PRACTICAL 4

Greedy Approach

PRACTICAL 4.1 A

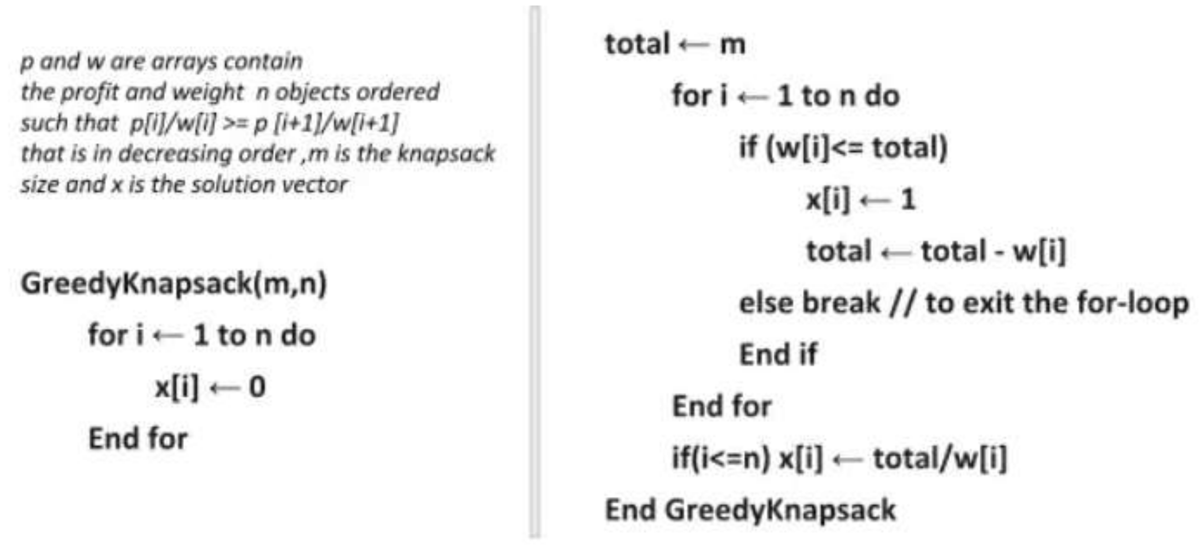
Aim : A Burglar has just broken into the Fort! He sees himself in a room with n piles of gold dust. Because each pile has a different purity, each pile also has a different value (v[i]) and a different weight (w[i]). A Burglar has a bag that can only hold W kilograms. Given n=5, v={4,2,2,1,10}, c={12,1,2,1,4} and W=15,calculate which piles Burglar should completely put into his bag and which he should put only fraction into his bag. Design and implement an algorithm to get maximum piles of gold using given bag with W capacity, Burglar is also allowed to take fractional of pile

Software Requirement : Compiler, MS Excel

Theory :

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Algorithm :



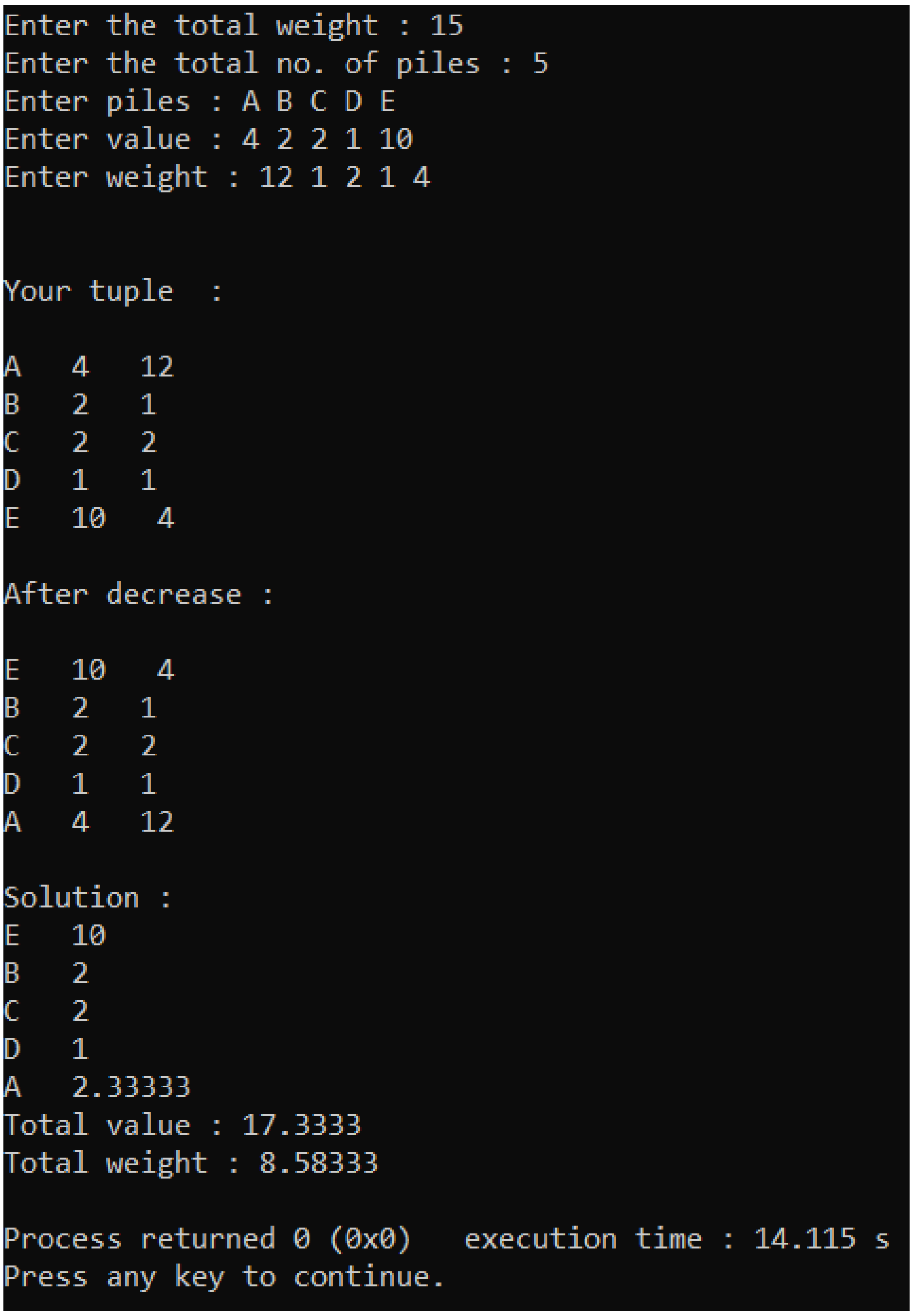
*p and w are arrays contain
the profit and weight n objects ordered
such that p[i]/w[i] >= p [i+1]/w[i+1]
that is in decreasing order ,m is the knapsack
size and x is the solution vector*

```
GreedyKnapsack(m,n)
    for i ← 1 to n do
        x[i] ← 0
    End for
total ← m
    for i ← 1 to n do
        if (w[i]<= total)
            x[i] ← 1
            total ← total - w[i]
        else break // to exit the for-loop
        End if
    End for
    if(i<=n) x[i] ← total/w[i]
End GreedyKnapsack
```

Code :

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

using namespace std;

void decrease (tuple<string, float, float> value[], int len) {
    tuple<string, float, float> temp;
    for (int i=0; i<len-1; i++) {
        for (int j=0; j<len-i-1; j++) {
                float v =  get<1>(value[j])/get<2>(value[j]);
                float d =  get<1>(value[j+1])/get<2>(value[j+1]);
                if (v < d) {
                //if (get<1>(value[j]) < get<1>(value[j+1])) {
                temp = value[j];
                value[j] = value[j+1];
                value[j+1] = temp;
            }
        }
    }
}

void make_profit (tuple<string, float, float> value[], int len, int w) {
    float total = w;
    float profit = 0, weight = 0;
    tuple<string, float, float> sol[len];
    for (int i=0; i<len; i++) {
        if ((get<2>(value[i]) <= total) && (total != 0)) {
            sol[i] = make_tuple(get<0>(value[i]),get<1>(value[i]),get<2>(value[i]));

            total -= get<2>(value[i]);
            profit += get<1>(value[i]);
            weight += get<2>(value[i]);
        } else if ((get<2>(value[i]) >= total) && (total != 0)){
            float r = total/get<2>(value[i]);
            float p = get<1>(value[i])*r;
            sol[i] = make_tuple(get<0>(value[i]), p, r);

            total -= r;
            profit += p;
            weight += r;
        }

        if (total == 0)
            break;
    }

    cout<<endl<<"\nSolution :";
    for (int i=0; i<len; i++) {
        cout<<endl<<get<0>(sol[i])<<"   ";
        cout<<get<1>(sol[i]);
    }


    cout<<endl<<"Total value : "<<profit;
    cout<<endl<<"Total weight : "<<weight;
    cout<<endl;
}

int main () {
    int totalWeight, noOfSack;

    cout<<"Enter the total weight : ";
    cin>>totalWeight;
    cout<<"Enter the total no. of piles : ";
    cin>>noOfSack;

    string sack[noOfSack];
    cout<<"Enter piles : ";
    for(int i=0; i<noOfSack; i++) {
        cin>>sack[i];
    }

    int profit[noOfSack];
    cout<<"Enter value : ";
    for(int i=0; i<noOfSack; i++) {
        cin>>profit[i];
    }

    int weight[noOfSack];
    cout<<"Enter weight : ";
    for(int i=0; i<noOfSack; i++) {
        cin>>weight[i];
    }
    tuple <string, float, float> value[noOfSack];
    cout<<endl<<endl<<"Your tuple  :\n";
    for (int i=0; i<noOfSack; i++) {
        value[i] = make_tuple(sack[i], profit[i], weight[i]);
    }
    for (int i=0; i<noOfSack; i++) {
        cout<<endl<<get<0>(value[i])<<"   ";
        cout<<get<1>(value[i])<<"   ";
        cout<<get<2>(value[i]);
    }


    decrease(value, noOfSack);
    cout<<endl<<endl<<"After decrease : \n";
    for (int i=0; i<noOfSack; i++) {
        cout<<endl<<get<0>(value[i])<<"   ";
        cout<<get<1>(value[i])<<"   ";
        cout<<get<2>(value[i]);
    }

    make_profit(value, noOfSack, totalWeight);
    return 0;
}
```

Output :

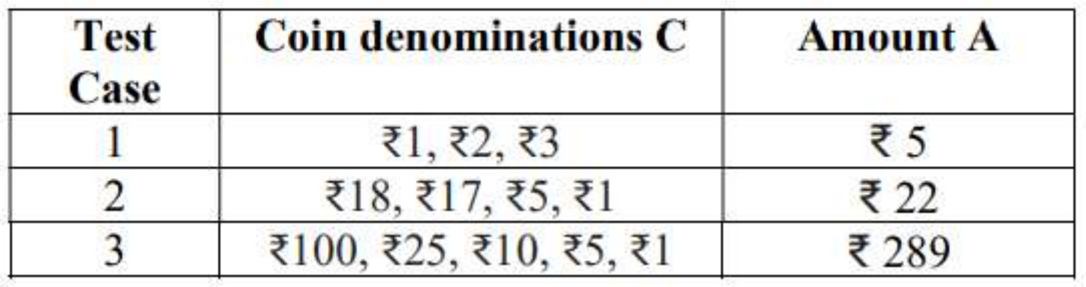


```
Enter the total weight : 15
Enter the total no. of piles : 5
Enter piles : A B C D E
Enter value : 4 2 2 1 10
Enter weight : 12 1 2 1 4


Your tuple  :

A   4   12
B   2   1
C   2   2
D   1   1
E   10   4

After decrease :

E   10   4
B   2   1
C   2   2
D   1   1
A   4   12

Solution :
E   10
B   2
C   2
D   1
A   2.33333
Total value : 17.3333
Total weight : 8.58333

Process returned 0 (0x0)   execution time : 14.115 s
Press any key to continue.
```

Conclusion : As we have seen in knapsack problem we can not take full portion of any item without using fractional knapsack problem. If we can use it then we can make more profit.

PRACTICAL 4.1 B

Aim : A cashier at any mall needs to give change of an amount to customers many times in a day. Cashier has multiple number of coins available with different denominations which is described by a set C. Implement the program for a cashier to find the minimum number of coins required to find a change of a particular amount A. Output should be the total number of coins required of given denominations. Check the program for following test cases:



Test Case	Coin denominations C	Amount A
1	₹1, ₹2, ₹3	₹ 5
2	₹18, ₹17, ₹5, ₹1	₹ 22
3	₹100, ₹25, ₹10, ₹5, ₹1	₹ 289

Is the output of Test case 2 is optimal? Write your observation

Software Requirement : Compiler, MS Excel

Theory :

The change-making problem addresses the **question of finding the minimum number of coins** (of certain denominations) that add up to a given amount of money. It is a special case of the integer knapsack problem, and has applications wider than just currency.

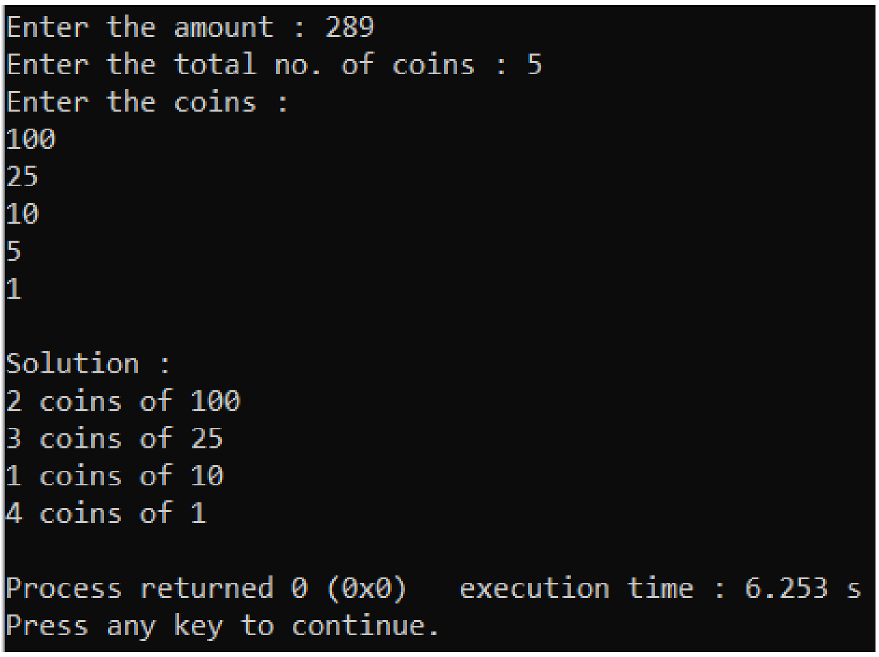
Algorithm :

1. Sort the array of coins in decreasing order.
2. Initialize result as empty.
3. Find the largest denomination that is smaller than current amount.
4. Add found denomination to result. Subtract value of found denomination from amount.
5. If amount becomes 0, then print result.
6. Else repeat steps 3 and 4 for new value of V.

CODE :

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

void decrease (int coins[], int len) {
    int temp;

    for (int i=0; i<len-1; i++) {
        for (int j=0; j<len-i-1; j++) {
                if (coins[j] < coins[j+1]) {
                temp = coins[j];
                coins[j] = coins[j+1];
                coins[j+1] = temp;
            }
        }
    }
}

void make_change (int amount, int coins[], int len) {
    int n = 0;
    pair <int, int> p[len];

    for (int i=0; i<len; i++) {
        if ((coins[i] <= amount) && (amount != 0)) {
            n = amount/coins[i];

            p[i] = make_pair(coins[i], n);

            amount = amount - n*coins[i];
            if (amount == 0)
                break;
        }
    }

    int c=0;
    for (int i = 0; i<len; i++) {
        if (p[i].second == 0)
            c++;
    }

    if ((c == len) || (amount != 0)) {
        cout<<"\nChange not possible";
    } else {
        cout<<endl<<"Solution : ";
        for (int i=0; i<len;i++) {
            if (p[i].second != 0)
                cout<<endl<<p[i].second<<" coins of "<<p[i].first;
        }
    }
    cout<<endl;
}

int main () {
    int amount, noOfCoins;

    cout<<"Enter the amount : ";
    cin>>amount;
    cout<<"Enter the total no. of coins : ";
    cin>>noOfCoins;

    int coins[noOfCoins];

    cout<<"Enter the coins :\n";
    for (int i=0; i<noOfCoins; i++) {
        cin>>coins[i];
    }

    decrease(coins, noOfCoins);

    make_change(amount, coins, noOfCoins);

    return 0;
}
```

<u>OUTPUT :</u>

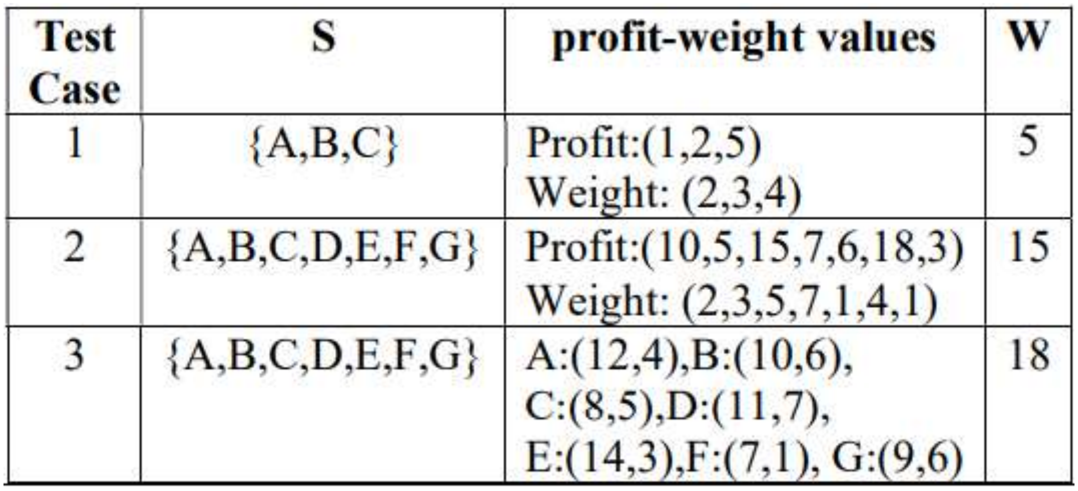


```
Enter the amount : 289
Enter the total no. of coins : 5
Enter the coins :
100
25
10
5
1

Solution :
2 coins of 100
3 coins of 25
1 coins of 10
4 coins of 1

Process returned 0 (0x0)   execution time : 6.253 s
Press any key to continue.
```

<u>CONCLUTION :</u> As we have seen that this method is better for some cases but it's not possible to use everywhere this. For example if we have to do total amount is 22 and the cons are 18, 17, 5 and 1. Then by this method we get one coin of 18and 4 coin of 1. But it's not optimal solution. We want one coin of 17 and 5. So this method is false for this case.

PRACTICAL 4.2

Aim : Let S be a collection of objects with profit-weight values. Implement the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight W. Check the program for following test cases:



Test Case	S	profit-weight values	W
1	{A,B,C}	Profit:(1,2,5) Weight: (2,3,4)	5
2	{A,B,C,D,E,F,G}	Profit:(10,5,15,7,6,18,3) Weight: (2,3,5,7,1,4,1)	15
3	{A,B,C,D,E,F,G}	A:(12,4),B:(10,6), C:(8,5),D:(11,7), E:(14,3),F:(7,1), G:(9,6)	18

Software Requirement : Compiler, MS Excel

Theory :

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. In **Fractional Knapsack**, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.

Algorithm :

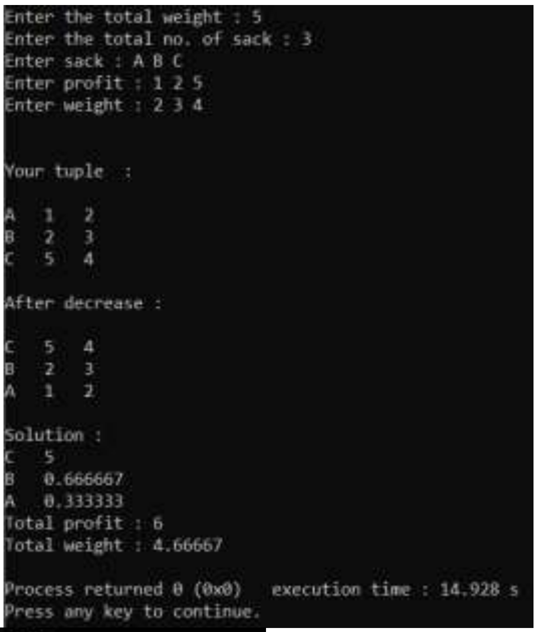
Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)

```
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
```

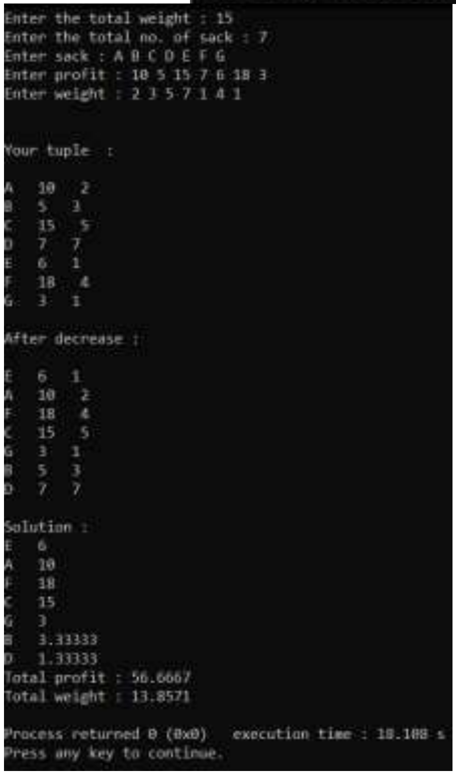
CODE :

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

using namespace std;

void decrease (tuple<string, float, float> value[], int len) {
    tuple<string, float, float> temp;
    for (int i=0; i<len-1; i++) {
        for (int j=0; j<len-i-1; j++) {
                float v =  get<1>(value[j])/get<2>(value[j]);
                float d =  get<1>(value[j+1])/get<2>(value[j+1]);
                if (v < d) {
                //if (get<1>(value[j]) < get<1>(value[j+1])) {
                temp = value[j];
                value[j] = value[j+1];
                value[j+1] = temp;
            }
        }
    }
}

void make_profit (tuple<string, float, float> value[], int len, int w) {
    float total = w;
    float profit = 0, weight = 0;
    tuple<string, float, float> sol[len];
    for (int i=0; i<len; i++) {
        if ((get<2>(value[i]) <= total) && (total != 0)) {
            sol[i] = make_tuple(get<0>(value[i]),get<1>(value[i]),get<2>(value[i]));

            total -= get<2>(value[i]);
            profit += get<1>(value[i]);
            weight += get<2>(value[i]);
        } else if ((get<2>(value[i]) >= total) && (total != 0)){
            float r = total/get<2>(value[i]);
            float p = get<1>(value[i])*r;
            sol[i] = make_tuple(get<0>(value[i]), p, r);

            total -= r;
            profit += p;
            weight += r;
        }

        if (total == 0)
            break;
    }

    cout<<endl<<"\nSolution :";
    for (int i=0; i<len; i++) {
        cout<<endl<<get<0>(sol[i])<<"   ";
        cout<<get<1>(sol[i]);
    }


    cout<<endl<<"Total profit : "<<profit;
    cout<<endl<<"Total weight : "<<weight;
    cout<<endl;
}

int main () {
    int totalWeight, noOfSack;

    cout<<"Enter the total weight : ";
    cin>>totalWeight;
    cout<<"Enter the total no. of sack : ";
    cin>>noOfSack;

    string sack[noOfSack];
    cout<<"Enter sack : ";
    for(int i=0; i<noOfSack; i++) {
        cin>>sack[i];
    }

    int profit[noOfSack];
    cout<<"Enter profit : ";
    for(int i=0; i<noOfSack; i++) {
        cin>>profit[i];
    }

    int weight[noOfSack];
    cout<<"Enter weight : ";
    for(int i=0; i<noOfSack; i++) {
        cin>>weight[i];
    }
    tuple <string, float, float> value[noOfSack];
    cout<<endl<<endl<<"Your tuple  :\n";
    for (int i=0; i<noOfSack; i++) {
        value[i] = make_tuple(sack[i], profit[i], weight[i]);
    }
    for (int i=0; i<noOfSack; i++) {
        cout<<endl<<get<0>(value[i])<<"   ";
        cout<<get<1>(value[i])<<"   ";
        cout<<get<2>(value[i]);
    }


    decrease(value, noOfSack);
    cout<<endl<<endl<<"After decrease : \n";
    for (int i=0; i<noOfSack; i++) {
        cout<<endl<<get<0>(value[i])<<"   ";
        cout<<get<1>(value[i])<<"   ";
        cout<<get<2>(value[i]);
    }

    make_profit(value, noOfSack, totalWeight);
    return 0;
}
```

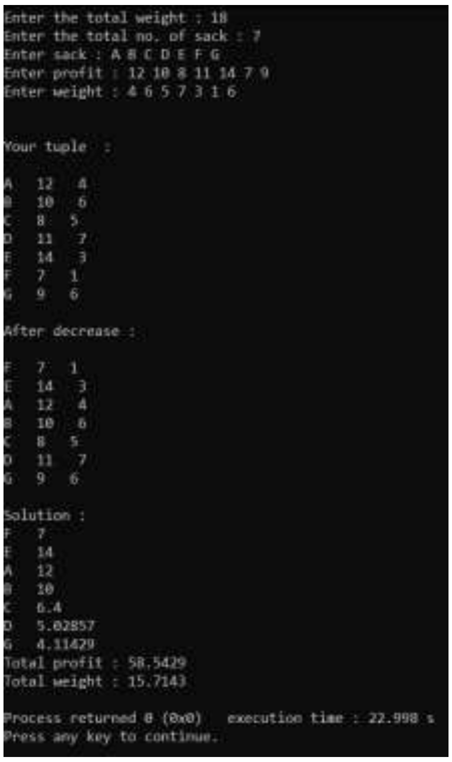
OUTPUT :



```
Enter the total weight : 5
Enter the total no. of sack : 3
Enter sack : A B C
Enter profit : 1 2 5
Enter weight : 2 3 4


Your tuple  :

A   1   2
B   2   3
C   5   4

After decrease :

C   5   4
B   2   3
A   1   2

Solution :
C   5
B   0.666667
A   0.333333
Total profit : 6
Total weight : 4.66667

Process returned 0 (0x0)   execution time : 14.928 s
Press any key to continue.
```



```
Enter the total weight : 15
Enter the total no. of sack : 7
Enter sack : A B C D E F G
Enter profit : 10 5 15 7 6 18 3
Enter weight : 2 3 5 7 1 4 1


Your tuple  :

A   10   2
B   5   3
C   15   5
D   7   7
E   6   1
F   18   4
G   3   1

After decrease :

E   6   1
A   10   2
F   18   4
C   15   5
G   3   1
B   5   3
D   7   7

Solution :
E   6
A   10
F   18
C   15
G   3
B   3.33333
D   1.33333
Total profit : 56.6667
Total weight : 13.8571

Process returned 0 (0x0)   execution time : 18.108 s
Press any key to continue.
```

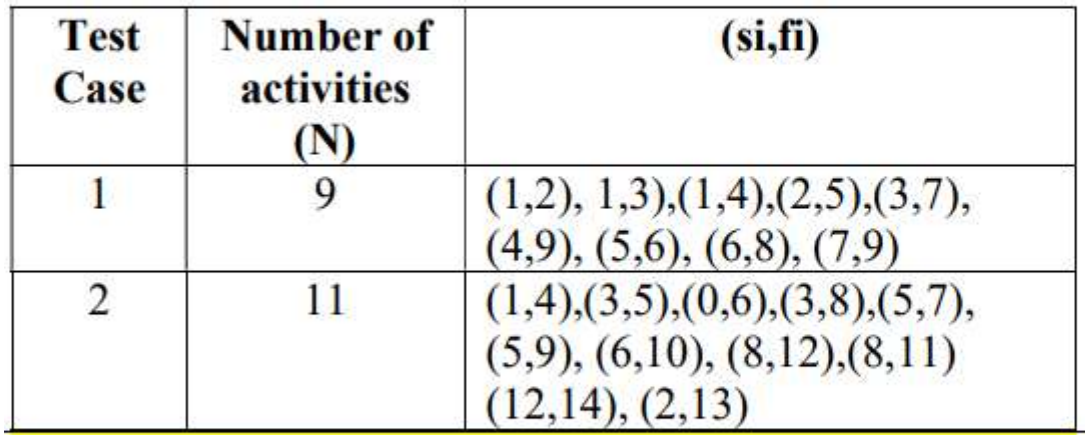


```
Enter the total weight : 18
Enter the total no. of sack : 7
Enter sack : A B C D E F G
Enter profit : 12 10 8 11 14 7 9
Enter weight : 4 6 5 7 3 1 6


Your tuple  :

A   12   4
B   10   6
C   8   5
D   11   7
E   14   3
F   7   1
G   9   6

After decrease :

F   7   1
E   14   3
A   12   4
B   10   6
C   8   5
D   11   7
G   9   6

Solution :
F   7
E   14
A   12
B   10
C   6.4
D   5.02857
G   4.11429
Total profit : 58.5429
Total weight : 15.7143

Process returned 0 (0x0)   execution time : 22.998 s
Press any key to continue.
```

CONCLUTION : As we have seen before the knapsack problem as compare to that this fractional method is more efficient for profit. So we have to used fractional knapsack problem instead of without fractional.

PRACTICAL 4.3

Aim : Suppose you want to schedule N activities in a Seminar Hall. Start time and Finish time of activities are given by pair of (si,fi) for ith activity. Implement the program to maximize the utilization of Seminar Hall. (Maximum activities should be selected.)



Test Case	Number of activities (N)	(si,fi)
1	9	(1,2), 1,3),(1,4),(2,5),(3,7), (4,9), (5,6), (6,8), (7,9)
2	11	(1,4),(3,5),(0,6),(3,8),(5,7), (5,9), (6,10), (8,12),(8,11) (12,14), (2,13)

Software Requirement : Compiler, MS Excel

Theory :

The activity selection problem is a combinatorial optimization problem concerning the selection of non-conflicting activities to perform within a given time frame, given a set of activities each marked by a start time and finish time.

Algorithm :

1)Sort the activities according to their finishing time

2)Select the first activity from the sorted array and print it.

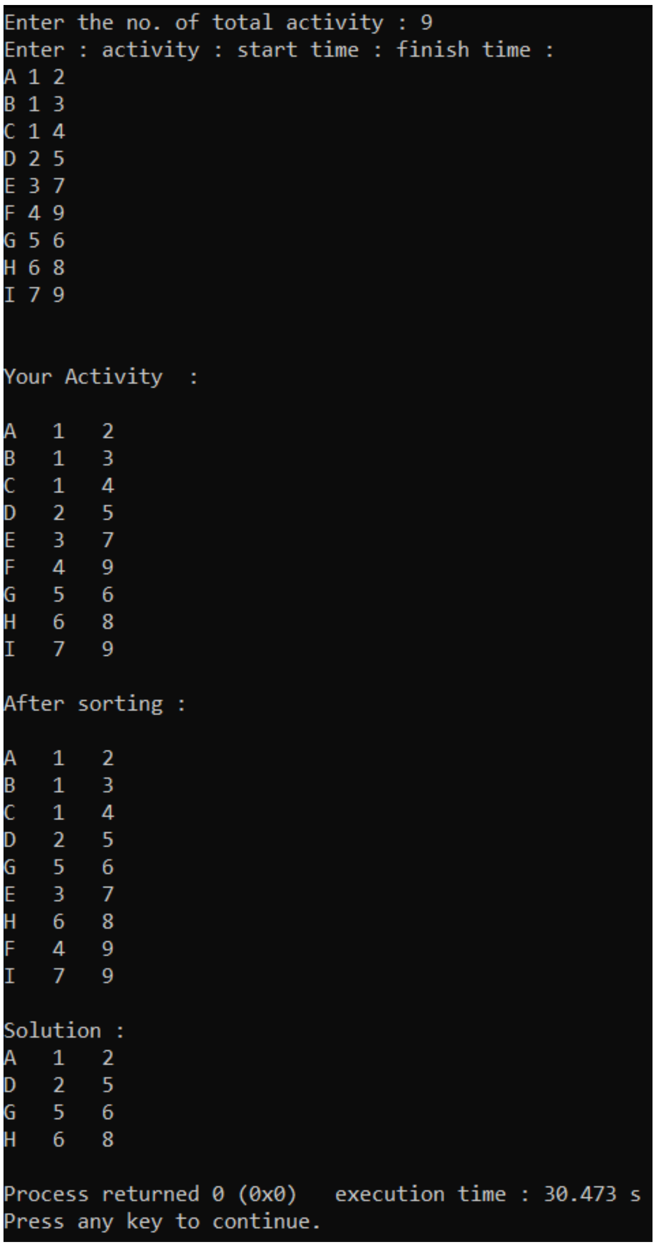
3)Do the following for the remaining activities in the sorted array.

a) If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it.

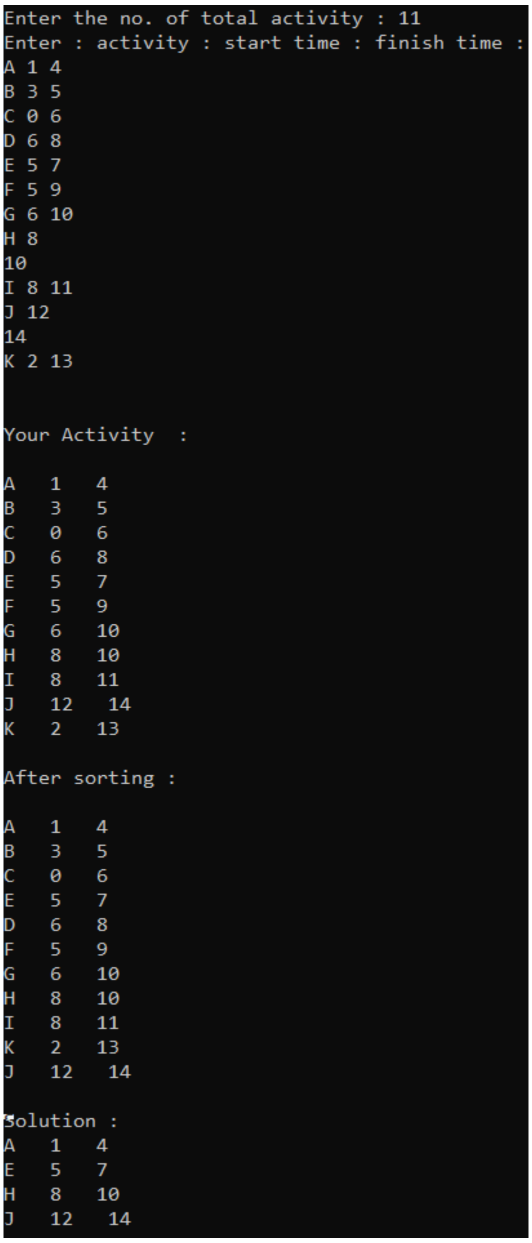
CODE :

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

using namespace std;

void decrease (tuple<string, int, int> selection[], int len) {
    tuple<string, int, int> temp;
    for (int i=0; i<len-1; i++) {
        for (int j=0; j<len-i-1; j++) {
                if (get<2>(selection[j]) > get<2>(selection[j+1])) {
                    temp = selection[j];
                    selection[j] = selection[j+1];
                    selection[j+1] = temp;
            }
        }
    }
}

void activity_selection (tuple<string, int, int> selection[], int len) {
    tuple<string, int, int> sol[len];
    int index = 0;
    for (int i=0; i<len; i++) {
        if (i==0) {
            sol[i] = make_tuple(get<0>(selection[i]),get<1>(selection[i]),get<2>(selection[i]));
            continue;
        } else {
            if (get<2>(selection[index]) <= get<1>(selection[i])) {
                sol[i] = make_tuple(get<0>(selection[i]),get<1>(selection[i]),get<2>(selection[i]));
                index = i;
            }
        }
    }

    cout<<endl<<"\nSolution :";
    for (int i=0; i<len; i++) {
        if (get<2>(sol[i]) != 0)
            cout<<endl<<get<0>(sol[i])<<"   "<<get<1>(sol[i])<<"   "<<get<2>(sol[i]);
    }
    cout<<endl;
}

int main () {
    int totalActivity;

    cout<<"Enter the no. of total activity : ";
    cin>>totalActivity;

    string activity[totalActivity];
    int startTime[totalActivity];
    int finishTime[totalActivity];

    cout<<"Enter : activity : start time : finish time :\n";
    for(int i=0; i<totalActivity; i++) {
        cin>>activity[i];
        cin>>startTime[i];
        cin>>finishTime[i];
    }

    tuple <string, int, int> selection[totalActivity];

    for (int i=0; i<totalActivity; i++) {
        selection[i] = make_tuple(activity[i], startTime[i], finishTime[i]);
    }

    cout<<endl<<endl<<"Your Activity  :\n";

    for (int i=0; i<totalActivity; i++) {
        cout<<endl<<get<0>(selection[i])<<"                    "<<get<1>(selection[i])<<"
"<<get<2>(selection[i]);
    }

    decrease(selection, totalActivity);
    cout<<endl<<endl<<"After sorting : \n";
    for (int i=0; i<totalActivity; i++) {
        cout<<endl<<get<0>(selection[i])<<"                    "<<get<1>(selection[i])<<"
"<<get<2>(selection[i]);
    }

    activity_selection(selection, totalActivity);
    return 0;
}
```

OUTPUT :



```
Enter the no. of total activity : 9
Enter : activity : start time : finish time :
A 1 2
B 1 3
C 1 4
D 2 5
E 3 7
F 4 9
G 5 6
H 6 8
I 7 9


Your Activity  :

A   1   2
B   1   3
C   1   4
D   2   5
E   3   7
F   4   9
G   5   6
H   6   8
I   7   9

After sorting :

A   1   2
B   1   3
C   1   4
D   2   5
G   5   6
E   3   7
H   6   8
F   4   9
I   7   9

Solution :
A   1   2
D   2   5
G   5   6
H   6   8

Process returned 0 (0x0)   execution time : 30.473 s
Press any key to continue.
```



```
Enter the no. of total activity : 11
Enter : activity : start time : finish time :
A 1 4
B 3 5
C 0 6
D 6 8
E 5 7
F 5 9
G 6 10
H 8
10
I 8 11
J 12
14
K 2 13


Your Activity  :

A   1   4
B   3   5
C   0   6
D   6   8
E   5   7
F   5   9
G   6   10
H   8   10
I   8   11
J   12   14
K   2   13

After sorting :

A   1   4
B   3   5
C   0   6
E   5   7
D   6   8
F   5   9
G   6   10
H   8   10
I   8   11
K   2   13
J   12   14

Solution :
A   1   4
E   5   7
H   8   10
J   12   14
```

CONCLUTION : As we define activity selection method so this method is used in school or any office work for meetings and lecture scheduling. We can easily manage that thing using this method.

PRACTICAL 4.4

Aim : In a conference, N people from a company XYZ are present to attend maximum number of presentations. Each presentation is scheduled between 8:00 and 8:00. Here the second 8:00 means 8:00 pm. Our task is to assign people to presentations such that the number of unique presentations attended by XYZ as a company is maximized. Input Format Input is provided in the form a file (taken as command line argument). The first line contains N i.e # of people. Second line contains M i.e # of presentation on that day. M lines follow each containing start and end time of presentation. Time will be in format of HH:MM. Sample Input
#01
2
5
09:00 08:00
08:00 12:00
12:00 08:00
08:00 08:00
08:00 08:00
Sample Output#01 3 Explanation#01 One person can cover the 8- 12 presentation and the 12-8 presentation. The other person can cover one of the all-day presentations.

Software Requirement : Compiler, MS Excel

Theory :

The activity selection problem is a combinatorial optimization problem concerning the selection of non-conflicting activities to perform within a given time frame, given a set of activities each marked by a start time and finish time.

Algorithm :

1) Sort the activities according to their finishing time

2) Select the first activity from the sorted array and print it.

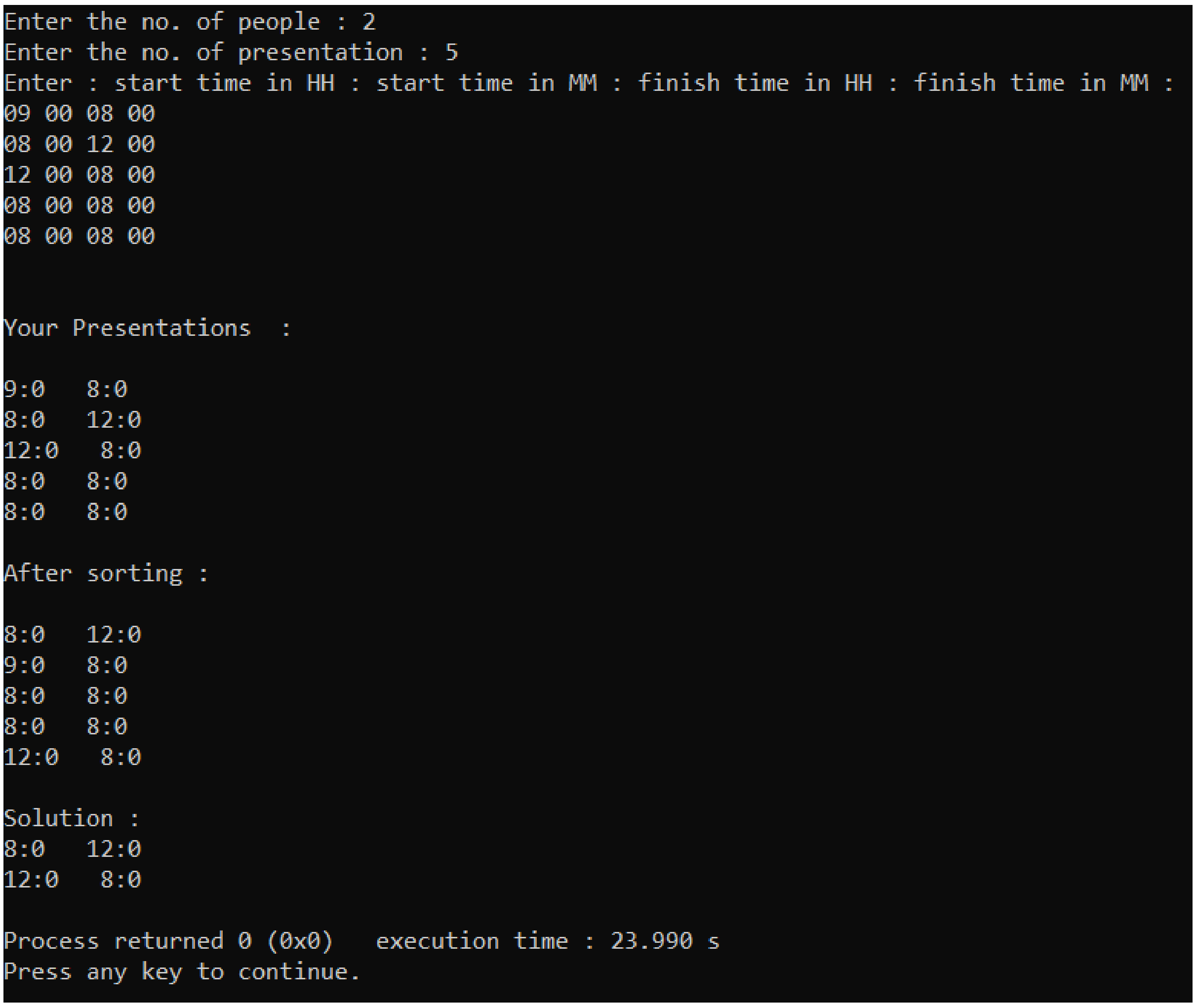
3) Do the following for the remaining activities in the sorted array.

  a) If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it.

CODE :

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

using namespace std;

void decrease (tuple<int, int, int, int> selection[], int len) {
    tuple<int, int, int, int> temp;
        for (int j=0; j<len-1; j++) {
                if (get<0>(selection[j]) > get<0>(selection[j+1])) {
                    temp = selection[j];
                    selection[j] = selection[j+1];
                    selection[j+1] = temp;
            }
        }
}

void activity_selection (tuple<int, int, int, int> selection[], int len) {
    tuple<int, int, int, int> sol[len];
    int index = 0;
    for (int i=0; i<len; i++) {
        if (i==0) {
            sol[i]                                                              =
make_tuple(get<0>(selection[i]),get<1>(selection[i]),get<2>(selection[i]),get<3>(selection[i]
));
            continue;
        } else {
            if (get<2>(selection[index]) <= get<0>(selection[i])) {
                sol[i]                                                          =
make_tuple(get<0>(selection[i]),get<1>(selection[i]),get<2>(selection[i]),get<3>(selection[i]
));
                index = i;
            }
        }
    }

    cout<<endl<<"\nSolution :";
    for (int i=0; i<len; i++) {
        if ((get<0>(sol[i]) != 0) && (get<2>(sol[i]) != 0))
            cout<<endl<<get<0>(sol[i])<<":"<<get<1>(sol[i])<<"
"<<get<2>(sol[i])<<":"<<get<3>(sol[i]);
    }
    cout<<endl;
}

int main () {
    int totalActivity, totalPerson;

    cout<<"Enter the no. of people : ";
    cin>>totalPerson;
    cout<<"Enter the no. of presentation : ";
    cin>>totalActivity;

    int startTimeInHH[totalActivity];
    int startTimeInMM[totalActivity];
    int finishTimeInHH[totalActivity];
    int finishTimeInMM[totalActivity];

    cout<<"Enter : start time in HH : start time in MM : finish time in HH : finish time in MM
:\n";
    for(int i=0; i<totalActivity; i++) {
        cin>>startTimeInHH[i];
        cin>>startTimeInMM[i];
        cin>>finishTimeInHH[i];
        cin>>finishTimeInMM[i];
    }

    tuple <int, int, int, int> selection[totalActivity];

    for (int i=0; i<totalActivity; i++) {
        selection[i] = make_tuple(startTimeInHH[i], startTimeInMM[i], finishTimeInHH[i],
finishTimeInMM[i]);
    }

    cout<<endl<<endl<<"Your Presentations  :\n";

    for (int i=0; i<totalActivity; i++) {
        cout<<endl<<get<0>(selection[i])<<":"<<get<1>(selection[i])<<"
"<<get<2>(selection[i])<<":"<<get<3>(selection[i]);
    }

    decrease(selection, totalActivity);
    cout<<endl<<endl<<"After sorting : \n";
    for (int i=0; i<totalActivity; i++) {
        cout<<endl<<get<0>(selection[i])<<":"<<get<1>(selection[i])<<"
"<<get<2>(selection[i])<<":"<<get<3>(selection[i]);
    }

    activity_selection(selection, totalActivity);
    return 0;
}
```

OUTPUT :



```
Enter the no. of people : 2
Enter the no. of presentation : 5
Enter : start time in HH : start time in MM : finish time in HH : finish time in MM :
09 00 08 00
08 00 12 00
12 00 08 00
08 00 08 00
08 00 08 00


Your Presentations  :

9:0   8:0
8:0   12:0
12:0   8:0
8:0   8:0
8:0   8:0

After sorting :

8:0   12:0
9:0   8:0
8:0   8:0
8:0   8:0
12:0   8:0

Solution :
8:0   12:0
12:0   8:0

Process returned 0 (0x0)   execution time : 23.990 s
Press any key to continue.
```

CONCLUTION : As we seen in previous practiccal this method is same as that. This is the one type of use it eith real life example. This method is very useful for anyone for time sceduling.