

Adversary Lower Bound Technique

November 4, 2003

One way to think of the adversary lower bound technique is that we are devising a method to construct a worst case run for an unknown algorithm. As with other lower bound techniques, we count the number of times some operation (e.g. comparison of two elements) must be performed. By obtaining a lower bound on the number of times this operation must be performed before knowing the solution, we obtain a lower bound for any algorithm. For the below examples we focus on the number of comparisons made and thus our lower bounds apply to any comparison-based algorithm (i.e. the algorithm can only access the array elements by asking questions of the form is $A[i] \leq A[j]$ for any i, j). For all problems the input will be an array of n elements stored as $A[1], A[2], \dots, A[n]$.

Finding the Maximum and Minimum Elements

As our first example, we prove that $\lceil 3n/2 \rceil - 2$ comparisons are necessary in the worst case to find both the maximum and minimum of n numbers. Initially, there are n candidates for the maximum and minimum and at the end there is just one candidate each. Thus we must eliminate $2(n - 1) = 2n - 2$ candidates overall.

We now describe the adversary's strategy. Each element is kept in one of the following four bins: max/min (if candidates for both), min (if candidate for only min), max (if candidate for only max), neither (if candidate for neither). The adversary responds to the comparisons using the following strategy:

Comparison	Answer
1. max/min vs max/min	answer either way
2. max/min vs neither	answer max/min is larger
3. min vs min	answer either way
4. max vs max	answer either way
5. max vs "other"	answer max is larger
6. min vs "other"	answer other is larger
7. neither vs neither	answer consistently with past

In the first case the larger element is moved to max and the other to min. In this case exactly *two* candidates have been eliminated (one for min and one for max). In the second case, we move the element from max/min to max. So in this case, one candidate has been eliminated. In the third (respectively, fourth) case, the larger (respectively, smaller) element is moved to neither. Thus in both of these cases exactly *one* candidate is eliminated. Finally in cases 5–7 nothing is moved and no candidates are eliminated.

Note that case 1 can occur at most $\lfloor n/2 \rfloor$ times since this case removes 2 elements from the max/min bin. Each of the remaining comparisons can eliminate at most one candidate, and thus the number of comparisons is at least

$$((2n - 2) - 2\lfloor n/2 \rfloor) + \lfloor n/2 \rfloor = 2n - \lfloor n/2 \rfloor - 2.$$

As desired, for n even we get a lower bound of $2n - n/2 - 2 = \lceil 3n/2 \rceil - 2$, and for n odd we get a lower bound of $2n - (n - 1)/2 - 2 = \lceil 3n/2 \rceil - 2$.

Sorting

Here we give an alternate way to obtain the $\Omega(n \log n)$ decision tree lower bound that is covered in the text book. The adversary strategy described here can be used as an alternative to the decision tree technique.

Notice that there are $n!$ different permutations (and thus solutions) that the sorting algorithm must decide between. The adversary maintains a list L of all of the permutations that are consistent with the comparisons that the algorithm has made so far. So initially L contains all $n!$ permutations. The adversary's strategy for responding to "Is $A[i] \leq A[j]$?" is as follows. Let L_{yes} be the permutations in L for which $A[i] \leq A[j]$ and let L_{no} be the permutations in L for which $A[i] > A[j]$. (So $L = L_{yes} \cup L_{no}$). Then the adversary responds "yes" exactly when $|L_{yes}| \geq |L_{no}|$. In other words, the adversary answers in such a way to keep L as large as possible. Then the adversary updates L .

Notice that at each step the adversary will assure that at least half of the elements of L remain. Equivalently, at most half of the elements of L are removed. The algorithm cannot be done until $|L| = 1$ and hence the number of comparisons required is at least $\lceil \log(n!) \rceil = \Omega(n \log n)$.

Median Finding

Using an adversary argument we prove that *any* comparison-based algorithm for finding the median must make at least $\frac{3}{2}(n - 1)$ comparisons in the worst case. For convenience we shall assume that $n = 2k + 1$. This argument is easily modified for the case when n is even. Since the adversary has the right to select the input we can assume, without loss of generality, that the input elements are $1, \dots, 2k + 1$ and thus the algorithm must determine which element holds $k + 1$. We begin with the following definition. Let A_1, \dots, A_n be the n elements for which we are to find the median. Suppose m is the index of the median for $1 \leq m \leq n$. (And thus, A_m would be the median.) We define a *crucial* comparisons for element A_i (for $i \neq m$) to be either

1. $A_i < A_m$ or $A_i < A_j$ for some j for which $A_j < A_m$ (so A_i is less than the median),
2. $A_i > A_m$ or $A_i > A_j$ for some j for which $A_j > A_m$ (so A_i is greater than the median).

The intuition behind why these comparison are crucial is that any algorithm that claims to have computed the median must prove that exactly k elements are smaller than the median, and exactly k elements are larger than the median. Thus for each element besides the median there must be a crucial comparison made. We will make this argument more formal later.

We are now ready to define the adversary's strategy for answering the comparison questions. Each element can be in either heaven, purgatory, or hell. Also, a partial order is kept for the comparisons already made. Let ℓ denote the number of elements that have been placed in heaven (which is also the number of elements that have been placed in hell). Thus, initially $\ell = 0$. The adversary uses the following rules to answer the comparison question "Is $A_i < A_j$?" :

1. If A_i and A_j are in purgatory then respond “yes.” Then A_i is moved to hell and A_j is moved to heaven. Finally, the adversary updates the partial order and increments ℓ .
2. If A_i is in purgatory and A_j is in heaven then reply “yes” (and updates the partial order if needed).
3. If A_i is in purgatory and A_j is in hell then reply “no” (and updates the partial order if needed).
4. In all remaining cases (both A_i and A_j in heaven or hell) answer consistently with the partial order. If A_i and A_j are incomparable answer “yes” and updates the partial order accordingly.

First of all observe that ℓ is only incremented by rule 1. We now argue that rule 1 must have been executed exactly k times when the algorithm has completed. Observe that rule 1 is the only way that elements are moved, and furthermore it can be executed only when at least two elements are in purgatory. Thus it can be executed at most k times. Notice that by construction we can always assign values to A_1, \dots, A_n , so that the elements placed in hell will have values $1, \dots, \ell$ and the elements placed in heaven will have values $n - \ell + 1, \dots, n$. Thus, there is always a way to assign values to all elements in purgatory so that any given element in purgatory is the median. Namely, assign the desired element the value $k + 1$ and arbitrarily assign the others with the remainder of the numbers from $\ell + 1, \dots, n - \ell$. Hence, when any correct median finding algorithm is finished, a single element must be in purgatory. From this it immediately follows that exactly $k = (n - 1)/2$ comparison must be made between two elements in purgatory (i.e. rule 1).

We now complete the argument by proving that when the algorithm is completed, a crucial comparison must have been made for each element besides the one that remains in purgatory. Furthermore, none of the k comparisons made in rule 1 are crucial. These two observations complete the proof that $n - 1 + k = \frac{3}{2}(n - 1)$ comparisons must be made.

First let's argue that no comparison between two items in purgatory are crucial. When making this comparison, one element is earmarked to get some value $< k + 1$ and sent to hell and the other is earmarked to get some value $> k + 1$ and sent to heaven. Furthermore, this is the first comparison made for both of these elements. Thus by the definition of a crucial comparison, this comparison cannot be crucial for either element. We now argue, by contradiction, that when the algorithm has finished (i.e. it can correctly output the median), a crucial comparison must have been made for each element in heaven and hell. Suppose that A_i has been placed in hell (without loss of generality) and no crucial comparison has been made for it. Namely, A_i has not been compared to the median and A_i has not been compared to any item in hell that is known to be less than the median. But then the partial order must be such that there is a total order (i.e. a possible input) consistent with it for which A_i is the median. Thus, there are at least two possible candidates for the median and so the algorithm could not have computed the median. So there must be a crucial comparison made for each of the $n - 1$ elements besides the median in addition to the k comparisons between two items in purgatory.