JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 448–461 (1973)

Time Bounds for Selection*

MANUEL BLUM, ROBERT W. FLOYD, VAUGHAN PRATT,
RONALD L. RIVEST, AND ROBERT E. TARJAN

Department of Computer Science, Stanford University, Stanford, California 94305

Received November 14, 1972

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.

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

* This work was supported by the National Science Foundation under grant GJ-992.

(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 $i - 1$ 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:

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

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

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

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

to measure the relative difficulty of computing percentile levels.

In Sec. 2 we prove our *main result, that* $f(i, n) = \mathcal{O}(n)$, by analysis of the basic selection algorithm, PICK.

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

$$\max_{0\leqslant\alpha\leqslant 1} F(\alpha) \leqslant 5.4305\dot{}\tag{1}$$

and

$$F(\alpha) \leqslant 1 + 4.4305\dot{}\alpha/\beta + 10.861\dot{}\lceil \log_2(\beta/\alpha) \rceil\alpha, \qquad \text{for} \quad 0 < \alpha \leqslant \beta, \tag{2}$$

where $\beta = 0.203688^-$. In Sec. 4 we derive the lower bound:

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

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 Sec. 3 regarding the case when $n \bmod c \neq 0$.

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

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 $T =_{\text{def}}$ 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 $D = \{x \mid x \geqslant m\}$ and set $n \leftarrow n - |\,D\,|$, otherwise discard $D = \{x \mid x \leqslant m\}$ 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:

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

The cost of step 1(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 1(b) by $P(n/c)$. After step 1, the partial order determined for S may be represented as in Fig. 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 1(b) determines which elements of T are $<m$, and which are $>m$, we separate the columns as in Fig. 1. Every element

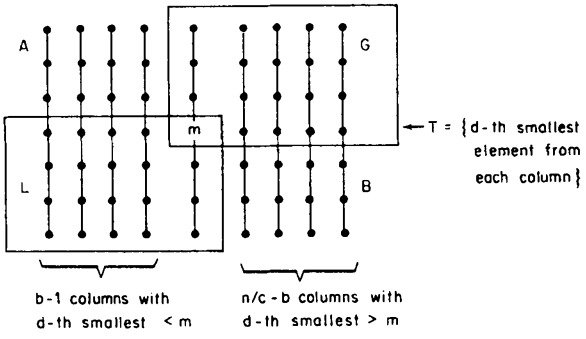


FIGURE 1

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

$$P(n) \leqslant \frac{n \cdot h(c)}{c} + P\left(\frac{n}{c}\right) + n + P(n - \min(|L|, |G|)). \tag{5}$$

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

$$P(n) \leqslant \frac{66n}{21} + P\left(\frac{n}{21}\right) + n + P\left(\frac{31n}{42}\right) \tag{6}$$

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

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

The basis for the induction is that since $h(n) < 19n$ 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 that we must have $c \geqslant 5$ for PICK to run in linear time. Q.E.D.

3. IMPROVEMENTS TO PICK

The main result that $f(i, n) = \mathcal{O}(n)$, 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 over-all 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 $\beta = 0.203688^{-}$. 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 efficiently, 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 $i \leqslant \lceil n/2 \rceil$ throughout this section.

THEOREM 2. $F(\alpha) \leqslant 5.4305$, *for* $0 \leqslant \alpha \leqslant 1$.

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 $1/2$ 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 $1 \leqslant i \leqslant \lceil n/2 \rceil$).

1. If $n \leqslant 45$, 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 $n \leqslant 45$, sort S, print $i\,\theta\,S$, and halt.

2. Sort the set T of column medians into 15-columns and possibly one ($\lceil n/15 \rceil \bmod 15$)-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 Fig. 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 in a set X. Let U be the set of columns of lengths <15 (in fact, each column of U has length $\leqslant 7$). 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 $n \leqslant 45$, sort S, print $i\,\theta\,S$ 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 $>m$.

$la, lb \underset{\text{def}}{=}$ 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:

$$P1(n) \leqslant (42n/15) + P1a(n) = 2.8n + P1a(n), \tag{8}$$

$$P1a(n) \leqslant (42n/225) + P1b(n) = 0.18\dot{6}n + P1b(n). \tag{9}$$

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

LEMMA 1. $v \leqslant d + n/30$.

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 $|L| + la + lb = i + 1 < \lceil n/2 \rceil + 1 \leqslant |L| + |B| \leqslant |L| + gb + lb$. Thus $gb \geqslant la$, but $la \geqslant v - n/30$ 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:

LEMMA 2. $|X| < 6d/7.$ (11)

Proof. We have $d \geqslant 7(w + x + y) - |W| - |X| - |Y|$, and $7w - |W| = |X|$, yielding $d \geqslant 7x + 7y - |Y| \geqslant 7x$, but $6x \geqslant |X|$, so that $d \geqslant 7|X|/6$. 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 $7n/200$ comparisons. We may now write the following recurrence for $P1b(n)$:

$$P1b(n) \leqslant \underbrace{P1a(\lceil n/15 \rceil)}_{\text{step 1}} + \underbrace{3(n/30)}_{\text{step 2i}} + \underbrace{(d + n/30)}_{\text{step 2ii}} + \underbrace{3(6d/7)}_{\text{step 4i}}$$
$$\underbrace{13(7n/30 - d)/15}_{\text{step 4ii}} + \underbrace{4(7n/30 - d)/15}_{\text{step 4iii}} + \underbrace{7n/200}_{\text{step 5i}}$$
$$+ \underbrace{42(7n/30 - d)/225}_{\text{step 5ii}} + \underbrace{P1b(11n/15 - d)}_{\text{subsequent iterations}}.$$

Simplifying yields

$$P1b(n) \leqslant \left(\frac{5n}{n + 5d}\right)\left(\frac{13197n}{27000} + \frac{3546d}{1575}\right). \tag{12}$$

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

$$P1b(n) \leqslant 13197n/27000 = 2.443\dot{8}n, \tag{13}$$

$$P1a(n) \leqslant 2.630\dot{5}n, \tag{14}$$

and

$$P1(n) \leqslant 5.430\dot{5}n. \tag{15}$$

Since

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

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 $F(\alpha)$, 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. $F(\alpha) \leqslant 1 + 4.4305\alpha/\beta + 10.861\lceil \log_2(\beta/\alpha) \rceil \alpha$, *for* $0 < \alpha \leqslant \beta$, (17)
where $\beta = 0.203688^{-}$.

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 leftover.

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 Fig. 2.

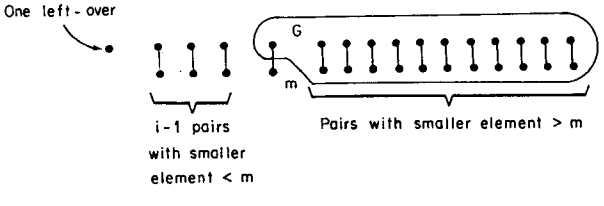


FIGURE 2

3. Discard all elements known to be $>m$, that is, those elements in the circle G of Fig. 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:

$$P2(i, n) \leqslant \underset{\text{step 1}}{\lfloor n/2 \rfloor} + \underset{\text{step 2}}{\min(P1(\lfloor n/2 \rfloor), P2(i, \lfloor n/2 \rfloor))} + \underset{\text{step 4}}{P1(2i)}. \tag{18}$$

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,

$$P2(i, n) \leqslant \sum_{0<j\leqslant t} \lfloor n/2^j \rfloor + P1(\lfloor n/2^t \rfloor) + tP1(2i). \tag{19}$$

This directly implies our theorem. The proper value for β, 0.203688^-, 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 Fig. 3, where our bounds for $F(\alpha)$ are plotted against α. It is not unreasonable to conjecture that the true curve for

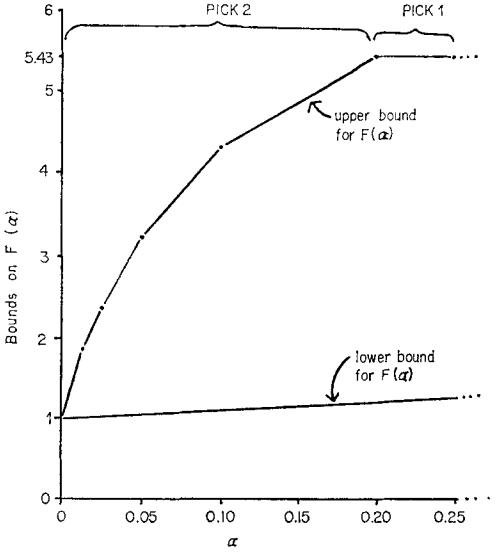


FIGURE 3.

$F(\alpha)$ 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], Sec. 5.3.2.) The selection 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 nontrivial 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 usual minimax cost, that is,

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

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

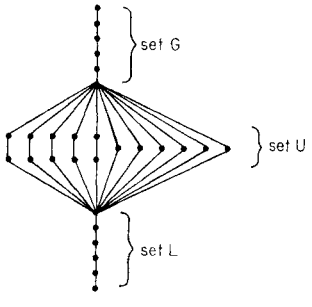
Player B in this game of course plays the role of the "data." A strategy for player B is in effect a rule for calculating a particularly bad (that is, costly) set of data for A's strategy since an actual set of numbers can always be constructed that are consistent with B's replies. A good strategy for 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 (anyone) 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 $|\,L\,| \geqslant i$ or $|\,G\,| \geqslant n - i + 1$. 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 *for each element $x \in U$, there will be at most one $y \in U$ for which it is known whether $x < y$ or $y < x$.* The partial order for S assumed by B may thus always be diagramed:



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 $\sigma(x) \leqslant \sigma(y)$):

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

(ii) if $\sigma(x) = \sigma(y) = 0$ 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 $\sigma(x) = \sigma(y) = 0$, 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

$$c - p + 2 \mid U \mid = 2n \tag{21}$$

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

$$\mid U \mid \leqslant n - \min(i - 1, n - i). \tag{22}$$

Furthermore, it must take player A at least $\mid U \mid - 1 - p$ 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 $\mid U \mid - 1$ comparisons, of which p have already been made. The total number of comparisons made is thus at least

$$f(i, n) \geqslant c + \mid U \mid - 1 - p \geqslant n + \min(i - 1, n - i) - 1, \quad \text{for } 1 \leqslant i \leqslant n \tag{23}$$

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

$$F(\alpha) \geqslant 1 + \min(\alpha, 1 - \alpha). \tag{24}$$

This bound is also plotted in Fig. 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 $5.4305\dot{5}n$ comparisons are required to select the i-th smallest of n numbers, for any i, $1 \leqslant i \leqslant n$. 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. Lewis Carroll, "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. L. R. Ford and S. M. Johnson, A tournament problem, *The American Mathematical Montly* **66** (May 1959), 387–389.
3. Abdollah Hadian, 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.

4. ABDOLLAH HADIAN AND MILTON SOBEL, Selecting the t-th largest using binary errorless comparisons, Technical Report 121, Dept. of Statistics, Univ. of Minnesota, May 1969.
5. C. A. R. HOARE, Find (Algorithm 65), Communications of the ACM, July 1961, pp. 321–322.
6. S. S. KISLITSIN, On the selection of the k-th element of an ordered set by pairwise comparisons, *Sibirsk Math. Z.* **5** (1964), 557–564 (MR 29, No. 2198) (Russian).
7. DONALD E. KNUTH, "The Art of Computer Programming," Vol. III, Sorting and Searching, Addison–Wesley, 1973.
8. JÓSEF SCHREIER, O systemach eliminacjii w turniejach, (On elimination systems in tournaments), *Mathesis Polska* **7** (1932), 154–160 (Polish).

Printed in Belgium