Basic Algorithms CSCI-UA.0310

Homework 6

Instructions

You do NOT need to submit your solutions to this homework.

Problem 1

Let $T(n)$ denote the running time of Approach 1 to find the nth Fibonacci number discussed in the lecture. Use strong induction to show that $T(n) = \Omega(2^{n/2})$.
Hint: Use the inequality $T(n) \geq T(n-2)$ which holds for all $n \geq 3$.

Problem 2

Given two strings $S[1 \dots n]$ and $T[1 \dots m]$, let $\mathrm{LCS}(n, m)$ denote the length of the longest common substring of $S[1 \dots n]$ and $T[1 \dots m]$. Note that unlike a subsequence, a substring is required to occupy consecutive positions within the original strings.

(a) Find the recursion that $\mathrm{LCS}(n, m)$ satisfies. Fully justify your answer.

(b) Identify the base cases for your recursion in part (a) and find their corresponding values. Justify your answer.

(c) Write the pseudo-code for the bottom-up DP algorithm to compute $\mathrm{LCS}(n, m)$.

(d) Find and justify the time complexity of your algorithm in the form of $\Theta(.)$.

Problem 3

Alice and Bob want to play the following game by alternating turns: They have access to a row of n coins of values $v_1, \dots, v_n$, where n is even. In each turn, a player selects either the first or the last coin from the row, removes it from the row, and receives the value of the coin. Alice starts the game.
Devise a dynamic programming algorithm to determine the maximum possible amount of money Alice can definitely win (assume that Bob will play in such a way to maximize the amount he gets). State a $\Theta(.)$ expression for the running time of your algorithm.

Problem 4

A palindrome is a non-empty string that spells the same forward and backward. As an example, "civic" is a palindrome. Given the string $S[1 \dots n]$, we want to find the length of the longest palindromic subsequence of S. For example, for the string "character", the answer is 5 since the longest palindromic subsequence is "carac".

(a) Let $P(i, j)$ denote the length of the longest palindromic subsequence of the string $S[i \dots j]$. Find the recursion that $P(i, j)$ satisfies. Justify your answer.

(b) Identify the base case(s) for your recursion in part (a) and find their corresponding value(s). Justify your answer.

(c) Write the pseudo-code for the bottom-up DP algorithm to compute $P(1, n)$.

(d) Find and justify the time complexity of your algorithm in the form of $\Theta(.)$.

Problem 5

Recall *the longest common subsequence problem* discussed in the lecture. Directly solve Problem 4 by using the longest common subsequence problem.

Problem 6

Consider the two-dimensional array $A[1 \dots m][1 \dots n]$, where each entry $A[i][j]$ is filled with a positive integer-valued reward. We start from the bottom leftmost corner, i.e., $A[0][0]$, and in each step, we are allowed to move to the right adjacent cell or to the top adjacent cell, until we reach to the top rightmost corner, i.e., $A[m][n]$. We collect the reward of each cell we step on. Let $\textsc{Max Reward}(m, n)$ denote the maximum amount of reward we can collect by starting from $A[0][0]$ and reaching $A[m][n]$.

(a) Find the recursion that $\textsc{Max Reward}(m, n)$ satisfies. Fully justify your answer.

(b) Identify the base cases for your recursion in part (a) and find their corresponding values. Justify your answer.

(c) Write the pseudo-code for the bottom-up DP algorithm to compute $\textsc{Max Reward}(m, n)$.

(d) Find and justify the time complexity of your algorithm in the form of $\Theta(.)$.

Problem 7

Given the array $A[1 \dots n]$ consisting of n distinct integers, devise a dynamic programming algorithm to output the length of the longest increasing subsequence of A, i.e., a subsequence of A whose elements are sorted in an increasing order. Find the running time of your algorithm.