Name	Shivam S. Kamble
UID	2022301007
Subject	DAA
Exp No.	05

Aim – Experiment based on greedy approach (fractional knapsack problem).

Theory: **Knapsack Problem Using Greedy Method:** The selection of some things, each with profit and weight values, to be packed into one or more knapsacks with capacity is the fundamental idea behind all families of knapsack problems. The knapsack problem had two versions that are as follows:

1. **Fractional Knapsack Problem**
2. **0 /1 Knapsack Problem**

The fractional **Knapsack problem using the Greedy Method** is an efficient method to solve it, where you need to sort the items according to their ratio of value/weight. In a fractional knapsack, we can break items to maximize the knapsack's total value. This problem in which we can break an item is also called the **Fractional knapsack problem**.

Code :

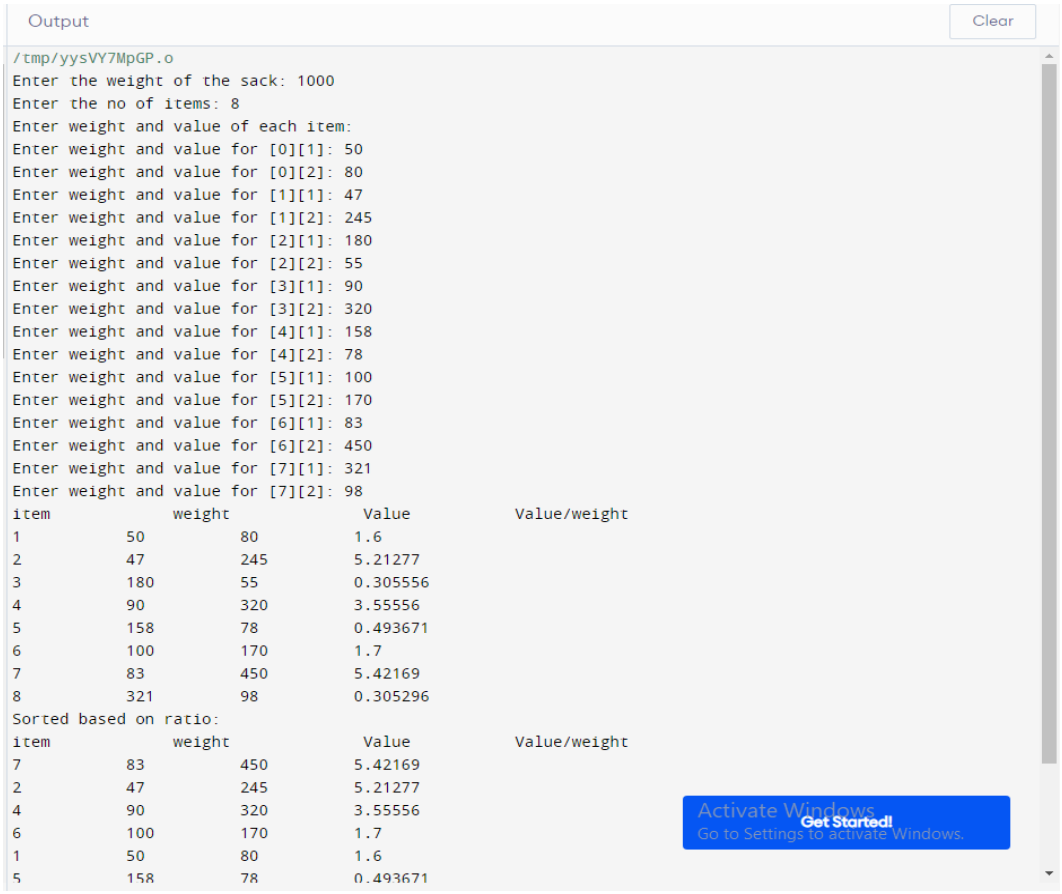
```
#include <bits/stdc++.h>

#include <iostream>

#include <algorithm>

#include <string>

#include <cmath>

using namespace std;

void printarr(double **arr,int n) {

cout << "item \t\t\t weight \t\t\t Value \t\t\t Value/weight\n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < 4; j++) {

cout << arr[i][j] << "\t\t\t";

}

cout << "\n";

}

}

int main()

{
```

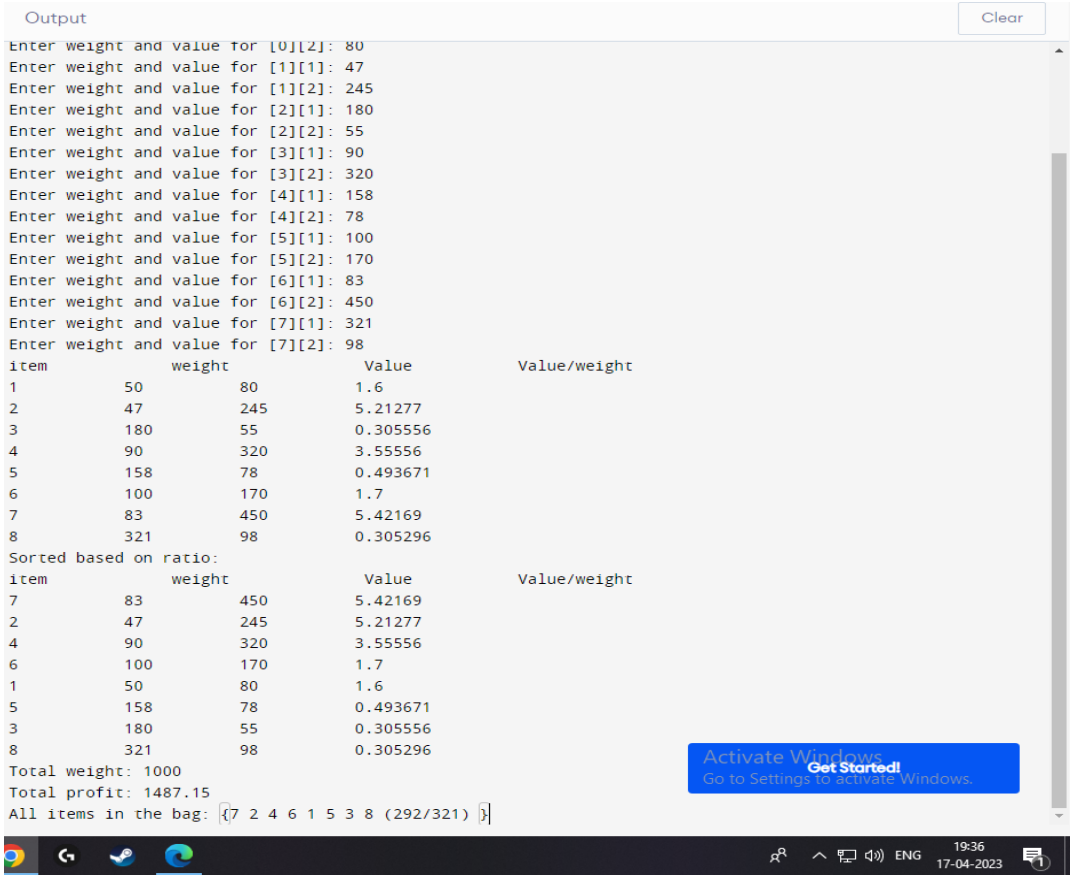
```
int c, n;
double profit = 0.0,weight = 0.0;
cout << "Enter the weight of the sack: ";
cin >> c;
cout << "Enter the no of items: ";
cin >> n;
cout << "Enter weight and value of each item: \n";
vector<string> s (n);
double **arr = new double*[n];
for (int i = 0; i < n; i++) {
arr[i] = new double[4];
arr[i][0]=i+1;
for (int j = 1; j < 4; j++) {
if (j == 3)
{
arr[i][j] = arr[i][2] / arr[i][1];
}
else {
cout << "Enter weight and value for [" << i << "][" << j << "]: ";
cin >> arr[i][j];
}
}
}
printarr(arr,n);
cout << "Sorted based on ratio: " << endl;
sort(arr, arr + n, [](const double* a, const double* b) {
return a[3] > b[3];
});
printarr(arr,n);
int remain = 0;
```

```
double remain_pro = 0.0;
string coco = "";
ostringstream ss;
for (int i = 0; i < n; i++) {
if (c >= weight + arr[i][1]){
weight += arr[i][1];
s[i] = to_string(lround(arr[i][0]));
profit += arr[i][2];
}
else
{
remain = c - weight;
weight += remain;
remain_pro = (remain * arr[i][2]) / arr[i][1];
profit += remain_pro;
ss << remain << "/" << arr[i][1];
coco = ss.str();
s[i] = to_string(lround(arr[i][0])) + " (" + coco + ")";
break;
}
}
cout << "Total weight: " << weight << endl;
cout << "Total profit: " << profit << endl;
cout << "All items in the bag: {";
for (int i = 0; i < s.size(); i++)
{
cout << s[i] << " ";
}
cout << "}";
return 0;
```

}

Output :


Output
Clear
/tmp/yysVY7MpGP.o
Enter the weight of the sack: 1000
Enter the no of items: 8
Enter weight and value of each item:
Enter weight and value for [0][1]: 50
Enter weight and value for [0][2]: 80
Enter weight and value for [1][1]: 47
Enter weight and value for [1][2]: 245
Enter weight and value for [2][1]: 180
Enter weight and value for [2][2]: 55
Enter weight and value for [3][1]: 90
Enter weight and value for [3][2]: 320
Enter weight and value for [4][1]: 158
Enter weight and value for [4][2]: 78
Enter weight and value for [5][1]: 100
Enter weight and value for [5][2]: 170
Enter weight and value for [6][1]: 83
Enter weight and value for [6][2]: 450
Enter weight and value for [7][1]: 321
Enter weight and value for [7][2]: 98
item weight Value Value/weight
1 50 80 1.6
2 47 245 5.21277
3 180 55 0.305556
4 90 320 3.55556
5 158 78 0.493671
6 100 170 1.7
7 83 450 5.42169
8 321 98 0.305296
Sorted based on ratio:
item weight Value Value/weight
7 83 450 5.42169
2 47 245 5.21277
4 90 320 3.55556
6 100 170 1.7
1 50 80 1.6
5 158 78 0.493671
Activate Windows
Go to Settings to activate Windows.
Get Started!



Conclusion : Thus, after performing this experiment I understood and implemented the fractional knapsack problem in which you have to put items in a knapsack of capacity W in order to get the maximum total value in the knapsack and we can also break the items in order to maximize the profit.