15-451/651 Algorithm Design & Analysis, Fall 2023

Recitation #1

Objectives

- Practice writing recurrence relations to analyze the runtime of algorithms
- Understand the concept of a *lower bound* for the running time 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. **(Why do I feel empty inside)** Recall that a graph property is said to be evasive if for every algorithm to verify the property, there is at least one graph such that its membership in G cannot be decided until all possible edges in the graph have been searched.

   For this problem, we'll reason about the emptiness property. A graph is considered empty if there are no edges in the graph. Describe the evader's strategy to show that emptiness in a graph is evasive.

2. **(Quartile of quartiles)** Wurzelbrunft is taking 451, but he hates medians, and so he comes up with a new algorithm for deterministic selection that pivots on the first quartile of the medians (the element 25% of the way through in order), rather than the median of the medians:

   1. Group the array into $n/5$ groups of size 5 and find the median of each group. (For simplicity, we will ignore integrality issues.)
   2. Recursively, find the true quartile of the medians. Call this p.
   3. Use p as a pivot to split the array into subarrays LESS and GREATER.
   4. Recurse on the appropriate piece.

   He tasks you, his group member, with analyzing his new algorithm before the oral.

   (a) Write the recurrence relation for Wurzelbrunft's algorithm and compute its time complexity.

(b) Can you improve the time complexity of Wurzelbrunft's algorithm by changing only the elements used in the recursive call?

3. **(Sorting few distinct numbers)** Suppose you have a list of n numbers $a_1, a_2, \ldots, a_n$, but there are at most D distinct numbers in this list. You would like to design a comparison-based algorithm for sorting these n numbers, (i.e., the only operation allowed is to choose two input numbers a_i and a_j and check whether or not $a_i \leq a_j$, at a cost of one operation).

   (a) Describe an algorithm that can sort a list containing at most D distinct numbers 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 numbers, 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 and each must have a distinct leaf a correct decision tree. Therefore, by an information-theoretic argument we get a lower bound on tree height 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 numbers 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, each representing a leaf in a decision tree.

Taking the log of this quantity, we get the minimum height of any decision tree with that many leaves, and have a 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. This means we have at least this many leaves in the decision tree (intuitively, each leaf of the tree can correspond to at most $\left(\frac{n}{D}!\right)^D$ inputs), so we can take the log of this quantity to achieve a lower bound on height 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. This means we have at least this many leaves in a decision tree for this problem, which must therefore have height at least

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