

15-451/651 Algorithm Design & Analysis, Fall 2025

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. (**Sorting few distinct elements**) Suppose you have a list of n elements a_1, a_2, \dots, a_n , but you are guaranteed that 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 incorrect¹.

i. In a list of size n consisting of at most D distinct elements, each element has at most D possibilities, so the number of possible input lists for this problem is D^n . Since an algorithm performing c comparisons can output 2^c different answers, it takes at least $\log(D^n) = n \log D$ comparisons to solve all these inputs, and so we have an $\Omega(n \log D)$ lower bound in the comparison model.

ii. Given D distinct elements, which we will represent as $1, \dots, D$, the sorted output will look like:

$$\underbrace{1, 1, \dots, 1}_{c_1}, \underbrace{2, 2, \dots, 2}_{c_2}, \dots, \underbrace{D, D, \dots, D}_{c_D}$$

for some values of c_i , where $c_1, c_2, \dots, c_D \geq 0$ and $c_1 + c_2 + \dots + c_D = n$. We can count the number of possible values of the c 's to count the number of possible outputs. By the "stars and bars" combinatorial argument, this quantity 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

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

¹You 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

- 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 $1 \dots D$ 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 $(D!)^{n/D}$.

Now, critically, observe that since we have exactly the same number of each distinct element in the input, the correct sorted order is

$$\underbrace{1, 1, \dots, 1}_{n/D}, \underbrace{2, 2, \dots, 2}_{n/D}, \dots, \underbrace{D, D, \dots, D}_{n/D}.$$

So, consider two distinct input sequences, there will be some position at which they contain a different element, say WLOG 1 and 2. If we apply the same output permutation to both, one of them will end up with a 2 in the 1s chunk or a 1 in the 2s chunk and hence must be sorted incorrectly. **Therefore, no two input sequences are ever sorted by the same permutation.** Therefore, we require at least $(D!)^{n/D}$ different possible permutations to sort all of these possible input sequences.

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

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

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

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

- 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, so we need just one of each, and hence we require

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

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

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

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, \dots, 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.

- (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.