CS4102 - ALGORITHMS - SPRING 2022

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

A3BASIC(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 *QuickSort*

1. Briefly describe a scenario when Quicksort runs in $O(n \log n)$ time.

   Solution: *Quicksort will run in $O(n \log n)$ time in the best case, when for each value, p, on which we partition the list (or sublist), p happens to be the median of the sublist the algorithm is currently working on.*

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: *The Partition algorithm would need to treat values that are later on in the list and $==p$ as "greater than" p, and it would need to treat values that come sooner than p and $==p$ as "less than" p. Thus the sory would be stable because all duplicate values would remain in their original order after the sort is done.*

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 (again, recursively).*

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:**

$$T(n) \approx T(.7n) + T(.3n) + \Theta(n)$$

   Since the median of medians algorithm will find a value that is within the middle 40% of values, a 30% and 70% split between the two sublists is the most uneven that is possible.

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:
   Using substitution and rearranging terms, Karatsuba's algorithm allowed us to only need to make one multiplication in each subproblem (instead of the 2 multiplications needed in D & C). His strategy used more additions to make this possible. This reduced the overall runtime of the algorithm because multiplications are much more expensive than additions.

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. The width of the runway is dependent on the minimum distance between the closest pair of points on the left and the right, so it would not be possible to do that step first.

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.
   Solution:
   True. The algorithm would have already found that pair of points if they were close enough together and on the same side of the midpoint, however if they represent the true minimum they would be found again by the step that finds the minimum in the runway. Thus the minimum of the left, right, and runway would just be that pair.
   (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).
   Solution:
   False. 7 and 50 are both constants so they will not influence the $O/\Theta/\Omega$ time complexity of the algorithm, even if they may add a small amount of time to the algorithm in reality. Thus the increase from 7 to 50 will not make the time complexity of the algorithm worse.
   (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:
   True. This would make the time complexity worse because $\sqrt{n}$ is not a constant. It grows as n grows, thus it must be added to the time complexity, and in this case it would make the algorithm slower.

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:
   Each internal node represented a comparison that the sorting algorithm might need to execute.

2. What did leaf nodes of the decision tree represent?
   Solution:
   Each leaf node represented a permutation of the elements that need to be sorted.

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.