A Minimal Space Selection Algorithm That Runs in Linear Time

by

David Dobkin and Ian Munro

Research Report #106

A MINIMAL SPACE SELECTION ALGORITHM THAT RUNS IN LINEAR TIME*

by

David Dobkin

Department of Computer Science

Yale University

New Haven, Connecticut

Ian Munro

Department of Computer Science

University of Waterloo

Waterloo, Ontario, Canada

ABSTRACT

An algorithm is given for computing medians in minimal space. This algorithm is shown to run in linear time and hence is asymptotically optimal with respect to time requirements and optimal with respect to space requirements. The algorithm is derived as an iterative procedure in which increasingly larger sets of data are shown to be non-medians. At each step of the algorithm, after suitable initialization, only a constant number of operations is done for each element deleted.

---

*This research was done while the first author was visiting the University of Waterloo and the second author was visiting Yale University. Portions of this research were supported by the National Science Foundation under Grant MCS76-11460 and the National Research council under Grant A8237.

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[K]. Of particular interest have been algorithms for computing medians or doing particular selections that have running times which are linear in n, the number of elements[BFPRT,SPP]. The existance of such algorithms has created the following open question:

> What is the minimum space required by an algorithm using only comparisons which computes the median in linear time?

In this paper we show that a linear algorithm is possible for the median problem which requires a workspace of only [n/2] + 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 n/2 of the other elements, this space requirement is clearly within an additive constant of being optimal. Variations of the algorithm yield similar results for other selection problems.

The algorithm is derived as an iterative scheme which is based on a procedure to eliminate sets of increasing size from consideration as the median at succesive iterations. After an initial preprocessing consisting of finding the largest, third largest, seventh largest,... and smallest, third smallest, seventh smallest... of the first half of the original list, it is possible to eliminate n/2 elements from consideration as possible medians by a set of operations which requires a constant number of comparisons to eliminate each single element. And, we show that the

initial preprocessing can be achieved in linear time. Continued application of this procedure results in an algorithm for finding the median in linear time and at no time in the computation are more than [n/2]+1 storage cells accessed beyond the read-only tape on which the original input is written.

2. MAIN RESULTS

We shall use as our model of computation the RAM model as presented in [AHU] with the added constraint that the memory of the machine will be limited. The RAM will consist of a oneway

read-only input tape, a write-only output tape and k words of internal memory. We 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 that may possibly be done by an algorithm for its worst case input. Throughout, whenever we require the median of a set of t elements and have t available words of memory, we will use the algorithm of [SPP] to compute this median. This algorithm requires $3t+o(t)$ comparisons to do this operation and is the currently best known algorithm for computing medians. Since we are showing the existance of a linear time algorithm, any linear time median algorithm could be used. We shall use the notation $M(t)$ to represent the complexity of computing the median of a set of t integers via a linear time median algorithm. Thus, at present $M(t)$ is $3t+o(t)$.

Basically, our algorithm works by iteratively considering larger sets of elements and showing for each set that half of its members cannot be medians. This is accomplished for a given element by showing that the element is either larger or smaller than at least half of the elements in the original set. Since our algorithm will always maintain a balance by discarding equal numbers of elements on each side of the actual median, this condition is equivalent to showing that a discarded element is larger or smaller than half of the remaining elements. We initialize our algorithm by taking the first half of the input and finding the largest, third largest, seventh largest,... of the set as well as the smallest, third smallest, seventh smallest, ... of the set. As our iteration we then consider the set of the 2^{i-1} largest remaining elements of the initial set, the 2^{i-1} remaining smallest elements of this set, 2^{i-1} new input elements and the $2^{i-1}-1$ elements remaining as the residue of the previous iteration. For this set, we find the sets consisting of the largest 2^{i-1} elements, the smallest 2^{i-1} elements and the remaining $2^{i}-1$ elements. The first two sets can be shown to be non-medians and the elements of the third set remain as the residue to be used in the next iteration. If the initial input consisted of $2^{k}+1$ elements, then after k-2 iterations, $2^{k-1}+3$ elements remain whose median is the median of the original set. This median can then be computed by any linear median algorithm.

Before presenting a thorough analysis of this algorithm, we present an example of its performance on a set of 33 elements. At the start, the first 17 elements of the input are read and their largest, 3-rd largest, 7-th largest, smallest, 3-rd smallest, and 7-th smallest elements are found. At the first iteration, we sort the largest and smallest of this

set along with a new element off the input tape. The largest of these three is at least as large as 17 inputs and hence is not the median. A similar argument handles the smallest. And the median of this set then forms the residue for the next iteration. At this iteration, we consider the 2 largest and 2 smallest remaining of the original set and two new inputs along with this residue. Of these seven elements, the two largest (resp. smallest) cannot possibly be the median as each is larger (smaller) than 17 of the elements of the original set. These four elements are discarded and the remaining three form the residue for the next iteration.

The algorithm can then be stated as the following Pidgin ALGOL procedure:

```
Procedure MSPMED

Input:A set X of 2^k+1 elements

Output: The median of the set
  begin
    1. Let L be composed of the first 2^(k-1)+1 elements of X
       and
          for i from k-1 to 0 step-1 do
          find the 2^i largest and smallest elements of
          L and appropriately partition the set such elements determined
          larger by this
          partition follow elements determined smaller.

    2. For i from 1 to k-2 do
```

begin

3. read in the next 2^{i-1} elements from X to the front of L.
4. find the upper and lower quartiles (2^{i-1} elements each) of the set of $2^{i+1}-1$ elements composed of the first 2^{i} and last 2^{i-1} elements of L
5. drop elements in the upper and lower quartiles, place other considered elements at the end of L (in the last $2^{i}-1$ locations)(the residue is now of size $2^{i}-1$ and 2^{i} locations are now vacant at the front of the list).

end

6. Find the median of the $2^{k-1}+3$ remaining elements
7. Output this median

end

Next we turn to a proof of the correctness of the above algorithm. We do so via a pair of assertions about its state at various stages in the computation.

Claim 1: For each i in the range $1 \leq i \leq k-2$, the elements dropped in step 5. of the algorithm cannot be medians.

Proof: We do the proof by induction, we observe that for i=1, one element is dropped at this stage because it has a value bigger than 2^{k-1} of the original elements as well as one new element and one element is dropped because it has a value smaller than 2^{k-1} of the original elements as well

as one new element. Furthermore, the elements dropped are larger (resp. smaller) than any remaining elements that have been considered. Assume that the claim is true for all i<I<k-3 and assume that in the first I-1 iterations of the loop, 2^I elements have been dropped, 2^{I-1} of which are larger than any of those remaining and 2^{I-1} of which are smaller than any of those remaining. Then at the Ith stage we have a residue set of 2^I-1 elements which we consider in connection with the 2^I remaining largest and 2^I remaining smallest element of the original set and with 2^I new inputs. We observe that elements in the top quartile of this set are larger than the $3*2^I-1$ elements in this set and are larger than the 2^I elements which have already been discarded for being too small as well as being larger than the $2^{k-1}+1-(2^{I+2}-2)$ remaining elements and hence each is larger than at least $2^{k-1}+1$ elements and cannot be the median of the original set.

Claim 2: The elements remaining after k-2 iterations of the for loop have the same median as the original set.

Proof : We observe that at each iteration of the for loop, equal numbers of elements are discarded for being too large and too small. Hence the remaining elements have the same center as did the original set.

Therefore we have the following:

Theorem: The above algorithm correctly computes the median of the set X while never using more than $2^{k-1}+3$ storage locations on an input of size 2^k+1

We now turn to a proof of the linearity of the algorithm.

Theorem:Algorithm MSPMED has a running time of O(N) on an input of size N.

Proof: (We assume that $N=2^k+1$ for some k, (other cases folow easily). the We observe that the only steps of the algorithm which require any work are steps 1,4 and 6. In step 1 we are computing the first, third, seventh,...2^{k-2}-1th largest and first, third,seventh,...2^{k-2}-1th smallest elements of the set L. We do this by recursively doing problems on smaller sets. To begin we find the 2^{k-2}-1 largest and smallest of the original set of 2^{k-1}+1 elements. This requires $2M(2^{k-1}+1)$operations. Next we find the 2^{k-3}-1 largest(smallest) of the first(second) of these sets requiring $2M(2^{k-2}-1)$ operations. Continuing this procedure, we see that the first step can be performed in

$$2M(2^{k-1} + 1) + 2M(2^{k-2} - 1) + 2M(3)$$

or $M(2^{k+1}-2(k+1))$ operations because of the linearity of M(.). Step 4 requires that for each i in the range $1 \leq i \leq k-2$, we find the upper and lower quartiles of a set of 2^{i+1}-1 elements. Each of these operations can be done in at most $M(2^{i+1}-1)+2M(2^i-1)$ operations and hence, at the ith iteration, at most $M(2^{i+2}-3)$ comparisons need be done. Thus, the entire set of iterations requires at most $M(2^{k+1}-2(k+1))$ comparisons. In the sixth step we are merely computing the median of a set of $2^{k-1}+3$ elements at a cost of $M(2^{k-1}+3)$ operations. The total cost is therefore bounded by $M(9x2^{k-1}-(4k+1))$ or 4.5M(N).

4. REFERENCES

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

[BFPRT] 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.

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

[SPP] Schonhage,A., M. Paterson and N. Pippenger
Finding the Median.
JCSS vol.13,pp.184-199 (1976).