

OPTIMAL RANDOMIZED ALGORITHMS FOR LOCAL SORTING AND SET-MAXIMA*

WAYNE GODDARD[†], CLAIRE KENYON[‡], VALERIE KING[§], AND
LEONARD J. SCHULMAN[¶]

Abstract. We present randomized algorithms for two sorting problems. In the local sorting problem, a graph is given in which each vertex is assigned an element of a total order, and the task is to determine the relative order of every pair of adjacent vertices. In the set-maxima problem, a collection of sets whose elements are drawn from a total order is given, and the task is to determine the maximum element in each set. We describe lower bounds for the problems in the comparison model, and show that the algorithms are optimal within a constant factor.

Key words. sorting, randomized algorithms, comparison model, partial order, graph algorithms

1. Introduction. In this paper we study two sorting problems. The first is the *local sorting* problem: given a graph in which each vertex is assigned an element of a total order, one must determine the relative order of every pair of adjacent nodes. This problem restricts to standard sorting when the graph is complete, but in general its complexity depends on the graph selected. The second problem is *set-maxima*, where the task is to identify the maximum element in each of a collection of sets drawn from a total order. Set-maxima was investigated in [1], [3] and [5]. Local sorting appears to be new, although a restricted version was suggested in [6].

We present randomized algorithms for these problems and measure their complexity in the comparison (decision-tree) model. In that model, one pays only for comparisons between elements of the total order. The algorithms use random bits to help determine which comparisons will be made, but their outcomes must be correct. The complexity of the algorithm is thus the expected number of comparisons made on a worst-case input. We obtain information-theoretic lower bounds for both problems, and show that our algorithms attain these bounds.

The output of a local-sorting algorithm is exactly an acyclic orientation of the graph G . Thus an information-theoretic lower bound on the complexity of local sorting, even for randomized algorithms, is given by the logarithm of the number, $\alpha(G)$, of acyclic orientations of the graph G . We present an algorithm which is optimal for all graphs—it makes an expected $\Theta(\log \alpha(G))$ comparisons.

We derive an estimate of $\alpha(G)$ in terms of the degrees, $\{d_v\}$, of the vertices $\{v\}$ of G : specifically $\log \alpha(G)$ is $\Theta(\sum_{v \in G} \log(d_v + 1))$. Thus the number of comparisons our algorithm makes contrasts with the $\Theta(\sum_{v \in G} d_v)$ comparisons made by the naive algorithm which examines all edges.

Closely related to local sorting is the *set-sort* problem: sort each of a family of possibly overlapping sets. Set-sort is equivalent to locally sorting the graph in which

*This paper includes the extended abstracts [2] and [4]

[†]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by NSF grant 8912586-CCR and DARPA contract N00014-89-J-1988.

[‡]LIENS, Ecole Normale Supérieure, 75230 Paris Cedex 05, France. The research of this author was supported in part by INRIA.

[§]NECI, Princeton, New Jersey 08540. The research of this author was supported in part by a grant from NSERC and ITRC at the University of Toronto.

[¶]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by NSF grant 8912586-CCR, DARPA contract N00014-89-J-1988 and an ONR Graduate Fellowship

each set induces a clique. We also observe that local sorting may be thought of as a special case of set-maxima (where every set is a pair).

One application of local sorting is to the *element uniqueness problem on a graph*: given a graph G with each vertex assigned an element of a total order, are all pairs of adjacent values distinct? Manber and Tompa [7] introduced this question and showed that $\log \alpha(G)$ comparisons are necessary for this decision problem. Up to a constant factor, this is the bound achieved by our local sorting procedure (which can identify every equality).

The second problem we study, set-maxima, was introduced in [3] where an argument of Fredman was presented showing that, given m sets over an n -element universe, at most $\binom{m+n-1}{n-1}$ arrangements of maxima are possible. We in fact observe an information-theoretic lower bound of $\Omega(n \log(m/n) + n)$, matching this limitation.

We describe a randomized algorithm for set-maxima which performs an expected $O(n \log(m/n) + n)$ comparisons. By the above this is optimal in terms of n and m .

Local sorting provides a reasonable solution to the *partial order verification* problem suggested by A. Yao [9]: given a directed graph G with elements of a total order at each of the n vertices, verify that the orientation of every edge agrees with the order. But it is best to reduce the problem to set-maxima. In the reduction each set consists of the immediate predecessors of a vertex of the graph. Our set-maxima algorithm yields an optimal $O(n)$ -comparison algorithm for this problem.

Deterministic algorithms are known for a few special cases of set-maxima. Komlós [5] made use of a reduction to set-maxima when he solved the *minimum spanning tree verification* problem. His deterministic algorithm finds the tree edge of largest weight in every simple cycle containing exactly one non-tree edge. It solves this instance of set-maxima in $O(n \log(m/n))$ comparisons, where n and m are the number of tree and non-tree edges. Then it verifies that the every non-tree edge is greater than the appropriate maximum. We have been informed of an $O(m+n)$ -time implementation of this algorithm by Tarjan [8].

Recently Bar-Noy, Motwani, and J. Naor [1] gave a deterministic algorithm which uses $O(n)$ comparisons when the n sets are the hyperplanes in a projective geometry. They also gave an $O(n)$ -comparison algorithm for the case where n sets are chosen randomly, so that each element appears in each set with probability $p(n)$.

This paper has three main sections: the first deals with local sorting, the second with set-maxima, and some open problems are discussed in the third.

2. Local Sorting. At the heart of our approach to local sorting is the *limited-degree* algorithm (§2.1) whose complexity, $O(n \log(\Delta + 1))$, is a function only of the number n of vertices and the maximum degree Δ of a graph G (where Δ may be a function of n). However, the maximum degree of G is too crude a measure of its complexity for local sorting. A more accurate measure is the logarithm of $\mathcal{D}(G)$, defined by:

$$\mathcal{D}(G) = \prod_{v \in G} (d_v + 1),$$

where d_v is the degree of v in G . In §2.3 we show that $\log \mathcal{D}(G)$ is $\Theta(\log \alpha(G))$ which is a lower bound on the complexity of local sorting. In §2.2 we describe a *reduction* procedure which alters the graph so that we can achieve the optimal $\Theta(\log \mathcal{D}(G))$ comparisons with several applications of the limited-degree algorithm.

We present our results under the assumption that the values at the vertices are distinct. Minor modifications suffice to cover the more general case where equalities

FIG. 2.1.

are allowed, and “local sorting” includes reporting the equalities as such.

2.1. The Limited-Degree Local Sorting Algorithm. We show here how to locally sort in a graph with vertex set X ($|X| = n$) and maximum degree Δ . The idea is to take a series of increasingly large random samples of the vertices and partially order each sample using the information given by the partial order on the previous sample.

We produce a series of samples $R_k, R_{k-1}, \dots, R_0 = X$ (where k will be defined later). Starting with R_k , we then:

- Partially order each R_i so that it is *locally sorted to radius 2^{2^i}* .

By this we mean that the relative order of vertices in R_i is known if, as measured in G , they are within distance 2^{2^i} of each other. The end result is that $R_0 = X$ is locally sorted to radius 1, as called for.

Algorithm. We choose the samples as follows. We let $R_0 = X$. Then we randomly choose R_i from within R_{i-1} by taking elements independently with probability p_i/p_{i-1} . Thus the sample R_i has expected size np_i . (The descending sequence p_i will be specified later.)

We continue until k such that R_k has expected size $O(1)$. This we then sort using an expected $O(1)$ comparisons.

The main part of the algorithm is an iterative process in which the partial order on R_i is used to obtain the requisite partial order on R_{i-1} . The R_i are used as “signposts” for the two parts of this process. Let B_x^r denote the set of vertices of distance at most r from vertex x . Proceed as follows.

Step 1. For each $x \in R_{i-1}$ do:

- Find the rank of x in $R_i \cap B_x^{2^{2^{i-1}}}$ using a binary search.

Any two elements of R_i in $B_x^{2^{2^{i-1}}}$ are within a distance of 2^{2^i} of each other—so, since R_i is locally sorted to distance 2^{2^i} , their relative order is already known (i.e. $R_i \cap B_x^{2^{2^{i-1}}}$ has been totally ordered). This allows us to find x ’s rank with a binary search.

Step 2. For each $x \in R_{i-1}$ do:

- Determine C_x , the set of all $y \in R_{i-1} \cap B_x^{2^{2^{i-2}}}$ that have the same rank as x with respect to $R_i \cap B_x^{2^{2^{i-2}}}$.

Note that for y in $B_x^{2^{2^{i-2}}}$, $B_y^{2^{2^{i-1}}}$ contains $B_x^{2^{2^{i-2}}}$. (See Fig. 2.1.) Thus C_x can be constructed using no further comparisons.

Step 3. For each pair $x, y \in R_{i-1}$ do:

- If $x \in C_y$ and $y \in C_x$ then compare x and y .

If either $x \notin C_y$ or $y \notin C_x$ while x and y are at most distance $2^{2^{i-2}}$ apart, then there is at least one intervening signpost (element of $R_i \cap B_x^{2^{2^{i-2}}}$ or $R_i \cap B_y^{2^{2^{i-2}}}$ whose value is between x and y), and therefore their relative order is known. Thus Step 3 is sufficient to determine for each $x \in R_{i-1}$ the relation between it and every element of $R_{i-1} \cap B_x^{2^{2^{i-2}}}$. I.e. Step 3 locally sorts R_{i-1} to radius $2^{2^{i-2}}$.

Analysis. How many comparisons do we use?

Step-by-step analysis. For each $x \in R_{i-1}$, the number of comparisons in Step 1 is at most $\log |B_x^{2^{2^{i-1}}}|$ which, by our assumption on the degrees, is at most

$2^{2i-1} \log(\Delta + 1)$. Thus the expected number of comparisons for the whole of Step 1 is at most:

$$(2.1) \quad np_{i-1} 2^{2i-1} \log(\Delta + 1).$$

Step 2 involves no comparisons.

We bound the number of comparisons in Step 3 by $\frac{1}{2} \sum_{x \in R_{i-1}} |C_x|$, since each $x \in R_{i-1}$ is involved in at most $|C_x - \{x\}|$ comparisons. The following lemma provides a bound on the conditional expectation of $|C_x|$ given R_{i-1} :

LEMMA 2.1. *For any R_{i-1} and any $x \in R_{i-1}$, the expected size of C_x is at most $2p_{i-1}/p_i$.*

Proof. Consider first the set B^+ consisting of those elements of $B_x^{2^{2i-2}} \cap R_{i-1}$ greater than x . Then let z^+ be the smallest element of $B^+ \cap R_i$, and note that $B^+ \cap C_x$ consists of those elements of B^+ less than z^+ . Each element of B^+ is in R_i with probability $p = p_i/p_{i-1}$. Hence (as this is a Bernoulli process), if B^+ were infinite, $|B^+ \cap C_x| + 1$ would have a geometric distribution with parameter p and thus expected value $1/p$. The finiteness of B^+ only reduces this value.

Now define B^- and z^- analogously (so that $E(|B^- \cap C_x|) < (1-p)/p$). As $C_x = (B^+ \cap C_x) \cup (B^- \cap C_x) \cup \{x\}$, it follows that $E(|C_x|) < 1 + 2(1-p)/p$. \square

Thus the expected number of comparisons in Step 3, given *any* particular R_{i-1} , is bounded by $|R_{i-1}| p_{i-1}/p_i$. Therefore the expected number of comparisons for Step 3 is at most the expectation (ranging over R_{i-1}) of this bound, which is:

$$(2.2) \quad p_{i-1}/p_i E(|R_{i-1}|) = np_{i-1}^2/p_i.$$

Definition of the parameters. A good choice of the $\{p_i\}$ is obtained by balancing the costs (2.1) and (2.2). This requires $2^{2i-1} \log(\Delta + 1) = p_{i-1}/p_i$. Solving for p_i in terms of p_{i-1} and noting that $p_0 = 1$, this leads us to choose:

$$p_i = \frac{1}{(\log(\Delta + 1))^{i^2} 2^{i^2}}.$$

The value k is chosen such that $E(|R_k|) = O(1)$. For this it is sufficient that $k = \lceil \sqrt{\log n} \rceil$.

Total number of comparisons. Substituting this choice of p_i into bounds (2.1) and (2.2) and summing, it follows that the expected number of comparisons in going from R_i to R_{i-1} is at most:

$$\frac{n}{(\log(\Delta + 1))^{i-2} 2^{i^2-4i+1}}.$$

Therefore the expected number of comparisons in going from R_k to R_1 is at most:

$$\sum_{i=2}^k \frac{n}{(\log(\Delta + 1))^{i-2} 2^{i^2-4i+1}} \leq n \sum_{i \geq 2} 2^{-i^2+4i-1} < 13n.$$

(This bound is independent of Δ : this is possible because Δ determined the size of R_1 .) Thus the overall number of comparisons is dominated by the transition from R_1 to R_0 , and we have:

THEOREM 2.2. *The limited-degree local sorting algorithm requires*

$$E(\text{comparisons}) \leq 4n \log(\Delta + 1) + 13n$$

on a graph with n vertices and maximum degree Δ .

As an example, if the degree of every vertex is polylogarithmic in n , then the procedure runs in $O(n \log \log n)$ comparisons.

FIG. 2.2. *Example of the Graph Reduction*

2.2. A Reduction for Arbitrary Graphs. In this subsection we show how to locally sort an arbitrary graph in $O(\log \mathcal{D}(G))$ comparisons. This is achieved through a reduction to the limited-degree case. In the reduction we transform G into a new graph H in which every edge of G is represented exactly once and each vertex of G is represented several times. H is constructed such that locally sorting each component of H with the limited-degree algorithm represents an efficient way of locally sorting G . Our reduction is deterministic and requires no comparisons.

Given a graph G , first discard all isolated vertices. Then let U_0 be the set of vertices of degree 1, and $G_0 = G - U_0$. We define U_i and G_i inductively for $i \geq 0$ by: U_i is the set of vertices of degree at most $a_i = 2^{2^i} - 1$ in G_{i-1} , and $G_i = G_{i-1} - U_i$. Halt when G_i is the null graph.

We now consider the graph whose vertex set is $U_i \cup N_{i-1}(U_i)$ (where $N_{i-1}(U_i)$ denotes the set of neighbors of U_i in G_{i-1}), and whose edges are the edges that are interior to U_i or that connect U_i to $N_{i-1}(U_i)$. In that graph, some vertices of $N_{i-1}(U_i)$, the “red” vertices, have degree at most a_i . Others have degree higher than a_i : we split each of these into several “green” vertices each retaining between $a_i/2$ and a_i of its originator’s edges. (The coloring will be used in the analysis.) Let H_i be this new graph. An example is shown in Fig. 2.2.

We now apply the limited-degree local sorting algorithm (§2.1) to each H_i . This is equivalent to locally sorting G .

THEOREM 2.3. *Given any graph G , the local sorting algorithm uses an expected $O(\log \mathcal{D}(G))$ comparisons.*

Proof. Let $h_i = |H_i|$. The maximum degree of H_i is at most a_i . Thus we require an expected $O(\sum_i h_i \log(a_i + 1))$ comparisons to locally sort H . We separate the contributions to $\eta = \sum_i h_i \log(a_i + 1)$ into the contribution of the vertices in $\bigcup U_i$, the contribution of the red vertices, and that of the green vertices. (Note that $\log(a_i + 1) = 2^i$.)

Every non-isolated vertex v of G shows up, perhaps lacking some of its original edges, in just one U_i . For each $j < i$, $N_{j-1}(U_j)$ may contain vertices corresponding to v : either several green vertices, or just one red vertex. Note that $i \leq \lceil \log \log(d_v + 1) \rceil$.

Consider a vertex v' in U_i , derived from v in G . As v was *not* chosen for U_{i-1} , its degree d_v must exceed a_{i-1} . Therefore $\log(d_v + 1) \geq \log(a_{i-1} + 1) = \frac{1}{2} \log(a_i + 1)$. Thus the contribution of $\bigcup U_i$ to η is at most $2 \log \mathcal{D}(G)$.

A vertex v of G has at most one red vertex derived from it in each $N_{i-1}(U_i)$ for $j \leq \lceil \log \log(d_v + 1) \rceil - 1$. Hence its total red contribution is at most:

$$\sum_{j=1}^{\lceil \log \log(d_v + 1) \rceil - 1} 2^j \leq 2 \log(d_v + 1).$$

Therefore the contribution of the red vertices is at most $2 \log \mathcal{D}(G)$.

The green vertices of H_i are adjacent only to vertices of U_i . Further, green vertices of H_i are of degree at least $a_i/2$ while those of U_i are of degree at most a_i . Thus there are at most $2|U_i|$ green vertices, and their contribution is at most twice that from $\bigcup U_i$.

Combining all the above it follows that $\eta \leq 8 \log \mathcal{D}(G)$. \square

Set-Sort. Using the concavity of the log function, we can conclude from the above that if G has m edges, then our local sorting algorithm uses $O(n \log((2m + n)/n))$ comparisons. Applying this to the set-sort problem, we obtain:

COROLLARY 2.4. *Let S_1, S_2, \dots, S_m be sets from a totally ordered universe of size n . Then these sets can be sorted using an expected $O(n \log(\sum_j |S_j|^2/n))$ comparisons.*

2.3. The Acyclic Orientations of a Graph. In this subsection we provide an estimate of the number $\alpha(G)$ of acyclic orientations of a graph G , based solely on the degrees of the vertices of G . The upper bound is due to Manber and Tompa [7]. (The parameter α has also been studied, for instance, in [?].)

Define:

$$\mathcal{F}(G) = \prod_{v \in G} f(d_v + 1),$$

where d_v is the degree of v in G , and $f(x) = (x!)^{1/x}$.

THEOREM 2.5. *For any graph G ,*

$$\sqrt{\mathcal{D}(G)} \leq \mathcal{F}(G) \leq \alpha(G) \leq \mathcal{D}(G).$$

In particular $\log \mathcal{D}(G) = \Theta(\log \alpha(G))$.

Proof. The proof that $\alpha(G) \leq \mathcal{D}(G)$ is in [7]. For positive integral x , $\sqrt{x+1} \leq f(x+1)$, hence $\sqrt{\mathcal{D}(G)} \leq \mathcal{F}(G)$. Now we show that $\mathcal{F}(G) \leq \alpha(G)$.

The proof is by induction on the number of vertices of G . The case of a single vertex is trivial.

Let v be a vertex of minimum degree δ . If $\delta = 0$ then $\alpha(G) = \alpha(G - v)$ and $\mathcal{F}(G) = \mathcal{F}(G - v)$; so hereafter we assume that $\delta \geq 1$. We prove first that:

$$(2.3) \quad \alpha(G) \geq (\delta + 1)\alpha(G - v).$$

Take any acyclic orientation \mathcal{A} of $G - v$ and look at v 's neighbors N in G . If N is totally ordered by \mathcal{A} then there are $\delta + 1$ ways of extending \mathcal{A} to an acyclic orientation of G (by choosing v 's rank with respect to N); otherwise there are even more ways to extend \mathcal{A} .

The next thing to notice is that

$$\mathcal{F}(G) = \mathcal{F}(G - v)f(\delta + 1) \left(\prod_{v \in N} \frac{f(d_v + 1)}{f(d_v)} \right),$$

where $\{d_v\}$ indicate degrees in G (each greater than or equal to δ). It can be shown that $f(x+1)/f(x)$ is monotone decreasing for positive integral x . Therefore

$$\mathcal{F}(G) \leq \mathcal{F}(G - v)f(\delta + 1) \left(\frac{f(\delta + 1)}{f(\delta)} \right)^\delta.$$

Thus from inequality 2.3 above and the induction hypothesis, we find that

$$\alpha(G) \geq (\delta + 1)\mathcal{F}(G - v) \geq \mathcal{F}(G) \frac{(\delta + 1)(f(\delta))^\delta}{(f(\delta + 1))^{\delta+1}} = \mathcal{F}(G).$$

□

The bounds on $\alpha(G)$ are the best possible of the form $\prod_v g(d_v)$. The lower bound is attained by any disjoint union of cliques. For the upper bound consider $G = K(a, b)$ with $b \gg a$: here $\alpha(G) \sim a!(a+1)^b$. (We would like to thank Mark Haiman for discussions on this point.) The latter graphs also disprove the upper bound $\prod_v \max\{2, d_v\}$ claimed in [6].

Conclusion. The results of this section establish the following:

THEOREM 2.6. *For any graph G , the local sorting algorithm uses an expected*

$$\Theta(\log \alpha(G)) = \Theta\left(\sum_{v \in G} \log(d_v + 1)\right)$$

comparisons, and is optimal up to a constant factor.

3. Set-Maxima. One can use the set-sort algorithm to solve set-maxima; this is fine when the sets are very small, but is inefficient for large sets. The key idea in handling large sets is a *reduction* process: take a random sample, partially sort it, and then use this to reduce the sizes of the sets. Another useful idea is a generalization of set-maxima, which we call *t-maxima*, in which one must find and sort the t largest elements in each set.

In §3.1 we discuss our approach to set-maxima, and in §3.2 we provide the algorithm for *t-maxima*. In §3.3 we analyze the algorithm and show that for $t = O(1)$ it uses an expected $O(n \log(m/n) + n)$ comparisons, where m is the number of sets and n the size of the universe. This we show is optimal as a function of m and n .

3.1. The Approach to Set-Maxima. In this subsection we outline the set-maxima algorithm. Assume we have sets S_1, S_2, \dots, S_m in a universe X of size n . Our general strategy for set-maxima can be summarized in three stages.

1. *Choose a random sample R , and determine for each set S_i a “representative” r_i which is the maximum sample point in that set.*
2. *Determine for each set S_i the reduced set S'_i formed by discarding those elements less than the representative.*
3. *Solve set-maxima on the reduced system.*

Observe that the maximum element in the reduced set is the same as that in the original set.

Implementation of Stage 2 if R is sorted. We look here at an efficient implementation of Stage 2 when the sample R has been sorted. Define for each x the “dual” set

$$T_x = \{r_j : x \in S_j\}.$$

To determine the reduced sets we need to determine the relationship between x and r_j for all j and $x \in S_j$. This is equivalent to finding for each x , the set $\{r_j : r_j \in T_x \& r_j < x\}$.

We perform a “doubling” search, rather than a binary search, to find the interval of T_x in which x lies: One compares x with the elements of T_x of ranks $1, 2, 4, \dots, 2^l$ from the bottom, until an element greater than x , if there is any, is found, and then performs a binary search in the interval $[2^{l-1}, 2^l)$ to find where x lies.

A doubling search is more efficient than an ordinary binary search because the average x is likely to be smaller than most of the representatives, and thus near the bottom of T_x .

Example: Let m be $\Theta(n \log^c n)$. We solve set-maxima as follows. First we take a random sample R where each element of X is chosen independently with probability $1/\log n$. Then we sort R , achieving Stage 1. It can be shown that then the expected size of each reduced set is $O(\log n)$, and thus that the doubling searches of Stage 2 take a total of $O(n \log \log n)$ comparisons. (Proofs omitted.) We then

use an expected $O(n \log \log n)$ comparisons to apply the set-sort algorithm to the reduced sets, determining their maxima. Thus the entire procedure requires an expected $O(n \log \log n) = O(n \log(m/n))$ comparisons.

Generalization. It appears that the $O(n \log \log n)$ comparisons of the example is the best one can do with a completely sorted sample. However, if m is small this quantity is $\omega(n \log(m/n))$, and so, in general, we cannot afford to completely sort the sample. We therefore must handle Stage 2 differently. We note that for most $x \in X$, the search of T_x ends after x has been compared with only the smallest elements of T_x . This motivates an extension of set-maxima:

Definition: In the t -maxima problem, one is given a set system and a parameter t , and one must find and sort the t largest elements of each set. The t -minima problem is defined analogously.

It is this level of generality which proves robust in our recursion.

We implement Stage 2 as follows. We first invoke u -minima on the $\{T_x\}$ to find and sort the u smallest elements in each T_x . (The parameter u will be specified later.) We then attempt for each x the doubling search of T_x as before. But we are able to complete the doubling search only when x is smaller than the u^{th} smallest element of T_x . We set aside the few “bad” x ’s for which the doubling searches fail, and handle them separately.

3.2. The Set-Maxima Algorithm. In this subsection we present the full set-maxima algorithm. Given an element x and a set A containing x , we say that the rank (bottom-rank) of x in A is the number of elements not less than (not greater than) x .

We are given a universe \mathcal{X} of n elements from some total order; and S_1, S_2, \dots, S_m , subsets of \mathcal{X} . We will assume that $m \geq 2n$.

The algorithm **Large** is defined below, recursively for X, S_1, S_2, \dots, S_m subsets of the universe, and a parameter $t \geq 16$. **Large** solves t -maxima on the collection $\{X \cap S_j\}_{j \leq m}$. **Small** is defined analogously for t -minima.

We call the procedure **Large**(16, \mathcal{X} , $\{S_j\}_{j \leq m}$) to solve the set-maxima problem on \mathcal{X} and S_1, S_2, \dots, S_m .

Large($t, X, \{S_j\}_{j \leq m}$):

If $|X| \leq 16$ then sort X .

Else

Step 1. Let R be the sample generated by choosing each $x \in X$ independently with probability $1/t$.

Do **Large**($t, R, \{S_j\}$). For $1 \leq j \leq m$, let r_j be the element of rank t in $R \cap S_j$, or $-\infty$ if there is no such element.

The call to **Large**($t, R, \{S_j\}$) finds the t largest sample elements in each set S_j . Then, rather than taking the representative to be the maximum sample element in each set, we choose the t^{th} largest value. This is so that each reduced set contains at least t elements, if the original set started with that many.

The implementation of Stage 2 has three steps:

Step 2A. For each $x \in X$, let $T_x = \{r_j : S_j \ni x\}$. Do **Small**($mt^3/n, R, \{T_x\}_{x \in X}$).

This finds and sort the mt^3/n smallest elements in each T_x .

Step 2B. For each $x \in X$, compare x with the element τ_x of T_x of rank mt^3/n from the bottom. If $x > \tau_x$ then put x in Y .

If $x \leq \tau_x$, then do a doubling search on the sorted portion of T_x : compare x to the elements of T_x of bottom-ranks $2^0, 2^1, \dots$ until you find x is less than

some element of bottom-rank $2^k \leq mt^3/n$. Then perform a binary search in the interval between 2^{k-1} and 2^k until x is placed among the elements of T_x .

If x is high up in T_x , specifically, greater than the element of bottom-rank mt^3/n , it is placed aside in Y . Otherwise a doubling search is performed. After this we know for each S_j the subset S_j^1 consisting of the $x \in S_j \cap (X - Y)$ which are at least as big as the representative r_j .

Step 2C. Do **Large**($t, Y, \{S_j\}$).

This yields for each S_j the subset S_j^2 consisting of the t largest elements in $S_j \cap Y$ (if there are that many). By combining S_j^1 and S_j^2 , we obtain for each S_j the reduced set S_j' which contains its t largest elements (if S_j had that many originally).

Step 3. Let $S_j' = \{x \in S_j \cap (X - Y) : x \geq r_j\} \cup \{\text{largest } t \text{ elements of } S_j \cap Y\}$.

Apply the set-sort algorithm (cf. Corollary 2.4) to the sets $\{S_j'\}_{j \leq m}$.

Step 3 sorts each reduced set, and yields the t largest elements in each $S_j \cap X$, as required.

3.3. Analysis. We analyze the expected number of comparisons performed during the procedure

Large($t, X, \{S_j\}_{j \leq m}$). All comparisons occur in Steps **2B** and **3**, and in the recursive calls.

Comparisons in Steps 2B and 3. For $1 \leq j \leq m$ let $\rho(r_j)$ denote the rank of r_j in $S_j \cap X$.

The number of comparisons needed for each element $x \in X$ in Step **2B**, is the cost of searching for the interval in which x lies, and, if one is found, the cost of locating x within the interval. Thus, Step **2B** requires at most:

$$\sum_{x \in X} 2 \log (|\{j : r_j \leq x, x \in S_j \cap X\}|)$$

comparisons. But $\sum_{x \in X} |\{j : r_j \leq x, x \in S_j \cap X\}|$ is equal to $\sum_j |\{x \in S_j \cap X : x \geq r_j\}|$, the sum of the ranks of the representatives. Hence, by the concavity of the log function, Step **2B** requires at most:

$$(3.1) \quad 2n \log \left(\sum_j \rho(r_j)/n \right)$$

comparisons.

In Step **3**, the m sets S_j' have size at most $t + \rho(r_j)$. Therefore the set-sort algorithm (cf. Corollary 2.4) takes at most

$$(3.2) \quad c_0 n \log \left(\sum_j (t + \rho(r_j))^2 / n \right)$$

comparisons, for c_0 a constant.

To compute the expectations of expressions (3.1) and (3.2), we determine the following upper bounds.

LEMMA 3.1. $E(\rho(r_j)) \leq t^2$ and $E((t + \rho(r_j))^2) \leq 2t^4$.

Proof. If $S_j \cap X$ were infinite, then $\rho(r_j)$ would be the rank of the t^{th} sample element in that set. Elements are in the sample independently with probability $1/t$; so we would have a Bernoulli process, and $\rho(r_j)$ would be given by the sum of t independent geometric random variables with parameter $1/t$. For a geometric random

variable Z with parameter p , $E(Z) = 1/p$ so that $E(\rho(r_j)) \leq t^2$. Further, $E(Z^2) = (2-p)/p^2$ and therefore a simple calculation shows that $E((t + \rho(r_j))^2) \leq 2t^4$. \square

Because of the concavity of logarithms, we thus have that the expected sum of expressions (3.1) and (3.2) is at most:

$$2n \log((mt^2)/n) + c_0 n \log((2mt^4)/n) \leq cn \log(mt/n),$$

with $c = 4 + 4c_0$, since $m \geq 2n$.

Full analysis. Let $f(t, n, m)$ denote the average cost of **Large**($t, X, \{S_j\}$), including recursive calls, when X and $\{S_j\}$ are the worst possible input such that $|X| = n$ and the number of subsets is m . We will show by induction on n that for $m \geq 2n$ and $t \geq 16$:

$$(3.3) \quad f(t, n, m) \leq 2cn \log(mt/n).$$

This is true for $n \leq 16$: then $t \geq n$, while the sort of X takes at most $dn \log(n)$ for d a small constant.

So assume that $n > 16$. Then, examining all stages of the algorithm, we see that $f(t, n, m)$ satisfies the following equation:

$$f(t, n, m) \leq cn \log(mt/n) + E(f(t, |R|, m)) + E(f(mt^3/n, |R|, n)) + E(f(t, |Y|, m)),$$

where the expectations are taken over the distribution of the sample R .

By induction and because $n \log 1/n$ is concave, we may write:

$$f(t, n, m) \leq cn \log(mt/n) + f(t, E(|R|), m) + f(mt^3/n, E(|R|), n) + f(t, E(|Y|), m).$$

LEMMA 3.2. $E(|R|) = n/t$ and $E(|Y|) \leq n/t$.

Proof. The first part of the lemma is obvious. To prove the second part, we note that $x \in Y$ if and only if x is greater than at least mt^3/n values r_j for which $S_j \ni x$. Thus $(mt^3/n)|Y| \leq |\{(x, j) : x \geq r_j, x \in S_j\}| = \sum_j \rho(r_j)$. Hence $E(|Y|) \leq mt^2/(mt^3/n) = n/t$. \square

The equation bounding f can now be formulated as

$$(3.4) \quad f(t, n, m) \leq cn \log(mt/n) + 2f(t, n/t, m) + f(mt^3/n, n/t, n).$$

Observe that if $m \geq 2n$ and $f(t', n', m')$ is one of the terms on the right-hand side of (3.4), then $t' \geq t$ and $m' \geq 2n'$.

Thus, by our inductive hypothesis (3.3),

$$\begin{aligned} f(t, n, m) &\leq cn \log(mt/n) + 2(2c(n/t) \log(mt^2/n)) + 2c(n/t) \log(mt^4/n) \\ &\leq cn \log(mt/n)(1 + 8/t + 8/t) \\ &\leq 2cn \log(mt/n), \end{aligned}$$

since $t \geq 16$ and $m \geq 2n$.

THEOREM 3.3. *Let m sets be given from a totally ordered n -element universe, with $m \geq 2n$, and a parameter $t \geq 16$. Then the expected number of comparisons required by the t -maxima algorithm is $O(n \log(mt/n))$. In particular for all m we solve set-maxima in*

$$\Theta(n \log(m/n) + n)$$

expected comparisons which is optimal as a function of n and m .

We established the upper bound above. Here is the lower bound. (Note that for $m \geq n$, $\Theta(\log(mt/n)) = \Theta(\log(mt^2/n))$.)

THEOREM 3.4. *If mt^2 is $O(n^2)$ then there exist collections of m sets such that*

$$\Omega(n \log(mt^2/n) + n)$$

comparisons are required for t -maxima.

Proof. If one set contains all elements then $n - 1$ comparisons are necessary. On the other hand, one may partition the universe into blocks of size B and arrange $\Theta(nB/t^2)$ sets of size t such that t -maxima forces each block to be sorted. This requires $\Omega(n \log B)$ comparisons. \square

4. Open Questions. We list below some questions which we have encountered.

(1) What about deterministic algorithms? No nontrivial deterministic algorithm for local sorting is known other than on dense graphs where one might as well totally sort the elements. The same is true of set-maxima with the exception of the restricted set systems discussed in [5] and [1].

(2) Can the time complexity of the set-maxima and local sorting algorithms (currently polynomial) be brought into line with their comparison model complexity?

(3) We have shown that there exist collections of m sets for which the information-theoretic lower bound for set-maxima matches the complexity of our algorithm. However, some collections of m sets have fewer arrangements of maxima. Is there a set-maxima algorithm which does better on such collections? We do not know of a simple characterization of the number of possible arrangements of maxima for an arbitrary collection of sets.

(4) A fourth question is raised by our local sorting algorithm, which makes comparisons between distant elements of the graph in order to finally determine the local order relationships. Is this necessary? Specifically: Is there a local sorting algorithm which uses $\Theta(\log \alpha(G))$ comparisons but never compares elements over a distance greater than some constant *comparison horizon* h ? Our algorithm, modified so that $E(|R_k|) = n/\log n$, needs $h(n) = \sqrt{\log \log n}$. The question is particularly intriguing because our algorithm actually makes rather few long-distance comparisons (e.g. only linearly many beyond distance 4).

We know only that $h = 1$ is not sufficient. This answers a question of Linial [6]. Consider a bipartite graph where all values in one color class are greater than those in the other: all edges must be checked in order to certify this.

(5) A combinatorial question is that of obtaining better upper and lower bounds on $\alpha(G)$.

Acknowledgements. The authors would like to thank Michael Sipser for many discussions and considerable advice. Further, they would like to thank Michelangelo Grigni, Mark Haiman, Russell Impagliazzo and Mauricio Karchmer for helpful discussions.

- [1] A. BAR-NOY, R. MOTWANI, AND J. NAOR, *A linear-time approach to the set maxima problem*, SIAM J. Discrete Math., 5 (1992), pp. 1–9.
- [2] W. GODDARD, V. KING, AND L. SCHULMAN, *Optimal randomized algorithms for local sorting and set-maxima*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 45–53.
- [3] R. GRAHAM, F. YAO, AND A. YAO, *Information bounds are weak in the shortest distance problem*, J. ACM, 27 (1980), pp. 428–444.
- [4] C. KENYON AND V. KING, *Verifying partial orders*, in Proceedings of the 21st ACM Symposium on Theory of Computing, 1989, pp. 367–374.
- [5] J. KOMLÓS, *Linear verification for spanning trees*, Combinatorica, 5 (1985), pp. 57–65.
- [6] N. LINIAL, *Legal coloring of graphs*, Combinatorica, 6 (1986), pp. 49–54.
- [7] U. MANBER AND M. TOMPA, *The effect of number of Hamiltonian paths on the complexity of a vertex-coloring problem*, SIAM J. Comput., 13 (1984), pp. 109–115.
- [8] R. TARJAN. Private communication.
- [9] A. YAO. Private communication.