15-451/651 Algorithm Design & Analysis
Spring 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

Mathematical Preparation

1. **(Asymptotic analysis)** For each pair $\langle f, g\rangle$ of functions below, list which of the following are true: $f(n) = o(g(n))$, $f(n) = \Theta(g(n))$, or $g(n) = o(f(n))$.

   (a) $f(n) = \ln(n), g(n) = \log_{10}(n)$.

   (b) $f(n) = n^{1.5}, g(n) = n\log^4(n)$.

   (c) $f(n) = 2^{2n}, g(n) = 2^n$.

   (d) $f(n) = n^{0.001}, g(n) = (\log_2 n)^{100}$.

2. **(Solving recurrences)** Solve the following recurrences in Θ notation by the tree/brick method (assume all base cases have $T(1) = 0$):

   (a) $T(n) = 1 + 2T(n-1)$.

   (b) $T(n) = n + T(n/2)$.

   (c) $T(n) = n + 2T(n/2)$.

Recitation Problems

1. **(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 $1 \ldots D$ 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

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

2. **(Breaking Eggs)** Say I choose a number between 1 and N. You want to guess it in as few questions as possible (each time you make an incorrect guess, I'll tell you if it is too high or too low). As we know, the strategy that minimizes the worst-case number of guesses is to do binary search: it takes $\lceil \log_2 N \rceil$ guesses.

   But, what if you are only allowed **one** guess that is too high? Can you still solve the problem in $o(N)$ guesses? [If you were not allowed **any** guesses that are too high, the only option you would have would be to guess 1,2,3,... in order].

   (a) Show how to figure out how many centimeters high you can drop an egg without it breaking, using $2\sqrt{N}$ guesses, when you only have two eggs.

   (b) Can you show an $\Omega(\sqrt{N})$ lower bound for any deterministic algorithm via an adversarial argument?

3. **(Optional: 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?

Further Review

1. **(Short answer / multiple choice)**

   (i) What is the solution of $A(n) = 3A(n-1) + 3^n$ with $A(1) = 1$

   (a) $\Theta(3^n)$

   (b) $\Theta(n3^n)$

   (c) $\Theta(3^{n^2})$

   (d) $\Theta(9^n)$

   (e) $\Theta(3^n \log n)$

   (ii) What is a correct bound on $A(n)$, where

$$A(n) = \begin{cases} 8A(n/2) + n^2 & \text{for } n \geq 2, \\ 1 & \text{for } 0 \leq n \leq 1. \end{cases}$$

   (a) $A(n) = \Theta(n^2 \log n)$

   (b) $A(n) = \Theta(n^3)$

   (c) $A(n) = \Theta(n^3 \log n)$

   (d) $A(n) = \Theta(n^4)$

   (e) $A(n) = \Theta(n^4 \log n)$

   (iii) What is a correct bound for $T(n)$, where

$$T(n) = \begin{cases} 4T(n/2) + \sqrt{n} & \text{for } n \geq 2, \\ 1 & \text{for } 0 \leq n \leq 1. \end{cases}$$

   (a) $T(n) = \Theta(\sqrt{n})$

   (b) $T(n) = \Theta(\sqrt{n} \log n)$

   (c) $T(n) = \Theta(n)$

   (d) $T(n) = \Theta(n \log n)$

   (e) $T(n) = \Theta(n^2)$

   (f) $T(n) = \Theta(n^2 \log n)$

   (iv) In the `DeterministicSelect` algorithm, the median of medians procedure

   (a) recursively calls itself

   (b) calls `DeterministicSelect`

   (c) Both of the above

(v) What would be the worst-case running time of the `Quicksort` algorithm if we pick an arbitrary (fixed) element as the pivot?

(a) $\Theta(n)$

(b) $\Theta(n \log n)$

(c) $\Theta(n^2)$

(d) $\Theta(n^2 \log n)$

(vi) What would be the **expected** worst-case running time of the `Quicksort` algorithm if we pick a random element as the pivot?

(a) $\Theta(n)$

(b) $\Theta(n \log n)$

(c) $\Theta(n^2)$

(d) $\Theta(n^2 \log n)$

(vii) What would be the worst-case running time of the `Quicksort` algorithm if we used `DeterministicSelect` to select the pivot element?

(a) $\Theta(n)$

(b) $\Theta(n \log n)$

(c) $\Theta(n^2)$

(d) $\Theta(n^2 \log n)$

(viii) Regarding lower bounds for problems in the comparison model, which of the following statements are true and which are false?

(a) $\exists M$ distinct inputs $\implies$ lower bound of $\lg M$

(b) $\exists M$ distinct inputs that require distinct outputs $\implies$ lower bound of $\lg M$

(c) $\exists M$ distinct outputs, each required by some input $\implies$ lower bound of $\lg M$

(d) $\exists M$ distinct outputs, each possible for some input $\implies$ lower bound of $\lg M$

(ix) Out of the following choices, what is the largest lower bound on the number of comparisons required to find the second largest element in an array?

(a) $n + \lg n - 2$

(b) n

(c) $n - 1$

(d) $2n - 3$

(e) $n \lg n$

2. **(Partition)** Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{25}$ of the functions such that $g_n = \Omega(g_{n+1})$ for $n = 1, \ldots, 24$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.

$(3/2)^n$	$(\sqrt{2})^{\lg n}$	$\lg^* n$	n^2	$(\lg n)!$
n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\lg\lg n$	$n \cdot 2^n$	$n^{\lg\lg n}$	$\ln n$	2^n
$2^{\lg n}$	$(\lg n)^{\lg n}$	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$n!$	$2^{\sqrt{2\lg n}}$	n	$n \lg n$	1

3. **(More recurrences)** Solve the following recurrences, assuming a base case of $T(0) = 1, T(1) = 1$

   (a) $T(n) = 3T(n/2) + n^2$.

   (b) $T(n) = 4T(n/2) + n^2$.

   (c) $T(n) = 5T(n/2) + n^2$.

4. **(Probability Facts)** Let's follow the convention of using uppercase letters X, Y, Z to denote *random variables* (which we often abbreviate to *r.v.*s).

   Independence Random variables X, Y are *independent* if for any values a, b, we have
$$\Pr[X = a, Y = b] = \Pr[X = a] \cdot \Pr[Y = b].$$

   (a) Consider flipping two fair coins. Let $X \in \{0, 1\}$ be the outcome of the first coin (where we think of heads as 1 and tails as 0) and let $Y \in \{0, 1\}$ be the outcome of the second coin. Let $Z = X + Y \bmod 2$. Are X and Y independent? What about X and Z?

Linearity of expectation Given any two r.v.s X, Y,

$$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y].$$

This is true even if X and Y are <u>*not independent*</u>*!*

(b) Suppose we take n books numbered 1 to n, and place them on a shelf in a (uniformly) *random* order. What is the expected number of books that end up in their correct location? ("correct" = location it would be in if they were sorted).

Markov's inequality given a *non-negative random variable* X with mean/expectation $\mu = \mathbb{E}[X]$,

$$\Pr[X > c \cdot \mu] \leq \frac{1}{c}.$$

(c) Show that if X is allowed to take negative values then the above inequality is no longer true.

5. **(Summations)** Prove the following theorem: For $x < 1$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

6. **(Deciding the maximum)** In lecture we used an adversary argument to obtain a lower bound of $n-1$ comparisons for finding the maximum element of a list.

   (a) Suppose we instead used an information-theoretic argument. What lower bound would we obtain for the problem? Is it useful?

   (b) Suppose we instead use decision trees to analyze the problem

      i. How many outputs does the root node of any decision tree for this problem contain?

      ii. How many outputs does each internal node of any decision tree for this problem contain?

      iii. Are the leaves of these decision trees unique?

      iv. How many nodes do each of the possible decision tree have?

      v. What lower bound is implied by the structure of these decision trees?

7. **(Lower bounds via reductions)** In 15-251, you learned how to use *reductions* to show that one problem is just "as hard" as another problem. 251 taught you how to show that a problem is loosely in the same complexity class as another (e.g., NP-hard). This technique is not only applicable to loose complexity classes, but can also be used to establish tight lower bounds!

   Lets do an example. Recall the partitioning algorithm used as a subroutine of Quicksort, which, given an array A and some "pivot" element $A[i]$ of the array, rearranges the elements of A so that all elements that are $\leq A[i]$ come before all elements that are $> A[i]$. What if we generalize this problem to pick k distinct pivot elements, $A[i_1], A[i_2], ..., A[i_k]$? (The pivots do not have to be given in sorted order). The problem then becomes to rearrange the array so that all elements that are at most as large as the smallest pivot come first, followed by those that are larger but at most as large as the second-smallest pivot and so on...

   (a) Give an algorithm that solves k-partitioning in $\Theta(n \log k)$ comparisons.

   (b) Show that there can not possibly exist an algorithm that performs k-partitioning in fewer than $\Theta(n \log k)$ comparisons. To do so, show that if you assume such an algorithm exists, then it must be possible to sort in fewer than $\Theta(n \log n)$ comparisons.

8. **(Breaking Eggs Extended)**

   (a) Improve the algorithm to have an upper bound of $\sqrt{2N}$ guesses.

   (b) Show upper/lower bounds of $\Theta(N^{1/3})$ if you're allowed *two* guesses that are too high.

9. **(Sorting nuts and bolts)** Consider the following problem. You have a bag of n nuts and a bag of n bolts. The nuts and bolts are all of different sizes, but for each nut there is a matching bolt (and vice versa). What you want to do is to find all the matching pairs. Unfortunately, you can't compare a nut to a nut, or a bolt to a bolt (your eyes are too weak). Instead, what you can do is to compare a nut to a bolt and see if the bolt is too small, too big, or just right (a perfect match). For instance, a strategy that makes $O(n^2)$ comparisons to find the matching pairs is for each nut to go through all the bolts until you find the matching one, and then continue with the next nut, and so on.

   Find a *randomized* algorithm for this problem that uses an expected $O(n \log n)$ comparisons.

10. **(Median of two sorted lists)** Let $X[1 \ldots n]$ and $Y[1 \ldots n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\log n)$-time algorithm to find the median of all $2n$ elements in arrays X and Y.