DETERMINING THE MODE[1]

by

David Dobkin[2] and J. Ian Munro[3]

Computer Science Dept.
University of Waterloo
Waterloo, Ontario
Canada

Research Report CS-79-27

July 1979

Abstract

The complexity of computing modes and sorting multisets is considered. Previous lower bounds are improved and an algorithm is given to determine the mode of a multiset in a number of comparisons differing from the lower bound by only a "lower order term".

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

[2]Computer Science Dept.
University of Arizona
Tucson, Arizona, USA

[3]Computer Science Dept.
University of Waterloo
Waterloo, Ontario N2L 3G1
CANADA

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 direct our attention to a closely related problem, that of finding the mode, or most frequently occurring element in a list. This problem has been previously investigated in [7]. Letting n denote the number of elements in the list and m the frequency of the mode, upper and lower bounds of roughly 3n log (n/m) and n log (n/m) comparisons (in the worst case) have been shown. In this paper we make a slight improvement in the lower bound on the number of comparisons required. The main contribution is, however, an algorithm to find the mode using n log (n/m) + o(n log (n/m)) + O(n) comparisons in the worst case. The gap between upper and lower bounds on this problem is thus reduced to "a lower order term plus O(n)". Our main algorithm, as presented, is rather complicated and not what one would be inclined to use in practice. However, as is often the case, the general approach can easily be followed to produce a practical method which runs in a near minimal number of comparisons on the average, although not necessarily

[†] all logarithms in this paper are to base 2.

in the worst case. This simplified version is used here to introduce the more complicated scheme.

The basic model of computation is a random access machine whose memory can store one data element in each location. The basic operation employed is the 3-way (<,=,>) branch comparison. Our measure of complexity is the maximum number of comparisons used by an algorithm. We assume indexing is performed at no cost.

2. SORTING MULTISETS

We consider first the problem of sorting a list containing repeated elements. Munro and Spira [7] have shown that, given a list of n elements, k of which are distinct, where m_i are of the i^{th} type (hence $\sum_{i=1}^{k} m_i = n$),

$$n \log n - \sum_{i=1}^{k} m_i \log m_i - (n-k) \log \log k - O(n)$$

(3 way branch) comparisons are necessary (on the average) to sort the list. Furthermore, by applying a modification of treesort, they demonstrate that $n \log n - \sum m_i \log m_i + O(n)$ comparisons suffice. The general aim of this paper is to demonstrate upper and lower bounds which differ by "a lower order term plus O(n)". For this reason we find it appropriate to rephrase and slightly extend the lower bound quoted above.

Theorem 1: Let S be a multiset with multiplicities $m_1,\ldots,m_k$ (where $n = \sum m_i$). Then at least

$$n \log n - \sum_{i=1}^{k} m_i \log m_i - n \log (\log n - (\sum_{i=1}^{k} m_i \log m_i)/n) - O(n)$$

3-way branch comparisons are required, on average, to sort S even if the m_i are given.

Outline of Proof: Following [7] we note that the sequence of results of comparisons performed by any algorithm can be viewed as a word

over the ternary alphabet {>,= <}. Furthermore, any such word based on the "optimal" algorithm will contain exactly n-k occurrences of "=" (since any additional "=" 's are redundant). There are $\binom{n}{m_1 \ldots m_k}$ ways to place the elements of S into their equivalence classes. Therefore, letting s = n-k and T be the complexity of a sorting algorithm,

$$\binom{T}{s} 2^{T-s} \geq \binom{n}{m_1 \ldots m_k}$$

First note that if T = O(n) then $n \log n - \sum m_i \log m_i = O(n)$, **and so the theorem holds in this case. Now consider the case in which T = Ω(n).**

In this case $\binom{T}{s}$ is $T^s/s!$ (to terms of O(n)), and so taking logarithms we get

$$\log \left[\binom{T}{s} 2^{T-s} \right] = s \log T - s \log s + s + T - s \pm O(n)$$
$$\geq \log \binom{n}{m_1 \ldots m_k}$$

and so

$$T \geq \log \binom{n}{m_1 \ldots m_k} - s \log (T/s) - O(n)$$

Noting that under the constraints $s \leq n$, $T = \Omega(n)$, $s \log (T/s)$ is maximized when s = n we have

$$T \geq \log \binom{n}{m_1 \ldots m_k} - n \log (T/n) - O(n)$$

which implies the theorem. □

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 k distinct elements $\{a_i\}$ and their probabilities of being accessed $\{p_i\}$, an optimal binary search tree can be constructed. The problem of locating each element in the tree (with weight corresponding to the probability of the element) is, then, analogous to the sorting problem. We emphasize that although the analogy is rather close, there are important differences between the two problems. For example, more "information" is available in the case of searching the optimal tree, while the "searches" are not required to be independent in the case of sorting. These differences are sufficient to prevent the immediate translation of bounds on one problem to the other. The known bounds are, however, remarkably similar. Bayer [2] has shown that the weighted average number of comparisons to find an element in an optimal binary search tree lies between

$$\sum p_i \log\frac{1}{p_i} + O(1) \text{ and } \sum p_i \log\frac{1}{p_i} - \log(\sum p_i \log\frac{1}{p_i}) + O(1).$$

Rewriting p_i as m_i/n and multiplying through by n to indicate locating all n elements in the sorting problem, we have 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 this gap can be largely due to the order of the p_i. He shows a gap of $\log \left(\sum_{i=1}^{n} p_i(\log(1/p_i)) \right)$ (which in his example is roughly log n) between the average cost of the two orderings of the same set of probabilities. Indeed he constructs an example in which $\sum p_i \log (1/p_i)$ is almost its maximum possible value, log n, yet under 2 different orderings of the probabilities, the costs of the optimal binary search trees differ by almost $\log (\sum p_i \log (1/p_i))$, the maximum possible difference.

Our main problem, that of determining the mode, or most frequently occurring element in a multiset, is closely related to sorting a multiset. Based on the lower bound of

$$n \log n - \sum m_i \log m_i - (n-k) \log \log k - O(n)$$

for sorting, a lower bound of $n \log (n/m) - (n-k) \log \log k - O(n)$ is demonstrated for determining the mode [7]. The method of proof is to show that if we have found the mode (and m, its frequency of occurrence) then given any set of m+1 elements we know at least one (specific) element which is greater than another (specific) element. This allows us to "finish" sorting the list in $n \log m - \sum_{i=1}^{k} m_i \log m_i + O(n)$ comparisons. The lower bound on the mode problem then follows by subtraction from lower bound on sorting. Theorem 1, then, leads immediately to a slight improvement on the lower bound for finding the mode.

Corollary 2 - $n \log(n/m) - n \log(\log n - \sum_i ((m_i/n) \log m_i) - O(n)$

comparisons are necessary to determine the mode.

3. FINDING THE MODE

In this section we derive the main result of this paper, namely an algorithm for finding the mode in a number of comparisons differing from the lower bound by only a lower order term. An algorithm given in [7] provides an upper bound of $3n \log (n/m)$ where m denotes the frequency of the mode. The basic approach of that technique and those of this section is to obtain a good estimate of the median of the multiset, and so partition it into those elements above, below and equal to this estimate. If there are more elements equal to the estimate than there are elements above that value and also more than there are below, then of course the mode has been found. Otherwise the process continues by repeatedly splitting the largest segment which may contain elements of unequal value. Using the $3n + o(n)$ median algorithm of Schönhage, Paterson and Pippenger [10], the mode can thus be found in about $3n \log (n/m)$ comparisons. Our main goal is, of course, to reduce the constant to 1.

To formalize this procedure, define a multiset to be homogeneous if all elements are known to be equal and heterogeneous otherwise. By a segment of the multiset, S, we will mean a submultiset which contains all elements of S in a given range. Observe that a segment of a segment of S is itself a segment of S.

A natural, and indeed practical, approach to determining the mode of S is formalized as follows.

ALGORITHM EASYMODE

```
Begin
    Initially the only segment in the system is S which
        is, of course, heterogenous;
    While the largest heterogeneous segment, H (which
        contains h elements), is larger than any homo-
        geneous segment    do
            begin
                In o(h) comparisons find a reasonable estimate,
                    mid, of the median of H;
                By comparing each element with mid, split H into
                    2 heterogeneous segments (those less than and
                    those greater than mid), and 1 homogeneous
                    segment (those equal to mid)
            end;
    The value in the largest homogeneous segment is the
        mode
end.
```

It is not hard to see that this algorithm will correctly determine the mode. Furthermore, ignoring the cost of determining estimates of medians, fewer than n comparisons are used to split all 2^i heterogeneous segments of size about $n/2^i$ into 2^{i+1} of size $n/2^{i+1}$. Viewing the bookkeeping in this way, we see that no more than about n log (n/m) comparisons are used, provided we can get a good median estimate "free". If our interest is in an algorithm which runs quickly on the average, this is rather easy. In a manner analogous

to the Rivest-Floyd [9] median algorithm, we take a random sample of $o(h/\log h)$ (say $\sqrt{h}$ to be concrete) elements and determine the median by some straightforward method such as sorting. It is not difficult to show that the probability of this element being more than $0(\sqrt{h})$ elements away from the true median of H, goes to 0 as h becomes large. Such an approximation is then, satisfactory for a simple technique which, on the average and also with probability tending to 1, finds the mode in $n \log(n/m) + 0(\sqrt{n} \log(n/m))$ comparisons. Indeed, this is the method we would recommend in practice.

The main goal in this study is, however, to find an algorithm guaranteed to determine the mode in the desired number of comparisons. For this reason we must be able to guaranteee that our estimate of the median of a segment is very near the true median. With this goal in mind, a technique analogous to the Blum et al [3] median algorithm is employed. The basic method is given below, the details of the splitting and merging follow. In the preliminary exposition we will assume that m, the frequency of the mode, is known and large. We then return to the (more realistic) case in which m is unknown.

ALGORITHM mMODE

Begin
Split S into sublists of length $\ell = \lceil \log n/m \rceil$ elements and sort each sublist (a sorted sublist will be called a column)

While the largest heterogeneous segment, H, (containing h elements) is larger than any homogeneous segment do

begin
Using o(h) comparisons, find an element, mid, such that at most $\frac{1}{2} + o(1)$ of the elements of H exceed mid and at most $\frac{1}{2} + o(1)$ are less than mid;

Split H into 2 heterogeneous and 1 homogeneous segments H_1, H_2 and E whose elements are, respectively, less than, greater than and equal to mid;

Repeatedly merge pairs of columns of H_1 until the average column length is restored to about ℓ. Do the same with H_2.
end;

The value contained in the largest homogeneous segment is the mode.
end.

3.1 The Splitting

We present a fairly informal description of the splitting procedure and an analysis of its cost. A more formal version of the process follows.

A reasonable, but naive, approach to splitting would be to take the median of the column medians, and then perform a binary search in each column to partition H into those elements above, below and equal to this median of medians. Unfortunately, as in the Blum et al [3] median algorithm, it is possible that all but ¼ of the elements exceed this median of medians even if we are able to guarantee that all columns are of length ℓ on each iteration. In fact we will not even be able to make this guarantee as the column lengths will vary later in the computation. We note furthermore that about ½ of all the elements could lie between a pair of consecutively ranked column medians. It was this difficulty which led, in a preliminary version of this paper [4], to the notion of splitting the multiset into three heterogeneous segments. While this approach does lead to the desired solution to the problem, we find it extremely difficult to present a reasonably comprehensible proof and so follow a different approach.

The first time a splitting is performed, our approach is to begin by simply finding the median of the column medians, and determining its rank with respect to all the elements. This splitting leaves one homogeneous and two heterogeneous segments. If all columns are of length ℓ, as they are on the first splitting, we can claim that neither heterogeneous segment contains more than 3/4 of the elements. However we will be attempting to get a better estimate of the median and this will involve working with subcolumns of varying sizes. Indeed after the first splitting has been performed we will no longer be able to guarantee that all columns are of the

same length. For this reason we introduce the notion of finding a weighted median.

Suppose we have a number of sorted columns of various sizes. The median of the column medians is not guaranteed to rank in the middle $\frac{1}{2}$ of the multiset. However, if each column median "represents" a weight equal to the number of elements in its column, then the weighted median of the medians is guaranteed to exceed (or equal) at least $\frac{1}{4}$ of all the elements. The weighted median of k elements is easily found in O(k) comparisons, independent of the weights. The technique is simply to find the actual median. The weighted median then lies in one of the halves of the k elements. We again find the median of that half, and so on recursively. The cost of determining the weighted median is, then, about twice that of a simple median computation.

The basic approach, then, is to find the weighted median of the column medians of H, and to partition all elements of H about this value. Using a binary search on each column to achieve the latter task we note that about $(h/\ell)\log \ell$ comparisons are used if the average column length is ℓ. The technique is reapplied to the subsegment of H (and so h/ℓ subcolumns of various sizes) which contains the median of H. This process is iterated a number of times. After i iterations the subsegment containing the median of H contains at most $(3/4)^i$ of the elements. To apply this technique to actually find the median of H would be too costly for our purposes. However, as long as we choose the number of iterations to be $\omega(1)$, we know H can be split about this value into one homogeneous segment and two heterogeneous segments, neither of which contain more than $\frac{1}{2} + o(1)$ of the elements. This is satisfactory for our purposes. As long as $o(\ell/\log \ell)$ iterations are performed the entire splitting process will cost o(h)

comparisons. The (rather arbitrary) choice of $\sqrt{\ell/\log \ell}$ iteration suffices, and the maximum size of a resulting heterogeneous segment will be $1/2 + O((3/4)^{\sqrt{\ell/\log \ell}}) = 1/2 + o(1)$.

More formally, we may write the splitting procedure as follows:

Procedure SPLIT

Initially, let M denote the entire segment H;

For i = 1 until $\sqrt{\ell/\log \ell}$ do

    begin

        Determine mid, the weighted median of the subcolumn medians of elements of H;

        Using binary search, partition M into three subsegments, those elements above, below and equal to mid;

        Let M denote the subsegment containing the median of H
    end;

Partition H into three subsegments H_1, H_2 and E which consist of the elements respectively, above, below and equal to mid.

3.2 The Merging

The main goal of the merging process is to reconstitute both H_1 and H_2 into a form consisting of columns whose average length is about ℓ. Any pairwise merging of the subcolumns of H which constitute H_1 and of those which constitute H_2 can accomplish this goal in h or fewer comparisons. It is, perhaps, more satisfying to be able to keep not only the average column length at the desired level, but to keep all (except perhaps 1) of the columns in the range of $\ell/2$ to 2ℓ elements. It is not difficult to check that this can be accomplished if the smallest remaining pair of subcolumns of H_1 are merged repeatedly until either no columns of length $\leq \ell$ remain or no more columns can be merged without an accumulated cost of

more than one comparison for each element of H_1. The process is then applied to H_2.

Details concerning the effects of the splitting not being "perfect" and that E, the homogeneous segment , may be of a nontrivial size, are minor and tedious, and so omitted.

3.3 Estimating m Quickly

We have now shown that the mode can be found quickly if its frequency m, is known. An intelligent scheme must be found here to estimate the cardinality of the mode, since if our estimate is too high too much work will be done in repeatedly finding medians, and if our estimate is too low, extra work done in sorting lists will dominate the computation. Of the two possibilities, it is easier to resolve being too high, so we begin by over estimating m. Our mode estimates are taken as $n/(2^{2^i}-1)$ with i beginning at 2 and we begin with columns of size 5 (rather than 4). When we have determined that our estimate is too high (i.e. when the largest heterogeneous segments size has been reduced by a factor of about 2^{2^i}), we increment i and repeat step 1 of mMODE for all remaining heterogeneous segments with the column size doubled. However, since all sorting information is available, columns are now formed by merging. This extra work requires O(n) operations, and must be done loglog n/m times. Hence, we can use algorithm mMODE with an additional $O(n \text{ loglog } n/m) = o(n \log n/m) + O(n)$ operations. The cost of doing the mMODE algorithm does not change, so that our total cost, even though m is unknown is still $n \log n/m + o(n \log n/m) + O(n)$. We summarize these ideas in the following

ALGORITHM MODE

Set $i \leftarrow 2$; the estimate of m, $mguess \leftarrow \lfloor n/(2^{2^i}-1) \rfloor = \lfloor n/15 \rfloor$, and so $\ell \leftarrow 5$;

Divide S into sublists of size 5 and sort each sublist;

Until the mode has been found do

Begin

Until we have determined that m < mguess do

Begin

Apply Step 2 of algorithm mMODE to all heterogeneous segments

End;

Set $i \leftarrow i+1$, $mguess \leftarrow \lfloor n/2^{2^i}-1 \rfloor$, $\ell \leftarrow 2\ell$

Merge columns pairwise in each heterogeneous segment so that the average column length is about ℓ,

end

Output the mode

Theorem 3: Algorithm MODE correctly computes the mode of a multiset of size n with mode cardinality m in n long $n/m + o(n \log n/m) + O(n)$ compares.

Proof: Correctness is obvious from the algorithm's construction. The time bound follows from the analysis of mMODE and the observation that the column merging, the only deviation here from mMODE requires $O(n \log \log(n/m))$ operations.

4. CONCLUSION

Improvements to a previous lower bound for sorting of multisets are presented. An interesting connection between this problem and a problem on binary search trees is given which suggests that further improvements may not be possible. This lower bound is used to generate a lower bound on the complexity of computing the mode of a multiset. Using a technique for quickly estimating the median of a multiset and various submultisets, an algorithm is developed for matching this lower bound up to lower order terms. While initial versions of the algorithm require advanced knowledge of the mode cardinality, we show that this assuption can be removed through appropriate estimation procedures.

5. REFERENCES

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

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

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

[4] Dobkin, D., and I. Munro, "Time and Space Bounds for Selection Problems", Proc. ICALP 5, July 1978, Springer-Verlag Lecture Notes in Computer Science 62, pp. 192-204.

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

[6] Knuth, D.E., Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

[7] Munro, I., and P. Spira, "Storing and Searching in Multisets", SICOMP 5, 1 (March 1976) pp. 1-9.

[8] Pratt, V., and F. Yao, "On Lower Bounds for Computing the i^{th} Largest Element", Proc. 14th IEEE Symp. on S.W.A.T., October 1973, pp. 70-81.

[9] Rivest, R. and R. Floyd, "Bounds on the Expected Time for Median Computations", in Combinatorial Algorithms ed. R. Rustin, Courant C.S. Symp. 9, Algorithmics Press, 1973, see also Rivest and Floyd "Algorithm 489"(Select) CACM 18 (1975), p. 173.

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