Assignment - 17

Name - Md. Firoze Baba
firozebaba68@gmail.com

Task 1:

Knapsack Problem:

Write a function int Knapsack(int W, int[] weights, int[] values) in C# that determines the maximum value of items that can fit into a knapsack with a capacity W. The function should handle up to 100 items. Find the optimal way to fill the knapsack with the given items to achieve the maximum total value. You must consider that you cannot break items, but have to include them whole.

Program:

```
import java.util.ArrayList;
import java.util.List;

public class KnapsackProblem {

	public static void main(String[] args) {
		int W = 50;
		int[] weights = { 10, 20, 30 };
		int[] values = { 60, 100, 120 };
		int maxvalue = knapsack(W, weights, values);
		System.out.println("Maximum value of items that can
			fit into the knapsack: " + maxvalue);
	}

	private static int knapsack(int size, int[] weights,
					int[] values) {
		int n = weights.length;
		int[][] dp = new int[n + 1][size + 1];
		for (int i = 0; i <= n; i++) {
			for (int j = 0; j <= size; j++) {
				if (i == 0 || j == 0)
					dp[i][j] = 0;
				else if (weights[i - 1] <= j)
					dp[i][j] = Math.max(values[i - 1] +
					dp[i - 1][j - weights[i - 1]],
					dp[i - 1][j]);
				else
					dp[i][j] = dp[i - 1][j];
			}
		}

		int remainingweight = size;
		List<Integer> includeItems = new ArrayList<Integer>();
		for (int k = n; k > 0 && remainingweight > 0; k--) {
			if (dp[k][remainingweight] != dp[k - 1]
				[remainingweight]) {
				includeItems.add(k - 1);
				remainingweight -= weights[k - 1];
			}
```

```
            }
            System.out.println("Items included in the knapsack: ");
            for (int x : includeItems) {
                  System.out.println("Weight: " + weights[x] +
                                          ", Value: " + values[x]);
            }
            return dp[n][size];
      }
}
```

Output:

```
Items included in the knapsack:
Weight: 30, Value: 120
Weight: 20, Value: 100
Maximum value of items that can fit into the knapsack: 220
```

Task 2:

Longest Common Subsequence:

Implement int LCS(string text1, string text2) to find the length of the longest common subsequence between two strings.

Program:

```
public class LCSProblem {

      public static void main(String[] args) {
            String txt1 = "abcdefg";
            String txt2 = "aceg";

            int len = LCS(txt1, txt2);
            System.out.println("Length of LCS: " + len);
      }

      private static int LCS(String txt1, String txt2) {
            int n = txt1.length();
            int m = txt2.length();

            int[][] dp = new int[n + 1][m + 1];

            for (int i = 1; i <= n; i++) {
                  for (int j = 1; j <= m; j++) {
                        if (txt1.charAt(i - 1) == txt2.charAt(j - 1))
{
                              dp[i][j] = dp[i - 1][j - 1] + 1;
                        } else {
                              dp[i][j] = Math.max(dp[i - 1][j],
                                          dp[i][j - 1]);
                        }
                  }
            }
            return dp[n][m];
      }
}
```

Output:

```
Length of LCS: 4
```