Programming Techniques
R. Morris Editor

A Sorting Problem and Its Complexity

Ira Pohl
University of California*

A technique for proving min-max norms of sorting algorithms is given. One new algorithm for finding the minimum and maximum elements of a set with fewest comparisons is proved optimal with this technique.

Key Words and Phrases: sorting, computational complexity, computational combinatorics
CR Categories: 5.29, 5.31

* Information and Computer Science Department, and Stevenson College, University of California, Santa Cruz, CA 95060. This paper is a revision of a privately distributed communication, March 1970.

A sorting problem is a special case of finding a set partition. The elements are identified by their rank in a linear order. For a given finite set

$$X = \{x_1, x_2, \ldots, x_n\}$$

There is some permutation π^*, such that

$$X\pi^*_{(1)} < X\pi^*_{(2)} < \cdots < X\pi^*_{(n)}.$$

(Note: we will ignore throughout the case of $X\pi^*_{(i)} = X\pi^*_{(i+1)}$).

Other partitions which do not completely sort the set are often of interest, such as finding the minimum element, the maximum element, the first K smallest elements, the median or, as will be discussed here, the minimum and maximum elements.

Algorithms which take unordered (or partially ordered) sets and produce the desired partition are fundamental to computing, making this area one of the most intensely studied both practically and theoretically. In the theoretical attacks, the principal unit of work has been the comparison. An algorithm is compared to some norm on the number of comparisons. The two norms of most interest are the maximum and the average. The maximum norm (M-norm) is the greatest number of comparisons the algorithm takes over all possible inputs ($n!$ orderings are possible). The average norm (E-norm) is the average number of comparisons the algorithm takes with respect to the uniform distribution on the possible orderings. The best algorithm for a given norm is the one that minimizes that norm.

i.e. proving lower bound

The goals of this paper are: (1) to demonstrate a technique for proving algorithms optimal in the maximum norm; and (2) using the technique developed, to exhibit a new algorithm P2 for finding both the minimum and maximum of a set, which is then proven optimal in the M-norm.

A well-known result [1] is that $n - 1$ comparisons is M-optimal for finding the minimum (or maximum) element of an unordered set. (It is also E-optimal.)

Algorithm P1

```
min := x1;
for i := 2 step 1 until n do
  min := if min < xi then min else xi ;
Comment  Algorithm P1 takes n − 1 comparisons.
```

THEOREM 1. *Algorithm P1 is optimal in the M-norm.*

PROOF. Let A^i be the set of elements that are still possibly the minimum after i comparisons.

Let B^i be the set of elements that are known *not* to be the minimum after i comparisons.

Initially $A^o = X$, $B^o = \emptyset$, and $(|A^o|, |B^o|) = (n, 0)$. There are three categories of comparisons possible:

aa: comparing two elements in A^i;
ab: comparing an element in A^i to one in B^i;
bb: comparing two elements in B^i.

When an aa-comparison is executed, the result has to be that one and only one element remains in A^{i+1}, with the larger element being placed in B^{i+1}. The effects of the three comparison types on the cardinality of the resulting sets is:

aa: $(n_1, n_2) \rightarrow (n_1 - 1, n_2 + 1)$,
bb: $(n_1, n_2) \rightarrow (n_1, n_2)$,
ab: $(n_1, n_2) \rightarrow (n_1, n_2) \mid (n_1 - 1, n_2 + 1)$.

The ab-comparison has two outcomes that are possible: either the element in A^i will be less than the element in B^i or vice versa.

In finding the minimum, we start with $(|A^o|, |B^o|) = (n, 0)$ and must end up with $(|A^{\text{final}}|, |B^{\text{final}}|) = (1, n - 1)$. The most progress any single step can make is $(n_1, n_2)(n_1 - 1, n_2 + 1)$, and $n - 1$ of these are required to find the minimum. Q.E.D.

Remark 1. An algorithm which only uses aa-comparisons is optimal. If ab-comparisons are used, it is possible to produce a set which would require more than $n - 1$ comparisons. Algorithm P1 is only one of a class of algorithms which can perform this computation optimally. Any algorithm that uses only aa-comparisons (the way the set is indexed is unimportant) has to be optimal.

Remark 2. The result is also derivable from the observation that all binary trees with n leaf nodes have $n - 1$ interior nodes.

The proof technique for optimality is to consider sorting as a state space rewriting problem where inductive arguments are used to exhibit min-max chains from the initial to the final state [2, 3, 4].

Algorithm P2:/ Minimum and Maximum

```
  k := 1;
for i := 1 step 2 until n − 1 do
  if xi < xi+1 then
    begin
      bk := xi ; ck := xi+1 ;
      k := k + 1
    end
  else
    begin
      bk := xi+1 ;  ck := xi ;
      k := k + 1
    end;
if odd (n) then
  if b1 < xn then ck := xn
  else bk := xn ;
comment  Use P1 to find minimum over b and maximum over c
separately;
```

Algorithm P2 executes $\lceil \frac{3}{2} n \rceil - 2$ comparisons. It can take one less in the case of n odd where $b_k := x_n$. However in the sense of the min-max norm it is optimal.

THEOREM 2. *Algorithm P2 is optimal in the M-norm.*

PROOF. Let A^i be the set of elements that are still possibly the minimum after i-comparisons, but not maximum.

Let B^i be the set of elements that are still possibly the maximum after i-comparisons, but not the minimum.

Let C^i be the set of elements that are neither the maximum or minimum after i-comparisons.

Let D^i be the set of elements that are possibly either the maximum or minimum element.

Initially $D^o = X$, $A^o = B^o = C^o = \emptyset$ and $(|A^o|, |B^o|, |C^o|, |D^o|) = (0, 0, 0, n)$. There are 10 categories of comparison possible:

aa, ab, ac, ad, bb, bc, bd, cc, cd, dd.

The alternatives as written in the notation of Theorem 1 are:

aa: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1 - 1, n_2, n_3 + 1, n_4)$
ab: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1, n_2, n_3, n_4) \mid (n_1 - 1, n_2 - 1, n_3 + 2, n_4)$
ac: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1, n_2, n_3, n_4) \mid (n_1 - 1, n_2, n_3 + 1, n_4)$
ad: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1, n_2 + 1, n_3, n_4 - 1)$
$\mid (n_1, n_2, n_3 + 1, n_4 - 1)$
bb: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1, n_2 - 1, n_3 + 1, n_4)$
bc: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1, n_2, n_3, n_4) \mid (n_1, n_2 - 1, n_3 + 1, n_4)$
bd: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1 + 1, n_2, n_3, n_4 - 1)$
$\mid (n_1, n_2, n_3 + 1, n_4 - 1)$
cc: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1, n_2, n_3, n_4)$
cd: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1 + 1, n_2, n_3, n_4 - 1)$
$\mid (n_1, n_2 + 1, n_3, n_4 - 1)$
dd: (n_1, n_2, n_3, n_4)
$\rightarrow (n_1 + 1, n_2 + 1, n_3, n_4 - 2)$

If aa, bb, and dd comparisons alone are used, then

$$|\,aa\,| + |\,bb\,| + |\,dd\,| = n - 2 + (n/2)$$

where n is even, and if n is odd, then using one ad (or bd) the number is at most

$$n - 2 + 1 + \frac{n-1}{2} = n - 2 + \frac{n+1}{2} = \left\lceil \frac{3}{2} n \right\rceil - 2.$$

This is the bound that Algorithm P2 attains.

However it is possible to improve on this if the second alternative of ab, ad, and bd comparisons occur. Other comparisons are not of interest since even a favorable outcome as in the second alternative of ac,

$$\text{ac: } (n_1, n_2, n_3, n_4) \rightarrow (n_1 - 1, n_2, n_3 + 1, n_4),$$

is no better than using an aa and does not improve the bound.

The bounds in these cases are of the min-max type. Thus, if a sequence could be produced for each algorithm that resulted in the first alternatives of ad, bd, and ab comparisons, these less favorable alternatives do not improve the bound and the theorem is proved.

Consider the case of ad (the symmetric argument holds for bd) to prove the first alternative is always realizable:

After i comparisons the set is partitioned into $(|A^i|, |B^i|, |C^i|, |D^i|) = (n_1, n_2, n_3, n_4)$. The relationship between an element of D^i and A^i is unknown, since elements of D^i have not yet been involved in some comparison. It therefore could always be the case that $d \in D^i$ is greater than any element in A^i, and the result would be the first alternative. (e.g. $A^i = (1, 2, 3, \ldots n_1)$, $D^i = (n_1 + 1, n_1 + 2, \ldots, n_1 + n_4)$).

Now consider the case of ab where we wish to have the first alternative occur:

Let

$$A^i = (1, 2, \ldots, n_1),$$
$$B^i = (n_1 + 1, \ldots, n_1 + n_2).$$

Then $a \in A^i$, $b \in B^i$ implies $a < b$ and the first alternative occurs. Q.E.D.

COROLLARY.[1] *For an arbitrary starting state* $(|A|, |B|, |C|, |D|)$, *the number of comparisons attainable in the min-max norm is*

$$|A| + |B| + \lceil \tfrac{3}{2} \cdot |D| \rceil - 2.$$

These results have a state space, production flavor which is often a fruitful viewpoint in computation. This style of case generation would be amenable to a computer generated analysis needed for more difficult min-max norm problems such as finding the median or computing matrix products.

Received February 1971

[1] S. Winograd pointed out that the proof of Theorem 2 contains this stronger result.

References

1. Hadian, A., and Sobel, M. Selecting the tth largest using binary errorless comparisons. TR No. 121, U. of Minnesota, Department of Statistics, May 1969.
2. Pohl, I. Heuristic search viewed as path finding in a graph. *J. Artif. Intell. 1* (1970), 193–204.
3. Pohl, I. A minimum storage algorithm for computing the median. IBM Tech. Rep. RC 2701, Nov. 1969.
4. Pohl, I. Syntactic models of cognitive behavior. ICS Dep. Rep., U. of California, Santa Cruz. Presented at the NATO Symposium on Human Thinking, St. Maximin, France, Aug. 1971.