CS325: Homework 3 - Dynamic Programming

1. Solve problem using top-down + Bottom-up Approach

1A: in python file

1B: in Python file

1C: my TOP-down approach differs from Bottom-up due to how it solves. For my Top-down, I used memoization which begins with the original problem and stores the results to speed up calculations. In which, if a subproblem has already been found, it will return the result from the table instead of re-calculating.
On the other hand, my bottom-up approach used a tabulation approach. this means that It starts with base case of subproblems in a table then use those results to solve larger subproblems until it arrives to the problem and solves it.

1D: Time + space complexity of Top-down Approach
the time complexity for TOP-down is $\Theta(x \cdot y)$ which are the length of string DNA1 and string DNA2. The space complexity is $O(x \cdot y)$, x is the length of DNA1 and y is length of DNA2, this is because the memoization table uses that much space

1E: Time + space complexity for Bottom-up Approach
The time complexity is $\Theta(x \cdot y)$ which are the length of string DNA1 and DNA2. The space complexity is $\Theta(x \cdot y)$ for the tabulation which is length of DNA1 and DNA2.

1F. Recurrences

the recurrence for top-down:

length(i, j) = 0 if i == 0 or j == 0
max(length(i-1, j), length(i, j-1)) if DNA2[i-1] != DNA2[j-1].
1 + length(i-1, j-1) if DNA1[i-1] == DNA2[j-1]

For bottom-up:

dp[i][j] = 0 if i == 0 or j == 0
DP[i-1][j-1] + 1 if DNA1[i-1] == DNA2[j-1]
max(DP[i-1][j], DP[i][j-1]) if DNA1[i][j] != DNA2[i][j]

2. Solve DP + compare

2A. Pseudocode to solve using Bottom-down:

```
DP = new array of size n+1
DP[0] = 1
DP[1] = 1
For i from 2 to n:
    DP[i] = DP[i-1] + DP[i-2]
return DP[n]
```

2B. Pseudocode for Brute force Approach

```
function countWays(n):
    if n==0 || n==1
        return 1
    return countWays(n-1) + countWays(n-2)
```

2C. The time complexity for the bottom-down is $O(n)$ due to the action in this is filling up the DP[]. For the brute force, the time complexity is $O(n^2)$, this is because it recursively makes two calls to the countways function, with n-1 and n-2.

2D. Recurrence formula

$$T(n) = T(n-1) + T(n-2)$$