Problem 7-1 (Maximize Sum/Product) 14 points

Suppose you are given an array $A[1,\ldots,n]$ of numbers, which may be positive, negative, or zero.

(a) (4 points) Let $S_{i,j}$ denote $A[i] + A[i+1] + \cdots + A[j]$. Use dynamic programming to give an $O(n^2)$ algorithm to compute $S_{i,j}$ for all $1 \leq i \leq j \leq n$, and hence compute $\max_{i,j} S_{i,j}$.

(b) (6 points) Let $L[j]$ denote $\max_{i \leq j} S_{i,j}$. Give a recurrence relation for $L[j]$ in terms of $L[1,\ldots,j-1]$. Use your recurrence relation to give an $O(n)$ time dynamic programming algorithm to compute $L[1\ldots n]$, and hence compute $\max_{i,j} S_{i,j}$.

(c) (2 points) Assume you use recursion (without memoization) to compute the answers to part (a) and part (b). Will both running times stay at $O(n^2)$ and $O(n)$, respectively, only one of them (which one?), or none?

(d) (4 points) Suggest appropriate modifications to your algorithm in part (b) to give an $O(n)$ algorithm to compute $\max_{i,j} P_{i,j}$, where $P_{i,j} = A[i] \cdot A[i+1] \cdots A[j]$. Assume that multiplication of any two numbers takes $O(1)$ time.

Problem 7-2 (Greatest Value Path) 8 points

Assume that you have an $n$ by $n$ checkerboard. You must move a checker from the bottom left corner (position $(1,1)$) square the board to the top right corner (position $(n,n)$) square. In each step you may either

- move the checker up one square, or
- move the checker diagonally one square up and to the right, or
- move the checker right one square.

If you move a checker from square $x = (i,j)$ to square $y = (i',j')$ you get $p(x,y)$ dollars. You are told all of the $p(x,y)$ a priori. The $p(x,y)$ may be negative, zero or positive. You want to get as much money as possible.

(a) (4 pts) Let $M[i,j]$ be the highest profit you can collect from position $(1,1)$ to $(i,j)$. Write a dynamic programming recurrence relation for $M[i,j]$ (do not forget the initial condition). Based on this recurrence relation, analyze the running time of the dynamic programming algorithm to compute $M[n,n]$.

(b) (4 pts) Give the “bottom-up” pseudocode for an efficient procedure `CheckerBoard(n)` for computing $M[n,n]$. 

PS7, Page 1
Problem 7-3 (Sending a Message down the Tree) 8 (+6) points

Suppose we need to distribute a message to all the nodes in a rooted (not necessarily binary) tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. For example, the minimum number of rounds it takes to distribute the message in the tree given below is 3.

Note that the order in which the messages are distributed matters. For example, in the above tree, if the root node sends the message to the right child in the first round, then the number of rounds will be 4.

Assume that a tree $T$ is given with nodes labeled $\{0, 1, 2, \ldots, n-1\}$ and the node 0 is the root of the tree. Further there is a two-dimensional $n \times n$ array $\text{Child}[i][j]$, where $k = \text{Child}[i][0]$ is the number of children of the node labeled $i$ and $\text{Child}[i][1], \text{Child}[i][2], \ldots, \text{Child}[i][k]$ denote the labels corresponding to the children of node $i$. The remaining entries in the array are $-1$.

(a) (4 points) Let $\text{Rounds}(i)$ be a function computing the minimum number of rounds it takes to distribute the message from node $i$ to all nodes in the subtree rooted at $i$. Give a recursive formula to compute $\text{Rounds}(i)$ as a function of $\text{Rounds}(j_1), \ldots, \text{Rounds}(j_k)$, where $j_1, \ldots, j_k$ denote the children of node $i$.

HINT: What is the order in which each of the children get the message?

(b) (4 points) Write the pseudocode for the recursion with memoization dynamic programming procedure for computing $\text{Rounds}(0)$.

(c) (4 points (Extra credit)) Analyze the running time of your algorithm.

Problem 7-4 (Max subtree) 10 points

Given an undirected rooted tree $T$ with $n$ nodes, with possibly negative weights $w(v)$ assigned to its vertices $v$, the weight of the tree is the sum of the weights of all the nodes in the tree. (The weight of the empty tree is 0.)

A subtree is any connected subgraph of a tree, including an empty tree as a pathological special case. For example, in the tree below the subtrees are $\emptyset$, $\{3\}$, $\{1\}$, $\{2\}$, $\{4\}$, $\{3,1\}$, $\{3,2\}$, $\{1,4\}$, $\{3,1,4\}$, $\{3,1,2\}$ and $\{3,1,2,4\}$, but not $\{3,4\}$.
The goal of the problem is to design a polynomial time algorithm to find a subtree (possibly empty) with maximum weight, and analyze its running time.

(a) (4 pts) Given a node $v$, let $T(v)$ be the complete subtree of $T$ rooted at $v$ (so $T = T(T.root)$), and $\text{WithRoot}(v)$ to be the maximum weight of all subtrees of $T(v)$ which must include the node $v$. For example, if the tree above, $\text{WithRoot}(1)$ is the maximum weight of sub-trees $\{1\}$ and $\{1,4\}$ (but not $\{4\}$ or the empty tree). Give the dynamic programming recurrence relation for $\text{WithRoot}(v)$ (don’t forget the base when $v$ is a leaf in $T$), and use it to analyze the running time of a dynamic-programming procedure to compute the values $\text{WithRoot}(v)$ for all nodes $v \in T$ (including with $T.root$).

(b) (4 pts) Given a node $v$, let $T(v)$ be the complete subtree of $T$ rooted at $v$ (so $T = T(T.root)$), and $\text{Total}(v)$ to be the maximum weight of all subtrees of $T(v)$ which must may or may not include the node $v$. Assume you already solved part (a) and computed all values $\text{WithRoot}(v)$.

Give the dynamic programming recurrence relation for $\text{Total}(v)$ (don’t forget the base when $v$ is a leaf in $T$), and use it to analyze the running time of a dynamic-programming procedure to compute the values $\text{Total}(v)$ for all nodes $v \in T$ (including with $T.root$, which gives the answer to the original problem). What is the running time of this algorithm?

(c) (2 pts) What will the running-time of the procedures in parts (a) and (b) be if we use standard recursion, and not dynamic programming?