CSC 270F 1999, Assignment 4, St. George Campus

Option 1.
Exercises required: Problems 1 and 2 below **only**.
Due: Wednesday December 1, 11:00 AM (in usual box for evening section, and at the beginning of lecture for day section).
Lateness: one day (Thursday December 2, 11:00 AM, in usual box); penalty: 20%.
Worth: 8%.

Option 2.
Exercises required: all problems below.
Due: Wednesday December 8, 5:00 PM (in lecture's box).
Lateness: one day (Thursday December 9, 5:00 AM); penalty: 20%.
Worth: 8%.

Problem 1: **Longest Increasing Consecutive Subsequence**

Consider the following "Longest Increasing Consecutive Subsequence" (**LICS**) problem: given a list of floating-point numbers $[v_0, v_1, \ldots, v_{n-1}]$, we want to find a consecutive subsequence of numbers $[v_i, v_{i+1}, \ldots, v_j]$ such that $v_i < v_{i+1} < \cdots < v_j$ and the number of elements in the subsequence $(j - i + 1)$ is as large as possible.

For example, in the list $[3.5, -6, 4.112, 1.6, 2.7, 7, -5.01]$, the second and third elements form a consecutive subsequence $[-6, 4.112]$ that is increasing $(-6 < 4.112)$ and whose length is 2, while the fourth, fifth, and sixth elements form a consecutive subsequence $[1.6, 2.7, 7]$ that is increasing $(1.6 < 2.7 < 7)$ and whose length is 3; it turns out that $[1.6, 2.7, 7]$ is the longest increasing consecutive subsequence for this example.

To solve the **LICS** problem using dynamic programming, we define an array $A[j]$ = the length of the longest increasing consecutive subsequence of $v_0, \ldots, v_j$ whose last element is exactly v_j (that is, the subsequence *must* have the form $v_i, \ldots, v_j$ for some $i \leq j$), for $0 \leq j < n$.

Part A:

Given the list $[3.12, 4, -2.07, 0.15, 4.25, 297, 14.5, -7]$ (that is, $v_0 = 3.12, v_1 = 4, \ldots, v_7 = -7$), complete the following table that stores the values of $A[j]$:

j	0	1	2	3	4	5	6	7
$A[j]$								

Briefly justify your answer.

Part B:

Write a *general* recurrence relation for $A[j]$, in the following way:

$A[0] =$

$A[j] =$ $\qquad\qquad\qquad$ for $j > 0$.

You **must** justify your answer.

Part C:

Write in paper (no implementation is needed) a dynamic programming function in C to determine the length of a longest increasing consecutive subsequence of a given list of numbers. Your function should take `list` (the list of numbers) and `n` (the number of elements in the list) as parameters and return the length of a longest increasing consecutive subsequence of `list`.

Briefly explain why the program is correct.

Problem 2: **Long Integer Addition**

Consider the *long integer addition* problem: We will represent a "long integer" with an array of characters: '0', '1', '2', ... '9'. The size of the array is the number of decimal digits in the integer. Given n numbers, $x_1, x_2, \ldots, x_n$, represented with arrays, we must compute their sum. However, we are not allowed to change the order of the numbers!

In our inefficient implementation, adding a number with a digits to a number with b digits *always* results in a number with

$$\max(a, b) + 1$$

digits, since there is a possibility of a carry. Thus, the *cost* of adding such numbers is $\max(a, b) + 1$.

Consider adding two numbers, one having d_1 digits and the other having d_2 digits. Here are examples of the cost of such an addition for different values for the ds:

d_1	d_2	Cost
3	5	6
2	2	3

Part A:

Consider adding three long integers, $x_1 + x_2 + x_3$, with sizes $d_1 = 1$, $d_2 = 5$, and $d_3 = 7$ (where d_i is the number of decimal digits in x_i). Compare the total cost of computing $(x_1 + x_2) + x_3$ with the total cost of computing $x_1 + (x_2 + x_3)$. Show your work.

Part B:

Suppose we want to add six integers: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$. The numbers have the following lengths:

d_i	d_1	d_2	d_3	d_4	d_5	d_6
value	1	1	5	7	2	8

You will use dynamic programming to find the summing order that has minimum total cost. In doing so, you will fill in the two tables below with your intermediate results. Remember that you may not change the order of the numbers!

In the left table, the element in column c, row r contains the minimum total cost of adding the subsequence $x_c, x_{c+1}, \ldots, x_{c+r}$. Note that row numbers increase upward.

In the right table, the element in column c, row r contains the index of the subsequence $x_c, x_{c+1}, \ldots, x_{c+r}$ after which the last addition should be done. For example, if we decide to add $(x_2 + x_3 + x_4) + x_5$, then the column 2, row 3 entry in the right table will contain a '4'.

Use dynamic programming to fill in the two tables. *Only fill the entries that you would normally compute with dynamic programming.*

Part C:

From your "minimum costs" table, what is the minimum cost for adding the sequence?

From your "summing positions" table, what is the summing order? Write it as a single expression with nested parentheses.

6						
5						
4						
3						
2						
1						
	1	*2*	*3*	*4*	*5*	*6*

minimum costs

6						
5						
4						
3						
2						
1						
	1	*2*	*3*	*4*	*5*	*6*

summing positions

Problem 3: **Recursion**

For each $1 \leq i \leq j$, we define $f_{i,j}$ as follows:

$$f_{1,j} = 1$$

For $j \geq i > 1$:

$$f_{i,j} = i + j + \sum_{k=1}^{i-1} f_{i-k,j-k}$$

(a) Write a simple function in C which will use dynamic programming to compute $f_{n,m}$.

(b) Write a recursive function in C which will compute $f_{n,m}$. Argue that your function will make an unacceptably long number of recursive calls. Compare running times of both functions with same inputs.

Problem 4: **Allocation of Store Commodities**

Eatons went bankrupt. Sears bought them out. Sears has taken over the flagship Eaton Centre store and is trying to decide what merchandise to sell.

Like Eatons, Sears will continue selling "high-end" merchandise for the moment. Based on Eatons' data, Sears' consultants are trying to estimate the amount of space to allocate to each category of merchandise to minimize their chance of failure. The store has been divided into blocks. The possibility of failure is based on the amount of space (number of blocks) that are devoted to each of the 4 categories of merchandise: electronic entertainment equipment, cosmetics, mens' wear and womens' wear. Sears is committed to each of the 4 categories, so at least one block of each category must be present in the store. They have limited the maximum number of blocks for any category of commodity to 4 blocks. The consultants are using dynamic programming to decide how to best allocate the blocks in the store. With your expertise in dynamic programming, you have been hired on to help them out.

The following table shows the probability of failure given a specified allocation of blocks for each commodity.

number of blocks	1	2	3	4
Cosmetics	.8	.6	.3	.1
Entertainment	.85	.5	.2	.15
Mens	.6	.55	.5	.4
Womens	.75	.65	.55	.45

Table of probabilities of failure

The store has been divided into 7 equal blocks. At least one block must be allocated to each commodity with a maximum of 4 blocks for any category of commodity.

Probabilities are combined using multiplication. For example, if one block of cosmetics, one block of entertainment, two blocks of mens and 1 block of women are allocated then the combined probability of failure is .8 * .85 * .55 * .75 = .2805.

The above table of failure probabilities (PF) is stored as a matrix: $PF[m, n]$. The m dimension refers to the different commodities. The n dimension refers to different numbers of blocks.

We will have a different matrix called CP (combined probabilities) which contains the solution to our problem as well as to various subproblems. For each $1 \le i \le 4$ and $i \le j \le 4i$, we define $CP[i, j]$ to be the optimal probability of failure if we are only using commodities $1, \ldots, i$ and if we only have j blocks. Thus, $CP[4, 7]$ is the solution to our problem. For $i = 1$, i.e. if we are only using the first commodity, then CP is obviously defined as follows:

$$CP[1, j] = PF[1, j].$$

For all other values, we define CP recursively as follows:

$$CP[i, j] = \min_k CP[i-1, j-k] * PF[i, k],$$

where the minimum is taken over all values of k such that

(i) $1 \le k \le 4$ and

(ii) $i - 1 \le j - k \le 4(i - 1)$.

Your assignment is as follows:

(a) Solve this problem by hand using dynamic programming. In your solution show your work, i.e. show the sub-problems, and describe how you picked optimal sub-solutions.

(b) Explain why the recursive definition of CP is valid. Why is $CP[i, j]$ only defined for $i \le j \le 4i$?