Proving Lower Bounds

The following examples relate to proving lower bounds for comparison-based algorithms, using both decision trees and an adversary style proof. The first example considers finding duplicates in a sorted list of size n (lower bound of $n-1$), the second considers merging two sorted lists of size n (lower bound of $2n-1$), and the last two relate to using a decision tree to lower bound the number of comparisons in finding a value in a sorted array.

Example 1

Suppose we are given a sorted list of n numbers, $X = x_1 \leq x_2 \leq \ldots \leq x_n$, and we are asked to check whether or not there are any duplicates in the list. We are limited to algorithms which compare pairs of list elements (with a procedure COMPARE(i, j) which returns $<$, $>$ or $=$, depending on the values of x_i and x_j). Assume the algorithm returns either *distinct* (if there are no duplicates) or "i-th element equals j-th element" if $x_i = x_j$. Note that if there are mulitple ties, any one can be reported. Prove a good lower bound for the number of calls to COMPARE to solve this problem.

Solution:

Before selecting a proposed lower bound, and using an adversary argument to prove it, it is useful to determine the lower bound on the simplest and most straightforward algorithm. The naive approach simply walks the list and compares the current element to the next element, looking for duplicates. This takes $(n-1)$ comparisons. Thus, we will try to prove (using an adversary argument) that $(n-1)$ is the best (largest) lower bound possible. Our proof is as follows:

1. Assume there exists an algorithm A which runs in $(n-2)$ comparisons which correctly finds duplicates in an ordered list of size n.
2. Let X be a list such that $x_i = 2i$ for $i = 1$ to n. Note that all elements are unique.
3. Run algorithm A on input X. Since it only takes $(n-2)$ comparisons, there is at least 1 element which is not compared to its next element.
4. Find that element $x_i = 2i$ and set $x_{i+1} = 2i$. Remember that previously, $x_{i+1} = 2(i+1)$.
5. Rerun the algorithm. The algorithm will report no duplicates, as it did before, but it will be wrong.

Thus, there cannot exist any *correct* comparison-based algorithm that finds duplicates in a sorted list in less than $(n-1)$ comparisons.

Example 2

Give a $(2n-1)$ lower bound on the number of comparisons to merge two sorted lists $A_1 < A_2 < \ldots < A_n$ and $B_1 < B_2 < \ldots < B_n$ into a single sorted list.

Solution: It is useful to notice that with $(2n-1)$ comparisons, every element in the final merged list of $2n$ elements has been compared to its next element (except the last element). Thus, our adversarial argument is as follows:

1. Assume there exists an algorithm A which runs in $(2n-2)$ comparisons or less which correctly merges two sorted lists of size n.
2. Let X be a list such that $x_i = 2i-1$ for $i = 1$ to n. Let Y be a list such that $y_i = 2i$ for $i = 1$ to n. Note that all elements in both lists are unique.
3. Run algorithm A on input X and Y. Since it only takes $(2n-2)$ comparisons, there must be at least 1 element x_i in the final merged list which has not been compared to both y_i and y_{i+1}.

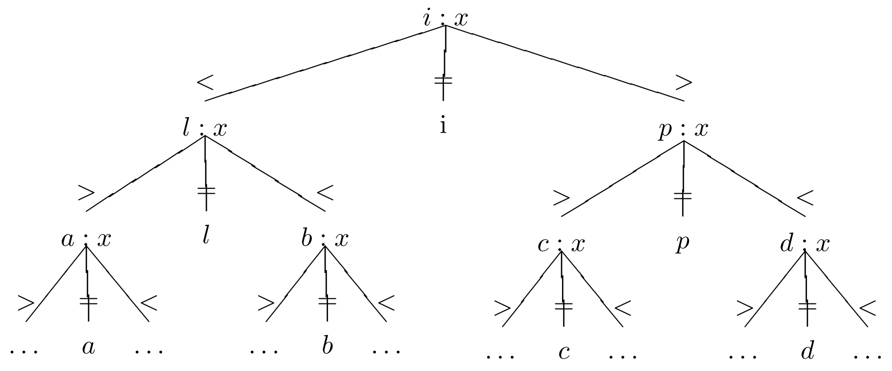
4. There are two cases. Either x_i has not been compared to y_i or it has not been compared to y_{i+1}. We will show the case where x_i has not been compared to y_i. The other case is similar. Find that element $x_i = 2i - 1$ and set $x_i = 2i$ and $y_i = 2i - 1$. This does not change the order of either list.

5. Rerun the algorithm. Since y_i and x_i were not compared, and the order is now different, the algorithm will sort the output in the same way as before, but now it will be wrong.

Thus, there cannot exist any *correct* comparison-based algorithm that merges two sorted lists of size n in less than $(2n - 1)$ comparisons.

Example 3

Suppose we are given a sorted list $A_1 < A_2 < \ldots < A_n$, and we are asked to find a lower-bound on a comparison-based algorithm for finding an arbitray value x in A. Assume that the algorithm outputs i if $x = A_i$ and zero otherwise. Consider the following decision tree:



Thus, we see that every "=" branch leads to a single leaf. Thus, the decision tree must have one node which compares x to A_i for $i = 1$ to n. If not, we could easily come up with an adversarial-type argument where the algorithm outputs an incorrect value. Since every internal node produces at most two non-leaf nodes, there are at most 2^k comparison nodes at level k. Since there are no comparisons performed on the last level of the decision tree (there are only leaf nodes), we can sum the number of comparisons performed at k levels as:

$$2^0 + 2^1 + 2^2 + \ldots + 2^{k-1} = \sum_{i=0}^{k-1} 2^i \quad (1)$$

$$= \frac{1 - 2^k}{1 - 2} \quad (2)$$

$$= 2^k - 1 \quad (3)$$

Furthermore, since there must be at least n comparison nodes in the tree:

$$2^k - 1 \geq n \quad (4)$$

$$2^k \geq n + 1 \quad (5)$$

$$\log 2^k \geq \log (n + 1) \quad (6)$$

$$k \geq \log (n + 1) \quad (7)$$

Thus, we have shown using a decision tree that the number of comparisons must be greater than or equal to $\log (n + 1)$.

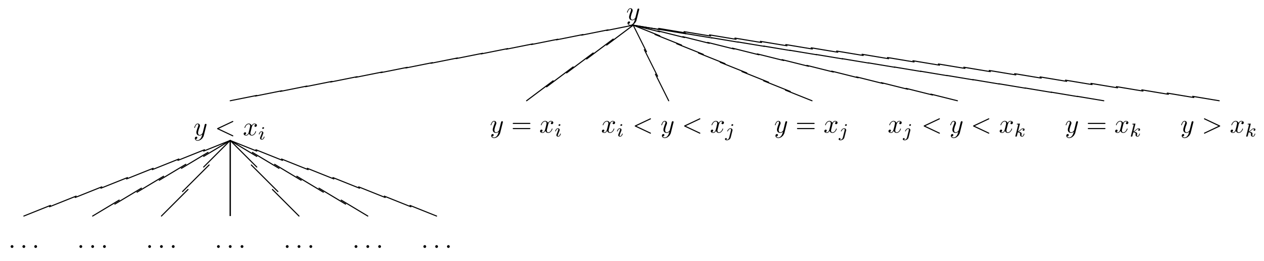
Example 4

Suppose we are given a sorted list $X = x_1 < x_2 < \ldots < x_n$. We want to search the list for a value y, and output 0 if y is not present and i if $y = x_i$ otherwise. However, the only way we can access X is using a procedure called $\textsc{Test}(i, j, i, k, y)$, which, when $0 < i < j < k \leq n$, returns one of seven answers:

1. $y < x_i$
2. $y = x_i$
3. $x_i < y < x_j$
4. $y = x_j$
5. $x_j < y < x_k$
6. $y = x_k$
7. $y > x_k$

Use a decision tree to prove a lower bound on the number of calls to TEST required to determine if y is in X, and return an index if yes.

Solution: Consider the following decision tree.



This tree is clearly not drawn with all of the branches, but every "=" branch leads to a single leaf as before. But in this case, each call to TEST has three "=" leaf nodes and 4 real branches. Since the decision tree must have one node which compares x to A_i for $i = 1$ to n, there must be at least $n/3$ internal nodes (each of which represents a call to TEST). Since every internal node produces at most 4 non-leaf nodes, there are at most 4^k comparison nodes at level k. Since there are no comparisons performed on the last level of the decision tree (there are only leaf nodes), we can sum the number of comparisons performed at k levels as:

$$4^0 + 4^1 + 4^2 + \ldots + 4^{k-1} = \sum_{i=0}^{k-1} 4^i \tag{8}$$

$$= \frac{1 - 4^k}{1 - 4} \tag{9}$$

$$= \frac{4^k - 1}{3} \tag{10}$$

Furthermore, since there must be at least $n/3$ comparison nodes in the tree:

$$\frac{4^k - 1}{3} \geq \frac{n}{3} \tag{11}$$

$$4^k - 1 \geq n \tag{12}$$

$$4^k \geq n + 1 \tag{13}$$

$$\log_4 4^k \geq \log_4 (n+1) \tag{14}$$

$$k \geq \log_4 (n+1) \tag{15}$$

$$k \geq \log(n+1)/2 \tag{16}$$

The last line follows from using the change of basis formula. Thus, we have shown using a decision tree, that the number of comparisons must be greater than or equal to $\log(n+1)/2$.