CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Practice Midterm Exam #1

Name: _____ *NetID:* _____

Instructions

1. Do not forget to write your name and NetID above, and to sign Rutgers honor pledge below.
2. The exam contains 4 problems worth 100 points in total *plus* one extra credit problem worth 10 points.
3. The exam should be done **individually** and you are not allowed to discuss these questions with anyone else. This includes asking any questions or clarifications regarding the exam from other students or posting them publicly on Piazza (**any inquiry should be posted privately on Piazza**). You may however consult all the materials used in this course (video lectures, notes, textbook, etc.) while writing your solution, but **no other resources are allowed**.
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 there is almost no 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 or homeworks as a building block for your solutions.

Rutgers honor pledge:

On my honor, I have neither received nor given any unauthorized assistance on this examination.

Signature:_____

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

Problem 1.

(a) Determine the *strongest* asymptotic relation between the functions

$$f(n) = \sqrt{\log n} \quad \text{and} \quad g(n) = \frac{n}{2^{(\log \log n)}},$$

i.e., whether $f(n) = o(g(n))$, $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \omega(g(n))$, or $f(n) = \Theta(g(n))$. Remember to prove the correctness of your choice. **(15 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. Return the two smallest numbers among m_1, m_2, and $A[n]$.

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.

Remember to *separately* write your algorithm (**10 points**), the proof of correctness (**10 points**), and runtime analysis (**5 points**).

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).

Problem 4. You are given a set of n items with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$. You are also given *two* knapsacks with sizes W_1, W_2. The goal is to pick a subset of items with maximum value such that we can place each of the chosen items in one of the two knapsacks with the restriction that the total weight of items in each knapsack should be smaller than its size. Design an $O(n \cdot W_1 \cdot W_2)$ time dynamic programming algorithm that outputs the maximum value we can obtain by picking a subset of items that fits the two knapsacks.

Remember to *separately* write your specification of recursive formula in plain English (**7.5 points**), your recursive solution for the formula and its proof of correctness (**10 points**), the algorithm using either memoization or bottom-up dynamic programming (**7.5 points**), and runtime analysis (**5 points**).

Example: For items with weights $[1, 4, 2, 5, 1]$ and values $[2, 3, 1, 10, 1]$, and knapsacks of size 4 and 3, the correct output is 6: this is achieved by picking items 1 and 3 in knapsack two and picking item 2 in knapsack one – note that we cannot fit item 4 in either knapsack as its weight is larger than the size of each one.

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