Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**

Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing 0/1 Knapsack Problem using Dynamic Programming Approach	
Experiment No: 09	**Date: 13\09\2025**	**Enrollment No:92301733054**

Knapsack Problem

Code :-

```
#include<bits/stdc++.h>
using namespace std;

int knapsack(int weights[], int profits[], int n, int capacity) {
   vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));


   for (int i = 1; i <= n; i++) {
      for (int w = 1; w <= capacity; w++) {
         if (weights[i - 1] <= w) {
            dp[i][w] = max(dp[i - 1][w], profits[i - 1] + dp[i - 1][w - weights[i - 1]]);
         }
         else {
            dp[i][w] = dp[i - 1][w];
         }
      }
   }

   cout << "DP Table (Max Value for Each Capacity):\n";
   for (int i = 0; i <= n; i++) {
      for (int w = 0; w <= capacity; w++) {
         cout << setw(4) << dp[i][w] << "\t";
      }

      cout << endl;
   }

   return dp[n][capacity];
}

int main() {
   int weights[] = { 2, 3, 4, 5 };
   int profits[] = { 3015, 4026, 5789, 6147 };
   int capacity = 5;
   int n = sizeof(weights) / sizeof(weights[0]);

   int max_profit = knapsack(weights, profits, n, capacity);

   cout << "Maximum value in Knapsack = " << max_profit << endl;

   return 0;
  }
```

Output :-

Conclusion:

We learnt in this experiment that the Knapsack Problem can be efficiently solved using dynamic programming. It helps in selecting items to maximize profit without exceeding the capacity, which is useful in resource allocation and optimization problems.