

PRACTICAL 4

AIM: Greedy Approach

4.1)

(A) 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.

(B) 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.

4.2) 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 .

4.3) Suppose you want to schedule N activities in a Seminar Hall. Start time and Finish time of activities are given by pair of (s_i, f_i) for i th activity. Implement the program to maximize the utilization of Seminar Hall. (Maximum activities should be selected.)

4.4) 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.

SOFTWARE REQUIREMENT:

C++11 Compiler, MS Word.

HARDWARE REQUIREMENT:

Desktop Computer.

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.

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.

The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this 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  $\text{weight} + w[i] \leq W$ 
    then  $x[i] = 1$ 
    weight = weight +
     $w[i]$  else
     $x[i] = (W - \text{weight}) /$ 
     $w[i]$  weight =  $W$ 
  br
eak
return
x

```

Code:

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

struct Golddust{
    int value,weight;

    Golddust(int value,int weight){
        this->value=value;
        this->weight=weight;
    }
};

bool cmp(struct Golddust a,struct Golddust b)
{
    double r1 = (double) a.value / (double) a.weight;
    double r2 = (double) b.value / (double) b.weight;
    return r1>r2;
}

double FractionalKnapsack(int W,struct Golddust arr[],int n){

    sort(arr,arr+n,cmp);
    int curWeight=0;
    double finalvalue=0.0;

    for(int i=0;i<n;i++){
        if((curWeight + arr[i].weight) <= W){
            curWeight+=arr[i].weight;
            finalvalue+=arr[i].value;
        }
        else{
            int remain=W-curWeight;
            finalvalue+=arr[i].value*((double)remain / (double)arr[i].weight);
            break;
        }
    }
    return finalvalue;
}

int main(){
    int W=15;
    Golddust arr[]={ {4,12},{2,1},{2,2},{1,1},{10,4}};
    int n=sizeof(arr)/sizeof(arr[0]);

    cout<<"Maximum Gold dust he can take is "<<FractionalKnapsack(W,arr,n);
    return 0;
}
```

OUTPUT:

```
Maximum Gold dust he can take is =17.3333
```

CONCLUSION

In this practical I have implemented the Greedy Algorithm concepts in a c++ program for this problem which is similar to fractional Knapsack Problem to find the optimal solution using Greedy Algorithm. I have observed that this algorithm is not giving optimal solution for every inputs.

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.

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.

THEORY:

Input: V = 70

Output: 2

We need a 50 Rs note and a 20 Rs note.

Input: V = 121

Output: 3

We need a 100 Rs note, a 20 Rs note and a 1 Rs coin.

Approach: A common intuition would be to take coins with greater value first. This can reduce the total number of coins needed. Start from the largest possible denomination and keep adding denominations while the remaining value is greater than 0.

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 <bits/stdc++.h>
using namespace std;

// All denominations of Indian Currency
// int deno[] = { 1,2,3 };
int deno[] = { 18,17,5,1 };
// int deno[] = { 100,25,10,5,1 };
int n = sizeof(deno) / sizeof(deno[0]);

void findMin(int V)
{
    sort(deno, deno + n); // nlogn

    vector<int> ans;

    for (int i = n - 1; i >= 0; i--) {

        while (V >= deno[i]) {
            V -= deno[i];
            ans.push_back(deno[i]);
        }
    }

    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << " ";
}

int main()
{
    // int n = 5;
    int n = 22;
    // int n = 289;
    cout << "Following is minimal"
        << " number of change for " << n
        << ": ";
    findMin(n);
    return 0;
}
```

OUTPUT:

```
Following is minimal number of change for 5: 3 2
```

```
Following is minimal number of change for 22: 18 1 1 1 1
```

```
Following is minimal number of change for 289: 100 100 25 25 25 10 1 1 1 1
```

OBSERVATION:

Here from our calculation, we can see that for first test case 1 solution is optimal.

But for the test case 2 for change of 22 we can calculate the optimal solution is 17 and 5 coins. But the program is giving 18 1 1 1 1 which is wrong.

So, we can see that Greedy Algorithm not always give optimal solution.

CONCLUSION

In this practical I have implemented the Greedy Algorithm concepts in a c++ program for this problem which is similar to 0/1 Knapsack Problem to find the optimal solution using Greedy Algorithm. I have observed that this algorithm is not giving optimal solution for every inputs.

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.

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

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.

The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this 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
br
eak
return
x

```


Code:

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

struct fknapsack{
    int value,weight;

    fknapsack(int value,int weight){
        this->value=value;
        this->weight=weight;
    }
};

bool cmp(struct fknapsack a,struct fknapsack b)
{
    double r1 = (double) a.value / (double) a.weight;
    double r2 = (double) b.value / (double) b.weight;
    return r1>r2;
}

double FractionalKnapsack(int W,struct fknapsack arr[],int n){

    sort(arr,arr+n,cmp);
    int curWeight=0;
    double finalvalue=0.0;

    for(int i=0;i<n;i++){
        if((curWeight + arr[i].weight) <= W){
            curWeight+=arr[i].weight;
            finalvalue+=arr[i].value;
        }
        else{
            int remain=W-curWeight;
            finalvalue+=arr[i].value*((double)remain / (double)arr[i].weight);
            break;
        }
    }
    return finalvalue;
}

int main(){
    // int W=5;
    // int W=15;
    int W=18;
```

```
// fknapsack arr[]={ {1,2},{2,3},{5,4}};  
// fknapsack arr[]={ {10,2},{5,3},{15,5},{7,7},{6,1},{18,4},{3,1}};  
fknapsack arr[]={ {12,4},{10,6},{8,5},{11,7},{14,3},{7,1},{9,6}};  
int n=sizeof(arr)/sizeof(arr[0]);  
  
cout<<"Maximum Profit is "<<FractionalKnapsack(W,arr,n);  
return 0;  
}
```

OUTPUT:

```
Maximum Profit is =5.66667
```

```
Maximum Profit is =55.3333
```

```
Maximum Profit is =49.4
```

CONCLUSION

In this practical I have implemented the Greedy Algorithm concepts in a c++ program for this Fractional Knapsack Problem to find the optimal solution using Greedy Algorithm. I have observed that this algorithm is not giving optimal solution for every inputs.

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)

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.

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 ScheduleActivity(int s[], int f[], int n)
{
    int i, j;
    cout << "Selected Activities" << endl;
    i = 0;
    cout << " " << i;
    for (j = 1; j < n; j++)
    {
        if (s[j] >= f[i])
        {
            cout << " " << j;
            i = j;
        }
    }
}

int main()
{
#ifdef ONLINE_JUDGE
    //for getting input from input.txt
    freopen("input1.txt", "r", stdin);
    //for writing output from output.txt
    freopen("output1.txt", "w", stdout);
#endif

    //(1,2), (1,3),(1,4),(2,5),(3,7),(4,9), (5,6), (6,8), (7,9)
    //(1,4),(3,5),(0,6),(3,8),(5,7),(5,9), (6,10), (8,12),(8,11),(12,14), (2,1
    3)

    //Input 1
    // int s[] = {1, 1, 1, 2, 3, 4, 5, 6, 7};
    // int f[] = {2, 3, 4, 5, 7, 9, 6, 8, 9};

    //Input2
    // int s[] = {1, 3, 0, 3, 5, 5, 6, 8, 8, 12, 2};
    // int f[] = {4, 5, 6, 8, 7, 9, 10, 12, 11, 14, 13};

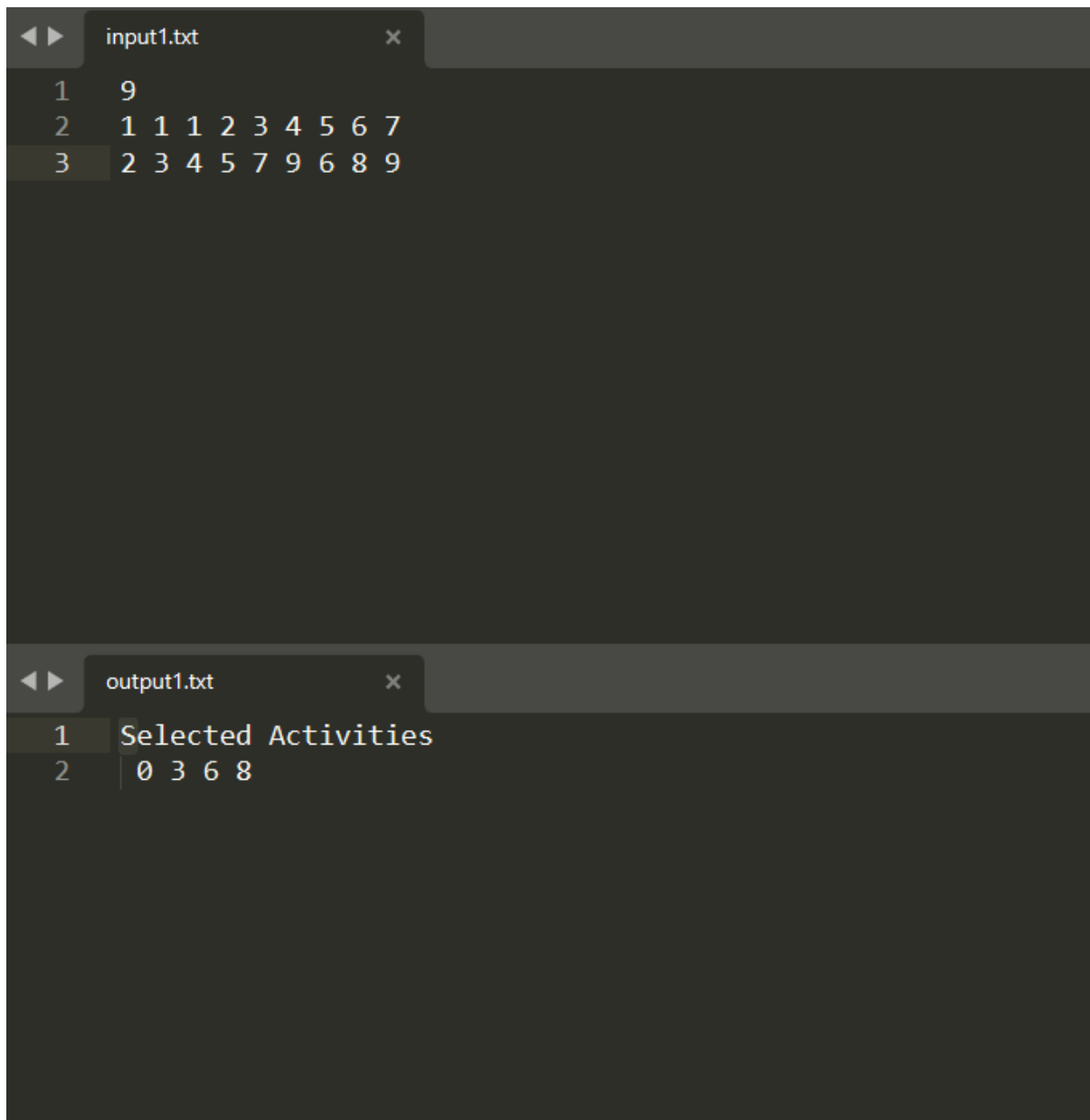
    int n;
    cin >> n;
    int *s = new int(n);
    int *f = new int(n);
```

```
// cout<<"Enter Start Times:";
for (int i = 0; i < n; i++) {
    cin >> s[i];
}
// cout<<"Enter Finish Times:";
for (int j = 0; j < n; j++) {
    cin >> f[j];
}

// int n = sizeof(s) / sizeof(s[0]);

ScheduleActivity(s, f, n);

return 0;
}
```

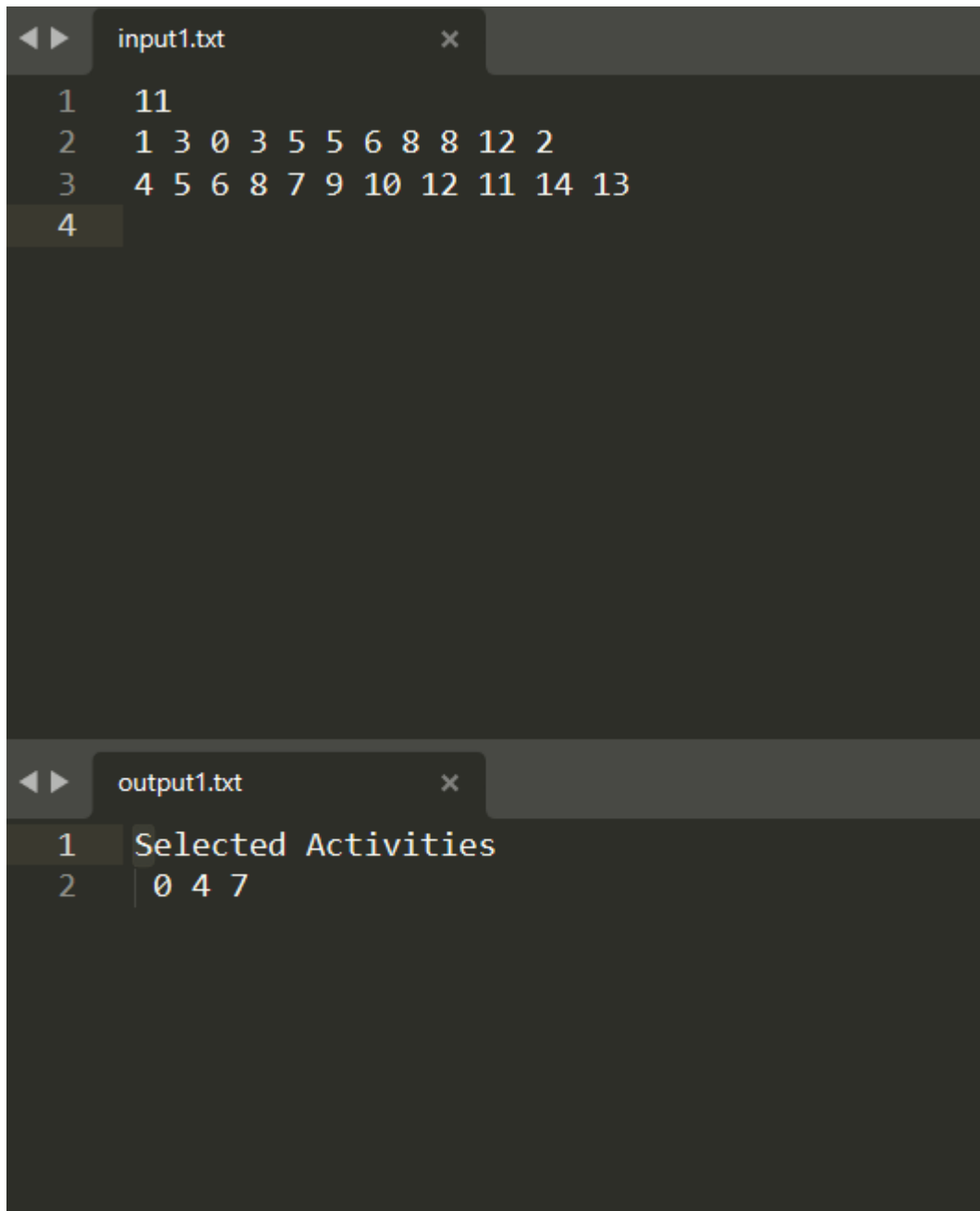
OUTPUT:

The screenshot shows a text editor with two tabs: 'input1.txt' and 'output1.txt'. The 'input1.txt' tab is active and displays the following content:

```
1 9
2 1 1 1 2 3 4 5 6 7
3 2 3 4 5 7 9 6 8 9
```

The 'output1.txt' tab is also visible and displays the following content:

```
1 Selected Activities
2 0 3 6 8
```



The image shows a code editor with two tabs: 'input1.txt' and 'output1.txt'. The 'input1.txt' tab is active and displays the following content:

```
1 11
2 1 3 0 3 5 5 6 8 8 12 2
3 4 5 6 8 7 9 10 12 11 14 13
4
```

The 'output1.txt' tab is also visible and displays the following content:

```
1 Selected Activities
2 0 4 7
```

CONCLUSION

In this practical I have implemented the Greedy Algorithm concepts in a c++ program for this problem which is similar Job scheduling Problem to find the optimal solution using Greedy Algorithm. I have observed that this algorithm is not giving optimal solution for every inputs.

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.

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:

- 4) Sort the activities according to their finishing time
- 5) Select the first activity from the sorted array and print it.
- 6) 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;

struct node
{
    int s;
    int e;
};

bool comp(node a, node b)
{
    return (a.e < b.e);
}

int main()
{
    #ifndef ONLINE_JUDGE
        freopen("input1.txt", "r", stdin);
        freopen("output1.txt", "w", stdout);
    #endif

    int n;
    //scanf("%d",&n);
    cin >> n;
    int p;
    //scanf("%d",&p);
    cin >> p;
    vector<int> v[n];
    vector<node> v2;
    for (int i = 0; i < p; i++)
    {
        string str;
        cin >> str; //09:00
        //cout<<"hi"<<endl;
        char h1, h2, m1, m2;
        //cin>>h1>>h2;
        h1 = str[0];    //"0"
        h2 = str[1];    //"9"
        int t = 0;
        t = t * 10 + h1 - '0'; //0
        t = t * 10 + h2 - '0'; //9
        //cout<<"t: "<<t<<endl;
```

```
int s = 0;
if (t >= 8)
{
    s += (t - 8) * 60; //60
}
else
{
    s += (t + 4) * 60;
}
char z, z2;
//cin>>z;
//cin>>m1>>m2;
m1 = str[3]; //0
m2 = str[4]; //0
t = 0;
t = t * 10 + m1 - '0'; //0
t = t * 10 + m2 - '0'; //0
s += t; //60

cin >> str;
h1 = str[0]; //0
h2 = str[1]; //8
m1 = str[3]; //0
m2 = str[4]; //0
//cin>>z;
//cin>>h1>>h2>>z>>m1>>m2;

t = 0;
t = t * 10 + h1 - '0'; //0
t = t * 10 + h2 - '0'; //8
//cout<<"h1: "<<h1<<" "<<h2<<" "<<t<<endl;
int e = 0;
if (t > 8)
{
    e += (t - 8) * 60;
}
else
{
    e += (t + 4) * 60; //720
}

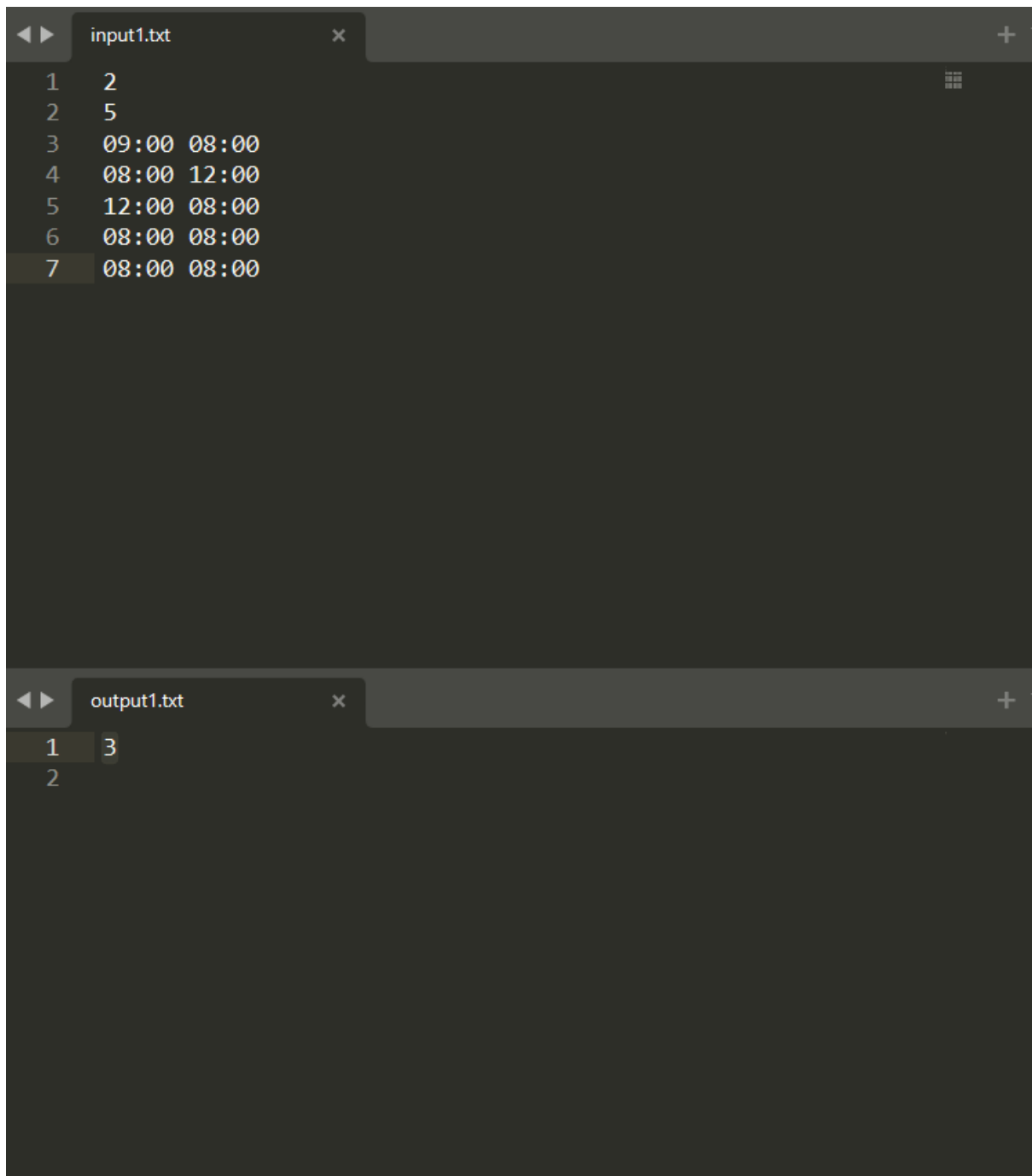
//cin>>z;
//cin>>m1>>m2;
int t2 = 0;
t2 = t2 * 10 + m1 - '0'; //0
t2 = t2 * 10 + m2 - '0'; //0
```

```
    if (t == 8 && t2 > 0)
    {
        e = 0;
        e += (t - 8) * 60;
    }
    e += t2;//720
    node temp;
    temp.s = s;
    temp.e = e;
    v2.push_back(temp);
    //cout<<s<<" "<<e<<endl;

}

sort(v2.begin(), v2.end(), comp);
for (int i = 0; i < n; i++)
{
    v[i].push_back(0);
}

for (int i = 0; i < p; i++)
{
    int f = 0;
    int max = -1;
    int ind = -1;
    for (int j = 0; j < n; j++)
    {
        if (v2[i].s >= v[j][v[j].size() - 1] && v[j][v[j].size() - 1] > max)
        {
            max = v[j][v[j].size() - 1];
            ind = j;
        }
    }
    if (ind != -1)
        v[ind].push_back(v2[i].e);
}
int ans = 0;
for (int i = 0; i < n; i++)
{
    ans = ans + v[i].size() - 1;
}
//int ans = func(v2,v,n,p,0);
cout << ans << endl;
//cin>>n;
return 0;
}
```

OUTPUT:

The image shows two text files. The first file, 'input1.txt', contains seven lines of data. The first two lines are '1 2' and '2 5'. The next five lines are '3 09:00 08:00', '4 08:00 12:00', '5 12:00 08:00', '6 08:00 08:00', and '7 08:00 08:00'. The second file, 'output1.txt', contains two lines: '1 3' and '2'.

```
input1.txt
1 2
2 5
3 09:00 08:00
4 08:00 12:00
5 12:00 08:00
6 08:00 08:00
7 08:00 08:00

output1.txt
1 3
2
```

CONCLUSION

In this practical I have implemented the Greedy Algorithm concepts in a c++ program for this problem which is similar Job scheduling Problem to find the optimal solution using Greedy Algorithm. I have observed that this algorithm is not giving optimal solution for every inputs.