Lower bounds for comparison based algorithms for selection of maximum, minimum, second smallest element, the median and sorting

Jørgen Bang-Jensen
Imada, SDU

9. maj 2019

1 Introduction

The purpose of this note is to prove lower bounds for the number of comparisons of numbers which every algorithm must make in order to determine both the maximum and the minimum element, respectively the second smallest element, respectively the median. Such lower bounds are already discussed in the copy of Chapter 3 from the book the book by Baase. Here the lower bounds are established by showing how an adversary may generate bad input for any algorithm by following certain rules (which among other things involves assignment of values to the elements and possibly changing these values while the data set is still under construction).

One weakness of this way of formulating the adversary's strategy is that it may be confusing that the values of the elements are changing during the construction. Hence, even though it is intuitively clear that the method works, one is left with a feeling of not really having seen a real proof for the lower bound. In these notes we do obtain precise proofs of the lower bounds by formulating the adversary's strategy as that of forming an acyclic partial orientation of a complete graph. The orientation is constructed concurrently with the execution of the algorithm for which the lower bound is been established (that is, one arc is oriented for each comparison that is made by the algorithm) and no orientation will ever be changed after it is assigned to an edge. Only at the very end of the process the actual values of the elements

are decided, based on the final acyclic partial orientation when the algorithm terminates. As we shall show below, using the orientation at that time, the adversary can easily assign values to the elements in such a way that all comparisons made by the algorithm receive a valid answer. In fact, the adversary can always use the numbers $\{1, 2, \ldots, n\}$ when producing a bad input of size n. Thus we can obtain a lower bound for the number of comparisons which must be made by the algorithm, provided that we can argue that this many arcs must always be oriented for a unique answer to exist.

2 Some terminology and notation

We denote a digraph by $D = (V, A)$ where V is the set of vertices and A the set of arcs. We may also use the notation $V(D)$ for the set of vertices and $A(D)$ for the set of arcs of D. An arc from a vertex u to a vertex v is denoted $u \to v$. We say that the arc **starts** in u and **ends** in v. For every vertex $x \in V(D)$ we denote by $d_D^+(x)$ ($d_D^-(x)$) the number of arcs which start (end) in x. We also use $d_D(x)$ for the total number of arcs in D which are incident with the vertex x. Thus $d_D(x) = d_D^+(x) + d_D^-(x)$. The **complete graph** K_n is the unique graph (up to isomorphism) on n vertices in which every par of distinct vertices is joined by an edge.

An **out-arborescence** is an orientation of a tree such that every vertex except one (called the **root**) has precisely one arc ending in it. No arc ends in the root and it is easy to show that if r is the root in an out-arborescence T, then r can reach every other vertex $x \in V(T)$ by a directed path (Exercise 2).

A digraph is **acyclic** if it has no directed cycle. If a digraph D on n vertices is acyclic, then we may label its vertices $v_1, v_2, \ldots, v_n$ such that there is no arc of the kind $v_j \to v_i$ where $i < j$ (Exercise 1). Clearly the opposite is also true: if D has an acyclic ordering, then D must be acyclic (all arcs go forward with respect to the ordering).

A **tournament** is an orientation of a complete graph. It is easy to show that up to isomorphism there is precisely one acyclic tournament on n vertices (Exercise 3). We denote this by TT_n and also call it the **transitive tournament** on n vertices, because it has the property that if $x \to y$ and $y \to z$ are arcs, then $x \to z$ is also an arc.

3 Sorting and acyclic partial orientations of K_n

Given n distinct real numbers $x_1, x_2, \ldots, x_n$ we can define a transitive tournament $TT_n = (V, A)$ where $V = \{x_1, x_2, \ldots, x_n\}$ and $A = \{x_i \to x_j : x_i < x_j\}$. Thus we translate the property that $x_i < x_j$ to an arc from x_i to x_j in TT_n. The tournament TT_n is acyclic and has the property that if $x_i \to x_j$ and $x_j \to x_k$ then we also have that $x_i \to x_k$. In other words, the relation $R = \{(x_i, x_j) : x_i \to x_j \in A\}$ is transitive. The proof of the following small observation is left to the reader as Exercise 4.

Lemma 3.1 *Let $M = (V, A \cup E)$ be obtained from the complete graph K_n by orienting a subset A of the edges of K_n (the set E denotes those edges which still have no orientation assigned to them). Let $D = (V, A)$ be the digraph consisting of all the vertices of V and the arcs A. Then one can orient the edges in E such that the resulting tournament $T = (V, A \cup A')$ is acyclic (that is, it is isomorphic to TT_n) if and only if D is acyclic (here A' denotes the set of arcs we obtain by orienting the edges of E).*

From the discussion above it follows that given any acyclic digraph $D = (V, A)$ with n vertices $V = \{v_1, v_2, \ldots, v_n\}$ we can assign a set S of n distinct real numbers $x_1, x_2, \ldots, x_n$ (one for each vertex), with the property that $x_i > x_j$ if and only if there is no directed path from v_i to v_j in D. To accomplish this, simply start from an acyclic ordering $\mathcal{L}$ of $V(D)$ and let the numbers in S be arbitrary distinct numbers which satisfy that $x_i > x_j$ if and only if v_i is after v_j in $\mathcal{L}$. In particular, we may always chose S to be the set $\{1, 2, \ldots, n\}$ (by associating the number i with the i'th vertex in $\mathcal{L}$).

Thus we have shown that a partial orientation $M = (V, A \cup E)$ of the complete graph K_n corresponds to one or more permutations of the numbers $1, 2, \ldots, n$ if and only if the oriented part $D = (V, A)$ is acyclic. It is also clear that every permutation of $\{1, 2, \ldots, n\}$ can be realized in this way. We will make use of these facts below by constructing partial orientations of K_n which will always be acyclic and hence will correspond to some suitably chosen permutation of $\{1, 2, \ldots, n\}$.

Our arguments below concerning partial orientations D of K_n can be translated to concrete inputs which will force the actual algorithm to perform the postulated number of comparisons, simply by replacing the vertices of D by the numbers $1, 2, \ldots, n$ in such a way that the number x_k (the k'th

number of the input) is assigned the value i precisely if the vertex $v_k \in V$ is assigned the number (place) i in the current acyclic ordering $\mathcal{L}$ of D.

We denote by $Q(x, y)$ the query which asks for a comparison of x and y by the algorithm. We shall always "answer" such a query by assigning an orientation to the edge xy of K_n. Since no optimal algorithm would compare the same numbers twice, we may assume that x and y have not been compared earlier by the algorithm, nor will they be so at a later stage of the algorithm.

We will always use D to denote the actual oriented graph with n vertices and those edges which have so far been oriented (that is in the beginning when no edge has been oriented we have $D = (V, \emptyset)$) and we will let $\mathcal{L}(D)$ denote the current acyclic ordering of D. Furthermore, we denote by $\mathcal{L}_D(x)$ the position of the vertex x in $\mathcal{L}(D)$ (that is, $\mathcal{L}_D(x) = i$ if and only if x is the i'th vertex in $\mathcal{L}(D)$). The adversary will update $\mathcal{L}(D)$ as new arcs are oriented so that at any time $\mathcal{L}(D)$ is an acyclic ordering of the current digraph D.

4 Lower bound for finding both the maximum and the minimum element among n numbers

The adversary will build a partial orientation of K_n which is always acyclic by always answering the query $Q(x, y)$ according to the following strategy.

1. If $d_D^+(x) = 0 = d_D^+(y)$, then add the arc $x \to y$ to D if $d_D^-(y) > 0$ and add the arc $y \to x$ to D if $d_D^-(y) = 0$.

2. If $d_D^+(x) = 0 < d_D^+(y)$, then add the arc $y \to x$ to D.

3. If $d_D^+(x) > 0$ and either $d_D^+(y) = 0$, or $d_D^-(x) = 0$, then add the arc $x \to y$ to D.

4. If $d_D^+(x)d_D^-(x) > 0$ and $d_D^+(y) > 0$, but $d_D^-(y) = 0$, then add the arc $y \to x$ to D.

5. If $d_D^+(x)d_D^-(x), d_D^+(y)d_D^-(y) > 0$, the add the arc $x \to y$ to D if $\mathcal{L}_D(x) < \mathcal{L}_D(y)$ and otherwise add the arc $y \to x$ to D.

Lemma 4.1 *If D is acyclic and the edge xy is oriented as above then the resulting digraph is also acyclic.*

Proof: Clearly it is enough to show the the new arc that is added cannot be part of a cycle. In 1. the arc $x \to y$ cannot be part of a cycle as $d_D^+(y) = 0$. The same argument shows that $y \to x$ cannot be part of a cycle in 2. In 3. we either have $d_D^+(y) = 0$, or $d_D^-(x) = 0$, implying that the arc $x \to y$ cannot be part of a cycle. In 4. the arc $y \to x$ cannot be part of a cycle as $d_D^-(y) = 0$. Finally, in 5. we add the arc such that is goes forward with respect to the acyclic ordering of D and hence the new arc cannot be part of a cycle. $\diamond$

Every algorithm for finding both the maximum and the minimum number among n distinct numbers must obtain at least $2n - 2$ units of information. Namely, it has to exclude $n-1$ numbers from being the maximum and $n-1$ numbers from being the minimum among the n numbers. The only step in which the adversary is forced to deliver 2 units of new information is when rule 1. is applied and we have $d_D^-(x) = d_D^-(y) = 0$. When rules 2.-4. are applied at most one unit of new information is given and when rule 5. is applied no new information is given.

Theorem 4.2 *Every algorithm which finds both the maximum and the minimum element among n distinct numbers must perform at least $\lceil 3n/2 \rceil - 2$ comparisons.*

Proof: Let $\mathcal{A}$ be an arbitrary algorithm for the problem. By orienting K_n concurrently with the execution of $\mathcal{A}$ according to the rules above (which tell the adversary which orientation to choose when $\mathcal{A}$ asks the query $Q(x, y)$) the adversary will deliver 2 pieces of new information (that is one of x and y is excluded from being the maximum and the other from being the minimum) at most $\lfloor n/2 \rfloor$ times. Thus in order for $\mathcal{A}$ to collect the required information $\mathcal{A}$ must make at least $2n - 2 - \lfloor n/2 \rfloor = \lceil 3n/2 \rceil - 2$ comparisons. $\diamond$

5 Lower bounds for finding the second smallest element among n distinct numbers

Obviously the problem is precisely as difficult as that of finding the second largest element among n distinct numbers. In the notes from Baase it is

shown that one can solve that problem using $n + \lceil \log n \rceil - 2$ comparisons by using the so-called tournament method (not to be confused with the mathematical concept of a tournament). It can easily be seen that when we use this method for finding the second smallest element, every element which has only been compared with the final minimum element could potentially still be the second smallest. Thus the adversary's strategy will be to ensure that the minimum element is compared with $\lceil \log n \rceil$ different numbers.

The adversary constructs an acyclic partial orientation of K_n, starting from $D = (V, \emptyset)$ by answering the queries of the form $Q(x, y)$ as follows. The adversary maintains a collection of disjoint out-arborescenses $T_1, T_2, \ldots, T_k$. Every vertex x for which we currently have $d^-_D(x) = 0$ is the root of one of these arborescenses (we also denote this arborescence by $T(x)$). At the beginning (when no edges have been oriented) every vertex is the root of its own private out-arborescence which consists only of the vertex itself and has no arcs. In order to distinguish between arcs which are part of an out-arborescence and arcs which are not we partition the arc set of D into two sets: black arcs corresponding to arcs in the arborescences and red arcs corresponding to the rest of the arcs of D (see Figure 1).

1. If $d^-_D(x) = d^-_D(y) = 0$ (that is x and y are both the roots of out-arborescences), then add a black arc $x \to y$ to D if $|V(T(x))| > |V(T(y))|$ and otherwise add a black arc $y \to x$ to D.

2. If $d^-_D(x) = 0 < d^-_D(y)$, then add a red arc $x \to y$ to D.

3. If $d^-_D(y) = 0 < d^-_D(x)$, then add a red arc $y \to x$ to D.

4. If $d^-_D(x), d^-_D(y) > 0$, then add a red arc $x \to y$ to D if there is no directed (y, x)-path in D and otherwise add a red arc $y \to x$ to D.

Lemma 5.1 *If D is acyclic and the edge xy is oriented according to the rules above, then the resulting digraph (obtained by adding the new arc to D) is still acyclic and the black arcs form a forest of out-arborescences.*

Proof: The last claim follows from the fact that a new black arc will always start in the root of a black out-arborescence to the root of another one (implying that the later ceases being the root of an out-arborescence and that the resulting out-arborescence contains all the vertices of the two

Figur 1: A time picture of the oriented graph D which is constructed by the adversary. Dotted arcs correspond to red arcs and full arcs to black arcs. There are currently 5 disjoint out-arborescenses with respectively 8,4,2,1 and 1 vertices.

original out-arborescences). None of the rules 1.-3. can result in a cycle after adding the new arc as the arcs will always start in a vertex z with $d_D^-(z) = 0$. In rule 4. we orient in such a way that D plus the new arc that we add is still acyclic (note that, as D was acyclic before we added the arc, either it has no (x, y)-path or no (y, x)-path).

$\diamond$

Theorem 5.2 *By applying the above strategy the adversary will force any comparison based algorithm $\mathcal{B}$ to use at least $n + \lceil \log n \rceil - 2$ comparisons in order to determine the second smallest element.*

Proof: Let $\mathcal{B}$ be an arbitrary comparison based algorithm for the problem. As long as a vertex x has $d_D^-(x) = 0$, x is a possible candidate for being the minimum element and as soon as x has $d_D^-(x) > 0$, then x can no longer be the minimum (according to the way we assign values to the element of D upon termination of the algorithm). Hence when $\mathcal{B}$ terminates there will be precisely one vertex x with $d_D^-(x) = 0$. It follows from the rules above that a vertex z can only change from having $d_D^-(z) = 0$ to having $d_D^-(z) > 0$ when a black arc is added into the vertex hence it follows (by induction) that the unique vertex x which has $d_D^-(x) = 0$ when $\mathcal{B}$ terminates is precisely the root of the resulting spanning out-arborescence which contains all n vertices of D. All vertices in this out-arborescence which are children of x are candidates for being the second smallest element after making all the comparisons corresponding to the black arcs. The follows from the fact that the only vertex that can reach these (besides themselves) by a directed path using only black arcs is x. Hence such a vertex can only be excluded from ending up at the

second place in the final acyclic ordering by having a red arc into it. Every time a black arc is added, the size of the resulting black out-arborescence is increased by at most a factor two (see rule 1.). Thus x has been the root of at least $\lceil \log n \rceil$ different out-arborescenses during the execution of $\mathcal{B}$ and hence x has at least $\lceil \log n \rceil$ children in the final out-arborescence.

It follows from rules 1.-4. that, at any time during the execution of $\mathcal{B}$, if z has a red arc into it, then z also has a black arc into it. In order to ensure that only one of the children of x can be number two in the final acyclic ordering at least $t-1$ red arcs must be generated where t denotes the number of children of x in the final out-arboresecence (which contains precisely $n-1$ black arcs at termination of $\mathcal{B}$). Thus $\mathcal{B}$ must make at least $(n-1)+(\lceil \log n \rceil - 1) = n + \lceil \log n \rceil - 2$ comparisons in order to determine the second largest element. $\diamond$

6 A lower bound for the number of comparisons needed to find the median of n numbers

For simplicity we assume below that the numbers are all distinct and that n is odd, implying that the median is unique.

Every algorithm for determining the median m among n distinct numbers must determine, for each of the other $n-1$ numbers, whether they are smaller or larger than the median. This must be accomplished either by a direct comparison with the median or indirectly through a chain of comparisons.

Again the adversary will use partial orientations of K_n to force any algorithm to perform many comparisons before the median is found. Observe that if we already know that $x < m < y$, then the information $x < y$ does not help us find the median. We call such a comparison (of x and y) a **useless** comparison. The aim of the adversary is to convey at much useless information as possible while at the same time preserving consistency. This is achieved by maintaining the acyclic orientation D such that as many queries as possible can be answered by useless information while still preserving consistency. Below we describe the adversary's strategy.

At any time we maintain three sets S, L, U which form a partitioning of V. These are defined by

$S = \{x \in V : d_D(x) > 0 \text{ and } \mathcal{L}_D(x) < (n+1)/2\}$,
$L = \{x \in V : d_D(x) > 0 \text{ and } \mathcal{L}_D(x) > (n+1)/2\}$,
$U = V - S - L$.

Note that at the beginning we have $U = V$ as no edges have been oriented yet.

The adversary answers the query $Q(x, y)$ according to the rules below with the additional restriction that we must always have $|S|, |L| \leq (n-1)/2$. If we reach a point where $|S| = (n-1)/2$ and $|L| < (n-1)/2$, then all remaining vertices $z \in U$ except one are placed in L and are moved to the end of $\mathcal{L}(D)$. Similarly if we reach a point where $|L| = (n-1)/2$ and $|S| < (n-1)/2$, then all remaining vertices $z \in U$ except one are placed in S and are moved to the beginning of $\mathcal{L}(D)$. When only one vertex z is left in U the adversary moves this element to position $(n+1)/2$ in the current acyclic ordering. This is the element which will finally become the median (denoted m below). As of this point m will remain on position number $(n+1)/2$ in the acyclic ordering until the algorithm terminates. Note also that the element m is considered undefined until U becomes empty.

In order to be able to distinguish comparisons which are known to be useless from (potentially) useful comparisons we denote the first ones by red arcs and the others by black arcs.

1. If $x, y \in U$, then $S := S + x$, $L := L + y$, add a red arc $x \to y$, move x to position 1 and y to position n in the new acyclic ordering.

2. If $x \in L$ and $y \in U$, then $S := S + y$, add a red arc $y \to x$ move y to position 1 in the new acyclic ordering.

3. If $x \in S$ and $y \in U$, then $L := L + y$, add a red arc $x \to y$ move y to position n in the new acyclic ordering.

4. If $y \in L$ and $x \in U$, then $S := S + x$, add a red arc $x \to y$ move x to position 1 in the new acyclic ordering.

5. If $y \in S$ and $x \in U$, then $L := L + x$, add a red arc $y \to x$ move x to position n in the new acyclic ordering.

6. If $x \in S, y \in L$, then add a red arc $x \to y$.

m

Figur 2: A possible structure of D at the time when the median m has been determined (that is it is the only possible candidate left). The dotted arcs are red arcs and the others are black arcs. To ease visibility only a minimal set of black arcs has been included

7. If $x \in L, y \in S$, then add a red arc $y \rightarrow x$.
8. If $\mathcal{L}_D(x) < \mathcal{L}_D(y)$ and $x, y \in S + m$, or $x, y \in L + m$, then add a black arc $x \rightarrow y$.
9. If $\mathcal{L}_D(y) < \mathcal{L}_D(x)$ and $x, y \in S + m$, or $x, y \in L + m$, then add a black arc $y \rightarrow x$.

Note that in the last two rules we only allow x or y to equal m if we have $U = \emptyset$ at this point.

Lemma 6.1 *If D is acyclic and a new arc between x and y is oriented according to the rules above and added to D then the resulting digraph is acyclic and every red arc $u \rightarrow v$ satisfies $\mathcal{L}_D(u) < (n+1)/2 < \mathcal{L}_D(v)$.*

Proof: Exercise 7. ◇

Figure 2 shows an example of the current digraph D at the time when the median m has been found.

Theorem 6.2 *By applying the above strategy the adversary will force every comparison based algorithm $\mathcal{C}$ to perform at least $3n/2 - 3/2$ comparisons before the median is determined among n distinct numbers.*

Proof: Note that if D is an acyclic digraph with an acyclic ordering $\mathcal{L}(D)$, where $\mathcal{L}_D(x) < \mathcal{L}_D(y)$ and there is no directed path from x to y in D, then $\mathcal{L}(D)$ can be changed into a new acyclic ordering in which y appears before x (Exercise 8). This implies that when $\mathcal{C}$ terminates there must exist paths from x to m and from m to y in D for arbitrary $x, y \in V(D)$ such that $\mathcal{L}_D(x) < (n+1)/2 < \mathcal{L}_D(y)$. No red arc can be on such a path as every red arc $u \to v$ satisfies $\mathcal{L}_D(u) < (n+1)/2 < \mathcal{L}_D(v)$ and all arcs of D go forward with respect to $\mathcal{L}(D)$. This implies that every vertex x with $\mathcal{L}_D(x) < (n+1)/2$ must have at least one black arc out of it and every vertex y with $\mathcal{L}_D(y) > (n+1)/2$ has at least one black arc into it. Thus there are at least $n-1$ black arcs since all arcs $u \to v$ with $\mathcal{L}_D(u) < (n+1)/2 < \mathcal{L}_D(v)$ are red. At the same time at least $(n-1)/2$ red arcs can be added by the adversary since the rules 1.-7. may be applied at least $(n-1)/2$ times (until either $|S| = (n-1)/2$, or $|L| = (n-1)/2$). This shows that in total $\mathcal{C}$ must perform at least $n - 1 + (n-1)/2 = 3n/2 - 3/2$ comparisons. $\diamond$

7 An $\Omega(n \log n)$ lower bound for sorting by comparisons

Simplifying assumption: $n = 2^k$ for some natural number k.

The adversary maintains a full binary tree T of depth $k = \log n$ whose vertices are bags of elements. Initially all but the root bag is empty. The root bag contains the n elements to be sorted.

We assign levels to T so that the root is at level 0 and the leaves at level k. There are precisely 2^ℓ bags (vertices of T) at level ℓ. At any time during the process the subtree $T[b]$ of T rooted at a bag b at level ℓ contains at most $\frac{n}{2^\ell} = 2^{k-\ell}$ elements and there are precisely n elements in T (so many bags are empty).

During the run of the algorithm (where the adversary responds to the queries one by one and in a consistent maner) the elements move down through T until all n elements are in their own private bag of size 1 at level $k-1$. At this point the elements are sorted.

At any point during the exceution of the strategy a bag can be in one of the following states **open, right-flushed** or **left-flushed**. Initially all bags in T are open.

The adversary makes use of the following subroutines to perform his strategy.

- We denote by $MOVE(u)$ the subroutine which whenever it is called after an element u has arrived at a bag b' does the following: If b' is open, u stays in b' but if b' is already marked right-flushed, respectively left-flushed, the element u is moved one step down to root bag b'' of the right subtree, respectively left subtree and $MOVE(u)$ is called again. Hence, at any point during the run of the strategy an element may move down by several levels.

- We denote by $CHECK(b)$ the subroutine which for a given open bag b checks whether either the right or left subtree $T[b]$ of the bag b at level ℓ contains $2^{k-\ell-1}$ elements and if so marks the bag b left-flushed respectively right-flushed and then calls $MOVE$ on all elements in b (they will all end up in the same bag when the recursion stops).

Here is the adversary's strategy: while answering the queries of the algorithm the adversary moves either 2, 1 or zero elements down in the tree with one exception: After each move from a bag b we perform $CHECK(b)$ which could potentially move up to $2^{k-\ell-1}$ elements from a bag b at level ℓ down in $T[b]$. After each query which results in at least one element being moved, the procedure $MOVE$ is called recursively on the moved elements.

Assume the sorting algorithm $\mathcal{A}$ asks for the result of a comparison of u and v. Let $\ell(u), \ell(v)$ be the levels of u, v repectively and let $b(u), b(v)$ be the bags containing u and v respectively.

(1) If the least common ancestor b of $b(u)$ and $b(v)$ in T is distinct from both $b(u)$ and $b(v)$, then answer "$u < v$" if u is in the left subtree of b and "$u > v$" otherwise. No element is moved (this is a **"useless"** comparison for $\mathcal{A}$).

(2) If the least common ancestor b of $b(u), b(v)$ is in $\{b(u), b(v)\}$ then w.l.o.g. $b = b(u)$ (otherwise perform the analogous steps for $b = b(v)$).

(2a) If $b(u) = b(v)$ then answer "$u < v$" and move u into the root of left subtree of $T[b]$ and v into the root of the right subtree of $T[b]$; perform $MOVE(u)$, $MOVE(v)$ and finally $CHECK(b)$.

(2b) If $b(u) \neq b(v)$ and $b(v)$ is in the right subtree of $b(u)$, then answer $u < v$, move u into the root of the left subtree of $T[b(u)]$ and perform $MOVE(u)$ and then $CHECK(b)$.

(2c) If $b(u) \neq b(v)$ and $b(v)$ is in the left subtree of $T[b(u)]$, then answer $u > v$, move u into the root of the right subtree of $T[b(u)]$ and perform $MOVE(u)$ and then $CHECK(b)$.

Theorem 7.1 *By following the stategy above, the adversary can force any comparison based sorting algorithm $\mathcal{A}$ to perform $\Omega(n \log n)$ comparisons.*

Proof:
Recall that initially the root bag at level 0 contains all n elements and at the end (when the permutation has been fixed) all elements reside in the leaf bags at level $k = \log n$. In between these two events every other bag becomes non-empty and then empty again at least once. We shall show that we can associate at least $2^{k-\ell-1}$ comparisons made by the algorithm $\mathcal{A}$ privately to each bag at level ℓ. This means that at least $n/2$ comparisons are associated privately with each of the $\log n$ levels, implying that $\mathcal{A}$ makes $\Omega(n \log n)$ comparisons before the input is sorted.

Let b be any bag at level ℓ and let us show how to associate a set of at least $2^{k-\ell-1}$ private comparisons of $\mathcal{A}$ to b. These will be all comparisons between elements u, v where at least one of the elements belongs to b and the other to $T[b]$ (and thus possibly also to b). This does indeed indentify private comparisons for b so we just need to show that there are enough of them. This follows from the fact that the only comparisons that cause one or more elements to be moved out of b are those with u, v as above and the bag b cannot be flushed until at least $2^{k-\ell-1}$ elements have been moved into either the right or the left subtree of b and this will require at least $2^{k-\ell-1}$ comparisons of the type above. $\diamond$

Corollary 7.2 *The adversary's strategy can be performed in time $O(n \log n)$.*

Proof: We may represent T virtually by denoting the 2^ℓ bags at level ℓ by $b_{\ell,0}, b_{\ell,1}, \ldots, b_{\ell,2^\ell-1}$ and keeping, for each element u two values $\ell(u)$ and $b(u)$, where $0 \leq b(u) \leq 2^{\ell(u)} - 1$. If u is moved down one level into the left (right) subtree of $b(u)$ we just replace $b(u)$ by $2b(u)$ $(2b(u) + 1)$. Hence moving an element down one level take constant time and since an element

will move down precisely $\log n$ levels this takes $O(n \log n)$ in total provided each call to $MOVE$ takes constant time. This can be accomplished by keeping track, for each bag b whether b is open right-flushed or left-flushed. Similarly, $CHECK(b)$ takes constant time besides the work done by calls to $MOVE$ if we just keep two counters $left(b), right(b)$ which keep track of the number of elements in the left, respectively, the right subtree of $T[b]$ and update these (in constant time per update) when we move one element one level down. What remains is how to test whether case (2b) or (2c) applies. This can be done in constant time given $b(u), b(v), \ell(u), \ell(v)$ we leave this to the interested reader.

$\diamond$

8 Exercises

1. Prove that if $D = (V, A)$ is an acyclic digraph on n vertices, then we may order its vertices $v_1, v_2, \ldots, v_n$ so that there is no arc $v_j \rightarrow v_i$ where $i < j$. Hint: Argue that D must have a vertex x with $d^-(x) = 0$ and use induction.

2. Let T be an out-arborescence with root r. Prove that T contains a directed path from r to x for every vertex x of T. Hint: apply induction.

3. An **isomorphism** between two digraphs $D = (V, A)$ and $H = (V', A')$ is a mapping $f : V \rightarrow V'$ which is 1-1 and onto and which preserves arcs, that is, if $u \rightarrow v \in A$ then $f(u) \rightarrow f(v) \in A'$. Prove that any two transitive tournaments on the same number of vertices are isomorphic. Then derive a linear algorithm for finding such an isomorphism given two transitive tournaments on n vertices.

4. Prove Lemma 3.1

5. Give an ordered set of comparisons which an algorithm $\mathcal{B}$ (for finding the second smallest element among n) could perform and which would result in the adversary constructing the digraph in Figure 1.

6. Argue that if $\mathcal{B}$ is the tournament method, described on page 129 in the first part of these notes (Baase) then the adversary will generate precisely $\lceil \log n \rceil - 1$ red arcs.

7. Prove Lemma 6.1.

8. Suppose that D is an acyclic digraph with two vertices x, y such that D contains no path from x to y. Let $\mathcal{L}(D)$ be an acyclic ordering of D such that $\mathcal{L}_D(x) < \mathcal{L}_D(y)$. Prove that there exists another acyclic ordering of D in which y is before x. Hint: it is not always enough just to interchange the positions of x and y.

9. Find an acyclic ordering of the digraph Figure 2. Then describe a sequence of comparisons that will result in the adversary constructing precisely this digraph D.

10. Consider the representation used in the proof of Corollary 7.2 and explain how to test whether case (2b) or (2c) applies in constant time given $b(u), b(v), \ell(u), \ell(v)$

11. Consider the adversary strategy that forces any algorithm for finding the minimum and the second smallest element among n distinct numbers. Show that when the minimum element is determined, the spanning out-arborescence consisting of the black arcs (those that show that the root is the smallest element) has at least $n/2$ leaves.

12. Consider again the adversary strategy that forces any algorithm for finding the minimum and the second smallest element among n distinct numbers. Show by induction over the number of red arcs added so far that when answering a request $Q(x, y)$ for which case 4. applies and y is a leaf in the black out-tree to which it belongs but x is not a leaf in the black out-tree to which it belongs (we consider a vertex which a root of an out-tree with just one vertex to be a leaf of that tree) we can always orient the new red arc so that it enters y.

13. Use the result in Exercise 12 to show that when the algorithm $\mathcal{B}$ for which the adversary is constructing a bad input has determined the minimum element (the root of a spanning black out-tree T is determined), every leaf of T is still a candidate to be the maximum element, except if it has a red arc entering from another leaf in T.