Practical:4

Greedy Approach

4.1

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

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
int n,a;
void findMin(int sum,int coins[])
{
   int ans[sum],j=0,i=0;

   for(i=n-1;i>=0;i--)
   {
      while(sum>=coins[i])
      {
         sum=sum-coins[i];
         ans[j]=coins[i];
         j++;
      }
   }
if(sum==0)
{
```

```
            cout<<"Denomination sequence for amount of "<<a<<" are :";
    for(i=0;i<j;i++)
        cout<<ans[i]<<" ";
}
else if(sum!=0)
{
    cout<<"Not possible to achieve required Amount with given denominations ..!";
}
}
void Sort(int arr[], int n)
{
    int temp;
    if (n == 1)
        return;
    for (int i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
            {
              temp=arr[i];
              arr[i]=arr[i+1];
              arr[i+1]=temp;
            }
    Sort(arr, n-1);
}
int main()
{
    int deno[n];
    int i ;
            cout<<"Enter the total amount :-";
            cin>>a;
            cout<<"Enter the total number of denominations :";
            cin>>n;
            cout<<"Enter the denominations :";
            for(i=0;i<n;i++)
            {
               cin>>deno[i];
            }
            Sort(deno,n);
            findMin(a,deno);
            return 0;
}
```

OUTPUT:

```
Enter the total amount :-5
Enter the total number of denominations :3
Enter the denominations :1
2
3
Denomination sequence for amount of 5 are :3 2

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

```
Enter the total amount :-22
Enter the total number of denominations :4
Enter the denominations :18
17
5
1
Denomination sequence for amount of 22 are :18 1 1 1 1

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

```
Enter the total amount :-289
Enter the total number of denominations :5
Enter the denominations :100
25
10
5
1
Denomination sequence for amount of 289 are :100 100 25 25 25 10 1 1 1 1

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

CONCLUSION:

This problem is a variation of 'Coin Change Problem'. Worst case: When the only coin present is Rs. 1 Coin. So in that case, time complexity becomes O (A) where A is the amount to be paid.

4.2

AIM: Let S be a collection of objects with profit-weight values. Implement the fractionalKnap sack 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

PROGRAM:

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

int main(){
   int n, weight;
cout<< "enter the number of element : ";
cin>>n ;
   float p[n],w[n];
   float pw[n];
cout<< "enter the Profit for all " << n << " elements." <<endl;
for(int i=0;i<n;i++){
cout<< "enter the profit of "<< i+1 <<"thelement : " ;
cin>> p[i];
```

```
cout<< "enter the weight of "<< i+1 <<"thelement : " ;
cin>> w[i];
        pw[i] = p[i]/w[i];
    }
cout<<endl<< "Enter the maximum weight : ";
cin>> weight;

for(int i=0;i<n;i++){
for(int j=0;j<n-1;j++){
            if(pw[j] < pw[j+1]){
                float temp = pw[j];
                pw[j] = pw[j+1];
                pw[j+1] = temp;

                temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;

                temp = w[j];
                w[j] = w[j+1];
                w[j+1] = temp;
            }
        }
    }
cout<<endl<< "Considering weight of : " <<endl;
    float ans_p =0;
    int ans = 0, i = 0;
    while((ans+w[i]) <= weight){
ans = ans + w[i];
```

```
ans_p = ans_p + p[i];
cout<< w[i]<<"  "<< p[i] << "  "<< pw[i] <<"  "<<endl;
i++;
   }
if(ans<= weight){
      float temp1 = weight-ans;
      float temp2 = temp1/w[i];
ans_p = ans_p + (p[i]*temp2);
cout<<"Weight : "<< temp1 << " profit : "<< (p[i]*temp2) <<" this is the " <<temp1<<" part of having weight " <<w[i]<<" and profit " << p[i]<<endl;

   }
cout<<endl<< "Final max profit is : "<<ans_p<<endl;
}
```

OUTPUT:

```
enter the number of element : 3
enter the Profit for all 3 elements.
enter the profit of 1thelement : 1
enter the weight of 1thelement : 2
enter the profit of 2thelement : 2
enter the weight of 2thelement : 3
enter the profit of 3thelement : 5
enter the weight of 3thelement : 4

Enter the maximum weight : 5

Considering weight of :
4  5  1.25
Weight : 1 profit : 0.666667 this is the 1 part of having weight 3 and profit 2

Final max profit is : 5.66667
```

```
enter the number of element : 7
enter the Profit for all 7 elements.
enter the profit of 1thelement : 10
enter the weight of 1thelement : 2
enter the profit of 2thelement : 5
enter the weight of 2thelement : 3
enter the profit of 3thelement : 15
enter the weight of 3thelement : 5
enter the profit of 4thelement : 7
enter the weight of 4thelement : 7
enter the profit of 5thelement : 6
enter the weight of 5thelement : 1
enter the profit of 6thelement : 18
enter the weight of 6thelement : 4
enter the profit of 7thelement : 3
enter the weight of 7thelement : 1

Enter the maximum weight : 15

Considering weight of :
1  6  6
2  10  5
4  18  4.5
5  15  3
1  3  3
Weight : 2 profit : 3.33333 this is the 2 part of having weight 3 and profit

Final max profit is : 55.3333
```

```
enter the number of element : 7
enter the Profit for all 7 elements.
enter the profit of 1thelement : 12
enter the weight of 1thelement : 4
enter the profit of 2thelement : 10
enter the weight of 2thelement : 6
enter the profit of 3thelement : 8
enter the weight of 3thelement : 5
enter the profit of 4thelement : 11
enter the weight of 4thelement : 7
enter the profit of 5thelement : 14
enter the weight of 5thelement : 3
enter the profit of 6thelement : 7
enter the weight of 6thelement : 1
enter the profit of 7thelement : 9
enter the weight of 7thelement : 6

Enter the maximum weight : 18

Considering weight of :
1  7  7
3  14  4.66667
4  12  3
6  10  1.66667
Weight : 4 profit : 6.4 this is the 4 part of having weight 5 and profit 8

Final max profit is : 49.4
```

CONCLUSION:

In this practical we implemented the concept of knapsack problem.

4.3

AIM:-Suppose you want to schedule N activities in a SeminarHall. Start time and Finish time of activities are given bypair 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)

PROGRAM CODE:-

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
typedef struct {
  int s;
  int f;
} activ;
void input(activ arr[],int lng) {
  cout << "Enter total "<< lng <<" Item's Start and Finish time :-\n\n";
  for(int i = 0; i < lng; i++) {
    cout << "Enter Start Time For Activity "<< i+1<<":" ;
    cin >> arr[i].s;
    cout << "Enter Finish Time For Activity "<< i+1<<":";
    cin >> arr[i].f;
    cout<<"\n";
  }
}
void display(activ arr[], int lng) {
  int i;
  cout << "Start Time: ";
  for(i = 0; i < lng; i++) {
    cout << "\t"<< arr[i].s ;
  }
  cout << endl << "Finish Time: ";
  for (i = 0; i < lng; i++) {
```

```
      cout << "\t"<< arr[i].f ;
   }
   cout << endl;
}
bool compare(activ a, activ b) {
    return a.f < b.f;
}
void MaxAct(activ arr[], int n)
{
    sort(arr, arr+n, compare);

    cout << "Following activities are selected :\n";
    int i = 0;
    cout << "(" << arr[i].s << ", " << arr[i].f << "), ";
    for (int j = 1; j < n; j++)
    {
      if (arr[j].s >= arr[i].f)
      {
          cout << "(" << arr[j].s << ", "
<< arr[j].f << "), ";
          i = j;
      }
    }
}
int main() {
   int n;
   cout<<"Enter total number of Activities :";
   cin>>n;
   activ arr[n];
   input(arr, n);
   cout << "Entered data \n";
   display(arr,n);
   MaxAct(arr, n);
}
```

OUTPUT:

```
Enter total number of Activities :9
Enter total 9 Item's Start and Finish time :-

Enter Start Time For Activity 1:1
Enter Finish Time For Activity 1:2

Enter Start Time For Activity 2:1
Enter Finish Time For Activity 2:3

Enter Start Time For Activity 3:1
Enter Finish Time For Activity 3:4

Enter Start Time For Activity 4:2
Enter Finish Time For Activity 4:5

Enter Start Time For Activity 5:3
Enter Finish Time For Activity 5:7

Enter Start Time For Activity 6:4
Enter Finish Time For Activity 6:9

Enter Start Time For Activity 7:5
Enter Finish Time For Activity 7:6

Enter Start Time For Activity 8:6
Enter Finish Time For Activity 8:8

Enter Start Time For Activity 9:7
Enter Finish Time For Activity 9:9

Entered data
Start Time:     1       1       1       2       3       4       5     7
Finish Time:    2       3       4       5       7       9       6     9
Following activities are selected :
(1, 2), (2, 5), (5, 6), (6, 8),
```

```
Enter total number of Activities :11
Enter total 11 Item's Start and Finish time :-

Enter Start Time For Activity 1:1
Enter Finish Time For Activity 1:4

Enter Start Time For Activity 2:3
Enter Finish Time For Activity 2:5

Enter Start Time For Activity 3:0
Enter Finish Time For Activity 3:6

Enter Start Time For Activity 4:3
Enter Finish Time For Activity 4:8

Enter Start Time For Activity 5:5
Enter Finish Time For Activity 5:7

Enter Start Time For Activity 6:5
Enter Finish Time For Activity 6:9

Enter Start Time For Activity 7:6
Enter Finish Time For Activity 7:10

Enter Start Time For Activity 8:8
Enter Finish Time For Activity 8:12

Enter Start Time For Activity 9:8
Enter Finish Time For Activity 9:11

Enter Start Time For Activity 10:12
Enter Finish Time For Activity 10:14

Enter Start Time For Activity 11:2
Enter Finish Time For Activity 11:13

Entered data
Start Time:     1       3       0       3       5       5       6       8       8       12      2
Finish Time:    4       5       6       8       7       9       10      12      11      14      1
3
Following activities are selected :
(1, 4), (5, 7), (8, 11), (12, 14),
```

CONCLUSION:-In this practical we have attempted to code a cpp program to solve the activity selection problem. The problem is based on getting maximum activities completed such that maximum amount of the schedule can be utilized .In the problem definition we are given to solve this problem for seminar hall. Here greedy approach is used to solve the problem which allows to complete maximum activities so that maximum schedule time of seminar hall can be used.