CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2021**

Midterm Exam #1

Due: Tuesday, March 2nd, 9:00am EST

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 5 problems worth 100 points in total plus one extra credit problem worth 10 points.
3. This is a take-home exam. You have until Tuesday, March 2nd, 9:00am EST to finish the exam.
4. The exam should be done **individually** and you are not allowed to discuss these questions with anyone. This includes asking any questions or clarifications regarding the exam from other students or posting them publicly on Piazza (any inquiry should be sent directly to the Instructor or 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**.
5. 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% as long as you are on the right track. But keep in mind that if you simply do not know the answer, writing a totally 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.
6. **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.
7. You may use any algorithm presented in the class or homeworks as a building block for your solutions.

Rutgers honor pledge: Please sign the Rutgers honor pledge below.

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

Signature:______________________________

Problem. #	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
6	+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. **(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)**

$$T(n) \leq 8 \cdot T(n/2) + O(n^3).$$

(You do *not* have to prove that your function is the tightest one.)

Problem 2. Consider the algorithm below for finding the total sum of the numbers in any array $A[1:n]$.

`TOTAL-SUM`$(A[1:n])$:

1. If $n = 0$: return 0.
2. If $n = 1$: return $A[1]$.
3. Otherwise, let $m_1 \leftarrow$ `TOTAL-SUM`$(A[1:\frac{n}{2}])$ and $m_2 \leftarrow$ `TOTAL-SUM`$(A[\frac{n}{2}+1:n])$.
4. Return $m_1 + m_2$.

We analyze `TOTAL-SUM` in this question.

(a) Use *induction* to prove the correctness of this algorithm. **(10 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 a collection of n integers $a_1, \ldots, a_n$ with positive weights $w_1, \ldots, w_n$. For any number a_i, we define the *bias* of a_i as:

$$bias(a_i) = |\sum_{j:a_j<a_i} w_j - \sum_{k:a_k \geq a_i} w_k|;$$

i.e., the absolute value of the difference between the weights of elements smaller than a_i and the remaining ones. Design and analyze an algorithm that in $O(n \log n)$ time finds the element that has the *smallest* bias. You can assume that the input is given in two arrays $A[1:n]$ and $W[1:n]$ where $a_i = A[i]$ and $w_i = W[i]$.

Examples:

- When $n = 5$, and $A = [1, 5, 3, 2, 7]$ and $W = [3, 6, 2, 8, 9]$, the smallest biased element is $a_2 = A[2] = 5$ with $bias(a_2) = |(3 + 2 + 8) - (6 + 9)| = 2$.
- When $n = 5$, and $A = [1, 2, 3, 4, 5]$ and $W = [8, 6, 5, 2, 6]$, the smallest biased element is $a_3 = A[3] = 3$ with $bias(a_3) = |(8 + 6) - (5 + 2 + 6)| = 1$.

(a) *Algorithm:* **(7 points)**

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

(c) *Runtime Analysis:* **(3 points)**

Problem 4. You are given three arrays $A[1:n]$, $B[1:n]$, and $C[1:n]$ of positive integers. The goal is to decide whether or not there are indices $i, j, k \in [1:n]$ such that $A[i] \cdot B[j] = C[k]$; in other words, is it the case that there are numbers in A and B whose multiplication belongs to C.

Examples:

- When $n = 3$, and $A = [1, 3, 4]$, $B = [2, 3, 5]$, and $C = [1, 3, 5]$, the answer is *Yes*, because for instance we have $A[1] \cdot B[3] = C[3]$ or $A[1] \cdot B[2] = C[2]$.
- When $n = 3$ and $A = [1, 3, 4]$, $B = [2, 4, 6]$, and $C = [7, 9, 11]$, the answer is *No*.

(a) Suppose all the numbers in C belong to the set $\{1, 2, \ldots, n^2\}$. Design and analyze an algorithm with **worst-case runtime** of $O(n^2)$ for the problem in this case. **(10 points)**

(b) Now suppose C can be any arbitrary array of n integers. Design and analyze a **randomized** algorithm with **expected worst-case runtime** of $O(n^2)$ for the problem in this case. **(10 points)**

Note: Actually, this problem also has a deterministic algorithm that runs in worst-case $O(n^2)$ time. But you do not need to design such an algorithm for this problem (although if you do, you will receive the full credit for both parts (a) and (b)).

Problem 5. Please solve the problems in **exactly one** of the two parts below.

Part 1 (dynamic programming): We want to purchase an item of price M and for that we have a collection of n different coins in an array $C[1:n]$ where coin i has value $C[i]$ (we only have one copy of each coin). Our goal is to purchase this item using the *smallest* possible number of coins or outputting that the item cannot be purchased with these coins. Design a **dynamic programming** algorithm for this problem with worst-case runtime of $O(n \cdot M)$. **(20 points)**

Examples:

- Given $M = 15$, $n = 7$, and $C = [4, 9, 3, 2, 7, 5, 6]$, the correct answer is 2 by picking $C[2] = 9$ and $C[7] = 6$ which add up to 15.
- Given $M = 11$, $n = 4$, and $C = [4, 3, 5, 9]$, the correct answer is that 'the item cannot be purchased' as no combination of these coins adds up to a value of 11 (recall that we can only use each coin once).

(a) *Specification of recursive formula for the problem (in plain English)*: **(5 points)**

(b) *Recursive solution for the formula and its proof of correctness:* **(7 points)**

(c) *Algorithm (either memoization or bottom-up dynamic programming):* **(3 points)**

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

Part 2 (greedy): We want to purchase an item of price M and for that we have an unlimited (!) supply of $\lceil \log M \rceil$ types of coins with value $1, 2, 4, \cdots, 2^i, \cdots, 2^{\lceil \log M \rceil}$. Our goal is to purchase this item using the *smallest* possible number of coins (it is always possible to buy this item by picking M coins of value 1). Design and analyze a **greedy** algorithm for this problem with $O(\log M)$ runtime. **(20 points)**

Examples:

- Given $M = 15$ (and so $\lceil \log M \rceil = 4$), the correct answer is 4 coins by picking one copy of each of the coins $8, 4, 2, 1$. Note that here we cannot pick the coin of value $2^{\lceil \log M \rceil} = 2^4 = 16$.
- Given $M = 32$ (and so $\lceil \log M \rceil = 5$), the correct answer is 1 coin by picking one copy of the coin $32 = 2^5 = 2^{\lceil \log M \rceil}$.

(a) *Algorithm*: **(5 points)**

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

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

Problem 6. [**Extra credit**] You are given two *unsorted* arrays $A[1:n]$ and $B[1:n]$ consisting of $2n$ *distinct* numbers such that $A[1] < B[1]$ but $A[n] > B[n]$. Design and analyze an algorithm that in $O(\log n)$ time finds an index $j \in [1:n]$ such that $A[j] < B[j]$ but $A[j+1] > B[j+1]$.

Hint: Start by convincing yourself that such an index j always exist in the first place. **(+10 points)**

Extra Workspace