OPTIMAL TIME MINIMAL SPACE SELECTION ALGORITHMS*

by

David Dobkin§

and

J. Ian Munro†

Research Report CS-79-24

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

§ Department of Computer Science
University of Arizona
Tucson, Arizona

† Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

* Portions of this research were supported by the National Science Foundation under Grant MCS76-11460 and the National Research Council under Grant A8237.

OPTIMAL TIME MINIMAL SPACE SELECTION ALGORITHMS*

By

David Dobkin
Department of Computer Science
University of Arizona
Tucson, Arizona

J. Ian Munro
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

Algorithms for finding medians and solving arbitrary selection problems in minimum space are investigated. A linear time algorithm is given in the first case, and it is shown that no such scheme exists for many other interesting selection problems, such as finding a quartile. A tight trade-off is demonstrated balancing extra space versus time.

---

*Portions of this research were supported by the National Science Foundation under Grant MCS76-11460 and the National Research Council under Grant A8237. This research was begun when both authors were in Pittsburgh, continued by the first author at Yale and the second on the Thaynes Lift at Park City, Utah, and extended while the first author visited Waterloo and the second visited Yale. Portions of Section 3 were presented at the 1977 CISS Conference at which time the results of Section 4 were discovered. Finally, the research was completed while the second author was visiting the University of Warwick. Portions of Section 4 were presented at ICALP V in Udine, Italy. The final paper was written in Tucson, Arizona one sunny Friday.

1. INTRODUCTION

During the past few years, considerable attention has been focused on the discovery and improvement of algorithms for computing the t-th largest from a set of n elements [3]. Of particular interest have been algorithms for computing medians (or doing particular selections) that have running times which are linear in the number of elements [2,5]. The existence of such algorithms has created the following open question:

> What is the minimum space required by an algorithm which, with a single pass through its input determines the median using a number of comparisons linear in the number of inputs?

In this paper we show that a linear algorithm is possible for the median problem which requires a workspace of only $\lceil n/2 \rceil + 1$ cells. Since the generation of the median of a set requires showing that all but one element have the property of being either larger or smaller than $\lceil n/2 \rceil$ of the other elements, this space requirement is clearly minimal. We also consider the more general problem of determining the k^{th} largest element of a set (for arbitrary k) under similar storage constraints. We note that our scheme does not generalize to a (simultaneously) linear time and minimum space selection algorithm for arbitrary ranks. Indeed we show that no such algorithm exists and demonstrate a time-space trade-off for such selection problems. En route to these results, attention is turned to the problem of determining the elements of an arbitrary set of ranks in a near minimal number of comparisons (without space constraints).

Throughout this paper our model of computation is the RAM model presented in [1] with the added constraint that the memory of the machine which can contain data elements may, at times, be limited. The RAM will consist of a one way read-only input tape, a write-only output tape and k words of internal memory each capable of storing a single data element. We will not concern ourselves with the details of storage required for indices and so shall refer to computations on such a machine as being of space complexity k. Our basic operation will be the comparison and our time complexity measure will be the number of comparisons possible for an algorithm on its worst case input. Throughout, whenever we require the median or arbitrarily ranked element of a set of t elements and have t available words of memory, we will use a known linear algorithm for computing such an element and denote its complexity by M(t). At present, the best known algorithm is that of Schönhage, Paterson and Pippenger [5] giving M(t) as $3t + o(t)$.

2. THE COMPLEXITY OF MULTIPLE SELECTION

An essential part of the minimal space median algorithm involves selecting a subset of elements having specified ranks from a set of n elements. Since the general multiple selection problem is of interest, we present here upper and lower bounds on this problem.

We represent the complexity of selecting the $i_1,i_2,\ldots,i_k$ largest of a set of n elements as $ms(n; i_1,i_2,\ldots,i_k)$ and establish the following results, throughout our discussion, we adopt the conventions $i_j < i_{j+1}$, $i_o=0$ and $i_{k+1}=n+1$

Theorem 1:

$$ms(n; i_1,i_2,\ldots,i_k) \geq n\log n - \sum_{i=0}^{k} (i_{j+1}-i_j) \log (i_{j+1}-i_j) - O(n)$$

Proof: We observe that having determined, by performing comparisons alone, the $i_1,i_2,\ldots,i_k$ ranking elements in the set, we must know which elements lie between elements i_j and i_{j+1} for $j=1,\ldots k-1$. If we then sort each of these sets, we will have found the sorted order of the entire list. Since $\ell \log \ell - O(\ell)$ comparisons are both necessary and sufficient for sorting a set of ℓ elements, we observe that our bound follows since we can transform the output of our multiple selection algorithm into a sorted set of n elements in a total of

$$\sum_{i=0}^{k} (i_{j+1} - i_j) \log (i_{j+1} - i_j) - O(n)$$

comparisons. The theorem follows by subtraction of this time bound from the lower bound for sorting the entire list. The bound holds not only in the worst case, but also on the average. □

Next we present an algorithm which approaches this bound.

Theorem 2:

$$ms(n; i_1, i_2, \ldots, i_k) = \theta \left(\max\left(n, \left(n \log n - \sum_{j=0}^{k} (i_{j+1} - i_j) \log (i_{j+1} - i_j)\right)\right)\right)$$

Proof: The lower bound follows from the previous theorem, to demonstrate the upper bound, we propose the following algorithm.

Algorithm MULTISELECT $(n: i_1, i_2, \ldots, i_k)$ for finding the $i_1, i_2, \ldots, i_k$ largest elements of a set of n elements.

If k = 1

then

find the i_1st largest element of the set

else

for the i_j closest to n/2 find the i_jth largest element of the set and call MULTISELECT$(i_j - 1; i_1, \ldots, i_{j-1})$ on the $i_j - 1$ largest elements of the set and MULTISELECT $(n - i_j; i_{j+1} - i_j, \ldots, i_k - i_j)$ on the $n - i_j$ smallest elements.

It is clear that the run time of this algorithm leads to the recurrence

$$ms(n;i_1,i_2,\ldots,i_k) \le ms(i_j - 1;i_1,i_2,\ldots,i_{j-1}) + ms(n - i_j;i_{j+1} - i_j,\ldots,i_k - i_j) + M(n) \quad (1)$$

where M(n) is the number of comparisons necessary to do a selection from a set of n elements ($M(n) \le 3n + o(n)$). We prove the theorem by induction on k, the number of selections to be done. For k=1, the algorithm requires M(n) operations which obviously satisfies the theorem. To simplify what follows, we define $f(n;i_1,\ldots,i_p)$ as

$$\max\left(n,\ n \log n - \sum_{j=0}^{p} (i_{j+1} - i_j) \log (i_{j+1} - i_j)\right)$$

Now, assume that the theorem is true for all $\ell < k$, so that there is a constant c, with

$$ms(n;\ i_1,\ldots,i_\ell) \le c \cdot f(n;\ i_1,\ldots,i_\ell)$$

Then, from (1) it follows that

$$ms(n;\ i_1,\ldots,i_k) \le c \cdot f(i_j - 1;\ i_1,\ldots,i_{j-1}) + c \cdot f(n - i_j;\ i_{j+1},\ldots,i_k) + M(n) \le c \cdot f(n;\ i_1,\ldots,i_k)$$

from which the theorem follows. Given that $M(n) \le 3n + o(n)$, we note

$$ms(n;\ i_1\ldots,i_k) \le 3 \cdot f(n,\ i_1,\ldots,i_k) + o(f(n;\ i_1,\ldots,i_k))$$

□

3. A MINIMAL SPACE LINEAR TIME MEDIAN FINDER

We turn now to the main problem of the paper, and present a minimal space median finder. Our algorithm involves an initializing step in which the first $n/2 + 1$ elements of the input are organized using the multiple selection algorithm of the previous section. Next, we begin an iterative process using this organization to eliminate 2^i non-medians at the i^{th} step. Deleted elements are always balanced so that the median of remaining elements is the median of the original set. We continue this process until about half of the original elements have been deleted whence any linear median finder can be applied to obtain the final result.

We present the algorithm here for inputs of size $2^k - 1$. The generalization to inputs of arbitrary size is immediate.

Algorithm MEDIAN

Input: A set of $2^k - 1$ elements

Output: The median of the set

Workspace: $2^{k-1} + 1$ cells of memory

Step 1: Read in the first 2^{k-1} elements into the workspace and, using MULTISELECT find the $2^i - 1$ largest and smallest of these elements (for $i = 1, \ldots, k - 2$). This leaves the initial portion of the input partitioned into sets $B_1, \ldots, B_{k-2}$ and $S_1, \ldots, S_{k-2}$, where B_i contains the elements ranked 2^{i-1} through $2^i - 1$ largest (inclusive), and

S_i is similarly defined for the smallest elements. The remaining element constitutes a set, J. Initialize sets R (residue) and I (input) as empty.

Step 2: For i = 1 step 1 until k - 2 do

begin

Let I denote the next 2^{i-1} input elements

Logically form $T = I \cup R \cup S_i \cup B_i$

Find and discard the 2^{i-1} largest and 2^{i-1} smallest elements of T

Set R to be the remaining elements of T

end

Step 3: Read in the remaining elements

Find and output the median of the $2^{k-1} + 1$ elements remaining in $R \cup J$.

We observe that in Step 2 the i^{th} deletion step involves a balanced set consisting of 2^{i-1} elements on each side of the median. Furthermore, the 2^{i-1} largest of these elements are larger than the $2^i - 1$ elements placed into R, the 2^{i-1} smallest elements, all elements in $B_{i+1},\ldots B_{k-2}$, J and $S_{i+1},\ldots S_{k-2}$ and all $2^{i-1} - 1$ previously deleted small elements. This is a total of

$$2^i - 1 + 2^{i-1} + \sum_{j=i+1}^{k-2} |B_j| + \sum_{j+i+1}^{k-2} |S_j| + |J| + 2^{i-1} - 1$$

or

$$2^i - 1 + s^{i-1} + \sum_{j+i+1}^{k-2} 2^{j-1} + \sum_{j=i+1}^{k-2} 2^{j-1} + 1 + 2^{i-1} - 1 = 2^{k-1} - 1$$

elements. Hence none of these elements could have been the median. A similar count shows that none of the elements deleted as being too small could have been the median. Because of the balance of deletion, it is clear that the median of all elements remaining in $R \cup \bigcup_{j=i+1}^{k-2} (S_i \cup B_i)$ or on the input tape is the median of the entire set ($i=1, \ldots, k-2$). Hence, the result of Step 3 yields the actual median and we have

Theorem 3: Algorithm Median correctly computes the median of $2^k - 1$ elements using the minimal space, $2^{k-1} + 1$ memory cells.

Proof: We have shown above that the algorithm correctly computes the median. Now, we observe that initially 2^{k-1} cells of memory are used to form J, the B_i and S_i, and one new input is added to the first iteration of Step 2. In the i^{th} iteration of Step 2, 2^i elements are deleted, allowing for 2^i new elements to be input at the next iteration. Hence, no more than $2^{k-1} + 1$ cells are ever used for Step 2. During the execution of Step 2, a total of $\sum_{i+1}^{k-2} 2^i = 2^{k-1} - 1$ elements are discarded, hence there is room for the remaining elements together with all remaining inputs in Step 3.

That the storage used by this algorithm is minimal, follows from an adversary argument which can make any of the first 2^{k-1} elements the median if it is discarded before the $2^{k-1} + 1$ element is read. □

Theorem 4: It is possible to compute the median of a set in linear time and minimal space.

Proof: It remains to prove that median operates in linear time. We observe that Step 1 requires

$$ms(2^{k-1}+1;\ 1,3,\ldots 2^{i}-1,\ldots,2^{k-2}-1,2^{k-2},\ldots 2^{k-1}-2^{i},\ldots 2^{k-1}-1)$$

or

$$\Theta((2^{k-1}-1)\log(2^{k-1}-1) - 2\sum_{i=0}^{k-3}(2^{i+1}-2^{i})\log(2^{i+1}-2^{i})$$

At the i^{th} iteration of Step 2, we find the first and third quartiles of a set of $2^{i+1}-1$ elements requiring $2M(2^{i+1}-1)$ operations. By the linearity of M(r), the total work in Step 2 is

$$\sum_{i=1}^{k-2} 2M(2^{i+1}-1)\ 3\cdot 2^{k+1}+o(2^{k}) = O(n)\ .$$

Finally, in Step 3, finding the median of a set of $2^{k-1}+1$ elements requires O(n) operations. □

4. MINIMAL SPACE SELECTION ALGORITHMS

Surprisingly, the algorithms of the previous section do not immediately extend to arbitrary selection problems. In fact, the following anomolous situation results:

Theorem 5: For every $p \varepsilon (0,1)$ such that $p \neq 1/2$, there exists a constant $c_p > 0$ such that $c_p\, n \log n - O(n)$ comparisons are necessary to determine the pn^{th} largest element of a set of n elements in minimal storage.

Proof: Without loss of generality we will assume $0<p<1/2$, and so $pn+1$ storage locations are necessary to solve our problem in a single pass. The case in which $p>1/2$ is completely analogous. The proof is by the construction of an adversary strategy.

We observe that in order to remove an element from consideration as the pn^{th} largest, we must show that either pn elements are larger than it or that (1-p)n are smaller. Failure to do so premits an adversary to force the algorithm to produce an incorrect result. We therefore assume this strategy is followed by any algorithm.

Let $\tilde{p}$ denote min (p, 1-2p), and assume the first $\tilde{p}n$ elements which are read are smaller than the next pn. This forces any algorithm to reject the smallest element still in the system as each of the final $\tilde{p}n$ of the first $(p+\tilde{p})n$ elements is read. Effectively, then, we have forced the sorting of $\tilde{p}n$ elements, and have proved our

our theorem. With some care we can extend this argument to show that $p(1-2p)\, n \log n - O(n)$ comparisons are necessary to find the pn^{th} largest element in minimum space when $0<p<1/2$, and (substituting $(1-p)$ for p) $(1-p)\,(2p-1)\, n \log n - O(n)$ comparisons are required when $1/2<p<1$. □

Perhaps surprisingly, only a small increase in storage is necessary in order to make this problem feasible as the following theorem shows.

Theorem 6: For all $\varepsilon>0$, the pn^{th} largest or smallest element of a set may be found in linear time and $(p+\varepsilon)n$ space.

Proof: The bound is achieved by using the extra space to enable us to eliminate εn elements at a time by having $(p+\varepsilon)n$ elements in the storage locations and using a linear selection algorithm to find the εn smallest. Since the linear selection algorithm requires only $M((p+\varepsilon)n)$ operations to do the necessary elimination, in a total of at most $\frac{1}{\varepsilon}M((p+\varepsilon)n)$ comparisons we are left with $pn+1$ elements of which the smallest is the desired result. It is obvious why this result does not generalize to satisfy the constraints of the previous theorem. □

These two results may be combined to result in a continuous time space tradeoff whereby we may measure the effects of added storage according to the following.

Theorem 7: If $\varepsilon(n) = o(n)$, then for every $p\varepsilon(0,1)$, $p\neq 1/2$, $\Theta(n \log \frac{n}{\varepsilon(n)})-O(n)$ comparisons are necessary and sufficient to determine the pn^{th} largest element of a set of n elements using $pn+\varepsilon(n)$ storage locations.

Outline of Proof: The proof of the upper bound follows immediately from that of Theorem 6. The argument for the lower bound is similar to that used in the proof of Theorem 5.

We observe that we must output a structure consisting of $\min(p,1-2p)n$ elements divided into sets of ε elements. And within this structure, we know the relative sizes of differing sets of ε elements. A total of $\min(\frac{(pn)!}{(\varepsilon!)^{pn/\varepsilon}}, \frac{((1-2p)n)!}{\varepsilon!^{(1-2p)n/\varepsilon}})$ such structures exist from which an information theoretic argument generates the given bound. □

Putting this result in another way we see that if $\varepsilon(n) = n/k(n)$ extra space is available, then $\Theta(n \log k(n))$ extra time is required, providing an interesting time space tradeoff.

5. CONCLUSION

From a consideration of algorithms that solve arbitrary selection problems in minimal space an interesting anomoly has been explored. The median can be found in minimal space and linear time while all other percentiles require either a linear amount of extra space or nonlinear time. A tight time-space tradeoff is established in the latter case. Work on the multipass generalization of this problem appears in [4].

6. REFERENCES

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman,
The Design and Analysis of Computer Algorithms,
Addison-Wesley, Reading, Massachusetts, 1974.

[2] Blum,M., R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E. Tarjan,
Time Bounds for Selection,
JCSS, Vol. 7, No. 4, pp. 448-461.

[3] Knuth, D. E.
Sorting and Searching,
Addison-Wesley, Reading, Massachusetts, 1973.

[4] Munro, J.I. and M. Paterson,
Selection and Sorting with Limited Storage,
Proceedings of Nineteenth Symposium on Foundations of
Computer Science, October, 1978, pp. 253-258.

[5] Schönhage, A., M. Paterson and N. Pippenger
Finding the Median,
JCSS, Vol. 13, pp. 184-199, (1976).