JOURNAL OF COMPUTER AND SYSTEM SCIENCES **24**, 197–208 (1982)

The Complexity of Selection and Ranking in $X + Y$ and Matrices with Sorted Columns

GREG N. FREDERICKSON*

AND

DONALD B. JOHNSON†

Computer Science Department, The Pennsylvania State University, University Park, Pennsylvania 16802

Received September 12, 1980; revised December 4, 1981

The complexity of selection is analyzed for two sets, $X + Y$ and matrices with sorted columns. Algorithms are presented that run in time which depends nontrivially on the rank k of the element to be selected and which is sublinear with respect to set cardinality. Identical bounds are also shown for the problem of ranking elements in these sets, and all bounds are shown to be optimal to within a constant multiplicative factor.

INTRODUCTION

We consider the problem of selecting the kth largest element (and the related problem of ranking a value) in sets constrained to possess certain structures. The structures we consider are simple, mathematically interesting, and arise in practical applications. An example is a set that is generated as the Cartesian sum $X + Y$, where X and Y are real vectors. An application in which this set arises is the computation of the Hodges–Lehmann estimator in statistics, for which the median of $X - Y$ must be found. A second example is a set that is presented as a real matrix with sorted columns. An application from operations research in which this set arises is the computation of the optimum distribution of effort for concave functions. Consider m activities with cumulative value functions $f_j(\cdot)$, and n units of effort to be distributed among the activities in integral amounts. If the functions are concave, then the marginal return $f_j(i+1) - f_j(i)$ for any activity is nonincreasing. Thus the marginal returns for all activities may be presented in an $n \times m$ matrix with sorted columns, and the problem may be solved by selecting the nth largest value.

* Work of this author was partially supported by the National Science Foundation under Grant MCS-7909259.

† Work of this author was partially supported by the National Science Foundation under Grants MCS-7721092 and MCS-8002684.

Selection in the sets mentioned is interesting for several reasons. Due to the constraints placed on the sets, selection may be performed in time sublinear in the cardinality of the set. Hence the linear-time selection algorithm of [2] is non-optimal in such circumstances. To support sublinearity of the problems, we make the following observations about inputting the set. For selection in $X+Y$, where we assume without loss of generality that $|X| = n \leqslant |Y| = m$, we make use of the fact that the set may be represented with $n+m$ elements, while the cardinality is nm. For selection in matrices with sorted columns, we assume that either the matrix is already resident in memory, or each value may be computed as needed in constant time (as is the case, for instance, with the problem of optimum distribution of effort).

Unlike selection in unconstrained sets, the complexity of selection in constrained sets is dependent asymptotically on the rank k of the element being selected. (We assume $k \leqslant \lceil nm/2 \rceil$, with $k > \lceil nm/2 \rceil$ being handled symmetrically.) The complexity for both of our problems is $\theta(m + p \log(k/p))$, with $p = \min\{k, m\}$. For example, the complexity is $\theta(m)$ for $k = O(m)$ and $\theta(m \log n)$ for $k = \theta(mn)$. For selection in $X+Y$, previous work has concentrated on finding the median and has thus not identified this fact. For selection in matrices with sorted columns, previous work has focused on the case $k = n$, for which non-optimal algorithms were presented.

Selection is one of the three quantile operations (the other two are ranking and verification) that demonstrate asymptotically identical complexity for the sets considered here. The lower bounds we present for selection and ranking are based on verification: Given a set, a rank k, and a value v, determine whether v can have rank k in the set.

The bounds that we obtain improve substantially on previous results, and furthermore are optimal to within a constant factor for all values of the three parameters k, m, and n. Johnson and Mizoguchi [6] present an $O(m \log m)$ algorithm for selecting the kth largest element in $X+Y$, and Johnson and Kashdan [5] give a lower bound of $\Omega(m + \sqrt{k} \log k)$. These results are optimal only for the limited case where $n = \theta(m)$ and $k = \theta(mn)$. Bounds of $\theta(m \log m)$ for the same limited case appear in Shamos [8]. Our bound of $\theta(m + p \log(k/p))$ is better than the above bounds whenever $n = o(m)$ or $k = o(mn)$. For the problem of optimum distribution of effort, previous upper bounds are $O(m(\log n)^2)$ in [4] and $O(m + n \log m)$ (suggested by the result in [3]). Our bounds of $\theta(m + p \log(k/p))$ for selection in matrices with sorted columns yield $\theta(\max\{m, m \log(n/m)\})$ for this problem, where $k = n$. The $O(m + n \log m)$ bound can match our bound only when n is $O(m/\log m)$.

Selection in Matrices with Sorted Columns

In this section, we present an algorithm for finding a kth smallest element in a given $n \times m$ matrix X with columns sorted in nondecreasing order. Our algorithm SELECT (Fig. 1) consists of three phases. In the first phase, the number of elements is reduced to $O(k)$ by a procedure CUT. In the case in which k is $o(mn)$, this procedure is able to rule out a large number of elements in a small amount, $O(m)$, of

```
proc SELECT ($X, k$)
    (1) It is given that $k \leqslant \lceil mn/2 \rceil$. Apply CUT to $X$, after which the additional
        conditions $N \leqslant 9k/2$ and $m \leqslant k$ hold.
    (2) While $N > m + 1/\alpha^2$ apply REDUCE to $(X, k)$ yielding a new problem $(X, k)$.
    (3) Find a $k$th element in $X$ using linear time selection.
```

FIG 1. Algorithm to select a kth element in matrix X with m sorted columns of length n. N is the number of elements that remain.

time. In the second phase, the number of elements is reduced to $O(m)$ by a procedure REDUCE. This procedure is able to rule out a constant percentage of remaining elements on each of $O(\log(k/m))$ iterations, each costing $O(m)$ time. In the third phase, the appropriate element is selected from the remaining $O(m)$ elements, using a linear-time selection algorithm.

The major result of this section concerns the rapid elimination of all but $O(k)$ elements, as performed by procedure CUT. As a result of this step we can derive an upper bound that exhibits nontrivial dependence of the running time on k. The remaining phases of our algorithm are a careful development of the ideas of Jefferson, Shamos and Tarjan in Shamos [8].

We assume k is no larger than $\lceil nm/2 \rceil$. Problems with larger k can be solved by selecting the $(nm - k + 1)$th element in $-X$ (indexed so columns are nondecreasing). We further assume that $m \geqslant 2$ (otherwise, the solution is trivial) and that matrix X is supplemented with two m-vectors, *first* and *last*. Initially, *first*$(j) = 1$ and *last*$(j) = n$ for each column j. The number of column elements n_j may be computed as *last*(j) − *first*$(j) + 1$. During the execution of the algorithm, *first* and *last* values are adjusted when elements are discarded. At any point, the number of elements that remain is $N = \sum_j n_j$.

Our algorithm uses two subroutines that are based on selection algorithms. By a partition (I_1, i^*, I_2) about an mth element in Y with index set I, we mean the following: Select an mth element y_{i^*}, using a linear-time selection algorithm. Then partition the index set I into I_1, $\{i^*\}$, and I_2, so that $|I_1| = m - 1$, $j \in I_1$ implies $y_j \leqslant y_{i^*}$, and $j \in I_2$ implies $y_j \geqslant y_{i^*}$.

We use linear-time weighted selection [6] similarly. By a partition (Q_1, i^*, Q_2) of index set Q consistent with the weighted selection of an Mth weighted element in Y, with element y_j having weight w_j, we mean: Partition Q into Q_1, $\{i^*\}$, and Q_2 so that $\sum_{j \in Q_1} w_j < M \leqslant w_{i^*} + \sum_{j \in Q_1} w_j$, $j \in Q_1$ implies $y_j \leqslant y_{i^*}$, and $j \in Q_2$ implies $y_j \geqslant y_{i^*}$.

The first step of SELECT is to apply the procedure CUT (Fig. 2) to the given problem. There are three steps. First, the case in which $k < m$ is handled by ruling out all but k columns. Second, the number of elements is reduced to $O(k \log k)$ by a sequence of selections that allow portions of columns to be discarded in groups. Third, the number of elements is reduced to $O(k)$ by a weighted selection on representatives from segments of the remaining columns.

A further discussion of step (1) of CUT is unnecessary. Step (2) of CUT transforms X by operating on the vectors *first* and *last* so that N, the number of

```
proc CUT $(X, k)$
(1) If $k < m$, select a $k$th smallest element $x_{1j^*}$ in the first row of $X$. Find a partition
    $(I_1, j^*, I_2)$ of column indices consistent with this selection, and reindex the
    columns so that $I_2$ maps to $\{k+1,..., m\}$. Set $m$ to $k$.
(2) If $mn > 9k/2$:
    (2.1) Compute $(i_l, j_l)$, where
          $j_l = \lfloor m/2^l \rfloor + 1$ for $l = 0,..., q+1$,
          $i_l = \lceil (k+1)/j_l \rceil$ for $l = 1,..., q$,
          $i_0 = 1$, $i_{q+1} = n+1$,
          where $q = \max\{l \mid j_l < j_{l-1}$ and $i_l \leqslant n\}$.
    (2.2) For $l = 1,..., q$,
          Select a $j_l$th element $v_l$ from the first $j_{l-1} - 1$ elements of row $i_l$. Reindex
          columns 1 through $j_{l-1} - 1$ corresponding to a partition consistent with
          this selection. Set $last(j) = i_l - 1$ for $j = j_l,..., j_{l-1} - 1$.
(3) If $N > 9k/2$:
    (3.1) Let $Q = \{(i,j) \mid i = i_l$ and $j = 1,..., j_l$ for $l = 0,..., q\}$.
          Let $(i_l, j) \in Q$ have weight $i_{l+1} - i_l$.
    (3.2) Find a partition $(Q_1, (i^*, j^*), Q_2)$ of $Q$ consistent with a weighted
          selection of a $4k$th element of $\{x_{ij} \mid (i,j) \in Q\}$.
    (3.3) For all $j$, reset $last(j)$ to $\max\{\{0\} \cup \{i_{l+1} - 1 \mid (i_l, j) \in Q - Q_2\}\}$.
```

FIG. 2. CUT removes elements quickly from any problem where k is far from the median.

elements that remain, is $O(k \log k)$. To do this, indices (i_l, j_l) are identified for which j_l is approximately $m/2^l$ and $i_l j_l > k$. Then, in order of increasing l, columns are reindexed so that the first j_l elements of row i_l are no greater that any of the next j_l elements in row i_l. Then, as allowed by Lemma 1, all elements x_{ij} with $i \geqslant i_l$ and $j \geqslant j_l$, for every $l = 0,..., q$, are discarded.

LEMMA 1. *At least k elements remain in X following step* (2) *in* CUT, *and any* kth *element among these solves the original problem.*

Proof. It may be verified by induction that the condition imposed by step (2.2) on the order of the first $j_{l-1} - 1$ elements in row i_l is preserved under rearrangements made when l takes on larger values. Thus, on completion of step (2.2), the stated conditions hold simultaneously for all $l = 1,..., q$. Each discarding step retains at least k elements. Therefore, $N \geqslant k$ when step (2.2) is finished. To complete the proof it will suffice to show that every discarded element x_{ij} is no smaller than the kth value in X after step (2.2). But this follows immediately from the fact that there exists an l, $1 \leqslant l \leqslant q$, for which $i_l \leqslant i$ and $j_l \leqslant j < j_{l-1}$. Thus, x_{ij} is no smaller than $i_l j_l - 1 \geqslant k$ elements retained. ∎

Step (3) of CUT reduces the number of elements to no more than $9k/2$. It selects a $4k$th weighted element in the set of all elements on which rearrangements were performed in step (2). These elements are taken to represent the column segments running from i_l to $i_{l+1} - 1$ in each column j in which $x_{i_l j}$ is retained, and their weights in the weighted selection are, therefore, $i_{l+1} - i_l$. Column segments with indices (i,j),

$i \geqslant i_l$, are discarded whenever the weighted selection discards an element with index (i_l, j) because the weight of $4k$ has been exceeded. It is necessary to break ties so as to retain values with smallest i indices for a given j.

LEMMA 2. *Let the columns be rearranged and their lengths be n_j as computed in* CUT. *No more than $9k/2$ elements remain, and any* kth *element among these is a solution to the original problem.*

Proof. If step (3) is not performed, Lemma 1 gives the result. Otherwise, let $x^* = x_{i^*j^*}$, the last weighted element retained in step (3.2), and call any element x_{ij} *large* if $x_{ij} > x^*$. We proceed to show that no element less than x^* is discarded and fewer than $(\beta/2+1)k$ large elements are retained, where βk is the actual weight retained in step (3.2).

A column retaining a large element with row index $i < i_1$ retains at most a total of $(i_1 - i_0 - 1) \leqslant (\lceil (k+1)/(\lfloor m/2 \rfloor + 1) \rceil - 2) < 2k/m$ elements. With m columns there are at most $z < 2k$ large elements retained in this way. A column retaining a large element with row index $i > i_l$, for $l > 0$, will have $i_{l+1} - 1$ elements of which at least i_l will not be large. Hence, fewer than half of the elements in such a column will be large. Thus, an upper bound on the total number of large elements retained will be $z + (\beta k - z)/2 < (\beta/2 + 1)k$. Step (3.2) requires $\beta k \geqslant 4k$. Thus $\beta k - (\beta/2+1)k \geqslant k$ elements no larger than x^* remain including all given elements less than x^*. The fact that no n_j exceeds k means the weight of x^* is less than $k/2$ and, therefore, $\beta k \leqslant 9k/2$. ∎

This completes the discussion of CUT.

Step (2) of SELECT employs a procedure REDUCE, shown in Fig. 3. On each iteration, REDUCE returns a problem in which a constant fraction of the current

```
proc REDUCE $(X, k)$
    (1) If $k \geqslant N/2$:
        (1.1) For each column $j$, find $i_j = first(j) + \lfloor a n_j \rfloor$.
              Let element $x_{i_j j}$ have weight $n_j$.
        (1.2) Find a partition $(Q_1, j^*, Q_2)$ of $Q = \{1,\dots,m\}$ consistent with a weighted
              selection of the $(aN)$th element of $\{x_{i_j j} \mid j \in Q\}$.
        (1.3) For each $j$ in $Q_1 \cup \{j^*\}$ set $first(j) = i_j + 1$. Reset $n_j$, $k$, and $N$, accordingly.
    (2) Otherwise:
        (2.1) For each column $j$, find $i_j = last(j) - \lfloor a n_j \rfloor$.
              Let element $x_{i_j j}$ have weight $n_j$.
        (2.2) Find a partition $(Q_1, j^*, Q_2)$ of $Q = \{1,\dots,m\}$ consistent with a weighted
              selection of the $(1-a)N$th element of $\{x_{i_j j} \mid j \in Q\}$.
        (2.3) For each $j$ in $Q_2 \cup \{j^*\}$, set $last(j) = i_j - 1$.
              Reset $n_j$ and $N$ accordingly.
```

FIG. 3. REDUCE discards a constant fraction of elements in time proportional to m.

number of elements, $N = \sum n_j$, has been removed either from the beginning or the end of selected columns. The removal of elements is done in such a way as to preserve the remaining problem as a selection problem, a solution to which solves the given problem. When k selects an element no smaller than the median of the current problem, elements are discarded from the beginning of certain current columns, and k is adjusted accordingly. First, the portion of each column j between *first*(j) and *last*(j) is partitioned at an element, roughly α of the way down, where α may be chosen as less than or equal to $1 - \sqrt{\frac{1}{2} + 1/N}$. Then, using linear-time weighted selection, a partition of these column representatives is found consistent with a weighted selection of an (αN)th representative. The beginning segments in each of the columns in the first block of the partition induced by the weighted selection are discarded by adjusting appropriate values in the vector *first*.

LEMMA 3. *Let $N = \sum n_j$ and $\alpha = 1 - \sqrt{\frac{1}{2} + 1/N}$.* REDUCE *discards at least $\alpha^2 N$ elements, yielding a problem (X, k), the solution of which is a solution to the original problem.*

Proof. We consider the case for which $k \geqslant N/2$. The number of elements rejected is

$$\lfloor \alpha n_{j^*} \rfloor + 1 + \sum_{j \in Q_1} (\lfloor \alpha n_j \rfloor + 1) > \alpha n_{j^*} + \alpha \sum_{j \in Q} \alpha n_j \geqslant \alpha^2 N.$$

All elements discarded are no larger than $v^* = x_{i_{j^*} j^*}$. The number of elements retained that are known to be no smaller than v^* is at least

$$\begin{aligned} &\lceil (1-\alpha) n_{j^*} \rceil - 1 + \sum_{j \in Q_2} \lceil (1-\alpha) n_j \rceil \\ &\quad \geqslant (1-\alpha) n_{j^*} - 1 + \sum_{j \in Q_2} (1-\alpha) n_j \\ &\quad \geqslant (1-\alpha)^2 N - 1 = (\tfrac{1}{2} + 1/N) N - 1 = N/2. \end{aligned}$$

Hence any kth element in the elements retained (where k is adjusted to take into account the number of small elements discarded over all previous iterations) is a solution to the original problem.

When $k < N/2$, similar arguments yield the result. ∎

We may now state the main result.

THEOREM 1. *Let $p = \min\{k, m\}$. Algorithm* SELECT *finds a kth smallest element in an $n \times m$ matrix with sorted columns in $O(m + p \log(k/p))$ time.*

Proof. Lemmas 1–3, together with the evident correctness of step (3) of SELECT give the correctness argument. In CUT, step (1) clearly requires at most $O(m)$ time. Step (2) requires $O(p + p/2 + \cdots)$ which is $O(p)$ time, as does step (3), using a weighted selection on a subset of the union of the sets involved in step (2). REDUCE also runs in $O(p)$ time, since it involves one weighted selection on p elements. Since $N > 1/a^2$ at every iteration, we may conclude from Lemma 3 that a satisfies

$$1 - \sqrt{\tfrac{1}{2} + a^2} < a < 1 - \sqrt{\tfrac{1}{2}},$$

from which it may be determined that

$$\tfrac{1}{4} < a < 1 - \sqrt{2}/2.$$

Thus, REDUCE will be called in $O(\log k/p)$ iterations of step (2) of SELECT. Step (3) of SELECT runs in $O(p)$ time. The claimed result follows. ▮

SELECTION IN $X + Y$

Previous solutions to selection in $X + Y$ [6, 8] have required that at least X be sorted, which yields a problem with sorted columns to which SELECT can be applied. While sorting X does not increase the asymptotic complexity when selecting for the median, it is too time-consuming when k is small. We thus resort to only "semi-sorting" X to an extent allowed by the value of k. We then apply essentially the same algorithm as in the preceding section. Our analysis will be of interest, however, since we must show that our semi-sorting is sufficiently fine to give well-positioned representatives of columns in the REDUCE phase.

Our semi-sort of X is generated in two steps. When $k < n$, we assume that only the k smallest elements in X are retained. In the first step, an ordering is imposed on X so that $i' < i_l < i''$ implies $x_{i'} \leqslant x_{i_l} \leqslant x_{i''}$, for $l = 0,..., q$ with i_l as defined in procedure CUT. Since CUT uses only the i_l rows, this ordering is sufficient to preserve the correctness of the procedure. This ordering can be performed in $O(\min\{k, n\})$ time, if the partitioning proceeds from $l = q$ to $l = 1$.

In the second step, the ordering is refined so that REDUCE may use it essentially as though X is totally sorted. For REDUCE to operate exactly as before, it would be necessary that the element at position $i = \mathit{first}(j) + \lfloor a n_j \rfloor$ in X be the same value that would be at this position if X were sorted, and that X be partitioned on this value so that $i' < i$ and $i'' \geqslant i$ implies $x_{i'} \leqslant x_{i''}$. We do not achieve this result, in general, precisely at position i, but at a position preceding i but close enough so that the algorithm behaves as if $a \geqslant \frac{1}{4}$ in all cases. The symmetric result is also obtained where REDUCE finds index $i = \mathit{last}(j) = \lfloor a n_j \rfloor$.

Our method is to repeatedly subdivide the intervals in X between indices i_l and $i_{l+1} - 1$, at each subdivision placing a new correctly positioned partition element in the middle of the previously unordered elements of each subinterval. For each pass over X, the potential distance from any index i chosen in REDUCE to the nearest subdivision point, or *useful index*, is cut in half. The number of passes over X is $t/2 + 6$, where t is the number of iterations of REDUCE that are required. After the passes over X are completed, we record the mapping from each $i = 1,..., \min\{k, n\}$ to the nearest useful indices above and below the given i. Accessing a useful index, given $i = \mathit{first}(j) + \lfloor \alpha n_j \rfloor$ or $i = \mathit{last}(j) - \lfloor \alpha n_j \rfloor$ will then take constant time.

As before we choose $\alpha = 1 - \sqrt{\frac{1}{2} + 1/N}$. However, the effect of X being not wholly sorted will be to discard a number of elements from, say, column j which is equal to $\lfloor \alpha' n_j \rfloor + 1$ for some $\alpha' \leqslant \alpha$. In the proof of Theorem 1 we showed that $\alpha \geqslant \frac{1}{4}$ during the entire execution of SELECT. We now show that the analogous result $\alpha' \geqslant \frac{1}{4}$ holds for a sufficiently fine subdivision of X.

LEMMA 4. *Let t iterations of* REDUCE *suffice to complete the execution of* SELECT *for $\alpha' = \frac{1}{4}$. Then $t/2 + 6$ passes over X will subdivide X sufficiently to realize $\alpha' \geqslant \frac{1}{4}$ over the entire execution of* SELECT.

Proof. Let n_j elements remain in column j after t iterations. The initial number of elements n_j^0 in column j, therefore, satisfies $n_j^0 \leqslant f^t(n_j)$, where $f(n) = \sqrt{2}n + \sqrt{2}$.

This gives

$$n_j^0 \leqslant n_j 2^{t/2} + \sum_{i=1}^{t} 2^{i/2} < 2^{t/2}(n_j + 2 + \sqrt{2}).$$

If $n_j < 61$, then

$$t/2 + 6 > \log(64 n_j^0/(n_j + 2 + \sqrt{2})) > \log n_j^0$$

which is a sufficient number of passes for column j to have been sorted. If $n_j \geqslant 61$, then $N \geqslant 61 > 1024/17$. Let δ be the maximum distance between useful elements in a subdivision. A sufficient condition for $\alpha' \geqslant \frac{1}{4}$ is $\lfloor \alpha n_j \rfloor - \lfloor n_j/4 \rfloor \geqslant \delta - 1$, which we show:

$$\begin{aligned}\delta &\leqslant \lceil n_j^0/2^{t/2+6} \rceil < \lceil 2^{t/2}(n_j + 2 + \sqrt{2})/2^{t/2+6} \rceil < \lceil n_j/32 \rceil \\ &= \lceil n_j(\tfrac{3}{4} - \sqrt{\tfrac{1}{2} + 17/1024}) \rceil < \lceil n_j(\tfrac{3}{4} - \sqrt{\tfrac{1}{2} + 1/N}) \rceil \\ &= \lceil n_j(\alpha - \tfrac{1}{4}) \rceil \leqslant \lfloor \alpha n_j \rfloor - \lfloor n_j/4 \rfloor + 1. \quad \blacksquare\end{aligned}$$

THEOREM 2. *Selection in $X + Y$ can be performed in $O(m + p\log(k/p))$ time, where $|X| = n \leqslant m = |Y|$ and $p = \min\{k, m\}$.*

Proof. We use the modified algorithm SELECT as just described. Correctness follows from this discussion and Theorem 1. By Lemma 4, the passes to semi-sort X cost time proportional to $(t/2 + 6)\min\{k, n\}$ which is $O(m + p\log(k/p))$ since $n \leqslant m$. Lemma 4 ensures that the complexity arguments of Theorem 1 apply here also. $\blacksquare$

Ranking

In this section we present algorithms for finding the rank k of a given value v in both of the structures previously considered. While our result for ranking in $X+Y$ is more interesting, for purposes of exposition we first consider ranking in a matrix with sorted columns.

In the problem of ranking a value v in a matrix with sorted columns, we perform a one-sided binary search [1] on each column to find the insertion position k_j in each column. For $v \geqslant x_{1j}$, the first stage of the one-sided binary search identifies an interval $x_{ij} \leqslant v < x_{2ij}$, where i is a power of 2. The second stage, then, uses ordinary binary search to locate the position of v in this interval. Assume RANK to be the above procedure.

THEOREM 3. *Algorithm* RANK *solves a ranking problem on an* $n \times m$ *matrix with sorted columns in* $O(m + p \log(k/p))$ *time, where* $p = \min\{k, m\}$.

Proof. No more than $O(m)$ time is required to examine every element in the first row. Let $x_{k_j j} \leqslant v < x_{k_j+1,j}$. Then, $O(\log k_j)$ time is sufficient to find k_j in column j, where $k_j > 0$. For such columns, we have

$$\sum_{j=1}^{p} \log k_j = \log \sum_{j=1}^{p} k_j \leqslant \log(k/p)^p = p \log(k/p).$$

Correctness is immediate. ∎

One way to rank in $X+Y$ is to sort X and use the algorithm RANK. When the rank k turns out to be sufficiently small, however, this method is not optimal. Hence, we employ an approach that relies on a semi-sorting of X similar to that used in selection in $X+Y$. As in the ranking algorithm for a matrix with sorted columns, we use the one-sided binary search on each column. However, to efficiently produce a semi-sorting on X, we need a good estimate k' of k, and to efficiently generate a good estimate k' we need some ordering on X. Hence, we interleave the task of producing a semi-sorting with the task of determining k.

Our ranking algorithm for $X+Y$ shown in Fig. 4, consists of four steps. The initial part of the semi-sorting of X is done in step (1). First, the $2^{\lfloor \log n \rfloor}$th element is selected and X is reordered around this element. Then the first interval in X is repeatedly partitioned around its median in order to guarantee that $x_j \leqslant x_l \leqslant x_h$ whenever $j < l < h$ and $l = 2^i$, for $i = 1, \ldots, \lfloor \log n \rfloor$.

Step (2) generates an estimate k' of k as follows: In each column, if $x_1 + y_j \geqslant v$, the first part of a one-sided binary search for value $v - y_j$ is performed, using the subdivision elements x_l, where $l = 2^i$ for integers i. The resulting insertion position k_j' approximates the actual position k_j by $k_j < k_j' \leqslant 2k_j$ when $k_j > 0$ and by $k_j' = 1$ when $k_j = 0$. The sum $k' = \sum_{j=1}^{m} k_j'$ has the property that $k < k' \leqslant 2k + m$.

Step (3) completes the task of producing a semi-sorting on X, proceeding as in the algorithm for selection in $X+Y$ and using the estimated rank k'. The intervals found

```
proc RANKXY (X, Y, v)
(1) For $i = \lfloor \log n \rfloor, \ldots, 1$, partition and reorder $\{x_j \mid j = 1, \ldots, \min\{2^{i+1}, n\}\}$ consistent
    with the selection of a $2^i$th element.
(2) For each column $j$ of $X + Y$, set $k_j' = 1$. While $k_j' < n$ and $x_{k_j'} + y_i \leqslant v$, reset
    $k_j' = 2k_j'$.
    Set the estimate $k' = \sum_{j=1}^{m} k_j'$.
(3) Let $\{(0, 1), (1, 2), (2, 4), \ldots, (2^{\lfloor \log n \rfloor}, n)\}$ be the set $L$ of intervals. For each of
    $\lfloor \log(k'/m) \rfloor$ passes, for each interval $(a, b)$ in $L$ with $b > a + 1$, reorder $\{x_i \mid i =
    a, \ldots, b\}$ consistent with the selection of a median, and replace $(a, b)$ in $L$ with
    the two new intervals.
(4) For each column $j$, binary search interval $(\lfloor k_j'/2 \rfloor, k_j')$ using the endpoints of
    intervals in $L$, and then exhaustively search the remaining interval to find $k_j$.
    Set $k = \sum_{j=1}^{m} k_j$.
```

FIG. 4. RANKXY ranks a value v in $X + Y$.

in step (1) are repeatedly subdivided around the medians in $\log(k'/m)$ separate passes.

Step (4) completes the search for v in each column. It first performs a binary search over rows $\lfloor k_j'/2 \rfloor$ through k_j', using the subdivision points computed in step (3). The search in the column is completed by partitioning the final unordered interval on v.

THEOREM 4. *Algorithm* RANKXY *finds the rank in $X + Y$ of a given value v in $O(m + p \log(k/p))$ time, where $|X| = n \leqslant m = |Y|$ and $p = \min\{k, m\}$.*

Proof. Correctness follows from the above discussion and previous results. We analyze running time as follows: Step (1) uses $O(n)$ (and therefore $O(m)$) time to partition X. Step (2) requires $O(1 + \log k_j')$ time for each column j, which by the proof of Theorem 3 can be seen to be $O(m + p \log(k/p))$ overall. Step (3) may be seen to cost $O(n \log(k/p))$. The binary search of step (4) can be seen to be $O(m + p \log(k/p))$ by an argument similar to that in the proof of Theorem 3. For each column, the final partitioning can examine at most $k_j'/(k'/m)$ elements. Summing over all j yields $O(m)$ work. ∎

OPTIMALITY

The algorithms we have presented for selection and ranking in $X + Y$ and in matrices with sorted columns, as well as the other algorithms cited [3, 4, 6, 8], belong to the class of algorithms in which the comparisons performed are always between linear combinations of the inputs. In this section we establish a single lower bound on the number of comparisons needed for each of these problems. Our approach will be to show a lower bound for the related verification problem in which an $n \times m$ problem matrix with sorted columns, a rank k, and a trial value t are tested to determine if t is a kth element in the given matrix. If verification on an array with sorted columns requires $T(n, m, k)$ comparisons in the worst case, then any algorithm which produces either k or t from an $n \times m$ matrix (with or without sorted columns) and the value t or k, respectively, will expend at least $T(n, m, k) - 1$ comparisons in the worst case.

THEOREM 5. *Any algorithm which makes comparisons only between linear combinations of its inputs will require at least* $\max\{m-1, \lfloor (m-2)/2 \rfloor \log(\lceil (2k-1)/m \rceil - 1)\}$ *comparisons in the worst case to verify that a number t is the kth smallest in an $n \times m$ rational matrix with m sorted columns, whenever* $1 \leqslant k \leqslant \lceil mn/2 \rceil$.

Proof. Let the given problem matrix be represented by a vector $\mathbf{x} = (x_{11},..., x_{n1}, x_{12},..., x_{n2},..., x_{1m},..., x_{nm})$ in Euclidean nm space. We will prove the theorem subject to the following restrictions: all elements of $\mathbf{x}$ are distinct, and the trial value t is given as an element $x_{i^*j^*}$ of $\mathbf{x}$ restricted to satisfy $i^* \leqslant n_0 = \lceil (2k-1)/m \rceil$. Any input vector with distinct elements will be called *valid.* Since the theorem requires that $n \geqslant n_0$ it is clear that i^* may range up to n_0. Any lower bound obtained under these restrictions will also be a lower bound for the more general verification problem of the theorem.

The proof is in two parts. First we show that all possible inputs can be partitioned into at least $(n_0 - 1)^{\lfloor (m-2)/2 \rfloor}$ equivalence classes. Then we show that there is at least one leaf for each equivalence class in any decision tree which solves the verification problem.

Consider column j of the given input. If $j \neq j^*$, then there is an index i_j, $0 \leqslant i_j \leqslant n$, which defines the point after which $t = x_{i^*j^*}$ can be inserted in sorted column j. Otherwise, let $i_{j^*} = i^*$. Call the vector $\mathbf{i} = (i_1,..., i_m)$ so induced by an input, the *configuration* of the input.

Any vector $\mathbf{i}$ which satisfies $\sum_{i_j \in \mathbf{i}} i_j = k$ corresponds to some possible input. If we assume without loss of generality that $j^* > \lfloor (m-2)/2 \rfloor$, then it is clear that the set of all configurations contains the set $\{(i_1,..., i_m) \mid 0 \leqslant i_j \leqslant n_0 - 1 \text{ for } 1 \leqslant j \leqslant \lfloor (m-2)/2 \rfloor, \sum_{j=1}^{m} i_j = k\}$ which is of cardinality at least $(n_0 - 1)^{\lfloor (m-2)/2 \rfloor}$. Therefore, there are at least this many configurations.

Let $\mathscr{T}(n, m, k, i^*, j^*)$ be an optimal decision tree which decides if $x_{i^*j^*}$ is kth smallest in an $n \times m$ matrix with sorted columns. Furthermore, let $\mathscr{T}$ make comparisons between linear combinations of the inputs and let the outcomes of these comparisons select one of three subtrees depending on whether the outcome is $<$, $=$, or $>$. Let any leaf reached over strict outcomes (i.e., $=$ does not hold) be called *strict.* By the density of the rational numbers and the finiteness of the tree, for each configuration there exists at least one strict leaf which is chosen by some input with that configuration. Assume that two inputs $\mathbf{x}'$ and $\mathbf{x}''$ have distinct configurations but choose the same strict leaf. Any convex combination of $\mathbf{x}'$ and $\mathbf{x}''$ also chooses the same leaf since any strict leaf defines a convex and open subset of R^{nm}. Not all of these convex combinations are valid inputs, of course, but because $\mathbf{x}'$ and $\mathbf{x}''$ have distinct configurations there is an index pair (i,j) with the properties $x'_{ij} < x'_{i^*j^*}$ and $x''_{ij} > x''_{i^*j^*}$. Therefore, there exists a vector $\mathbf{y}$, a convex combination of $\mathbf{x}'$ and $\mathbf{x}''$, for which $y_{ij} = y_{i^*j^*}$. Since $\mathbf{y}$ selects the same leaf as $\mathbf{x}'$ and $\mathbf{x}''$, for a sufficiently small $\varepsilon > 0$ every vector in an ε-neighborhood of $\mathbf{y}$ selects the same leaf as well. Among these, let $\mathbf{y}'$ and $\mathbf{y}''$ be valid inputs that are equal except for $y'_{ij} > y'_{i^*j^*} = y''_{i^*j^*} > y''_{ij}$. If by the correctness of $\mathscr{T}(n, m, k, i^*, j^*)$ the configuration of $\mathbf{y}'$ sums to k, then the configuration of $\mathbf{y}''$ sums to $k+1$. The contradiction thus induced establishes that any correct, finite, ternary, linear decision tree must have at least one strict leaf for each configuration, and, therefore, must perform at least $\log((n_0 - 1)^{\lfloor (m-2)/2 \rfloor}) = \lfloor (m-2)/2 \rfloor \log(\lceil (2k-1)/m \rceil - 1)$ comparisons on some input.

Since no information is given relating one column to another, it follows from the results of Rabin [7] that $m - 1$ comparisons are required in any case. ∎

When $k \geqslant m$ the bound in Theorem 5 is $\Omega(m + p \log(k/p))$, as it is when $k < m$, since $p \log(k/p) = 0$ in this case. Thus Theorem 5 establishes that our algorithms for selection and ranking in matrices with sorted columns are optimal to within a constant factor. In the case of $X + Y$, a simple construction can provide a valid input with X in sorted order for any configuration. A refinement of such a construction supports the existence of $\mathbf{y}'$ and $\mathbf{y}''$ in the proof of Theorem 5. Thus, Theorem 5 also establishes optimality of our algorithms for $X + Y$.

References

1. J. L. Bentley and A. C. Yao, An almost optimal algorithm for unbounded searching, *Inform. Process. Lett.* **5** (1976), 82–87.
2. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, Time bounds for selection, *J. Comput. System. Sci.* **7** (1972), 448–461.
3. B. L. Fox, Discrete optimization via marginal analysis, *Management Sci.* **13** (1966), 210–216.
4. Z. Galil and N. Megiddo, A fast selection algorithm and the problem of optimum distribution of effort, *J. Assoc. Comput. Mach.* **26** (1979), 58–64.
5. D. B. Johnson and S. D. Kashdan, Lower bounds for selection in $X + Y$ and other multisets, *J. Assoc. Comput. Mach.* **25** (1978), 556–570.
6. D. B. Johnson and T. Mizoguchi, Selecting the Kth element in $X + Y$ and $X_1 + X_2 + \cdots + X_m$, *SIAM J. Comput.* **7** (1978), 147–153.
7. M. O. Rabin, Proving simultaneous positivity of linear forms, *J. Comput. System. Sci.* **6** (1972), 639–650.
8. M. I. Shamos, Geometry and statistics: Problems at the interface, *in* "Algorithms and Complexity: New Directions and Recent Results" (J. F. Traub, Ed.), pp. 251–280, Academic Press, New York, 1976.