Aadv(ylw4sj)

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: list collaborators's computing IDs

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.
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 $\Theta(log(n))$.
3. Prove that your algorithm above finds the correct answer. *Hint: Do induction on the size of the input.*

Solution:

1. Imagine the databases stored as arrays. First, query both database 1 and database 2 using k = n/2 and store the medians as m_1 and m_2. The overall database median is a value no greater than $max(m_1, m_2)$ and no smaller than $min(m_1, m_2)$. If $m_1 < m_2$, then query the right half of database 1 and the left half of database 2. Else, query the left half of database 1 and the right half of database 2. Recursively find the median of these subarrays until there is only one element left in subarray 1 and subarray 2. These two elements represent the n^{th} and $n^{th} + 1$ highest salary. Compare them and find the smaller value, that value is the overall median.
2. Let $T(n)$ be the total number of queries performed. Every time we call the algorithm, we use two queries and reduce the size of the problem by half, so $T(n) = T(n/2) + 2$. Using

the unrolling method:
$T(n/2) = T((n/2)/2) + 2 = T(n/4) + 2$, so $T(n) = T(n/4) + 4$
$T(n/4) = T((n/4)/2) + 2 = T(n/8) + 2$, so $T(n) = t(n/8) + 6$
Pattern: $T(n) = T(n/2^i) + 2i$
Base Case: $T(1) = 2$ since there is 1 element in each database and the algorithm queries both for the median element and returns the smaller value. $n/2^i = 1 \rightarrow \log n = i$
T(n) = T(n/2$^{\log n}$) $+ 2\log n = T(1) + 2\log n = \Theta(log(n))$

3. Claim: The algorithm finds the correct answer for databases of size $n \in \mathbb{N}$.

   Base Case: For n = 1, each database contains one element. If database 1 = [5] and database 2 = [7], the algorithm would query both databases for the value stored at index $n/2$, compare the values, and return the smaller element. This correctly returns 5 as the overall median.

   Inductive Step: Suppose the algorithm is true for $n = k$, it is also true for $n = 2k$. Following the algorithm, we first determine the median of each database and store those values as m_1 and m_2. There are then two cases: $m_1 < m_2$ and $m_1 > m_2$. For $m_1 < m_2$:
   We know the overall median has to be between m_1 and m_2, so we can use those values as pivots. We would look at the elements from index $m_1 + 1$ to $2k$ for database 1 since those values will be greater than m_1 and the elements from index 0 to $m_2 - 1$ for database 2 since those values will be less than m_2. The other half of both lists can be ignored, meaning the remaining halves of each list form an overall size of k, which is our base case. A similar argument holds for $m_1 > m_2$. Thus, the algorithm holds true for 2k.

   By the principal of induction, our algorithm is correct for databases of size $n \in \mathbb{N}$

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.

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:

1. Let the board be B. First, find the midpoint, $m = n/2$. LeftBound = 0 and RightBound = n. There are three cases to check.

   (a) If $B[m-1] \leq B[m] \geq B[m+1]$, return B.

   (b) If $B[m-1] > B[m]$, RightBound = $B[m-1]$.

   (c) If $B[m+1] > B[m]$, LeftBound = $B[m+1]$

   Recursively call the algorithm on the resulting array until there is one element in the array. That element is a local maximum.

   Proof of $O(\log n)$ Queries:
   Let T(n) be the total number of queries performed. Every time we call the algorithm, we use one query to ask for the value of m, one to ask for the value of $m+1$, and one to ask the value of $m-1$. The size of the problem is also reduced by half, so $T(n) = T(n/2) + 3$.
   Base Case: For $n = 5$, $T(5) = 3$
   We must show that $T(n) \leq c * \log n$ for all $n_0 \geq 0$.
   For $n_0 = 5, c = 2$:
   $T(n) \leq 2 * \log n$
   $T(5) \leq 2 \log 5$
   $3 \leq 2 \log 5$
   Therefore, there are $O(\log n)$ queries.

   Proof of Correctness:
   Claim: The algorithm correctly finds a local maximum for boards of size $n \in \mathbb{N}$
   Base Case: For $n = 5$, the board's sequence looks like: $0, 1, x, 1, 0$. For this example, $x = 3$. The algorithm first finds the midpoint, $m = 5/2 = 2$. The value at index two is x. Then, the algorithm checks the index $m-1$ and the index $m+1$, finds that the values stored there are both less than x, and correctly returns x.
   Inductive Step: Suppose this algorithm is true for n, it is also true for $2n$. Following the algorithm, we first find the midpoint m and check for three cases: $B[m-1] \leq B[m] \geq B[m+1]$, $B[m-1] > B[m]$, and $B[m+1] > B[m]$. For the first case, the local maximum is found and returned. For the second case, the algorithm is recursively called on first half of the board because the maximum must be greater than its surrounding elements, leaving the other half of the board to be ignored. This effectively reduces our board by half to a size of n, which is our base case. A similar argument holds for the third case. Thus, our algorithm holds true for $2n$
   By the principal of induction, our algorithm is correct for boards of size $n \in \mathbb{N}$.

2. In the worst case, the local maximum would not be found until there is only one element left. This means the local maximum would be located at either at the very left or the very right of the board. To achieve this, the opponent can continue to pick greater values for the right and smaller values for the left. For example, let the board have an index i with a value of 5. If asked, the opponent should pick a number > 5 for index $i+1$ and a number < 5 than $i-1$. They player will be forced to search the board in $\Omega(\log n)$ queries.

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 $O(n^2)$.

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:

1. Put the bears in a line and the bowls in a line. Have the first bear taste the bowls one by one until a bowl is "just right." Repeat this process until there are no bears left. This is $O(n^2)$ total tastes because each bear has to taste every bowl. The algorithm is correct because given a bear b, there exists a bowl that matches b. By having b try all the bowls until the correct one is reached, it guarantees that b will encounter the correct bowl. This argument holds true for other bears and bowls of porridge.

2. Put the bears in a line and the bowls in a line. Pick a random bowl, $bowl_1$ and have each bear taste it. If the bear says it's too cold, then put the bear in subset 1. If the bear says it's too hot, then put the bear to the subset 2. There will be one bear $bear_1$ who says it's "just right." After the bears have all been partitioned, pick another random bowl. Have $bear_1$ taste the bowl; if the bowl is too cold then call the algorithm on subset 1. Else, call the algorithm on subset 2. The bowl will be correctly matched with a bear, $bear_2$ and the bears will continue to be partitioned. Recursively call the algorithm until all bears and bowls are matched. The runtime is $O(n \log n)$ because this uses a similar sorting method as Quicksort, which also has a runtime of $O(n \log n)$. In addition, using the tree method we see that the problem divides so that the time complexity is $O(n) + 2 * O(n/2) + 4 * O(n/4)... = O(n \log n)$