CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, FEBRUARY 18, 2022 AT 11:30 PM VIA GRADESCOPE

A3BASIC(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' computing IDs

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

PROBLEM 1 *QuickSort*

1. Briefly describe a scenario when Quicksort runs in $O(n \log n)$ time.
   Solution: When by luck or by using an algorithm, the median value is found by Partition for *every* call to Quicksort.

2. For Quicksort to be a stable sort when sorting a list with non-unique values, the Partition algorithm it uses would have to have a certain property (or would have to behave a certain way). In a sentence or two, explain what would have to be true of Partition for it to result in a stable Quicksort. (Note: we're not asking you to analyze or explain a particular *implementation* of Partition, but to describe a general behavior or property.)
   Solution: To maintain stability, Partition must make sure that duplicate values are still in the same relative order to each other as elements are swapped around to put the partition value in its correct location. (Note: different implementations of Partition vary in this. If you're curious, the one in the slides, we need to change $<$ to $\leq$ to make it handle duplicate values correctly, but this breaks stability when there is a second element with the value of the pivot value.)

PROBLEM 2 *QuickSelect and Median of Medians*

1. When we add the median-of-medians method to QuickSelect in order to find a good pivot for QuickSelect, name the algorithm we use to find the median value in the list of medians from the 5-element "chunks".
   Solution: QuickSelect (called recursively inside QuickSelect itself).

2. Let's say we used the median-of-medians method to find a "pretty good" pivot and used that value for the Partition we use for Quicksort. (We're *n*ot using that value with QuickSelect to find the real median, but instead we'll just use this "pretty good" value for the pivot value before we call QuickSort recursively.) Fill in the blanks in this recurrence to show the time-complexity Quicksort if the size of the two sub-lists on either side of the pivot were as uneven as possible in this situation:

   $$T(n) \approx T(??) + T(??) + \Theta(n)$$

   Replace each "??" with some fraction of n, such as $0.5n$ or $0.95n$ etc.
   Solution: $T(n) \approx T(0.3n) + T(0.7n) + \Theta(n)$
   The median of medians method will find a "pretty good" pivot value that is guaranteed to be larger or smaller than 30% of the items in the list, so the most uneven split would be a 30%/70% split.

PROBLEM 3 *Other Divide and Conquer Problems*

1. What trade-off did the arithmetic "trick" of both Karatsuba's algorithm allow us to make, compared with the initial divide and conquer solutions for the problem that we first discussed? Why did making that change reduce the overall run-time of the algorithm?
   Solution: It was able to solve one less subproblem. The cost of combining grew, but only by a constant factor. This improved the order-class as we can see by applying the Master Theorem to the two recurrences.

2. Would it be feasible (without reducing the time complexity) to implement the closest pair of points algorithm from class by handling the points in the runway first, and then recursively solving the left and right sub-problems? If your answer is "no", briefly explain the reason why.
   Solution: No. You need the best value from the left and right subproblems to determine the width of the runway.

3. In the closest pair of points algorithm, when processing points in the runway, which of the following are true?

   (a) It's possible that the pair of points we're seeking could be in the runway and both points could be on the same side of the midpoint.

   (b) The algorithm will have a worse time-complexity if we needed to check 50 points above a given point instead of 7 (as we did in class).

   (c) The algorithm will have a worse time-complexity if we needed to check $\sqrt{n}$ points above a given point instead of 7 (as we did in class).

   Solution: The first is not true. If they were on the same side, that pair would have been found when solving the left or right subproblem, which happens before we process the runway. The second is false, because as long a processing the runway is $\Theta(n)$ we get the recurrence that makes the algorithm log-linear. For the same reason, the third item is ~~false~~ true. This would make the combine step $\omega(n)$.

PROBLEM 4 *Lower Bounds Proof for Comparison Sorts*

In class, we saw a lower-bounds proof that general comparison sorts are always $\Omega(n \log n)$. Answer the following questions about the decision tree proof that we did.

1. What did the internal nodes in the decision tree represent?
   Solution: A comparison between two elements in the list being sorted.

2. What did leaf nodes of the decision tree represent?
   Solution: A permutation of the elements in the list, i.e. an answer to the sorting problem, what order the elements should be in.

PROBLEM 5 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added. You should only submit your `.pdf` and `.tex` files.