Selecting the $t^{th}$ Best in Average $n+O(\log \log n)$ Comparisons

David W. Matula

Department of Applied Mathematics and Computer Science
October 1973

AMCS-73-9
Selecting the $t^{th}$ Best in Average $n+O(\log \log n)$ Comparisons

David W. Matula
October 1973

ABSTRACT

An algorithm for selecting the $t^{th}$ best element of an $n$-membered linearly ordered set $(R,<)$ is described. Assuming all initial orderings of $R$ are equally likely (hence unrelated to $<$), it is shown that the average number of comparisons of pairs of elements of $R$ utilized by the algorithm to select the $t^{th}$ best for fixed $t$ asymptotically in $n$ is equal to $n+O(\log \log n)$.

AMCS-73-9

*This research was partially supported by the National Science Foundation under Grant GJ-40487.*
I. Introduction and Summary

A sorting problem is composed of (i) a set \( R \) of elements initially sequenced \( r_1, r_2, \ldots, r_n \), possessing a linear ordering, \( \prec \), for which an appropriate rearrangement yields

\[
(1) \quad r_{i_1} \prec r_{i_2} \prec r_{i_3} \prec \ldots \prec r_{i_n}
\]

and (ii) a goal specifying how much of the ordering relation (i) is to be determined. In this paper we shall investigate the problem of selecting the \( \text{th} \) best. That is, we shall investigate several algorithms that determine by a sequence of comparisons of pairs of elements of \( R \) that element

\( r_{i_{n-t+1}} \), which is \( t \)-th largest in the linear ordering given by (1) on \( R \). Our efficiency measures for comparing these algorithms will be the number of comparisons utilized by the algorithm, particularly the asymptotic behavior of this number with increasing \( n \) for fixed \( t \).

The number of comparisons utilized by a specific sequential comparison algorithm for a sorting problem can depend on the initial sequencing of the elements of \( R \), and we shall be concerned with determining both the maximum (max-measure) and average (mean-measure) number of comparisons utilized by such an algorithm over all \( n! \) possible initial sequencings of \( R \).

The asymptotic theory of minimum comparison sorting for determining the full ordering (1) of \( R \) has received considerable attention in the literature. Several methods are described in [6] which require only \( n \log_2 n + O(n) \) comparisons by both
the max-and mean-measures, and it is known that no sequential comparison algorithm can determine the full ordering in less than \((\log_2 n) = n \log_2 n - (\log_2 e) n + O(\log n)\) comparisons by either the max-or mean-measures. The problem of determining the single largest element of \(R\) can be resolved readily utilizing exactly \(n-1\) comparisons independent of the initial ordering of \(R\), and this result is clearly optimal for both the max-and mean-measures.

Blum et al. [1] have shown that for \(1 \leq n\), the \(t^{th}\) largest can be selected in less than \(6n\) comparisons by the max-measure. Floyd and Rivest [2] have shown that the \(t^{th}\) largest can be selected in an average of \(n + t - o(n^2)\) comparisons, e.g., \(2n/n^2\) for determining the median.

For fixed \(t\) asymptotically in \(n\), several known methods select the \(t^{th}\) best in average \(n + o(\log n)\) comparisons. Two such procedures from the literature, termed tree-selection and restricted-insertion, are described in section 2, where tree-selection is also shown to require only \(n + o(\log n)\) comparisons by the max-measure. An important result due to Schreier [6] and Kisletsyn [7] states that the \(2^{nd}\) best element can be selected in \(n + 1 + (\log_2 n)\) comparisons (e.g., by tree-selection), and further that this result is optimal for the max-measure. The Schreier-Kisletsyn result is utilized in section 2 to show that for fixed \(t > 2\), no sequential comparison procedure can select the \(t^{th}\) best in as few as \(n + o(\log n)\) comparisons by the max-measure.

Sobel has devoted considerable attention [9,10] to the
problem of determining the two best elements, and in [10] he
describes and analyses an algorithm by which he claims the
two best elements can be found in an average of \( n+2(\log n) \)
+ \( O(1) \) comparisons. The major result of this paper is that
for any fixed \( t \), asymptotically in \( n \), the \( t \)th largest can be
selected in average \( n+O(\log \log n) \) comparisons. This is
established in section 3 by describing in detail an algorithm,
which we term the eliminator algorithm, and then proving that
the average number of comparisons utilized by the eliminator
algorithm to find the \( t \)th best of an \( n \)-membered linearly
ordered set, denoted \( E_L(n) \), satisfies the inequality

\[
E_L(n) \leq n + t\lceil \log_t n \rceil (11 + \ln \ln n).
\]

Both base 2 logarithms and natural logarithms are employed in
this inequality to reflect the nature of their occurrence in
the analysis of the eliminator algorithm.

We then discuss how the selection result of Floyd and
Rivest [2] could be utilized to modify the eliminator algorithm
and provide an algorithm for determining the \( t \)th best in an
average of no more than \( n+c_1 t (\log \log n) + o(\log \log n) \)
comparisons for some constant \( c_1 \) independent of \( t \). Finally we
close with the conjecture that there is some constant \( c_2 > 0 \)
independent of \( t \) and \( n \) such that for any fixed \( t \geq 2 \), asymptot-
ically in \( n \), any sequential comparison algorithm for the \( t \)th
best requires an average of at least \( n+c_2 t \log \log n + o(\log \log n) \)
comparisons to determine the \( t \)th best.
II. Determining the $t^{th}$ best of $n$

A linearly ordered set $(R,\prec)$ is composed of a set $R = \{r_1, r_2, \ldots, r_n\}$ and a linear ordering $\prec$ on $R$. The linearly ordered set is $n$-membered if $|R| = n$. The linear ordering on $R$ assures that either $r_i \prec r_j$ or $r_j \prec r_i$, but not both, for $r_i, r_j \in R$, $i \neq j$. The specific case that holds is determined by executing a comparison of $r_i$ and $r_j$.

In various computer applications $R$ might be a collection of distinct keys with $\prec$ referring to numeric order, or $R$ might be a set of alphanumeric records with $\prec$ referring to alphanumeric order. A comparison of elements of $R$ might correspond to a single machine instruction or to an intricate subroutine. For the purposes of our model, the execution of a comparison of $r_i$ and $r_j$ will be considered an atomic computing unit, and our measure of efficiency of sorting will pertain solely to the number of comparisons utilized.

Given an $n$-membered linearly ordered set $(R,\prec)$ and a particular $t$, $1 \leq t \leq n$, we are concerned here with the analysis of algorithms that determine by a sequence of comparisons of elements of $R$ that particular element, $r_k \in R$, which ranks $t^{th}$ best according to the linear ordering $\prec$ on $R$. The sequential comparison algorithms we consider for determining the $t^{th}$ best implicitly depend on some initial sequencing of $R$, unrelated to $\prec$. Two measures of the efficiency of these sequential comparison algorithms are
considered:

(1) **max-measure**: the maximum number of comparisons needed by the algorithm to determine the \( t \)th best over all initial sequencings of \( R \).

(2) **mean-measure**: the average number of comparisons needed by the algorithm to determine the \( t \)th best over all initial sequencings of \( R \).

In this article we do not pursue the development of a rigorous formulation for specifying the full class of allowable "sequential comparison algorithms" for determining the \( t \)th best. The interested reader is referred to Knuth [6] for more details on this subject. We assume, however, that this notion is sufficiently self evident so that the following may be defined.

For \( 1 \leq t \leq n \), the **minimax number of comparisons** \( V_t(n) \) and the **minimean number of comparisons** \( V_t(n) \) are the minimum over all sequential comparison algorithms of (respectively) the maximum and the average over all \( n \) initial sequencings of the \( n \)-membered linearly ordered set \( (R, <) \) of the number of comparisons needed to determine the \( t \)th best of \( (R, <) \).

As a foundation for our subsequent results we shall first describe and analyze the standard tree selection procedure for the \( t \)th best. Here \( [x] \) will denote the least
integer greater than or equal to \( x \), and \( [x] \) will denote the greatest integer less than or equal to \( x \).

**TS-Algorithm:** Tree selection for the \( t^{th} \) best [3], [6], pages 142-145.

Assume given the \( n \)-membered non-void linearly ordered set \((R, <)\) with \( R \) labeled in the initial order \( r_1, r_2, \ldots, r_n \) unrelated to \(<\), where \( R \) may be rearranged so that \( r_1, r_2 < \ldots < r_n \). For a given \( t \), with \( 1 \leq t \leq n \), this algorithm determines that element, \( r_{i_{n-t+1}} \), which is \( t^{th} \) largest in \((R, <)\).

**Phase 1:** Form a rooted binary tree, termed the selection tree, with \( n \) terminal nodes all at distances \([\log_2 n]\) or \([\log_3 n]\) from the root, and enter the \( n \) elements of \( R \) into the terminal nodes of the selection tree. In a bottom up manner enter into every other node of the selection tree the larger element, determined by a comparison, of the elements in the two adjacent nodes farther from the root. [This procedure is analogous to a knock-out tournament and is illustrated in figure 1a. Note that exactly \( n-1 \) comparisons are utilized and the largest element of \( R \) is then at the root of the selection tree]

**Phase 2:** Remove the element at the root of the selection tree both from the root node and from all other nodes of the selection tree in which it was entered. Enter a marker, \( X \), into the vacated terminal node, and recompute
Figure 1: Application of the To-Algorithm for determining the 3rd largest of the following twenty-seven integers given in initial order 42, 25, 47, 24, 65, 76, 15, 71, 04, 21, 75, 08, 33, 50, 26, 43, 63, 60, 77, 57, 11, 30, 01, 37, 23, 67, 53. Figure 1(a) shows the initial entries into the selection tree to determine the largest integer, and figures 1(b) and 1(c) show the paths of recomputed entries of the selection tree for determining the 3rd and 2nd largest integers. A total of thirty-four comparisons is utilized.
entries along the path of vacated nodes where the former root element had prevailed. No comparison with \( X \) need be made, the appropriate element of \( R \) is simply moved up. This cycle of removing the root element from the selection tree and recomputing a path to the root to determine a new root element is performed \( t-1 \) times, at which point the \( t^{th} \) largest element is found at the root. [Two cycles of this phase 2 procedure are illustrated in figures 1b and 1c, where the third largest element is then determined.]

Let \( T_e(n) \) and \( \overline{T}_e(n) \) be, respectively, the maximum and average number of comparisons of elements \( r_i, r_j \in R \) utilized by the TS-Algorithm to determine the \( t^{th} \) best of the \( n \)-membered linearly ordered set \( (R, \prec) \) over all initial sequencings of \( R \).

Note that successive cycles of phase 2 of the TS-Algorithm determine the \( 2^{rd} \) best, \( 3^{rd} \) best, \ldots, \( t^{th} \) best, with at most \( \log_{2n} - 1 \) additional comparisons needed to determine the \( j^{th} \) best after the \( (j-1)^{th} \) best has been computed. Thus

\[
T_e(n) \leq n - t + (t-1) \log_{2n} n.
\]

For fixed \( t \geq 1 \), the asymptotic behavior of both \( T_e(n) \) and \( \overline{T}_e(n) \) are readily determined.

Theorem 1: For fixed \( t \geq 1 \) and asymptotically in \( n \),

\[
(3) \quad T_e(n) = n + (t-1) \log_{2n} + O(1),
\]

\[
(4) \quad \overline{T}_e(n) = n + (t-1) \log_{2n} + O(1).
\]
Proof: Consider the \( j^{th} \) cycle of phase 2 of the TS-Algorithm for computing the \( (j+1)^{th} \) best after the \( j^{th} \) best has been removed from the root node. The number of comparisons needed to recompute the appropriate path to the root is no more than \( \lfloor \log_2 n \rfloor - 1 \) and no less than \( \lfloor \log_2 n \rfloor - j \). Thus the \( t-1 \) cycles of phase 2 require no more than \( (t-1) \lfloor \log_2 n \rfloor \) comparisons and no less than \( (t-1) \lfloor \log_2 n \rfloor - t \) comparisons. Adding the \( n-1 \) comparisons of phase 1 of the TS-Algorithm yields equations (3) and (4).

The TS-Algorithm for determining the first best requires \( n-1 \) comparisons, and this is optimal in both the minimax and minimean sense, since it is easily shown that \( V_1(n) = \bar{V}_1(n) = n-1 \).

Although the TS-Algorithm is not optimal in general in either the minimax or minimean sense, Schreier [3] claimed and Kislitsyn [5] proved that the value \( V_1(n) = n + \lfloor \log_2 n \rfloor - 2 \) is the optimal minimax number of comparisons for determining the second best.

We state the following theorem and refer the reader to Knuth [5, p. 212] for a proof.

**Theorem 2:** (Schreier-Kislitsyn Theorem): For \( n \geq 2 \),

\[
V_1(n) = n + \lfloor \log_2 n \rfloor - 2.
\]

The following corollary of the Schreier-Kislitsyn theorem provides a lower bound for general \( t \) on the minimax number of comparisons needed to determine the \( t^{th} \) best.
Corollary 2.1: For $n \geq t \geq 2$,

$$V_t(n) \geq n - t + \lceil \log_2(n-t+2) \rceil.$$ 

Proof: For $n \geq t \geq 2$, let $(R, \less)$ be an $(n-t+2)$-membered linearly ordered set. Form the $n$-membered linearly ordered set $(R \cup B, \less)$ by extending the relation $\less$ to $R \cup B$, where all $t-2$ members of $B$ are larger than any member of $R$. A minimax procedure that determines the $t^{th}$ best in $(R \cup B, \less)$ then provides a feasible method for determining the $2^{nd}$ best in $(R, \less)$. Hence

$$V_t(n) \geq V_t(n-t+2)$$

and equation (6) follows from (7) and (5).

Since $V_t(n) \leq V_t(n)$, from (3) and (6) we conclude for fixed $t$ that $V_t(n) = n + O(\log n)$, where in addition the $O(\log n)$ cannot be replaced by $o(\log n)$. This asymptotic behavior of $V_t(n)$ with $n$ can be summarized in the following corollary that follows directly from (2) and (6).

Corollary 2.2: Given $t \geq 2$ and $\epsilon > 0$, there exists an $N$ such that

$$n + (1-\epsilon)\log_2 n \leq V_t(n) \leq n + (t-1)\log_2 n \quad \text{for } n \geq N.$$

Little is known about the behavior of $V_t(n)$ for fixed $t$ and increasing $n$. More light will be shed on the minimean problem by examining a particularly straightforward algorithm for the $t^{th}$ best based on restricted insertion. This algorithm has
been analyzed by Sobel [9] for t=2, and was earlier considered
by Pickard [7].

RI-Algorithm: Restricted insertion for the $t^{th}$ best [7,9].

Assume given the n-membered non-void linearly ordered
set $(R, <)$ with $R$ labeled in the initial order $r_1, r_2, \ldots, r_n$
unrelated to $<$, where $R$ may be rearranged so that $r_1 < r_2 < \ldots$
$< r_n$. For a given $t$, with $1 \leq t \leq n$, this algorithm determines
that element, $r_{n-t+1}$, which is $t^{th}$ largest in $(R, <)$.

Phase 1: Form a rooted binary tree, termed the selection
tree, with $t$ terminal nodes all at distances $\lceil \log t \rceil$ or
$\lfloor \log t \rfloor$ from the root, and enter the initial $t$ elements
$r_1, r_2, \ldots, r_t$ into the terminal nodes. In a bottom up manner
enter into other nodes of the selection tree the smaller
element determined by a comparison of the elements in the two
adjacent nodes farther from the root. [The smallest element
cf the initial $t$ elements of $R$ then is at the root after $t-1$
comparisons].

Phase 2: For $i=t+1, t+2, \ldots, n$, compare $r_i$ with the "root
element" at the root of the selection tree. If $r_i$ is smaller
and $i<n$, we proceed on to compare $r_{i+1}$ with the root element.
If $r_i$ is larger than the root element, all occurrences of
the root element in the selection tree are deleted, $r_i$ is
inserted in the terminal node position previously occupied
by the former root element, and the path from this node
to the root is recomputed, thereby determining a new smallest
element at the root of the selection tree. Then if \( i < n \), we
proceed on to compare \( r_{i+1} \) with the new root element, and if
\( i = n \), we terminate with the root element being the \( t \)th best.

Figure 2 illustrates the initial and four subsequent up-
dated entries for the selection tree occurring in the applica-
tion of the RI-Algorithm to the same problem illustrated for
the TS-Algorithm in figure 1. Those comparisons wherein the
successive integers are found to be less than the current
root element are not shown in figure 2. A total of thirty-
three comparisons is needed for this example.

![Diagram](image)

**Figure 2:** Application of the RI-Algorithm for determining the
3rd largest of the following twenty-seven integers given in
initial order 42, 25, 47, 24, 95, 76, 15, 71, 04, 21, 75, 00,
33, 70, 26, 43, 63, 60, 77, 57, 11, 30, 01, 37, 23, 67, 53.
Figure 2(a) shows the initial entries in the selection tree,
and 2(b) shows the four updated selection trees encountered.
For $t \leq n$, note that the RI-Algorithm always has the $t$th largest of the subset $\{r_1, r_2, \ldots, r_t\}$ at the root node of the selection tree, so this algorithm may be effectively used to find the $t$th best when the size of the set $R$ is unknown, e.g., if a list of elements of $R$ is being processed in sequence from an external source until an end of list record is processed.

Let $I_t(n)$ and $\bar{I}_t(n)$ be, respectively, the maximum and average number of comparisons of elements $r_i, r_j \in R$ utilized by the RI-Algorithm to determine the $t$th best of the $n$-membered linearly ordered set $(R, <)$ over all initial sequencings of $R$. It is possible in phase 2 of the RI-Algorithm for each of the elements $r_{t+1}, r_{t+2}, \ldots, r_n$ to be larger than the root element of the selection tree and thereby trigger the need for as many as $(\log_2 t)$ more comparisons to update the selection tree. Thus

$$I_t(n) = (\lceil \log_2 t \rceil + 1)n - t\log_2 t - 1.$$

This result indicates poor performance for the RI-Algorithm as regards the max-measure in view of the bounds for $V_t(n)$ in (8). For the mean-measure we now show that this algorithm is quite comparable to the TS-Algorithm. The following generalizes to arbitrary $t$ the result of Sobel [8] for $t=2$.

Theorem 3: For fixed $t \geq 1$ asymptotically in $n$, 


(10) \( \Gamma_t(n) = n + t(\log_2 t) \ln n + O(1), \)

where

\[
\frac{[\log_2 t]}{\log_2 t} = \frac{2}{t} - \frac{t}{e}.
\]

Note: Use of base 2 and natural logarithms is intentional to reflect the distinct nature of their origins in the algorithm.

Proof: The AI-Algorithm phase 1 comparisons and those phase 2 comparisons where each new element from \( R \) is compared with the root element of the selection tree are \( n-1 \) in number. The average number of comparisons, \( \Gamma_t(n) \), is the expected number given equal probability of all initial sequencings of \( R \). Note that the succession of root elements determined by the algorithm depends only on the initial sequencing of \( R \) and not on the structure of the selection tree. Thus a root element replacement requires an average number of additional comparisons given by the average distance of the terminal nodes from the root of the selection tree. For a rooted binary tree with \( t \) terminal nodes all at distances \( [\log_2 t] \) or \( \log_2 t \) from the root, this average, denoted by \( \log_2 t \), is readily computed to be

\[
\frac{[\log_2 t]}{\log_2 t} = \frac{2}{t} - \frac{t}{e}.
\]

Therefore

\( \Gamma_t(n) = n - 1 + (\log_2 t) E(\text{rep}) \)
where $E(\text{rep})$ is the expected number of root element replacements in phase 2 of the RI-Algorithm. For $t+1 \leq j \leq n$, the uniform probability for all initial sequencings of $R$ assures that $r_j$ will be greater than the root element of the selection tree (which is the $t^{th}$ largest of $(r_1, r_2, \ldots, r_{t-1})$) with probability $\frac{t}{i}$. Hence

$$E(\text{rep}) = t \sum_{i=t+1}^{n} \frac{1}{i}.$$ 

It is well known that

$$\frac{1}{i} = \ln n + O(1),$$

so for fixed $t$ asymptotically in $n$

$$E(\text{rep}) = t \ln n + O(1),$$

and then

$$\overline{I}_t(n) = n + t \left(\log_2 t\right) \ln n + O(1).$$

Any sequential comparison algorithm for determining the $t^{th}$ best of an $n$-numbered linearly ordered set $(R, <)$ with $1 \leq t \leq n$ must require at least $n-1$ comparisons simply to assure at completion that there is no partition $R=A \cup B$, $A \cap B = \emptyset$, without some comparison having been made of some element of $A$ with some element of $B$. It is interesting that $\overline{I}_t(n) - n$ and $\overline{I}_t(n) - n$ are both of order $\log n$, and it is instructive to analyze the $\tau S$- and RI-Algorithms to question if $\log n$ is indeed the order of $\overline{U}_t(n) - n$ for fixed $t \geq 2$. The analysis for $t=2$ will be sufficiently instructive, so from (4) and (10)

(11) \hspace{1cm} \overline{U}_2(n) - n = \log n + O(1) = 1.44 \ldots \ln n + O(1)

(12) \hspace{1cm} \overline{I}_2(n) - n = 2 \ln n + O(1).
In the TS-Algorithm for second best, the set of elements compared with that element determined to be best in phase 1 has at least \( \lceil \log_2 n \rceil \) members, and the subsequent need to determine the best of this set explains the order \( \log n \) behavior for \( T_2(n) \cdot n \) in (11). In the RI-Algorithm for second best, the set of elements found to be larger than the root element (a previously dominated element) has an expected size of \( \sum_{i=3}^{n} \frac{1}{i} = 2 \ln n + O(1) \), and the subsequent need to perform comparisons with all elements of this set explains the order \( 2 \ln n \) behavior for \( T_2(n) \cdot n \) in (12). [Note that the major contribution to the harmonic number \( H_n \) occurs from a relatively small number of initial terms of the sum. Thus for most initial sequenceings of \( R \), the major contribution to the order \( \log n \) behavior for (12) occurs before either the best or second best elements of \( R \) have even been considered for comparisons in the RI-Algorithm.]

The Eliminator Algorithm we propose in the next section is a hybrid which avoids the major contributions from the two situations that lead to the order \( \log n \) terms in (11) and (12). This new algorithm constructively demonstrates that \( T_{\mathcal{E}}(n) \cdot n = O(\log \log n) \). Further analysis of the inherent level of prevalence of the two situations cited above leads us to believe that \( T_{\mathcal{E}}(n) \cdot n \) can not be as small as \( o(\log \log n) \).
III. The Eliminator Algorithm for the $t^{th}$ Best.

Given an $n$-membered linearly ordered set $(R, <)$ with $1 \leq t \leq n$, a $t$-eliminator of $(R, <)$ is any element of $R$ which is not one of the $t-1$ best elements of $R$. Thus if $r \in R$ is known to be a $t$-eliminator of $(R, <)$ and if $r < i$ is determined by executing a comparison for a particular $r \in R$, then that particular element $r$ may be "eliminated" from any possibility of being the $t^{th}$ best in $(R, <)$. For the RI-Algorithm of the preceding section, each of the successive root elements are $t$-eliminators, and these successive root elements form a monotonically increasing sequence terminating with the $t^{th}$ best of $(R, <)$.

Phases 1 and 2 of the hybrid 4-phase algorithm for $t^{th}$ best which we now describe determine for a specified initial sequence $r_1, r_2, \ldots, r_{mt}$ of $R$ (utilizing only $mt-1$ comparisons) an element $r_j$, $1 \leq j \leq mt$, which is guaranteed to be a $t$-eliminator of $(R, <)$ and also is expected to be larger than "most" members of $r_{mt+1}, r_{mt+2}, \ldots, r_n$. Phase 3 processes the subsequence $r_{mt+1}, r_{mt+2}, \ldots, r_n$ in an eliminator update manner equivalent to phase 2 of the RI-Algorithm, with, however, fewer eliminator replacements occurring than in the RI-Algorithm.

With an appropriate choice for $m$, there is only a small probability that the final $t$-eliminator determined at the end of phase 3 is not the $t^{th}$ best of $(R, <)$. When this happens phase 4 is invoked to pursue the additional comparisons necessary to select the $t^{th}$ best of $(R, <)$.
A precursor of this hybrid algorithm for the special case \( t=2 \) was one of several algorithms analyzed by Sobel in [10] although the critical choice of the set size \( m \) for our phase 1 for \( n \gg 2 \) was treated in [13] with some inconsistency in the algorithm description and subsequent optimization analysis.

**RL-Algorithm:** Eliminator selection and replacement for the \( t^{th} \) best.

Assume given the \( n \)-membered non-void linearly ordered set \( (R, \prec) \) with \( R \) labeled in the initial order \( r_1, r_2, \ldots, r_n \) unrelated to \( \prec \), where \( R \) may be rearranged so that \( r_1 \prec r_2 \prec \ldots \prec r_n \).

For a given \( t \), with \( 1 \leq n \), let \( m \) be chosen so that \( \frac{1}{t} \leq \frac{m}{n} \).

(The value \( \frac{\log n}{\log n} \) is recommended for \( n \gg t \).) This algorithm determines that element, \( r_{n-t+1} \), which is the \( t^{th} \) largest in \( (R, \prec) \).

**Phase 1:** Primary candidate selection.

1.1 [Initialize] Set \( j-1 \).

1.2 [Select primary candidates] Form a rooted binary tree, termed the \( j^{th} \) selection subtree, with \( m \) terminal nodes all at distances \( |\log m| \) or \( |\log m| \) from the root, and enter elements \( r_{m(j-1)+1}, r_{m(j-1)+2}, \ldots, r_m \) into the terminal nodes. In a bottom up manner enter into each other node of the \( j^{th} \) selection subtree the larger element, determined by a comparison, of the elements in the two
adjacent nodes farther from the root. The element entering the root is called the $j^{th}$ candidate and denoted by $c_j$.

1.3 [Loop on j] If $j < t$, set $j = j+1$ and go to step 1.1, otherwise go to step 2.1.

Phase 2: Eliminator determination.

2.1 [Determine eliminator] Form a rooted binary tree, termed the eliminator tree, with $t$ terminal nodes all at distances $|\log t!$ or $|\log 2t!$ from the root, and enter the candidates $c_1, c_2, \ldots, c_t$ into the terminal nodes. In a bottom up manner enter into each other node of the eliminator tree the smaller element, determined by a comparison, of the elements in the two adjacent nodes farther from the root. The element entering the root is denoted by $t$, and recognized to be a $t$-eliminator since $t$ is smaller than the other $t-1$ elements in the eliminator tree. (At this point note that

\begin{equation}
\ell = \min \{c_j \mid c_j = \min_j \max_{i \leq j} |(j+1) - i|, 1 \leq i \leq 2j\}.
\end{equation}

Although known to be less than $t-1$ relatively large elements of $R$, the $t$-eliminator $t$ is also greater than $n-1$ other typical members of $R$, so $t$ itself can be expected to be relatively large in $R$.)

2.2 [Test for exhaustion of $R$] If $n = m$, go to step 4.1, otherwise go to step 3.1.
Phase 3: Eliminator replacement.

3.1 [Initialize] Set \( i = \text{mt} + 1 \), \( l_{i-1} = \ell \).

3.2 [Comparison with eliminator] Compare \( r_1 \) with \( l_{i-1} \). If \( r_1 < l_{i-1} \), set \( l_{i-1} = r_1 \) and go to step 3.4, otherwise go to step 3.3.

3.3 [Eliminator replacement] Remove the element \( l_{i-1} \) from all nodes it occupies in the eliminator tree (which are nodes of a path from a terminal node to the root) and enter element \( r_1 \) into the terminal node just vacated by \( l_{i-1} \). Recompute the entries in the nodes of the eliminator tree just vacated, and denote the new \( t \)-eliminator entered into the root by \( l_1 \).

3.4 [Loop on i] If \( i < n \), set \( i = i + 1 \) and go to step 3.2, otherwise go to step 4.1. (Note that the \( t \)-eliminator \( l_1 \) is the \( t \text{th} \) best of \( \{c_1, c_2, \ldots, c_{\ell}, r_{\text{mt}+1}, r_{\text{mt}+2}, \ldots, r_1\} \) for \( \text{mt} + 1 \leq i \leq n \), but it is not true in general that \( l_1 \) is the \( t \text{th} \) best of \( \{r_1, r_2, \ldots, r_1\} \)).

Phase 4: Reinspection of candidate selection subtrees.

4.1 [Trivial completion test] If \( n = 1 \), go to step 4.11.

4.2 [Implicit completion test] Let \( \{c_{i_1}, c_{i_2}, \ldots, c_{i_g}\} \) denote the candidates which are still members of the eliminator tree other than the element \( l_n \). If this set is void go to step 4.11, otherwise \( g > 1 \) go to step 4.3. (Note that if no candidate \( c_j \), \( 1 \leq j \leq t \), remains in the eliminator tree other than possibly the root element \( l_n \), then
no element \( r_1, 1 \leq i \leq m \) can be larger than \( i_n \), so \( i_n \) is the \( t \)th largest of \((R, <)\).

4.3 (Determine best of \([r_1, \ldots, r_m]\)). Form a rooted binary tree with \( s \) terminal nodes all at distances \([\log_2 s]\) or \([\log_2 s]\) from the root, and enter the elements \( c_{i_1}, c_{i_2}, \ldots, c_{i_s} \) into the terminal nodes. In a bottom-up manner enter into each other node the larger element, determined by a comparison, of the elements in the two adjacent nodes farther from the root. Denote the element entered into the root by \( b_1 \). (Note that \( b_1 = \max(r_1, r_2, \ldots, r_m) \).

Some of the comparisons called for in step 4.3 may have been made in step 2.1 and need not be repeated. On the average the small probability of reaching step 4.3 for \( n \gg t \) is such that repeating these comparisons is immaterial.

4.4 (Form selection tree) In the rooted binary tree of step 4.3 replace the terminal node containing \( c_{i_j}, 1 \leq j \leq s \), by the \( j \)th selection subtree determined in step 1.2, carrying along all elements of \( R \) associated with all nodes of the selection subtrees. The new tree created is termed the selection tree, and has \( s \) terminal nodes all at distances \( n \) greater than \([\log_2 m] + [\log_2 s] \) and no less than \([\log_2 m] + [\log_2 s] \) from the root. (Note that every element of \([r_1, r_2, \ldots, r_m] \) which is larger than the \( t \)-eliminator \( i_n \) must now be at a terminal node of the selection tree, and by our construction, \( b_1 \) is already in the eliminator tree and \( b_1 \) is not \( i_n \).

4.5 (Initialize loop) Set \( i=2 \), \( i_{n+1} = i_n \).
4.6 [Determine next largest in \( \{r_1, r_2, \ldots, r_m\} \)]. Remove \( b_{i-1} \) from the root node and from all other nodes of the selection tree in which it was entered. Enter a marker, \( X \), into the vacated terminal node, and recompute entries along the path of vacated nodes where \( b_{i-1} \) had prevailed. No comparison with \( X \) need be made, the appropriate element is simply moved up. The new element thereby entered into the root of the selection tree is denoted by \( b_i \).

4.7 [Test for candidate status] If \( b_i \in \{c_{i_1}, c_{i_2}, \ldots, c_{i_s}\} \), go to step 4.10, otherwise go to step 4.8.

4.8 [Comparison with eliminator] Compare \( b_i \) with \( b_{n+i-1} \).
If \( b_i < b_{n+i-1} \), go to step 4.11, otherwise go to step 4.9.

4.9 [Eliminator replacement] Remove the root element \( b_{n+i-1} \) from all nodes it occupies in the eliminator tree, and enter \( b_i \) into the terminal node just vacated by \( b_{n+i-1} \). Recompute the entries for the eliminator tree nodes just vacated, and denote the new \( t \)-eliminator entered into the root by \( b_{n+i} \).

4.10 [Loop on \( i \)] If \( i = m \), go to step 4.11, otherwise set \( i = i+1 \), and go to step 4.6.

4.11 [Termination] Terminate the algorithm. The root of the eliminator tree contains the \( t^{th} \) best of \( (R, \langle \rangle) \), and the terminal nodes of the eliminator tree contain the \( t \) best elements of \( (R, \langle \rangle) \).
Consider the application of the ESI-Algorithm to the same example data for which the TS- and RI-Algorithms were illustrated in figures 1 and 2. The problem is to determine the 3rd largest of the 27 integers initially sequenced (42, 25, 47, 24, 05, 76, 15, 71, 04, 21, 75, 00, 33, 70, 26, 43, 63, 60, 77, 57, 11, 30, 61, 37, 23, 67, 53). For n=27, t=3, we determine \( m = \lceil \frac{n \log t}{t \log n} \rceil = 3 \), so each of the \( t=3 \) selection subtrees of phase 1 will have \( m=3 \) terminal nodes, as seen in figure 3(a). The eliminator tree of phase 2 has \( t=3 \) terminal nodes into which the three candidates determined as largest in the respective selection subtrees of phase 1 are entered. The smallest, 47, of the three candidates is then determined as the initial eliminator, as seen in figure 3(b).

Proceeding through phase 3, the integer 21 is smaller than the the eliminator, 47, but then 75 is larger, causing an eliminator replacement with new eliminator 71 (see figure 3(c)). Then 00, 73, 70, 26, 43, 63, and 60 are successively less than the eliminator 71, but 77 is larger, effecting an eliminator replacement with new eliminator 75. The remaining comparisons of phase 3 cause no additional eliminator replacements.
Figure 3: Application of the EL-Algorithm for determining the \(3^{rd}\) largest of the 27 integers (42, 25, 47, 24, 05, 76, 15, 71, 04, 21, 75, 00, 33, 70, 26, 43, 63, 60, 77, 57, 11, 30, 01, 37, 23, 67, 53) using \(m=3\). 3(a): Phase 1 selection subtrees, 3(b): Phase 2 eliminator tree, 3(c): Phase 3 eliminator replacements, 3(d): Phase 4 selection tree.
The second initial candidate, 76, is still a member of the final entries in the eliminator tree at the end of phase 3, so the selection tree described in steps 4.3 and 4.4 of phase 4 must be determined. Since only 1 candidate remains, s=1, and the second selection subtree becomes the full selection tree. The next largest element in the selection tree, 24, is determined as seen in figure 3(b). A final comparison of 24 with the eliminator 75 signals that the algorithm terminates, having determined 75 to be the third largest with a total of thirty comparisons having been utilized. Note that elements 76 and 77 have not been compared with each other, so in this case the 3rd largest is determined without the ordering of the 3 largest being determined.

The EL-algorithm for the best of \( R_i, < \) with \( m < n \) can require close to a logarithm comparisons for the max-measure since each \( r_{i1}, m + 1 \leq i \leq n \), can cause an eliminator replacement in phase 3. Over all initial sequencings of \( R_i \), however, the average number of eliminator replacements in phase 3 will be relatively small. This is particularly the case since, as noted in the commentary about step 2.1, the initial \( t \)-eliminator is chosen so as to be relatively large. Specifically, if the initial subsequence \( r_1, r_2, \ldots, r_{mt} \) is ordered by \( < \) as

\[
\begin{align*}
& r_{i1} < r_{i2} < \ldots < r_{i(m+t-1)+1} < \ldots < r_{i_{mt}}
\end{align*}
\]

where \( i = r_{i(m+t-1)+1} \) is the \( t \)-eliminator determined in step 2.1 of the EL-algorithm, then we assert that over all initial
sequencings of $R$ the expected value of $\hat{t}^*$, $E(\hat{t}^*)$, satisfies

$$E(\hat{t}^*) \leq t \ln t + t.$$  

For the recommended value $m = \lfloor \frac{\ln t}{\ln n} \rfloor$, with $n \gg t$, this then implies that the $t$-eliminator $r_{\lfloor \frac{\ln t}{\ln n} \rfloor}$ selected in step 2.1 would have an approximate expected position no worse than $(t \ln n)^{th}$ best out of the full set $(R, <)$. The truth of the assertion (15) follows from the following combinatorial lemma.

**Lemma 4:** Let $\phi(1), \phi(2), \ldots, \phi(mt)$ be a uniformly chosen permutation of the integers $1, 2, \ldots, mt$ for $m, t \geq 1$. Let the random variable $\hat{t}^*$ be given by

$$\hat{t}^* = \max_{j \leq 1} \min_{1 \leq j \leq m} \{ j \mid (j-1)m+1 \leq j \leq jm \}. $$

Then

$$E(\hat{t}^*) \leq t \ln t + t.$$  

**Proof:** The result is immediate for $t=1$, so assume $t \geq 2$, $m \geq 1$. For a given permutation $\phi$ and each $j$, $1 \leq j \leq t$, let $Q_j = \phi(1) | (j-1)m+1 \leq j \leq jm$. For $1 \leq k \leq n$, the probability that a particular set $Q_j$ contains no integer less than or equal to $k$ is no greater than $\frac{t-1}{t}^k$, and the expected number of sets $Q_j$, $1 \leq j \leq t$, containing no integer less than or equal to $k$ is no greater than $t \frac{t-1}{e}^k$. With $\hat{t}^* = \max_j \min \{ s \mid s \in Q_j \}$, $\text{Prob}(\hat{t}^* > k)$ is equal to the probability that at least one set $Q_j$ contains no integer less than or equal to $k$, which in turn is no greater than the expected number of $Q_j$ containing no integer less than or equal to $k$, so that for any $k \geq 1$.
\[
\text{Prob} \{ \ell^* > k \} \leq \min\{1, t \left( \frac{t-1}{t} \right)^k \}.
\]

Since \( E(X) = \sum_{i=0}^{m} \text{Prob}(X=k) \) for any discrete non-negative integer valued random variable \( X \),
\[
E(\ell^*) \leq \sum_{k=0}^{m} \min\{1, t \left( \frac{t-1}{t} \right)^k \}
\]
\[
\leq \sum_{k=0}^{[t \ln t] - 1} 1 + \sum_{k=[t \ln t]}^{m} t \left( \frac{t-1}{t} \right)^k.
\]
\[
\leq \lfloor t \ln t \rfloor + t \left( \frac{t-1}{t} \right)^{\lfloor t \ln t \rfloor}.
\]

Then for \( t \geq 2 \), using the identity \( b^{\lfloor a \rfloor} = a^{\lfloor \ln b \rfloor} \),
\[
\left( \frac{t-1}{t} \right)^{\lfloor t \ln t \rfloor} \leq \left( \frac{t-1}{t} \right)^{t \ln t} \leq e^{t \ln (1 - \frac{1}{t})} \leq e^{t \left( - \frac{1}{t} \right)} = \frac{1}{e}.
\]

and inequality (16) follows immediately.

Let \( \bar{E}_c(n) \) be the average, over all initial sequencing\# of \( R \), of the number of comparisons of elements \( r_i, r_j \in R \) utilized by the EL-Algorithm to determine the \( t \)th best of the \( n \)-membered linearly ordered set \( (R, \delta) \) with \( m = \left\lfloor \frac{n \ln t}{\ln n} \right\rfloor \).
Theorem 5: For $2 < n$,

\[ L_t(n) \leq n + t \log_2 (1 + \ln \ln n) \]

\[ L_t(n) \geq n + t \log_2 (\ln \ln n - \ln \ln t) - 1 - \frac{2t \ln n}{n} \]

where $\log_2 = (\log_2 t) - \frac{\ln t}{t}$

[Note that $|\log_2| \leq \log_2 \leq |\log_2|$.]

Proof: Assume for the EL-Algorithm that

\[ n = \left\lfloor \frac{n \ln t}{t \ln n} \right\rfloor, \]

\[ L_t(n) = F_1 + F_2 + F_3 + F_4 \]

\[ F_1 = F_{3,2} + F_{3,3} \]

where $F_i$ is the expected number of comparisons of elements of $R$ computed in phase $i$, $1 \leq i \leq 4$, of the EL-Algorithm, and

where $F_{3,2}$ and $F_{3,3}$ are the expected number of comparisons in steps 3.2 and 3.3 of the algorithm. Clearly $F_1 = t(m-1)$, $F_2 = t-1$, and $F_{3,2} = c - m t$, so

\[ L_t(n) = n - 1 + F_{3,2} + F_4 \]

Since $n-1$ comparisons are always needed to find the largest element of $R$, $F_{3,2} + F_4$ represents the additional effort needed to find the $t^{th}$ rather than first largest.

As noted in the proof of Theorem 3, each successive $t$-eliminator at the root of the eliminator tree is equally likely to be at any of the terminal nodes of the
eliminator tree, so each eliminator replacement made in
step 3.3 requires an expected number of comparisons equal
to the average distance of the terminal nodes from the
root in the eliminator tree, namely
\[ \log_{2t} \left( \log_{2t} 1 \right) = \frac{1}{2^{t}}. \]
Thus
\[ F_{t,1} = E(\text{rep}) \frac{1}{2^{t}} \]
where \( E(\text{rep}) = \sum_{i=mt+1}^{n} \text{Prob} \{ r_{i} \geq 1 \} \) is the expected number of
eliminator replacements (i.e., executions of step 3.3) for
the El-Algorithm.

For \( mt+1 \leq n \), the \( t \)-eliminator \( l_{t-1} \) is known to be no
better than \( t \)-th best of the set \( \{ r_{1}, r_{2}, \ldots, r_{t-1} \} \), thus
\[ \text{Prob} \{ r_{i} \geq 1 \} \geq \frac{1}{2^{t}} \]
and
\[ E(\text{rep}) \geq t \sum_{i=mt+1}^{n} \frac{1}{2^{t}} \left( \frac{\ln n}{n} \right)^{\frac{1}{2^{t}}} \]
and since \( 2m \geq \frac{1}{2} \ln n \),
\[ E(\text{rep}) \geq t \left( \frac{\ln n}{n} \right)^{\frac{1}{2^{t}}} \]
therefore
\[ L_{t}(n) \geq n-1 + t \log_{2t} \left( \frac{\ln n}{n} \right) - \ln n \cdot \ln t \]
verifying the lower bound (18).
For $t+1 \leq n$, it follows by lemma 6 that $\text{Prob}(r_i > t_i)$ for $t \leq \ln t + t/(mt+1)$, so

$$\text{Prob}(r_i > t_i) \leq \frac{\text{Prob}(r_i > t_i)}{mt+1} \leq \frac{t + \ln t + t}{mt+1} \leq \frac{2+\ln t}{m}.$$  

Since, necessarily, $\text{Prob}(r_i > t_i) \leq 1$, we conclude

$$\text{Prob}(r_i > t_i) \leq \frac{4+\ln t}{mt+2} \text{ for } mt+1 \leq i \leq n.$$  

The $t$-eliminator $l_i$ can be no worse than the $t$th best of $(r_{mt+1}, r_{mt+2}, \ldots, r_{t_i})$ for $i \geq mt+2$, so additionally

$$\text{Prob}(r_i > t_i) \leq \frac{t}{t-mt} \text{ for } mt+1 \leq i \leq n.$$  

Hence for any $1 \leq k$, recalling $E(\text{rep}) = \sum_{i=mt+1}^{n} \text{Prob}(r_i > t_i)$,

$$E(\text{rep}) \leq \frac{4+\ln t}{mt+2} k + t \sum_{j=k+1}^{n-mt} \frac{1}{j} \leq \frac{4+\ln t}{mt+2} k + t \ln \frac{n}{k}.$$  

Choosing $k \geq \frac{\ln t}{\ln t} > \frac{(mt+1)t}{\ln t} > \frac{\ln t}{\ln n}$, then also $k \geq \frac{\ln t}{\ln t}$, so that

$$E(\text{rep}) \leq t(1 + \frac{4}{\ln t} + \ln \ln n)$$

and finally from (19), (20) and (21)

$$E_t(n) \leq n - 1 + t \log_2 (1 + \frac{4}{\ln t} + \ln \ln n) + F_k.$$  

To determine $F_k$, assume exactly $j \geq 1$ of the best $t-1$
elements of $R$ are in $\{r_1, r_2, \ldots, r_{mt}\}$. Then $1 \leq j$ in step 4.2, and the number of comparisons required are then at most $j-1$ in step 4.3, at most $j(\lceil \log_2 m \rceil + \lceil \log_2 j \rceil)$ in the $j$ passes through step 4.6, at most $j$ in the passes through step 4.8, and at most $(j-1) \lceil \log_2 t \rceil$ in the passes through step 4.9. Letting $p_j$, $0 \leq j \leq t-1$, be the probability that exactly $j$ of the best $t-1$ elements of $R$ are in $\{r_1, r_2, \ldots, r_{mt}\}$,

$$F_* \leq \sum_{j=1}^{t-1} j p_j [2(\lceil \log_2 m \rceil + 2 \lceil \log_2 t \rceil)]$$

Now $\sum_{j=0}^{t-1} j p_j$ is the expected number of the best $t-1$ elements of $R$ in $\{r_1, r_2, \ldots, r_{mt}\}$, and this is

$$\sum_{j=0}^{t-1} j p_j = (t-1) \frac{mt}{n} < t \frac{\log_2 t}{\log_2 n}.$$  

Also $m \geq 2$ may be assumed for otherwise $F_* = 0$, and since $n \geq mt$,

$$[2(\lceil \log_2 m \rceil + 2 \lceil \log_2 t \rceil)] \leq 4 \log_2 m + 4 \log_2 t \leq 4 \log_2 n$$

hence

$$F_* \leq 4 \frac{t \log_2 t}{\log_2 n},$$

and finally from (22) and (23),

$$L_t(n) \leq n + t \frac{\log_2 (n+1) + \log_2 n}{n}. $$
Corollary 5.1: For fixed \( t > 2 \), asymptotically in \( n \),

\[
L_{\leq t}(n) = n + (t \log t) \ln n + o(n).
\]

For \( t \) growing linearly with the logarithm of \( n \), the following is obtained.

Corollary 5.2: For any fixed \( a > 0 \),

\[
L_{\leq a \log n}(n) = n + o(n \log \log n) \left( \log \log n \right)^2.
\]

\( L_{\leq t}(n) \) provides an upper bound on \( U_{\leq t}(n) \), and for \( t < n \),

(24) and (25) provide upper bounds on \( U_{\leq t}(n) \) that are sharper than those previously announced in the literature. For \( t \) growing more rapidly with \( n \), such as in determining a percentile or median element, the EL-Algorithm is not as good as a variation of the SELECT algorithm of Floyd and Rivest [2] which they use to establish

\[
U_{\leq t}(n) \leq n + \min(t, n-t) + O(n^{1/2}).
\]

Returning to the case of fixed \( t \) and increasing \( n \), it follows from theorem 5 that there is a non-negative valued function \( b(t) \) defined for \( t > 2 \) such that

\[
U_{\leq t}(n) \leq n + b(t) \ln n \ln n + o(\ln \ln n)
\]

where \( 0 < b(t) = O(t \log t) \). For large \( t \) this upper bound on \( b(t) \) may be improved. Specifically, if the elements determined to be larger than the eliminator in step 3.2 of the EL-Algorithm
are saved and if, at appropriate points, a new \( t \)-eliminator is determined by utilizing Floyd and Rivest's SELECT procedure instead of the eliminator replacement step 3.2, it can be shown that \( b(t) \leq c t \) for some constant \( c \). More generally, the following theorem is obtained.

**Theorem 6.** For each \( t \geq 1 \), asymptotically in \( n \),

\[
V_t(n) \leq n + (6t + a(t)) n \log n + o(n \log n)
\]

where

\[
\lim_{t \to \infty} \frac{a(t)}{t} = 0.
\]

Apparently no non-trivial lower bounds on \( V_t(n) \) for \( t < n \) are known. It is conceivable that the ideas touched on in the final paragraphs of section 2 could be formalized and utilized to establish the following lower bound to complement theorem 6.

**Conjecture:** There exists a constant \( c > 0 \) such that for every fixed \( t \geq 2 \),

\[
V_t(n) \geq n + ct \log n + o(n \log n).
\]
References


