CS 170 Algorithms
Spring 2009 David Wagner

MT2

PRINT your name: _____, _____
(last) (first)

SIGN your name: _____

PRINT your Unix account login: _____

Your TA's name: _____ Discussion section time: _____

Name of the person sitting to your left: _____ Name of the person sitting to your right: _____

You may consult any books, notes, or other paper-based inanimate objects available to you. Calculators and computers are not permitted. Please write your answers in the spaces provided in the test. We will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there. Unless specified otherwise, you do not need to justify your answers on this exam, and we will not grade any justification you may provide. When asked for answers using $O(\cdot)$ notation, provide the most specific answer possible (for example: do not write $O(n^3)$ if $O(n^2)$ is also correct).

You have 50 minutes. There are 4 questions, of varying credit (100 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

Do not turn this page until your instructor tells you to do so.

Problem 1	
Problem 2	
Problem 3	
Problem 4	
Total	

Problem 1. [Huffman coding] (25 points)

Suppose we have an alphabet with only five letters, A, B, C, D, E, which occur with the following frequencies:

letter	A	B	C	D	E
frequency	0.25	0.05	0.2	0.1	0.4

Use Huffman coding to find the optimal prefix-free variable-length binary encoding of this alphabet.

(a) Draw a binary tree that represents the optimal encoding. Make sure to label the leaves with the letters A, B, C, D, E.

(b) Fill in the table below with the binary encoding of each letter.

letter	encoding
A	
B	
C	
D	
E	

Problem 2. [Fill in the blank] (20 points)

Fill in the boxes below with the best answer. You don't need to justify your answer.

(a) Suppose that a data structure supports an operation ADD, so that any sequence of n ADD's takes at most $O(n \lg n)$ time in the worst case. Then the amortized running time of an ADD operation is [ ], while actual running time of a single ADD operation could be as much as [ ] in the worst case.

(b) Let $a_1, a_2, \ldots, a_n$ be a sequence of integers. Recall that a subsequence is a subset of these integers taken in order, i.e., $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ such that $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. A decreasing subsequence is one for which every integer is smaller than previous one, i.e., it is a subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ satisfying $a_{i_1} > a_{i_2} > a_{i_3} > \cdots > a_{i_k}$. In the longest decreasing subsequence problem, we are given the sequence $a_1, a_2, \ldots, a_n$, and we want to compute the length of the longest decreasing subsequence of $a_1, a_2, \ldots, a_n$.

Example: If we are given the sequence $5, 2, 4, 8, 3, 40$, then the longest decreasing subsequence is $5, 4, 3$. (Other decreasing subsequences include $5, 2$ and $8, 3$, but they are shorter.)

Let's build a dynamic programming algorithm for this problem. I will give you the definition of the subproblems. We will let $L(j)$ denote the length of the longest decreasing subsequence that starts with a_j (i.e., the longest decreasing subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ such that $i_1 = j$). Each $L(j)$, where $1 \leq j \leq n$, represents a single subproblem. Your job is to find a recursive relation that can be used to obtain a dynamic programming algorithm that runs in $O(n^2)$ time. You must use this definition of subproblems, but you do not need to justify your answer.

First, fill in a base case:

$$L(n) = \boxed{}.$$

Next, find a recurrence relation for $L(j)$, if $1 \leq j < n$:

$$L(j) = \begin{cases} \boxed{} & \text{if } a_j \leq \min(a_{j+1}, a_{j+2}, \ldots, a_n), \\ \boxed{} & \text{otherwise.} \end{cases}$$

Problem 3. [More scheduling] (25 points)

We have n jobs to schedule on a large cluster, where there are an effectively unlimited number of computers in the cluster. We are given a set of m "same-computer" constraints, where the ith constraint says that jobs $S[i]$ and $T[i]$ must be run on the same computer in the cluster (e.g., because these two jobs communicate with each other frequently and so need to be co-located). We are also given a set of k "different-computer" constraints, where the ith such constraint says that jobs $D[i]$ and $E[i]$ must not be assigned to the same computer (e.g., because they would interfere with each other). We want to determine whether it is possible to assign jobs to computers in a way that satisfies all of these constraints.

(a) Prof. Violet has the idea of using the Union-Find data structure to solve this problem, given $S[1..m]$, $T[1..m]$, $D[1..k]$, $E[1..k]$, and n. She helpfully fills out most of the algorithm for you, but she gets bored before she can finish writing down the full algorithm and she wanders off. Fill in the blank below to get a correct and efficient algorithm for this problem.

1. For $i := 1, 2, \ldots, n$:
2. Makeset(i).
3. For $i := 1, 2, \ldots, m$:
4. Union($S[i], T[i]$).
5. For $i := 1, 2, \ldots, k$:
6. If [ ], then return "Impossible".
7. Return "Possible".

(b) Estimate the total running time that this algorithm spends on lines 1–4 (ignoring lines 5–7), as a function of n, m, k. You may use $O(\cdot)$ notation. Justify your answer briefly.

(c) Estimate the total running time (all lines) of this algorithm, as a function of n, m, k. You may use $O(\cdot)$ notation. You don't need to justify your answer.

Problem 4. [Rental car scheduling] (30 points)

You're managing a branch office of Hurtz, the infamous car rental company. You just received a single Tesla Roader (a sporty electric car), and it is proving to be very popular: all sorts of customers have called up to request to reserve the Roadster for a range of days at some point in the future. You have the brilliant idea to run an auction, asking customers to specify a range of days and to bid on how much they are willing to pay to rent the Roadster for those days.

In particular, you want to plan a schedule for the Roadster for the next T days. On each of the next T days, there is a customer who has submitted a bid to rent the Roadster for a certain range of days starting on that day. In particular, for each $i \in \{1,2,\ldots,T\}$, there is a customer c_i who has offered to pay $P[i]$ dollars if you will rent him the Roadster for days $i, i+1, i+2, \ldots, E[i]$. (Note that on each day, there is only one customer who wants to start on that day.) If you accept customer c_i's offer, you won't be able to rent the Roadster out to anyone else during those days. For each customer, you must either accept or reject their bid, with no modifications (for instance, you can't give them the Roadster for fewer days than requested). You'd like to select a subset of bids that maximizes your total revenue, subject to the constraint that the Roadster can be rented to at most one person on each day. You may assume that $E[i] \leq T$ for every i.

Your job is to design an efficient dynamic programming algorithm to compute the maximum amount of money you can collect over days $1,\ldots,T$, given $P[1..T]$, $E[1..T]$, and T. The running time of your algorithm should be at most $O(T)$. Do not show the pseudocode of your algorithm—instead, just answer the following two questions. You do not need to justify your answers.

(a) Prof. Indigo suggests you identify T subproblems and define an array $A[1..T]$ so that $A[i]$ holds the solution to the ith subproblem—but then she absent-mindedly walks out of the room and leaves it up to you to work out the rest of the details of the algorithm. With this hint, specify the value of $A[i]$ precisely, in English:

$A[i] =$.

(b) Show us a recursive expression for $A[i]$ that can be used to obtain a dynamic programming algorithm with the stated running time. (You don't need to show us the base case(s). You don't need to justify your answer.)

$A[i] =$.