## Experiment – 7

**Student Name:** Garvi Dabas
**UID:** 23BCS11346
**Branch:** BE-CSE
**Section/Group:** KRG-2-B
**Semester:** 5th
**Date of Performance:** 14/8/25
**Subject Name:** DAA
**Subject Code:** 23CSH-301

1. **Aim:** Develop a program and analyze complexity to implement 0-1 Knapsack using Dynamic Programming.
2. **Procedure:**
   - Define the 0-1 Knapsack problem where each item has a value and weight, and the goal is to maximize total value within a weight limit.
   - Take input for number of items, their values, weights, and knapsack capacity.
   - Create a 2D DP table where each cell represents the maximum value for a given number of items and weight limit.
   - Use the recurrence relation to decide whether to include or exclude an item based on its weight.
   - Fill the DP table iteratively and compute the optimal solution
   - Display the final maximum value that can be obtained within the given weight constraint.
3. **Code:**

```
#include <iostream>
using namespace std;

int knapSack(int W, int wt[], int val[], int n) {
    int K[n + 1][W + 1];
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }
    return K[n][W];
}
```

```
int main() {
    int n, W;
    cout << "Enter number of items: ";
    cin >> n;
    int val[n], wt[n];
    cout << "Enter values of items: ";
    for (int i = 0; i < n; i++) cin >> val[i];
    cout << "Enter weights of items: ";
    for (int i = 0; i < n; i++) cin >> wt[i];
    cout << "Enter maximum capacity of knapsack: ";
    cin >> W;
    cout << "Maximum value in knapsack = " << knapSack(W, wt, val, n);
    return 0;
}
```

## 4. Output:

```
Enter number of items: 4
Enter values of items: 60 100 120 90
Enter weights of items: 10 20 30 40
Enter maximum capacity of knapsack: 50
Maximum value in knapsack = 220
```

## 5. Learning Outcomes:

- Gained knowledge of applying Dynamic Programming to optimization problems.
- Learned how to formulate and solve the 0-1 Knapsack problem using tabulation.
- Developed understanding of optimal substructure and overlapping subproblems.
- Gained experience in analyzing time and space complexity of DP algorithms.
- Learned to design efficient solutions for resource allocation and decision-making problems.