<u>Day 15 and 16</u>

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.

Ans)

<u>Code:-</u>

```
package com.wipro.patterns;
import java.util.Arrays;
class Item {
  String name;
  int weight;
  int profit;
  double ratio;
  public Item(String name, int weight, int profit) {
     this.name = name;
     this.weight = weight;
     this.profit = profit;
     this.ratio = (double) profit / weight;
  }
}
public class FractionalKnapsack {
  public static void main(String[] args) {
     Item[] items = {
        new Item("Tea", 2, 10),
        new Item("Coffee", 3, 5),
        new Item("Sugar", 5, 15),
        new Item("Rice", 7, 7),
        new Item("Chilli", 1, 6),
        new Item("Oil", 4, 18),
        new Item("Jira", 1, 3)
     };
     int capacity = 15;
     double maxProfit = getMaxProfit(items, capacity);
     System.out.println("Max Profit is = " + maxProfit);
  }
  private static double getMaxProfit(Item[] items, int capacity) {
     Arrays.sort(items, (a, b) -> Double.compare(b.ratio, a.ratio));
     double totalProfit = 0.0;
     int remainingCapacity = capacity;
     for (Item item : items) {
        if (remainingCapacity == 0) {
           break;
        }
```

```
            if (item.weight <= remainingCapacity) {
                totalProfit += item.profit;
                remainingCapacity -= item.weight;
            } else {
                totalProfit += item.profit * ((double) remainingCapacity / item.weight);
                remainingCapacity = 0;
            }
        }
        return totalProfit;
    }
}
```

OUTPUT:-

```
Max Profit is = 55.333333333333336
```

Task 2: Longest Common Subsequence
Implement int LCS(string text1, string text2) to find the length of the longest common subsequence between two strings.
Ans)
Code:-

```
package WiprpTask;
public class LongestCommonSubsequence {
    public static void main(String[] args) {
        String text1 = "abccba";
        String text2 = "aceeca";
        int length = LCS(text1, text2);
        System.out.println("Length of the longest common subsequence: " + length);
    }
    public static int LCS(String text1, String text2) {
        int m = text1.length();
        int n = text2.length();
        int[][] dp = new int[m + 1][n + 1];
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (text1.charAt(i - 1) == text2.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]);
                }
            }
            printDPTable(dp, m, n);
        }
        int lcsLength = dp[m][n];
        char[] lcs = new char[lcsLength];
        int i = m, j = n, k = lcsLength - 1;
        while (k >= 0) {
            if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
```

```
            lcs[k--] = text1.charAt(i - 1);
            i--;
            j--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }
    System.out.println("Longest common subsequence: " + new String(lcs));
    return lcsLength;
}
private static void printDPTable(int[][] dp, int m, int n) {
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            System.out.print(dp[i][j] + " ");
        }
        System.out.println();
    }
    System.out.println();
}
}
```

OUTPUT:-

```
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
```

```
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 1 2 2 2 3 3
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 2 2 2 2 2
0 1 2 2 2 3 3
0 1 2 2 2 3 3
0 1 2 2 2 3 4
Longest common subsequence: acca
Length of the longest common subsequence: 4
```