# Expected Time Bounds for Selection

Robert W. Floyd and Ronald L. Rivest
Stanford University

**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 percent of the above formula is also derived.**

**Key Words and Phrases: selection, computational complexity, medians, tournaments, quantiles**

**CR Categories: 5.30, 5.39**

## 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 $X$.



This paper gives the theoretical background of Algorithm 489, "The Algorithm *SELECT*—for finding the $i$th smallest of $n$ elements," appearing on p. 173 of this issue.

This work was supported by the National Science Foundation under grants GJ-992 and GJ-33170X. Authors' addresses: R.W. Floyd, Stanford Computer Science Department, Stanford University, Stanford, CA 94305; R.L. Rivest, NE 43-807, Project MAC, 545 Technology Square, Cambridge, MA 02139.

[1] We use the notations $O(n)$ and $o(n)$ in the following way: $f(n) \leq g(n) + O(n)$ means $(\exists k > 0)(\forall n) f(n) - g(n) \leq kn$, and $f(n) \leq g(n) + o(n)$ means $\lim_{n\to\infty}((f(n) - g(n))/n) = 0$.

Let $f(i,n)$ denote the expected number of comparisons required to select $i\,\theta\,X$. (We assume throughout that all possible input orderings of the set $X$ are equally likely.) Since a selection algorithm must determine, for every $t \in X$, $t \neq i\,\theta\,X$ whether $t < i\,\theta\,X$ or $i\,\theta\,X < t$, we have as a trivial lower bound

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

The best previously published selection algorithm is *FIND*, by C.A.R. Hoare [3]. Knuth [4] 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[1]

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

and

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

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

$$f(i,n) \leq n + \min(i,n-i) + O(n^{\frac{1}{2}} \ln^{\frac{1}{2}}(n)). \tag{6}$$

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

$$f(i,n) \leq n + \min(i,n-i) + O(n^{\frac{1}{4}}). \tag{7}$$

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

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

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

$$F(\alpha) \geq 1 + \min(\alpha, 1-\alpha), \quad \text{for} \quad 0 \leq \alpha \leq 1. \tag{9}$$

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

These expected time results should be compared with the worst-case result of Blum et al. [1], which is that $i\,\theta\,X$ can always be found with at most 5.4305 comparisons. While this result is rather surprising, since it gives a time bound linear in $n$, the algorithm presented there is hardly practical. Thus the algorithm presented here is probably the best practical choice. Even for those applications for which only an approximate quantile is desired, which can be obtained by sampling, applying *SELECT* to the sample extracted is probably the most efficient procedure (unless the data has a small finite range).

In what follows $t\,\rho\,X$ will denote the rank of an element $t \in X$, 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.

An abstract of these results has previously appeared in [6].

## 2. The Algorithm *SELECT*

We give an outline here of the algorithm *SELECT*; an Algol definition is given in [2]. The algorithm utilizes sampling.

Step 1. A small random sample $S$ of size $s = s(n)$ is drawn from $X$.

Step 2. Two elements, $u$ and $v$, $(u < v)$, are selected from $S$, using *SELECT* recursively, such that the set $\{x \in X \mid u \leq x \leq v\}$ is expected to be of size $o(n)$ and yet 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 $C$).

Step 3. 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.

Step 4. 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 $s(n) = o(n)$, $E(\mid B \mid) = o(n)$, and $P(i\,\theta\,X \notin B) = o(n^{-1})$, then the total number of comparisons expected is:

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

This can in fact be done; the low order term is $O(n^{\frac{2}{3}} \ln^{\frac{1}{3}}(n))$. Note that for Step 3 the probability that two comparisons are made (against $u$ and $v$) for an arbitrary element in $X - S$ is just $(\min(i,n-i) + o(n))/n$ since $E(\mid B \mid) = o(n)$. Figure 1 shows a geometric analogy of the procedure *SELECT*.

Fig. 1.

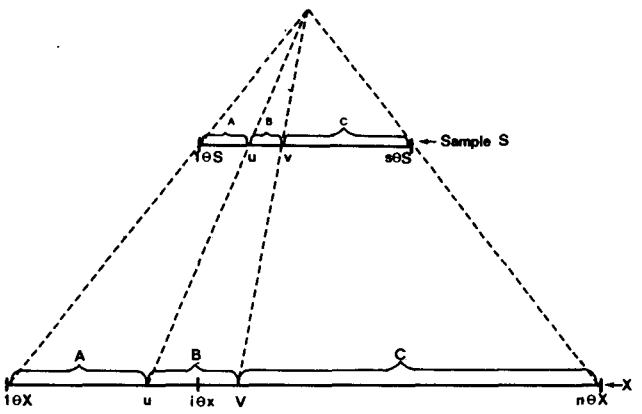
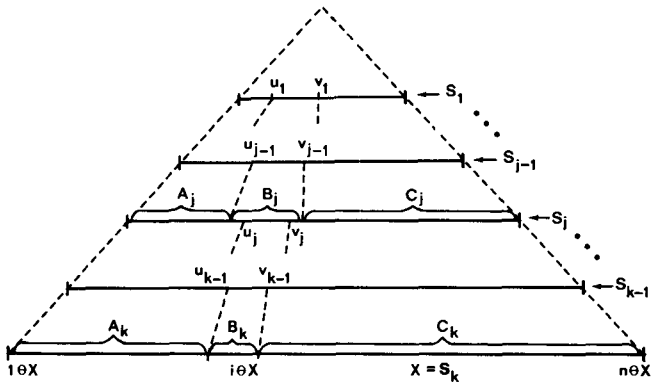



Fig. 2.




### 2.1 Choice of $u$ and $v$

For any $t \in S$ of fixed rank $t\,\rho\,S$ in $S$ we can calculate where $t$ should fall in $X$; that is, $t$'s rank in $X$, $t\,\rho\,X$. Since $S$ is a random sample of $X$ we can calculate only the expected value $E(t\,\rho\,X)$ and its variance.

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

$$
\begin{aligned}
\sigma(t\,\rho\,X) &= \left(\frac{(t\,\rho\,S)(s-(t\,\rho\,S)+1)(n+1)(n-s)}{(s+1)^2(s+2)}\right)^{\frac{1}{2}}, \qquad (11)\\
&\leq \frac{1}{2}\left(\frac{(n+1)(n-s)}{s}\right)^{\frac{1}{2}} \leq \frac{1}{2}\,\frac{n}{(s)^{\frac{1}{2}}}.
\end{aligned}
$$

We wish to choose $u$ and $v$ so that $E(u\,\rho\,X) \leq i \leq E(v\,\rho\,X)$, $E(\mid B \mid) = E(v\,\rho\,X) - E(u\,\rho\,X)$ is $o(n)$, and $P(i < u\,\rho\,X \text{ or } i > v\,\rho\,X) = o(n^{-1})$ (this latter condi-

tion ensures that expected work will be $o(1)$ for the cases we "miss" $i\,\theta\,X$ and have $O(n)$ work yet to do). To do this we choose $u\,\rho\,S$ and $v\,\rho\,S$ so that

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

where $d = d(n)$ is a slowly growing unbounded function of $n$. What is intended here is that it will become extremely unlikely (as $n \to \infty$) that one has chosen a sample $S$ such that $(u\,\rho\,X) > i$ or $(v\,\rho\,X) < i$. Thus as $S$ varies, $(u\,\rho\,X)$ may vary as indicated in Figure 2, but we almost always have $u\,\rho\,X < i < v\,\rho\,X$. Since

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

we will choose $d = (\ln(n))^{\frac{1}{4}}$. This ensures that $P(i < u\,\rho\,X \text{ or } i > v\,\rho\,X) = o(n^{-1})$. The above equations mean that

$$u\rho S \cong \left(i - d\left(\frac{(n+1)(n-s)}{s}\right)^{\frac{1}{2}}\right)\left(\frac{s+1}{n+1}\right) \ge \frac{i(s+1)}{(n+1)} - d(s)^{\frac{1}{2}},$$

and (14)

$$v\rho S \cong \left(i + d\left(\frac{(n+1)(n-s)}{s}\right)^{\frac{1}{2}}\right)\left(\frac{s+1}{n+1}\right) \le \frac{i(s+1)}{(n+1)} + d(s)^{\frac{1}{2}}.$$

### 2.2 Analysis of the Basic Algorithm

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

$$g(i,n) = n + \min(i,n-i) + O(n^{\frac{3}{4}}\ln^{\frac{1}{4}}(n)) \quad (15)$$

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

$$\begin{aligned} &g(u\,\rho\,S,s) + g(v\,\rho\,S - u\,\rho\,S + 1, s - u\,\rho\,S) \\ &\quad \le 2s + v\,\rho\,S - u\,\rho\,S + O(s^{\frac{3}{4}}\ln^{\frac{1}{4}}(s)) \\ &\quad \le 2s + 2d(s)^{\frac{1}{2}} + O(s^{\frac{3}{4}}\ln^{\frac{1}{4}}(s)) \end{aligned} \quad (16)$$

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

$$(n - s(n))(1 + \min(i,n-i)/n + ds^{-\frac{1}{4}}). \quad (17)$$

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

$$E(|B|) = (v\,\rho\,S - u\,\rho\,S)n/s = 2dns^{-\frac{1}{4}} \quad (18)$$

so that

$$g(|B|/2, |B|) = 3dns^{-\frac{1}{4}} + O((dns^{-\frac{1}{4}})^{\frac{3}{4}}(\ln(dns^{-\frac{1}{4}}))^{\frac{1}{4}}). \quad (19)$$

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

$$\begin{aligned} g(i,n) &\le 2s + 2d(s)^{\frac{1}{2}} + O(s^{\frac{3}{4}}\ln^{\frac{1}{4}}(s)) \\ &\quad + (n-s)(1 + \min(i,n-i)/n + ds^{-\frac{1}{4}}) \\ &\quad + 2dns^{-\frac{1}{4}} + 3c/(2d) \\ &\le n + \min(i,n-i) + s + ds^{\frac{1}{4}} \\ &\quad - \min(i,n-i)s/n \\ &\quad + 3dns^{-\frac{1}{4}} + 3c/(2d) + O(s^{\frac{3}{4}}\ln^{\frac{1}{4}}s). \end{aligned} \quad (20)$$

The principal increasing and decreasing terms in $s$ in this expression are $s$ and $3dns^{-\frac{1}{4}}$. Choosing $s(n)$ to make them equal will approximately minimize $g(i,n)$. Thus we choose

$$s(n) \sim n^{\frac{3}{4}}\ln^{\frac{1}{4}}(n) \quad (21)$$

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

### 2.3 An Improved Version of *SELECT*

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

$$u_j\rho S_j = \left(i - d\left(\frac{(n+1)(n-s_j)}{s_j}\right)^{\frac{1}{2}}\right)\cdot\left(\frac{s_j+1}{n+1}\right)$$

and (22)

$$v_j\rho S_j = \left(i + d\left(\frac{(n+1)(n-s_j)}{s_j}\right)^{\frac{1}{2}}\right)\cdot\left(\frac{s_j+1}{n+1}\right).$$

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

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

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*.

**The Improved Algorithm.** The modified algorithm runs as follows.

Step 1. 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).

Step 2. 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*).

Step 3. Next, determine $u_2$ and $v_2$ by applying this algorithm recursively to $B_2$ (in the most likely case; else $A_2$ or $C_2$).

Step 4. 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.

Step 5. Continue in this fashion until a partition $A_k$, $B_k$, $C_k$ of the set $S_k = X$ has been created.

Step 6. 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 $d(n) = \ln^{\frac{1}{2}}(n)$ 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

$$\sigma(u_{j-1}\,\rho\,S_j) \le \frac{s_j}{2(s_{j-1})^{\frac{1}{2}}} \quad \text{and} \tag{24}$$

$$E(u_j\,\rho\,S_j - u_{j-1}\,\rho\,S_j) = \frac{d\cdot(s_j - (s_j)^{\frac{1}{2}})}{(s_{j-1})^{\frac{1}{2}}}. \tag{25}$$

### 2.4 Analysis of the Improved Algorithm

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

$$g(j,m) = m + \min(j,m-j) + O(m^{\frac{1}{2}}), \quad \text{for } m < n,\ 1 \le j \le m. \tag{26}$$

The expected size of $B_j$ is easily estimated:

$$\begin{aligned} E(|B_j|) &= (v_{j-1}\,\rho\,S_{j-1} - u_{j-1}\,\rho\,S_{j-1}) \cdot \left(\frac{s_j}{s_{j-1}}\right) \\ &\le 2ds_j/(s_{j-1})^{\frac{1}{2}}. \end{aligned} \tag{27}$$

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

$$\begin{aligned} &\sum_{2\le j\le k-1} (g(u_j\,\rho\,B_j, |B_j|) + g(v_j\,\rho\,B_j - u_j\,\rho\,B_j \\ &\quad + 1, |B_j| - u_j\,\rho\,B_j)) \\ &\quad \le \sum_{2\le j\le k-1} (4ds_j/(s_{j-1})^{\frac{1}{2}} + 2d(s_j)^{\frac{1}{2}}), \end{aligned} \tag{28}$$

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

$$2s_1 + 2d(s_1)^{\frac{1}{2}}, \tag{29}$$

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

$$3dn/(s_{k-1})^{\frac{1}{2}}. \tag{30}$$

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 $v_2, \ldots, u_{k-1}$ and $v_{k-1}$ is just

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

Adding these all together, we have

$$\begin{aligned} g(i,n) \le\ & n + \min(i,n-i) + \\ & \sum_{2\le j\le k} (5ds_j/(s_{j-1})^{\frac{1}{2}} + d(s_j)^{\frac{1}{2}}) \\ & + s_1(1 - \min(i,n-i)/n) + d(s_1)^{\frac{1}{2}} \\ & - dn/(s_{k-1})^{\frac{1}{2}}. \end{aligned} \tag{32}$$

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

$$\begin{aligned} g(i,n) &\le n + \min(i, n-i) \\ &\quad + \left(\frac{5d}{(s_1)^{\frac{1}{2}}} + \frac{(s_1)^{\frac{1}{2}}}{r}\right) \cdot \sum_{2\le j\le k} r_j \\ &\le n + \min(i, n-i) \\ &\quad + \left(\frac{5d}{(s_1)^{\frac{1}{2}}} + \frac{(s_1)^{\frac{1}{2}}}{r}\right) \cdot \left(\frac{r^{k-1}-1}{r-1}\right) \cdot r^2 \\ &\le n + \min(i, n-i) \\ &\quad + (n)^{\frac{1}{2}} \left(\frac{r^2}{r-1}\right)\left(\frac{5d}{s_1} + \frac{1}{r}\right). \end{aligned} \tag{33}$$

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

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

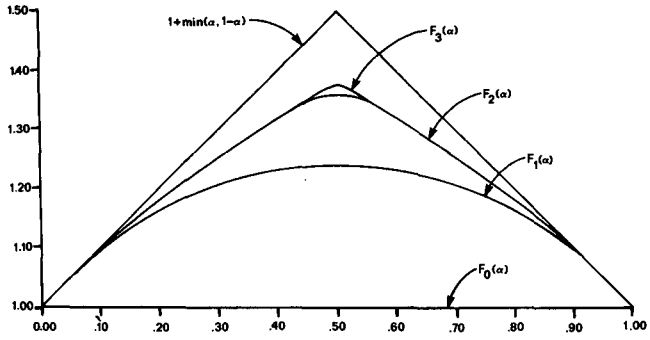
which was to be shown.

## 3. Implementation and Timing Results

In this section we present an Algol implementation of *SELECT* and give timing results that demonstrate that our theoretical results yield fruit in practice. While this is a revised form of the simpler version of *SELECT* given in Section 2, it again uses the optimal or near optimal number of comparisons $n + \min(i,n-i) + o(n)$. The total work is also seen to be a small constant times the number of comparisons made, showing that the number of comparisons made is indeed a valid efficiency measure in practice as well as theory. The purpose of this section is to show that *SELECT* is very efficient for practical values of $n$, as well as asymptotically as $n \to \infty$.

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 of $X[L:R]$, $L \le I \le K$ implies $X[I] \le X[K]$, and $K \le I \le R$ implies $X[I] \ge X[K]$. An implementa-

Fig. 3.




tion 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 $u$ or $v$ before beginning the partition about the other can be done very fast. We therefore use an improved version of Hoare's *PARTITION* algorithm [3] 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.

The procedure as written in ALGOL 60 appears in this issue [2]. The algorithm first determines the element $T$ about which to partition. 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 either $L$ or $R$ changes at each iteration, the number of elements remaining always decreases by at least one, thus ensuring termination.

Timing results were then obtained for *FIND* [3] (exactly as published) and *SELECT* [2]. 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 173.

*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(α)

In this section we present new lower bounds for the expected number of comparisons required for selection, again, assuming all input orderings are equally likely. 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 percent of that value, for all $\alpha$, 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 $0 \leq j \leq \infty$, of lower bounds for $F(\alpha)$ such that $F_j(\alpha) \leq F_{j+1}(\alpha)$, for all $j \geq 0$ and $\alpha$, $0 \leq \alpha \leq 1$. 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 $F(\alpha)$. These bounds have been plotted against $\alpha$ in Figure 3. The value of $F_3(\alpha)$ at $\alpha = .5$ is 1.375, which tapers off as $\alpha$ 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 $y \in X$ must also have determined, for any $x \in X$, $x \neq y$, 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 $i\,\theta\,X$. □

The following definition provides the basis for the lower bound computations. We use notation "$x : y$" to denote a comparison between elements $x$ and $y$.

*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

$$y = i\,\theta\,X \text{ or } x < y < i\,\theta\,X \text{ or } i\,\theta\,X < y < x. \quad (35)$$

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* $n - 1$ *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 "$\leq$" restricted to $S$ is a connected graph. In other words, elements $x$ and $y$ are in the same fragment of the partial ordering if there is a sequence of elements $x = z_1, z_2, \cdots, z_k = y$ such that the comparison $z_i : z_{i+1}$ has been made for $1 \leq i < k$ (with no restrictions on the results of these comparisons). These comparisons need not have been performed in the indicated order; any order will do. The Hasse diagram of a partial order is a directed graph with vertices corresponding to each element of the partial order and an arc $x \rightarrow y$ indicated whenever $(x < y) \wedge \neg\,(\exists z)(x < z < y)$. We adopt the usual convention of omitting arrowheads and requiring $y$ to be higher on the page than $x$ if $x \rightarrow y$ is an arc of the Hasse diagram.

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:

$\mathcal{F}_1 = \{\bullet\}$,

$\mathcal{F}_2 = \{\bullet, \overset{\bullet}{\bullet}\}$,

$\mathcal{F}_3 = \{\bullet, \overset{\bullet}{\bullet}, \overset{\bullet\ \bullet}{\ \ \bullet}, \overset{\ \ \bullet}{\bullet\ \bullet}, \overset{\bullet}{\bullet}\overset{\bullet}{\bullet}\}$, and so on.

*Definition* 5. A *joining comparison* 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.

LEMMA 2. *A selection algorithm must make exactly* $n - 1$ *joining comparisons to select* $i\,\theta\,X$, *where* $|X| = n$.

PROOF. As long as more than one fragment exists, there must be some element incomparable with $i\,\theta\,X$, 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:

CONJECTURE. *Asymptotically (as* $n \rightarrow \infty$*), the average probability that a joining comparison will turn out to be a key comparison is at most*

$$\max(\alpha, 1 - \alpha). \tag{36}$$

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 of the computation 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 comparisons 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

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

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 $\alpha$. 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 $F(\alpha)$.

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 $\epsilon$, $0 < \epsilon < 1$, 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 $\epsilon$ 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

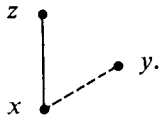
$$\begin{aligned} P(x = i\,\theta\,X) &\leq (2\pi m\alpha(1 - \alpha))^{-\frac{1}{2}} \\ &\cong \binom{m}{\alpha m}\alpha^{\alpha m}(1 - \alpha)^{m - \alpha m} \end{aligned} \tag{38}$$

which is less than $\epsilon$ for $m > (2\pi\alpha(1 - \alpha)\epsilon^2)^{-1}$. So except for a finite number of comparisons near the end, the probability that any element is $i\,\theta\,X$ is at most $\epsilon$. As $n \rightarrow \infty$, 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

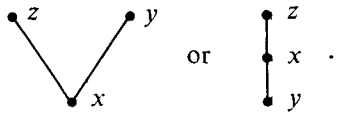
that each joining comparison in which the smaller fragment has at most $k$ elements will turn out to be a key comparison. These comparisons can be divided into two types: those for which both fragments belong to $\mathfrak{F}_k$, and those for which only one fragment has $k$ or fewer elements. The first type 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 proportional to 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:

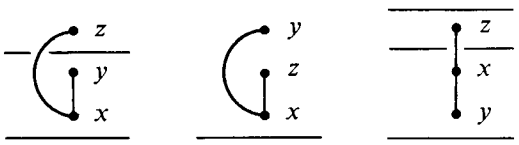
(i) $i\,\theta\,X < x < y < z$
this will be a key comparison for $y$,

(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 $i\,\theta\,X$ 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!):

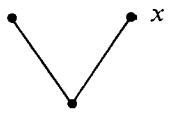
$$P(x{:}y \text{ is key}) = (1 - \alpha)^3 + 2\alpha^2(1 - \alpha) + \frac{\alpha^3}{3}. \quad (42)$$

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 $i\,\theta\,X$ the comparison can not be a key comparison. It is thus easy to see that

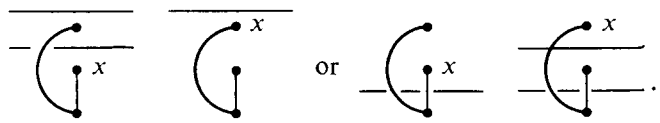
$$P(x : y \text{ is key}) \leq \max(P(x < i\,\theta\,X), P(x > i\,\theta\,X)). \quad (43)$$

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 $i\,\theta\,X$ making a key comparison possible:




(45)

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

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

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 *any* 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 $n - 1$ (for the joining comparisons) plus the weight of the partial ordering (to ensure that $n - 1$ 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 is a number assigned to that fragment which will be computed from the specific probability results already calculated by means of a linear programming technique. A fragment weight is an invariant number associated with that fragment type. The weight of a partial order varies only as its composition as a set of fragment types varies.

What we want to ensure is that as a result of a joining comparison the expected weight of a partial ordering does not decrease 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 • and let $w_2$ be the weight of ⁝. The weight of the initial partial ordering is therefore just $nw_1$. 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:

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

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

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

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 $\mathfrak{F}_k$, 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 [7] 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 $F(\alpha)$ (that is, $F_k(\alpha) = 1 + w_1$).

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

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

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

$$F_2\left(\frac{1}{2}\right) = \frac{49}{36}n \quad \text{and} \tag{50}$$

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

which is within 9 percent of $1.5n$ (the performance of *SELECT*). It is clear from the figure that $F_k(\alpha)$ probably converges rather slowly to $1 + \min(\alpha, 1 - \alpha)$, if our conjecture is correct.

This completes the description of our lower bound derivations. The results show that *SELECT* is at least near-optimal with respect to the number of comparisons used, and we suspect that a more powerful combinatorial analysis would demonstrate optimality. The weakness in our method lies in the restricted nature of the inequalities derivable for the case of a comparison between a small fragment and an arbitrary element belonging to a large fragment. In any case these lower bounds are the first nontrivial lower bounds published for this problem.

Received October 1973; revised July 1974

**References**


1. Blum, M., Floyd, R.W., Pratt, V., Rivest, R., and Tarjan, R. Time bounds for selection. *JCSS* 7 (Aug. 1973), 448–461.
2. Floyd, Robert W., and Rivest, Ronald W. Algorithm 489, The algorithm *SELECT* for finding the *i*th smallest of *n* elements. *Comm ACM* this issue.
3. Hoare, C.A.R. Algorithm 63 (*PARTITION*) and Algorithm 65 (*FIND*). *Comm. ACM 4*, 7 (July 1961), 321.
4. Knuth, Donald E. Mathematical analysis of algorithms. Computer Sci. Dept. Rep. STAN-CS-71-206. Stanford U., Mar. 1971. 27 pp.
5. Lindgren, B.W. *Statistical Theory*. The MacMillan Co., New York, 1962.
6. Rivest, Ronald L., and Floyd, Robert W. Bounds on the expected time for median computations (extended abstract). Courant Computer Science Symposium 9, Randall Rustin [Ed.] Algorithmics Press, New York, 1973, pp. 69–76.
7. Salazar, Rodolfo C., and Sen, Subrata K. Algorithm 333 (*MINIT* algorithm for linear programming. *Comm. ACM 11*, 6 (June 1968), 437–440.