NAME:	Eshan Bhuse
UID:	2021300013
SUBJECT	Design and analysis of algorithm
EXPERIMEN TNO:	5
AIM:	**Experiment based on greedy approach (fractional knapsack problem).**

ALGORITHM:

```
for i = 1 to n
do x[i] = 0
weight = 0
for i = 1 to n
if weight + w[i] ≤ W
then
x[i] = 1
weight = weight + w[i]
else
x[i] = (W - weight) / w[i]
weight = W
break
return x
```

THEORY:

Knapsack Problem-

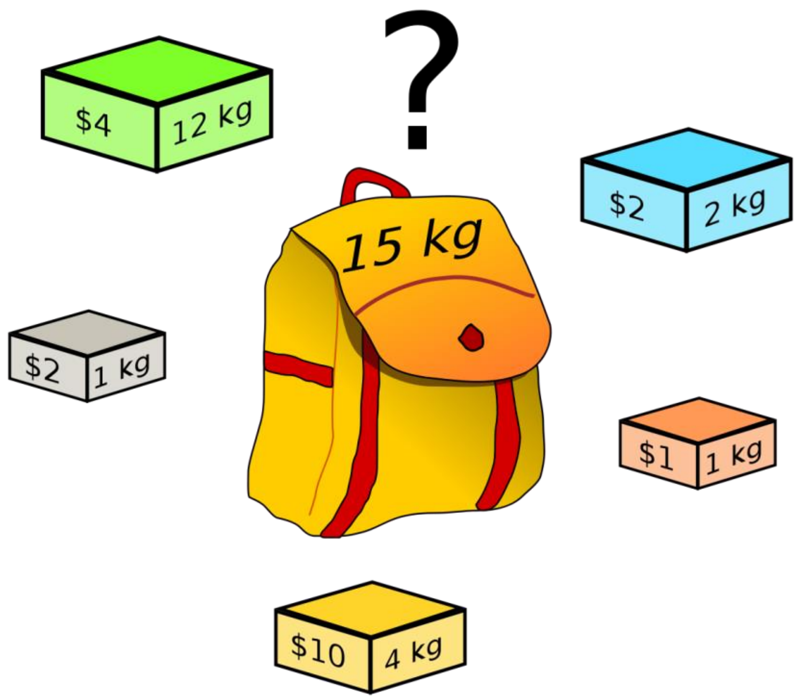
You are given the following-

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.



Knapsack Problem Variants-

Knapsack problem has the following two variants-

- Fractional Knapsack Problem
- 0/1 Knapsack Problem

Fractional Knapsack Problem-

In Fractional Knapsack Problem,

- As the name suggests, items are divisible here.
- We can even put the fraction of any item into the knapsack if taking the complete item is not possible.
- It is solved using Greedy Method.

Step-01:

For each item, compute its value / weight ratio.

Step-02:

Arrange all the items in decreasing order of their value / weight ratio.

Step-03:

Start putting the items into the knapsack beginning from the item with the highest ratio.

Put as many items as you can into the knapsack.

Time Complexity-

The main time taking step is the sorting of all items in decreasing order of their value / weight ratio.

If the items are already arranged in the required order, then while loop takes O(n) time.

The average time complexity of **Quick Sort** is O(nlogn).

Therefore, total time taken including the sort is O(nlogn).

This problem can be solved with the help of using two techniques:

- Brute-force approach: The brute-force approach tries all the possible solutions with all the different fractions but it is a time-consuming approach.
- Greedy approach: In Greedy approach, we calculate the ratio of profit/weight, and accordingly, we will select the item. The item with the highest ratio would be selected first.

There are basically three approaches to solve the problem:

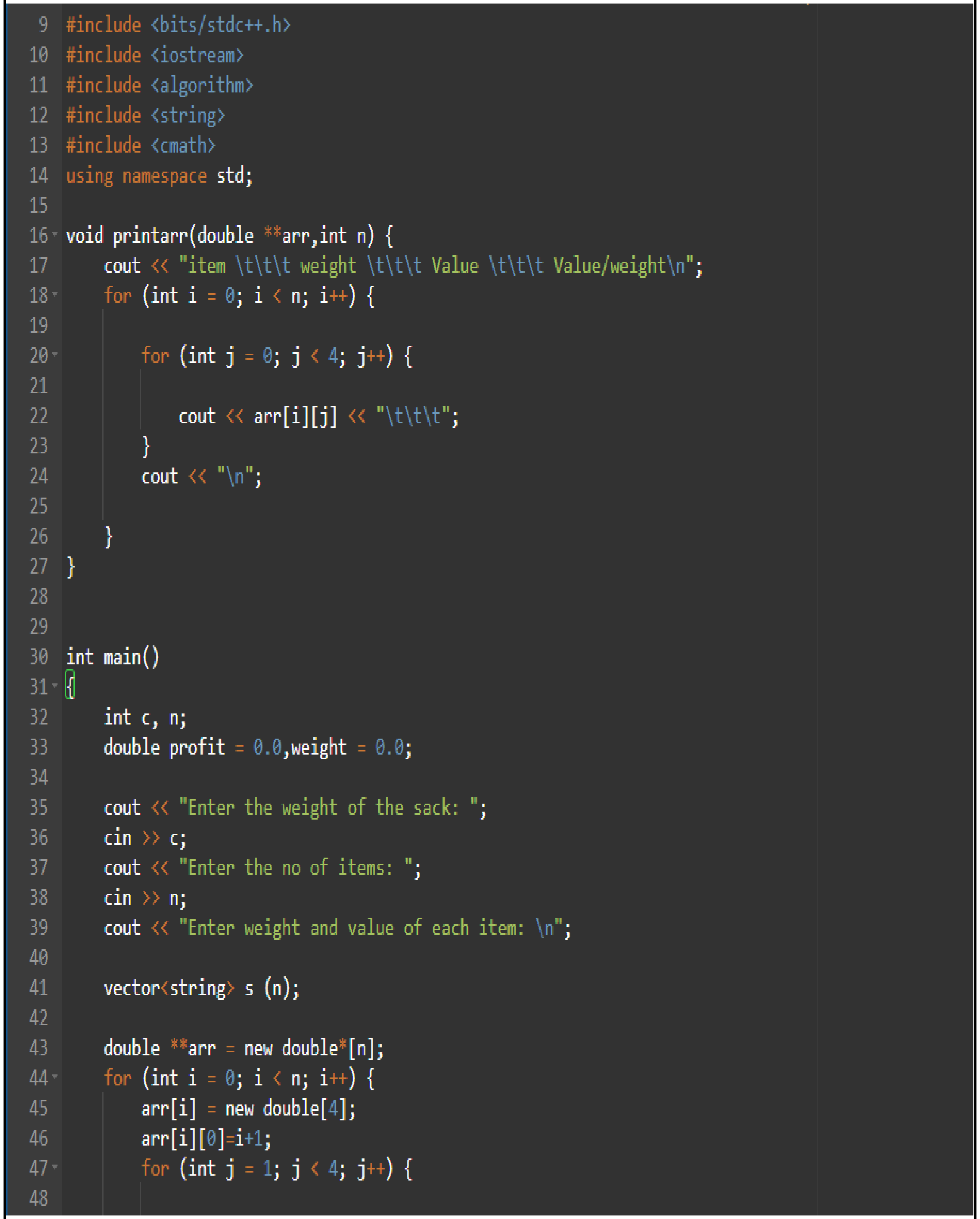
- The first approach is to select the item based on the maximum profit.
- The second approach is to select the item based on the minimum weight.
- The third approach is to calculate the ratio of profit/weight.

Given a set of items, each with a weight and a value, determine which items to include in the collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

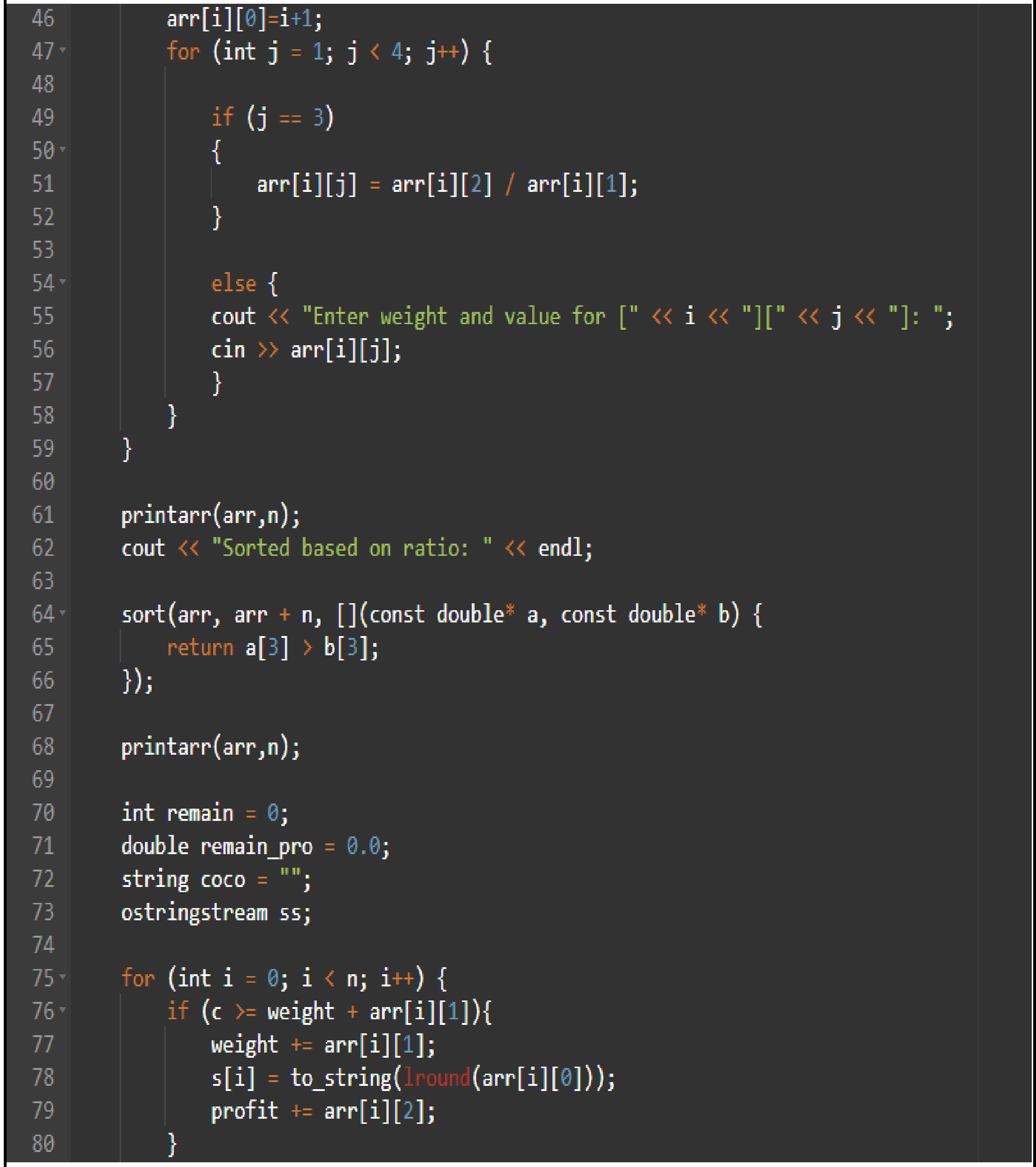
It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision-makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

The knapsack problem has been studied for more than a century, with early works dating as far back as 1897.[1] The name "knapsack problem" dates back to the early works of the mathematician Tobias Dantzig (1884–1956),[2] and refers to the commonplace problem of packing the most valuable or useful items without overloading the luggage.

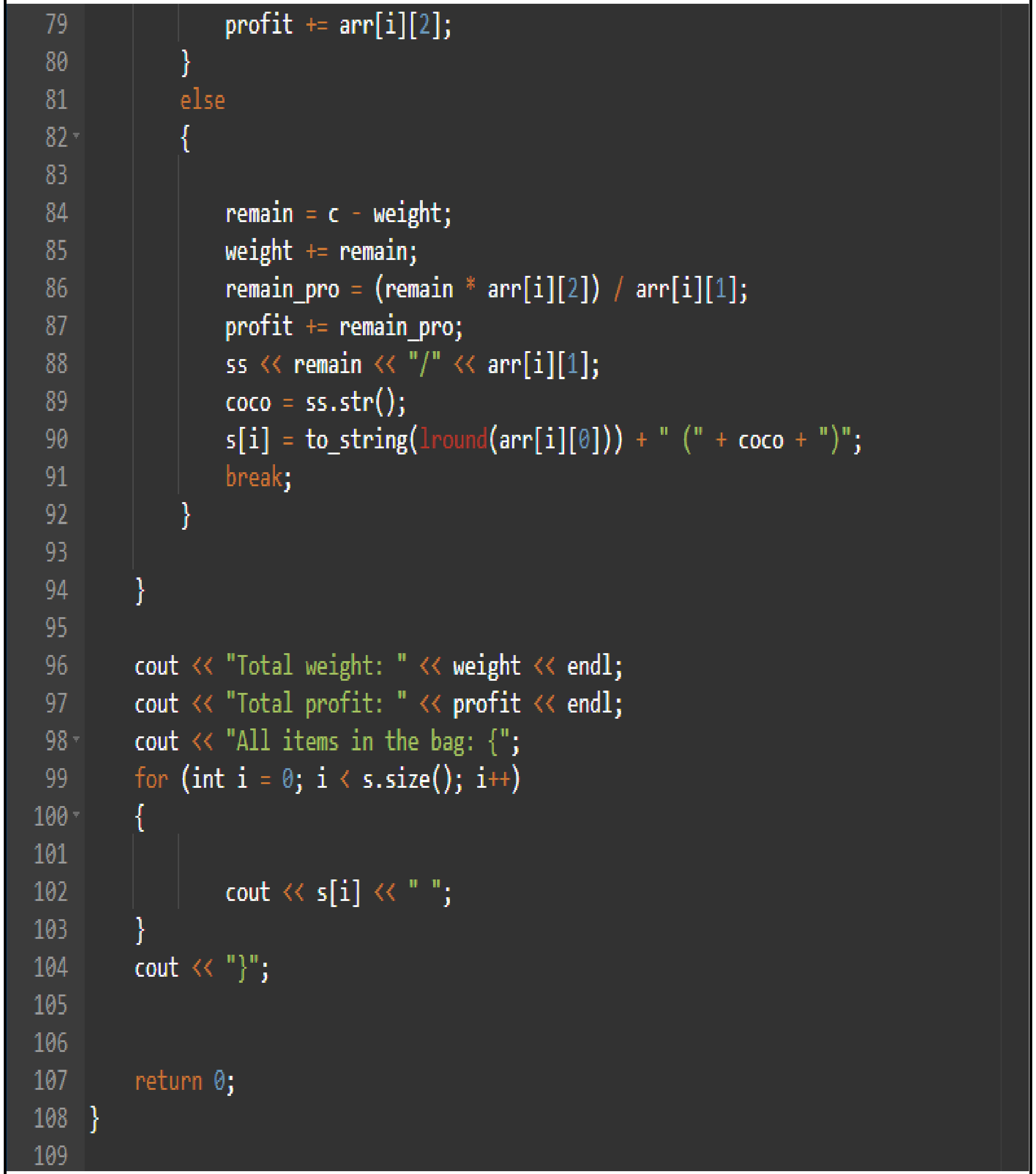
0/1 knapsack problem	**Fractional knapsack problem**
This problem is solved using dynamic programming approach.	This problem is solved using greedy approach.
For example: Suppose we have 10 amount of space. The amount of A is 5, the amount of B is 4, and the amount of C is 3. First, we put A then we put B in the knapsack. The total space occupied by the knapsack is 9. Since the total space in knapsack is 10 so we cannot put 'C' in the knapsack.	For example, suppose we have 10 amount of space. The amount of A is 5, the amount of B is 4,and the amount of C is 3.First, we put A then we put B in the knapsack. Till now,the total space occupied By the knapsack is 9. 1 space is remaining in the knapsack. So, we can take 1 amount from the C and put it in the knapsack.
This problem either takes an item or not. It does not take a part of it.	In this problem, we can also take a fraction of item.
It has optimal structure.	It has a optimal structure.
It finds a most valuable subset item with a total value less than or equal to weight.	It finds item with a total value equal to weight.

PROGRAM:	

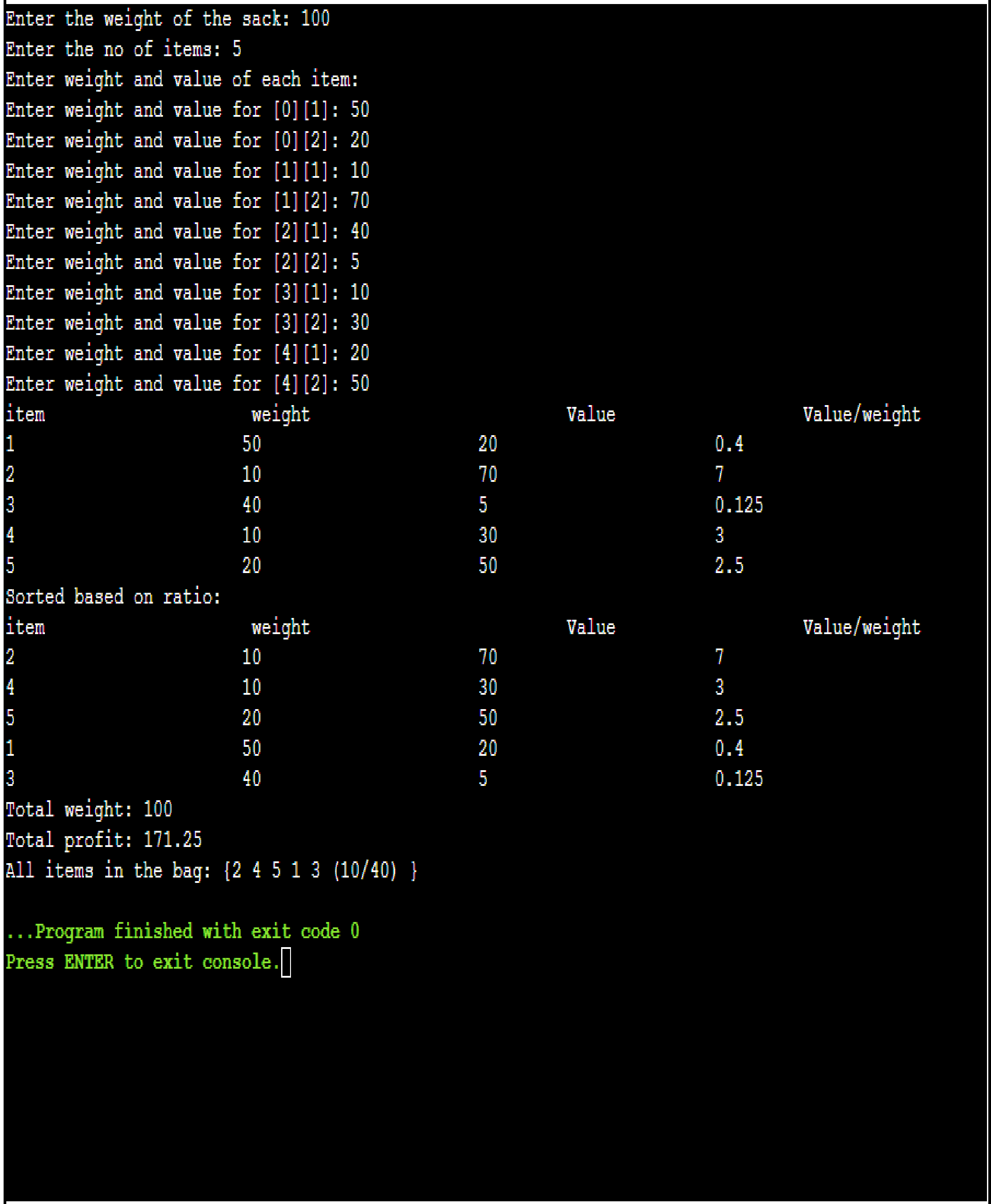
```
#include <bits/stdc++.h>
#include <iostream>
#include <algorithm>
#include <string>
#include <cmath>
using namespace std;

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++) {

```

```cpp
        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];
        }
```



```
                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;
}

```

```
Enter the weight of the sack: 100
Enter the no of items: 5
Enter weight and value of each item:
Enter weight and value for [0][1]: 50
Enter weight and value for [0][2]: 20
Enter weight and value for [1][1]: 10
Enter weight and value for [1][2]: 70
Enter weight and value for [2][1]: 40
Enter weight and value for [2][2]: 5
Enter weight and value for [3][1]: 10
Enter weight and value for [3][2]: 30
Enter weight and value for [4][1]: 20
Enter weight and value for [4][2]: 50
item                     weight                         Value                   Value/weight
1                       50                      20                      0.4
2                       10                      70                      7
3                       40                      5                       0.125
4                       10                      30                      3
5                       20                      50                      2.5
Sorted based on ratio:
item                     weight                         Value                   Value/weight
2                       10                      70                      7
4                       10                      30                      3
5                       20                      50                      2.5
1                       50                      20                      0.4
3                       40                      5                       0.125
Total weight: 100
Total profit: 171.25
All items in the bag: {2 4 5 1 3 (10/40) }

...Program finished with exit code 0
Press ENTER to exit console.
```

OBSERVATION:	Thus, we observe that by using the greedy approach which allows us to get the best option possible without worrying about optimization, we can get the maximum profit possible for the total weight by calculating the value and weight ratio and sorting the table accordingly and also get the maximum number of items in the bag.
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.
