Abstract

The complexity of a number of selection problems is considered. An algorithm is given to determine the mode a multiset in a number of comparisons differing from the lower bound by only a "lower order term." The problems of finding the kth largest element in a set in minimal and near minimal space are also discussed. A time space tradeoff is demonstrated for these problems.

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

Time and Space Bounds for Selection Problems†

by

David Dobkin

and

J. Ian Munro*

*University of Waterloo

TR 134

1. Introduction

The inherent complexity of sorting and selection problems is important both from a practical and theoretical point of view. Certainly the realization of an n log n comparison sorting algorithm is fundamental to practical computing while the O(n) median algorithms ((3)(10)) and lower bounds (8) strike at the heart of the complexity of "frequently computed functions." Furthermore, work of the latter type has clearly led to results of both mathematical and computational interest (9). In this paper we determine new bounds on the number of comparisons required for several problems, in particular finding the mode of a multiset, an arbitrary set of k-tiles in a list, and finding a k-tile in minimum space. While the thrust of the work is in dealing with worst case behaviour, and with algorithms of primarily mathematical interest, some of our techniques lead to simply implementable algorithms with near optimal average run time.

The basic operation of our model is the comparison of two inputs. In the next section the comparison is a 3-way (<,=,>) branch, while in the remainder of the paper the "=" branch can be assumed not to occur. The precise description of the model for dealing with minimum space is left until section 4.

2. Multisets and Modes

In (6) it is shown that given a list of n elements, k of which are distinct, and m_i are of the ith type (hence $\sum_{i=1}^{k} m_i = n$),

$$(n \log n - \Sigma\, m_i \log m_i - (n-k) \log \log k - O(n)$$

comparisons are necessary to sort the list on the average and $n \log n - \Sigma\, m_i \log m_i + O(n)$ suffice. Theorem 1 is a minor extension of this result.

Theorem 1: $n \log n - \Sigma\, m_i \log m_i - n \log(\log n - \frac{1}{n} \Sigma\, m_i \log m_i)$ comparisons are necessary to sort a multiset of n elements, k of which are distinct of m_i of the ith type, even if $\{m_i\}$ is given.

While we conjecture that the upper bound noted is within O(n) of optimal, Theorem 1 provides an interesting analogy with binary search trees. Given a set of distinct elements $\{a_i\}$ and probabilities $\{p_i\}$, an optimal binary search tree can be constructed. The problem of locating the elements in the tree (with weight corresponding to the probability of the element) is then analogous to the sorting problem. Bayer (2) has shown that the weighted average number of comparisons to find an element in an optimal binary search tree lies between

$$\Sigma\, p_i \log \frac{1}{p_i} + O(n) \text{ and}$$

$$\Sigma\, p_i \log \frac{1}{p_i} - \log\left(\Sigma\, p_i \log \frac{1}{p_i}\right) + O(1).$$

Rewriting p_i as m_i/n and multiplying through by n to indicate locating all n elements in the sorting problem, we have precisely the sorting bounds outlined above.

It may well be that these bounds are as tight as possible given the set (rather than sequence) $\{m_i\}$. Our reasoning is based on the fact that Bayer's bounds are tight in the sense that there exist probability distributions which achieve his upper bound and others achieving the lower bound. Allen (1) has observed that a gap of $\Omega(\log \Sigma\, p_i \log \frac{1}{p_i})$ can be due to the order of the p_i.

The problem of determining the mode, or most frequently occurring element in a list is also discussed in (6). A lower bound of $n \log n/m - O(n-k) \log \log k - O(n)$ is shown (where m is the number of times the mode occurs and k is the number of distinct elements). Since this bound hinges on the lower bound for the sorting problem, we have a slight improvement.

Corollary 1: $n \log (n/m) - n \log(\log n - (m_i/n) \log m_i) - O(n)$ comparisons are necessary to determine the mode.

An upper bound of $3n \log(n/m)$ comparisons is given (and a 1.54 $n \log(n/m)$ technique is suggested) in (6) for the mode problem. We now show that it can be reduced to $n \log(n/m) + o(n \log n/m)$, that is, "lower bound plus lower order term." The algorithm is fairly complicated and so we will first sketch the basic idea, then add the tricks which reduce the

run time.

First note that if we had a "median oracle," we could, in n-1 comparisons, divide a multiset, S, into three segments, those less (L), equal (E) and greater (G) than the median. Furthermore neither L nor G could contain more than 1/2 the elements of S. Call L and G hetergeneeous, since we do not know that all elements in either are the same, and E homogeneous, since we have ascertained this fact. By repeatedly applying this splitting to the largest remaining heterogeneous one, the mode is found in at most $n \log(n/m)$ comparisons. Note that the fact that m is not known at the beginning of the computation does not cause any problems.

From this scheme, the $3n \log(n/m)$ algorithm should be clear. Of more practical interest is the fact that by sampling, say $O(s^{\frac{1}{4}})$, elements to estimate the median of a segment containing s elements (in the style of (8)) we can find the mode in $n \log n/m + o(n \log(n/m))$ not only on the average, but if we assume random sampling, over all inputs with probability 1.

While this simple algorithm is the way in which we could be inclined to actually compute the mode, it is nonetheless interesting to develop a "lower bound plus lower order" worst case method.

Suppose now that we know the value of m, we will return to the more realistic case in which it is not known.

Algorithm M for finding the mode of a set:

Step 1. Break the list into sublists (columns) of ℓ = (log(n/m) elements each. Sort each column.

Step 2. While the largest heterogeneous segment H is larger than the largest homogeneous one,

do begin

a) find two medians of columns of H such that at most (1/2 + o(1)) of the elements of H exceed both, precede both or lie between both;

b) use these medians to split H into 3 heterogeneous and o homogeneous segments and merge the short columns of the segments resulting from this split so that all columns are of length between ℓ and 2ℓ.

end

It is clear from the construction of this algorithm that it halts and returns the value of the mode when the while loop of step 2b is no longer executed. It remains to analyze its complexity. We observe first that since as many as 1/2 of the original elements may lie between two consecutive column medians, no better split is possible. To find an appropriate pair of medians, we do what may be called a truncated binary search. To begin, we find the median of the medians and determine its relation to each element of each column by performing a set of binary searches on the columns. The cost of this operation on a set of s elements broken into columns of size between ℓ and 2ℓ is $O(s/\ell)$ operations to find the median median and $O((s \log \ell)/\ell)$ operations to do the binary searches for a total of $O((s \log \ell)/\ell)$ operations. We continue the binary search by finding the median of the medians of the larger segment and continue this operation for a total of $\frac{\ell}{(\log \ell)^2}$ steps. We stop at this point and observe that going any further would be too costly but at this stage of the computation we are guaranteed that at most (1/2 + o(1))s elements lie between our two

remaining medians. The total cost of our search has been $\frac{\ell}{(\log \ell)^2}$ steps at a cost of $O((s \log \ell)/\ell)$ operations per step for a total cost of $O(s/\log \ell) = o(s)$ operations.

The run time of the algorithm is dominated by the operations of Step 2b in which the smaller columns are reconstructed into larger columns for the recursion. If the above partitioning were to divide the original segment into 2 segments of roughly half its size, with the third (homogeneous) segment being small, then s comparisons would suffice for the reconstruction, even if all ns columns were cut in half. However, it is possible for the heterogeneous segments to be such that reconstruction in s comparisons is not possible.

We may use the Hwang-Lin merge algorithm (5) and revise our bookkeeping to look at the total merging cost up to the point where the maximum segment size is r. We now show this cost to be $n \log \frac{n}{r} + o(n \log \frac{n}{r})$ which for $r = m$ yields the desired result.

Since the Hwang-Lin algorithm is within a lower order term of the information theoretic lower bound, we are guaranteed that the revision can be done in this bound. Now, we consider the cost of all revisions done throughout the entire execution of the algorithm and observe that they number the information theoretic bound for constructing an output in form we obtain plus other lower order terms resulting from the gap between the performance of the Hwang-Lin algorithm and the information theoretic lower bound and from the information lost when columns are divided by the operations of Step 2a. For each column, this last quantity

is bounded by its number of elements. Hence the total amount of information lost in terms of comparison is no more than the total number of elements considered in all stages of the algorithm and this can be shown to be $o(n \log \frac{n}{m})$.

There now remains only the problem of not knowing m, the cardinality of the mode. We will "assume" m is $\frac{n}{2^{2^i} - 1}$ starting with i = 1 (though another starting choice for i would work as well). This implies that the minimum column length will be log(n/(assumed m)) which is roughly $2^i - 1$. The value of i is incremented (column length doubled) when the maximum heterogeneous segment size has been reduced by a factor of 2^{2^i}, which in the ideal case would correspond to 2^i halvings in half. We see then that the work done when i is at its ultimate (penultimate) value will dominate that of the rest of the computation. That means that most of the computation is performed when the "assumed" value of m is within a factor of 2 of the actual value. Hence we have

Theorem 2: The mode of a muliset containing n elements including m copies of the mode can be found in

$$n \log n/m + o(n \log n/m) + O(n)$$

comparisons.

3. Multiple Selection

Consider the problem of performing multiple selections from a set. In particular, we will consider the problem of selecting the $i_1, i_2, \ldots, i_k$ largest elements in a set of n elements. Denoting the complexity of this problem by $ms(n; i_1, i_2, \ldots, i_k)$. By appealing to results on the complexity of sorting we may establish the lower bound

Theorem 3: To determine the $i_1, i_2, \ldots, i_k$ largest elements in a set of n elements requires

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

comparisons where $i_0 = 0$ and $i_{k+1} = n + 1$.

Proof: We observe that having determined, by performing comparisons alone, the i_1st, i_2nd, ... i_kth elements in the set, we must know which elements lie between elements i_j and i_{j+1}. If we then sort each of these sets, we will have found the sorted order of the entire list. Since $k \log k - O(k)$ comparisons are both necessary and sufficient for sorting a set of k 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

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

comparisons.

Next we turn to the problem of finding an algorithm which approaches this bound.

Theorem 4: The $i_1, i_2, \ldots, i_k$ largest elements of a set may be determined in

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

Proof: We propose the following algorithm for the process:

Algorithm ms(n:i,i,...,i) 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 solve $ms(i_j - 1; 1_i, \ldots, i_{j-1})$ on the i - 1 largest elements of the set and $ms(n-i_j - 1; i_{j+1} - i, \ldots, i_k - i_j)$ on the $n-i_j-1$ smallest elements.

It is clear that the run time of this algorithm satisfies the recurrence

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

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

where $M(n)$ is the number of comparisons necessary to do a selection from a set of n elements. At present the best known value of $M(n)$ is $3n + o(n)$ (10). We prove the theorem by induction on k, the number of selections to be done. For $k = 1$, the algorithm requires $M(n)$ operations and this is certainly

$$\Omega(n \log n - i_1 \log i_1 - (n - i_1) \log (n - i_1)).$$

Now assume that the result is true for all $K < k$. Then we may apply the above recurrence to state that there is a constant t such that

$$ms(n; i_1, i_2, \ldots, i_k) \le t(\{(i_j - 1) \log (i_j - 1) +$$

$$(n - i_j - 1) \log (n - i_j - 1)\} + \{\sum_{m=0}^{k} (i_m - 1) \log (i_m - 1) +$$

$$\sum_{m=j+1}^{k} (n + 1 - i_m) \log (n + 1 - i_n) + M(n)\}) \le$$

$$t(n \log n + \sum_{m=0}^{k} (n + 1 - i_m) \log (n + 1 - i_m))$$ and hence

the theorem holds.

3. Selection in Minimum Space

In this section we are concerned with the problem of finding the kth largest element of a set in minimal space as well as time. Our model of computation will be essentially that outlined in section 1. The basic operation is the element comparison and we do not charge for "overhead." The input, however, is presented on a one-way read only tape. Inputs are read into any of c cells of memory and comparisons may be made between the contents of any pair of cells.

Under this model, it is clear that $k + 1$ storage locations are necessary to find the kth largest $(k \leq n/2)$ element in a set (6). We wish to consider here whether this number of storage locations is sufficient for doing this computation in linear time. None of the standard linear time algorithms can be implemented to run in this space bound and linear time. However, it is possible to use a heap of $k + 1$ elements as a priority queue, throwing away the smallest remaining element as each new term is read (after a steady state of $k + 1$ inputs have been reached). This algorithm leads to an algorithm to solve the problem in $n \log k + O(n)$ comparisons.

We have previously (4) given a method equivalent to the following linear time algorithm for finding the median in minimal space.

Read $\lfloor n/2 \rfloor$ elements into the unordered set U.

For i = 0 until $\lfloor \log n \rfloor - 3$ do

Find by a standard linear selection algorithm, the $2^{\lfloor \log n \rfloor - 3 - i}$ largest and smallest elements remaining in U. Logically remove these elements from U and place them respectively in the sets $L_{\lfloor \log n \rfloor -3-i}$ and $S_{\lfloor \log n \rfloor -3-i}$. L_j and S_j will, then, contain respectively the 2^{j-1} elements which are not in the 2^{j-1} largest or smallest but are among the 2^j largest and smallest of the $\lfloor n/2 \rfloor$. Clearly each iteration of this step takes time linear in the number of elements actually considered. Since this number is effectively halved on each iteration the total run time is also linear.

An unordered "residue" set, R, will be maintained in the remainder of the algorithm. Initially, R contains $0 = 2^0 - 1$ elements.

For i = 0 until $\lfloor \log(\lceil n/2 \rceil + 1) \rfloor - 1$ do

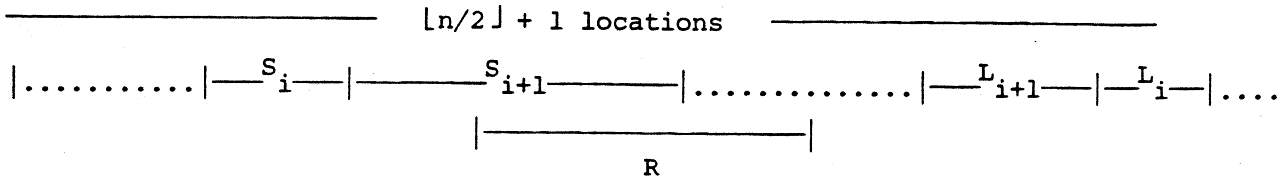
Read 2^i elements adding them to R which now contains 2^{i+1} elements. By a linear selection algorithm, discover and discard (as being too large or small to be the median) the largest and smallest 2^i elements of the $2^{i+2} - 1$ elements in L_i, S_i, and R. Logically transfer the remaining $2^{i+1} - 1$ to R. Note the run time of iteration, and hence the entire step is proportional to the number of elements discarded.

end

Read in the rest of the elements and find the median of those remaining by a linear algorithm.

The corrections of the above method are based on the observation that elements are discarded when (and only after) $\lfloor n/2 \rfloor$ elements are known to be larger and $\lfloor n/2 \rfloor$, smaller. The linear run time is a consequence of the analysis contained within the algorithm, as is the space bound. The

diagram below shows a typical configuration during the second iteration step:



R is logically separate from the S-L storage, but can physically occupy the space vacated by discarded L_j and S_j.

This is somewhat of an anomalous situation in light of the following:

Theorem 5: For every $0<p<1/2$ 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: The proof is by the construction of an adversary which forces any algorithm to sort $\min(p, 1-2p)n$ elements smaller than it. Now suppose that the input is arranged in such a manner that the first 2pn inputs are the 2pn smallest elements of the original set. Then in the first pn steps of elimination, we will either have to eliminate the smallest of pn + 1 elements resulting in the sorting of the smallest pn elements of the set, or we will have to eliminate the largest of pn + 1 elements resulting in the sorting of a set of $(1 - 2p)n$ elements.
Surprisingly, only a small increase in storage is necessary in order to make this problem feasible as the following theorem shows.

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

<u>Proof:</u> 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 space tradeoff whereby we may measure the effects of added storage according to the following.

<u>Theorem 7:</u> If $\varepsilon(n) = o(n)$, then for every $0<p<1/2$ there exists a constant $c_p > 0$ such that $c_p n \log \frac{n}{\varepsilon(n)} - O(n)$ comparisons are necessary to determine the pn^{th} largest element of a set of n elements in storage $pn + \varepsilon(n)$.

<u>Proof:</u> The argument is similar to that used in proving Theorem 5. We observe that we must output a structure consisting of $\min(p, 1 - 2p)n$ elements divided into sets of elements. Within this structure, we know the relative sizes of differing sets of elements. A total of $\min(\frac{(pn)!}{(\varepsilon!)^{pn/}}, \frac{((1 - 2p)n)!}{\varepsilon!^{(1 - 2p)n/\varepsilon}})$ such structures exist from which an information theoretic argument generates the given bound. From this result we observe that if $\varepsilon(n) = n/k(n)$ extra space is available, then $\Omega(n \log k(n))$ extra time is required, providing an interesting time space tradeoff.

5. Conclusion

A number of selection problems have been considered and solved. In particular we have sharpened bounds on the problem of finding the mode of a multiset and related an algorithm for multiset selection, as well as demonstrating an interesting instance of time space tradeoffs in the consideration of space limited selection algorithms. The general form of our algorithms have been to deal with worst case behaviour. While most of the algorithms presented here are not recommended for practical implementations, it should be clear that drastic simplifications can be made to these algorithms such that the expected number of comparisons is close to the lower bound.

6. References

1. B. Allen, "On Binary Search Trees," Research Report CS-77-27, Department of Computer Science, University of Waterloo.

2. Bayer, P. J., "Improved Bounds of the Costs of Optimal and Balanced Binary Search Trees," Project MAC Technical Memorandum 69, M. I. T., November 1975.

3. Blum, M., Floyd, R., Pratt, V., Rivest, R., and Tarjan, R., "Time Bounds for Selection," JCSS 7(1973), pp. 448-461.

4. Dobkin, D., and Munro, I., "A Minimal Space Selection Algorithm that Runs in Linear Time," Proceedings of the Johns Hopkins CISS Conference, April 1977. Also appears as Yale Department of Computer Science Technical Report #106.

5. Hwang, F. K., and Lin S., "A Simple Algorithm for Merging Two Disjoint Linearly Ordered Sets," SICOMP 1,1 (March 1972), pp. 31-39.

6. Munro, I., and Spira, P., "Sorting and Searching in Multisets," SICOMP, 5,1 (March 1976), pp. 1-9.

7. Pohl, I., "A Minimum Storage Algorithm for Computing the Median," IBM Technical Report RC2701, November 1969.

8. Pratt, V., and Yao, F., "On Lower Bounds for Computing the ith Largest Element," Proceedings of the 14th Annual IEEE Symposium on SWAT, October 1973, pp. 70-81.

9. Rivest, R., and Floyd, R., "Bounds of the Expected Time for Median Computations," in Combinatorial Algorithms, ed. R. Rustin, Courant C. S. Symposium 9, Algorithmics Press 1973, see also Rivest and Floyd "Algorithm 488 (Select) CACM 18(1975), p. 173.

10. Schonage, A., Paterson, M., and Pippenger, N., "Finding the Median," JCSS 13 (1976), pp. 184-199.