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

Midterm Exam #1

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

Name: Ryan Coslove *NetID:* rmc326

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: R Cos

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)**

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{\sqrt{\log n}}{\frac{n}{2^{\log\log n}}} = 0 \qquad f(n) = O(g(n))$$

$$g(n) = \frac{n}{2^{\log\log n}} > f(n) = \sqrt{\log n} \qquad g(n) = \Omega(f(n))$$

Given $f(n) = O(g(n))$ and $f(n) \neq \Theta(g(n))$

then the strongest relation is

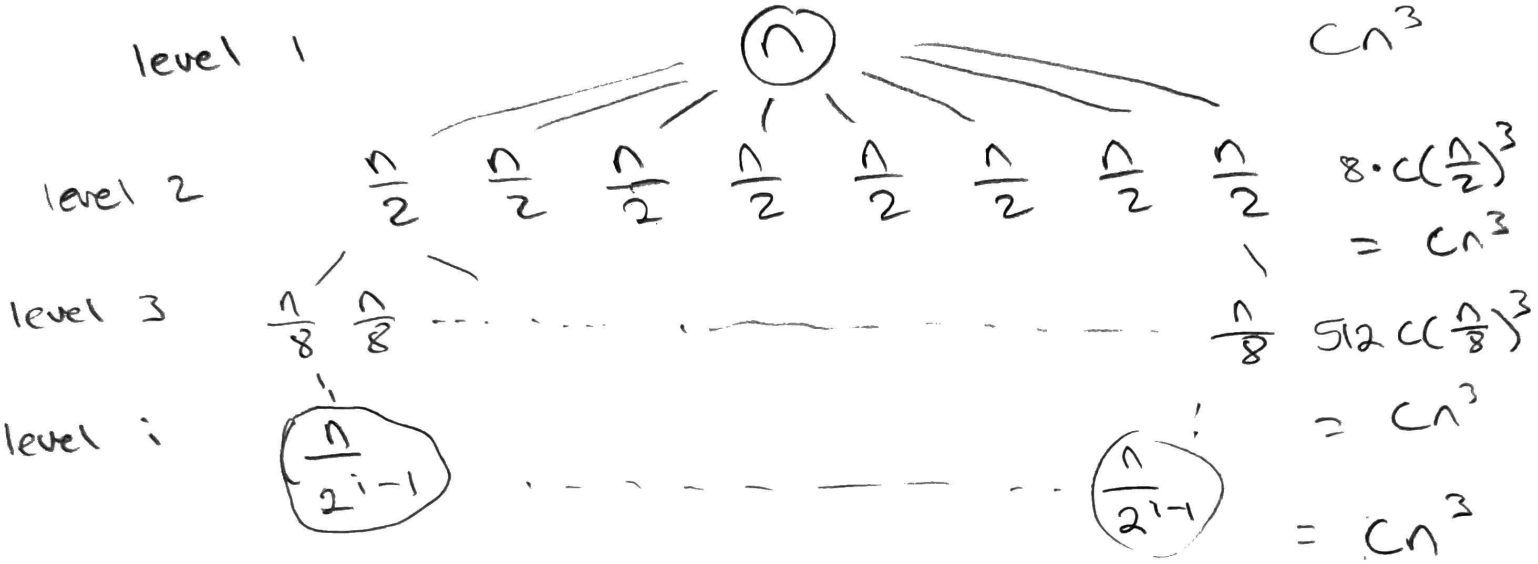
$$f(n) = o(g(n))$$

↖ little o

(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.)



$$T(n) \leq 8 \cdot T(\tfrac{n}{2}) + O(n^3)$$

Prove $T(n) = O(n^3)$

Replace $O(n^3)$ with $c \cdot n^3$ $c > 0$.

There are $\log_2 n + 1$ levels in the tree.

$$T(n) \leq c \cdot n^3 \sum_{i=0}^{\log_2 n} (\tfrac{8}{2})^i \leq c \cdot n^3 \sum_{i=0}^{\log_2 n} (\tfrac{8}{2})^i = O(n^3)$$

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)**

hypothesis: For any integer $n \geq 2$ and array A of size n, TOTAL-SUM($A[1:n]$) returns the total sum of the numbers in the array; the correct answer.

base: $n = 2$, Totalsum($A[1:1]$) $\rightarrow m_1$,
Totalsum($A[2:2]$) $\rightarrow m_2$
$m_1 + m_2 = 1 + 2 = 3$, which is the correct answer.

induction: Suppose induction is true for all $n \leq i$ where $i \geq 2$ and prove it for $n = i+1$.

First run Total-sum($A[1:\frac{n}{2}-1]$) Total-sum($A[\frac{n}{2}:n-1]$) and prove $m_1 + m_2$ gives the sum of A in $A[1:\frac{n}{2}-1]$ and $A[\frac{n}{2}:n-1] = A[1:i]$

Given that the sum among $A[1:\frac{n}{2}-1]$ exists within $A[1:\frac{n}{2}]$ and $A[\frac{n}{2}:n-1]$ exists within $A[\frac{n}{2}+1:n]$ and subsets of $\{m_1, m_2, A[n]\}$. so by returning these sums done in the algorithm, we obtain the answer.

(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)**

The algorithm recursively calls itself on an array of length $\frac{n}{2}$, twice, then spends $O(1)$ time to output the solution. $T(n)$ is worst case runtime of array length n. We have

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(1)$$

$$\Rightarrow T(n) \leq 2T\left(\frac{n}{2}\right) + c$$

$$S(k) = S(k-1) + \Theta(1)$$

$$S(0) = \Theta(1)$$

$$T(n) \leq S(k)$$

$$T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(1) \leq S(k-1) + \Theta(1)$$

$$= S(k)$$

$$S(k) = S(k-1) + \Theta(1) = \cdots = \underbrace{\Theta(1) + \cdots + \Theta(1)} = \Theta(n)$$

$$T(n) = \Theta(n)$$

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)**

a. Create an array B containing pairs [Ai, Wi]

b. Merge sort the array B in increasing order according to A[i].

c. store sum of weights in variable Sum, Wprev = 0, total weight Wsum = 0

d. for i to n in B

bias_i = absolute(Wsum - Wprev)

min = minimum{mini, bias_i}

Wsum = Sum - Wprev

Wprev = Wprev + Wi

e. output min

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

With B sorted, A_i is in increasing order so calculating bias for the current element we subtract all element weights left of the index by the weight of the elemats right of the index, and output their absolute value.

When looping through, we are calculating the biases and reassigning the minimum value bias.

By the end, we output the bias with the smallest value, which is the correct answer.

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

Merge sort generally takes $O(n \log n)$ time, and the loop takes $O(n)$ time, resulting in a total time $T(n) = O(n \log n)$

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)**

i, j, k ∈ [1 : n]
set count = 0
for i = 0 to n
  for j = 0 to n
    if A[i] · B[j] = C[k]
      then count = count + 1
      if count = n
        output yes
else
  no

We use a nested for loop to iterate through A[i] and B[j], checking at each index if their product is equal to a value of C[k], outputting yes or no.

(*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*)).

We will use universal hash table and chaining to avoid collisions, constructing a hash table of size n.

```
For i = 1 to n
    For j = 1 to n
        product = A[i] · B[j]
        If product = T.search(C[k])
            return true "yes"
        else
            insert product into hash table
    return "no"
```

Correctness: Suppose product has a duplicate and k is the first index where there exists a product within C[k]. In this case, we return T as true or 'Yes'. If there is no duplicate, we never find a product within C[k] and return no.

Run time: Creating the table takes $O(n)$ and running through the for loops takes $O(n)$ time each, so expected run time is $O(n^2)$

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)**

For integers $0 \le i \le n$ and $0 \le j \le M$:

$C(i,j)$: the minimum coins we can obtain by picking a subset of the first i items $\{1, \ldots, i\}$ when we have a M of size j.

By returning $C(n, M)$ we can solve the original problem because it's the max coins (value) we use to get size M, using fewest coins

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

$$C(n, M) = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ C(i-1, j) & \text{if } i = 1 \\ \max\{C(i-1, j-v_i), C(i-1, j)\} & \text{else} \end{cases}$$

base: $i=0, j=0$. $C(i,j) = 0$ which is the value achieved by using first 0 items or no size.

if $v_i > j$ $\therefore$ $C(i,j) = C(i-1, j)$.

if $v_i \le j$ $\therefore$ obtain $C(i-1, j-v_i)$ or $C(i-1, j)$

Pick max value of i to get j.

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

MemCoins (i,j):

2-D table
U[1:n][1:M]

1. If $i=0$ or $j=0$, return 0
2. IF $U[i][j] \neq$ 'undefined', return $U[i][j]$
3. Otherwise, if $v_i > j$, let $U[i][j] =$ MemCoins $(i-1, j)$;
4. Else, let $U[i][j] = \max\{$ MemCoins $(i-1, j-v_i)$, + MemCoins $(i-1, j)\}$
5. Return $D[i][j]$

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

There are n choices for i and M choices for j so there are $n \cdot M$ subproblems. Each takes $O(1)$ time, so runtime algorithm is $O(n \cdot M)$

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