BIT 31 (1991), 194–201.

THE RECURSIVE STRUCTURE OF SOME ORDERING PROBLEMS

M. D. ATKINSON

School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6

Abstract.

Some classical ordering problems (sorting, finding the maximum, finding the maximum and the minimum, finding the largest and the next largest, merging, and finding the median) are considered from a recursive viewpoint. If $X(n)$ denotes an instance of size n of any one of these problems then $X(n)$ can be solved by finding the solution to a number $\xi(n, k)$ of problems $X(k)$ for some fixed k; $\xi(n, k)$ is called the *relative complexity*. Upper and lower bounds on the relative complexity are found. For the problem of finding the maximum, finding the maximum and the minimum, and finding the largest and the next largest these bounds are optimal.

AMS Classification numbers: 06A10, 68C05

1. Introduction.

Divide and conquer is one of the most useful paradigms in algorithm design. One of its key aspects is the technique of solving a problem $X(n)$, of size n, by making use of solutions to problems $X(k)$ with $k < n$; thus divide and conquer algorithms are applicable for those problems whose solution can be expressed recursively. We shall investigate, for several classical ordering problems $X(n)$, the *number* of solutions to problems $X(r)$ with r bounded by some fixed k, needed to solve $X(n)$. In all the problems that we consider $X(2)$ will be the problem of comparing two numbers. Our results reduce to known ones on the number of comparisons in the case $k = 2$. In various degrees of detail we shall consider the following problems:

1. $A(n)$: sort n elements into ascending order,
2. $B(n)$: find the maximum in a set of n elements,
3. $C(n)$: find the maximum and the minimum in a set of n elements,
4. $D(n)$: find the largest and the next largest in a set of n elements,
5. $E(n)$: merge two sorted lists of length $n/2$,
6. $F(n)$: find the medians in a set of n elements.

All of these problems have traditionally been studied for their worst case comparison complexity and substantial progress has been made on all of them. In fact, apart

Received July 1990. Revised February 1991.

from problem $F(n)$, their comparison complexity has been found *almost exactly* in the sense that the worst case comparison complexity is known to be of the form $f(n) + o(f(n))$ for some known function $f(n)$; these results may be found in [5]. The complexity of $F(n)$ is known to be linear in n but the exact constant is as yet unknown; upper and lower bounds for $F(n)$ are proved in [6] and [4] respectively. Table 1 summarises what is known about the worst case comparison complexity and defines some notation which will be explained below (here, and subsequently, all logarithms are to base 2).

Table 1. *Relative complexity notation.*

Problem	Comparison complexity	Relative complexity
$A(n)$	$n \log n \pm O(n)$	$\alpha(n,k)$
$B(n)$	$n-1$	$\beta(n,k)$
$C(n)$	$\lceil 3n/2 \rceil$	$\gamma(n,k)$
$D(n)$	$\lceil n + \log n - 2 \rceil$	$\delta(n,k)$
$E(n)$	$n-1$	$\varepsilon(n,k)$
$F(n)$	between $2n$ and $3n$	$\phi(n,k)$

When $n = 2$ all of these problems are equivalent to the problem of comparing two numbers. Thus, if $X(n)$ is any one of $A(n), \ldots, E(n)$, the number of problems $X(2)$ which have to be solved in order to solve $X(n)$ is known almost exactly. We can paraphrase this statement as: the number of $X(2)$ operations required to simulate the $X(n)$ operation is known almost exactly. We now define the *relative complexity* $\xi(n,k)$ of a problem $X(n)$ to be the number of solutions to problems $X(r)$, $r \le k$, necessary to solve $X(n)$ in the worst case. In other words $\xi(n,k)$ is the number of operations of the form $X(r)$, $r \le k$, required to simulate the operation $X(n)$. Notice that $\xi(n,2)$ is the comparison complexity of $X(n)$. To clarify this concept we give an example.

EXAMPLE 1. $\alpha(4,3) = 3$.

Suppose we have to sort 4 numbers a, b, c, d using the operations $A(2)$ and $A(3)$ (clearly, $A(1)$has no effect). We can begin by applying $A(3)$ to 3 of the numbers, discovering say that $a < b < c$. In the second step we apply $A(3)$ to a, c, d. If either $d < a$ or $c < d$ then the sorting is complete. If $a < d < c$ then a third operation $A(2)$ applied to b, d completes the sorting. This shows that $\alpha(4,3) \le 3$ and it is easy to verify that no algorithm which uses only the operations $A(2)$ and $A(3)$ can sort 4 numbers in fewer than 3 operations in the worst case.

General lower bounds on the relative complexity can be obtained from two sources: bounds for $k = 2$ and information theory.

LEMMA 1. *For any problem $X(n)$ listed above, the relative complexity $\xi(n,k)$ satisfies $\xi(n,k) \ge \xi(n,2)/\xi(k,2)$.*

PROOF. Consider an algorithm for solving $X(n)$ which uses $\xi(n,k)$ solutions to problems $X(r)$, $r \leq k$. If we solve each subproblem $X(r)$ by using $\xi(r,2)$ comparison operations then, since $\xi(r,2) \leq \xi(k,2)$, we shall obtain an algorithm for $X(n)$ which uses at most $\xi(n,k)\xi(k,2)$ comparisons. Hence $\xi(n,k)\xi(k,2) \geq \xi(n,2)$. ■

Information theoretic lower bounds are derived using the principle that each operation can reduce the number of possibilities for the answer by at most a certain operation-dependent factor. The classical information theoretic lower bound on sorting generalises to the operation $A(k)$ as follows:

EXAMPLE 2. $\alpha(n,k) \geq \log n!/\log k!$.

We begin with $n!$ linear extensions. After t operations $A(r)$, $r \leq k$, the number of linear extensions is at least $n!/k!^t$ in the worst case. Termination cannot occur until $n!/k!^t \leq 1$, that is, $t \geq \log n!/\log k!$.

The bound given by Lemma 1 is $\alpha(n,k) \geq n \log n/\alpha(k,2)$ which, because of Stirling's approximation, appears to be similar. However, k is constant so the approximation is not a good one. In fact the information-theoretic bound is superior in this case. But sometimes Lemma 1 gives the better lower bound (for example, problems $B(n)$, $C(n)$, $D(n)$).

In section 2 we consider simple upper and lower bounds on the relative complexity of problems $A(n)$, $B(n)$, $D(n)$, $E(n)$, $F(n)$. In the cases of $B(n)$ and $D(n)$ these bounds are sufficiently close that $\beta(n,k)$ is determined and $\delta(n,k)$ is determined almost exactly. In section 3 we consider problem $C(n)$ in some detail and, by a linear programming argument, determine $\gamma(n,k)$ to within a small constant.

2. Upper bounds.

As we have seen $\alpha(n,k) \geq \log n!/\log k!$. Two upper bounds on $\alpha(n,k)$ were given in [3], namely, $4n \log n/k \log k$ and $n \log n/(k-1)$; the latter bound (which appears also in [2]) is superior up to $k = 12$. Thus, even the case $k = 3$ is not completely solved. For $k = 3$ the best algorithm known is based on a method for merging 4 lists of length $n/4$ in n applications of the $A(3)$ algorithm. This merging algorithm always knows the ranking of two of the 4 maximal elements in the lists. Therefore it can identify the overall maximum using one $A(3)$ operation which can then be placed on some output stream and removed from further consideration. The $A(3)$ operation discovers, if need be, the ranking between two out of the four remaining maximal elements of the lists; so the process can be repeated. The execution time $T(n)$ of a merge sort based on this procedure satisfies $T(n) \leq 4T(n/4) + n$ from which we find $T(n) \leq \frac{1}{2} n \log n$.

The problem $B(n)$ is much easier to analyse. Lemma 1 tells us that $\beta(n,k) \geq (n-1)/(k-1)$. On the other hand, there is a natural algorithm whose number of $B(k)$ operations meets this bound almost exactly: apply $B(k)$ to the first k elements to get a candidate for the maximum; repeatedly (until $k-1$ or fewer elements remain unexamined) apply $B(k)$ to a set consisting of the current candidate and $k-1$ new elements to update the candidate for the maximum; apply $B(r)$, $r \leq k$, to the current candidate and the remaining unexamined elements. This requires $1+\lceil (n-k)/(k-1) \rceil = \lceil (n-1)/(k-1) \rceil$ operations. Thus $\beta(n,k) = \lceil (n-1)/(k-1) \rceil$.

Problem $D(n)$ can also be solved almost exactly. The lower bound from Lemma 1 is $\lceil (n+\log n - 2)/(k+\log k - 2) \rceil$ but this bound is not tight. For observe that when an operation $D(r)$, $r \leq k$, is applied to some subset of the set of n elements the number of elements which remain candidates for the maximum element is reduced by at most $r - 1 \leq k-1$; hence at least $(n-1)/(k-1)$ such operations will be necessary. An algorithm whose number of $D(k)$ operations meets this bound almost exactly is easy to obtain and we shall describe it recursively. We divide the set of n elements into approximately n/k groups G_i of size k and find the largest and next largest elements μ_i, ν_i in each group using n/k operations $D(k)$. Next we find the largest and second largest elements in the n/k-element set $M = \{\mu_1, \mu_2, \ldots\}$ by this same algorithm used recursively. If μ_j is the maximal element of M then μ_j is certainly the overall maximal element. If μ_k is the second largest element of M, then the overall second largest element will be the larger of ν_j and μ_k so the operation $D(2)$ must be applied to this pair to complete the algorithm. The total number of operations $T(n)$ used by this procedure satisfies the recurrence $T(n) \leq n/k + T(n/k) + 1$ the solution to which is $T(n) = n/(k-1)$ almost exactly. Thus $\delta(n,k) = n/(k-1)$ almost exactly.

Rather less satisfactory results are available for problem $E(n)$. One of the merging procedures in [2] shows that $\varepsilon(n,k) \leq 2n/k$ while Lemma 1 gives the lower bound $\varepsilon(n,k) \geq n/(k-1)$. A better lower bound is obtained from the information theoretic approach. Each $E(k)$ operation has at most $\binom{k}{k/2}$ outcomes and so, in the worst case, each operation can reduce the number of possibilities for the result by at most a factor $\binom{k}{k/2}$. Hence

$$\frac{\log\binom{n}{n/2}}{\log\binom{k}{k/2}} \simeq \frac{n}{\log\binom{k}{k/2}}$$

is a lower bound for $\varepsilon(n,k)$.

Finally in this section we consider the problem $F(n)$ of finding the medians in a set of n elements. We take a naive approach merely considering how the operation $F(k)$ can be used effectively in one of the standard median finding algorithms. We shall assume that, when applied to a set of k elements, $F(k)$ returns not only their median but also the two subsets of elements which are above and below it respectively. Our treatment is based loosely on the SELECT algorithm in [1]. This means that we must have $k \geq 5$ and consider the more general problem $F(n,s)$ of selecting the sth largest element from a set of size n using $F(r)$, $r \leq k$, as the primitive operations (where, in order to avoid inconsequential details, we shall assume that k is odd and that the numbers are distinct).

1. Divide the n numbers into n/k groups of size k and, with n/k applications of $F(k)$, find the median of each group,
2. By solving a problem of the type $F(n/k, n/2k)$ find the median x of this set of medians,
3. Rank the elements into two subsets $S_1 = \{y \mid y < x\}$ and $S_2 = \{y \mid y > x\}$ as follows: if there remain at least $k - 1$ elements as yet unranked apply $F(k)$ to this set augmented by x. It is easy to see that at least $(k + 1)/2$ elements get ranked with respect to x; so this step requires $2n/(k + 1)$ such operations. Because of the choice of x in the first two steps it is easy to see that $|S_i| \leq (3k - 1)n/4k$.
4. From the last step we shall know the exact size of S_1, S_2 and in which of them, and at which rank, the required element is to be found. Hence we can apply this algorithm recursively to the appropriate one of S_1 and S_2.

It follows from this description that the number $T(n)$ of operations $F(k)$ required to solve the problem $F(n, s)$ (and hence the median problem $F(n)$) satisfies the inequality

$$T(n) \leq \frac{n}{k} + T\left(\frac{n}{k}\right) + \frac{2n}{k+1} + T\left(\frac{(3k-1)n}{4k}\right) \text{ for all } n \geq n_0$$

where n_0 is some constant depending on k. Putting $\alpha = \dfrac{4(3k+1)}{(k-3)(k+1)}$ we may write this recurrence inequality as

$$T(n) - \alpha n \leq T\left(\frac{n}{k}\right) - \alpha\frac{n}{k} + T\left(\frac{3k-1)n}{4k}\right) - \alpha\frac{(3k-1)n}{4k} \text{ for all } n \geq n_0$$

from which it follows that $T(n) - \alpha n = o(n)$, that is, $T(n) \leq \dfrac{4(3k+1)}{(k-3)(k+1)}n + o(n)$.

Table 2. Upper and lower bounds.

Relative complexity	Lower bound	Upper bounds
$\alpha(n, k)$	$n \log n/\log k!$	$n \log n/(k-1), 4n \log n/k \log k$
$\beta(n, k)$	$n/(k-1)$	$n/(k-1)$
$\gamma(n, k)$	$n/k + 2n/k(k-1)$	$n/k + 2n/k(k-1)$
$\delta(n, k$	$n/(k-1)$	$n/(k-1)$
$\varepsilon(n, k)$	$\dfrac{n}{\log\binom{k}{k/2}}$	$2n/k$
$\phi(n, k)$	$2n/3k$	$\dfrac{4(3k+1)}{(k-3)(k+1)}n$

The upper bound given by this algorithm on $\phi(n, k)$ is probably far from optimal. Certainly it is much larger than the lower bound from Lemma 1: $\phi(n, k) \geq \phi(n, 2)/\phi(k, 2) \geq 2n/3k$.

Table 2 summarises the lower and upper bounds on various relative complexity functions; only the dominant term is given in these bounds in order to display the degree to which the relative complexities are known almost exactly.

3. Finding the maximum and minimum.

In this section we consider the problem $C(n)$, give an algorithm for solving it using operations $C(k)$, and prove that the algorithm is close to optimal (thereby obtaining the value of $\gamma(n, k)$ to within a small constant).

The algorithm is as follows:

1. Divide the n elements into $\lceil n/k \rceil$ groups of k elements (possibly with a final group of r elements, $r < k$) and apply $C(k)$ to each group (and $C(r)$ to the final group if necessary). There are now $q = \lceil n/k \rceil$ candidates for the maximum and q candidates for the minimum.
2. With $\lceil (q-1)/(k-1) \rceil$ applications of $C(k)$ (one of which may be $C(r)$ with $r < k$) reduce the number of candidates for the maximum down to one.
3. With $\lceil (q-1)/(k-1) \rceil$ applications of $C(k)$ (one of which may be $C(r)$ with $r < k$) reduce the number of candidates for the minimum down to one.

From this algorithm we obtain

LEMMA 2. $\gamma(n, k) \leq q + 2\lceil (q-1)/(k-1) \rceil$.

Consider an arbitrary algorithm for finding both the maximum and the minimum in an n element set by means of $C(r)$ operations with $r \leq k$. As the algorithm proceeds more and more elements are excluded as possibilities for being the maximum or minimum. We define the following notation:

$\mathscr{A}$: the set of elements which remain candidates for both the minimum and the maximum,
$\mathscr{B}$: the set of elements which remain candidates for the maximum only,
$\mathscr{C}$: the set of elements which remain candidates for the minimum only,
$\mathscr{D}$: the set of elements which are candidates for neither.

Put $a = |\mathscr{A}|$, $b = |\mathscr{B}|$, $c = |\mathscr{C}|$, $d = |\mathscr{D}|$.

Each $C(r)$, $r \leq k$, operation results in changes to a, b, c, d which we denote by Δa, Δb, Δc, Δd and we shall have to consider worst case situations.

Initially $a = n$ and $b = c = 0$, while finally $a = 0$ and $b = c = 1$. This means that the total change to a is $-n$ and the total change to $b + c$ is 2.

The operations of the algorithm are categorised according to how many operands lie in $\mathscr{A}$, how many lie in $\mathscr{B}$, how many lie in $\mathscr{C}$ and how many lie in $\mathscr{D}$. Specifically, let

t_{wxyz} be the number of operations in which w operands lie in (the current) $\mathscr{A}$, x operands lie in $\mathscr{B}$, y operands lie in $\mathscr{C}$, z operands lie in $\mathscr{D}$. Of course, $w + x + y + z \leq k$.

The total number of operations used is $\sum t_{wxyz}$ where the summation is over all quadruples (w, x, y, z) of non-negative integers which satisfy $w + x + y + z \leq k$.

We divide the quadruples into the following disjoint sets

$$\begin{aligned}
E &= \{(w,x,y,z) \mid (w \geq 2) \text{ or } (w = 1 \text{ and } (x \geq 1 \text{ or } y \geq 1)) \\
&\qquad \text{or } (w = 0 \text{ and } x \geq 1 \text{ and } y \geq 1)\} \\
F &= \{(0,x,0,z) \mid x \geq 2\} \\
G &= \{(0,0,y,z) \mid y \geq 2\} \\
H &= \{(1,0,0,z)\} \\
J &= \{(0,x,y,z) \mid (x,y) = (0,1),(1,0),(0,0)\}.
\end{aligned}$$

For a quadruple (w,x,y,z) in any of these sets the associated $C(r)$ operation results in $\Delta a = -w$. The value $\Delta(b + c)$ depends on the outcome of the operation and we must consider the worst case. The worst case outcome depends on which of the classes above the associated quadruple belongs to.

For quadruples (w,x,y,z) in E there are always two operands at least one of which, σ say, is still a candidate for the maximum, the other, τ say, still a candidate for the minimum. The worst case is when an element such as σ is found to be a maximum and an element such as τ is found to be a minimum in the $C(r)$ operation. In such a case we have $\Delta(b + c) = -x - y + 2$.

Similar considerations tell us that the worst case for an operation parameterised by a quadruple (w,x,y,z) in F is $\Delta(b + c) = -x + 1$; for a quadruple in G it is $\Delta(b + c) = -y + 1$; for a quadruple in H it is $\Delta(b + c) = 1$; and for quadruples in J it is $\Delta(b + c) = 0$.

Summing over all the operations in the algorithm to obtain the total changes to Δa and $\Delta(b + c)$ in the worst case gives the equations

$$\sum_E w t_{wxyz} + \sum_H t_{100z} = n \tag{1}$$

$$\sum_E (-x - y + 2) t_{wxyz} + \sum_H t_{100z} + \sum_F (-x + 1) t_{0x0z} + \sum_G (-y + 1) t_{00yz} = 2 \tag{2}$$

These equations are linear in the quantities t_{wxyz} and we can therefore obtain a lower bound on the total number $\sum t_{wxyz}$ of $C(r)$ operations by a technique in linear programming. We use the two equations to express t_{k000} and $t_{0k00} + t_{00k0}$ in terms of the other t_{wxyz}. Then we substitute for t_{k000} and $t_{0k00} + t_{00k0}$ in the expression $\sum t_{wxyz}$. Elementary, though tortuous, algebra results in the following equation:

$$\sum t_{wxyz} = \frac{n}{k} + \frac{2(n-k)}{k(k-1)} + \sum_{E - \{(k,0,0,0)\}} \frac{k(k + 1 - w - x - y) - w}{k(k-1)} t_{wxyz}$$

$$+ \sum_{F - \{(0,k,0,0)\}} \frac{k - x}{k - 1} t_{0x0z} + \sum_{G - \{(0,0,k,0)\}} \frac{k - y}{k - 1} t_{00yz} +$$

$$+\sum_{H}\frac{k(k-1)-1}{k(k-1)}t_{100z}+\sum_{J}t_{0xyz}.$$

Since the coefficients in the summations on the right hand side are strictly positive we have

$$\sum t_{wxyz}\geq\frac{n}{k}+\frac{2(n-k)}{k(k-1)}.$$

We can now prove the following.

THEOREM 1.

$$0\leq\gamma(n,k)-\left(\frac{n}{k}+\frac{2(n-k)}{k(k-1)}\right)\leq 3+\frac{2}{k-1}.$$

PROOF. The linear programming argument above has shown that

$$\gamma(n,k)\geq\frac{n}{k}+\frac{2(n-k)}{k(k-1)}$$

and so, in conjunction with Lemma 2, we have

$$0\leq\gamma(n,k)-\left(\frac{n}{k}+\frac{2(n-k)}{k(k-1)}\right)$$

$$\leq\left\lceil\frac{n}{k}\right\rceil+2\left\lceil\frac{\lceil n/k\rceil-1}{k-1}\right\rceil-\frac{n}{k}-2\frac{n/k-1}{k-1}$$

$$=\left\lceil\frac{n}{k}\right\rceil-\frac{n}{k}+2\left\lceil\frac{\lceil n/k\rceil-1}{k-1}\right\rceil-2\frac{\lceil n/k\rceil-1}{k-1}+2\frac{\lceil n/k\rceil-1}{k-1}-2\frac{n/k-1}{k-1}$$

$$e\leq 1+2+2\frac{\lceil n/k\rceil-n/k}{k-1}\leq 3+\frac{2}{k-1}.$$

REFERENCES

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

[2] M. D. Atkinson, *Sorting with powerful primitive operations*, J. Comb. Math. and Comb. Comput. 4 (1988), 29–36.

[3] R. Beigel, J. Gill, *Sorting n objects with a k-sorter*, IEEE Trans. on Computers, 39, No. 5 (May 1990), 714–716.

[4] S. W. Bent, J. W. John, *Finding the median requires 2n comparisons*, Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (1985), 213–216.

[5] D. E. Knuth, *Sorting and Searching*, The Art of Computer Programming, vol. 3, Addison-Wesley, Reading, Massachusetts, 1973.

[6] M. S. Paterson, A. Schönhage, N. Pippenger, *Finding the median*, J. Computer and System Sciences 13 (1976), 184–199.