KYBERNETIKA — VOLUME *17* (1981), NUMBER 2

BOUNDS FOR ON-LINE SELECTION

PETER RUŽIČKA

In this paper we show that

1. given a constant k the minimum number of binary comparisons necessary for the on-line selection of the k-th largest element of a totally ordered set X of size n is bounded below by $[\log_2 (k + 1)] \, . \, n - 0(1)$;
2. given a rational number $r \in (0, \frac{1}{2}\rangle$ there is a positive integer constant c_r such that the minimum number of binary comparisons necessary for the on-line selection of the $[r \, . \, n]$-th largest element of X is bounded below by $c_r \, . \, n \, . \, \log_2 n - 0(n)$.

1. INTRODUCTION

The theory of complexity of computations addresses itself to the quantitative aspects of the solutions of computational problems. There are usually several possible algorithms for solving a given problem. With each of the algorithms there are associated certain significant cost functions such as the time or the space as a function of the problem size. A lot of questions concerning complexity can be raised with respect to a given problem. The fundamental question is to establish and prove a lower bound for one of the cost functions associated with the algorithm.

In this paper we investigate the time complexity of the selection problem under on-line restriction. Given a set X of n distinct elements and an integer k, $0 < k \leqq n$, the selection problem is to determine the minimum number of pairwise comparisons needed to select the k-th largest element of X. In order to state with precision the amount of time consumed by an algorithm we need a well defined formal model of a computer. Our model of computation is a finite comparison tree with a finite number of memory cells and a one-way read-only input tape. Input elements are read from the input tape into specified memory cells. Comparisons may be made only between the contents of any pair of memory cells. We say an algorithm is on-line if it outputs the result on i input elements just before reading the $(i + 1)$-th input element for $i \geqq 1$. Otherwise it is called off-line.

An efficient off-line algorithm solving the selection problem was discovered by Paterson, Pippenger and Schönhage [1]. They have set up a method which needs only $3 \,.\, n + o(n)$ binary comparisons. In order to make it possible to determine how close is the given algorithm to the optimality, lower bounds on the complexity have been examined. The best lower bound result for off-line median selection problem is $\frac{11}{6} \,.\, n$ binary comparisons due to Yap [2].

In this paper we prove asymptotically optimal time bounds for on-line selection problem. We show that each on-line algorithm selecting the k-th largest element of X (for a constant k) must perform at least $\lceil \log (k + 1) \rceil \,.\, n - 0(1)$ binary comparisons.*) If $k = \lceil r \,.\, n \rceil$ for some rational number r from the interval $(0, \frac{1}{2}\rangle$, then there is an integer c_r such that each on-line selection algorithm computing the k-th element of X must perform at least $c_r \,.\, n \,.\, \log n - 0(n)$ binary comparisons.

2. PROBLEM AND MODEL OF COMPUTATION

One of the most widely studied problems in concrete complexity is the general selection problem. In this problem, we are given a set $X = \{a_1, a_2, \ldots, a_n\}$ of n different elements which has an implicit total ordering, and we are to find the k-th largest element of them using only binary comparisons on the input elements.

To precise the description of the general selection problem we present it as a special case of a more general so called decision problem. Consider the set S of all $n!$ orderings on X and let P be a partition over S. The decision problem $\mathscr{D}(S, P)$ is the problem for arbitrary ordering $w \in S$ to determine a partition set A from P such that $w \in A$. Now the general (k-th) selection problem can be defined as a decision problem $\mathscr{D}(S, \{Q_1, \ldots, Q_n\})$, where $Q_i = \{a_{k_1} \ldots a_{k_n} \mid a_{k_i}$ is the k-th largest element of $\{a_1, \ldots, a_n\}\}$.

To illustrate the connection between the decision problem and the general selection problem we take the case $k = 1$. A problem of selecting the maximal element from $X = \{a_1, \ldots, a_n\}$ can be described as a decision problem $\mathscr{D}(S, \{P_1, \ldots, P_n\})$, where $P_i = \{a_{k_1} \ldots a_{k_n} \mid a_{k_i} = \text{maximum}\, \{a_1, \ldots, a_n\}\}$. Here P_i is a set of all those orderings from S which have the maximal element on their i-th position. Hence, the problem of selecting the maximum element is identical with the problem of determining such a partition set to which the given ordering belongs.

In order to state with precision the amount of time consumed by an algorithm we need a well defined model of a computer. Our model can be viewed as a random access machine [3] with limited memory, restricted set of instructions and an input tape. The model of computation consists of a finite sequence $r_0, r_1, \ldots, r_p$ of registers

*) Remember that $[a]$ always denotes least integer equal to or greater than a.

(so called internal memory) together with a program. Each register is capable of holding any one integer. The program is in fact a finite comparison tree (for a finite number of input elements). Each such a program can be described by the following instructions:

```
read $(r_i)$
$r_i \leftarrow r_j$
if $a$ COMP $b$ then $S_1$ else $S_2$
ACCEPT
$r_i \leftarrow c$ ($c$ is a constant)
```

where a and b may be any operands of the form r_i, indicating the contents of register r_i; the relation COMP may be any of the binary symbols $<$, $\leqq$, $>$, $\geqq$, where these symbols have their usual interpretation. The effect of each of these instructions should be evident. For example, **read** (r_5) causes register r_5 to assume the value of the next input element and as a side effect the movement of the input head one step to the right. S_1 and S_2 are either statements of the type **if** – **then** – **else** prefixed by some **read** (r_i) instructions or instruction ACCEPT. The program is of the form

```
begin
  read $(r_i)$; ...; read $(r_j)$;
  if $a$ COMP $b$ then $S_1$ else $S_2$
end
```

Using this assembly language the programs are often difficult to write and even more difficult to read. So it is desirable to have a high level language to state programs in. One can safely use a small, but still rich, subset of some high level language which is easier to read than assembly, which unable us to write programs in a finite form, and which allows for easy estimation of the time needed to implement programs on our model. We can add to the program instructions some high level constructs as **for** – **until** – **do**, **while** – **do**, **repeat** – **until**, **recursion** or **goto** and so on. But we have in our mind the fact that each program specified in the high-level language and working over a finite input has to be directly transformed into an equivalent version of an assembly program.

Our model of computation receives a finite set of elements as input. A computation proceeds as follows. Initially, the input tape contains all elements of reservoir and all registers (i.e. memory cells) are set to be zero. The input tape is one-way read-only tape. The program is started at the first instruction. The instructions are executed in order until a conditional branch is encountered. If the comparison a COMP b is true, then the instruction is equivalent to S_1; if the comparison is false, then the instruction is equivalent to S_2. The program terminates as soon as some of the following types of instructions are encountered: ACCEPT or an instruction involving relations between operands which do not correspond to reservoir singletons. The result of the computation is obtained in the registers.

As an example consider the following simple problem of selecting the maximal element of the set X.

```
procedure SELECTMAX (X);
begin
  read (r_1);
  while the input is nonempty
  do
    begin
      read (r_2);
      if r_1 < r_2 then r_1 ← r_2
    end
end
```

In following algorithms we denote registers $r_1, r_2, \dots$ as $a, b, c, \dots$. We note that SELECTMAX algorithm is an on-line algorithm because before reading the k-th input element, it computes the maximal element over the set of $k - 1$ already read ones.

3. RESULTS

We first consider two special cases of the selection problem.

Fact 1. Each on-line algorithm selecting the second element of a totally ordered set of size n must perform at least $2n - 3$ binary comparisons.

Proof. The proof is based on the adversary argument. The adversary strategy is constructed in such a way that each element eliminated by the selection algorithm has to be involved in at least two different binary comparisons. Assume that the memory of the selection algorithm is represented in the form of the directed acyclic graph (further dag), whose nodes correspond to elements in memory cells and whose edges correspond to the ordering between elements in memory cells as specified by the partial order underlying the previous computation. As the computation proceeds, the dag is modified in the following way. Initially, it consists of three nodes and no edges and at each step of computation an edge between two compared elements is added and it is oriented in order to minimize the number of paths of the length greater than one. Whenever the node occurs from which two other nodes in the dag are reachable, it is eliminated and replaced by a singleton from the input reservoir. We next show that if a dag with 3 nodes is considered and there are additional $n - 3$ singletons in the input reservoir, then 2 comparisons are necessary for each of

$n - 2$ eliminations and 1 comparison is necessary to determine the second element, thus giving together $2n - 3$ binary comparisons. Consider the following adversary strategy, which give the prescription for answering an arbitrary binary comparison of the form $a : b$ of the selection algorithm.

```
procedure ADVERSARY (comparison $a : b$);
begin
  comment elements from the internal memory will be denoted as $\{a, b, c\}$;
  select
    no relation between $a, b, c$ is known: return $(a > b)$;
    $a > c$ or $b < c$ is known: return $(a > b)$;
    $a < c$ or $b > c$ is known: return $(a < b)$;
    $a < c$, $b < c$ is known: return $(a > b)$
  end;
  if $x$ is the minimal element of $\{a, b, c\}$
  then $x$ is deleted from the internal memory
end
```

The bound follows directly from the straightforward case by case analysis over all possible relations between elements of internal memory. □

We note that the trivial on-line selection algorithm for determining the second element is doing $2n - 3$ binary comparisons and needs 3 memory cells and thus it is time optimal.

Fact 2. Each on-line algorithm selecting the third element of a totally ordered set of the size n must perform at least $2n - 4$ binary comparisons.

Proof. Again the adversary argument is used. The adversary strategy will follow the principal idea to answer comparisons of the selection algorithm such that each eliminated element will be involved in at least two various binary comparisons with elements in the internal memory. Such a strategy could be the following:

```
procedure ADVERSARY (comparison $a : b$);
begin
  comment internal memory elements are denoted as $\{a, b, c, d\}$;
  select
    no relation is known among elements $\{a, b, c, d\}$: return $(a > b)$;
    $c > b$ or $a > c$ is known: return $(a > b)$;
    $c > a$ is known: return $(b > a)$;
    $a > c$, $a > d$ or $c > a$, $c > b$ or $c > d$, $a > d$ or
    $a > c$, $b > d$ or $c > a$, $d > b$ or $c > a$, $c > d$ or
    $a > c$, $d > b$ or $a > c$, $c > d$ or $c > a$, $a > d$ or
```

```
        $d > c,\ c > b$ is known: **return** $(a > b)$;
        $c > d,\ c > a$ or $c > a,\ d > a$ is known: **return** $(b > a)$;
        $c > a,\ c > b,\ b > d$ or $a > c,\ a > d,\ b > d$ or
        $c > a,\ c > b,\ d > b$ or $c > a,\ c > b,\ c > d$ or
        $d > c,\ c > a,\ c > b$ is known: **return** $(a > b)$
    **end**;
    **if** the minimal element $x$ from $\{a, b, c, d\}$ is known
    **then** eliminate $x$ from the internal memory
**end**
```

We note that all relationships between elements mentioned in the algorithm ADVERSARY are taken up to the isomorphic ones. The estimation follows directly from the straightforward examination of all ordering possibilities between elements in the internal memory. □

Again, the trivial on-line selection algorithm for the third element is doing $2n - 4$ binary comparisons and needs 4 memory cells and thus it is time optimal.

Now we turn our attention to the general case of the selection problem. First we consider the case of selecting the k-th element when k is a constant.

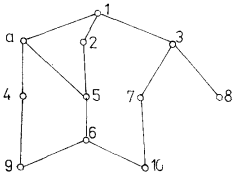
Theorem 1. Given a constant $k > 0$ each on-line algorithm selecting the k-th largest element of a totally ordered set of size n must perform at least $[\log_2 (k + 1)] \, . \, n - 0(1)$ binary comparisons.

Proof. We construct an adversary strategy which will give an answer to an arbitrary comparison "$a : b$" of the selection algorithm such that each eliminated element will be involved in at least $[\log_2 (k + 1)]$ binary comparisons between elements simultaneously contained in the internal memory.

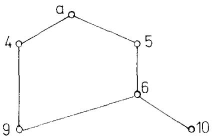
```
**procedure** ADVERSARY (comparison $a : b$);
**begin**
  **comment** there are $k + 1$ elements in the internal memory, $a$, $b$ are among them;
  1. A binary tree $T_a$ (respectively $T_b$) with a root $a$ (respectively $b$) is constructed
     in the following way:
     i.  Let $H = \langle N, E \rangle$ be a Hasse's diagram of a partial order over all elements
         in the internal memory with a set of nodes $N$ corresponding to all elements
         in the internal memory and with a set of edges $E$ such that there is an edge
         $c \to d$ in $E$ iff a comparison with the result $c < d$ was done in the previous
         computation, $c, d \in N$.
     ii. Delete all nodes from $N$ (together with incidental edges) such that there
         does not exist a path from $a$ (respeticvely $b$) to $n$. Denote the resulting
         directed acyclic graph as $\hat{H} = \langle \hat{N}, \hat{E} \rangle$.
```

```
    iii. Delete all crossing edges from $N$ (i.e. edges leading to the nodes of indegree 1
         during the computation of $H$). The resulting tree is denoted as $T_a$ (respectively $T_b$).
  2. **if** the number of nodes in the tree $T_a \geqq$ the number of nodes in the tree $T_b$
     **then return** $(a < b)$;
  3. **if** $x$ is known to be the minimal element from the internal memory
     **then** delete $x$ from the internal memory
**end**
```

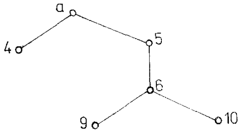
Consider an example of the Hasse's diagram H of the form



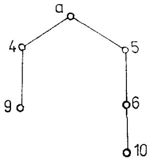
Then the dag $\hat{H}$ for the root a is of the form



The resulting tree T_a is either of the form



or of the form



depending on whether the comparison "4 : 9" has been done before the comparison "6 : 9" or vice versa.

The estimation of the number of binary comparisons forced by this adversary strategy is clear by using the following argument: the eliminated element x must be the root of the binary tree T_x with $k + 1$ nodes and so the element x must be compared with at least $\lceil \log_2 (k + 1) \rceil$ other elements simultaneously contained in the internal memory. □

A simple on-line selection algorithm for the k-th largest element (where k is a constant) constructs a binomial tree in the internal memory with a root corresponding to the minimal element. Binomial trees T_k are defined inductively as follows: a single node forms T_0 and two T_k with an additional edge between roots form T_{k+1}. An alternative representation of a binomial tree T_{k+1} can be viewed as a root of $k + 1$ sons being roots of $T_k, T_{k-1}, \ldots, T_0$, respectively. The time complexity of this algorithm is $\lceil \log (k + 1) \rceil . n - 0(1)$ and as follows from Theorem 1 this algorithm is time optimal.

Theorem 2. For any rational number $r : 0 < r \leqq \frac{1}{2}$ there exists a constant $c_r > 0$ such that each on-line algorithm determining $\lceil r . n \rceil$-th largest from n elements must perform at least $c_r . n . \log n - 0(n)$ binary comparisons.

Proof. We firstly consider the case $0 < r < \frac{1}{2}$. We show how an adversary strategy can be constructed such that each of the first $\min (r, 1 - 2r) . n$ eliminated elements is forced to be compared in average with at least $d_r . \log n - 0(\log r)$ other elements from the internal memory for some constant $d_r > 0$. If we take $\lceil r . n \rceil$ instead of k in the adversary strategy of the previous theorem, then the above proposition can be proved using step by step simulation of that proof. Here we present another adversary strategy in order to prove the theorem.

```
procedure ADVERSARY (comparison $a : b$);
begin
  comment initially, all elements among the first $\lceil r . n \rceil$ read ones are logically in $U$;
          Consider a complete binary tree with $2^{\lceil r.n \rceil} - 1$ nodes. Order the nodes
          of this tree in an inorder way and attach a set $U_i$ to each node with the
          ordering number $i$. Initially, let $U_{2^{\lceil r.n \rceil - 1}}$ to be equal to $U$; $U_j = \emptyset$,
          $j \neq 2^{\lceil r.n \rceil - 1}$; Elements $a$, $b$ are among $\lceil r . n \rceil + 1$ elements of the internal
          memory;
  select
    $a$ is among the first $\lceil r . n \rceil$ read elements, $b$ is not:
      return $(a < b)$;
    Both $a$ and $b$ are not among the first $\lceil r . n \rceil$ read elements:
      return $(a < b)$;
    $b$ is among the first $\lceil r . n \rceil$ read elements, $a$ is not:
      return $(a > b)$;
   Both $a$ and $b$ are among the first $[r \,.\, n]$ read elements:
      begin
         comment let $a \in U_i,\ b \in U_j$;
         select
            $i < j$ : return $(a < b)$;
            $i = j$ : return $(a > b)$;
            $i > j$ : change $a$ and $b$ and follow as in the case $i < j$
         end;
         comment let $i = (1 + 2\alpha) \,.\, 2^u,\ j = (1 + 2\beta) \,.\, 2^v$
                 for some $u, v \in \langle 1, [r \,.\, n] - 1 \rangle$,
                 $\alpha \in \langle 0, 2^{[r.n]-1-u} \rangle;\ \beta \in \langle 0, 2^{[r.n]-1-v} \rangle$;
         $a$ is transfered from $U_i$ to $U_{i+2^{u-1}}$;
         $b$ is transfered from $U_j$ to $U_{j-2^{v-1}}$
      end
   end
end
```

During the performance of the on-line selection algorithm two cases can occur: either the selection algorithm eliminates the minimal element from the set of $[r \,.\, n] + 1$ elements in the internal memory, and then after $[r \,.\, n]$ eliminations it makes at least

$$\left[\frac{r}{2} \,.\, n \,.\, ([\log_2 (r \,.\, n)] + 1)\right] - 2^{[\log(r.n)]-1}$$

binary comparisons or the selection algorithm eliminates the maximal element from $[r \,.\, n] + 1$ elements in the internal memory, and then during this execution at least

$$\left[\frac{1 - 2r}{2} \,.\, n \,.\, ([\log_2 ((1 - 2r) \,.\, n)] + 1)\right] - 2^{[\log((1-2r).n)]-1}$$

binary comparisons are needed. Both estimations follows from the minimal sum of lengths of all paths over all binary trees with either $r \,.\, n$ or $(1 - 2r) \,.\, n$ nodes. By this way an adversary strategy ADVERSARY was constructed which forced each selection algorithm for $[r \,.\, n]$-th largest element to make at least $c_r \,.\, n \,.\, \log n - 0(n)$ binary comparisons for some constant $c_r > 0$.

Now it is sufficient to prove the case $r = \frac{1}{2}$. An optimal on-line median selection algorithm for a set of n elements eliminates those elements from the internal memory which are known to be lower or greater than $[n/2] + 1$ other elements. Consider the computation state in which h elements greater than the median element and d elements lower than the median element has already been eliminated. There are just two candidates for the next eliminated element. Either the selection algorithm computes an element from the internal memory which is lower than the median or the selection algorithm computes an element which is greater than the median element. We present the strategy by means of which the following result will be obtained: if the eliminated element is lower than the median, then it is compared with at least $[\log(n/2 - h)]$ other elements from the internal memory; otherwise it is compared with at least $[\log(n/2 - d)]$ other elements from the working space.

```
procedure ADVERSARY (comparison $a : b$);
begin
  comment Consider that in the previous computation $h$ elements greater than the
          median and $d$ elements lower than the median has already been elimi-
          nated by the selection algorithm; Let $a$, $b$ be among $[n/2] + 1$ elements
          of the internal memory;
  1. A binary tree $T_a^<$ with a root $a$ is constructed in the following way:
     i.   Let $H = \langle N, E \rangle$ be a Hasse's diagram of a partial order (using $<$) over all
          elements in the internal memory with a set of nodes $N$ corresponding to all
          elements in the internal memory and with a set of edges $E$ such that there
          is an edge $c \rightarrow d$ in $E$ iff a comparison with the result $c < d$ was done in the
          previous computation, $c, d \in N$;
     ii.  Delete all nodes $n$ from $N$ (together with incidental edges) such that there
          does not exist a path from $a$ to $n$. Denote the resulting directed acyclic
          graph as $\hat{H} = \langle \hat{N}, \hat{E} \rangle$;
     iii. Delete all crossing edges from $\hat{N}$. The resulting tree is denoted as $T_a^<$;
  2. Analogically construct trees $T_a^>$, $T_b^<$, $T_b^>$;
  3. if $[n/2] - h -$ {the number of nodes in the tree $T_a^>$ + the number of nodes in
       the tree $T_b^>$}
       $\leqq [n/2] - d -$ {the number of nodes in the tree $T_a^<$ + the number of nodes
       in the tree $T_b^<$}
     then
       if the number of nodes in the tree $T_a^> \geqq$
       the number of nodes in the tree $T_b^>$
       then return $(a > b)$
       else return $(a < b)$
     else
       if the number of nodes in the tree $T_a^< \geqq$
         the number of nodes in the tree $T_b^<$
       then return $(a > b)$
       else return $(a < b)$
end
```

The estimation of the number of comparisons forced by this adversary strategy directly follows from the following argument. The eliminated element x must be the root of the binary tree $T_x^<$ (or $T_x^>$) depending on the fact whether the maximal or minimal element is eliminated. The binary tree has totally $n/2 - h$ (respectively $n/2 - d$) nodes and therefore at least $\lceil \log (n/2 - h) \rceil$ (respectively $\lceil \log (n/2 - d) \rceil$) binary comparisons has to be done.

(Received April 10, 1980.)

REFERENCES

[1] M. Paterson, N. Pippenger, A. Schönhage: Finding the median. Journal of Computer and System Sciences *13* (1976), 184—199.

[2] Chee - Keng Yap: New Lower Bounds for Median and Related Problems. Yale University, Department of Computer Science, Research Report No. 79, 1976.

[3] A. V. Aho, J. E. Hopcroft, J. D. Ullman: The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, 1974.

RNDr. Peter Ružička, Výskumné výpočtové stredisko (Computing Research Center), Dúbravská cesta 3, 885 31 Bratislava. Czechoslovakia.