Discussion 6

I. A "Recipe" for Dynamic Programming

1. Characterize the structure of an optimal solution, and recursively define the value of an optimal solution. In other word, come up with a **formula**
2. Compute the value of an optimal solution in a **bottom-up** fashion, and make use of the computed information (**momoization**)

Recap: Largest Common Subsequence Problem

II. Dynamic Programming VS. Divide-and-conquer

Dynamic Programming	**Divide-and-conquer**
Subproblems **overlap**	Subproblems are **disjoint**, mostly smaller instances of the **same** type
Use a lookup table and traceback table (memoization)	Solve the subproblems recursively
Bottom-up (iteration)	Top-down (recursion)

In the context of subproblems sharing subsubproblems, divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems while dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a lookup table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem. We can conclude that dynamic programming is a good approach to **optimization problem**, such as *max* and *min*.

III. Maximum common substring (Homework 5 Problem 4)

Two character strings may have many common substrings. Substrings are required to be **contiguous** in the original string. For example, *photograph* and *tomography* have several common substrings of length one (i.e., single letters), and common substrings *ph*, *to*, and *ograph* (as well as all the substrings of *ograph*). The maximum common substring (MCS) length is 6.

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$ be two character strings.

(a) Give a dynamic programming algorithm to find the MCS length for X and Y.

Hint: $c[i,j] \stackrel{?}{=} c[i-1,j-1]$

```
LCS-length(X,Y)
set c[i,0] = 0 and c[0,j] = 0
for i = 1 to m // Row-major order to compute c and b arrays
    for j = 1 to n
        if X(i) = Y(j)
           c[i,j] = c[i-1,j-1] + 1
           b[i,j] = 'Diag'      // go to up diagonal
        elseif c[i-1,j] >= c[i,j-1]
           c[i,j] = c[i-1,j]
           b[i,j] = 'Up'      // go up
        else
           c[i,j] = c[i,j-1]
           b[i,j] = 'Left'      // go left
        endif
    endfor
endfor
return c and b
```

(b) Analyze the worst-case running time and space requirements of your algorithm as functions of n and m.

(c) Demonstrate your dynamic programming algorithm for finding the MCS length of character strings *algorithm* and *logarithm* by constructing the dynamic programming table.

		l	o	g	a	r	i	t	h	m
	0	0	0	0	0	0	0	0	0	0
a	0									
l	0									
g	0									
o	0									
r	0									
i	0									
t	0									
h	0									
m	0									