CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, MARCH 4, 2022 AT 11:30 PM VIA GRADESCOPE

AADV(WYZ5RGE)

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: jhs8cue, kd5eyn, spj6s

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *Bazinga!*

Theoretical Physicist Sheldon Cooper has decided to give up on String Theory in favor of researching Dark Matter. Unfortunately, his grant-funded position at Caltech is dependent on his continued work in String Theory, so he must search elsewhere. He applies and receives offers from MIT and Harvard. While money is no object to Sheldon, he wants to ensure he's paid fairly and that his offers are at least the median salary among the two schools' Physics departments. Therefore, he hires you to find the median salary across the two departments. Each school mantains a database of all of the salaries for that particular school, but there is no central database.

Each school has given you the ability to access their particular data by executing *queries*. For each query, you provide a particular database with a value k such that $1 \leq k \leq n$, and the database returns to you the k^{th} smallest salary in that school's Physics department.

You may assume that: each school has exactly n physicists (i.e. $2n$ total physicists across both schools), every salary is unique (i.e. no two physicists, regardless of school, have the same salary), and we define the *median* as the n^{th} highest salary across both schools.

1. Design an algorithm that finds the median salary across both schools in $\Theta(log(n))$ total queries.

   Solution: Take the median of both schools' salary lists by taking making a query such that $k = \frac{n}{2}$. We then have two cases:

   (a) The median of the first list is larger than the median of the second list. This means the value we're looking for is either in the first half of the first list or the second half of the second list.

   (b) The median of the first list is smaller than the median of the second list. This means the value we're looking for is either in the second half of the first list or the first half of the second list.

   The lists can now be divided in half according to one of the cases stated above. Repeat this dividing process until we're left with two lists, each with two values, with the smaller value in front. The median of these four values can be computed by finding the max of the first value in both lists and the min of the last value in both lists, adding them together, and dividing by 2 to find the combined median of both lists.

2. State the complete recurrence for your algorithm. You may put your $f(n)$ in big-theta notation. Show that the solution for your recurrence is $\Theta(log(n))$.

Solution: $T(n) = T(\frac{n}{2}) + 2$, with base case $T(1) = 1$
This recurrence can be proven to be $\Theta(\log n))$ by the unrolling method.

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + 2 \\ &= T(\frac{n}{4}) + 4 \\ &= T(\frac{n}{8}) + 6 \\ &... \end{aligned}$$

We repeat this for i repetitions so that i tends towards $log_2 n$ and $\frac{n}{2^i}$ tends towards 1 and we arrive at a general form

$$\begin{aligned} T(n) &= T(\frac{n}{2^i}) + 2i \\ &= T(\frac{n}{2^{\log_2 n}}) + 2\log_2 n \\ &= 2\log n \\ T(n) &\in \Theta(\log n) \end{aligned}$$

3. Prove that your algorithm above finds the correct answer. *Hint: Do induction on the size of the input.*

   Solution:

   Proof. We proceed by induction.
   Base Case: $n = 2$
   With two lists of size $n = 2$, no dividing needs to be done and we can follow the above description of the algorithm, finding the middle two values and averaging to find the combined median.
   Inductive Hypothesis: For all n_0 larger than n, the algorithm finds the correct value as the combined median.
   Inductive Step: $n_0 > n$
   For any set of two lists of size n_0, we can keep dividing and eliminating half of the lists, reducing them both to size $\frac{n_0}{2}$, by selecting a midpoint through integer division and disregarding everything in the list before or after said midpoint, depending on the case encountered after comparing medians. This new problem size $\frac{n_0}{2}$ is less than the original problem size and thus can be repeated until we reach our base case where we have two lists of size 2, and we can once again compute median from these four values. Thus following our inductive step shows that our inductive hypothesis holds for all $n_0 > n$, so it must be the case that our described algorithm can find the correct answer for all $n \geq 2$. □

PROBLEM 2 *Castle Hunter*

We are currently developing a new board game called *Castle Hunter*. This game works similarly to *Battleship*, except instead of trying to find your opponent's ships on a two dimensional board, you're trying to find and destroy a castle in your opponent's one dimensional board. Each player will decide the layout of their terrain, with castles placed on each hill. Specifically, each castle is placed such that they are higher than the surrounding area, i.e. they are on a local maximum,

because hill tops are easier to defend. Each player's board will be a list of n floating point values. To guarantee that a local maximum exists somewhere in each player's list, we will force the first two elements in the list to be (in order) 0 and 1, and the last two elements to be (in order) 1 and 0.

To make progress, you name an index of your opponent's list, and she/he must respond with the value stored at that index (i.e., the altitude of the terrain). To win you must correctly identify that a particular index is a local maximum (the ends don't count), i.e., find one castle. An example board is shown in Figure 1. [We will require that all values in the list, excepting the first and last pairs, be unique.]

0	1	4	23	18	14	15	13	1	0
0	1	2	3	4	5	6	7	8	9

Figure 1: An example board of size $n = 10$. You win if you can identify any one local maximum (a castle); in this case both index 3 and index 6 are local maxima.

1. Devise a strategy which will guarantee that you can find a local maximum in your opponent's board using no more than $O(\log n)$ queries, prove your run time and correctness.

   Solution: Begin by choosing the midpoint of the list by integer division $\frac{n}{2}$, and comparing the value at this position to its adjacent elements. If the element to the right is greater, then use the current midpoint as the "start" of the right sub-list and repeat the process of choosing a midpoint and comparing against adjacent elements. Inversely, if the element to the left is greater, do the same process, but on the left sub-list. The process just described is very similar to binary serach. Continue until you reach the base case, which is when the elements adjacent to a position are less than or equal to the position being compared against. This indicates that the current position is either a plateau or a peak. The recurrence relation for this algorithm would be T(n) = T(n/2) + 2, the same as the algorithm described in question 1, which we have proven to be $\Theta(\log n)$.

2. Now show that $\Omega(\log n)$ queries are required by *any* algorithm (in the worst case). To do this, show that there is a way that your opponent could dynamically select values for each query as you ask them, rather than in advance (i.e. cheat, that scoundrel!) in such a way that $\Omega(\log n)$ queries are required by *any* guessing strategy you might use.

   Solution: The worst case solution for this problem would be if the value we're looking for is at the extremes (technically 2 positions away from the extremes).
   The fastest way to find this value is to take a divide-and-conquer approach by picking a midpoint, using a decision function to determine if the peak lies to the left or right of the midpoint, and recursing a smaller sub-problem. As we have proven, this divide-and-conquer strategy has a time complexity of $\Theta(\log n)$. If we were to do fewer than $\log n$ queries, we would not be thoroughly searching through the board and thus an algorithm that could theoretically solve this problem faster than $\log n$ time would not be checking, or at least ruling out, every possible value and thus would not be stable/deterministic. Even if an opponent could dynamically select values such that this problem would be worst-case, i.e., we have to keep recursing left or recursing right, searching for a value on the extremes would still require $\log n$ decisions.

PROBLEM 3 *Goldilocks and the n Bears*

BookWorld needs your help! Literary Detective Thursday Next is investigating the case of the mixed up porridge bowls. Mama and Papa Bear have called her to help "sort out" the mix-up

caused by Goldilocks, who mixed up their n bear cubs' bowls of porridge (there are n bear cubs total and n bowls of porridge total). Each bear cub likes his/her porridge at a specific temperature, and thermometers haven't been invented in BookWorld at the time of this case. Since temperature is subjective (without thermometers), we can't ask the bears to compare themselves to one another directly. Similarly, since porridge can't talk, we can't ask the porridge to compare themselves to one another. Therefore, to match up each bear cub with their preferred bowl, Thursday Next must ask the cubs to check a specific bowl of porridge. After tasting a bowl of porridge, the cub will say one of "this porridge is too hot," "this porridge is too cold," or "this porridge is just right."

1. Give a *brute force* algorithm for matching up bears with their preferred bowls of porridge which performs $O(n^2)$ total "tastes." Prove that your algorithm is correct and that its running time is $O(n^2)$.

   Solution: Have a list of porridge and a list of bears. Iterating through the list of porridge, have each bear taste the current bowl of porridge until the bear who prefers it is identified, removing the bear and the bowl from their respective lists and moving onto the next bowl of porridge until all bowls and matched with the correct bear. This is essentially a nested for loop, with n iterations being made for each n. Since we are doing n iterations n times, this brute-force algorithm is $O(n^2)$.
   To prove this algorithm is correct, we can start with a base case of $n = 2$. The worst case for this problem is that for each bowl, our linear search is worst case. In other words, the bear that prefers each bowl we examine is always at the end of the line. So for our first bowl, we have to have the first bear taste it before we get to the second bear. Then for the second bowl, the last remaining bear tastes it and then we are done. This base case took a total of 3 tastes, which is upper bounded by $n^2 = 4$. For all $n > 2$, a similar process take place, where we perform n linear searches on a list that decrements by 1 each time, which is upper bounded by n^2.

2. Give an *randomized* algorithm which matches bears with their preferred bowls of porridge and performs expected $O(n \log n)$ total "tastes." Prove that your algorithm is correct. Then, intuitively, but precisely, describe why the expected running time of your algorithm is $O(n \log n)$. *Hint: while this is not a sorting problem, your understanding of the sorts we've discussed in class may help when tackling this problem.*

   Solution: We can solve this in $n \log n$ time by taking an approach similar to quicksort. While we may not be able to judge the bowls' temperatures on a cardinal scale (i.e., degree temperature), we can still judge them on an ordinal/relative scale. Because temperature preferences are transitive, we can imagine a situation in which there are three bowls in ascending temperature (1, 2, 3); if bear A prefers bowl 2 and thinks bowl 3 is too hot, and bear B thinks bowl 2 is too hot, then we can infer than bear B also thinks bowl 3 is too hot.
   We start with a list of bears, with the first bear trying every bowl until he finds the one that's "just right". This bowl can then act as a pivot, around which the bowls that are "too cold" are placed to the left and the bowls that are "too hot" are placed to the right. Then we can move onto the next bear, who tries the pivot bowl. Based on this next bear's preference, we can identify which sub-list to search inside of. Once we have divided the list into two sub-lists, the previous bear and his corresponding bowl can be removed from the list. This process of each successive bear trying the previous bear's bowl and then searching for his preferred bowl within the appropriate sub-list and then sorting all other bowls in the sub-list around said preferred bowl is repeated until each bear is matched with his corresponding bowl and there are no more bowls left.
   This algorithm is log-linear because it is essentially quicksort which has a time complexity of $n \log n$. For each iteration in the list of n bears, we are performing an ordinal sort, which is determined by choosing a midpoint and dividing the list into two sub-lists.