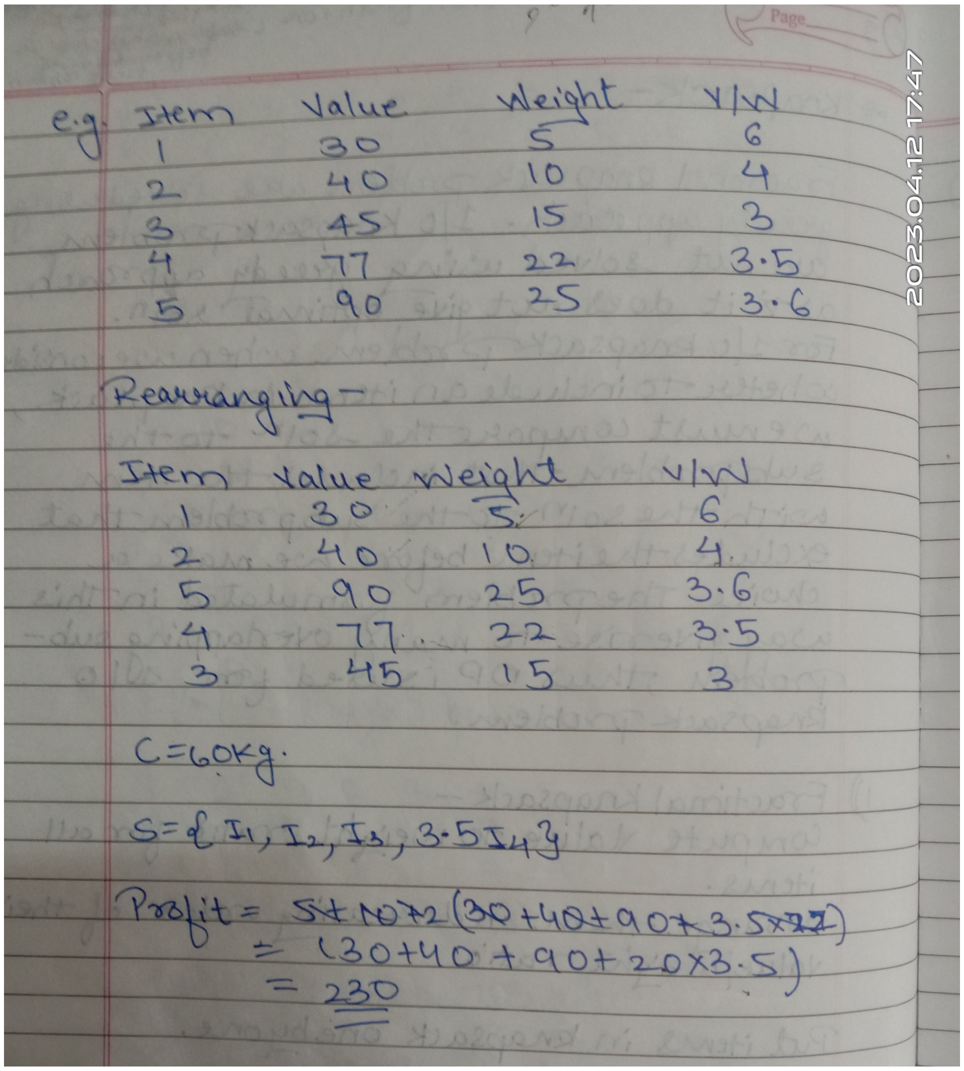
NAME:	Bhavesh Prashant Chaudhari
UID:	2021300018
SUBJECT	DAA
EXPERIMENT NO :	05
AIM:	To implement fractional knapsack.
PROBLEM STATEMENT 1:	
THEORY	Given the weights and profits of N items, in the form of (profit, weight) put these items in a knapsack of capacity W to get the maximum total profit in the knapsack. In Fractional Knapsack, we can break items for maximizing the total value of the knapsack. The basic idea of the greedy approach is to calculate the ratio profit/weight for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them as much as we can (can be the whole element or a fraction of it). This will always give the maximum profit because, in each step it adds an element such that this is the maximum possible profit for that much weight. This algorithm takes O(n log n) time to sort the items by the ratio in decreasing order, and another O(n) time to traverse and pick from the list of items until the knapsack is full. Hence the total running time is O(n log n). We will then prove the correctness of this greedy algorithm.



e.g.

Item	Value	Weight	V/W
1	30	5	6
2	40	10	4
3	45	15	3
4	77	22	3.5
5	90	25	3.6

Rearranging –

Item	Value	Weight	V/W
1	30	5	6
2	40	10	4
5	90	25	3.6
4	77	22	3.5
3	45	15	3

$C = 60 kg.$

$S = \{I_1, I_2, I_3, 3.5 I_4\}$

$$\text{Profit} = 5+10+2(30+40+90+3.5\times\cancel{22})$$
$$= (30+40+90+20\times3.5)$$
$$= \underline{\underline{230}}$$

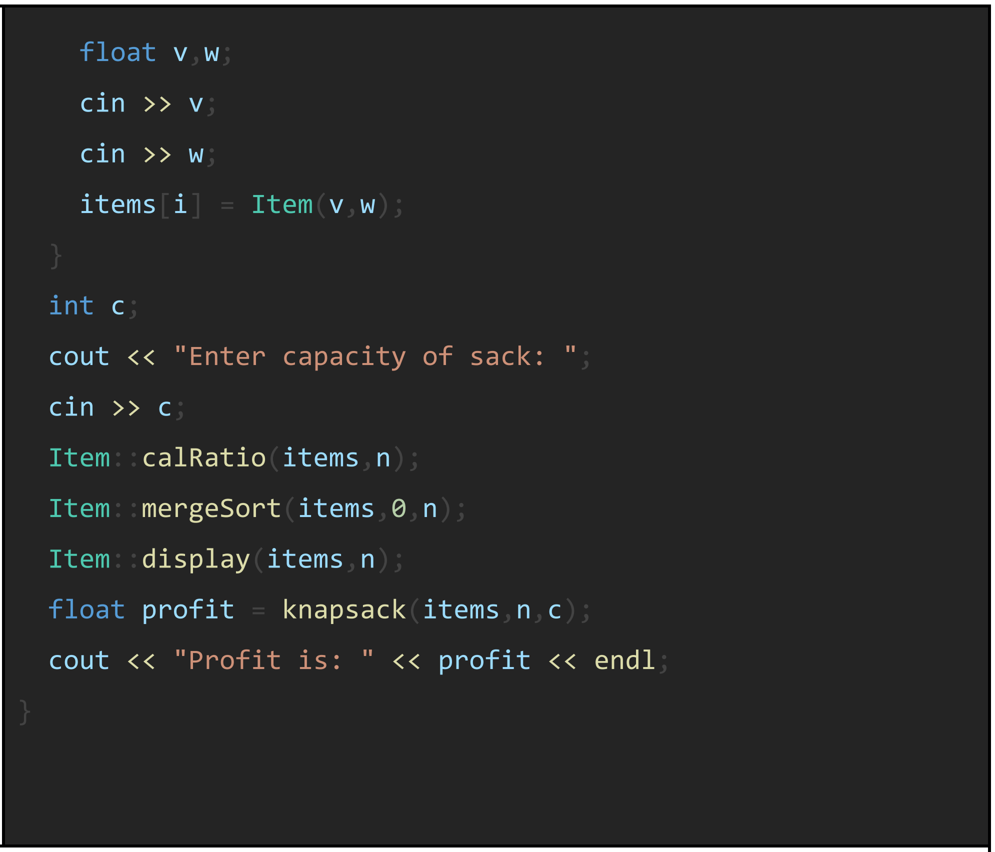
ALGORITHM

Calculate profile/weight ratio of each item and sort items according to their profit/weight ratio.
Iteratively picks the item with the greatest value-per-weight ratio (vi
wi).
If, at the end, the knapsack cannot fit the entire last item with greatest value-per-weight ratio among the
remaining items, we will take a fraction of it to fill the knapsack.

PROGRAM:

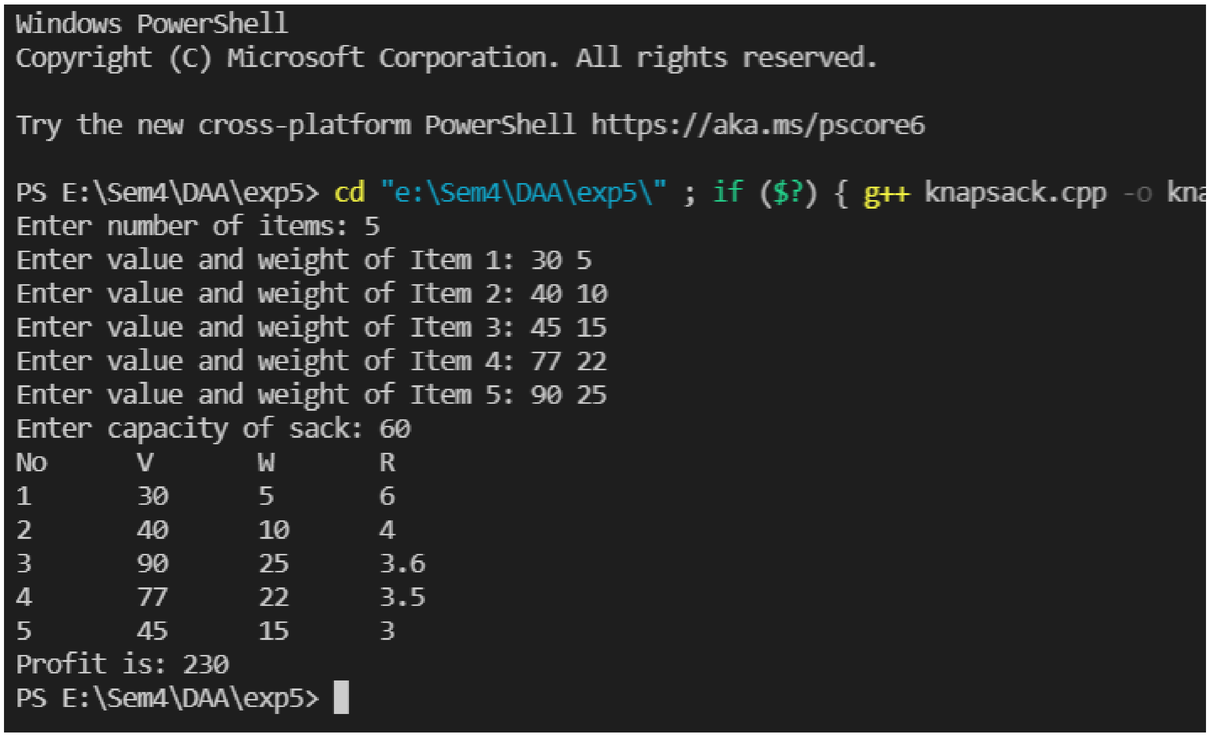
```
#include<iostream>
using namespace std;

class Item{
  public:
  float value;
  float weight;
  float ratio;

  public:
  Item(){
  }
  Item(float value,float weight){
    this->value = value;
    this->weight = weight;
  }
  static void calRatio(Item *items,int n){
    for(int i=0;i<n;i++)
      items[i].ratio = items[i].value/items[i].weight;

  }
static void merge(int *arr,int low,int high,int mid){
  int left_size = mid-low+1;
  int right_size = high-mid;
  int *left_arr = new int[left_size];
  int *right_arr = new int[right_size];
  for(int i=0;i<left_size;i++)
    left_arr[i] = arr[low+i];
  for(int i=0;i<right_size;i++)
    right_arr[i] = arr[mid+i+1];
  int i=0,j=0;
  int k=low;
  while(i<left_size && j<right_size){
    if(left_arr[i] < right_arr[j]){
      arr[k] = left_arr[i];
      i++;
    }else{
      arr[k] = right_arr[j];
      j++;
    }
    k++;
}
  while(i<left_size){
    arr[k] = left_arr[i];
    i++;
    k++;
}
  while(j<right_size){
    arr[k] = right_arr[j];
    j++;
    k++;
}
  delete[] left_arr;
  delete[] right_arr;
  return;
}



static void mergeSort(int *arr,int low,int high){
  if(low < high){
    int mid = (low+high)/2;
    mergeSort(arr,low,mid);
    mergeSort(arr,mid+1,high);
    merge(arr,low,high,mid);
}
  return;
```

```
}

  static void display(Item *items,int n){
    cout << "No\tV\tW\tR" << endl;
    for(int i=0;i<n;i++)
      cout << i+1 << "\t" << items[i].value << "\t" <<
items[i].weight << "\t" << items[i].ratio << endl;
  }


};


float knapsack(Item *items,int n,int c){
  int i;
  float profit = 0;
  for(i=0;i<n;i++){
    if(c < items[i].weight)
      break;
    profit += items[i].value;
    c -= items[i].weight;
  }
  profit += c*items[i].ratio;
  return profit;
}

int main(){
  int n;
  cout << "Enter number of items: ";
  cin >> n;
  Item items[n];
  for(int i=0;i<n;i++){
    cout << "Enter value and weight of Item " << i+1 << ": ";
```



```
    float v,w;
    cin >> v;
    cin >> w;
    items[i] = Item(v,w);
  }
  int c;
  cout << "Enter capacity of sack: ";
  cin >> c;
  Item::calRatio(items,n);
  Item::mergeSort(items,0,n);
  Item::display(items,n);
  float profit = knapsack(items,n,c);
  cout << "Profit is: " << profit << endl;
}
```

RESULT (SNAPSHOT)



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\Sem4\DAA\exp5> cd "e:\Sem4\DAA\exp5\" ; if ($?) { g++ knapsack.cpp -o kna
Enter number of items: 5
Enter value and weight of Item 1: 30 5
Enter value and weight of Item 2: 40 10
Enter value and weight of Item 3: 45 15
Enter value and weight of Item 4: 77 22
Enter value and weight of Item 5: 90 25
Enter capacity of sack: 60
No      V       W       R
1       30      5       6
2       40      10      4
3       90      25      3.6
4       77      22      3.5
5       45      15      3
Profit is: 230
PS E:\Sem4\DAA\exp5> 
```

CONCLUSION:	Through this experiment I understood fractional knapsack problem and how to implement it using greedy approach.The Time Complexity through greedy approach is O(nlogn) as its O(nlogn) time to sort all items.