CS4102 - ALGORITHMS - SPRING 2022

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

AADV(MST3K)

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.*

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

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.

board is shown in Figure 1. [We will require that all values in the list, excepting the first and last pairs, be unique.]

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.

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.*