CSC373 Summer '22
Assignment 1

Due Date: June 2, 2022, 11:59pm ET

Instructions

1. Typed assignments are preferred (e.g., PDFs created using LaTeX or Word), especially if your handwriting is possibly illegible or if you do not have access to a good quality scanner. Either way, you need to submit a single PDF named "hwk1.pdf" on MarkUS at `https://markus.teach.cs.toronto.edu/2022-05`.
2. You will receive 20% of the points for a (sub)question when you leave it blank (or cross off any written solution) and write "I do not know how to approach this problem." If you leave it blank but do not write this or a similar statement, you will receive 10%. This does not apply to any bonus (sub)questions.
3. You may receive partial credit for the work that is clearly on the right track. But if your answer is largely irrelevant, you will receive 0 points.

Q1 [20 Points] Decide the Candidate(s)!

Imagine you are the election officer of a residential township with n eligible voters. There has already been a round of preliminary voting in which each voter has given you the name of *one* person who they would like to see run as a candidate in the upcoming election. As a result, you have a list of n names (a name can, of course, repeat multiple times in this list – especially the popular ones!). Any person who receives (strictly) more than $n/3$ votes in the preliminary voting is to be declared a candidate. Note that hence, there can only be $0, 1$, or 2 candidates as a result of preliminary voting. Your task as the election officer is to figure out if there even is someone who is to be declared a candidate, and if so, who the candidate(s) is/are.
Note that the brute-force way to do this takes $O(n^2)$ time: for every name on the list, you run through the entire list, count how many times it appears on the list, and check if the count is strictly larger than $n/3$.

(a) [10 Points] Give a divide-and-conquer algorithm that solves this problem in $O(n \log n)$ time.

(b) [5 Points] Analyze the running time of your algorithm (and provide the recurrence relation).

(b) [5 Points] Briefly justify why your algorithm is correct.

Q2 [20 Points] Treasure hunt!

A treasure hunter has a map of n treasures. For each treasure i, the treasure hunter assigns two integer valued parameters, namely the easiness x_i to obtain the treasure, and its value y_i. The treasure hunter is organizing their trip and wants to identify treasures that are valuable to them. Specifically, a treasure i is *valuable* if there is no other treasure j with $x_j \geq x_i$ and $y_j \geq y_i$. The treasure hunter wants to count the number of valuable treasures on their map.

(a) [10 Points] Give an efficient divide-and-conquer algorithm to count the number of valuable treasures. You may assume that two distinct treasures differ in at least one of the two parameters i.e., for distinct i and j, either $x_i \neq x_j$ or $y_i \neq y_j$.

(b) [5 Points] Analyze the running time of your algorithm (and provide the recurrence relation).

(c) [5 Points] Briefly justify why the algorithm given in part (a) is correct.

Q3 [20 Points] Two Classrooms Only!

Here is a variant on the problem of Interval Scheduling. Suppose we now have *two* classrooms available, and we want to schedule as many lectures as we can. As before, the input is a collection of intervals $[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n)$ where $n \geq 1$ and all $s_i < f_i$ are non-negative integers.
A schedule is now defined as a pair of sets (A_1, A_2), with A_1 and A_2 being the sets of lectures scheduled in classrooms 1 and 2, respectively. A schedule is said to be *valid* if it satisfies the obvious constraints:

- $A_1, A_2 \subseteq \{1, 2, \ldots, n\}$,
- $A_1 \cap A_2 = \emptyset$ (i.e., a lecture can only be scheduled in *one* classroom), and
- for all $i \neq j$ such that $i, j \in A_1$ or $i, j \in A_2$, lectures i and j do not overlap (i.e., the lectures scheduled in a classroom are compatible with each other).

(a) [8 Points] Design a greedy algorithm to solve this problem i.e., on input $[s_1, f_1), \ldots, [s_n, f_n)$, your algorithm produces a valid schedule with the maximum number of lectures.

(b) [2 Points] Analyze the running time of your algorithm.

(c) [10 Points] Prove the correctness of your algorithm.

Q4 [20 Points] Help Your Professor!

Your Professor has agreed to assign as much of his TAs' time as necessary to ensure that his course's Piazza page is monitored around the clock (24 hours a day, 7 days a week) throughout the term. Suppose that his TAs have given him their availabilities, specified as intervals $[s_i, f_i)$ where $s_i < f_i$ are the start and end times of each available time slot, over some suitable range of non-negative integers.
Your Professor would like to draw up a monitoring schedule with the following properties:

- at every point in time, there is at least one TA available to monitor the course Piazza page, i.e., there are no gaps in the schedule, although it's okay if there is overlap, and
- the TAs are not overworked, i.e., the schedule uses as few available time slots as possible to achieve the first property, so that there are enough TA hours left over to help out with marking.

Note that this problem is a "dual" to the interval scheduling problem: with the same kind of input – intervals $[s_1, f_1), \ldots, [s_n, f_n)$ – we are asked a complementary question – to find a subset of intervals that totally cover the range from the earliest start time to the latest finish time, with overlap allowed, using as few intervals as possible.

(a) [5 Points] Give an ordering of the intervals for which the natural greedy algorithm corresponding to this ordering does not always find an optimal solution – include a description of some specific input for your algorithm, show the solution found by the greedy algorithm, and justify that it is not optimal.

(b) [5 Points] Design a greedy algorithm that solves this problem, i.e., on input $[s_1, f_1), ..., [s_n, f_n)$, your algorithm either produces a valid schedule that uses the smallest number of intervals, or it correctly states that no such schedule is possible.

(c) [2 Points] Analyze the running time of your algorithm.

(d) [8 Points] Prove the correctness of your algorithm.

Bonus Question

Q5 [20 Points] Voltage Optimization!

You have recently branched out into manufacturing bulbs! Your R&D department has produced several different designs. They don't know how much voltage each design can withstand, but using some physics, they can compare different designs in terms of their maximum voltage.

You have, in front of you, n bulbs sorted in a *non-increasing* order of the maximum voltage they can withstand. Your village uses 120V. You want to find the index r such that bulbs 1 through r can withstand 120V, but bulbs $r + 1$ through n cannot. In $O(1)$ time, you can test any bulb at 120V. Your first thought is to do a binary search in $O(\log n)$ time, but the problem is that every time you test a bulb that *cannot* withstand 120V, the bulb burns out. You want to limit the number of bulbs wasted. Of course, you can do a linear search and solve the problem while wasting at most one bulb, but that is too slow.

Suppose you are willing to waste at most k bulbs, where k is a constant. Design an algorithm for this problem. What is the asymptotic number of bulbs your algorithm tests in the worst case as a function of k?

[Note: As a warm start, design an $O(\sqrt{n})$ algorithm for $k = 2$. To receive any partial credit, you must solve the problem for at least $k = 3$ with an $o(\sqrt{n})$ algorithm.]