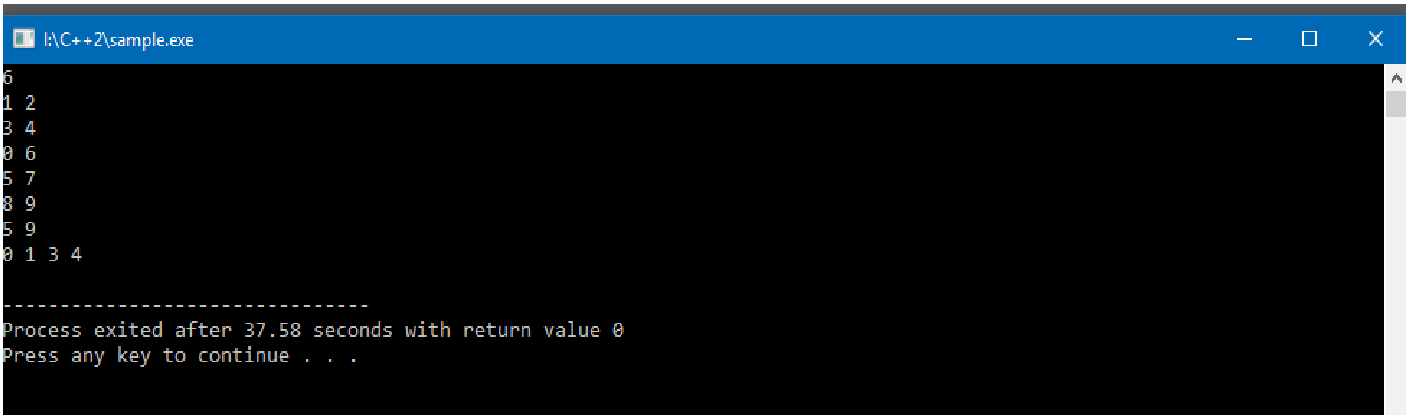
<u>LAB – 6</u>

1. Activity Selection Problem: Assume there exist n activities with each of them being represented by a start time si and finish time f . Two activities i and j are said to be non-conflicting if $si \geq fj$ or $sj \geq f$. Write a Greedy-based program to select the maximum number of non-conflicting activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. The time complexity of your designed program should not exceed ($nlogn$) time. Sample Input: No. of activities: 6 Job J0 J1 J2 J3 J4 J5 Start Time 1 3 0 5 8 5 Finish Time 2 4 6 7 9 9 Sample Output: Activities that can be executed are {J0, J1, J3, J4}

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

using namespace std;

int main()

{

        int n;

        cin>>n;

        pair<int,int> p[n+1];

        map <pair<int,int>, int> m;

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

        {

            int s,f;

            cin>>s>>f;

            p[i]=make_pair(f,s);

            m[p[i]]=i;

        }

        sort(p,p+n);

        vector <int> v;
```

```
    v.push_back(m[p[0]]);

    int z=p[0].first;

    for(int i=1;i<n;i++)

    {

        if(p[i].second>=z)

        {

            v.push_back(m[p[i]]);

            z=p[i].first;

        }

    }

    for(int i=0;i<v.size();i++)

    cout<<v[i]<<" ";

    cout<<"\n";

}
```



```
6
1 2
3 4
0 6
5 7
8 9
5 9
0 1 3 4

--------------------------------
Process exited after 37.58 seconds with return value 0
Press any key to continue . . .
```

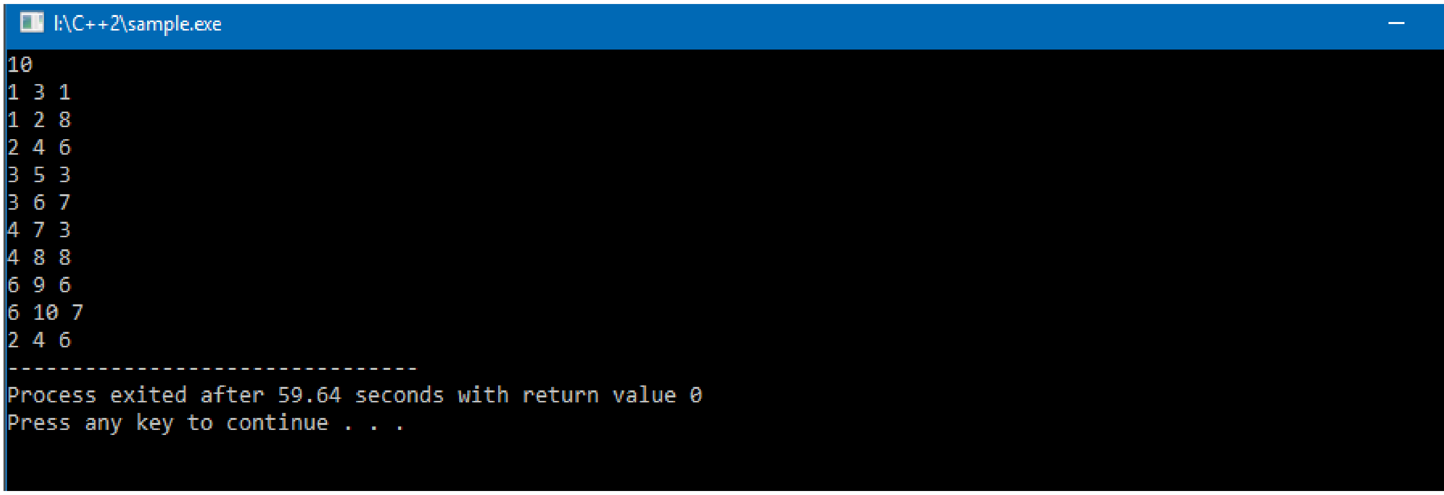
2. . Tree Vertex Splitting Problem: Consider a network of power line transmission. This network is represented by a directed and weighted binary tree.

Input Output Sample Input: No. of vertex: 10 Power loss: Node 1 to Node 2: 8 Node 1 to Node 3: 1 Node 2 to Node 4: 6…

Tolerance Limit: 10

Sample Output: Nodes where boosters are needed to be placed: 2, 4, 6

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
        int n;
        cin>>n;
        vector <pair<int,int> > v[n+1];
        v[0].push_back(make_pair(0,1));
        for(int i=0;i<n-1;i++)
        {
                int x,y,w;
                cin>>x>>y>>w;
                v[x].push_back(make_pair(w,y));
        }
        int t=n;
        int i=0;
        bool vis[n+1];
        memset(vis,false,sizeof(vis));
        int ed[n+1];
        ed[0]=0;
        while(t--)
        {
                int z,z1,z2;
                if(v[i].size()==1)
                {
                        z=v[i][0].second;
                        if(vis[i]==false)
                        ed[z]=ed[i]+v[i][0].first;
                        else
                        ed[z]=v[i][0].first;
                        if(ed[z]>10)
                        vis[i]=true;
                        else
                        vis[i]=false;
                }
                else if(v[i].size()==2)
                {
                        z1=v[i][0].second;
                        z2=v[i][1].second;
                        if(vis[i]==false)
                        {
                        ed[z1]=ed[i]+v[i][0].first;
                        ed[z2]=ed[i]+v[i][1].first;
                        }
                        else
                        {
                        ed[z1]=v[i][0].first;
                        ed[z2]=v[i][1].first;
                        }
                        int s=max(ed[z1],ed[z2]);
                        if(s>10)
                        vis[i]=true;
                        else
                        vis[i]=false;
                }
                i++;
        }
        for(int i=1;i<=n;i++)
        {
                if(vis[i]==true)
                cout<<i<<" ";
        }
}
```



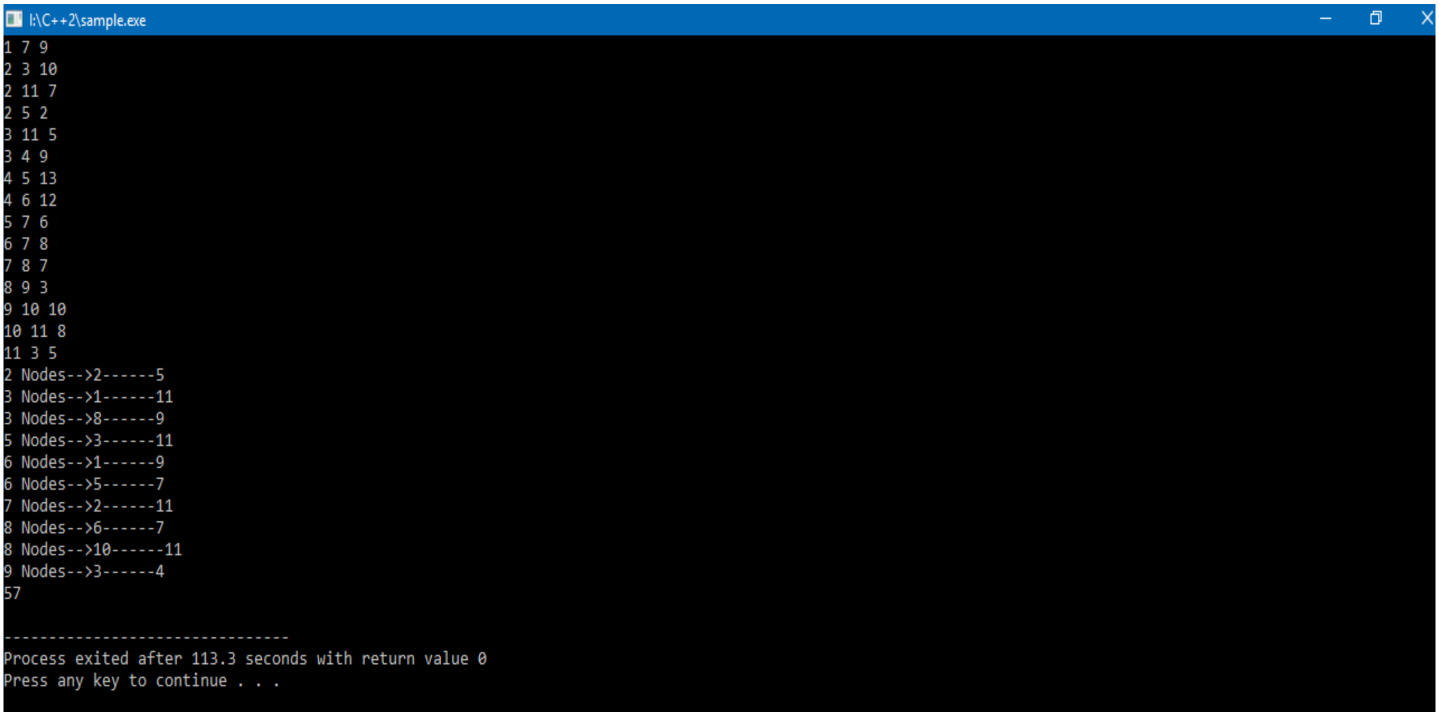
```
10
1 3 1
1 2 8
2 4 6
3 5 3
3 6 7
4 7 3
4 8 8
6 9 6
6 10 7
2 4 6
--------------------------------
Process exited after 59.64 seconds with return value 0
Press any key to continue . . .
```

3. Minimum Cost Spanning Tree: Given the weighted and undirected graph as input, store the attributes of the graph, i.e. the vertices and edges, in the memory using appropriate data structure to achieve better time complexity. Then find the minimum cost spanning tree using Kruskal’s algorithm. Also construct the minimum cost spanning tree.

Sample Input: Number of vertices: 11 Number of edges: 20 Enter source vertex: 1 Enter destination vertex: 2 Enter weight: 8 … Sample Output: Edges in the constructed MST: Edge Cost 2 3 … …

```
#include<bits/stdc++.h>
using namespace std;
int pp[1000];
int root(int x)
{
    while(pp[x]!=x)
    {
        pp[x]=pp[pp[x]];
        x=pp[x];
    }
    return x;
```

```
}
void unionn(int x,int y)
{
    int a=root(x);
    int b=root(y);
    pp[b]=a;
}
int main()
{
    int n;
    cin>>n;
    int e;
    cin>>e;
    pair<int,pair<int,int> > p;
    vector <pair<int,pair<int,int> > > v;
    for(int i=0;i<e;i++)
    {
        int x,y,w;
        cin>>x>>y>>w;
        p=make_pair(w,make_pair(x,y));
        v.push_back(p);
    }
    for(int i=1;i<=n;i++)
    pp[i]=i;
    sort(v.begin(),v.begin()+v.size());
    vector <pair<int,pair<int,int> > > vv;
    int minn=0;
    for(int i=0;i<e;i++)
    {
        int x,y,w;
        w=v[i].first;
        x=v[i].second.first;
        y=v[i].second.second;
        if(root(x)!=root(y))
        {
            p=make_pair(w,make_pair(x,y));
            vv.push_back(p);
            minn+=w;
            unionn(x,y);
        }
    }
    for(int i=0;i<vv.size();i++)
    cout<<vv[i].first<<" Nodes-->"<<vv[i].second.first<<"------"<<vv[i].second.second<<"\n";
    cout<<minn<<"\n";
}
```



```
1 7 9
2 3 10
2 11 7
2 5 2
3 11 5
3 4 9
4 5 13
4 6 12
5 7 6
6 7 8
7 8 7
8 9 3
9 10 10
10 11 8
11 3 5
2 Nodes-->2------5
3 Nodes-->1------11
3 Nodes-->8------9
5 Nodes-->3------11
6 Nodes-->1------9
6 Nodes-->5------7
7 Nodes-->2------11
8 Nodes-->6------7
8 Nodes-->10------11
9 Nodes-->3------4
57

--------------------------------
Process exited after 113.3 seconds with return value 0
Press any key to continue . . .
```