Worksheet-8

Student Name: Arpit Anand
UID: 23BCS12710
Branch: BE - CSE
Section/Group: KRG-3(A)
Semester: 5
Subject Name: DAA

Date of Performance: 23/09/2025
Subject Code: 23CSH-301

1. Aim:

Develop a program and analyze complexity to implement 0-1 Knapsack using Dynamic Programming.

2. Objective:

To implement the 0-1 Knapsack problem using Dynamic Programming (Bottom-Up Tabulation) and analyze its time and space complexity for efficient problem solving.

3. Requirements (Hardware/Software):

Online Java compiler.

4. Algorithm :

```
1. Input number of items n, weight array wt[], value array val[], and capacity W.
2. Create a table dp[n+1][W+1].
3. Initialize first row and first column as 0.
4. For each item i = 1 to n:
     • For each capacity w = 1 to W:
          a) If wt[i-1] ≤ w, set
             dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]).
          b) Else set dp[i][w] = dp[i-1][w].
1. Return dp[n][W] as the maximum value.
2. End.
```

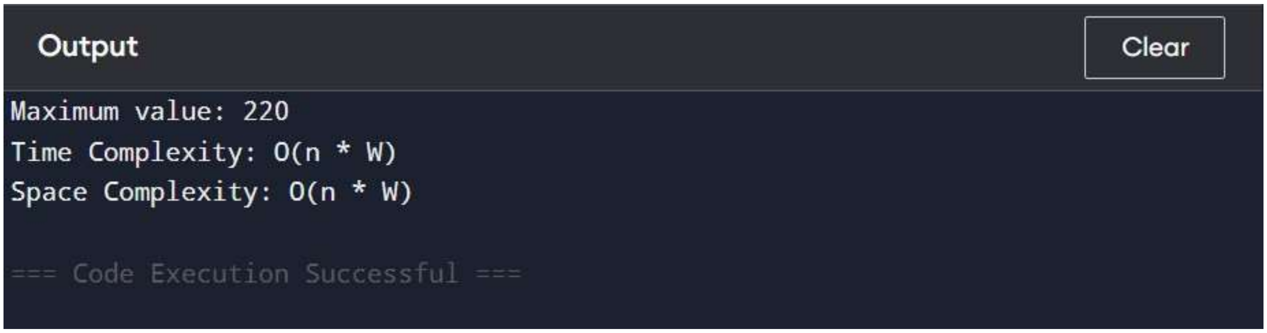
5.Procedure:

```
class Main {
    static int knapSack(int W, int wt[], int val[], int n) {
        int dp[][] = new int[n+1][W+1];
        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= W; w++) {
                if (wt[i-1] <= w) {
                    dp[i][w] = Math.max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);
                } else {
                    dp[i][w] = dp[i-1][w];
                }
            }
        }
        return dp[n][W];
    }
    public static void main(String[] args) {
        int val[] = {60, 100, 120};
        int wt[] = {10, 20, 30};
        int W = 50;
        int n = val.length;
        int result = knapSack(W, wt, val, n);
        System.out.println("Maximum value: " + result);
        System.out.println("Time Complexity: O(n * W)");
        System.out.println("Space Complexity: O(n * W)");
    }
}
```

Time Complexity : Best Case: O(n*w)

Space complexity : O(n*w)

Output:



```
Maximum value: 220
Time Complexity: O(n * W)
Space Complexity: O(n * W)

=== Code Execution Successful ===
```

Learning Outcomes :

1. Understand the application of Dynamic Programming to solve optimization problems like 0-1 Knapsack..
2. Gain the ability to analyze time and space complexities of DP-based solutions.
3. Develop skills to implement efficient algorithms in Java using bottom-up tabulation.