CMSI 2130 – Classwork 4

BONUS: This is an optional, bonus classwork exercise meant to help you study for the coming exam. Because this is a BONUS classwork, the solution for it will be posted AT the deadline, so late submissions will not be accepted.

Instructions:
This worksheet gives you some important practice with the fundamentals of dynamic programming – both bottom-up *and* top-down!

- Provide answers to each of the following questions and write your responses in the blanks. If you are expected to show your work in arriving at a particular solution, space will be provided for you.
- Place the names of your group members below:

Group Members:

1. Milo Fritzen
2. ______
3. ______

Problem 1 – The Changemaker Problem

Hey look, Changemaker again! The irony being: there is little change being made in our examples, but its utility for practice with dynamic programming should not be... shortchanged.

Bottom-up Dynamic Programming

1.1. Solve the Changemaker problem using Bottom-up Dynamic Programming using the table below (fill in all of the blanks).

$$D = \{1,4,5\};\ S = 9$$

$D \downarrow S_i \rightarrow$	0	1	2	3	4	5	6	7	8	9
1	0 ←	1 ←	2 ←	3 ←	4 ←	5	6	7	8	9
4	0	1	2	3	1	2	3	4	2	3
5	0	1	2	3	1	1	2	3	2	2

1.2. Indicate the path used in your table above to collect the solution to this Changemaking problem (you can either replicate the table from above or simply draw arrows through the cells used to find the solution). Clearly indicate when you've added a coin to your solution.

$D \downarrow S_i \rightarrow$	0	1	2	3	4	5	6	7	8	9
1										
4	0 ←	←	← +1 coin (4¢)	←	1					
5					↑ 1	←	←	← +1 coin (5¢)	←	2

What solution did you find? 5¢, 4¢

1.3. With some combinations of denominations and sums, it is not possible to make the desired amount of change for some subproblems; demonstrate (by placing an X in cells that cannot be made with the given denominations), in the table below, how this edge case affects the algorithm with the following arguments:

$$D = \{2, 3, 4\};\ S = 9$$

$D \downarrow S_i \rightarrow$	0	1	2	3	4	5	6	7	8	9
2	0	X	1	X	2	X	3	X	4	X
3	0	X	1	1	2	2	2	3	3	3
4	0	X	1	1	1	2	2	2	2	3

Problem 2 – Longest Common Subsequence

On to brighter pastures! Let's finally change from Changemaker and continue into the Longest Common Subsequence (LCS) problem. Once more, we'll juxtapose the bottom-up and top-down dynamic programming approaches to this problem.

Bottom-up Dynamic Programming

2.1. Solve the LCS problem by using bottom-up dynamic programming (fill in the entries of the table below with the length of the LCS for each subproblem).

$$lcs(\text{"GTGACAT"}, \text{"TGATCA"})$$

$R \downarrow C \rightarrow$	Ø	G	T	G	A	C	A	T
Ø	0	0	0	0	0	0	0	0
T	0	0	1	1	1	1	1	1
G	0	1	1	2	2	2	2	2
A	0	1	1	2	3	3	3	3
T	0	1	2	2	2	2	2	4
C	0	1	2	2	2	3	3	4
A	0	1	2	2	3	3	4	4

2.2. Indicate the path used in your table above to collect the solution to this LCS problem (you can either replicate the table from above or simply draw arrows through the cells used to find the solution). Clearly indicate when you've added a letter to your solution.

$R \downarrow C \rightarrow$	Ø	G	T	G	A	C	A	T
Ø								
T	0							
G	+G ↖	1						
A		↑ 1						
T		+T ↖	2	← 2	← 2			
C					+C ↖	3		
A						+A ↖	4	← 4

2.3. What subsequence solution did you find? GTCA

Top-Down Dynamic Programming

2.4. Consider the top-down dynamic programming approach to LCS, show the path that the algorithm would take, starting at the bottom-right cell, and indicating the following (assuming that the gutter / base-case values are known in advance):

- Place arrows in each cell starting from the bottom-right to highlight the evaluated path.
- Place a **star** [⋆] in a cell for which memoization would save effort for overlapping subproblems.
- Place an **ex** [×] in a cell that would never be evaluated in the top-down approach.

R ↓ *C* →	Ø	G	T	G	A	C	A	T
Ø	0	0	0	0	0	0	0	0
T	0	X	↖ ☆	← ↑ △ ★	← ↑ △ ★	← ↑ △ ★	X	X
G	0	↖ ☆	← ↑ △ ★	↖ ↑ ☆	← ↑ △ ★	← ↑ △ ★	X	X
A	0	← ↑ △ ★	↖ ← ↑ △ ★	← ↑ △ ★	↖ ☆	← ↑ △ ★	↖ ☆	X
T	0	X	☆	← △ ★	← ↑ △ ★	↖ ← ↑ △ ★	← ↑ △ ★	↖ ☆
C	0	X	X	X	X	↖ ★	← ↑ △ ★	← ↑ △ ★
A	0	X	X	X	X	X	★	← ↑ △ ★

2.5. Viewing the table above, characterize what scenarios (i.e., describe some property of the inputs) that will lead to *all* of the table's cells still needing to be filled in using the top-down approach.

If none of the characters matched, then no cell would move diagonally / every cell would be a max node, therefore, we would have to check each one to be certain the LCS is zero. For example: LCS (ABC, DEF).

2.6. What is the <u>best-case</u> asymptotic runtime and space complexities of the top-down dynamic programming approach to the LCS problem? Explain.

The inputs for LCS are the same string = Best-case
If each last character matches we never need to branch off i.e. we will have a branching factor of 1. This would greatly save time and space/memory