Two Papers on the Selection Problem

TIME BOUNDS FOR SELECTION

By

Manual Blum, Robert W. Floyd, Vaughan Pratt,

Ronald L. Rivest, and Robert E. Tarjan

and

EXPECTED TIME BOUNDS FOR SELECTION

By

Robert W. Floyd and Ronald L. Rivest

STAN-CS-73-349

April 1973

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Time Bounds for Selection

by

Manuel Blum, Robert W. Floyd, Vaughan Pratt,
Ronald L. Rivest, and Robert E. Tarjan

Abstract

The number of comparisons required to select the i-th smallest of n numbers is shown to be at most a linear function of n by analysis of a new selection algorithm -- PICK. Specifically, no more than $5.430\dot{5}$ n comparisons are ever required. This bound is improved for extreme values of i , and a new lower bound on the requisite number of comparisons is also proved.

This work was supported by the National Science Foundation under grants GJ-992 and GJ-33170X.

1. Introduction

In this paper we present a new selection algorithm, PICK, and derive by an analysis of its efficiency the (surprising) result that the cost of selection is at most a linear function of the number of input items. In addition, we prove a new lower bound for the cost of selection.

The selection problem is perhaps best exemplified by the computation of medians. In general, we may wish to select the i-th smallest of a set of n distinct numbers, or the element ranking closest to a given percentile level.

Interest in this problem may be traced to the realm of sports and the design of (traditionally, tennis) tournaments to select the first and second-best players. In 1883, Lewis Carroll published an article [1] denouncing the unfair method by which the second-best player is usually determined in a "knockout tournament" -- the loser of the final match is often not the second-best! (Any of the players who lost only to the best player may be second-best.) Around 1930, Hugo Steinhaus brought the problem into the realm of algorithmic complexity by asking for the minimum number of matches required to (correctly) select both the first and second-best players from a field of n contestants. In 1932, J. Schreier [8] showed that no more than $n+\lceil \log_2(n) \rceil -2$ matches are required, and in 1964, S. S. Kislitsin [6] proved this number to be necessary as well. Schreier's method uses a knockout tournament to determine the winner, followed by a second knockout tournament among the (at most) $\lceil \log_2(n) \rceil$ players who lost matches to the winner, in order to select the runner-up.

For values of i larger than 2 , the minimum number of matches required to select the i-th best player from n contestants is known only for small values of n . The best previous general selection procedure is due to Hadian and Sobel [4], which requires at most $n-i+(i-1)\lceil \log_2(n-i+2)\rceil$ matches. They create a knockout tournament of $n-i+2$ players and then successively eliminate i-l who are "too good" to be the i-th best (using replacement selection).

No consistent notation has developed in the literature for the "i-th best". We shall use the following two operators:

$i \ominus S \underset{\text{def}}{=}$ (read "i-th of S") the i-th smallest element of S , for $1 \le i \le |S|$. Note that the magnitude of $i \ominus S$ increases as i increases. We shall often denote $i \ominus S$ by $i \ominus$ when S is understood.

$x \rho S \underset{\text{def}}{=}$ (read "x's rank in S") the rank of x in S , so that

$$x \rho S \ominus S = x .$$

The minimum worst-case (minimax) cost, that is, the number of binary comparisons required, to select $i\ominus$ will be denoted by f(i,n) , where $|S| = n$. We also introduce the notation:

$$F(\alpha) \underset{\text{def}}{=} \limsup_{n \to \infty} \frac{f(\lfloor \alpha(n-1)\rfloor + 1, n)}{n} , \quad \text{for } 0 \le \alpha \le 1 ,$$

to measure the relative difficulty of computing percentile levels.

In Section 2 we prove our main result, that $f(i,n) = O(n)$, by analysis of the basic selection algorithm, PICK.

In Section 3 PICK is "tuned-up" to provide our tightest results:

$$\max_{0 \le \alpha \le 1} F(\alpha) \le 5.430\dot{5} \qquad (1)$$

and

$$F(\alpha) \le 1 + 4.430\dot{5}\,\alpha/\beta + 10.86\dot{1} \lceil \log_2(\beta/\alpha) \rceil \alpha \ , \text{ for } 0 < \alpha \le \beta \ , \qquad (2)$$

where $\beta = .203688^-$. In Section 4 we derive the lower bound:

$$F(\alpha) \ge 1 + \min(\alpha, 1-\alpha) \quad , \quad \text{for } 0 \le \alpha \le 1 \ . \qquad (3)$$

There is no evidence to suggest that any of the inequalities (1) - (3) is the best possible. In fact, the authors conjecture that they can be improved considerably.

2. The New Selection Algorithm, PICK

In this section we present the basic algorithm and prove that $f(i,n) = \mathcal{O}(n)$. We assume that it is desired to select $i \theta S$, where $|S| = n$.

PICK operates by successively discarding (that is, removing from S) subsets of S whose elements are known to be too large or too small to be $i\theta$, until only $i\theta$ remains. Each subset discarded will contain at least one-quarter of the remaining elements. PICK is quite similar to the algorithm FIND (Hoare [5]), except that the element m about which to partition S is chosen more carefully.

PICK will be described in terms of three auxiliary functions $b(i,n)$, $c(i,n)$, and $d(i,n)$, which will be chosen later. We will omit argument lists for these functions in general, as no confusion can arise. Since we are interested in the asymptotic properties of PICK, we will also omit details to be given in Section 3 regarding the case when $n \bmod c \neq 0$.

PICK: (Selects $i \theta S$, where $|S| = n$ and $1 \leq i \leq n$)

1. (Select an element $m \in S$):

   (a) Arrange S into n/c columns of length c , and sort each column.

   (b) Select $m = b \theta T$, where $T \underset{\text{def}}{=}$ the set of n/c elements which are the d-th smallest element from each column. Use PICK recursively if $n/c > 1$.

2. (Compute $m \rho S$): Compare m to every other element x in S for which it is not yet known whether $m < x$ or $m > x$.

3. (Discard or halt):

   If $m \rho S = i$, halt (since $m = i \theta S$), otherwise if $m \rho S > i$, discard $D = \{x \mid x \geq m\}$ and set $n \leftarrow n - |D|$, otherwise discard $D = \{x \mid x \leq m\}$ and set $n \leftarrow n - |D|$, $i \leftarrow i - |D|$.

   Return to step 1.

This completes the description of PICK. We are now ready to prove:

Theorem 1. $f(i,n) = \mathcal{O}(n)$.

Proof: We show that a reasonable choice of functions $b(i,n)$, $c(i,n)$, and $d(i,n)$ result in a linear time selection algorithm. Let $h(c)$ denote the cost of sorting c numbers using Ford and Johnson's algorithm [2]. It is known [3] that:

$$h(c) = \sum_{1 \le j \le c} \lceil \log_2(3j/4) \rceil \quad . \tag{4}$$

The cost of step l(a) is $n \cdot h(c)/c$, making obvious the fact that $c(i,n)$ must be bounded above 'by a constant in order for PICK to run in linear time.

Letting P(n) denote the maximum cost of PICK for any i , we can bound the cost of step l(b) by $P(n/c)$. After step 1, the partial order determined for S may be represented as in Figure 1:

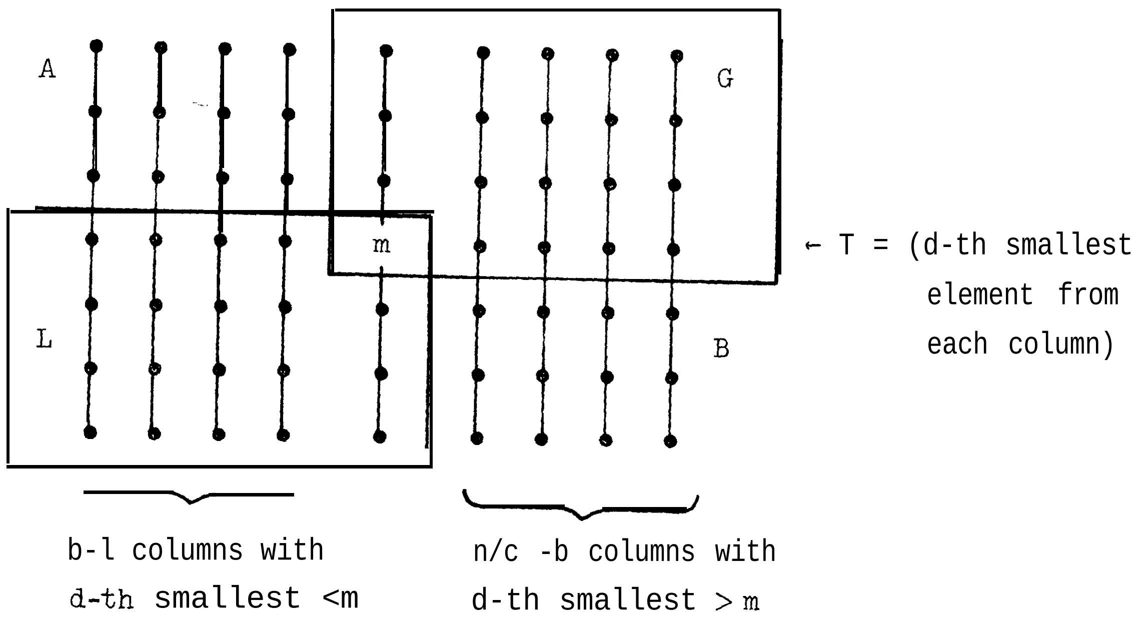

A
G
m
L
B
← T = (d-th smallest element from each column)
b-l columns with d-th smallest <m
n/c -b columns with d-th smallest > m

Figure 1

Here we have the n/c columns of length c portrayed with their largest elements on top. Since the recursive call to PICK in step l(b) determines which elements of T are <m , and which are > m , we separate the columns as in Figure 1. Every element in box G is clearly greater than m , while every element in box L is less. Therefore only those elements in quadrants A and B need to be compared to m in step 2.

It is easy to show that no elements are ever incorrectly discarded in step 3: if $m \rho S > i$, m is too large, so that m and all larger elements may be discarded, and symmetrically for the case $m \rho S < i$. Note that at least all of G or all of L will be discarded. It is now obvious that

$$P(n) \leq \frac{n \cdot h(c)}{c} + P(n/c) + n + P(n - \min(|L|, |G|)) \cdot \qquad (5)$$

To minimize $P(n)$ we choose $c = 21$, $d = 11$, and $b = n/2c = n/42$ (so that m is the median of T, and T is the set of column medians). This implies

$$P(n) \leq \frac{66\ n}{21} + P(n/21) + n + P(31n/42) , \qquad (6)$$

since $h(21) = 66$. This implies by mathematical induction that

$$P(n) \leq \frac{58\ n}{3} = 19.\dot{6}\ n . \qquad (7)$$

The basis for the induction is that, since $h(n) < 19\ n$ for $n < 10^5$, any small case can be handled by sorting. PICK runs in linear time because a significant fraction of S is discarded on each pass, at a cost proportional to the number of elements discarded on each step. Note, however, that we must have $c \geq 5$ for PICK to run in linear time. Q.E.D.

3. Improvements to PICK

The main result that $f(i,n) = \Theta(n)$, has now been proved. We thank the referee for his comment: "The authors have a right to optimize (if they don't, someone else will)." This section contains a detailed analysis of our improved versions of PICK.

We describe two modifications to PICK: PICK1, which yields our best overall bound for $F(\alpha)$, and PICK2, which is more efficient than PICK1 for i in the ranges $i < \beta n$ or $i > (1-\beta)n$ for $\beta = .203688^-$. The description and analysis of PICK1 is relatively detailed and lengthy -- we do not expect the average reader to wade through it! The optimized algorithm is full of red tape, and could not in practice be implemented efficie but in principle for any particular n could be expanded into a decision tree without red-tape computation. The basic differences between PICK and PICK1 will be listed shortly. We assume (by arguments of symmetry) that $i \leq \lceil n/2 \rceil$ throughout this section.

Theorem 2. $F(\alpha) \leq 5.430\dot{5}$, for $0 \leq \alpha \leq 1$.

Proof: By analysis of PICK1, which differs from PICK in the following respects:

(i) The elements of S are sorted into columns only once, after which those columns broken by the discard operation are restored to full length by a (new) merge step at the end of each pass.

(ii) The partitioning step is modified so that the number of comparisons used is a linear function of the number of elements eventually discarded.

(iii) The discard operation breaks no more than half the columns on each pass, allowing the other modifications to work well.

(iv) The sorting step implicit in the recursive call to select m is partially replaced by a merge step for the second and subsequent iterations, since (iii) implies that 2/3 of the set T operated on at pass j were also in the recursive call at pass j-1 .

The term "k-column" will be used to denote a sorted column of length k . The optimal value of the function c , 15 , will be used explicitly throughout for clarity. The algorithm is presented as three separate procedures, each of which selects $i \theta S$ from S , given that the partial order already determined for S is one of three types. Procedure PICK1 is the outermost procedure, which assumes that no information is known about the elements of S .

Procedure PICK1: (Selects $i\theta S$ from S, where $|S| = n$ and $1 \le i \le \lceil n/2 \rceil$).

1. If $n \le 45$, sort S, print $i\theta$, and halt.
2. Sort S into $\lfloor n/15 \rfloor$ 15-columns and possibly one (n mod 15)-column.
3. Use procedure PICK1a to select $i\theta$.

Procedure PICK1a: (Same as PICK1, except that S is already sorted into 15-columns).

1. If $n \le 45$, sort S, print $i\theta S$, and halt.
2. Sort the set T of column medians into 15-columns and possibly one ($\lceil n/15 \rceil$ mod 15) -column.
3. Use procedure PICK1b to select $i\theta S$.

Procedure PICK1b: (Same as PICK1a, except that T is also already sorted into 15-columns).

1. Use procedure PICK1a to select m, the median of T.
2. Partition $A \cup B$ of Figure 1 about m as follows, stopping as soon as it becomes clear that $m\rho S < i$ or $m\rho S > i$:

   (i) Insert m into each 7-column of B, using binary insertion (3 comparisons/column).

   (ii) Insert m into each 7-column of A, using a linear search technique beginning near each 15-column median.
3. If $m\rho S = i$, print m ($= i\theta S$), and halt, otherwise if $m\rho S > i$, discard $G \cup \{x \mid x \in B \text{ and } x > m\}$, otherwise discard $L \cup \{x \mid x \in A \text{ and } x < m\}$ and decrease i by the number of elements discarded.
4. Restore S to a set of 15-columns by the following merge operations. Here $|X|$ will denote the number of elements inaset X. Let U be the set of columns of lengths < 15 (in fact, each column of U has length ≤ 7). Let $Y \subset U$

be the set of shortest columns of U , such that $|Y| = |U|/15$, and let V be the set of all 7-columns in U-Y . Split $U-(V \cup Y)$ into two subsets X and W such that W contains w columns, W's columns are not shorter than X's , and $|W| + |X| = 7w$. Then

(i) Extend every column in W to length 7 by using binary insertion to place each element of X into a column of W .

(ii) Now every column in U-Y is a 7-column. Merge them pairwise to form 14-columns.

(iii) Use binary insertion to place each element of Y into a 14-column. Now S has been restored to a set of 15-columns.

5. Restore the set T of column medians to 15-columns as follows. Let $Z \subset T$ be those column medians which were column medians in step 1. The elements of Z are already sorted into columns of size 8 or greater, since step 3 of the recursive call at step 1 discarded Z in strings of those sizes.

(i) Merge the columns of Z together to form 15-columns and some left-overs, treating each column size separately:

8-columns: Merge these pairwise to form 15-columns with one element left over. Write this as 2(8): 8+7, 1 leftover.

9-columns: 5(9): 9+6, 9+6, 9+3+3, no leftovers.

10-columns: 3(10): 10+5, 10+5, no leftovers.

11-columns: Set aside 1/45 of the 11-columns and break them into 1-columns, then do 4(11)+1(1): 11+4, 11+4, 11+3+1, no leftovers.

12-columns and larger: set aside some elements for binary insertion into the remaining columns

Sort the leftovers into 15-columns.

(ii) Sort T-Z into 15-columns.

Now T has been restored to a set of 15-columns.

6. Decrease n by the number of elements discarded in step 5. If $n \le 45$, sort S, print $i \ominus S$ and halt, otherwise return to step 1.

This completes the description of the algorithm. To analyze PICK1, we introduce the following notation:

$P1(n), P1a(n), P1b(n)$	$\underset{\text{def}}{=}$	the maximum costs, respectively, of procedures PICK1, PICK1a, and PICK1b.
v	$\underset{\text{def}}{=}$	the number of comparisons made in step PICK1b (2ii).
d	$\underset{\text{def}}{=}$	the number of elements from $A \cup B$ discarded in step PICK1b (3).
ga, gb	$\underset{\text{def}}{=}$	the number of elements from A, B found in step PICK1b (2) to be $>m$.
$\ell a, \ell b$	$\underset{\text{def}}{=}$	the number of elements from A, B found in step PICK1b (2) to be $<m$.
w, x, y	$\underset{\text{def}}{=}$	the number of columns in sets W, X, Y in PICK1b (4).

Since $h(15) = 42$, we have immediately:

$$P1(n) \le \frac{42n}{15} + P1a(n) = 2.8n + P1a(n) \qquad (8)$$

$$P1a(n) \le \frac{42n}{225} + P1b(n) = .18\dot{6}n + P1b(n) \; . \qquad (9)$$

The following lemma demonstrates the tradeoff created in step PICK1b(2) between v and d :

Lemma 1. $v \leq d + n/30$.

Proof: There are two cases to consider, since either L or G is discarded in step PICK1b(3).

Case 1 (L is discarded): There can clearly be at most one comparison for every column in A , plus one for each element of A discarded.

Case 2 (G is discarded): Thus $|L| + \ell a + \ell b = i+1 < \lceil n/2 \rceil + 1 < |L| + |B| \leq |L| + gb + \ell b$. Thus $gb \geq \ell a$, but $\ell a \geq v - n/30$ as in case 1, yielding the lemma since $d = gb$ here.

Q.E.D.

The following lemma allows the costs of step PICK1b(4) to be bounded:

Lemma2: $|X| < 6d/7$. (11)

Proof: We have $d \geq 7(w + x + y) - |W| - |X| - |Y|$, and $7w - |W| = |X|$, yielding $d \geq 7x + 7y - |Y| \geq 7x$, but $6x \geq |X|$, so-that $d \geq 7|X|/6$.

Q.E.D.

Step PICK1b(5i) takes, in the worst case, 21/20 comparisons/element to merge and sort Z into 15-columns (detailed analysis omitted -- this happens when z contains only 8-columns). Since $|Z| = n/30$, this step takes at most 7 n/200 comparisons. We may now write the following recurrence for P1b(n) :

$$\mathrm{P1b}(n) \le \underbrace{\mathrm{P1a}(\lceil n/15\rceil)}_{\text{step 1}} + \underbrace{3(n/30)}_{\text{step 2i}} + \underbrace{(d+n/30)}_{\text{step 2ii}} + \underbrace{3(6d/7)}_{\text{step 4i}} +$$

$$\underbrace{13(7n/30-d)/15}_{\text{step 4ii}} + \underbrace{4(7n/30-d)/15}_{\text{step 4iii}} + \underbrace{7n/200}_{\text{step 5i}} +$$

$$\underbrace{42(7n/30-d)/225}_{\text{step 5ii}} + \underbrace{\mathrm{P1b}(11n/15-d)}_{\text{subsequent iterations}}$$

Simplifying yields

$$\mathrm{P1b}(n) \le \left(\frac{5n}{n+5d}\right) \cdot \left(\frac{13197\ n}{27000} + \frac{3546\ d}{1575}\right). \qquad (12)$$

The right-hand side of (12) is maximum at d = 0 , so that

$$\mathrm{P1b}(n) \le \frac{13197\ n}{27000} = 2.443\dot{8}\ n\ , \qquad (13)$$

$$\mathrm{P1a}(n) \le 2.630\dot{5}\ n\ , \qquad (14)$$

and

$$\mathrm{P1}(n) \le 5.430\dot{5}\ n\ . \qquad (15)$$

Since

$$\max_{1 \le c \le 45} \frac{h(c)}{c} = \frac{h(45)}{45} < 5.43\ n\ , \qquad (16)$$

the basis for the induction yielding (12) is justified, thus also taking care of steps PICK1(1), PICK1a(1), and PICK1b(6), and yielding our theorem.

Q.E.D.

While PICK1 provides a good uniform bound on F(a) , better results can be achieved for values of α near 0 or 1 . We now present the algorithm PICK2, which yields the following result.

Theorem 3. $F(\alpha) \leq 1 + 4.430\dot{5}\,\alpha/\beta + 10.86\dot{1}\lceil \log_2(\beta/\alpha)\rceil\alpha$, for $0 < \alpha \leq \beta$, (17)

where $\beta = .20368\dot{8}$.

Proof: By analysis of PICK2, which is essentially the same as PICK with the functions $b(i,n)$, $c(i,n)$, and $d(i,n)$ chosen to be i , 2 , and 1 , respectively, and with the partitioning step eliminated. In detail:

Procedure PICK2: (Selects $i \theta S$, where $|S| = n$, and $i < \beta n$) :

1. Compare the elements of S pairwise to form $\lfloor n/2 \rfloor$ pairs and possibly one left-over.
2. If $i < \beta \lfloor n/2 \rfloor$ use procedure PICK2, otherwise use PICK1, to select m as the i-th smallest element of the set T of lesser elements of each pair. See Figure 2.

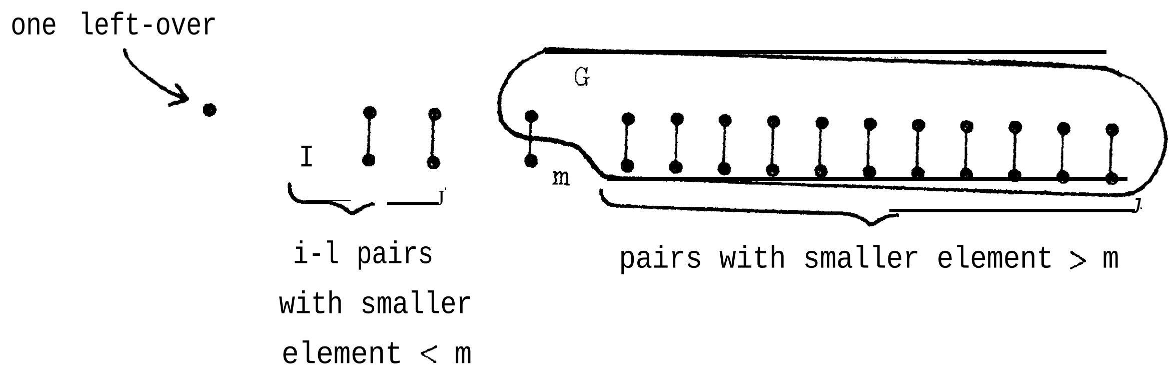

one left-over
G
I
m
i-1 pairs with smaller element < m
pairs with smaller element > m

Figure 2

3. Discard all elements known to be > m , that is, those elements in the circle G of Figure 2.
4. Use procedure PICK1 to select $i \theta S$ from S .

This completes the description of procedure PICK2. Note that this reduces to a simple knockout tournament when $i = 1$! Using $P2(i,n)$ to denote the maximum number of comparisons used by PICK2 to select $i\Theta$, we may derive:

$$P2(i,n) \le \underbrace{\lfloor n/2 \rfloor}_{\text{step 1}} + \underbrace{\min(P1(\lfloor n/2 \rfloor), P2(i, \lfloor n/2 \rfloor))}_{\text{step 2}} + \underbrace{P1(2\ i)}_{\text{step 4}} \qquad (18)$$

For particular values of i and n, procedure PICK2 is called $t = \lceil \log_2(\beta n/i) \rceil$ times in succession during the recursive calls at step 2, before procedure PICK1 is called. Thus

$$P2(i,n) \le \sum_{0<j<t} \lfloor n/2^j \rfloor + P1(\lfloor n/2^t \rfloor) + t\ P1(2\ i) \qquad (19)$$

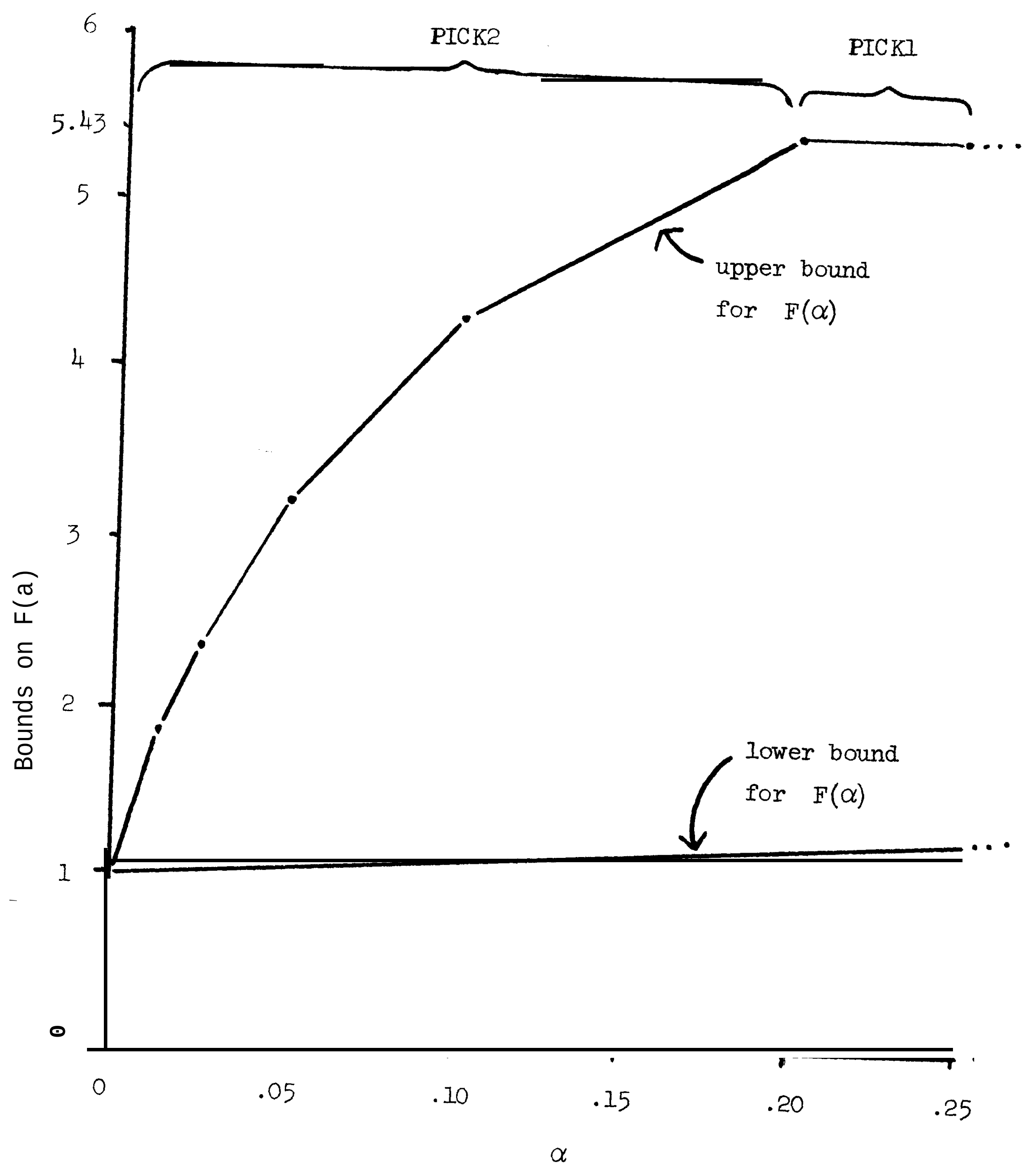
This directly implies our theorem. The proper value for β, $.203688^{-}$, is the largest value such that $P2(\lceil \beta n \rceil, n) < P1(n)$.

Q.E.D.

The results of this section are summarized in Figure 3, where our bounds for $F(\alpha)$ are plotted against α. It is not unreasonable to conjecture that the true curve for $F(\alpha)$ is unimodal and peaks at $\alpha = 1/2$. The relative complexity of the algorithm PICK1 leads the authors to conjecture that our upper bound can be significantly improved.

4. A Lower Bound

In this section a lower bound for $F(\alpha)$ is derived through the use of an "adversary" approach (this technique is called the construction of an "oracle" by Knuth. See for example [7], Section 5.3.2.) The selection



PICK2
PICK1
6
5.43
5
4
3
2
1
0
Bounds on F(a)
upper bound for F(α)
lower bound for F(α)
0
.05
.10
.15
.20
.25
α

Figure 3

process may be formulated as a game between the selection algorithm (player A), who is trying to find $i\theta S$ with as few comparisons as possible, and his adversary (player B), who is trying to force player A to make as many comparisons as possible. The players take alternate turns: each play by A consists of posing a "comparison question", such as "Is $x < y$?" (for any $x,y \in S$), to which player B on his turn must respond with either "Yes" or "No". Player B's responses may be completely arbitrary, as long as he does not contradict his previous responses to A's questions. When A has extracted enough information from B to determine $i\theta S$, the game is over.

The advantage of this approach is that a non-trivial lower bound for the length of this game can be found, independent of A's strategy, simply by supplying a sufficiently clever strategy for B. The length of this game is of course here the usualminimax cost, that is,

$$f(i,n) \underset{\text{def}}{=} \min_{A} \max_{B} c(A,B) , \qquad (20)$$

where $c(A,B)$ is the length of the game between particular A and B strategies.

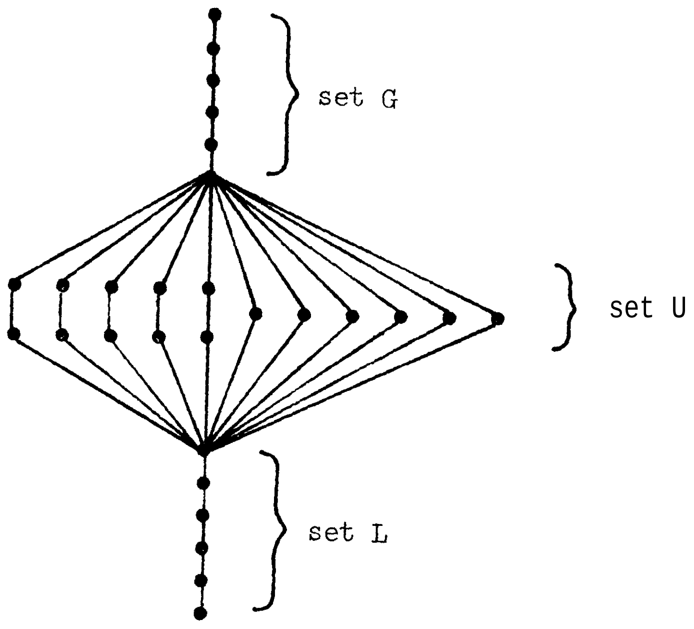
Player B in this game of course plays the role of the "data". A strategy for player B is in effect a rule for calculating a particularly bad (that is, costly) set of data for A's strategy, since an actual set of numbers can always be constructed that are consistent with B's replies. A good strategy for player B is thus a procedure for "bugging" any given player A strategy.

We will now describe the particular player B strategy which yields our lower bound. As the game progresses there will of course be many elements x such that player A has determined enough about x to know that $x \neq i\Theta$, that is either $x < i\Theta$ or $x > i\Theta$. Player B will initially consider all elements $x \in S$ to be members of the set U , meaning that player B (and thus player A as well) is uncertain as to whether $x < i\Theta$, $x = i\Theta$, or $x > i\Theta$. After a while, though, player A will be able to force the issue regarding particular elements, that is, force player B to decide the status of a particular element $x \in U$. If B decides that $x > i\Theta$, he will remove x from U and place it in set G . Similarly if he decides that $x < i\Theta$, he will remove x from U and place it in set L . Both G and L are initially empty. The element that turns out to be $i\Theta$ will thus be one (any one) of the elements still in U , so that as long as $|U| > 1$ the game must continue. Our player B strategy thus attempts to keep U as large as possible for as long as possible.

The game must actually consist of two phases as far as B's strategy is concerned. As long as $|L| < i-1$ and $|G| < n-i$, player B has complete freedom to put an element $x \in U$ into either L or G . After one of L or G fills up, however, B is quite restricted and must play differently, since he is not allowed to make $|L| > i$ or $|G| \geq n-i+1$. At that time, however, the game "degenerates" in the sense that player A has merely to find the minimum (or maximum) element of U .

During the first phase, player B will never remove more than one element x from U on a single turn. This will not cause any complications as long as x is a maximal (or minimal) element of U and player B puts x into set G (set L). Each element placed in set G (set L) is assumed to be less than (respectively, greater than) all previous elements placed in that set, as well as greater than (respectively, less than) any elements still remaining in U and L (respectively, U and G). This rule completely defines B's responses except when player A wishes to compare two elements $x,y \in U$. In addition, player B will only remove an element from U when A makes.such a request.

Player B will always restrict membership in U so that every member $x \in U$ is either a maximal or minimal element of U (or both) with respect to the partial order already fixed on S by B's previous responses. In fact, B will maintain the even stronger condition that <u>for each element</u> $x \in U$, <u>there will be at most one</u> $y \in U$ <u>for which it is known whether</u> $x < y$ or $y < x$. The partial order for S assumed by B may thus always be diagrammed:



Set U therefore contains only three "element-states", and we define $\sigma(x)$ to be -1 , 0 , or 1 respectively according to whether x is the lesser element of a pair, an isolated element, or the greater element of a pair. B's strategy for a comparison between two elements $x,y \in U$ is now easy to state (we assume without loss of generality that $a(x) < \sigma(y)$):

(i) respond " x is less than y ", and

(ii) if $a(x) = \sigma(y) = 0$ do nothing, otherwise

if $\sigma(x) = -1$ remove x from U and place it in L ,

otherwise remove y from U and place it in set G .

Essentially B's strategy creates a new pair in U if $a(x) = a(y) = 0$, otherwise one element is removed from U and the number of pairs in U decreases by one. Let

c = the number of comparisons made so far, and

p = the number of pairs currently in U .

It is simple to verify that B's strategy maintains the condition

$$c - p + 2|U| = 2n , \qquad (21)$$

as long as the game is still in the first phase (this is clearly true atthestart when $c = p = 0$ and $|U| = n$). At the end of phase one, either L or G is full, so that

$$|U| \leq n - \min(i-1, n-i) . \qquad (22)$$

Furthermore, it must take player A at least $|U|-1-p$ comparisons to finish the game during the second phase, since he must at least do the work of finding the smallest (or largest) element of U , which requires $|U|-1$ comparisons, of which p have already been made. The total number of comparisons made is thus at least

$$f(i,n) \geq c + |U| -1-p \geq n + \min(i-1, n-i) -1, \text{ for } 1 \leq i \leq n \qquad (23)$$

from (21) and (22). Taking the limit as $n \to \infty$, keeping $i = \lfloor \alpha(n-1) \rfloor +1$, we get

$$F(\alpha) \geq 1 + \min(\alpha, 1-\alpha) . \qquad (24)$$

This bound is also plotted in Figure 3.

5. summary

The most important result of this paper is that selection can be performed in linear time, in the worst case. No more than 5.4305 n comparisons are required to select the i-th smallest of n numbers, for any i, $1 \leq i \leq n$. This bound can be improved when i is near the ends of its range.

A general lower bound is also derived which shows, in particular, that at least $3n/2 - 2$ comparisons are required to compute medians.

The authors believe that the constants of proportionality in both the upper and lower bounds can be considerably improved.

References

[1] Carroll, Lewis. "Lawn Tennis Tournaments," St. James's Gazette (August 1, 1883), pp 5-6. Reprinted in The Complete Works of Lewis Carroll. New York Modern Library (1947).

[2] Ford, L. R. and S. M. Johnson. "A tournament problem," The American Mathematical Montly 66, (May 1959), pp 387-389.

[3] Hadian, Abdollah. "Optimality properties of various procedures for ranking n different numbers using only binary comparisons," Technical Report 117, Dept. of Statistics, Univ. of Minnesota, (May 1969). (Ph.D. Thesis). 61 pp.

[4] _____ and Milton Sobel. "Selecting the t-th largest using binary errorless comparisons," Technical Report 121, Dept. of Statistics, Univ. of Minnesota, (May 1969), 15 pp.

[5] Hoare, C. A. R. "Find (Algorithm 65)," Communications of the ACM (July 1961), pp 321-322.

[6] Kislitsin, S. S. "On the selection of the k-th element of an ordered set by pairwise comparisons," Sibirsk Math. Z. 5 (1964), pp. 557-564. (MR 29, no. 2198). (Russian).

[7] Knuth, Donald E. The Art of Computer Programming, Volume III, Sorting and Searching, Addison-Wesley (1973).

[8 3 Schreier Jósef. "O systemach eliminacjii w turniejach," ("On elimination systems in tournaments"), Ma-thesis Polska 7 (1932), pp. 154-160 (Polish).

Expected Time Bounds for Selection

by

Robert W. Floyd and Ronald L. Rivest

Expected Time Bounds for Selection

by

Robert W. Floyd and Ronald L. Rivest
Stanford Computer Science Department
Stanford University
Stanford, California 94305

Abstract

A new selection algorithm is presented which is shown to be very efficient on the average, both theoretically and practically. The number of comparisons used to select the i-th smallest of n numbers is $n + \min(i,n-i) + o(n)$. A lower bound within 9% of the above formula is also derived.

Keywords and Phrases: selection, computational complexity, medians, tournaments, quantiles

CR Categories: 5.30, 5.39

This work was supported by the National Science Foundation under grants GJ-992 and GJ-33170X. ,

1. Introduction

In this paper we present new bounds (upper and lower) on the expected time required for selection. The selection problem can be succinctly stated as follows: given a set X of n distinct numbers and an integer i , $1 \leq i \leq n$, determine the i-th smallest element of X with as few comparisons as possible. The i-th smallest element, denoted by $i \theta X$, is that element which is larger than exactly i-1 other elements, so that $1 \theta X$ is the smallest, and $n \theta X$ the largest, element in X .

Let f(i,n) denote the expected number of comparisons required to select $i \theta X$. Since a selection algorithm must determine, for every $t \epsilon X$, $t \neq i \theta X$, whether $t < i \theta X$ or $i \theta X < t$, we have as a trivial lower bound

$$f(i,n) \geq n-1 , \quad \text{for } 1 < i < n . \tag{1}$$

The best previously published selection algorithm is FIND, by C. A. R. Hoare [1]. Knuth [2] has determined the average number of comparisons used by FIND, thus proving that

$$f(i,n) \leq 2((n+1)H_n - (n+3-i)H_{n-i+1} - (i+2)H_i + n + 3) , \tag{2}$$

where

$$H_n = \sum_{1 \leq j \leq n} j^{-1} . \tag{3}$$

This yields as special cases

$$f(1,n) \leq 2n + o(n) , \tag{4}$$

and

$$f(\lceil n/2 \rceil, n) < 2n(1 + \ell n(2)) + o(n) \leq 3.39 n + o(n) . \tag{5}$$

No bounds better than (1) or (2) have previously been published.

In Section 2 we present our new selection algorithm, SELECT, and derive by an analysis of its -efficiency the upper bound

$$f(i,n) \le n + \min(i,n-i) + \mathcal{O}(n^{2/3} \ln^{1/3}(n)) \qquad (6)$$

A small modification to SELECT is then made, yielding the slightly improved bound

$$f(i,n) \le n + \min(i,n-i) + \mathcal{O}(n^{1/2}) \quad . \qquad (7)$$

An implementation of SELECT is given in Section 3 with timing results for both SELECT and FIND.

The authors believe that SELECT is asymptotically optimal in the sense that the function

$$F(\alpha) \underset{\text{def}}{=} \limsup_{n \to \infty} \frac{f(\lfloor \alpha(n-1) \rfloor + 1, n)}{n} \quad , \quad 0 \le \alpha \le 1 \qquad (8)$$

is bounded below by the analogue of the right-hand side of (7), so that

$$F(\alpha) \ge 1 + \min(\alpha, 1-\alpha) \quad , \quad \text{for } 0 < \alpha < 1 \ . \qquad (9)$$

A lower bound just a little better than $1 + .75 \min(\alpha, 1-\alpha)$ is derived in Section 4, within 9% of our conjecture and the performance of SELECT.

In what follows $t \rho X$ will denote the rank of an element $t \epsilon X$, so that $(t \rho X) \theta X = t$. $E()$ will denote the expected value of its argument, and $P()$ will denote the probability of an event.

2. The Algorithm SELECT

The algorithm SELECT utilizes sampling. A small random sample S of size $s = s(n)$ is drawn from X . Two elements, u and v , $(u < v)$, are selected from S , using SELECT recursively, such that the interval $[u,v]$ is quite small, yet is expected to contain $i \theta X$. Selecting u and v partitions S into those elements less than u (set A), those elements between u and v (set B), and those elements greater than v (set C). The partitioning of X into these three sets is then completed by comparing each element x in X-S to u and v . If $i < \lceil n/2 \rceil$, x is compared to v first, and then to u only if $x < v$. If $i \geq \lceil n/2 \rceil$, the order of the comparisons is reversed. With probability approaching 1 (as $n \to \infty$), $i \theta X$ will lie in set B , and the algorithm is applied recursively to select $i \theta X$ from B. (Otherwise SELECT is applied to A or C as appropriate.)

If $s(n)$, u , and v can be chosen so that $s(n) = o(n)$, $E(|B|) = o(n)$, and $P(i \theta X \notin B) = o(n^{-1})$, then the total work expected is:

$\mathcal{O}(s(n))$	to select u and v from S ,
$+ \ (n-s(n))(1+(\min(i,n-i)+o(n))/n)$	to compare each element in X-S to u , v ,
$+ \ \mathcal{O}(\lvert B \rvert)$	to select $i \theta X$ from B ,
$+ \ o(1)$	to select $i \theta X$ from A or C .
$= n+\min(i,n-i)+o(n)$	comparisons total.

This can in fact be done; the low order term is $\mathcal{O}(n^{2/3} \ln^{1/3}(n))$. Figure 1 shows a geometric analogy of the procedure SELECT.

It is not hard to show (see [3]) that for any $t \in S$ we have

$$E(t \ \rho X) = \frac{(n+1)}{(s+1)}(t \ \rho \ S) \ , \qquad (10)$$

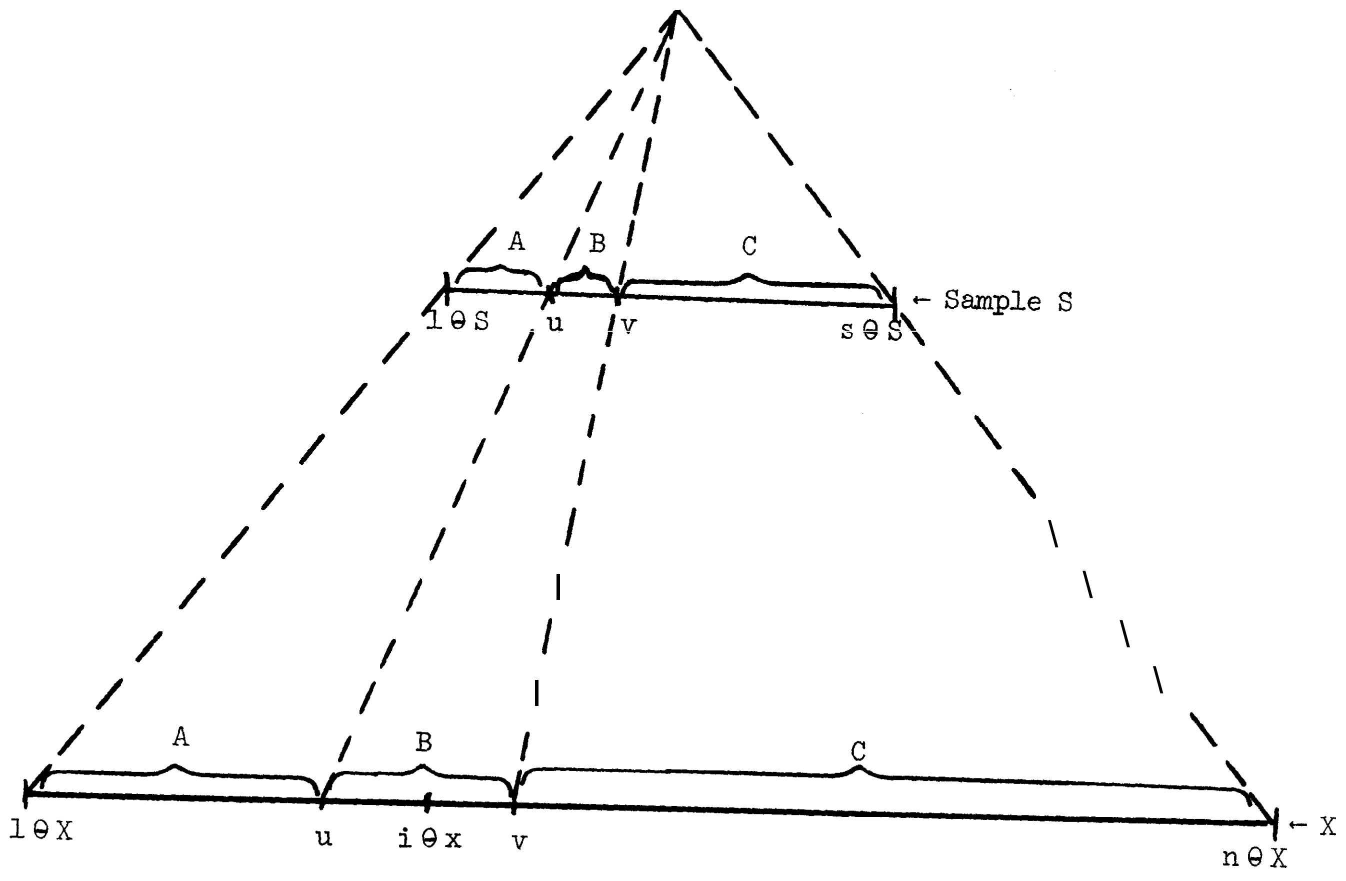


Figure 1

$$\sigma(t\,\rho X) = \sqrt{\frac{(t\,\rho S)(s - t\,\rho S - 1)(n+1)(n-s)}{(s+1)^2(s+2)}} \tag{11}$$

$$\leq \frac{1}{2}\sqrt{\frac{(n+1)(n-s)}{s}} \leq \frac{1}{2}\,\frac{n}{\sqrt{s}} \quad .$$

We wish to choose u and v so that $E(u\,\rho X) \leq i \leq E(v\,\rho X)$, $E(|B|) = E(v\,\rho X) - E(u\,\rho X)$ is $o(n)$, and $P(i < u\,\rho X \text{ or } i > v\,\rho X) = o(n^{-1})$. To do this we choose $u\,\rho S$ and $v\,\rho S$ so that

$$E(u\,\rho X) + 2\,d\,\sigma(u\,\rho X) \cong i \cong E(v\,\rho X) - 2\,d\,\sigma(v\,\rho X) \quad , \tag{12}$$

where $d = d(n)$ is a slowly growing unbounded function of n . In fact, since

$$2\cdot\int_d^\infty \mathrm{erf}(x)dx \leq \frac{ce^{-d^2}}{d} \quad , \quad \text{for some constant } c \ , \tag{13}$$

we will choose $d = \sqrt{\ell n(n)}$. This ensures that $P(i < u\,\rho X \text{ or } i > v\,\rho X) = o(n^{-1})$. The above equations mean that

$$u\,\rho S \cong \left(i - d\cdot\sqrt{\frac{(n+1)(n-s)}{s}}\right)\left(\frac{s+1}{n+1}\right) \geq \frac{i\cdot s}{n} - d\sqrt{s}$$

and

$$v\,\rho S \cong \left(i + d\cdot\sqrt{\frac{(n+1)(n-s)}{s}}\right)\left(\frac{s+1}{n+1}\right) \leq \frac{i\cdot s}{n} + d\sqrt{s} \quad . \tag{14}$$

Let g(i,n) denote the expected number of comparisons made by SELECT. It will be shown inductively that

$$g(i,n) = n + \min(i,n-i) + O(n^{2/3}\,\ell n^{1/3}(n)) \quad . \tag{15}$$

The above is true for all n less than some fixed N , so the basis for induction is clearly satisfied. We proceed with the inductive step by determining the cost of SELECT as a function of s(n) and n , and then optimizing the choice of s(n) .

The cost of selecting u and v can be estimated as follows. First we apply SELECT recursively to S to select u , then we extract v from those elements of S which are greater than u . (Note that selecting u means determining which elements of S are greater than u as well.) These two operations cost

$$\begin{aligned} &g(u\,\rho\,S, s) + g(v\,\rho\,S - u\,\rho\,S + 1, s \quad -ups) \\ &\le 2s + v\,\rho\,S - u\,\rho\,S + \mathcal{O}(s^{2/3}\,\ell n^{1/3}(s)) \\ &\le 2s + 2d\sqrt{s} + \mathcal{O}(s^{2/3}\,\ell n^{1/3}(s)) \end{aligned} \tag{16}$$

comparisons.

The cost of comparing each element in X-S to u and v is easy to compute. There are n-s(n) elements to compare, and the probability that two comparisons will be made for an element is just $\min(u\,\rho\,S, s+1-u\,\rho\,S)/(s+1)$, so that the total is

$$(n-s(n))(1+\min(i,n-i)/n+ds^{-1/2}) \ . \tag{17}$$

The cost of finishing up, if $i\,\theta\,X$ falls in B , is at most $g(|B|/2, |B|)$. But

$$E(|B|) \approx (v\,\rho\,S - u\,\rho\,S)n/s = 2dns^{-1/2} \tag{18}$$

so that

$$g(|B|/2, |B|) = 3dns^{-1/2} + \mathcal{O}((dns^{-1/2})^{2/3}(\ell n(dns^{-1/2}))^{1/3}) . \tag{19}$$

On the other hand, if $i\,\theta\,X$ falls in A or C , the expected cost of finishing up is at most $3n/2$, and the probability that $i\,\theta\,X \in A$ or $i\,\theta\,X \in C$ is, from (13), less than $c/(dn)$, so that the total work expected in this case is less than $3c/(2d)$, which goes to zero as $n \to \infty$.

The total expected cost of SELECT is thus

$$\begin{aligned} g(i,n) &\le 2s + 2d\sqrt{s} + \mathcal{O}(s^{2/3}\, \ell n^{1/3}(s)) \\ &\quad + (n-s)(1+\min(i,n-i)/n+ds^{-1/2}) \\ &\quad + 2d\,n\,s^{-1/2} + 3c/(2d) \\ &\le n + \min(i,n-i) + s + d\sqrt{s} - \min(i,n-i)s/n \\ &\quad + 3d\,n\,s^{-1/2} + 3c/(2d) + \mathcal{O}(s^{2/3}\, \ell n^{1/3} s) \ . \end{aligned} \tag{20}$$

The principal increasing and decreasing terms in s in this expression are s and $3d\,n\,s^{-1/2}$. Choosing $s(n)$ to make them equal will approximately minimize $g(i,n)$. Thus we choose

$$s(n) \sim n^{2/3}\, \ell n^{1/3}(n) \tag{21}$$

which, together with (20), yields (15), which was to be proved. This completes the analysis of SELECT.

We now introduce a small modification to SELECT in order to reduce the second-order term to the promised $\mathcal{O}(n^{1/2})$. Let $S_1 \subset S_2 \subset \ldots \subset S_k = X$ be a nested series of random samples from X of sizes $s_1, s_2, \ldots, s_k = n$. For each sample S_j, let u_j and v_j be chosen from S_j as in (14) so that

$$u_j \,\rho\, S_j = \left(i - d\sqrt{\frac{(n+1)(n-s_j)}{s_j}} \right) \cdot \left(\frac{s_j+1}{n+1} \right)$$

and

$$v_j \,\rho\, S_j = \left(i + d\sqrt{\frac{(n+1)(n-s_j)}{s_j}} \right) \cdot \left(\frac{s_j+1}{n+1} \right) . \tag{22}$$

Thus it is very likely, for any j, that $u_j \,\rho X \le i \le v_j \,\rho X$. Furthermore, as j approaches k (i.e., as s_j gets large), u_j and v_j surround $i \theta X$ ever more closely. In fact, $u_k = i \theta X = v_k$. The cost of finding u_j and v_j directly from S_j is of course prohibitive for large values of s_j. However, since

$$E(u_{j-1}\,\rho\,S_j) = (u_{j-1}\,\rho\,S_{j-1})\cdot\frac{s_j+1}{s_{j-1}+1} \le u_j\,\rho\,S_j\,, \qquad (23)$$

And similarly $E(v_{j-1}\,\rho\,S_j) \ge v_j\,\rho\,S_j$, we can use u_{j-1} and v_{j-1} to bound the search for u_j and v_j. See Figure 2 for a graphical representation of the modified SELECT.

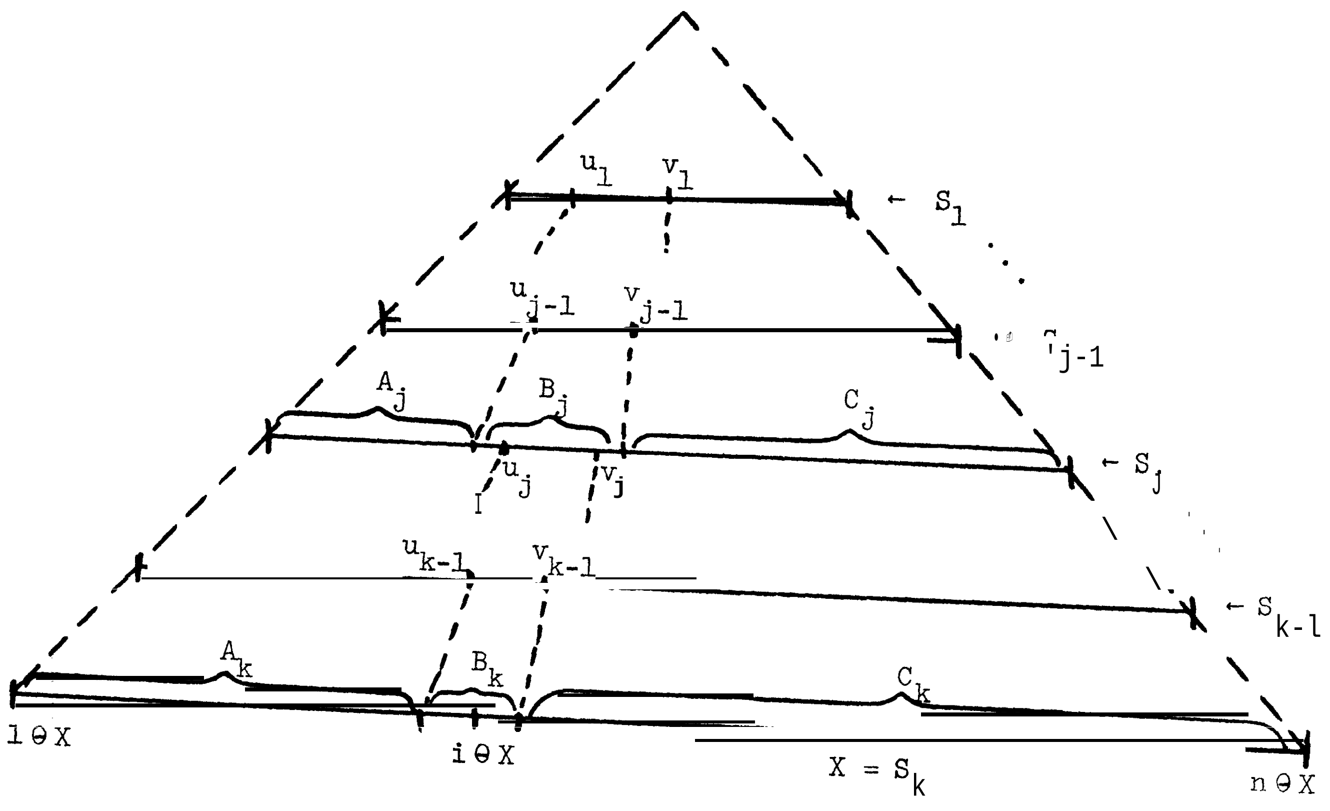


Figure 2

The modified algorithm runs as follows. Draw a random sample S_1 of size s_1 from X , and select u_1 and v_1 using this algorithm recursively (and the ranks given in (22)). Determine the sets A_2, B_2, and C_2, a partition of S_2, by comparing each element in S_2-S_1 to u_1 and v_1 (using the same order of comparison strategy as the original SELECT). Next, determine u_2 and v_2 by applying this algorithm recursively to B_2 (in the most likely case; else A_2 or C_2). Extend the partition of S_2 determined by u_2 and v_2 into a partition A_3, B_3, C_3 of S_3 by comparing each element of S_3-S_2 to u_2 and v_2 with the same comparison strategy. Continue in this fashion until a partition A_k, B_k, C_k of the set $S_k = X$ has been created. Then use the algorithm recursively once more to extract $i \theta X$ from B_k (or A_k or C_k, if necessary).

This "bootstrapping" algorithm has the advantage that the expense of computing a good bounding interval $[u_j, v_j]$ for $i \theta X$ is reduced by first computing at a fraction of the cost the less tight bounding interval $[u_{j-1}, v_{j-1}]$. We keep $d(n) = \ell n^{1/2}(n)$ as before, to ensure that the probability that $i \theta X$ is not in $[u_j, v_j]$ is of order $o(n^{-1})$. The probability that u_j or v_j is not in the interval $[u_{j-1}, v_{j-1}]$ is also negligible, since

$$\sigma(u_{j-1} \rho S_j) \leq \frac{s_j}{2\sqrt{s_{j-1}}} \tag{24}$$

and

$$E(u_j \rho S_j - u_{j-1} \rho S_j) = \frac{d \cdot (s_j - \sqrt{s_j})}{\sqrt{s_{j-1}}} \quad . \tag{25}$$

To compute the cost of the algorithm, we assume inductively, as before, that

$$g(j,m) = m + \min(j,m-j) + \mathcal{O}(m^{1/2}) \quad ,$$

$$\text{for } m < n \ , \quad 1 \le j \le m \ . \tag{26}$$

The expected size of B_j is easily estimated:

$$E(|B_j|) = (v_{j-1} \rho S_{j-1} - u_{j-1} \rho S_{j-1}) \cdot \left(\frac{s_j}{s_{j-1}} \right) \tag{27}$$

$$\le 2 d s_j / \sqrt{s_{j-1}} \quad .$$

The cost of selecting $u_2, v_2, \ldots, u_{k-1}, v_{k-1}$ from the sets $B_2, \ldots$ [illegible] is just

$$\sum_{2 \le j \le k-1} (g(u_j \rho B_j, |B_j|) + g(v_j \rho B_j - u_j \rho B_j + 1, |B_j| - u_j \rho B_j))$$

$$\le \sum_{2 \le j \le k-1} (4 d s_j / \sqrt{s_{j-1}} + 2 d \sqrt{s_j}) \ , \tag{28}$$

whereas the cost of selecting u_1 and v_1 from S_1 is less than

$$2s_1 + 2 d \sqrt{s_1} \quad , \tag{29}$$

while the cost of selecting $u_k = v_k = i \theta X$ from B_k is at most

$$3 d n / \sqrt{s_{k-1}} \quad . \tag{30}$$

The cost of partitioning $S_2 - S_1, S_3 - S_2, \ldots, S_k - S_{k-1}$ about u_1 and v_1, u_2 and v_2, . . . , u_{k-1} and v_{k-1} is just

$$\sum_{2 \le j \le k} (s_j - s_{j-1})(1 + \min(i,n-i)/n + d/\sqrt{s_{j-1}}) \quad . \tag{31}$$

Adding these all together, we have

$$g(i,n) \le n+\min(i,n-i)+\sum_{2\le j\le k}\left(5\,d\,s_j/\sqrt{s_{j-1}}+d\sqrt{s_j}\right)$$

$$+\,s_1(1-\min(i,n-i)/n)+d\sqrt{s_1}-dn/\sqrt{s_{k-1}}\ . \qquad (32)$$

This sum can be approximately minimized if we let $s_1, s_2, \ldots, s_k$ increase geometrically with ratio r^2, so that $s_j = r^{2j-2} s_1$, and

$$g(i,n) \le n+\min(i,n-i)+\left(\frac{5d}{\sqrt{s_1}}+\frac{\sqrt{s_1}}{r}\right) \bullet$$ [illegible] r_j

$$\le n+\min(i,n-i)+\left(\frac{5d}{\sqrt{s_1}}+\frac{\sqrt{s_1}}{r}\right)\cdot\left(\frac{r^{k-1}-1}{r-1}\right)\cdot r^2$$

$$\le n+\min(i,n-i)+\sqrt{n}\left(\frac{r^2}{r-1}\right)\left(\frac{5d}{s_1}+\frac{1}{r}\right)\ . \qquad (33)$$

This is approximately minimized when $s_1 = \ell n^{1/2} n$, and $r = 4.32$, yielding

$$g(i,n) \le n+\min(i,n-i)+O(n^{1/2})\ , \qquad (34)$$

which was to be shown.

3. Implementation and Timing Results

In this section we present an ALGOL implementation of SELECT (a revised form of the simpler version given in Section 2), and give timing results that demonstrate that our theoretical results yield fruit in practice.

We assume that it is desired to have the same input-output relationships as FIND . That is, we are given an array segment X[L:R] and an integer K such that $L \le K \le R$; we wish to rearrange the values in X[L:R] so that X[K] contains the (K-L+1)-th smallest value, $L \leq I \leq K$ implies $X[I] < X[K]$, and $K < I < R$ implies $X[I] \geq X[K]$. An implementation of the complicated version of SELECT given in Section 2 will not be given, since no advantage is obtained over the simpler version except for unrealistically large values of n .

The innermost loop of the algorithm is obviously the partitioning operation. Any reduction in the complexity of partitioning will show up as a significant increase in the efficiency of the whole algorithm. The basic algorithm, however, requires partitioning X about both u and v simultaneously into the three sets A , B , and C , an inherently inefficient operation. On the other hand, partitioning X completely about one of u , v before beginning the partition about the other can be done very fast. We therefore use an improved version of Hoare's PARTITION algorithm [1] to do the basic partitioning, A further (minor) difference is that after partitioning has been completed about one element another sample is drawn to determine the next element about which to partition. This permits a very compact control structure at little extra cost.

A listing of the procedure as written in ALGOL 60 is given on page 27. The element T about which to partition is first determined. It was found experimentally that sampling was worthwhile only for values of N (the size of input set) greater than 600 . This is due to the expense of computing square-roots, logarithms, etc., which cost more than they are worth for small N ! If sampling is performed, the recursive call to SELECT leaves the desired element T in X[K] ; if sampling is not done, the algorithm partitions about whatever was in X[K] initially (this'is good if X was already sorted). The partitioning phase is initialized to obviate subscript range checking.

Note that there is really no good way to avoid re-partitioning the sample or at least moving most of it later, but having it located around X[K] probably minimizes the number of exchanges necessary. Since one of L , R change each iteration, the number of elements remaining always decreases by at least one, thus ensuring termination.

Timing results were then obtained for FIND (exactly as published) and SELECT (as given on page 27). The testing was done in SAIL (an ALGOL dialect) on the PDP-10 at Stanford's Artificial Intelligence Laboratory. These results are given in the description of the algorithm on page 27.

SELECT clearly outperforms FIND. This results from a slightly faster partitioning scheme combined with a large reduction in the partitioning required due to the effective use of sampling.

4. Lower Bounds for F(a)

In this section we present new lower bounds for the expected number of comparisons required for selection. Although we believe SELECT to be (first-order) asymptotically optimal, we have been unable to derive a lower bound for $F(\alpha)$ equal to the upper bound of $1+\min(\alpha,1-\alpha)$ produced by our analysis of SELECT. The bounds derived here are within 9% of that value, for all α , though, and the strength of these results relative to the weakness of our methods lends support to our conjecture.

We will define a sequence $F_j(\alpha)$, for $0 \le j \le \infty$, of lower bounds for F(a) such that $F_j(\alpha) \le F_{j+1}(\alpha)$, for all $j \ge 0$ and α , $0 \le \alpha \le 1$. The functions $F_0(\alpha)$, $F_1(\alpha)$, $F_2(\alpha)$, and $F_3(\alpha)$ have been computed -- the function $F_3(\alpha)$ thus being our best lower bound for F(a) . These bounds have been plotted against α in Figure 3. The value of $F_3(\alpha)$ at $\alpha = .5$ is 1.375 , which tapers off as α approaches 0 or 1, essentially becoming identical with $1+\min(\alpha,1-\alpha)$ near the extremes.

We first prove a basic result.

Theorem 1. Any selection algorithm that has determined $i \theta X$ to be some element $y \epsilon X$ must also have determined, for any $x \epsilon X$, $x \neq y$, whether $x < y$ or $y < x$.

Proof. Assume that there exists an x incomparable with y in the partial order determined by the algorithm. Then there exists a linear ordering of X , consistent with the partial order determined, in which x and y are adjacent (since any element required to lie between x and y would imply a relationship between x and y in the partial order). But then x and y may be interchanged in the linear order without contradicting the partial order -- demonstrating an uncertainty of at least one in $y \rho X$, so that y is not necessarily $i \Theta X$.

Q.E.D.

The following definition provides the basis for the lower bound computations.

Definition 1. The key comparison for an element $x \in X$, $x \neq i \Theta X$, is defined to be the first comparison $x:y$ such that

$$y = i \Theta X \quad \text{or} \quad x < y < i \Theta X \quad \text{or} \quad i \Theta X < y < x . \tag{35}$$

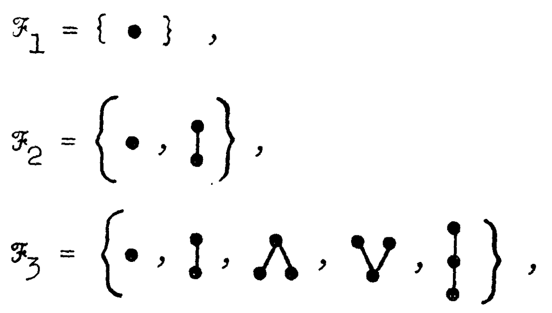
Note that determining which comparison is the key comparison for x can in general only be done after all. the comparisons have been made and $i \Theta X$ has been selected. Each element x, $x \neq i \Theta X$, must have a key comparison, otherwise x would be incomparable with $i \Theta X$, a contradiction by Theorem 1. This proves

Lemma 1. A selection algorithm must make exactly n-l key comparisons to select $i \Theta X$, where $|X| = n$.

We now define two more essential concepts.

Definition 2. A fragment of a partial ordering $(X, \leq)$ is a maximal connected component of the partial ordering, that is, a maximal subset $S \subseteq X$ such that the Hasse diagram of " < " restricted to S is a connected graph.

Any partial ordering can be uniquely described up to isomorphism as the union of distinct fragments. A selection algorithm thus begins with a partial ordering consisting of n fragments of size 1 . To illustrate, let $\mathcal{F}_k$ be the set of all fragments having at most k elements:



and so on.

<u>Definition 5</u>. A <u>Joining comparison</u> is any comparison between elements belonging to distinct fragments.

Note that each joining comparison reduces the total number of fragments by one, implying the following.

<u>Lemma 2</u>. A selection algorithm must make exactly n-1 joining comparisons to select $i \ominus X$, where $|X| = n$.

<u>Proof</u>. As long as more than one fragment exists, there must be some element incomparable with $i \ominus X$, since elements in distinct fragments are incomparable. The lemma then follows from Theorem 1.

Our lower bounds will be derived from the conflicting requirements of lemmas 1 and 2 -- a selection algorithm can not in general have all of its joining comparisons be key comparisons, or vice versa. In fact, the authors make the following conjecture:

<u>Conjecture.</u> Asymptotically (as $n \to \infty$), the average probability that a joining comparison will turn out to be a key comparison is at most

$$\max(\alpha, 1-\alpha) \quad . \qquad (36)$$

We must use the asymptotic average probability, since near the end of an algorithm, the probability of a particular joining comparison being a key comparison may easily exceed (36). This happens because near the end there are often elements with a significant probability of actually being $i \theta X$, and a comparison with one of these elements can have a somewhat larger probability *of* turning out to be key. As an example, consider the comparison of a previously uncompared element x with an element y which is known to be the i-th smallest of the remaining n-1 elements. Then

$$\begin{aligned} P(x{:}y \text{ is key}) &= P(y = i\theta X < x) + P(x = i\theta X < y) \\ &= (n-i+1)/n \ , \end{aligned} \tag{37}$$

which, for $\alpha < 1/2$, is a little larger than $\max(\alpha, 1-\alpha) = 1-\alpha = (n-i+1)/(n+1)$.

Unfortunately, we could not find a proof of our conjecture, which would imply the optimality of Select for all values of α . Our results stem therefore from an analysis of only those joining comparisons in which at least one of the fragments being joined is small. We are left with just a *small* finite number of cases (i.e., possible types of joining comparisons) to consider, since we will not distinguish between the various kinds of large fragments that might participate in a joining comparison. We want to estimate, for each type of joining comparison, the probability that it will turn out to be a key comparison. These probabilities will then be used in an interesting way to derive a lower bound for F(a) .

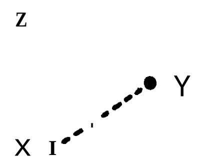
As noted above, the probability that a joining comparison will turn out to be a key comparison is certainly affected by the probability that one of the elements being compared is actually $i \theta X$. The following argument shows that we may treat this latter probability as being negligible, for large n . Given some ε , $0 < \varepsilon < 1$, it is easy to see that there exists an integer m such that the maximum probability that any element $x \in X$ is actually $i \Theta X$ is at most ε if the largest fragment has size at most $n-m'$. For if x is incomparable with m elements from other fragments, then it has a chance of being $i \Theta X$ of at most

$$P(x = i \Theta X) \leq (2\pi m \alpha(1-\alpha))^{-1/2} \tag{38}$$

which is less than ε for $m > (2\pi\alpha(1-\alpha)\varepsilon^2)^{-1}$. So except for a finite number of comparisons near the end, the probability that any element is $i \Theta X$ is at most ε . As $n \to \infty$, these latter comparisons form a negligible proportion of the total number of comparisons made, and their effect on the probability that an average joining comparison will be a key comparison becomes insignificant. We will therefore assume from now on that the probability that either element being compared is $i \Theta X$ is zero.

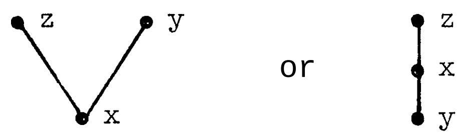
To derive $F_k(\alpha)$ we need to compute the probability that each joining comparison in which the smaller fragment has at most k elements will turn out to be a key comparison. These comparisons can be divided into two types: those for which both fragments belong to $\mathcal{F}_k$, and those for which only one fragment has k or fewer elements. The first case is somewhat simpler to handle so we shall treat it first, by means of an example.

Consider the comparison of the smaller of a pair of elements $x < z$, to an isolated element y :



(39)

As a result of this comparison, we will end up with either



. (40)

The probabilities of these two outcomes are not equal -- the first occurs with probability 2/3 while the second occurs with probability 1/3 . This happens because the first outcome is consistent with the two permutations $x < y < z$ and $x < z < y$, whereas the second outcome is only consistent with $y < x < z$. Since each permutation consistent with the input fragments is equally likely, the probability of each outcome is proportionalto the number of permutations consistent with that outcome.

We must now consider each permutation consistent with the input fragments separately, since to determine whether $x{:}y$ is a key comparison requires knowing the relative order of x , y , $i\theta X$, and all elements previously compared to either x or y . Let us consider the permutation $x < y < z$ first, consistent with the first outcome. With respect to $i\theta X$, these three elements may be in one of four positions. That is, $i\theta X$ may be greater than from zero to three of these three elements. In only two of these cases will $x{:}y$ turn out to be a key comparison:

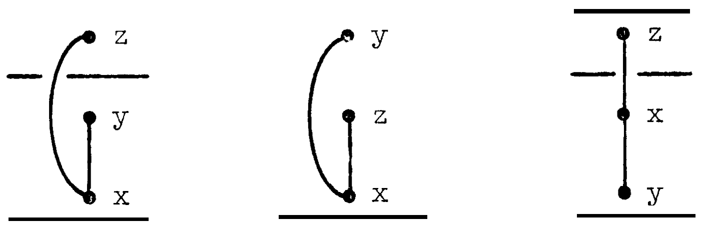
(i) $i\theta X < x < y < z$ -- this will be a key comparison for y ,

(ii) $x < i\theta X < y < z$ -- this will not be a key comparison,

(iii) $x < y < i\theta X < z$ -- this will be a key comparison for x ,

(iv) $x < y < z < i\theta X$ -- this will not be a key comparison, since x has already been compared to z .

The probability of each of these four cases occurring, given that $x < y < z$, follows the binomial distribution with $p = \alpha$, so that case (i) occurs with probability $(1-\alpha)^3$ and case (iii) occurs with probability $3\alpha^2(1-\alpha)$. The analysis of all three permutations consistent with (39) can be represented graphically, using horizontal lines to indicate the relative positions of $i \in X$ that make $x:y$ a key comparison:



(41)

The total probability that $x:y$ turns out to be a key comparison is thus the average probability that $x:y$ is a key comparison in each of these three cases. This is just (finally!):

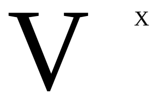
$$P(x:y \text{ is key}) = (1-\alpha)^3 + 2\alpha^2(1-\alpha) + \frac{\alpha^3}{3} . \qquad (42)$$

Whenever both fragments are small, the probability of a comparison joining them turning out to be key can be computed in the above fashion. This completes our description of the analysis of a comparison joining two small fragments.

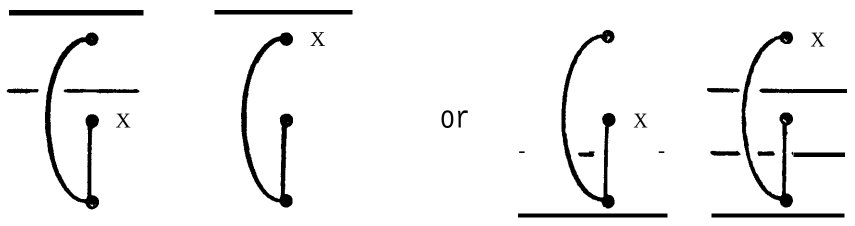
When an element x belonging to a small fragment is compared to an element y from an arbitrary fragment having more than k elements, the analysis can not be done in the above fashion since we essentially know nothing about y; its probability distribution and probability of already having had a key comparison must remain totally unspecified. It is still possible, however, to derive an upper bound on the probability that the comparison $x:y$ will turn out to be a key comparison, since if x and y fall on different sides of $i \ominus X$ the comparison can not be a key comparison. It is thus easy to see that

$$P(x:y \text{ is key}) \leq \max(P(x < i \ominus X), P(x > i \ominus X)) \quad . \qquad (43)$$

For example, to compare x of the fragment:

 (44)

against an arbitrary y , the case analysis can be represented graphically as before, using a horizontal line to indicate the relative position of $i \ominus X$ making a key comparison possible:



(45)

for $x < i \ominus X$ for $x > i \ominus X$

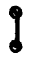
We have then directly from (43) and (45)

$$P(x:y \text{ is key}) \leq \max(\alpha^3 + 3\alpha^2(1-\alpha)/2, (1-\alpha)^3 + 3\alpha(1-\alpha)^2 + 3\alpha^2(1-\alpha)/2) \ . \qquad (46)$$

This kind of analysis is simple to carry out for an x belonging to any small fragment, so that we now have ways of computing (an upper bound for) the probability that <u>any</u> comparison joining a small fragment to another fragment will turn out to be a key comparison.

We will now describe how specific results such as (46) and (42) above can be combined to derive $F_k(\alpha)$. We will assign a weight to a partial ordering which is a lower bound on the expected number of non-key joining comparisons yet to be made in selecting $i \Theta X$. The total number of comparisons made on the average is thus bounded below by n-l (for the joining comparisons) + the weight of the partial ordering (to ensure that n-l key comparisons are made as well). The weight of a partial ordering is defined to be the sum of the weights of its constituent fragments. The weight of a fragment will be computed from the specific probability results already calculated by means of a linear programming technique.

What we want is to ensure that the expected weight of a partial ordering does not decrease as a result of a joining comparison by more than the--probability that that joining comparison was non-key. This guarantees that the weight of the initial partial ordering is a valid lower bound for the expected number of non-key joining comparisons made. Since we only have data for those fragments with k or fewer elements, only those fragments will be assigned *positive* weights -- all larger fragments will have weight zero. (In particular, the weight of the final partial ordering, in which $i \Theta X$ has been determined, must be zero.)

Let us consider the computation of $F_2(\alpha)$ as an example. Let w_1 be the weight of the fragment $\bullet$ and let w_2 be the weight of  .
The weight of the initial partial ordering is therefore just $n w_1$. We want to maximize w_1 subject to the constraints imposed by our previous computations about specific kinds of comparisons. For example, a comparison between two isolated elements is non-key with probability $2\alpha(1-\alpha)$, yielding the inequality:

$$2w_1 - w_2 \leq 2\alpha(1-\alpha) \quad . \tag{47}$$

Comparing an isolated element against an arbitrary element from a fragment with more than two elements yields the inequality

$$w_1 \leq \min(\alpha, 1-\alpha) \quad . \tag{48}$$

A computer program was written to generate all the relevant inequalities like (47) and (48) for a given k . Note that when two fragments are being joined such that two different outcomes are possible, both in $\mathcal{F}_k$, the probability of each outcome must be considered when computing the expected weight of the resultant fragment after the comparison has been made. The linear programming algorithm MINIT of Salazar and Sen [4] was used to determine the maximum weight w_1 possible for the isolated element. The value $1+w_1$ is then our lower bound for $F(\alpha)$.

When k= 1 the solution takes a particularly simple form:

$$F(a) \geq F_1(\alpha) = 2\alpha(1-\alpha) \quad . \tag{49}$$

The functions $F_2(\alpha)$ and $F_3(\alpha)$ are too complicated to give here, but are as plotted in Figure 3. For the case of computing medians they reduce to

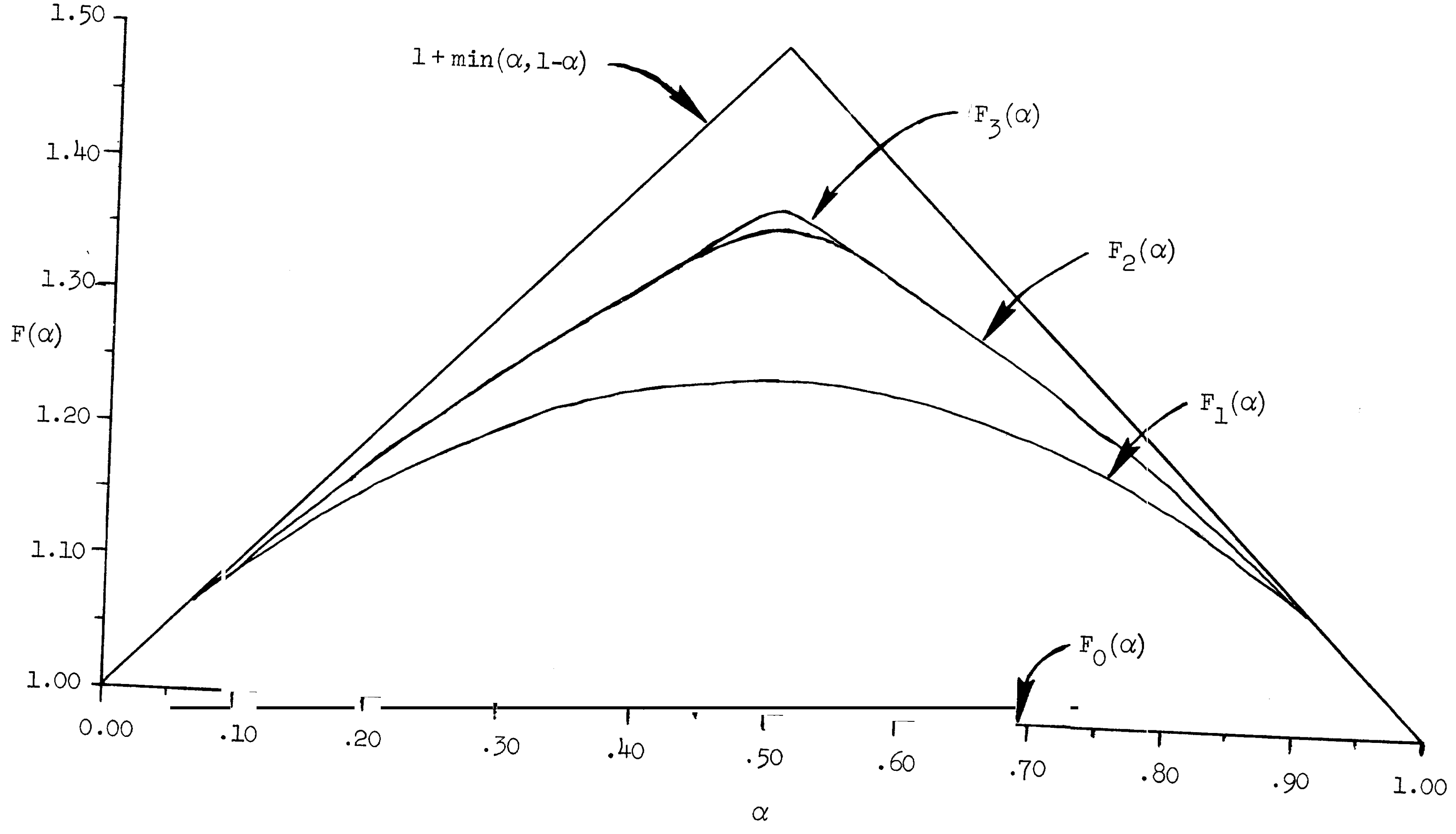
$$F_2(\tfrac{1}{2}) = \frac{49}{36}n \tag{50}$$

and

$$F_3(\tfrac{1}{2}) = \frac{11}{8}n \quad , \tag{51}$$

which is within 9% of the best possible value of 1.5 n .

This completes the description of our lower bound derivations. The results show that SELECT is at least near-optimal, and we suspect that a more powerful combinatorial analysis would demonstrate optimality. The weakness in our method lies in the restricted nature of the


1.50
1.40
1.30
1.20
1.10
1.00
$F(\alpha)$
$1+\min(\alpha,1-\alpha)$
$F_3(\alpha)$
$F_2(\alpha)$
$F_1(\alpha)$
$F_0(\alpha)$
0.00
.10
.20
.30
.40
.50
.60
.70
.80
.90
1.00
α

Figure

inequalities derivable for the case of a comparison between a small fragment and an arbitrary fragment belonging to a large fragment. In any case these lower bounds are the first non-trivial lower bounds published for this problem.

References

[1] Hoare, C. A. R. "Algorithm 63 (PARTITION)" and "Algorithm 65 (FIND)," C.ACM 4 (July 1961), 321.

[2] Knuth, Donald E. "Mathematical analysis of algorithms," Computer Science Dept. Report STAN-CS-71-206. Stanford University (March 1971). 27 pp.

[3] Lindgren, B. W. Statistical Theory. The MacMillan Co., New York (1962).

[4] Salazar, Rodolfo C. and Subrata K. Sen. "Algorithm 333 (MINIT algorithm for linear programming," C.ACM 11 (June 1968), 437-440.

The Algorithm SELECT - for finding the i-th smallest of n elements

by Robert W. Floyd and Ronald L. Rivest
Stanford Computer Science Department
Stanford University
Stanford, California 94305

Keywords: selection, medians, quantiles

CR Categories: 5.30, 5.39

DESCRIPTION: SELECT will rearrange the values of an array segment X[L:R] so that X[K] (for some given K ; $L \leq K \leq R$) will contain the (K-L+1)-th smallest value, $L \leq I \leq K$ will imply $X[I] \leq X[K]$, and $K \leq I \leq R$ will imply $X[I] \geq X[K]$. While SELECT is thus functionally equivalent to Hoare's algorithm FIND [1], it is significantly faster on the average due to the effective use of sampling to determine the element T about which to partition X . The average time over 25 trials required by SELECT and FIND to determine the median of n elements was found experimentally to be:

n	500	1000	5000	10000
SELECT	89 ms.	141 ms.	493 ms.	877 ms.
FIND	104 ms.	197 ms.	1029 ms.	1964 ms.

The arbitrary constants 600 , .5 , and .5 appearing in the algorithm minimize execution time on the particular machine used. SELECT has been shown to run in time asymptotically proportional-to N+min(I,N-I) , where N = L-R+1 and I = K-L+1 . A lower bound on the running time within 9% of this value has also been proved. [2]

REFERENCES:

[1] Hoare, C. A. R. "Algorithm 63 (PARTITION)" and "Algorithm 65 (FIND)", CACM 4 (July 1961), 321.

[2] Floyd, Robert W. and Ronald L. Rivest. "Expected Time Bounds for Selection," Stanford CSD Report No. 349 (April, 1973).

ALGORITHM:

```
procedure SELECT(X,L,R,K); value L,R,K; array X;
begin integer N,I,J,S,SD,LL,RR,T; real Z;
 while R > L do begin
   if R-L > 600 then begin
     comment Use SELECT recursively on a sample of size S to get an
       estimate for the (K-L+1)-th smallest element into X[K], biased
       slightly so that the (K-L+1)-th element is expected to lie in
       the smaller set after partitioning;
    N := R-L+1;
    I := K-L+1;
    Z := ln(N);
    S := .5 * exp(2*Z/3);
    SD := .5 * sqrt(Z*S*(N-S)/N) * sign(I-N/2);
    LL := max(L,K-I*S/N+SD);
    RR := min(R,K+(N-I)*S/N+SD);
    SELECT(X,LL,RR,K)
  end;
  T := X[K];
  comment The following code partitions X[L:R] about T. It is similar
    to PARTITION but will run faster on most machines since subscript
    range checking on I and J has been eliminated.;
  I := L;
  J := R;
```

```
          exchange(X[L],X[K]);
          if X[R] > T then exchange(X[R],X[L]);
          while I < J do begin
            exchange(X[I],X[J]);
            I := I+1;
            J := J-1;
            while X[I] < T do I := I+1;
            while X[J] > T do J := J-1;
         end;
         if X[L] = T then exchange(X[L],X[J])
           else begin J := J+1; exchange(X[J],X[R]) end;
         comment Now adjust L, R so they surround the subset containing
           the (K-L+1)-th smallest element;
        if J ≤ K := J+1;
        if K ≤ J then R := J-1;
      end
    end SELECT
```