Experiment:2.3

Student Name: Virat Samdarshi **UID: 22BCS12648**
Branch: B.E CSE **Section/Group:IOT-627-B**
Semester:5th **DateofPerformance:18-09-24**
Subject Name:DAAlab **SubjectCode:22CSH-311**

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

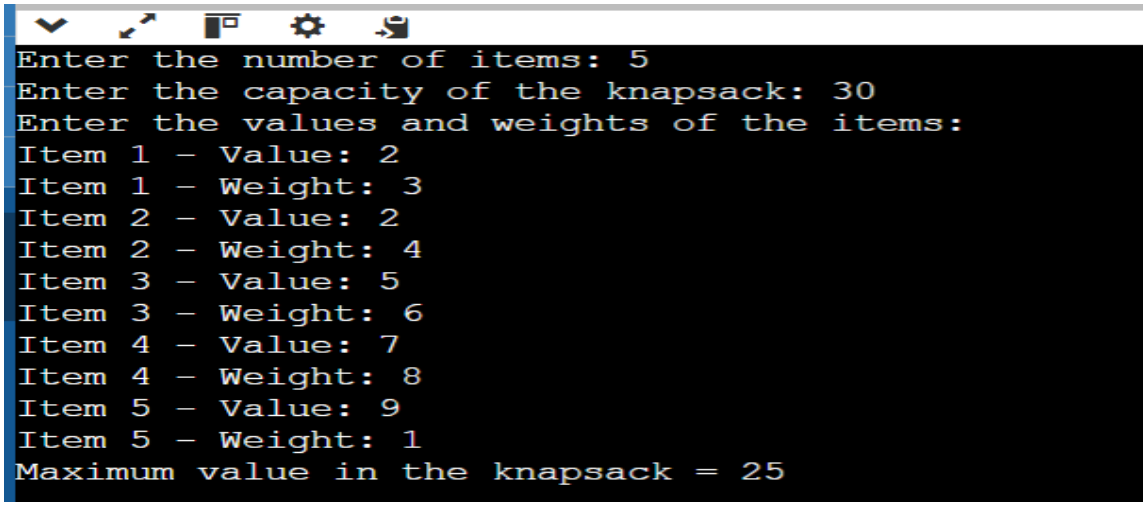
2. **Objective:**The 0-1 Knapsack problem aims to determine the maximum value that can be obtained by selecting items from a given set, each with a specific weight and value, such that the total weight does not exceed a given capacity.

3. **Implementation/Code:**

```
#include <iostream>
#include <vector>
using namespace std;
int knapsackDP(int W, vector<int> wt, vector<int> val, int n) {
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0) {
                dp[i][w] = 0;
            } else if (wt[i - 1] <= w) {

                dp[i][w] = 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];
}

int main() {
    int n, W;
```

```
cout << "Enter the number of items: ";
cin >> n;
cout << "Enter the capacity of the knapsack: ";
cin >> W;

vector<int> val(n), wt(n);

cout << "Enter the values and weights of the items:\n";
for (int i = 0; i < n; i++) {
    cout << "Item " << i + 1 << " - Value: ";
    cin >> val[i];
    cout << "Item " << i + 1 << " - Weight: ";
    cin >> wt[i];
}

int max_value = knapsackDP(W, wt, val, n);
cout << "Maximum value in the knapsack = " << max_value << endl;

return 0;
 }
```

4. Output:



5. Complexity:

Time Complexity: O(n * W)

Space Complexity: O(n * w)