15-451/651 Algorithm Design & Analysis, Spring 2024

Recitation #1

Objectives

- Understand the concept of a *lower bound* for the cost of an algorithm
- Understand the different proof techniques that can be used to establish lower bounds
- Practice identifying flawed lower bounds proofs and writing correct ones

Recitation Problems

1. **(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 lecture we gave an algorithm that uses $n + \log n - 2$ comparisons. In this problem we will show a tight lower bound using a *decision tree argument*.

   (a) 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}.

   (b) 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.

2. **(Sorting few distinct elements)** Suppose you have a list of n elements $a_1, a_2, \ldots, a_n$, but there are at most D distinct elements in this list. You would like to design a comparison-based algorithm for sorting these n elements.

   (a) Describe an algorithm that can sort a list containing at most D distinct elements in $O(n \log D)$ comparisons.

   (b) Now we would like to prove a lower bound for the problem. Consider the following attempted proofs. Some of them are correct and some are incorrect. Identify which are which, and for the incorrect ones, explain why they are incorrect[1].

      i. In a list of size n consisting of at most D distinct elements, each number has at most D possibilities, so the number of possible input lists for this problem is D^n. Since in the worst case any comparison can rule out half of the previously compatible inputs, to narrow down to just 1 input takes at least $\log(D^n) = n \log D$ comparisons, and so we have an $\Omega(n \log D)$ lower bound in the comparison model.

[1] You should be looking for mistakes in the *overall logic* of the proof. There are no intentional mistakes hidden in the math, so don't overthink the math parts. You can assume they are correct

ii. Consider the following set of possible sorted outputs to this problem:

$$\underbrace{1,1,\ldots,1}_{x_1},\underbrace{2,2,\ldots,2}_{x_2},\ldots,\underbrace{D,D,\ldots,D}_{x_D}$$

where $x_1, x_2, \ldots, x_D \geq 0$ and $x_1 + x_2 + \cdots + x_D = n$. By the "stars and bars" combinatorial argument, we can say that the number of such outputs is

$$\binom{n+D-1}{D-1}$$

and since no possible input can legally produce multiple of these outputs, any algorithm must correctly distinguish the 1 exact valid output among them. Therefore, by an information-theoretic argument we get a lower bound of

$$\log\binom{n+D-1}{D-1} \geq \log\left(\frac{n+D-1}{D-1}\right)^{D-1} \geq \Omega\left(D\log\left(\frac{n}{D}\right)\right)$$

iii. Consider a set of possible inputs of size n with D distinct elements constructed as follows. Create n/D contiguous chunks of size D. Each chunk contains the elements $1...D$ in a random order. Since each chunk is size D and is in a random order, there are $D!$ possible orders for each chunk. Since there are n/D chunks, the total number of inputs in the set is $(D!)^{\frac{n}{D}}$. Now observe that for each of these input sequences, no two of them are ever sorted by the same permutation. This is because each permutation selects exactly one item from each chunk for each element, and the elements of the chunks are unique. Therefore, we require at least $(D!)^{\frac{n}{D}}$ different possible permutations to sort them all.

Taking the log of this quantity, we get an information-theoretic lower bound of

$$\log\left((D!)^{\frac{n}{D}}\right) = \frac{n}{D}\log(D!) = \Omega\left(\frac{n}{D}(D\log(D))\right) = \Omega(n\log D),$$

where we have used the fact from lecture that $\log D! = \Theta(D\log D)$.

iv. Consider the class of input sequences in which each of the D elements occurs the same number of times, i.e., there are n/D copies of each of the D elements. Now note that each of our n elements can go in any of at most n locations in the array. This will create $\left(\frac{n}{D}!\right)^D$ copies of each array due to permuting the n/D identical elements of all D values. Since each non-duplicate value among these needs a distinct output (there is only one valid sorted order), there are

$$\frac{n^n}{\left(\frac{n}{D}!\right)^D}$$

different permutations required to sort each of the possible input sequences (intuitively, each permutation can be used to sort at most $\left(\frac{n}{D}!\right)^D$ inputs), so we can take the log of this quantity to achieve an information-theoretic lower bound of

$$\log \frac{n^n}{\left(\frac{n}{D}!\right)^D} \geq \frac{n^n}{\left(\left(\frac{n}{D}\right)^{\frac{n}{D}}\right)^D} = \frac{n^n}{\left(\frac{n}{D}\right)^n} = \Omega(n \log D)$$

v. Consider the class of input sequences in which each of the D elements occurs the same number of times, i.e., there are n/D copies of each of the D elements. Recall that for a sequence of n distinct elements, there are $n!$ distinct permutations. Note that for a sequence with n/D copies of each of the D elements, permuting any subsequence of equal elements results in an also correctly sorted sequence. Therefore for each distinct element, there are $(n/D)!$ ways they could be permuted and still be sorted. Combining all of these possibilities over the D elements, for any input sequence, there are exactly $((n/D)!)^D$ permutations that all correctly sort it. These sets of permutations are all disjoint, and hence there are

$$\frac{n!}{\left(\frac{n}{D}!\right)^D}$$

different permutations required to sort each of the possible input sequences. We therefore get an information-theoretic lower bound of

$$\log\left(\frac{n!}{\left(\frac{n}{D}!\right)^D}\right) \geq \log\left(\frac{D}{e}\right)^n = \Omega(n \log D).$$

3. **(Median of two sorted arrays)** Let A and B be two sorted arrays of n elements each. We can easily find the median element in A — it is just the element in the middle — and similarly we can easily find the median element in B. (Let us define the median of $2k$ elements as the element that is greater than $k-1$ elements and less than k elements.) However, suppose we want to find the median element overall — i.e., the nth smallest in the *union* of A and B. How quickly can we do that? You may assume there are no duplicate elements.

   Your job is to give tight upper and lower bounds for this problem. Specifically, for some function $f(n)$,

   (a) Give an algorithm whose running time (measured in terms of number of comparisons) is $O(f(n))$, and

   (b) Give a lower bound showing that any comparison-based algorithm must make $\Omega(f(n))$ comparisons in the worst case.

   In fact, see if you can get rid of the O and Ω to make your bounds *exactly* tight in terms of the number of comparisons needed for this problem.

   Some hints: You may wish to try small cases. For the lower bound, you should think of the output of the algorithm as being the location of the desired element (e.g, "$A[17]$") rather than the element itself. How many different possible outputs are there?