Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Knapsack Problem using Greedy Approach.	
Experiment No: 06	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing Knapsack Problem using Greedy Approach

IDE: Visual Studio Code

0/1 Knapsack Problem

Theory: -

The **0/1 Knapsack Problem** is a combinatorial optimization problem, typically solved using dynamic programming due to its structure. However, it's often contrasted with the Fractional Knapsack problem to illustrate the limits of the greedy approach.

- **Problem Definition**:
  - Given n items, each with a weight wi and a profit/value vi.
  - There is a knapsack with a maximum weight capacity of W.
  - The objective is to maximize the total profit without exceeding the weight capacity W.
  - In the 0/1 knapsack problem, each item can either be fully included in the knapsack or not included at all, hence the "0/1" nature.
- **Greedy Approach and Its Limitations**:
  - A greedy approach would typically sort items by their value-to-weight ratio (i.e., vi/wi) and then add items to the knapsack in descending order of this ratio.
  - Unfortunately, in the 0/1 knapsack problem, this approach does not always yield the optimal solution. This is because once an item is either chosen or discarded, we cannot split it, which might lead to suboptimal use of the knapsack's capacity.
  - Example: Consider two items where item 1 has a high value-to-weight ratio but would not allow any other item to fit if taken; a lower ratio item combination could achieve a higher overall value.

Algorithm: -

Programming Language: - C++

Code :-

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

int Knapsack_0_1(vector<int>& Profit, vector<int>& Weigth, int total_weight) {
    vector<pair<double, int>> Profit_weight;
    for (int i = 0; i < Profit.size(); i++) {
        Profit_weight.push_back({ (double)Profit[i] / Weigth[i], Weigth[i] });
    }

    sort(Profit_weight.begin(), Profit_weight.end(), [](pair<double, int>& a, pair<double, int>&
b) {
        return a.first > b.first;
        });

    int max_profit = 0;

    for (int i = 0; i < Profit_weight.size(); i++) {
        if (Profit_weight[i].second <= total_weight) {
            max_profit += Profit_weight[i].first * Profit_weight[i].second;
            total_weight -= Profit_weight[i].second;
        }
        else {
            break;
        }
    }
    return max_profit;
}
```

```
int main() {
    vector<int> Profit = { 60,100,120 };
    vector<int> Weight = { 10,20,20 };
    int total_weight = 40;
    int total_profit = Knapsack_0_1(Profit, Weight, total_weight);
    cout << "Total Profit Is : " << total_profit << endl;
    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6\" ; if ($?) { g++ 0_1_Knapsack.cpp -o 0_1_Knapsack } ; if ($?) { .\0_1_Knapsack }
Total Profit Is : 180
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6>
```

Space Complexity:- __________
Justification: -

Time Complexity:

Best Case Time Complexity: __________
Justification: -

Worst Case Time Complexity:- ____________
Justification: -

Fractional Knapsack Problem :-

Theory: -

The **Fractional Knapsack Problem** is similar to the 0/1 Knapsack Problem but allows for items to be split into fractions, making it more amenable to a greedy approach.

1. **Problem Definition**:
   - Given n items, each with a weight wi and a profit/value vi.
   - A knapsack has a weight capacity W.
   - The objective is to maximize the total profit, with the added ability to take fractions of items if necessary.
2. **Greedy Approach**:
   - In the fractional knapsack, we can break items into smaller parts, allowing the use of a **greedy algorithm** effectively.
   - The steps are:
     - Calculate the value-to-weight ratio (vi/wi) for each item.
     - Sort items in descending order of this ratio.
     - Start adding items to the knapsack from the top of the sorted list until the knapsack cannot take the full weight of the next item.
     - If an item cannot fully fit, add the fraction of it that maximizes the remaining capacity W.
   - Since fractions of items are allowed, this strategy always yields the optimal solution.

Algorithm: -

Programming Language: - C++

Code :-

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

double Fractional_Knapsack(vector<int>& Profit, vector<int>& Weight, int total_weight) {
    vector<pair<double, int>> Profit_weight;
    for (int i = 0; i < Profit.size(); i++) {
        Profit_weight.push_back({ (double)Profit[i] / Weight[i], Weight[i] });
    }
    sort(Profit_weight.begin(), Profit_weight.end(), [](pair<double, int>& a, pair<double, int>& 
b) {
        return a.first > b.first;
        });

double Max_Profit = 0.0;
    for (int i = 0; i < Profit_weight.size(); i++) {
        if (Profit_weight[i].second <= total_weight) {
            Max_Profit += Profit_weight[i].first * Profit_weight[i].second;
            total_weight -= Profit_weight[i].second;
        }
        else {
            Max_Profit += Profit_weight[i].first * total_weight;
            break;
        }
    }

    return Max_Profit;
}

int main() {
    vector<int> Profit = { 60, 100, 120 };
    vector<int> Weight = { 10, 20, 40 };
    int total_weight = 40;
    double total_profit = Fractional_Knapsack(Profit, Weight, total_weight);
    cout << "Total Profit Is : " << total_profit << endl;
    return 0;
}
```

Marwadi University	**Marwari University** **Faculty of Technology** **Department of Information and Communication Technology**
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing Knapsack Problem using Greedy Approach.

Experiment No: 06	**Date:**	**Enrollment No: 92200133030**

Output :-

```
PS C:\Users\Aaryan> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6\" ; if ($?) { g++ Fractional_Knapsack.cpp -o Fractional_Knapsack } ; if ($?) { .\Fractional_Knapsack }
Total Profit Is : 190
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 6>
```

Space Complexity:- ____________________

Justification: -

Time Complexity:

Best Case Time Complexity: ____________________

Justification: -

Worst Case Time Complexity:- ________________

Justification: -

Conclusion:-