that L_d is obtained from L_{d+1} by deleting K_d and move all keys with lower frequencies (hence located "under" K_d) up one location. This leads to

$$D(L_d) = D(L_{d+1}) - p_d \cdot (1 + A(d)) - W(d) \qquad (3)$$

where A(d) is the number of j's with $(j < d) \wedge (p_j > p_d)$ and

$$W(d) = \sum_{\substack{j<d \\ p_j < p_d}} p_j .$$

The following algorithm first computes A(d), W(d) for $d = n, n-1, \ldots, 1$ successively, then uses (3) to compute all the $D(L_d)$.

Algorithm for Computing $D(L_d)$ $(1 \leq d \leq n)$.

[Initialization]

A. Sort $\{p_1, p_2, \ldots, p_{n-1}\}$ into decreasing order $p_{j_1} \geq p_{j_2} \geq \cdots \geq p_{j_{n-1}}$.

B. Build a balanced tree with all leaves in the lowest two levels (or one level if n-l is a power of 2). Number the leaves from left to right as j_1, j_2 [illegible]

C. Associate to each internal node v, two fields (a(v), w(v)) where a(v) is initialized to be the number of leaves descendent from v , and w(v) is Initialized as $\sum_j p_j$ with j summing over all descendent-leaves of v .

[An example of an initialized tree is shown in Figure 1.1

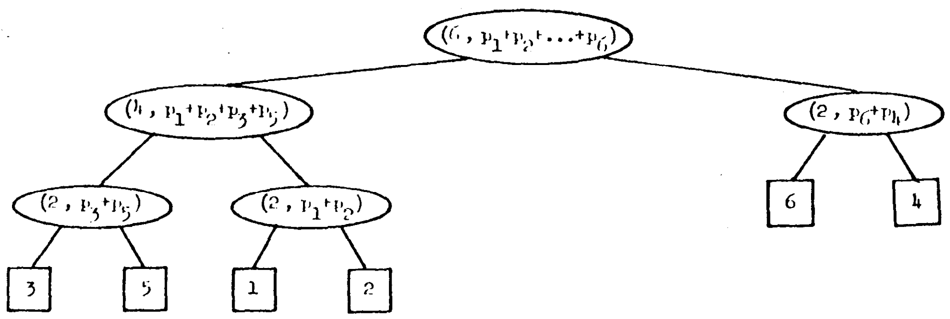


Figure 1. An Initialized tree with $n = 7$ and $p_3 \geq p_5 \geq p_1 \geq p_2 \geq p_6 \geq p_4$.

D. [Computing A(d) , W(d) for d = n-1 to 1 .]

```
for d := n-1 step -1 until 1 do
  [compute A(d) , W(d)]
    A(d) := 0; W(d) := 0;
    [let $v_k$ = leaf d)
    trace a path $v_k\ v_{k-1}\ v_{k-2}\ \cdots\ v_1$ from leaf d up to the root $v_1$;
      $A(d) := \sum_{\substack{j \\ v_{j+1} = \text{rightson}(v_j)}} a(\text{leftson}(v_j))$;
      $W(d) := \sum_{\substack{j \\ v_{j+1} = \text{leftson}(v_j)}} w(\text{rightson}(v_j))$;
      [update a(v) , w(v)]
      for each $v_j$ on the path,
         $a(v_j) := a(v_j)-1$;
         $w(v_j) := w(v_j)-p_d$;
```

E. Compute $D(L_n) = \sum_{1 \leq i \leq n-1} i\, p_{j_i}$.

F. Use(3)to compute $D(L_d)$ for d = n-1,n-2,...,1 successively.

In step D, when the loop parameter is d and A(d) , W(d) are being computed, the values of a(v), w(v) for any node v are given by

$$a(v) = |S(v)| \quad , \quad w(v) = \sum_{j \in S(v)} p_j$$

when $S(v) = \{j \mid \boxed{j}$ is a descendant leaf of v; $j < d\}$. It is not difficult to see that A(d) , W(d) are correctly computed. The updating operation keeps the interpretation of n(v) , w(v) valid for d-l . The rest of the program is obviously correct.

To count the cost: Initialization takes O(n log n) ; in step D, it takes O(k) = O(log n) steps for each d , hence O(n log n) is total time. Steps E and I" can bc done in O(n) time. Therefore, this algorithm computes $D(L_d)$ for $1 \leq d \leq n$ in O(n log n) .

Part 2.

This part is solved by dynamic programming. For any t-file, the keys to the left form a q-file (for some $1 \le q \le t-1$), and the keys to the right a $(t-q)$-file. The important observation is that for the t-file to be optimal, the associated q-file and $(t-q)$-file also have to be optimal. Let

$$s(t,i,j) = \begin{cases} \text{cost of optimal t-file for keys } \{K_i, K_{i+1}, \ldots, K_{i+j-1}\}; \\ (\infty, \text{ if no t-file exists or } i+j-1 > n). \end{cases}$$

The previous remark implies that, for $t > 2$,

$$s(t,i,j) = \min_{1 \le q \le t-1} \; \min_{1 \le j' \le j} (1 + s(q,i,j') + s(t-q, i+j', j-j')) \; . \qquad (4)$$

Therefore, if $s(q,i,j)$ are known for all $q < t$, and all i, j, then $s(t,i,j)$ can be computed in $O(t \cdot j) = O(tn)$ time for each i, j. The following program computes $s(t,i,j)$ for all $1 \le t \le \ell$, $1 \le i,j \le n$. In particular, it computes $A_\ell = s(\ell,1,n)$.

Algorithm for A_ℓ.

```
A.  [compute $s(1,i,j)$ $\forall i,j$]
    for each $i$ $(1 \le i \le n)$, compute $s(1,i,n), s(1,i,n-1), \ldots, s(1,i,i)$ by the method in part 1 for
       computing Ld $(d = n-1, n-2, \ldots, 1)$. (This is possible because 1-files are just sequential files.)
B.  for t := 2 until $\ell$ do
       for each $1 \le i,j \le n$, compute $s(t,i,j)$ using (3);
C.  $A_\ell := s(\ell,1,n)$;
```

cost analysis: cost for A $= n \cdot O(n \log n) = O(n^2 \log n)$

cost for B $= \sum_{2 \le t \le \ell} n^2 . O(tn) = O(\ell^2 n^3)$.

Therefore, the algorithm works in $O(n^3)$ time.

Part 3.

(a) We shall abbreviate $A_\ell(p_1^{(n)}, p_2^{(n)}, \ldots, p_n^{(n)})$ as $A_\ell(n)$. The idea is: With the present frequency distribution, the cost of an ℓ-file is dominated by the ℓ subfiles regarded as sequential files; thus, we can concentrate on, instead of $A_\ell(n)$, the cost of sequential files which have simpler analytic expressions. Formally, let us define

$$g_\ell(n\,; i_1, i_2, \ldots, i_{\ell-1}) = \left(\frac{1}{1} + \frac{1}{2} + \ldots + \frac{i_1}{i_1}\right) + \left(\frac{1}{i_1+1} + \frac{2}{i_1+2} + \ldots + \frac{i_2-i_1}{i_2}\right)$$

$$+ \ldots + \left(\frac{1}{i_{\ell-1}+1} + \frac{2}{i_{\ell-1}+2} + \ldots + \frac{n-i_{\ell-1}}{n}\right) \qquad (5)$$

Expression (5) is essentially H_n times the total cost of the ℓ subfiles formed by breaking the set of keys at positions $i_1, i_2, \ldots, i_{\ell-1}$. Let

$$f_\ell(n) = \min_{1 \le i_1 < i_2 < \ldots < i_{\ell-1} \le n} g_\ell(n; i_1, i_2, \ldots, i_{\ell-1}) \; .$$

The following lemma shows that we can study $\frac{1}{H_n} f_\ell(n)$ instead of $A_\ell(n)$.

Lemma 1. $|A_\ell(n) - \frac{1}{H_n} f_\ell(n)| \le \ell$.

Proof.

(A) Consider an optimal ℓ-file using $\{K_{i_1} < K_{i_2} < \ldots < K_{i_{\ell-1}}\}$ as the set of keys used in the internal nodes. The contribution to the cost from keys in sequential subfiles is at least

$$\frac{1}{H_n}\left(g_\ell(n\,; i_1, i_2, \ldots, i_{\ell-1}) - \frac{1}{i_1} - \frac{i_2-i_1}{i_2} - \ldots - \frac{n-i_{\ell-1}}{n}\right) \ge \frac{1}{H_n}(g_\ell(n\,; i_1, \ldots, i_{\ell-1}) - \ell) \; .$$

Therefore

$$A_\ell(n) \ge \frac{1}{H_n}(f_\ell(n) - \ell) \; . \qquad (6)$$

(B) Let $i_1 < i_2 < \dots < i_{\ell-1}$ be such that $f_\ell(n) = g_\ell(n\ ; i_1, i_2, \dots, i_{\ell-1})$. Choose any ℓ-file with $\{K_{i_1}, K_{i_2}, \dots, K_{i_{\ell-1}}\}$ as keys in internal nodes, and sort the subfiles according to decreasing frequency. As the internal keys and the first record in each sequential subfile is at most ℓ-distance away from the root, we have an ℓ-file with cost less than $\frac{1}{H_n} \cdot f_\ell(n) + \ell$. Therefore,

$$\frac{1}{H_n} \cdot f_\ell(n) + \ell \geq A_\ell(n) . \tag{7}$$

Formulas (6) and (7) imply the Lemma. ☐

<u>Lemma 2.</u> For each fixed ℓ , there exists a constant a_ℓ such that $\lim_{n\to\infty} \frac{f_\ell(n)}{n} = a_\ell$.

<u>Proof.</u> Define $f_1(n) = \frac{1}{1} + \frac{2}{2} + \dots + \frac{n}{n} = n$. We prove the lemma by induction. The <u>induction hypothesis</u> is: For each $\ell \geq 1$, there exists a constant $0 < a_\ell < 1$ such that

$$f_\ell(n) = a_\ell n + O(1) . \tag{8}$$

The $O(1)$ term in (8) may depend on ℓ .

The induction hypothesis is obviously true for $\ell = 1$. Now suppose we have proved it for $\ell-1$, we shall prove it for ℓ . We need the following fact :

Fact Let $h(k) = n + (a_{\ell-1}-1)k - k \ln \frac{n}{k}$ $(0 < a_{\ell-1} < 1)$.

Then $h(k) \geq (1-e^{-a_{\ell-1}})n$ for $n \geq k \geq 1$, and

$$h(\lceil n e^{-a_{\ell-1}} \rceil) = (1-e^{-a_{\ell-1}})n + O(1) .$$

(A) Proof that $f_\ell(n) \geq (1-e^{-a_{\ell-1}})n + O(1)$: Let $i_1, i_2, \dots, i_{\ell-2}, k$ be such that $f_\ell(n) = g_\ell(n\ ; i_1, i_2, \dots, i_{\ell-2}, k)$. From the definition of g_ℓ ,

$$f_\ell(n) = g_{\ell-1}(k\ ; i_1, i_2, \dots, i_{\ell-2}) + \frac{1}{k+1} + \frac{2}{k+2} + \dots + \frac{n-k}{n} .$$

Since $f_{\ell-1}(k) = \min\limits_{i_1, i_2, \dots} g_{\ell-1}$, we must have $f_\ell(n) = f_{\ell-1}(k) + \frac{1}{k+1} + \frac{2}{k+2} + \dots + \frac{n-k}{n}$. Using the induction hypothesis for $\ell-1$, we obtain

$$\begin{aligned} f_\ell(n) &\geq a_{\ell-1}k + O(1) + \sum_{k<j\leq n} \frac{j-k}{j} \\ &= a_{\ell-1}k + n - k - k \ln \frac{n}{k} + O(1) \\ &\geq (1-e^{-a_{\ell-1}})n + O(1) \qquad \text{by } \underline{\text{fact}}. \end{aligned}$$

(B) Proof that $f_\ell(n) \leq (1-e^{-a_{\ell-1}})n + O(1)$: We need only prove it for all sufficiently large n . Suppose n satisfies the condition $k \equiv \lceil n e^{-a_{\ell-1}} \rceil \geq \ell-1$. We choose $i_1, i_2, \dots, i_{\ell-2}$ such that $g_{\ell-1}(k\ ; i_1, i_2, \dots, i_{\ell-2}) = f_{\ell-1}(k)$. Then

$$\begin{aligned} f_\ell(n) &\leq g_\ell(n\ ; i_1, i_2, \dots, i_{\ell-2}, k) \leq f_{\ell-1}(k) + \frac{1}{k+1} + \frac{2}{k+2} + \dots + \frac{n-k}{n} \\ &\leq a_{\ell-1}k + (n-k) - k \ln \frac{n}{k} + O(1) . \end{aligned}$$

By fact, this is less than $(1-e^{-a_{\ell-1}})n + O(1)$.

Let $a_\ell = 1-e^{-a_{\ell-1}}$, clearly $0 < a_\ell < 1$. The induction hypothesis for ℓ is now established This completes the proof for Lemma 2. ☐

Now, Lemma 1 implies

$$\left| \frac{(\ln n)A_\ell(n)}{n} - \frac{\ln n}{H_n} \frac{f_\ell(n)}{n} \right| \leq \frac{\ell}{n} \ln n \tag{9}$$

Since $\lim_{n\to\infty} \frac{f_\ell(n)}{n} = a_\ell$ (Lemma 2), $\lim_{n\to\infty} \frac{\ln n}{H_n} = 1$, and $\lim_{n\to\infty} \frac{\ell}{n}\ln n = 0$, equation (9) implies that

$$\lim_{n\to\infty} \frac{A_\ell(n)}{n/\ln n} = \lim_{n\to\infty} \frac{f_\ell(n)}{n} = a_\ell .$$

(b) In (a), we have actually proved

$$\begin{cases} a_\ell = 1-e^{-a_{\ell-1}}, & \ell \geq 2 \\ a_1 = 1 . \end{cases}$$

Thus,

$$a_2 = 1-e^{-1},$$

$$a_3 = 1-e^{e^{-1}-1},$$

$$a_4 = 1-e^{e^{e^{-1}-1}-1} .$$

Problem 5.

Let $X = \{x_1 > x_2 > \dots > x_m\}$ and $Y = \{y_1 > y_2 > \dots > y_n\}$. The set of relations R between X and Y can be written as $R = \{x_{i_1} > y_{j_1}, x_{i_2} > y_{j_2}, \dots, x_{i_r} > y_{j_r}\}$ where $i_1 < i_2 < \dots < i_r$ and $j_1 < j_2 < \dots < j_r$. Let $p(R)$ denote the probability that $x_i > y_j$ under R. If we can show that

$$p(R) \geq p(R') \tag{1}$$

where $R' = R - \{x_{i_1} > y_{j_1}\}$, then the problem is solved by induction.

Let us use A_k to denote the number of permutations of $X \cup Y$ consistent with $R' \cup \{y_{k-1} > x_{i_1} > y_k\}$, and a_k the probability that $x_i > y_j$ knowing $R' \cup \{y_{k-1} > x_{i_1} > y_k\}$. Then

$$p(R) = \frac{\sum_{1\leq k\leq j_1} a_k A_k}{\sum_{1\leq k\leq j_1} A_k}, \tag{2}$$

and

$$p(R') = \frac{\sum_{1\leq k\leq j_2} a_k A_k}{\sum_{1\leq k\leq j_2} A_k} . \tag{3}$$

Equations (2) and (3) will imply (1) if we can show the following lemma.

Lemma 1. $a_1 \geq a_2 \geq \dots \geq a_{n+1}$.

Indeed, (1) follows from Equations (2), (3), Lemma 1, and the following simple algebraic fact:

Monotonicity Lemma. If $E_1, E_2, \dots, E_u$ are positive numbers, and $e_1 \geq e_2 \geq \dots \geq e_u$, then the function

$$f(t) = \frac{\sum_{1\leq i\leq t} e_i E_i}{\sum_{1\leq i\leq t} E_i}$$

is non-increasing in t for $1 \leq t \leq u$.

To prove Lemma 1, we first show a related lemma. Consider partial [illegible] on $X = \{x_1 > x_2 > \dots > x_m\}$ and $Y_k = \{y_k > y_{k+1} > \dots > y_n\}$ with relations $R = \{x_{i_1} > y_{j_1}, \dots, x_{i_r} > y_{j_r}\}$. (Figure 1.) Let b_k be the probability that $x_i > y_j$ in Q_k, and b_{k-1} the probability that $x_i > y_j$ in Q_{k-1} (i.e., the partial order with the same X and R, but an additional element y_{k-1} dominating y_k).

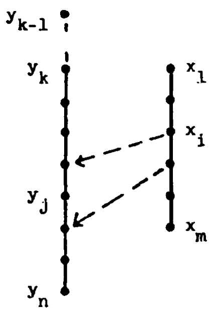


Figure 1

Lemma 2. $b_{k-1} \geq b_k$.

We prove this lemma by induction on m . For m = 1 , we have $x_i = x_1$. Let $\{x_1 > y_t\}$ be the relation R (t = n+1 if R = $\emptyset$) . From Figure 2, we see that either $b_k = 1$ (if $t \leq j$) or $b_k = \frac{j-k+1}{t-k+1}$ (if t > J) . In either case the lemma is true.

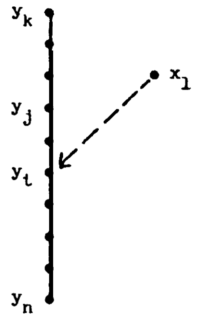


Figure 2

Induction step: Suppose the claim is true for m-1 , we will prove it for m . Let

C_ℓ = the number of permutations of $X \cup Y_k$ consistent with $Q_k \wedge (y_{\ell-1} > x_1 > y_\ell)$,

and

c_ℓ = the probability that $x_i > y_j$ under $Q_k \wedge (y_{\ell-1} > x_1 > y_\ell)$.

(See Figure 3 . .

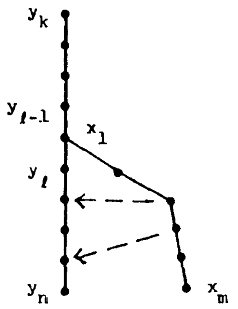


Figure 3

Thrn

$$b_k = \frac{\sum\limits_{k \leq \ell \leq j_1} c_\ell C_\ell}{\sum\limits_{k \leq \ell \leq j_1} C_\ell} .$$

Similarly, let c'_ℓ and C'_ℓ be the corresponding quantities for Q_{k-1} , then

$$b_{k-1} = \frac{\sum_{k-1 \le \ell \le j_1} C'_\ell c'_\ell}{\sum_{k-1 \le \ell \le j_1} \quad} \quad .$$

Clearly, $C_\ell = C'_\ell$ and $c_\ell = c'_\ell$ for $k \le \ell \le j_1$. Therefore

$$b_k = \frac{\sum_{k \le \ell \le j_1} c'_\ell C'_\ell}{\sum_{k \le \ell \le j_1} C'_\ell} \quad .$$

Now, consider two cases. Case 1) $x_i = x_1$. Then $c_\ell = 1$ for $\ell \le j$ and $c_\ell = 0$ for $\ell > j$. So c_ℓ is non-increasing. Case 2) $x_i \ne x_1$. Then c_ℓ for $\ell \le j$ is equal to the probability that $x_i > y_j$ in the partial order of Figure 4, hence non-increasing by induction hypothesis. Moreover $c_\ell = 0$ when $\ell > j$. Thus, in both cases, $b_{k-1} \ge b_k$ by the Monotonicity Lemma. This completes the induction step.

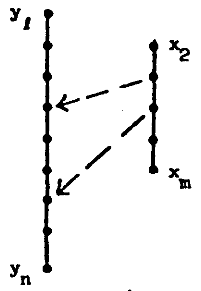


Figure 4

Proof of Lemma 1.

Case 1) $i > i_1$. For $k > j$, we have $a_k = 0$. For $1 \le k \le j$, a_k is the same as the probability for $x_i > y_j$ under the partial order of Figure 5. By Lemma 2, a_k is non-increasing.

Case 2) $i = i_1$. We have $a_k = 1$ if $k \le j$, and $a_k = 0$ if $k > j$.

Case 3) $i < i_1$. If $j_1 \le j$, then $a_k = 1$ for all k , and the lemma is true. We consider the case $j_1 > j$. Now, $a_k = 1$ for $k \le j$. For $k > j$, a_k is equal to the probability that $x_i > y_j$ under the partial order of Figure 6. By Lemma 2, a_k is non-increasing.

This completes the proof of Lemma 1, and hence equation (1).

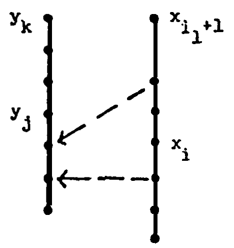


Figure 5

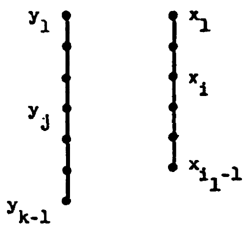


Figure 6

ANALYSIS OF ALGORITHMS - SOLUTIONS - Spring 1978

Problem 1.

(a) Consider the set of n nodes $T = \{\emptyset, 0, 01, 010, 0101, 01010, \ldots\}$. Then half of them are hashed into location 0, and half into location 1. Thus, for this T, $c(h_0,T) = \frac{1}{n}\left(\frac{n}{2}\lg\left(\frac{n}{2}+1\right) + \frac{n}{2}\lg\left(\frac{n}{2}+1\right)\right) = \lg n + O(1)$. This proves $f(h_0) \geq \lg n + O(1)$.

On the other hand, from the definition of $c(h,T)$, we have for any T,

$$c(h_0,T) \leq \lg(n+1)\ \frac{1}{n}\sum n_j = \lg(n+1)\ .$$

Thus, $f(h_0) \leq \lg n + O(1)$.

We have proved $f(h_0) = \lg n + O(1)$.

(b) Let $m = n+1$. We shall prove the following theorem.

Theorem. Let $\epsilon > 0$ be any fixed constant. Then a random hashing scheme $h \in H(n,n)$ will, with probability $1 - \text{const.} \times \frac{1}{2^{\epsilon n}}$, satisfy

$$f(h) \leq \left(1 + \frac{m}{n}\right)\lg\left(1 + \frac{n}{m}\right) + \frac{1}{n}\lg|\mathcal{T}_n| + \epsilon\ .$$

As $|\mathcal{T}_n| = \frac{1}{n+1}\binom{2n}{n} \leq 4^n$ (Knuth, 2.3.4.4), we have for a random h, $f(h) \leq 6 + \epsilon + O(1/n)$. This would solve our problem.

We start the proof of the theorem with two simple lemmas.

Lemma 1. For any positive integer x, $x! > e^{-x}(x+1)^x$.

Proof. We prove by induction. For $x = 1$, the lemma is true as $1 > e^{-1}\cdot 2$. For $x > 1$, we have by induction hypothesis $(x-1)! > e^{-(x-1)}x^{x-1}$. This implies $x! > e^{-x+1}x^x = e\left(1 + \frac{1}{x}\right)^{-x}e^{-x}(x+1)^x > e^{-x}(x+1)^x$, where we have used $e > \left(1 + \frac{1}{x}\right)^x$. This completes the induction. □

Lemma 2. Let $t(m,n)$ be the number of integer solutions $(n_1,n_2,\ldots,n_m)$ to the equation $\sum_{1\leq i\leq m} n_i = n$ with $n_i \geq 0$. Then $t(m,n) = \binom{m+n-1}{n}$.

Proof. $t(m,n)$ is the coefficient of x^n in the expansion of $(1+x+x^2+\cdots)^m = (1-x)^{-m}$. Therefore $t(m,n) = (-1)^n\binom{-m}{n} = \binom{m+n-1}{n}$ [Knuth 1.2.6.(17)]. □

Now, for any $T \in \mathcal{T}_n$ and any positive a, let $\lambda(a,T)$ be the proportion of $h \in H(m-1,n)$ such that $c(h,T) > a$. We are going to show that, abbreviating m/n by b,

$$\lambda(a,\ T) \leq \text{constant} \times \left(\frac{1}{2^a}\left(1 + \frac{1}{b}\right)^{b+1}\right)^n\ . \qquad (1)$$

Let us first show that (1) implies the theorem. We use $\lambda(a,\mathcal{T}_n)$ to denote the proportion of h in $H(m-1,\ n)$ such that $f(h) > a$. Then $\lambda(a,\mathcal{T}_n) \leq \sum_{T\in\mathcal{T}_n}\lambda(a,T)$. By (1), we have

$$\lambda(a,\mathcal{T}_n) \leq \text{constant} \times |\mathcal{T}_n|\cdot\left(\frac{1}{2^a}\left(1 + \frac{1}{b}\right)^{b+1}\right)^n \qquad (2)$$

When $a = (1+b)\lg\left(1 + \frac{1}{b}\right) + \frac{1}{n}\lg|\mathcal{T}_n| + \epsilon$, (2) implies

$$\lambda(a,\mathcal{T}_n) \leq \text{constant} \times \frac{1}{2^{\epsilon n}}\ .$$

But this is exactly the theorem!

It remains to prove (1). Let $g(n_1,n_2,\ldots,n_m)$ be the proportion of $h \in H(m-1,n)$ such that $|\{x \mid x\in T,\ h(x) = i\}| = n_i$ $(1 \leq i \leq m)$. Clearly, when $\sum n_i = n$,

$$g(n_1,n_2,\ldots,n_m) = \frac{1}{m^n}\ \frac{n!}{n_1!\,n_2!\ldots n_m!}\ . \qquad (3)$$

Let $\Lambda(a)$ be the set of integer m -tuples $(n_1,n_2,\ldots,n_m)$ satisfying

$$\begin{cases} n_i \geq 0 \ , \ i = 1,2,\ldots,m \ , \\ \sum\limits_{1\leq i\leq m} n_i = n \\ \sum\limits_{1\leq i\leq m} n_i \lg (n_i+1) > a\,n \ . \end{cases} \tag{4}$$

Then

$$\lambda(a,T) = \sum_{(n_1,n_2,\ldots,n_m)\in\Lambda(a)} g(n_1,n_2, \ldots \text{[illegible]} \tag{5}$$

<u>Lemma</u> If $(n_1,n_2,\ldots,n_m)\in\Lambda(a)$, then $g(n_1,n_2,\ldots,n_m) \leq \frac{n!}{m^n} e^n 2^{-an}$.

<u>Proof.</u> Formula (4) implies $\prod\limits_i (n_i+1)^{n_i} > 2^{an}$. This means, from Lemma 1,

$$\prod_i (n_i!) > e^{-\sum\limits_i n_i} \prod_i (n_i+1)^{n_i} > e^{-n}2^{an} \ .$$

Thus, from (3),

$$g(n_1,n_2,\ldots,n_m) \leq \frac{n!}{m^n} e^n 2^{-an} \ . \ \square$$

Now, from (5) and Lemma 3,

$$\lambda(a,T) \leq \frac{n!}{m^n} e^n 2^{-an} |\Lambda(a)| \ . \tag{6}$$

But $|\Lambda(a)| \leq t(m,n)$ from definitions. Using Lemma 2 and (6), we have

$$\begin{aligned} \lambda(a,T) &\leq \frac{n!}{m^n} e^n 2^{-an} \binom{m+n-1}{n} \\ &\leq \text{constant}\sqrt{\frac{m}{m+n}}\ \frac{1}{(2^a)^n} \left(1+\frac{n}{m}\right)^{m+n} \\ &\leq \text{constant}\left(\frac{1}{2^a}\left(1+\frac{1}{b}\right)^{b+1}\right)^n \ . \end{aligned}$$

This proves (1). [The "constant" is an absolute constant.]

We have completed the proof of the theorem.

Problem 3.

To solve this problem, we need a data structure with the following properties:

(i) entries consist of a value and a size;

(ii) the entries are sorted by value and can be accessed by value easily;

(iii) it is easy to insert or delete an entry;

(iv) it is easy to locate the smallest value of a given size or larger.

Various kinds of balanced trees can be adapted for such a data structure; we use 2-3 trees (see Aho, Hopcroft and Ullman, pp.146ff.). Two examples appear in Figure 4.26. Entries are stored in order by value in the external nodes. Each internal node contains the largest value in its left subtree and the largest value in its middle subtree (if it is a 3-node). In addition, each internal node contains the largest size in its subtree.

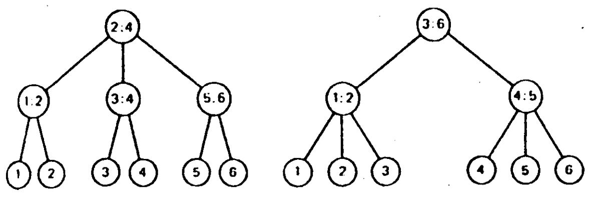


Fig. 4.26 2-3 trees.

Insertions and deletions work exactly as described in AHU except that we must update the size information on internal nodes along the path from the root to the inserted or deleted node. The time for either an insertion or a deletion is O(log n) , where n is the total number of values.

To carry out a locate given a size, say s , we search down from the root, always taking the leftmost branch which leads to a node having size s or larger. Eventually we end up at the leftmost internal node with size s or larger. This operation also takes O(log n) time.

(a) Our data structure consists of the one described above, with one entry corresponding to each free block of memory. The value of the entry is the starting point of the block and the size of the entry is the size of the block. Initially the structure contains one entry corresponding to the entire memory.

To schedule a task of length s , we locate the leftmost entry with size at least s . We delete this entry from the data structure, and insert a new entry corresponding to what is left of the memory block if the new task does not entirely fill it. Thus scheduling one task requires a location, a deletion, and possibly an insertion, for a total of O(log n) time.

To free the memory corresponding to a task, we look up the free blocks immediately to its left and right. We delete the entries corresponding to these blocks and add one, two, or three new entries depending upon whether the freed block is contiguous with the left block or right block or both. This requires two look-ups, two deletions, and at most three insertions, for a total of O(log n) time.

(b) To solve this part we use in addition to the data structure used in part (a), a similar data structure to represent the set of waiting tasks. In the second structure each entry is a task, whose value is the time it first became available for scheduling and whose size is the amount of memory it requires.

We schedule a task as before, except if it does not fit immediately, we insert it into the waiting list. This takes O(log n) time. We free memory as before, except after freeing a block of memory and updating the data structure representing the free blocks, we check to see if any task in the waiting list will fit into the new memory block created. If so, we choose the task which arrived earliest and schedule it. We repeat until no more tasks on the waiting list can be scheduled. This operation requires one location and one deletion in the waiting list, and one scheduling operation in the data structure for memory, for each task scheduled, and thus requires O(log n) time.

Note : To get the time bounds right, we must reinterpret n as the total number of jobs currently in memory and in the waiting list; this point was not spelled out in the problem description.

Problem 4.

(a) To show that the problem is NP-complete we must first phrase it as a yes-no problem, so let us assume that a height is given and we ask "Can the rectangles be packed within height h ?" To show that the problem is in NP, suppose the dimensions of all the rectangles, the width of the bin, and h , are integers. It is easy to guess a packing (by specifying, say, the coordinates of the lower left corner of each block and to test whether it works, in time polynomial in the number of digits needed to write down all the numbers (Note that we need only consider integer co-ordinates.)

The problem is in NP even if we allow rational numbers, since we can multiply all the fractions by the least common multiple of their denominators and thus convert everything to integers, while only getting a polynomial blow-up in the total number of digits.

(If we allow arbitrary real numbers, then the problem isn't in NP, since we have no reasonable way of representing arbitrary real numbers.)

To show that the problem is NP-complete, we reduce the knapsack problem to it. One version of the knapsack problem known to be NP - complete is the following. Given k integers, $l_1,\ldots,l_k$, with sum S is there a subset with sum S/2 ? We construct a corresponding packing problem with bin width 2 and k blocks, the i-th block of height l_i and width 1 . Then the knapsack problem has a solution if and only if there is a packing of height S/2 .

Although the knapsack problem is solvable in polynomial time if the numbers are small, the bin-packing problem is NP-complete in the strong sense; the three-partition problem can be reduced to bin-packing (see Garey and Johnson's manuscript on NP-completeness for a definition of strong NP-completeness and the three-partition problem).

(b) The basic BL (bottom-up, left-justified) algorithm, using a poorly ordered list L , can perform arbitrarily badly relative to an optimization algorithm. A simple example illustrating this fact is shown in Figure 1. The rectangles in the list $L = (p_1,p_2,\ldots,p_n)$ alternate between vertical and horizontal slabs. In particular, let $p_i = (x_i,y_i)$, where

$$(x_i,y_i) = \begin{cases} (\varepsilon, 1+i\varepsilon) & \text{if } i \text{ odd} \\ (w,\varepsilon) & \text{if } i \text{ even.} \end{cases}$$

The height of the BL packing is $\lceil n/2 \rceil + O(n\varepsilon)$ whereas an optimum packing can easily be seen to have a height of $1+O(n\varepsilon)$. Thus, the ratio of packing heights can be made as large as desired.

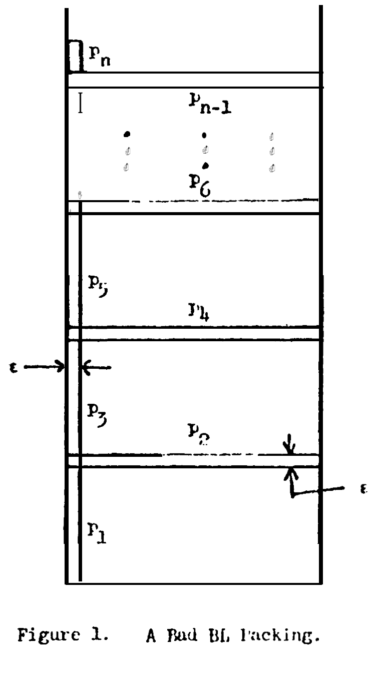


Figure 1. A Bad BL Packing.

(c)

Theorem 1. Let L be a list of rectangles ordered by decreasing widths. Then

$$\frac{h_{BL}}{h_{OPT}} \leq 3 \qquad (1)$$

This bound is best possible.

Proof: Let h* denote the height of the lower edge of a tallest piece whose upper edge is at height h_{BL}. If y denotes the height of this piece, then $h_{BL} = y + h^*$. Figure 2 shows an example. Let A denote the region of the bin up to height h*.

Suppose we can show that A is at least half occupied. Then we have $h_{OPT} \geq \max\{y, h^*/2\}$ and hence, $y > \frac{h^*}{2}$ implies

$$\frac{h_{BL}}{h_{OPT}} < \frac{y+h^*}{y} < \frac{y+2y}{y} = 3$$

and if $y \leq h^*/2$, we have

$$\frac{h_{BL}}{h_{OPT}} \leq \frac{h^*/2 + h^*}{h^*/2} = 3$$

The result will thus be proved. It remains to show that A is at least half occupied.

Any horizontal cut or line through A can be partitioned into alternating segments corresponding to cuts through unoccupied and occupied areas of the BL packing. We shall show that the sum of the occupied segments is at least the sum of the unoccupied segments. For convenience we may restrict ourselves to lines which do not coincide with the (upper or lower) edges of any piece. Since the set of such lines is of measure zero, ignoring them will not influence our claim that A is at least half occupied.

Initially, consider the partition of a given line just prior to when the first rectangle, say q, is assigned with a lower edge at a height exceeding the height, h, of the line. The piece q need not be in A; its existence is guaranteed by the fact that there is a piece packed above A. We claim that at that point in the assignment sequence the line is "half occupied".

First, bottom-up packing implies that all lines must cut through at least one piece. Second, all lines must cut through a piece abutting the left bin edge. For suppose not; then the left-most piece, say q', cut by the line must abut another piece, say q", to the left and entirely below the line. Thus, the length, x, of the unoccupied, initial segment of the line must be at least the width of q". But since q" was packed prior to q, the width of q must be less than that of q" and hence less than x. Since at the point in time we are considering, no piece has been assigned entirely above the line, the space vertically above the initial segment must be completely unoccupied. Thus, we have the contradiction that q would have fit into the space above q" in such a way that its lower edge is at a height less than h.

Now consider any segment, S, of the line which cuts through an unoccupied space. Let p be the piece bordering S on the left. Since when q is assigned, it is placed above the line, q must be wider than the length of S. (Once again, at the tine q is assigned its height could not prevent its placement in a sufficiently wide unoccupied space cut by the line.) But q is packed later than p; consequently, p is at least as wide as q. It follows that for each segment representing unoccupied space along the line there is a longer segment representing occupied space immediately to its left. Clearly, for any given line, the sum of the segment lengths corresponding to unoccupied space must be monotonically non-increasing as the packing sequence progresses. Therefore, the line continues to be at least half occupied. Finally, "integration" over the height of A verifies that A is at least half full. □

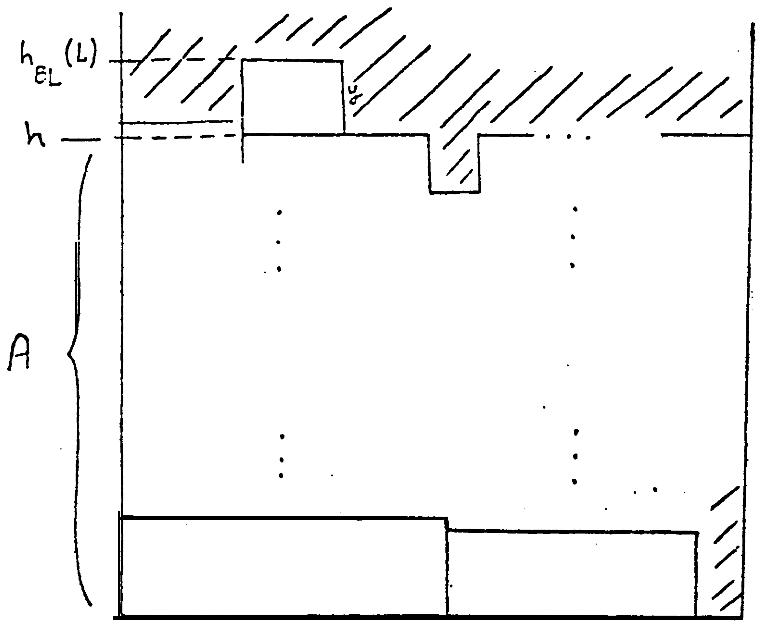


Figure 2. Defining the area A.

(d)

<u>Theorem 2</u> For any $\delta > 0$ there exists a list L of rectangles ordered by decreasing width such that the BL packing gives a height h_{BL} for which

$$\frac{h_{BL}}{h_{OPT}} > 3 - \delta \qquad (1)$$

If the pieces are restricted to squares then an L can be found such that for any $\delta > 0$

$$\frac{h_{BL}}{h_{OPT}} > 2 - \delta \qquad (2)$$

<u>Proof:</u> We shall prove the second result first, since the first result is but a slight modification.

The list proving (2) corresponds to the "checkerboard" packing in Fig. 3. The pieces are all either unit squares or approximately 2×2 squares. In particular, the larger squares are disposed on the bottom of the bin with the dimensions stepping down by ε from piece to piece. Hence the assignment of pieces in the second row must be made from right to left according to the bottom-up rule. Since two unit squares exceed the dimension of any larger square and squares are left-justified, only one unit square is placed on each large square. Except for the first and last pieces, this type of assignment repeats on the second row since the "holes" in the second row all have width less than 1 and squares to the right are lower than squares to the left. In general, the i^{th} row of unit squares alternates holes and pieces except for at most the initial and final i - 1 pieces of the row. Note that an optimum packing can be found which, except for possibly the last row, is within $O(\varepsilon)$ of being fully occupied.

The edge effects inhibiting the waste of half the space in the BL packing consist of

1. The row of larger pieces on the bottom
2. The triangular shaped solidly packed collections of squares on the left and right of the packing.

Holding piece sizes constant, the influence of the first edge effect is reduced by increasing the height of the packing, while the second is attenuated by widening the bin. Let k be two greater than the number of rows of unit squares. If the width of the bin is selected to be k^2, then the area of the bottom row and side edge effects is $O(k^2)$. Thus, ignoring $O(\varepsilon)$ terms, we can find a list such that

$$\frac{h_{BL}}{h_{OPT}} = \frac{k^3}{k^3/2 + O(k^2)}$$

In the limit $k \to \infty$, we have the bound of 2.

For the case of rectangles it is only necessary to augment the list for Fig. 3 by adding as a new, last piece a rectangle of unit width and a height which equals the height of the optimum packing corresponding to the new list. Omitting the details, the BL packing will correspond to Fig. 3 with the new piece placed on top. It is easy to verify that the height ratio can now be made to approach 3 as closely as desired. □

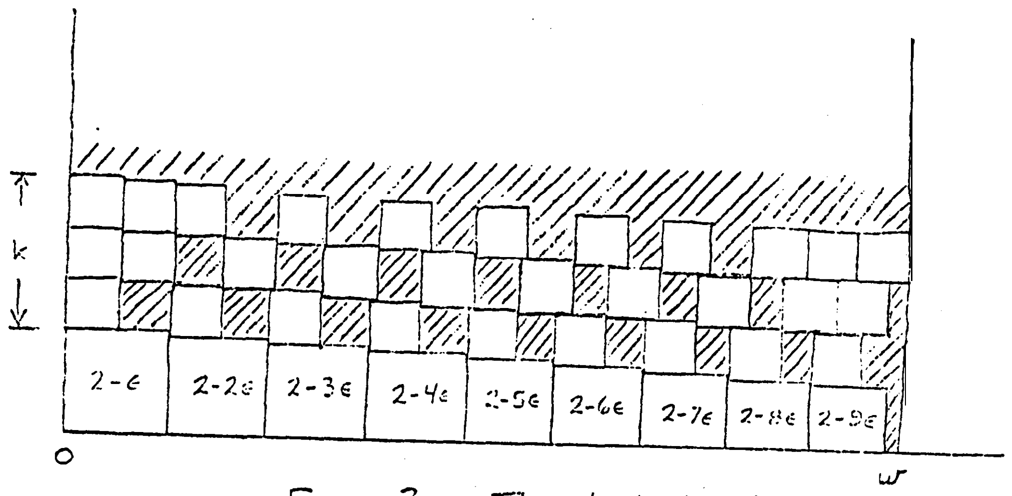


Figure 3 - The checkerboard example

Problem 5.

Without loss of generality, let the distribution have mean 0 and variance 1 . The density function then $n(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, with distribution function $N(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$. (Feller, pg. 174.) For large x , the approximation (ibid)

$$n(x)\left(\frac{1}{x} - \frac{1}{x^3}\right) < 1 - N(x) < \frac{n(x)}{x}$$

is useful. Let P (not N) be the number of processors.

A running time x has chance $N(x)^P$ of being $\geq$ the maximum. This is negligible if $N(x)^P \ll 1$, i. if $\left(1 - \frac{n(x)}{x}\right)^P \ll 1$, or $P\,n(x)/x \gg 1$. We can find where it becomes negligible by roughly equating $P\,n(x)/x$ to 1 ;

$$x = \frac{P e^{-x^2/2}}{\sqrt{2\pi}} \quad , \quad \text{or} \quad \frac{x^2}{2} = \ln \frac{P}{x\sqrt{2\pi}} \quad ;$$

$$x = \sqrt{2(\ln P - \ln x - \tfrac{1}{2}(\ln 2 + \ln\pi))}$$

$$x = \sqrt{2 \ln P - 2 \ln x - \ln 2 - \ln \pi}$$

$$\approx \sqrt{2 \ln P - 2 \ln \sqrt{2 \ln P} - \ln 2 - \ln \pi}$$

$$= \sqrt{2 \ln P - \ln 2 - \ln \ln P - \ln 2 - \ln\pi}$$

$$= \sqrt{2 \ln P - \ln \ln P - \ln (4\pi)} \quad .$$

Since the density of running times falls off faster than exponentially for larger values of x , and the likelihood of a running time being the maximum falls off exponentially with smaller values of x , the maximum must with high probability be very near $\sqrt{2 \ln P - \ln \ln P - \text{constant}}$.

We can bound the standard deviation by considering the rates at which the densities fall off on either of the estimated mean. The density of running times drops at least by a factor of e if we increase by provided that $(x+\Delta x)^2 - x^2 = 1$; $\Delta x^2 + 2x\Delta x = 1$; $\Delta x \approx \frac{1}{2x} \approx \frac{1}{\sqrt{8 \ln P}}$. In the other direction, the likelihood of a running time being the maximum drops by at least a factor of e if we decrease $P\,n(x)/x$ by 1 ; that is, if we increase $n(x)$ by a factor of about 2 . This gives rise to a Δx of the same order $O\left(\frac{1}{\sqrt{\ln P}}\right)$.

Together, this suggests:

$$\text{mean} = \sqrt{2 \ln P + o(\ln P)} \quad ,$$

$$\text{variance} = O\left(\frac{1}{\ln P}\right) \quad .$$

A more rigorous proof can probably be extracted from pp. 202 - 208 of Feller. A quick and dirty one is obtained by saying that the likely range of the maximum is around $N(x) = 1/P$, and allowing $N(x)$ to vary by a factor of 2 or so.

Another approach to the variance: the mean of the max for N numbers is about $\sqrt{2 \ln n}$. Now take the max of two such sets, S_1 and S_2 , of N numbers, the expected value is about

$$\sqrt{2 \ln(2n)} = \sqrt{2 \ln n + 2 \ln 2} \approx \sqrt{2 \ln n} + 2 \ln 2 \cdot \frac{1}{2\sqrt{2 \ln n}} = \sqrt{2 \ln n} + \frac{\ln 2}{\sqrt{2}} \frac{1}{\sqrt{\ln n}}$$

One can make a plausible argument that the extra $O\left(\frac{1}{\sqrt{\ln n}}\right)$ term is linearly related to the variance.

Problem 6.

The number of binary trees with N internal, and N+1 external, nodes, is found by writing names for the trees in Polish prefix notation: the symbol I must appear N times, E must appear N+1 , and the cumulative number of I 's must exceed the number of E 's until the end. The latter condition is true of exactly one cyclic permutation of any string of N E 's and N+1 I 's. So the total number of trees is

$$\binom{2N+1}{N}/(2N+1) = \frac{(2N)!}{N!(N+1)!} .$$

The information intent of the structure is the base 2 log , which by Stirling's approximation is about 2N + constant, or 2 bits per internal node. Conventional list representations use $\mathcal{O}(\log N)$ bits per internal node. The smaller number can be achieved by using prefix notation, where each node has one bit to mark it as internal or external, followed by B data bits if external.

How many dags are there with N internal nodes? They can have at most N+1 external nodes, which we will call $V_1 \dots V_{N+1}$. To write a dag in an efficient notation, observe that each internal node except the root must be pointed to by some other node. By ordering the nodes correctly, the first such pointer can be a short (2 bit) address. Put the root node in location 1 . Addresses contain 1 bit to distinguish internal from external nodes. External nodes have their B -bit values stored in the address field pointing to them. If an internal node has not been referred to before, it is allocated the first free address, and a single bit pointer suffices. If it has been referred to before, one bit says so, and is followed by a lg(N) bit pointer.

Of the 2N addresses, exactly N-1 are of 1 bit. The other N+1 are of length B+1 or lg N + 2 . Assuming $B > \lg N + 1$, we find the total storage is at most $N-1+(N+1)(B+1) = N(B+2) + B$, or B+2 bits per internal node. Again, conventional representations use much more.

To show a comparable lower bound, consider dags in which the left branches form a chain. There are more than N! different arrangements of the right branches, so the total number of bits required is at least lg(N!) , or lg N - constant bits per node. (Thanks to Dan Sleator.)

Problem 7.

Let f(D,N) be the expected number of maximal points out of N points in D dimensions. Consider the points $P_i = (x_1, x_2, \dots, x_D)$ in order of decreasing x_1 . Then P_i is maximal in D dimensions iff it is maximal among $P_1 \dots P_i$, restricted to D-1 dimensions, so $f(D,N) = f(D,N-1) + \frac{f(D-1,N)}{N}$, with boundary conditions $f(D,1) = f(1,N) = 1$. A routine array calculation, using min(N,D) cells, computes f in time $\mathcal{O}(N \cdot D)$.

Systems Reading List

Aho, A. and J. D. Ullman, *Principles of Compiler Design,* Addison-Wesley, 1977. [5/78]

Aho, A. and J. D. Ullman, *The Theory of Parsing, Translation, Compiling,* Vols. 1 and 2, Prentice-Hall, 1973. [5/78]

A1Gau1le, "Comments considered Carcinogenic", *ACM SICPLAN Notices, vol.* 12, no. 7, July 1977, p. 15. [5/78]

Algol W documentation. [2/69] [2/70] [2/71]

Allen, F. F., "Program Optimization", *The Annual Review of Automatic Programming,* M. Halpern and C. J. Shaw (eds.), Pergamon Press, 1969. [2/70] [2/71]

Amdahl, "Architecture of the IBM System 360", *IBM Journal of Research and Development,* vol. 8, no. 2, April 1964, pp. 87-101. [3/65] [2/69] [2/70] [2/71]

Amdahl, "The Structure of System 360", *IBM Systems Journal, vol. 3, nos. 2, 3,* 1964. [3/65] [2/69] [2/70] [2/71]

Arden, B., "Time-Sharing Systems: A Review", University of Michigan Summer Engineering Conference, June 19-30, 1967. [2/69] [2/70] [2/71]

Barron, D. W., *Assemblers and* Loaders, Elsevier, 1969. [2/71]

Barton, "A New Approach to the Functional Design of a Digital Computer", *Proceedings WJCC 19,* 1961, pp. 393-396. [3/65]

Barton, R. S., "A Critical Review of the State of the Programming Art", *AFIPS Conference Proceedings, (S* JCC 1963), pp. 169-177. [3/65]

Bauer and Eickel (ed.), *Compiler Construction, an Advanced Course, 2nd edition, Lecture Notes in Computer Science Vol. 21,* Springer-Verlag, 1976. [5/78]

Beckman, "Development in the Logical Organization of Computer Arithmetic and Control Units", *Proceedings of the IRE,* January 1961, p. 53. [3/65]

Bobrow, D. and B. Raphael, "A Comparison of List Processing Computer Languages", *CACM,* vol. 7, no. 4, April 1964, pp. 231-240. [3/65]

Bobrow, D. (ed.), *Symbol Manipulation Language Techniques,* North-Holland, 1968. [2/69] [2/70] [2/71]

Braun, *Digital Computer Design.* [3/65]

Brinch-Hansen, P., *Operating System Principles,* Prentice-Hall, 1973. [4/74] [2/75] [5/78]

Buchholz, *Planning a Computer System.* [3/65]

Burks, A. W., H. H. Coldstine, and J. von Neumann, "Preliminary Discussion of the Logical Design. of an Electronic Computing Instrument", *Collected Works of Von Neumann.* [3/65]

Buxton, J. *N., Simulation Programming Languages,* North-Holland, 1968. [2/70] [2/71]

Buzen, J. P. and U. 0. Cagliardi, "The Evolution of Virtual Machine Architecture", *Proceedings NCC 2973,* pp. 291-299. [2/75] [5/78]

Cocke, J. and J. T. Schwarz, *Programming Languages and Their Compilers,* Courant Institute, N. Y. U., 1969. [2/70] [2/71]

Coffman, E. C. and P. Denning, *Operating Systems Theory,* Prentice-Hall, 1973. [4/74]

CACM, "Survey of Programming Languages and Processors", vol. 6, no. 3, March 1963, pp. 93-99. [3/65]

Corbato, F. J. and V. A. Vyssotsky, "Introduction and Overview of the MULTICS System", *Proceedings AFIPS, vol. 27,* 1975. [2/70] [2/71]

Critchlow, A. J., "Generalized Multiprocessing and Multiprogramming Systems", *AFIPS Conference Proceedings* (FJCC 1963), pp. 107-126. [3/65]

Dahl, 0. J., E. J. Dijkstra, C. A. R. Hoare, *Structured Programming,* Academic Press, 1972. [4/74] [2/75] [5/78]

Denning, P. J., "Virtual Memory", *Computing Surveys,* vol. 2, no. 3, September 1970, pp. 155-190. [2/71] [2/72] [4/74] [2/75] [5/78]

Dennis, J. and E. Van Horn, "Programming Semantics for Multiprogrammed Computers', *CACM,* vol. 9, March 1966, pp. 143-155. [2/69] [2/70] [2/71]

Dijkstra, E. W., "Solution of a Problem in Concurrent Programming Control", *CACM,* vol. 8, September 1965, p. 569, [2/69] [2/70] [2/71]

Dijkstra, E. W., "Cooperating Seq uen tial Processes", in F. Cenuys (ed.), *Programming Languages,* Academic Press, 1968. [2/70] [2/71]

Dijkstra, E. W., "The Structure of the THE Multiprogramming System", *CACM,* vol. 11, no. 5, May 1968, pp. 341-346. [2/70] [2/71] [5/78]

Dijkstra, E. W., *A Discipline of Programming,* Prentice-Hall, 1976. [5/78]

Feldman, J. and D. Cries, "Translator Writing Systems", *CACM,* vol. 11, no. 2, February 1968, pp. [illegible] 3. [2/69] [2/70] [2/71] [2/72] [4/74] [2/75]

Floyd, R. W., "The Syntax of Programming Languages – A Survey", *IEEE Transactions on Electronic Computers,* vol. EC-l 3, no. 4, August 1964. [3/65]

Gear, C. W., *Computer Organization and Programming,* McGraw-Hill, 1969. [2/71]

Gotlieb, C. C., "Sorting on Computers", *CACM,* vol. 6, May 1963, pp. 194-201. [2/69]

Grau, A. A., U. Hill, and H. Langmaack, *Translation of ALGOL* 60, Springer-Verlag, 1967. [2/69] [2/70] [2/71]

Graham, *Principles of Systems Programming,* Wiley, 1975. [5/78]

Cries, D., *Compiler Construction for Digital Computers,* Wiley, 1971. [2/71] 121721 [4/74] [2/75] [5/78]

Haberman, A. N., *'Prevention of System Deadlocks", *CACM,* vol. 12, no. 7, July 1969, pp. 373-377. [2/701 [2/71]

Haberman, A. *N., Introduction to Operating System Design, S.* R. A., 1976. [5/78]

Hellerman and Conroy, *Computer Performance Evaluation,* Mcgraw-Hill, 1975. [5/78]

Hoare, C. A, R., "An Axiomatic Basis for Computer Programming', *CACM,* vol. 12, no. 10, October 1969. [5/78]

Hoare, C. A. R., "Hints on Programming Language Design', Stanford Artificial Intelligence Laboratory Memo AIM 224. [5/78]

Hoare, C. A. R., "Monitors: An Operating System Structuring Concept", *CACM,* vol. 17, no. 10, October 1974. [5/78]

Hopcroft, J. and J. Ullman, *Formal Languages and Their Relation to Automata,* Addison-Wesley, 1969. [5/78]

Hopgood, F. R. A., *Compiling Techniques,* American Elsevier, 1969. [2/70] [2/71]

IEEE Transactions on Electronic Computers, *Survey Issue on Programming Languages,* vol. EC-13, no. 4, August 1964. [3/65] [2/69]

Iverson, K. E., *A Programming Language,* Wiley, 1962. [3/65]

Knuth, D. E., *CACM, vol. 9, May* 1966, p. 322. [2/71]

Knuth, D. E., "The Remaining Trouble Spots in Algol 60", *CACM,* vol. 10, October 1967, pp. 611-618. [2/69] [2/70] [2/71]

Knuth, D. E., *The Art of Computer Programming, Vol. 1, Fundamental Algorithms,* Chapter 2, Addison-Wesley, 1973. [2/69] [2/70] [2/71] [2/72] [4/74] [2/75] [5/78]

Knuth, D. E., "Top-Down Syntax Analysis", *Acta Informatica,* vol. 1, pp. 79-l 10. [2/72] [4/74] [2/75]

Knuth, D. E., *The Art of Computer Programming, Vol. 3, Sorting and Searching,* Addison-Wesley, 1972. [2/72] [4/74] [2/75] [5/78]

Lampson, B. W., W. W. Lichtenberger, and M. W. Pirtle, 'A User Machine in a Time-Sharing System", *Proceedings of the IEEE,* vol. 54, December 1966, pp. 1766-1774. [2/69] [2/70] [2/71]

Lampson, B. W., "A Scheduling Philosophy for Multiprocessing Systems", *CACM,* vol. 11, no. 5, May 1968, pp. 347-359. [2/70] [2/71] [2/72] [4/74] [2/75]

Lampson, B. W., "Protection", *Operating Systems Review, vol.* 8, no. 1, January 1974. [5/78]

Ledley, *Digital Computer and Control Engineering.* [3/65]

Linden, T. A., "Operating System Structures to Support Security and Reliable Solftware", *Computing* Surveys, vol. 8, no. 4, December 1976, pp. 409-445. [5/78]

Liskov, B., "Abstraction Mechanisms in CLU", *CACM,* vol. 20, no. 8, 1977, p. 564. [5/78]

McGee, W. C., "On Dynamic Allocation", *IBM Systems Journal,* vol. 4, 1965, p. 194. [2/70] [2/69]

McKeeman, W., J. J. Horning, and D. Wortman, *A Compiler Generator,* Prentice-Hall, 1970. [2/71]

Morris, R., "Scatter Storage Techniques", *CACM,* vol. 11, no. 1, January 1968, pp. 38-44. [2/69] [2/70] [2/71] [2/72]

Parnas, D. L., "A Technique for Software Module Specification with Examples", *CACM,* vol. 15, no. 5, May 1972, p. 330. [4/74] [2/75]

Phister, *Logical Design of Digital Computers.* [3/65]

Popek, G. J. and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Archectures", *CACM,* vol. 17, no. 7, July 1974, p. 412. [2/75]

Rajchman, "Computer Memories: A Survey of the State of the Art", *Proceedings of the IRE,* January 1961, p. 104. [3/65]

Randell, B. and L. J. Russell, *Algol 60 Implementation,* Academic Press, 1964. [3/65] [2/70] [2/71]

Ritchie, D. M. and K. Thompson, "The UNIX Timesharing System", *CACM,* vol. 17, no. 9, July 1974, pp. 365-375. [5/78]

Rosen, S., "Programming Systems and Languages, A Historical Survey", *AFIPS Conference Proceedings,* (S JCC 1964), pp. 1- 15. [3/65]

Rosen, *S.* (ed.), *Programming Systems and Languages,* McGraw-Hill, 1966. [2/69] [2/70] [2/71]

Saltzer, J. H., "Traffic Control in a Multiplexed Computer System", Ph. D. thesis, MAC-TR-30, MIT, July 1966. [2/70] [2/71]

Sammet, J. E., *Programming Languages: History and Fundamentals,* Prentice-Hall, 1969. [2/70] [2/71]

Shaw, J. C., "JOSS, A designer's view of an experimental online computing system", *Proceedings IFIPS 1964 Fall joint Computing Conference,* pp. 455-464. [2/69]

Sorely, M., "High Speed Arithmetic in Binary Computers", *Proeedings of the IRE,* January 1961, p. 67. [3/65]

Stone, H. *S., Introduction to Computer Organization and Data Structures,* McGraw-Hill, 1972. [2/72] [4/ 741 [2/ 751

[2/71]e[2/72],[4/74]G[2/75]-Purpose Macrogenerator", *Computer Journal*, vol. 8, 1965, pp. 225-241.

Teichroew, D. and J. F. Lubin, "Computer Simulation – Discussion of the techniques and comparison of languages", *CACM*, vol. 9, October 1966, pp. 727-741. [2/69]

Tonge, F. M., "List Processing Techniques and Languages – Design Decisions", University of Michigan Engineering Summer Conference, June 19-30, 1967. [2/69]

van Wijngaarden, Adriaan, et. al., *Report on the Algorithmic Language ALGOL 68*, Mathematisch Centrum, 1969. [2/70] [2/71]

Watson, R. W., *Time Sharing System Design Concepts*, McGraw-Hill, 1970. [2/71] [2/72] [4/74] [2/75] [5/78]

Wegbreit, B., "The Treatment of Data Types in EL1", *CACM*, vol. 17, no. 5, May 1974. [5/78]

Wilkes, M. V., "The Design of Multiple-Access Computer Systems", *Computer Journal*, vol. 10, May 1967, pp. 1-9, and February 1968, pp. 315-320. [2/69] [2/70]

Wilkes, M. V., *Time-Sharing Computing Systems*, American Elsevier, 1968. [2/70] [2/71]

Wirth, N. and Hoare, C. A. R., "A Contribution to the Development of ALGOL,,, *CACM*, vol. 9, June 1966, pp. 413-432. [2/69] [2/70] [2/71]

Wirth, *N., Systematic Programming: An Introduction*, Prentice-Hall, 1973. [4/74] [2/75] [5/78]

Wirth, *N., Algorithms + Data Structures = Programs*, Prentice-Hall, 1976. [5/78]

Artificial Intelligence Aeading List

Abelson, Robert and Roger Schank, “Scripts, Plans, and Knowledge,,, *IJCAI 4*. [5/75]

Agin, G. and T. Binford, “Computer Description of Curved Objects”, *IJCAI 3, Advanced Papers*, 1973, pp. 629-640. [5/74] [5/75]

Allen, J. and D. Luckham, “An Interactive Theorem Proving Program”, *Machine Intelligence 5*, Edinburgh University Press, 1970, pp. 321-336. [5/75]

Amarel, Saul, “On Representations of Problems of Reasoning About Actions”, *Machine Intelligence* 3, American Elsevier, 1968, pp. 131-171. [4/72] [5/74] [5/75]

Amarel, S., “On the Representation of Problems and Goal-Directed Procedures for Computers”, *Comm. American Society for Cybernetics*, vol. 1, no. 2, 1969. [10/68] [11/69]

Anderson, J. R. and C. H. Bower, *Human Associative Memory*, Winston, 1973, especially Chapters 4, 5, 7. [5/75]

Anderson (ed.), [M&M] *Minds and Machines*, Contemporary Perspectives in Philosophy Series, Prentice-Hall, 1964. [5/75]

Arbib, Michael A., *The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory*, Wiley-Interscience, 1972. [5/75]

Ashby, *Design for A Brain*, 2nd edition. [3/65]

Banerji, R. B. and M. D. Mesarovic (eds.), *Theoretical Approaches to Non-numerical Problem Solving*, Springer-Verlag, 1970. [5/75]

Berliner, Hans, “Some Necessary Conditions for a Master Chess Program”, *IJCAI* 3, 1973, pp. 77-85. [5/75]

Bledsoe, W. W. and Bruell, “A Man-Machine Theorem-Proving System”, *Journal of AI, 5*, 1974, pp. 5 l-72. [5/75]

Bobrow, Dan and Allan Collins (eds.), [R&U] *Representation and Understanding*, Academic Press, 1975. [5/75]

Bobrow and Raphael, “New Programming Languages for AI Research,,, *Computing Surveys*, September 1974. [5/74] [5/75]

Buchanan, B., G. Sutherland, and E. A. Feigenbaum, “Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry”, *Machine Intelligence* 4, Edinburgh University Press, 1969. [1 l/69]

Buchanan, B., E. A. Feigenbaum, and J. Lederberg, “A Heuristic Programming Study of Theory Formation in Science”, *IJCAI* 2, The British Computer Society, 1971. [4/72]

Buchanan, B., Feigenbaum, and Sridharan, “Heuristic Theory Formation”, *Machine Intelligence 7,* 1972, pp. 267-280. [5/74][5/75]

Buchanan, Jack, “A Study in Automatic Programming”, Stanford Artificial Intelligence Laboratory Memo AIM-245, May 1974. [5/75]

Buchanan, J. and David C. Luckham, “On Automating the Construction of Programs”, Stanford AI Memo AIM-236, May 1974. [5/75]

Chandrasekharan and Reeker, “AI: A Case for Agnosticism”, *IEEE Transactions on Systems, Man. and Cybernetics,* January 1974, pp. 88-94. [5/75]

Chang, Chin-Liang and Richard Char-Tung Lee, *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, 1973. [5/75]

Chase, W. G. (ed.), [VIP]*Visual Information Processing,* Academic Press, 1973. [5/75]

Chomsky and Miller, “Introduction to the Formal Analysis of Natural Languages”, *Handbook of Math. Psychology, 2,* pp. 269-322, Wiley, 1963. [10/68]

Colby, Kenneth Mark, “Simulations of belief systems”, in Schank and Colby [S&C]. Also Colby, *Artificial Paranoia,* 1975. [5/75]

Colby and Enea, “Heuristic Methods for Computer Understanding of Natural Language in Con text-Restricted On-line Dialogues”, *Mathematical Biosciences,* 1, pp. l-25, 1967. [10/68]

Colby, K. M. and D. C. Smith, “Dialogues between Humans and an Artificial Belief System”, *IJCAI 1,* 1969. [11/69]

Coles, Steve and Rich Fikes (eds.), *SIGART Newsletter,* Special Interest Group on Artificial Intelligence (SICART), Association for Computing Machinery. [5/75]

Darlington and Burstall, “A System Which Automatically Improves Programs”, *IJCAI* 3, 1973, pp. 479-485. [5/74][5/75]

Davis, *Computability and Unsolvability.* [3/65]

Dij kstra, Dahl, and Hoare, *Structured Programming.* [5/75]

Dreyfus, Hubert, “Alchemy and Artificial Intelligence”. [5/75]

Dreyfus, Hubert, *What Computers Can’t* Do, Harper and Row, 1972. [5/75]

Duda, R., and P. Hart, *Pattern Classification and Scene Analysis,* Wiley, 1973. [5/75]

Erman, Fennell, Lesser, and Reddy, “System Orgranization for Speech Understanding”, *IJCAI 3,* 1973, pp. 194-199. [5/74]

Ernst, George W. and Allen Newell, *CPS: A Case Study in Generality and Problem Solving,* Academic Press, 1969. [11/69][5/75]

Evans, "A Heuristic Program to Solve Geometric Analogy Problems", *Proc.* SJCC 63, pp. 327-338. [1 0/68]

Evans, "ANALOGY", in Minsky [SIP]. [5/75]

Falk, Gilbert, "Computer Interpretation of Imperfect Line Data as a Three-dimensional Scene", Stanford AI Memo AIM-132, 1970. [4/72]

Falk, G., "Interpretation of Imperfect Line Data as a Three-dimensional Scene", *Artificial Intelligence Journal,* Vol. 3, Summer 1972, pp. 101-144. [5/75]

Feigen baum, E., "EPAM", in Feigenbaum [C&T]. [5/75]

Feigenbaum, "The Simulation of Verbal Learning Behavior", in Feigenbaum [C&T], pp. 295-309. [1 0/68]

Feigenbaum, E. A., "Artificial Intelligence: Themes in the Second Decade',, *Proceedings IFIP 68 Congress,* also Stanford AI Memo AIM-67, August 1968. [10/68] [11/69]

Feigenbaum, E. et. al., "On Generality and Problem Solving: A Case Study Using The DENDRAL Program", *Machine Intelligence 6,* 1971, pp. 165-190. [5/74] [5/75]

Feigenbaum, Edward A. and Julian Feldman (eds.), [C&T] *Computers and Thought,* McGraw-Hill, 1963. [3/65] [4/72] [5/74] [5/75]

Feigenbaum and Lederberg, "Mechanization of Inductive Inference in Organic Chemistry',, Stanford AI Memo AIM-54, 1967. [10/68]

Feldman, J. A., et. al., 'The Stanford Hand-Eye Project", *IJCAI* 1, May 1969. [11/69]

Feldman, J. A., et. al., "Grammatical Complexity and Inference',, Stanford CS Report CS-125, June 1969. [11/69]

Feldman, Pingle, Binford, Falk, Kay, Paul, Sproull, and Tenenbaum, "The Use of Vision and Manipulation to Solve the Instant Insanity Puzzle", *IJCAI* 2, 1971. [4/72] [5/74] [5/75]

Fikes, R. E. et. al., "Learning and Executing Generalized Robot Plans", *Artificial Intelligence,* Vol. 3, Winter 1972. [5/74] [5/75]

Findler, N. V. and B. Meltzer (eds.), *Artificial Intelligence and Heuristic Programming,* Edinburgh University Press, 1971. [5/75]

Finkel, R., Russel Taylor, Robert Bolles, Richard Paul, and Jerome Feldman, "AL, A Programming System for Automation", Stanford AI Memo AIM-243, November 1974. [5/75]

Firschein, 0. and S. Coles, "Forecasting and Assessing The Impact of Artificial Intelligence on Society", *IJCAI 3,* 1973, pp. 105- 120. [5/75]

Floyd, R. W., "Toward Interactive Design of Correct Programs', *IFIP 71,* Vol. 1, 1971, pp. 7-10. [5/74] [5/75]

Fogel, Lawrence J., Alvin J. Owens, and Michael J. Walsh, *Artificial Intelligence through a Simulation of Evolution,* Wiley, 1966. [5/75]

Celernter, "GEOMETRY", in Feigenbaum [C&T]. [5/75]

Gips, J., "Shape grammars and their uses", Stanford AI Memo AIM-231. [5/75]

Goldman, Neil, "Sentence Paraphrasing from a Conceptual Base", *CACM,* Vol. 18, No. 2, February 1975, pp. 96-107. [5/75]

Goldstein, Ira, "Understanding Simple Picture programs", see summary in Winston [MIT74]. [5/75]

Guard, J. R., et. al., "Semi-Automated Mathematics", *JACM* 16, January 1969, pp. 49-62. [5/75]

Green, C., "Application of Theorem Proving to Problem Solving", *IJCAI J,* 1969. [11/69]

Green, C. C., "The Application of Theorem Proving to QA Systems", Stanford AI Memo AIM-96, 1969. [5/75]

Green, C. C. and D. Barstow, "A Hypothetical Dialogue Exhibiting a Knowledge Base for a Program-Understanding System", Stanford AI Memo AIM-258, January 1975. [5/75]

Green and Raphael, "The Use of Theorem-Proving Techniques in Question-Answering Systems", *Proc. ACM Conference 68.* [10/68]

Green, Waldinger, Barstow, Elschlager, Lenat, McCune, Shaw, and Steinberg, "Progress Report on Program-Understanding Systems", Stanford AI Memo AIM-240, August 1974. [5/75]

Greenblatt, R. B., D. Eastlake, and S. Crocker, "The Greenblatt Chess Program", *Proceedings of the 1967 Joint Computer Conference.* [10/68] [11/69] [5/75]

Guzman, A., "Decomposition of a Visual Scene into Bodies", *Proceedings FJCC 68.* [10/68] [11/69]

Hart, Peter E., "Progress of a Computer Based Consultant", SRI AI Group Technical Note 99, January 1975. [5/75]

Hewitt, Carl and Brian Smith, "Towards a Programming Apprentice". [5/75]

Hiller and Isaacson, *Experimental Music.* [3/65]

Hunt, *Concept Formation: An Information Processing Problem.* [3/65]

Hunt, "Computer Simulation: Artificial Intelligence Studies and their Relevance to Psychology", *Annual Review of Psychology,* 1968. [10/68]

Hunt, Earl B., *Artificial Intelligence,* Academic Press, 1975. [5/75]

Jackson, Philip C., Jr., *Introduction to Artificial Intelligence,* Petrocelli Books, 1974. [5/75]

Kling, Robert E., "A Paradigm for Reasoning by Analogy", *Artificial Intelligence 2,* 1971, pp. 147- 178. [5/75]

Lenat, Doug, "BEINGS.. .". [5/75]

Lesser, Victor, Richard Fennell, Lee Erman, and D. Raj Reddy, "Organization of the Hearsay II Speech Understanding System", *IEEE Symposium on Speech Recognition,* Carnegie Mellon Univ., 1974. [5/75]

Lighthill, Sir J., Sutherland, Needham, Longuet-Higgins, and Michie, "AI: A Paper Symposium", British Science Research Council, April 1973. [5/75]

Lindsay, Peter H. and Donald A. Norman, *Human Information Processing: An Introduction to Psychology,* Academic Press, 1972. [5/75]

Low, James R., "Automatic Coding: Choice of Data Structures", Stanford AI Memo AIM-242, August 1974. [5/75]

Manna, Zohar, *Mathematical Theory of Computation.* [5/75]

Martin and Fateman, "The MACSY MA System", *2nd Symposium on Symbolic and Algebraic Manipulation,* 1971, pp. 59-75. [5/74] [5/75]

McCarthy, J., early papers, in *Mechanization of Thought Processes,* Her Majesty's Stationery Office, 1958. [5/75]

McCarthy, J., "Situations, Actions, and Causal Laws", Stanford AI Memo 2, July 1963. [10/68] [11/69] [5/75]

McCarthy, J., "Programs With Common Sense", Stanford AI Memo AIM-7, September 1963, also in Minsky [SIP]. [11/69] [5/75]

McCarthy, J., et. al., "A Computer with Hands, Eyes and Ears", *Proceedings FJCC 68.* [10/68] [11/69]

McCarthy, J., "Review of Lighthill debate", *AI Journal.* [5/75]

McCarthy, J. and P. Hayes, "Some Philosophical Problems from The Standpoint of AI", *Machine Intelligence* 4, Edinburgh University Press, 1969, pp. 463-502. [11/69] [4/72] [5/74] [5/75]

Meltzer, Bernard and Donald Michie (eds.), *Machine Intelligence,* Volumes l-6, American Elsevier, Volumes 7-, Halstead Press, 1967-. [5/75]

Meltzer, Bernard, and Bertram Raphael (eds.), *Artificial Intelligence: An International Journal,* North-Holland, 1970-. [5/75]

Michie, Donald, *On Machine Intelligence,* Wiley, 1974. [5/75]

Michie, D., J. G. Fleming, and J. V. Oldfield, "A Comparison of Heuristics, Interactive, and Unaided Methods of Solving a Shortest-Route Problem", *Machine Intelligence 3,* Edinburgh University Press, 1968. [11/69]

Minsky, Marvin, *Computation: Finite and infinite Machines,* Prentice Hall, 1968. [5/75]

Minsky, M., “FRAMES”, in Winston [MIT74]. [5/75]

Minsky, M., “Minds, Models, and Machines”, in Minsky [SIP]. [5/75]

Minsky, M. (ed.), [SIP] *Semantic Information Processing,* MIT Press, 1968. [4/72] [5/74] [5/75]

Minsky, M., “Steps toward Artificial Intelligence”, in Minsky [SIP]. [3/65] [10/68] [11/69] [5/75]

Minsky, M. and S. Papert, [PROGRESS] “Artificial Intelligence Progress Report”, MIT Project MAC AI Memo 252, 1972. [5/74] [5/75]

Minsky and Papert, *Perceptrons,* MIT, 1969. [5/75]

Moore, Jim and Allen Newell, “How can MERLIN understand?“, in Gregg (ed.), *Knowledge and Cognition,,* Lawrence Erlbaum Associates, 1973. [5/75]

Moorer, James A., “Music and Computer Composition”, *Comm. ACM,* January 1972. [5/75]

Nagy, “State of the Art in Pattern Recognition”, *Proc. IEEE, 56, 5,* pp. 836-862, 1968. [10/68]

Newborn, *M., Computer Chess,* Academic Press, 1975. [5/75]

Newell, A., “Limitations of The Current Stock of Ideas about Problem Solving”, in Kent and Taulbee (eds.), *Proceedings of a Conference on Electronic Information Handling,* Spartan, 1965, pp. 195-208. [11/69] [4/72] [5/75]

Newell, A., “Heuristic Programming: Ill-Structured Problems”, in Aronofsky (ed.), *Progress in Operations Research III,* Wiley, 1969. [10/68] [11/69] [5/74] [5/75]

Newell, “Some Issues of Representation in a General Problem Solver”, *Proc. SJCC 67,* 30, pp. 583-600. [10/68]

Newell, A., “Production Systems: Models of Control Structures”, CMU Report, May 1973, also in Chase [VIP]. [5/74] [5/75]

Newell, A., “Remarks on The Relationship Between AI and Cognitive Psychology”, in Banerji and Mesarovic (eds.), *Theoretical Approaches to Non-Numerical Problem Solving,* Springer-Verlag, 1970, pp. 363-400. [5/75]

Newell, A., Jeffrey Barnett, James W. Forgie, Cordell Green, Dennis Klatt, J. C. R. Licklider, John Munson, D. Raj Reddy, and William A. Woods, [SPEECH] *Speech Understanding Systems: Final Report of a Study* Croup, American Eisevier, 1973, especially Chapters 1, 4; Appendix A2. [4/72] [5/74] [5/75]

Newell and Ernst, “The Search for Generality”, *Proc. IFIP 65,* 1, pp. 17-24. [10/68]

Newell, Shaw, and Simon, "GPS", see article in Feigenbaum [C&T]. [5/75]

Newell and Simon, in *Handbook of Mathematical Psychology.* [3/65]

Newell, A., and Herbert A. Simon, [HPS] Human *Problem Solving,* Prentice-Hall, 1972, especially first and last chapters, look *over* Chapters 3, 4, 8. [5/74] [5/75]

Nilsson, N. J., "Searching Problem Solving and Came Playing Trees for Minimal Cost Solutions", *Proceedings IFIP 68 Congress.* [10/68] [11/69]

Nilsson, N. J., "A Mobile Automaton: An Application of Artificial Intelligence Techniques", *IJCAI* I, 1969. [11/69]

Nilsson, N. J., "Artificial Intelligence", SRI Technical Note 89, March 1974, also in *IFIP Congress* 74. [5/75]

Nilsson, Nils J., *Problem-Solving Methods in Artificial Intelligence,* McGraw-Hill, 1971. [4/72] [5/74] [5/75]

Norman, Donald, D. Rumelhart, and the LNR Research Group, *Explorations* in Cognition, Freeman, 1975. [5/75]

Papert, Seymour, "DREYFUS: Reply". [5/75]

Pingle, et. al., "Computer Control of a Mechanical Arm Through Visual Input", *Proc. IFIP* 68. [10/68]

Pohl, I., "Bi-Directional and Heuristic Search in Path Problems", Stanford CS Report CS-136, May 1969. [11/69]

Polya, G., Now *to* Solve lt, Doubleday Anchor Books, 1945; *Induction and Analogy in Mathematics,* Princeton Univ. Press, 1954; and *Patterns of Plausible Inference,* Princeton Univ. Press, 1968. [5/75]

Quillian, "Word Concepts", *Behavioral Science,* 12, 1967. [10/68]

Quillian, M. Ross, "Semantic Memory", in Minsky [SIP]. [11/69] [5/75]

Raphael, "Programming a Robot", *Proc. IFIP 68.* [10/68]

Raphael, Bertram, *The Challenge of Smarter Computers,* to be published. [5/75]

Reddy, "Computer Recognition of Connected Speech", *J. Acoust. Soc. Am., 42,* pp. 329-347, 1967. [10/68]

Reddy, D. Raj (ed.), *Speech Recognition: Invited Papers Presented at the 1974 IEEE Symposium,* Academic Press, 1975. [5/75]

Reddy, D. Raj, Erman, Fennell, and Neely, "The HEARSAY Speech Understanding System: An Example of the Recognition Process", *IJCAI 3,* 1973, pp. 185-193. [5/74]

Reddy, Erman, and Neely, "A Model and a System for Machine Recognition of Speech", *IEEE Transactions, Audio and Electrostatics,* Autumn, 21 (3), June 1973. [5/74]

Rieger, Chuck, "Conceptual Memory: A Theory and Computer Program for Processing the Meaning Content of Natural Language Utterances", Ph. D. thesis, Stanford AI **Memo** AIM-233, 1974. [5/75]

Roberts, L., "Machine Perception of Three Dimensional Solids", *Optical and Electro-optical Processing of Information,* MIT Press, 1965. [10/68] [11/69]

Rustin, R. (ed.), Natural *Language Processing,* Algorithmics Press, 1973. [5/75]

Sacerdoti, ABSTRIPS article, *IJCAI 3.* [5/75]

Samuel, A., "Studies in Machine Learning Using the Game of Checkers", in Feigenbaum [C&T]. [11/69]

Samuel, A., "Studies in Machine Learning Using the Came of Checkers II – Recent Progress", *IBM Journal R&D,* 1967. [10/68]

Schank, R. and K. Colby (eds.), [S&C] *Computer Models of Thought and Language,* Freeman, 1973, especially Chapters 1, 4, 5, 6. [5/74] [5/75]

Schank, Roger, Neil Goldman, Charles Rieger, and Chris Riesbeck, "MARGIE: Memory, Analysis, Response Generation, and Inference on English", *IJCAI 3,* 1973, pp. 255-261. [5/75]

hank, R. C. and L. C. Tessler, "A Conceptual Parser for Natural Language", *IJCAI I,* 1969. [11/69]

Sebestyen, *Decision Making Processes in Pattern Recognition.* [3/65]

Second International Joint Conference on Artificial Intelligence: Advance Papers of the Conference, OP 65449, University Microfilms, Ann Arbor, Michigan, 1971, xiv + 658 pages. [5/75]

Shannon and McCarthy, *Automata Studies.* [3/65]

Shortliffe, E., Axline, Buchanan, Merigan, Cohen, "An AI Program to Advise Physicians Regarding Antimicrobial Therapy", *Computers and Biomedical Research,* vol. 6, no. 6, December 1973, pp. 544-560. [5/74]

Shortliffe, Davis, Axline, Buchanan, Green, and Cohen, "Computer-based Consultations in Clinical Therapeutics: Explanation and Rule Acquisition Capabilities of the MYCIN System", *Journal for Computers in Biomedical Research,* Vol. 8, June 1975. [5/75]

Simmons, R., "Answering English Questions by a Computer: A Survey", *CACM* Vol. 8, No. 1, January 1965, pp. 53-70. [5/74] [5/75]

Simmons, R. F., "Natural Language Question-Answering Systems 1969", University of Texas Report TNN 87, 1969. [11/69]

Simmons, R., "Natural Language Question Answering Systems", *CACM* 13, No. 1, January 1970, pp. 15-30. [4/72] [5/74] [5/75]

Simon, H. A., *The Sciences of the Artificial,* MIT Press, 1969. [11/69]

Simon, H., “Lessons from Perception for Chess-Playing Programs (and vice versa)“, CMU Computer Science Research Review 1972- 1973, pp. 35-40. [5/74] [5/75]

Simon, Herbert A. and Laurent Siklossy (eds.), *Representation and Meaning: Experiments with Information Processing Systems,* Prentice-Hall, 1972. [5/75]

Slagle, J. R., *Artificial Intelligence: The Heuristic Programming Approach,* McGraw-Hill, 1971. [4/72] [5/75]

Slagle, J. R. and P. Bursky, “Experiments with a Multipurpose, Theorem-Proving Heuristic Program”, *JACM,* vol. 15, no. 1, 1968, pp. 85-99. [10/68] [11/69]

Sloman, Aaron, “Interactions Between Philosophy and Artificial Intelligence: The Role of Intuition and Non-Logical Reasoning in Intelligence”, *Journal of AI 2,* 1971, pp. 209-225. [5/75]

Sussman, Gerald, *A Computer Model of Skill Acquisition,* American Elsevier, 1975, see writeup in Winston [MIT74]. [5/75]

Third International Joint Conference on Artificial Intelligence: Advance Papers of the Conference, Stanford Research Institute, Menlo Park, California, 1973. [5/75]

Tompkins and Messick, *Computer Simulation of Personality.* [3/65]

Turing, “TEST”, in Feigenbaum [C&T]. Parodied in Anderson [M&M]. [5/75]

Uhr, Leonard, *Pattern Recognition, Learning, and Thought: Computer-programmed Models of Higher Mental Processes,* Prentice-Hall, 1973. [5/75]

Waldinger, R. J. and R. C. T. Lee, “PROW: A Step Toward Automatic Program Writing“, *IJCAI* I, 1969. [11/69]

Waldinger and Manna, “Reasoning about programs”, SRI technical report. [5/75]

Walker, Donald E. and Lewis M. Norton (eds.), *Proceedings of the International Joint Conference on Artificial Intelligence,* Cordon and Breach Science Publishers, New York, 1969. [5/75]

Waterman, D., “Generalization Learning Techniques for Automating the Learning of Heuristics”, Stanford AI Memo AIM-102 July 1969. [11/69]

Weiner, *N., The Human uses of Human Beings: Cybernetics and Society,* Anchor, 1954. [5/75]

Weizenbaum, J., "ELIZA", *CACM 9,* 1966, pp. 36-45. [5/75]

Wickelgren, Wayne A., *How to Solve Problems: Elements of a Theory of Problems and Problem-solving,* Freeman, 1974. [5/75]

Winograd, Terry, “Procedures as a Representation for Data in a Computer Program for Understanding Natural Language”, Ph. D. Thesis, M.I.T., 1971. [4/72]

Winograd, Terry, “Five Lectures on Artificial Intelligence”, Stanford AI Memo AIM-246, September 1974. [5/75]

Winograd, Terry, "A procedural model of language understanding", in Schank and Colby [S&C]. [5/75]

Winograd, Terry, *Understanding Natural Language,* Academic Press, 1972. [5/75]

W inograd, Terry, "Frame representations and the declarative/procedural controversy", in Bobrow and Collins [R&U]. [5/75]

Winograd, T. and G. J. Sussman, "Micro-Planner Reference Manual", 1971. [4/72]

Winston, P. H., "The M.I.T. Robot", *Machine Intelligence 7,* American Elsevier, 1972. [5/74] [5/75]

Winston, Patrick H. (ed.), *The Psychology of Computer Vision,* McGraw-Hill, 1975. [5/75]

Winston et. al., [MIT74] "New Progress in Artificial Intelligence", MIT AI Lab Report AI-TR-310, June 1974. [5/75]

Woods, William, Chapter in Schank and Colby [S&C]. [5/75]

Woods, W. A. and J. Makhoul, "Mechanical Inference Problems in Continuous Speech Understanding", *IJCAI* 3, 1973, pp. 200-207, Also see Woods' article in *IEEE Transactions on ASSP,* February 1975. [5/74] [5/75]

Yakimovsky, Y. and J. Feldman, "A Semantics-Based Decision Theory Region Analyzer", I*JCAI 3* Advanced Papers, 1973, pp. 580-588. [5/74] [5/75]

Numerical Analysis Reading List

Ahlberg, Nilson and Walsh, *The Theory of Splines and Their Applications.* [3/72]

Beckenbach, *Applied Combinatorial Mathematics.* [12/65]

Cheney, *Introduction to Approximation Theory.* [1/71] [3/72]

Conte and DeBoor, *Elementary Numerical Analysis.* [11/75]

Dahlquist, Björck and Anderson, *Numerical Methods.* [11/75]

Davis, *Interpolation and Approximation.* [12/65] [1/71] [3/72]

Davis and Rabinowitz, *Numerical Integration.* [1/71] [3/72]

Douglas, J., "A survey of numerical methods for parabolic differential equations", *Advances in Computers,* Vol. 2, 196 1. [12/65]

Faddeev and Faddeeva, *Computational Methods of Linear Algebra.* [12/65]

Fike, *Computer Evaluation of Mathematical Functions.* [3/72] [11/75]

Forsythe, "Notes on Computational Methods of Linear Algebra". [12/65]

Forsythe, Malcolm, and Moler, *Computer Methods for Mathematical Computations.* [11/75]

Forsythe and Moler, *Computer Solution of Linear Algebraic Systems.* [1/71] [3/72] [11/75]

Forsythe-W asow, *Finite-Difference Methods for Partial Differential Equations.* [12/65] [1/71] [3/72]

Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* [3/72] [11/75]

Greville, *Theory and Application of Spline Functions.* [1/71] [3/72]

Hammersley and Handscomb, *Monte Carlo Methods.* [12/65]

Hamming, *Numerical Methods for Scientists and Engineers.* [12/65]

Henrici, *Discrete Variable Methods in Ordinary Differential Equations.* [12/65] [1/71] [3/72] [11/75]

Henrici, *Elements of Numerical Analysis.* [12/65] [1/71] [3/72] [11/75]

Householder, *The Numerical Treatment of a Single Nonlinear Equation.* [3/72]

Isaacson and Keller, *Analysis of Numerical Methods.* [1/71] [3/72] [11/75]

Keller, *Numerical Methods for Two Point Boundary Value Problems.* [1/71] [3/72] [11/75]

Kowalik and Osborne, *Methods for Unconstrained Optimization Problems.* [3/72]

Krylov, *Approximate Calculation of Integrals.* [12/65]

Luen berger, *Optimization by Vector Space Methods.* [3/72]

Mitchell, A. R., *Computational Methods in Differential Equations.* [3/72]

Ortega, *Numerical Analysis, A Second Course.* [11/75]

Ortega and Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables.* [1/71][3/72]

Ostrowski, *Solution of Equations and Systems of Equations.* [12/65][1/71][3/72]

Ralston, *A First Course in Numerical Analysis.* [12/65][1/71][3/72]

Rice, J. R., *The Approximation of Functions.* [12/65]

Rice, *Mathematical Software.* [3/72][1 1/75]

Richtmyer and Morton, *Difference Methods for Initial Value Problems.* [1/71][3/72]

Rivlin, *An Introduction to Approximation of Functions.* [1/71][3/72]

Shampine and Gordon, *Computer Solution of Ordinary Differential Equations.* [11/75]

Smith, *Numerical Solution of Partial Differential Equations.* [11/75]

Stiefel, *Elements of Numerical Analysis.* [12/65][1/71]

Traub, *Iterative Methods for the Solution of Equations.* [12/65][1/71][3/72]

Varga, *Matrix Iterative Analysis.* [12/65][1/71][3/72]

Wendroff, *First Principles of Numerical Analysis.* [3/72][11/75]

Wilkinson, *The Algebraic Eigenvalue Problem.* [12/65][1/71][3/72]

Wilkinson, *Rounding Errors in Algebraic Processes.* [12/65]

Young, D., Chap. 11 of *Survey of Numerical Analysis,* John Todd, ed. [12/65]

Young, D. M., *iterative Solution of Large Linear Systems.* [3/72]

Computer Design Reading List

Arnold, B. H., *Logic and Boolean Algebra,* Prentice-Hall, 1965. [2/68]

Bartee, T. C., *Digital Computer Fundamentals,* 3rd ed., McGraw-Hill, 1971, [5/72]

Bell, C. G. **and** A. Newell, *Computer Structures: Readings and Examples,* McGraw-Hill, 197 1. [5/72]

Buchholz, W., *Planning a Computer System,* McGraw-Hill, 1962. [2/68]

Chu, Y., *Digital Computer Design Fundamentals,* McGraw-Hill, 1962. [2/68]

Eimbinder-Mactier, *Semiconductor Memories,* Wiley, 1970. [5/72]

Garrett, L. S., "Integrated-Circuit Digital Logic Families", *IEEE Spectrum,* October-December 1970. [5/72]

Gilbert, E. M. and E. F. Moore, "Variable-Length Binary Encoding,,, *SSTJ 38,* 1968, pp. 933-968, [2/68]

Golomb, *S.* W., *Digital Communications with Space Applications,* Prentice-Hall, 1964. [2/68]

Gschwind, H. W., *Design of Digital Computers: An Introduction,* Springer-Verlag, 1967. [2/68] [5/72]

Harrison, M. A., *introduction to Switching and Automata Theory,* McGraw-Hill, 1965. [2/68][5/72]

Hellerman, H., *Digital Computer System Principles,* McGraw-Hill, 1967. [2/68][5/72]

Hennie, F. *C., Finite-State Models for Logical Machines,* Wiley, 1968. [5/72]

Hill, F. J. and G. R. Peterson, *Introduction to Switching Theory and Logical Design,* Wiley, 1968. [5/72]

Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes',, *Proc. IRE 40,* 1961, pp. 1098-l 101. [2/68]

Kautz, W. H. (ed.), *Linear Sequential Switching Circuits: Selected Technical Papers,* Holden-Day, 1965. [2/68]

Lo, Arthur W., *Introduction to Digital Electronics,* Addison-Wesley, 1967. [2/68]

McCluskey, E. J., *Introduction to the Theory of Switching Circuits,* McGraw-Hill, 1965. [2/68][5/72]

Maley, G. A. and J. Earle, *The Logic Design of Transistor Digital Computers,* Prentice-Hall, 1963. [5/72]

Maley, C. A. and E. J. Skiko, *Modern Digital Computers,* Prentice-Hall, 1964. [2/68]

Marcus, M. P., *Switching Circuits for Engineers,* 2nd ed., Prentice-Hall, 1962. [2/68]

Murtha, J. C., "Highly Parallel Information Processing Systems", *Advances in Computers,* Vol. 7, pp. 1-116, Academic Press, 1966. [2/68]

Peterson, W. W., *Error-Correcting Codes,* MIT Press, Wiley, 1961. [2/68] [5/72]

Phister, M. *Logical Design of Digital Computers,* Wiley, 1958. [2/68]

Richards, R. K., *Arithmetic Operations in Digital Computers,* Van Nostrand, 1955. [2/68]

Richards, R. K., *Electronic Digital Systems,* Wiley, 1966. [2/68]

Texas Instruments, R. L. Morris and J. R. Miller, eds., *Designing with TTL Integrated Circuits,* McGraw-Hill, 1971. [5/72]

Whitesitt, J. E., *Boolean Algebra and its Application,* Addison-Wesley, 1961. [2/68]

Theory of Computation Reading List

Blum, M., "A Machine-Independent Theory of the Complexity of Recursive Functions", *JACM*, Vol. 14, 1967, pp. 322-336. [4/74]

Davis, M., *Computability and Unsolvability,* McGraw-Hill, 1958. [10/68] [1/70] [4/74]

Floyd, R. W., "Assigning Meanings to Programs", *Mathematical Aspects of Computer Science,* American Mathematical Society, 1967. [10/68] [1/70] [4/74]

Ginsburg, *S., The Mathematical Theory of Context-Free Languages,* McGraw-Hill, 1966. [10/68]

Hartmanis, J. and J. Hopcroft, "An Overview of Theory of Computational Complexity", *JACM*, Vol. 18, 1971, pp. 444-475. [4/74]

Hartmanis, J. and R. E. **Stearns,** *Algebraic Structure Theory of Sequential Machines,* Prentice-Hall, 1966. [1 0/68]

Hermes, H., *Enumerability, Decidability and Computability,* Academic Press, 1965. [1 0/68] [1/70] [4/74]

Hopcroft, J. and J. Ullman, *Formal Languages and their Relation to Automata,* Addison-Wesley, 1969. [1/70] [4/74]

Kleene, *S. C., Mathematical Logic,* Wiley, 1968. [10/68]

Lucas, P., P. Lauer, and H. Stigleitner, *Method and Notation for the Formal Definition of Programming Languages,* IBM Lab, Vienna, TR 25.087, June 1968. [1/70] [4/74]

Luckham, D. C., D. M. R. Park, and M. S. Paterson, *On Formalised Computer Programs,* (preliminary draft), Programming Research Croup, Oxford University, August 1967. [10/68] [1/70] [4/ 741

McCarthy, J., "Towards a Mathematical Science of Computation", *Proceedings IFIP Congress* 62, North-Holland, 1962, pp. 21-26. [10/68] [1/70] [4/74]

McCarthy, J., "Basis for a Mathematical Theory of Computation,,, *Computer Programming and Formal Systems,* Braffort and Hirschberg, eds., North-Holland, 1963, pp. 33-70. [1/70] [4/74]

McCarthy, J., "A Formal Description of a Subset of Algol", *Formal Language Description Languages for Computer Programing,* Steel, ed., North-Holland, 1966. [1/70] [4/74]

McCarthy, J. **and** J. Painter, "Correctness of a Compiler for Arithmetic Expressions", *Proceedings of Symposia in Applied Mathematics,* Vol. 19, 1967. [1/70] [4/74]

Manna, Z., "Termination of Algorithms", Ph. D. Thesis, Carnegie-Mellon University, 1968. [10/68]

Manna, Z., "Properties of Programs and the First-Order Predicate Calculus", *JACM*, Vol. 16, No. 2, April 1969. [1/70] [4/74]

Manna, Z. **and** A. Pnueli, "Formalization of Properties of Functional Programs", ACM *Symposium on Theory of Computing,* May 1969. [1/70] [4/74]

Mendelson, E., *Introduction to Mathematical* Logic, Van Nostrand, 1964. [10/68] [1/70] [4/74]

Minsky, M., *Computation: Finite and Infinite Machines,* Prentice-Hall, 1967. [10/68] [1/70] [4/74]

Nelson, R. J., *Introduction to Automata,* Wiley, 1968. [10/68] [1/70]

Rabin, M. 0. and D. Scott, "Finite Automata and Their Decision Problems", *Sequential Machines, E.* F. Moore, ed., Addison-Wesley, 1969. [10/68] [1/70] [4/74]

Robbin, J. W., *Mathematical Logic: A First* Course, W. A. Benjamin, 1969. [1/70] [4/74]

Robinson, J. A., "A Review of Automatic Theorem-Proving,,, *Mathematical Aspects of Computer Science,* American Mathematical Society, 1967. [10/68] [1/70] [4/74]

Rutledge, J. D., "On Ianov Program Schemata',, *JACM,* Vol. 11, No. 1, 1964, pp. l-9. [10/68] [1/70] [4/74]

Scott, D. and C. Strachey, "Towards a Mathematical Semantics for Computer Languages", *Proceedings Symposium on Computers and Automata,* Polytechnic Institute of Brooklyn, 1972. [4/74]

Weyhrauch, R. and R. Milner, "Program Semantics and Correctness in a Mechanized Logic", *First USA- Japan Computer Conference Proceedings,* 1972. [4/74]

Analysis of Algorithms Reading List

Aho, A. V., J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974. [5/78]

Berztiss, A. T., *Data Structures: Theory and Practice,* Academic Press, 1971. [5/74]

deBruijn, *N. C., Asymptotic Methods in Analysis,* 3rd ed., North-Holland, 1970. [5/74]

Even, Shimon, *Algorithmic Combinatorics,* MacMillan, 1973. [5/74]

Feller, W., *An Introduction to Probability Theory and its Applications,* Vol. 1, 3rd ed., Wiley, 1968. [5/74]

Harary, F., *Graph Theory,* Addison-Wesley, 1969. [5/74]

Hopcroft, J. E. and J. D. Ullman, *Formal Languages and Their Relation to Automata,* Addison-Wesley, 1969. [5/78]

Knuth, D. E., *The Art of Computer Programming, Volume I, Fundamental Algorithms,* 2nd ed., Addison-Wesley, 1973. [5/74] [5/78]

Knuth, D. E., *The Art of Computer Programming, Volume 2, Seminumerical Algorithms,* Addison-Wesley, 1969. [5/74] [5/78]

Knuth, D. E., *The Art of Computer Programming, Volume 3, Sorting and Searching,* Addison-Wesley, 1974. [5/74] [5/78]

Knuth, D. E., "Mathematical Analysis of Algorithms", *Proceedings IFIP Congress,* 1971, pp. 19-27. [5/74]

Nievergelt, J., J. Fariar, and E. M. Reingold, *Computer Approaches to Mathematical Problems,* Prentice-Hall, 1974. [5/74]

Parzen, E., *Modern Probability Theory and Its Applications,* Wiley, 1960. [5/74]

Reingold, E. M., J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice,* Prentice-Hall, 1977. [5/78]