CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2019**

Practice Midterm Exam #1

Name: _____ *NetID:* _____

Instructions

1. Do not forget to write your name and NetID above.
2. The exam contains 4 problems worth 100 points in total *plus* one extra credit problem worth 10 points. You have 75 minutes to finish the exam. The exam is closed-book and closed notes.
3. **Note that problems appear on both odd and even numbered pages.** There is more than enough space to write down your solution for each problem below the problem itself. But if you ran out of space, you can also use the extra sheet at the end of the exam; if you do so, be clear about which problem you are solving.
4. Remember that you can leave a problem (or parts of it) entirely blank and receive 25% of the grade for that problem (or part). However, this should not discourage you from attempting a problem if you think you know how to approach it as you will receive partial credit more than 25% if you are on the right track. But keep in mind that if you simply do not know the answer, writing a very wrong answer may lead to 0% credit.

   The only **exception** to this rule is the extra credit problem: you do not get any credit for leaving the extra credit problem blank, and it is harder to get partial credit on that problem.
5. **You should always prove the correctness of your algorithm and analyze its runtime.** Also, as a general rule, avoid using complicated pseudo-code and instead explain your algorithm in English.
6. You may use any algorithm presented in the class as a building block for your solutions.

Suggestion: Leave the extra credit problem for last as it is harder than the rest and worths fewer points.

Problem. #	Points	Score
1	20	
2	25	
3	25	
4	30	
5	+10	
Total	100 + 10	

Problem 1.

(a) Determine the *strongest* asymptotic relation between the functions $f(n) = n^{\log \log n}$ and $g(n) = n^5$, i.e., whether $f(n) = o(g(n))$, $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \omega(g(n))$, or $f(n) = \Theta(g(n))$. Prove the correctness of your choice. **(10 points)**

(b) Use the *recursion tree* method to solve the following recurrence $T(n)$ by finding the *tightest* function $f(n)$ such that $T(n) = O(f(n))$. **(10 points)**

$$T(n) \leq 2 \cdot T(n/3) + O(n)$$

Problem 2. Consider the algorithm below for finding the smallest two numbers in an array A of size $n \geq 2$.

`FIND-SMALLEST-TWO`$(A[1:n])$:

1. If $n = 2$: return $(A[1], A[2])$.
2. Otherwise, let $(m_1, m_2) \leftarrow$ `FIND-SMALLEST-TWO`$(A[1:n-1])$.
3. If $A[n]$ is smaller than at least one of m_1, m_2, return $A[n]$ and $\min\{m_1, m_2\}$; otherwise, return (m_1, m_2).

We analyze `FIND-SMALLEST-TWO` in this question.

(a) Use *induction* to prove the correctness of this algorithm. **(15 points)**

(b) Write a recurrence for this algorithm and solve it to obtain a tight upper bound on the worst case runtime of this algorithm. You can use any method you like for solving this recurrence. **(10 points)**

Problem 3. You are given an *unsorted* array $A[1:n]$ of n real numbers with possible repetitions. Design an algorithm that in $O(n \log n)$ time finds the largest number of time any entry is repeated in the array A.

Example. For $A = [1, 7.5, 2, 5.5, 7.5, 7.5, 1, 5, 9.5, 1]$ the correct answer is 3 (1 and 7.5 are repeated 3 times).

(a) *Algorithm:* **(10 points)**

(b) *Proof of Correctness:* **(10 points)**

(c) *Runtime Analysis:* **(5 points)**

Problem 4. You are given a list of n items with positive values $v_1, \ldots, v_n$. The goal is to pick as many items as possible to maximize the total value of the items taken. However, you are not allowed to pick any two (or more) consecutive items. Design an $O(n)$ time dynamic programming algorithm to compute the maximum obtainable value under this constraint.

(a) *Specification of recursive formula for the problem (in plain English)*: **(7.5 points)**

(b) *Recursive solution for the formula and its proof of correctness:* **(10 points)**

(c) *Algorithm (either memoization or bottom-up dynamic programming):* **(5 points)**

(d) *Runtime Analysis:* **(7.5 points)**

Problem 5. [**Extra credit**] You are given an *unsorted* array $A[1 : n]$ of n distinct numbers with the promise that each element is at most k positions away from its correct position in the sorted order. Design an algorithm that sort the array A in $O(n \log k)$ time. **(+10 points)**

Extra Workspace