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

Homework #2

Deadline: Thursday, October 3rd, 11:59 PM

Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question.
- You are allowed to discuss the homework problems with other students in the class. **But you must write your solutions independently.** You may also consult all the materials used in this course (notes, textbook, further reading, etc.) while writing your solution, but no other resources are allowed.
- Do not forget to write down your name and whether or not you are using one of your two extensions. Submit your homework on Canvas.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use counting sort to sort the array $A[1:n]$ with positive integer values smaller than M in $O(n+M)$ time".
- Remember to **always prove the correctness** of your algorithms and **analyze their running time**.

Problem 1. Your goal in this problem is to analyze the *runtime* of the following (imaginary) recursive algorithms for some (even more imaginary) problem:

(A) Algorithm A divides an instance of size n into 3 subproblems of size $n/2$ each, recursively solves each one, and then takes $O(n)$ time to combine the solutions and output the answer.

(B) Algorithm B divides an instance of size n into 2 subproblems, one with size $n/2$ and one with size $n/3$, recursively solves each one, and then takes $O(n)$ time to combine the solutions and output the answer.

(C) Algorithm C divides an instance of size n into 4 subproblems of size $n/5$ each, recursively solves each one, and then takes $O(n^2)$ time to combine the solutions and output the answer.

(D) Algorithm D divides an instance of size n in to 2 subproblems of size $n-1$ each, recursively solves each one, and then takes $O(1)$ time to combine the solutions and output the answer.

For each algorithm, write a recurrence that captures its runtime and *use the recursion tree method* to solve this recurrence and find the *tightest asymptotic* upper bound on runtime of the algorithm. **(25 points)**

Problem 2. You are given a permutation of $\{1,\ldots,n\}$ in an array $A[1:n]$. Design a *randomized* algorithm that finds an index of some *even* number in this array in only $O(1)$ *expected runtime*. You can assume that creating a random number from 1 to n can be done in $O(1)$ time. Note that the array A is already in the memory and your algorithm does not need to 'read' this array. **(25 points)**

Example: Suppose the input is $[3,2,4,5,1]$; then the algorithm can return either index 2 or index 3.

Problem 3. You are given an array $A[1:n]$ of n positive *real* numbers (not necessarily distinct). We say that A is satisfiable if after sorting the array A, $A[i] \geq i$ for all $1 \leq i \leq n$. Design an $O(n)$ time algorithm that determines whether or not A is satisfiable. **(25 points)**

Example: $[1.5, 4.1, 0.5, 5.3]$ is *not* satisfiable: after sorting we get $[0.5, 1.5, 4.1, 5.3]$ and $A[1] = 0.5 < 1$. But array $[1.5, 4.1, 2.4, 5.3]$ is satisfiable: after sorting we get $[1.5, 2.4, 4.1, 5.3]$ and for $i \in \{1,2,3,4\}$, $A[i] \geq i$.

Problem 4. Alice and Bob are delivering for the Pizza store that has n orders today. From past experience, they both know that if Alice delivers the i-th order, the tip would be $A[i]$ dollars, and if Bob delivers, the tip would be $B[i]$ dollars. Alice and Bob are close friends and their goal is to maximize the total tip they can get and then distribute it evenly between themselves. The problem is that Alice can only handle a of the orders and Bob can only handle b orders. They do not know how to do this so they ask for your help.

Design a dynamic programming algorithm that given the arrays $A[1:n], B[1:n]$, and integers $1 \leq a, b \leq n$, finds the largest amount of tips that Alice and Bob can collect together subject to their constraints. The runtime of your algorithm should be $O(n \cdot a \cdot b)$ in the worst case. **(25 points)**

Example: Suppose $A = [2, 4, 5, 1]$, $B = [1, 7, 2, 5]$, $a = 2$ and $b = 1$. Then the answer is 14 by Alice delivering $2, 3$ and Bob delivering 4 (so $4 + 5 + 5 = 14$), or Alice delivering $1, 3$ and Bob delivering 2 ($2 + 5 + 7 = 14$).

Challenge Yourself. A standard dynamic programming algorithm for Problem 4 requires $O(n \cdot a \cdot b)$ *space* in addition to the same runtime. The goal of this question is to improve this space bound and further extend the algorithm. **(0 points)**

(a) Design an algorithm for this problem that requires only $O(n + a \cdot b)$ space instead and still has the worst-case runtime of $O(n \cdot a \cdot b)$ (this part is rather standard and is not that challenging).

(b) Design an algorithm with the same space $O(n + a \cdot b)$ and time $O(n \cdot a \cdot b)$ that in addition to the value of the largest possible tip, can also output the set of deliveries Alice and Bob should take in order to achieve such a collection of tips (this part may be really challenging).

Fun with Algorithms. Recall that Fibonacci numbers form a sequence F_n where $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$. The standard algorithm for finding the n-th Fibonacci number takes $O(n)$ time. The goal of this question is to design a significantly faster algorithm for this problem. **(0 points)**

(a) Prove by induction that for all $n \geq 1$:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}.$$

(b) Use the first part to design an algorithm that finds F_n in $O(\log n)$ time.