**Aim** :

Given a bunch of projects, where every project has a deadline and associated profit if the project is finished before the deadline. It is also given that every project takes one month duration, so the minimum possible deadline for any project is 1 month. In what way the total profits can be maximized if only one project can be scheduled at a time.

**Description** :

We have been given an array of jobs with their respective Id, dead\_line and profits. Our job here is to maximise the profits such that only one project can be scheduled at a time.

The strategy that will be used to solve here is greedy strategy. And also here each project can is completed in one month duration.

**Algorithm** :

Algorithm JobScheduling(a[],n)

{

//a[] is the list of objects containing jobs with their respective Ids, deadlines and profits;

//First sort the job id’s with respect to their profits

slots[i]:=0

for i:=0 to n do{

for j:=min(n,a[i].dead\_line)-1 to 0 step -1 do{

if(slots[j]:=false){

slots[j]:=true

results[j]:=i

break

}

}

}

}

**Code** :

#include<bits/stdc++.h>

using namespace std;

class Job{

public:

string id;

int dead\_line;

int profits;

};

bool cmp(Job a,Job b){

return (a.profits>b.profits);

}

void jobScheduling(Job list[],int n){

int result[n];

bool slots[n];//Initially initialize all the slots to false

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

slots[i]=0;

}

sort(list,list+n,cmp);

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

for(int j=min(n,list[i].dead\_line)-1;j>=0;j--){

if(slots[j]==false){

slots[j]=true;

result[j]=i;

break;

}

}

}

int max\_profit=0;

cout<<"Order of execution : ";

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

if(slots[i]){

cout<<list[result[i]].id<<" ";

max\_profit+=list[result[i]].profits;

}

}

cout<<"\nMaximum Profit : "<<max\_profit;

}

int main(){

int n;

cout<<"Enter number of items : ";

cin>>n;

cout<<"Enter id/deadline/profits : \n";

int limit;

// cout<<"Enter the limit of your deadline : ";

// cin>>limit;

Job obj[n];

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

cout<<"Job "<<i+1<<" : ";

// obj[i].dead\_line=rand()%limit+1;

// obj[i].profits=rand()%500+1;

// cin>>obj[i].id;

// cout<<obj[i].dead\_line<<" "<<obj[i].profits<<"\n";

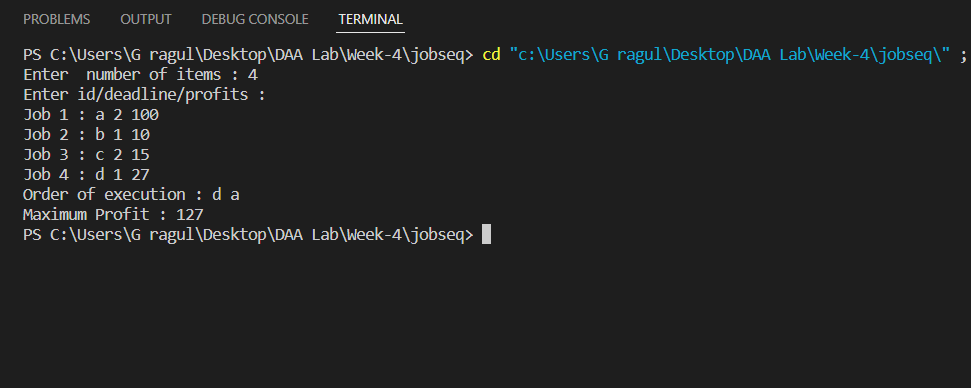
cin>>obj[i].id>>obj[i].dead\_line>>obj[i].profits;

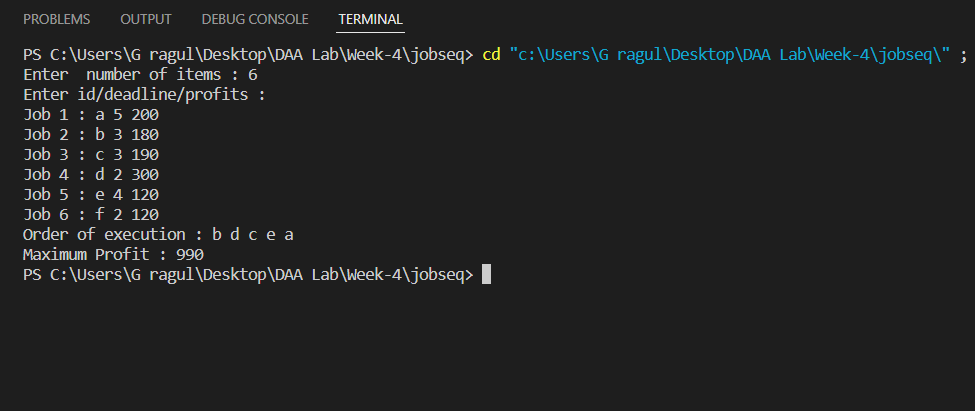
}

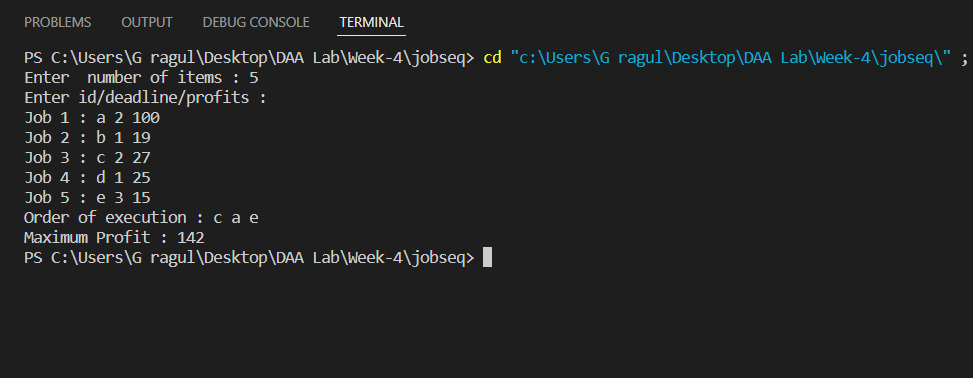
jobScheduling(obj,n);

}

**Result Analysis** :







**Time Complexity** :

Here as we go through each and every job and try to find the the free slots for each and every job , we use two for loops as a result the time complexity comes up to O(n^2)

**Space Complexity** :

O(n) where n is the number of jobs.

**Conclusion** :

Using the job scheduling algorithm we can prioritize the jobs and optimize maximum profit from it.

Also, the algorithms time complexity can be optimized using priority queue(max heap).