ICS 311: Algorithms Spring 2025

Problem Set 7

Prof. Nodari Sitchinava *Due: Friday, Feb 28, 2025 at 4pm*

You may discuss the problems with your classmates, however **you must write up the solutions on your own** and **list the names** of every person with whom you discussed each problem.

Start **every** problem on a separate sheet of paper, with the exception of Problems 1 (Peer credit assignment) – Problem 1 can be on the same page as any other problem. Any problem that starts on the same sheet of paper as some other problem will receive 0 points!

1 Peer Credit Assignment (1 point extra credit for replying)

Please list the names of the other members of your peer group for this week and the number of extra credit points you think they deserve for their participation in group work.

- You have a total of 60 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- ***You cannot allocate any points to yourself!*** Points allocated to yourself will not be recorded.

2 k-th biggest element via MEDIAN (30 pts)

Suppose you have a subroutine MEDIAN($A[1..n]$) that returns the median of a set of n items (i.e., the value of the $\lfloor (n+1)/2 \rfloor$-th smallest) in $O(n)$ time. Design an algorithm that uses MEDIAN($A[1..n]$) to find the k-th largest element (for an arbitrary k) in $O(n)$ time. Write down the pseudocode, analyze its running time and explain why it correctly returns the k-th largest element.

3 Sorting larger numbers (30 pts)

Show how to sort n integers in the range $[0, (n^5 - 1)]$ in $O(n)$ time and at most $O(n)$ space. Write down the pseudocode and justify why your algorithm correctly sorts any array of n integers in the above range and why it runs in $O(n)$ time. *Hint: Read CLRS section 8.3 and understand how Radix sort works.*

4 Lower bound for sorting partially pre-sorted arrays (40 pts)

Consider the following problem. You have an array A of size n that you would like to sort. However, someone was nice enough to sort parts of it for you already. In particular, every t consecutive elements are already sorted, i.e., the subarrays $A[1..t]$, $A[(t+1)..2t]$, $A[(2t+1)..3t]$, ..., $A[(n-t+1)..n]$ are already sorted.

(a) (20 pts) Write down the pseudocode for an algorithm that sorts such an array in $O(n \log(n/t))$ time. Explain why your algorithm is correct and analyze its running time.

(b) (20 pts) Prove the matching $\Omega(n \log(n/t))$ lower bound for sorting such a presorted array using a comparison-based sorting algorithm. *Hint: Think about how many different permutations are possible in such an array.*

5 Smallest depth in the decision tree (OPTIONAL - 0 pts)

What is the smallest possible depth of a leaf in a decision tree for a comparison-based sorting algorithm on input of size n. Use **precise number**, not the big-O notation.

6 Lower bound for k-sorted arrays (OPTIONAL - 0 pts)

We say that an array $A[1..n]$ is ***k*-sorted** if it can be divided into k blocks, each of size n/k, such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted. For example, the following array is 4-sorted:

1	2	4	3	7	6	8	5	10	11	9	12	15	13	16	14

For this problem, you may assume that k divides n evenly.

(a) (0 pts) Write down the pseudocode for an algorithm that completely sorts an already k-sorted array of n elements in $O(n \log(n/k))$ time. Write down the pseudocode, analyze its running time and explain why it correctly sorts a k-sorted array.

(b) (0 pts) Prove that *any* comparison-based algorithm to completely sort a k-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case, i.e., that your algorithm in part (a) is asymptotically optimal? *Hint: It is not sufficient to combine lower bounds to sort individual blocks! Think about how many different permutations are possible in a k-sorted array.*

(c) (OPTIONAL - 0 pts) Prove that *any* comparison-based algorithm requires at least $\Omega(n \log k)$ comparisons in the worst-case to k-sort an unsorted array of n elements. Can you design an algorithm that k-sorts an unsorted array that runs in $O(n \log k)$ time?