Practice Mid-term

Name: _____

Fall 2019, CSCE629 Analysis of Algorithms — XX/XX/2019

Texas A&M U — Prof. Fang Song

Instructions (please read carefully before start!)

- This take-home exam contains 7 pages (including this cover page) and 4 questions. Total of points is 80.
- You will have till **XX/XX/2019** to finish the exam. You must work on your own, and no collaboration or help from any resources other than those made available in class (lecture notes, texts, homework problems, etc.) is permitted.
- Submit your solutions in PDF on Gradescope before the deadline, either scanned or typeset in LaTeX. If you choose to hand-write and scan, *print out this exam sheet and write your solutions on it*. Do your best to fit your answers into the space provided, and attach extra papers only if necessary. If you typeset in LaTeX, *use the provided TeX file*. No other formats are accepted.
- Your work will be graded on correctness and clarity. Make sure your hand writing is legible.
- Don't forget to write your name on top (or update the "\studentname" command in the TeX file)!

Grade Table (for instructor use only)

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
Total:	80	

1. *Short answers*. Answer the following, and briefly justify your answer.

   (a) (5 points) (Sample problem) You are working on a divide-and-conquer algorithm that given an input of size n, calls itself recursively on 8 subproblems of size $\lceil n/3 \rceil$. Dividing and combining take time $f(n)$. Can the total running time of the algorithm be $O(n^2)$?

   Solution: Answer: Yes.
   Justification: The recurrence is $T(n) = 8T(\lceil n/3 \rceil) + f(n)$. Since $\log_b^a = \log_3 8 < \log_3 9 = 2$. By Master's theorem, if $f(n) = n^2$, it grows polynomially faster than $n^{\log_b a}$. Moreover, $8f(n/3) \leq cn^2$ for $c = 8/9 < 1$ (regularity condition). Hence the solution to the recurrence is $O(n^2)$.

   (b) (5 points) You are given a directed graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges. Let G^R be the graph obtained by reversing the directions of all the edges in G. Prove or disprove: you can compute adjacency matrix of G^R given the adjacency matrix of G in $o(n^2)$ time.

   (c) (5 points) Given a set V of n points in the plane, let δ be the distance between two closest points in V. Consider the graph $G_{2\delta}$ with vertex set V and an edge (u, v) for every pair of points u and v in V at distance 2δ or less. Prove or disprove: $G_{2\delta}$ can have $\Omega(n \log n)$ edges.

   (d) (5 points) Rank the following functions by ascending order of growth. That is, find an arrangement of $g_1, g_2, \ldots$ such that $g_1(n) = O(g_2(n)), g_2(n) = O(g_3(n)), \ldots$. If two function satisfy $g_i(n) = \Theta(g_j(n))$, they can be in either order. Recall that $\log(\cdot)$ is base 2 logarithm and $\log_b(\cdot)$ is base b logarithm.

$$3^{\log n}, \log^{19} n, \sqrt{n}, 2^{\sqrt[2]{n}}, \log(n!), 4^{log_4 n}, n^{\log 7}.$$

2. Given an array of temperatures $t_1, \ldots, t_n$ measured on n days, the weather service would like to compare the biggest increase in temperature to the biggest decrease in temperature over the given period. Specifically, you would like to compute two numbers: (a) $\max_{up}$, the maximum over pairs $i < j$ of $t_j - t_i$ and (b) $\max_{down}$, the maximum over pairs $i < j$ of $t_i - t_j$. For example, for the array $[10, 0, 1, 2, 3, 4, -1]$, you would compute $\max_{up} = 4$ and $\max_{down} = 11$.

   (a) (20 points) Give a divide and conquer algorithm that computes these numbers in $O(n)$ time. [Hint: Cut the array into two and make a recursive call on each half. It may be helpful to have the recursive calls pass up some extra information (in addition to $\max_{up}$ and $\max_{down}$).]

(b) Give a linear-time algorithm for the same problem which makes a single pass through the data and uses only a *constant* amount of workspace (beyond the space needed to store the input).

3. (20 points) You have reached the inevitable point in the semester where it is no longer possible to finish all of your assigned work without pulling at least a few all-nighters. The problem is that pulling successive all-nighters will burn you out, so you need to pace yourself (or something).

   Let's model the situation as follows. There are n days left in the semester. For simplicity, let's say you are taking one class, there are no weekends, there is an assignment due every single day until the end of the semester (would you like that?), and you will only work on an assignment the day before it is due. For each day i, you know two positive integers:

   - $Score[i]$ is the score you will earn on the ith assignment if you do not pull an all-nighter the night before.
   - $Bonus[i]$ is the number of additional points you could potentially earn if you do pull an all-nighter the night before.

   However, pulling multiple all-nighters in a row has a price. If you turn in the ith assignment immediately after pulling k consecutive all-nighters, your actual score for that assignment will be $(Score[i] + Bonus[i])/2^{k-1}$. Design and analyze an algorithm that computes the maximum total score you can achieve, given the arrays $Score[1, \ldots, n]$ and $Bonus[1, \ldots, n]$ as input.

4. (20 points) (Another algorithm design problem)

Scrap paper – no exam questions here.