CS4102 - ALGORITHMS - SPRING 2022

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

Aadv(rtg5xkh)

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: ra9ha

Sources: Cormen, et al, Introduction to Algorithms.

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:
   I would design a recursive implementation.
   Base Case: *when we've eliminated all elements except 1 in each list. When this is the case, we know that those two values are the middle values and we will take the average of them and return it as the median.*

   Recursive case: *Find the median salary of each school's remaining salaries in consideration (the $ceiling(k/2)th$ salary for an odd list, and the average between the $ceiling(k/2)th$ and the $floor(k/2)th$ salary for an even list). I'll call these values medh and medm for each respective school. If medh $==$ medm, return that number as the median. If medh $>$ medm, eliminate all elements in Harvard's list that are greater than medh and eliminate all of the elements in MIT's list that are less than medm. And vice versa.*

   Time Complexity: *since we cutting our search space in half at each recursion (by eliminating half of the values in each list that we know are not the middle values), the time complexity is $2\log_2 n$.*

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:
   $T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$, $T(1) = 1$
   Using the Master Theorem, $a = 1, b = 2, and f(n) = \Theta(1)$. $\log_2(1) = 0$, and $n^0 = 1$. $f(n) \in \Theta(1)$, so case 2 applies.
   Thus $T(n) \in \Theta(log(n))$

3. Prove that your algorithm above finds the correct answer. *Hint: Do induction on the size of the input.*
   Solution:
   Base case: n = 1 (1 salaries in each database): let's say Harvard = [100] and MIT = [200]. On the first recursion, the base case will be hit and the algorithm will simply return the average of the two salaries, 150. This is the correct median for this case.
   Assume the algorithm finds the correct median for some n. Then, it must find the correct median for the 2 databases of size $n+1$ because the algorithm will reduce the size of the values in those databases down to smaller collections for which we know the algorithm will work. Since we eliminate only the elements of one list that are lower than the smaller of the two medians and only the elements of the other list that are greater than the larger median, we can be sure that we are not removing the two middle elements in the overall database that will reveal the true median. Thus, within each divided sublist the two middle points will always be maintained, regardless of the sizes of the list or the number of divisions. Thus the algorithm will work for a size of $n+1$ and the inductive step holds. By the principle of induction the algorithm will work for any input size.

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:
   Base Case: the remaining list only contains 1 element. That element is a relative max, return it.
   Recursive case: find the middle element, $mid = board[n//2]$. Then check $midl = board[n//2-1]$ and $midr = board[n//2+1]$. If $mid > midl$ and $mid > midr$, return mid because it is a relative maximum. Else if $midl > mid > midr$, recurse on the left half of the list. Else, recurse on the right half of the list.
   This recursive strategy will repeat until a relative max is found.
   The runtime will be $3\log(n)$ because at each recursion, we do 3 queries and cut the list size in half. The recurrence relation can be given as:

$$T(n) = T(\frac{n}{2}) + 3$$

   Using the master Theorem, $a = 1, b = 2, and f(n) = 3$. $\log_2(1) = 0$, and $n^0 = 1$. $f(n) \in \Theta(1)$, so case 2 applies. Thus, $T(n) \in \Theta(logn)$.
   We can be sure this algorithm is correct because it clearly works in a base case where one value is identified and each of it's adjacent values are less than that value. The algorithm will simply return that value. And in any set of 3 queries in which a relative maximum is not identified, the algorithm will recurse on a subproblem just the same as it started with the original problem in such a way that a relative max must always be in the new range of values. Thus it will always be able to hit the base case eventually and return a relative max (a location of a castle).

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 opponent would always need to have a setup such that there is only one relative maximum in the whole board.
   For each of the algorithm's set of 3 queries (mid, midl, and midr), the opponent could always reply with values where midl ¡ mid ¡ midr, until eventually they would reach a point where they have to reveal the relative max (i.e. that spot is the only possible spot left for a relative maximum). The algorithm would need to split the search space in half over and over until the search space ended up as a small constant (between 1 and 3) and the opponent had no choice but to reveal a valid relative max).
   To explain a more general case, For any query, the opponent could select a value such that they would leave a possibility of a relative maximum in the largest part of the board that is unguessed and bounded by two revealed squares or a query and a revealed square. The opponent will delay the opponents ability to guess the maximum for the longest possible time with this approach. The best guessing strategy possible will eliminate half of the board with each constant set of queries, and clearly any other strategy will be even slower than this. Thus $\Omega(logn)$ queries are required by any algorithm to find a castle in the worst case due to the dynamic allocation of square values by the opponent.

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:
   This algorithm would have each bear taste each bowl until they find one for which they respond that the temperature is "just right". In this algorithm, the first bear will taste at most n bowls, the second bear will taste at most $n-1$ bowls, and so on (because after each bear finds their bowl we can remove that bowl). Thus, in the worst case each bear will, on average, taste n / 2 bowls until they find theirs (because the first bowl will taste n bowls and the last bear will not need to taste any because there will only be one left). Thus, the time complexity is $\frac{n}{2} * n$, which is $O(n^2)$.
   *We can be sure this algorithm is correct because we will iterate through each bear and have them taste every single bowl left until they find theirs. For any given bear that has not yet found their bowl, the collection of bowls that have not yet been claimed **must** contain it because each bear has exactly one bowl that is the correct temperature. Thus the algorithm will correctly identify all bowls for any number n.*

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:
   A randomized algorithm to solve this problem would first randomly select a bear to taste every bowl. We will then divide the list of bowls into ones that the bear said were too cold, C and a list of bowls that the bear said were too hot, H, while the partition element is the bowl that the bear said was just right. We will then have each remaining bear taste the partition bowl, and if the bear thinks the partition bowl is too hot, we will place them in a list of bears BC which will be associated with C. Else if a bear thinks the bowl is too cold we will place them in a list BH which will be associated with H. Then, we will repeat this same process in C with all the bears in BC, and in H with all the bears in BH. (i.e. will choose a random bear in BH to taste every bowl in H, divide into 2 sub lists based on the random bear's opinion, and then have every other bear in BH taste the partition bowl...). We will recurse on each sublist with this strategy until we hit a base case, which would be when only one bear and one bowl are left in a given sublist.
   <u>*Correctness:*</u>
   We can be sure that this algorithm is correct because the randomly chosen bear can determine which bowls are colder and which are hotter than their bowl. Thus, when we divide the bowls and assign bears to one side based on their preference of the partition bowl, we know that the bear will be with the sublist that contains their bowl. Since we repeat this same strategy in every step, every bear will always be assigned to a sublist that contains their bowl until either they are the next randomly

selected bear or they are the only bear left. In either case, they will be matched with their bowl.
Time Complexity:
Just like we saw with the probability of the partition element in quicksort being the worst possible every time, the probability that each randomly selected bear's bowl is either the hottest or the coldest is extremely unlikely. Although in this case, which is the very worst case, the algorithm would run in quadratic time, the expected runtime will be $\Theta(nlogn)$*. We can show this with an approximate recurrence relation for the algorithm assuming that the partition bowl will, on average, be somewhere near the middle of the list:*
$T(n) = 2T(\frac{n}{2}) + 2n$
This is the relation because at each recursion, we divide the list into 2 (hopefully somewhat equal parts), and then we have 1 bear taste every bowl and every bear taste 1 bowl (each of which are n steps, hence the $2n$*). Using the master theorem we can see that this relation is* $\Theta(nlogn)$*. For the same reason that unbalanced partitions are very unlikely in quicksort, the bowl partitions producing many uneven sublists is also very unlikely. This means we can accept this recurrence relation as the expected runtime for the randomized bears and porridge algorithm (probably with some extra constants). Thus we can conclude that the expected runtime is* $\Theta(nlogn)$*.*