Lower Bounds in Comparison Model and More

1 Summary

1. Lower Bounds
2. Ways of showing lower bounds:
   (a) Counting argument / information theoretic
   (b) Adversary (against deterministic algorithm)
   (c) Reason about size / structure of decision tree
3. Designing algorithms with good query upper bounds.
4. Example problems:
   (a) Sorting.
   (b) Sorting under d distinct values.
   (c) Finding largest, second largest.
   (d) Binary search on number line, but only told relative change in distance between query and target.
   (e) Sorting using B-way comparisons.

2 Example Problems

Problem 1: Algorithmic intermediate value theorem

Consider a binary array $A[1 \dots n] \in \{0,1\}^n$, such that $A[1] = 0$, $A[n] = 1$. Show that the number of queries of A needed to find some entry i such that $A[i] = 0$, $A[i+1] = 1$ is $\log_2 n + \Theta(1)$.

Proof.

Consider the set of inputs of the form i 0s followed by j 1s where $i+j = n$. Since there are exactly n ways to assign i and j to meet the constraint, there are n inputs in the set. Each input has only 1 valid place where a 0 is followed by a 1 which will always occur at index i. Now assume we have a binary decision tree capable of solving this problem. It must be able to differentiate

between all n inputs in the set. Therefore, it's depth must be at least $\log n$. Thus, no algorithm can solve this problem in less than $\log n$ steps.

□

Problem 2: Select top-two

Consider the problem of finding the two largest elements, i.e., the max and the second-max among a list of n unsorted elements $a_1, a_2, \ldots, a_n$ in the comparison model. In this problem we will show a tight lower bound using a *decision tree argument*.

At its core, the argument will essentially be a **reduction** from the select-max problem. Given an algorithm that solves select top-two, we can fix one element to be larger than all the others, and then use that algorithm to find the second maximum (which will be the maximum in the original input). Since we already know a lower bound for select-max of $n-1$, we can then use that to get a lower bound for select top-two.

1. Consider a decision tree for an algorithm solving the select-top-two problem. Each leaf in the decision tree corresponds to an output (i, j) where i is the index of the largest element and j is the index of the second-largest element. Argue that for a fixed value of i, the number of leaves for which i is the maximum is at least 2^{n-2}.
2. Argue that the total number of leaves in the decision tree is at least $n2^{n-2}$ and use this fact to deduce a lower bound on the cost of the problem.

1. *Proof.*

Assume we are given a binary decision tree which correctly solves the 2-max problem on inputs of length n. Let $i \in [n]$ be arbitrary, and let $\mathscr{L}_i$ be the set of leaves in the given binary decision tree such that the output value at a leaf in $\mathscr{L}_i$ is of the form (i, j) for some $j \in [n]$. We wish to show that $|\mathscr{L}_i| \geq 2^{n-2}$. Let $\mathscr{S}_i$ be the set of all internal nodes x in our binary decision tree such that the subtrees rooted at the left and right child of x both contain an element of $\mathscr{L}_i$.

Lemma 1. For all $\ell \in \mathscr{L}_i$, the path in the given binary decision tree from the root to ℓ contains at least $n-2$ nodes from $\mathscr{S}_i$.

Proof of Lemma 1.

First, we observe that one can convert a decision tree which solves the 2-max problem on inputs of size n into a decision tree which solves the 1-max problem on inputs of size $n-1$ as follows: Given input x of size $n-1$, insert ∞ at position i in the input array (so that we now have an array of size n), and run this input on the given decision tree for 2-max and output the index of the second largest element. In the above process, one knows that the output leaf must be an element of $\mathscr{L}_i$, so one only needs to perform the comparisons at nodes in $\mathscr{S}_i$.

Now, suppose towards a contradiction that the desired statement is false, that is, there exists $\ell \in \mathscr{L}_i$ such that the path to ℓ contains fewer than $n-2$ nodes in $\mathscr{S}_i$. This means, there exists an input to 1-max of length $n-1$ on which the "converted" decision tree as described above performs fewer than $n-2$ comparisons. However, this contradicts the known best case lower bound for the 1-max problem which says that an input of length $n-1$ requires at least $n-2$ comparisons.

□

Proof of Problem Statement.

Consider truncating the tree to only contain nodes from $\mathscr{L}_i$ and $\mathscr{S}_i$.

First, we claim that this tree will have a depth of at least $n-1$. That is because each leaf of the tree is in L_i, and therefore contains $n-2$ nodes of S_i on the path to it from the root. Since all of these nodes are present in the tree, each leaf sits at a depth of at least $n-2+1$, counting itself.

Second, we claim that this is a full binary tree. This is because every internal node of the tree is in S_i, and so by definition has both of its children, as by definition it has a path to an L_i on either side.

Therefore, as the tree is a full binary tree of depth at least $n-1$, it must have at least 2^{n-2} leaves. Finally, since each of these leaves are in L_i, we know that $|\mathscr{L}_i| \geq 2^{n-2}$.

Alternate proof using induction. This proof is via induction on n. The base case $n = 2$ is immediate. For the induction step, fix $n > 2$. The path in the tree to a leaf $\ell \in \mathscr{L}_i$ must contain at least $n-2$ nodes from $\mathscr{S}_i$. Let x be the first node on this path. The subtrees rooted at the left and right children of x must be such that any path to a leaf in $\mathscr{L}_i$ contains

$n-3$ nodes in $\mathscr{S}_i$, thus, the induction hypothesis to get that these subtrees each contain at least 2^{n-3} distinct elements which are in $\mathscr{L}_i$. Since these subtrees are disjoint, we have that the total number of elements in $\mathscr{L}_i$ is at least $2(2^{n-3}) = 2^{n-2}$.

□

2. *Proof.*

Since the sets $\{\mathscr{L}_i\}_{i=1}^n$ partition the set of leaves in the given decision tree, we have that the total number of leaves is given by

$$\left|\bigcup_{i=1}^{n} \mathscr{L}_i\right| = \sum_{i=1}^{n} |\mathscr{L}_i| \geq \sum_{i=1}^{n} 2^{n-2} = n2^{n-2}.$$

Since a binary tree with L leaves has depth at least $\log_2(L)$, this gives a lower bound on the 2-max problem of $\log_2(n2^{n-2}) = n-2+\log_2(n)$.

□