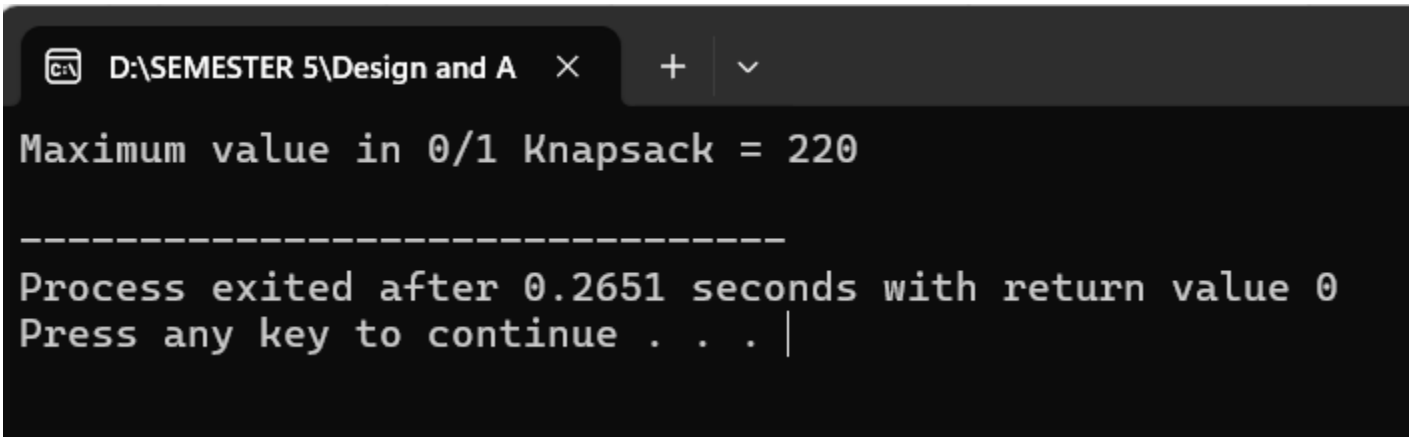
Marwadi University Marwadi Chandarana Group	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithm	**Aim: : Implementing Knapsack Problem using Greedy Programming Approach**	
Experiment No: 06	**Date: 13/09/2025**	**Enrollment No: 92301733049**

AIM: Implementing Knapsack Problem using Greedy Programming Approach

I. 0/1 Knapsack Problem

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

// Function to solve 0/1 Knapsack using DP
int knapsack01(int W, vector<int>& wt, vector<int>& val, int n) {
    vector< vector<int> > dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            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 val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    vector<int> values(val, val + n);
    vector<int> weights(wt, wt + n);

    cout << "Maximum value in 0/1 Knapsack = " << knapsack01(W, weights, values, n) << endl;

    return 0;
}
```

Marwadi University Marwadi Chandarana Group	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithm	**Aim: : Implementing Knapsack Problem using Greedy Programming Approach**	
Experiment No: 06	**Date: 13/09/2025**	**Enrollment No: 92301733049**



II. Fractional Knapsack Problem

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

struct Item {
   int value, weight;
};

//////////////////////////////////////////////////////////////////
// 0/1 Knapsack using Dynamic Programming
//////////////////////////////////////////////////////////////////
int knapsack01(int W, vector<int>& wt, vector<int>& val, int n) {
   // use vector< vector<int> > instead of vector<vector<int>>
   vector< vector<int> > dp(n + 1, vector<int>(W + 1, 0));

   for (int i = 1; i <= n; i++) {
      for (int w = 1; w <= W; w++) {
         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];
}

////////////////////////////////////////////////////////////////
// Fractional Knapsack using Greedy
////////////////////////////////////////////////////////////////
bool cmp(struct Item a, struct Item b) {
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

double fractionalKnapsack(int W, vector<Item>& arr, int n) {
    sort(arr.begin(), arr.end(), cmp);

    double finalValue = 0.0;

    for (int i = 0; i < n; i++) {
        if (arr[i].weight <= W) {
            W -= arr[i].weight;
            finalValue += arr[i].value;
        } else {
            finalValue += arr[i].value * ((double)W / arr[i].weight);
            break;
        }
    }
    return finalValue;
}

////////////////////////////////////////////////////////////////
// Main Function
////////////////////////////////////////////////////////////////
int main() {
    // 0/1 Knapsack Example
    int valArr[] = {60, 100, 120};
    int wtArr[] = {10, 20, 30};
    int W1 = 50;
    int n1 = sizeof(valArr) / sizeof(valArr[0]);

    vector<int> values(valArr, valArr + n1);
    vector<int> weights(wtArr, wtArr + n1);
```

```
    cout << "Maximum value in 0/1 Knapsack = "
         << knapsack01(W1, weights, values, n1) << endl;

    // Fractional Knapsack Example
    int W2 = 50;
    vector<Item> arr;
    arr.push_back((Item){60, 10});
    arr.push_back((Item){100, 20});
    arr.push_back((Item){120, 30});
    int n2 = arr.size();

    cout << "Maximum value in Fractional Knapsack = "
         << fractionalKnapsack(W2, arr, n2) << endl;

    return 0;
}
```