NOVEMBER 01,2020

DESIGN AND ANALYSIS OF ALGORITHMS

(**VIVA 2: ALGORITHM LAB**)

SUBMITTED BY-
NAME - RAJ KRISHNA
ROLL NO - 1805233
BATCH - CSE-G1
GROUP - B

1. **Execute the LCS program on pairs of input sequences of length 10,100 and 1000 and report the number of steps used in each case. Each element in the sequence is a random single digit number in the range 0 to 9.**

Code-

```
#include <bits/stdc++.h>
using namespace std;
long long int counter=0;
void lcsAlgo(char *S1, char *S2, int m, int n) {
  int LCS[m + 1][n + 1];

  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        ++counter;                          //incrementing steps
            if (i == 0 || j == 0)
            LCS[i][j] = 0;
            else
            if (S1[i - 1] == S2[j - 1])
                LCS[i][j] = LCS[i - 1][j - 1] + 1;
            else
                LCS[i][j] = max(LCS[i - 1][j], LCS[i][j - 1]);
    }
  }

  int index = LCS[m][n];                    //this portion is for printing LCS
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';
  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (S1[i - 1] == S2[j - 1]) {
      lcsAlgo[index - 1] = S1[i - 1];
      i--;
      j--;
```

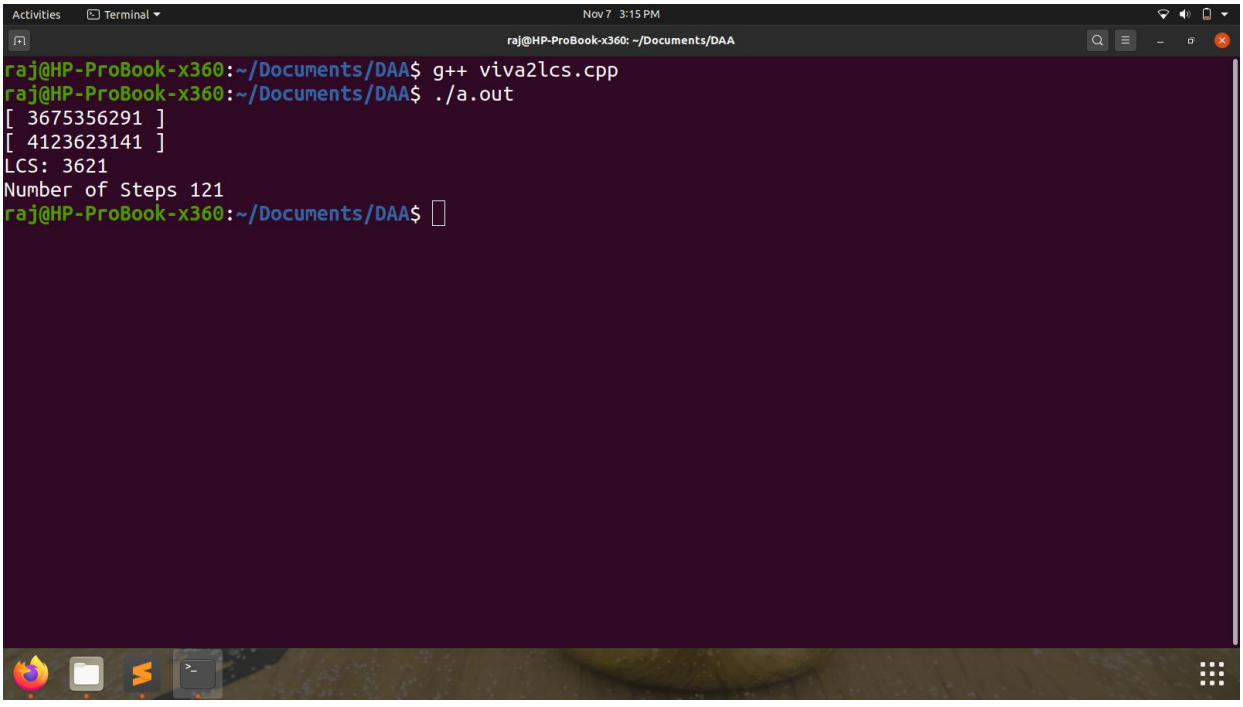
```
      index--;
    }
    else
                if (LCS[i - 1][j] > LCS[i][j - 1])
                i--;
                else
                j--;
  }
  cout << "[ " << S1 << " ]\n[ " << S2 << " ]\nLCS: " << lcsAlgo << "\n";
}

int main()
{
        string str1 ("");                   //string str1 declaration
        int i,j=0;
        for(i=0; i < 10; i++){              // i < 10, i < 100, i < 1000
        j = rand()%10;                      //generating number from rand()
        str1+=to_string(j);                 //converting int to string and
concatenate it in str1
        }
        char S1[str1.size() + 1];               //character array declaration
        strcpy(S1, str1.c_str());               //copying elements to array
(converting string to char array)
        int m = strlen(S1);

        srand((unsigned int)time(NULL));          //setting seed for the
random number generator

        string str2 ("");                   //string str2 declaration
        for(i=0; i < 10; i++){              // i < 10, i < 100, i < 1000
        j = rand()%10;                      //generating number from rand()
        str2+=to_string(j);                 //converting int to string and
concatenate it in str2
```

```
        }
        char S2[str2.size() + 1];              //character array declaration
        strcpy(S2, str2.c_str());              //copying elements to array
(converting string to char array)
        int n = strlen(S2);

        lcsAlgo(S1, S2, m, n);
        cout<<"Number of Steps "<<counter<<endl;                    //number
of steps
}
```

Output-

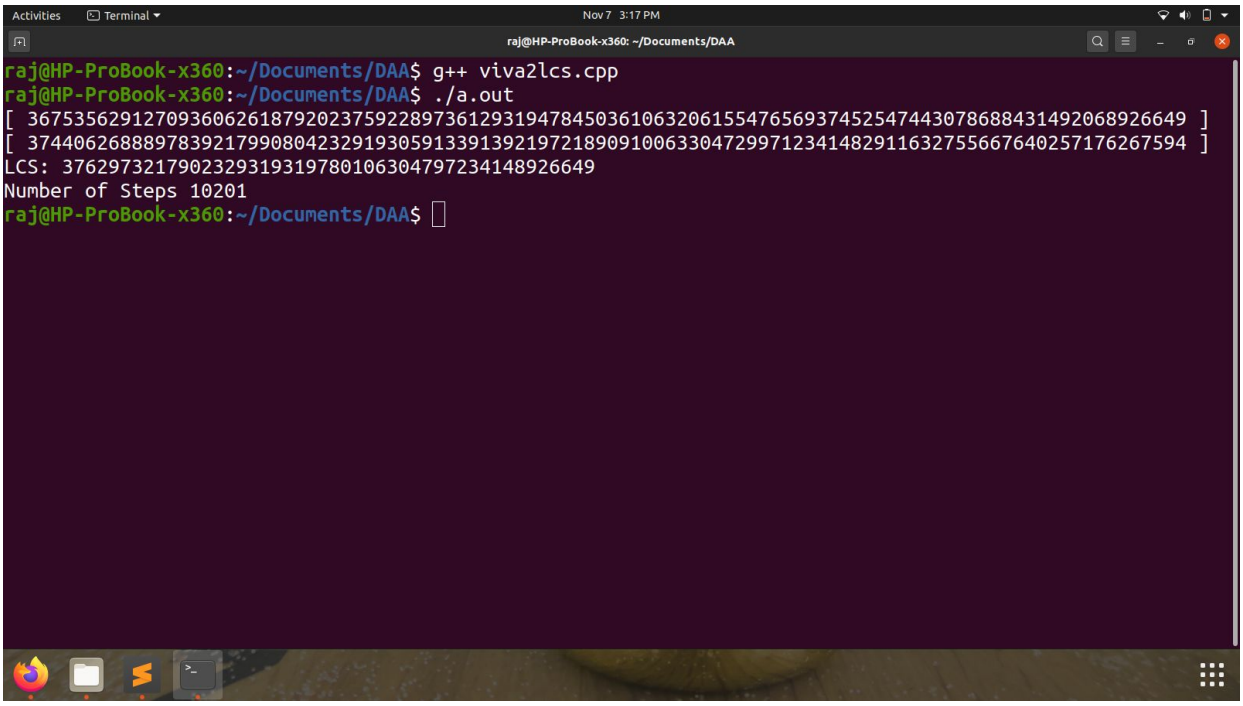
For Length 10- **// for(i=0; i< 10;++i)**

```
[ 3675356291 ]
[ 4123623141 ]
LCS: 3621
Number of Steps 121
```



For Length 100- **// for(i=0; i< 100;++i)**

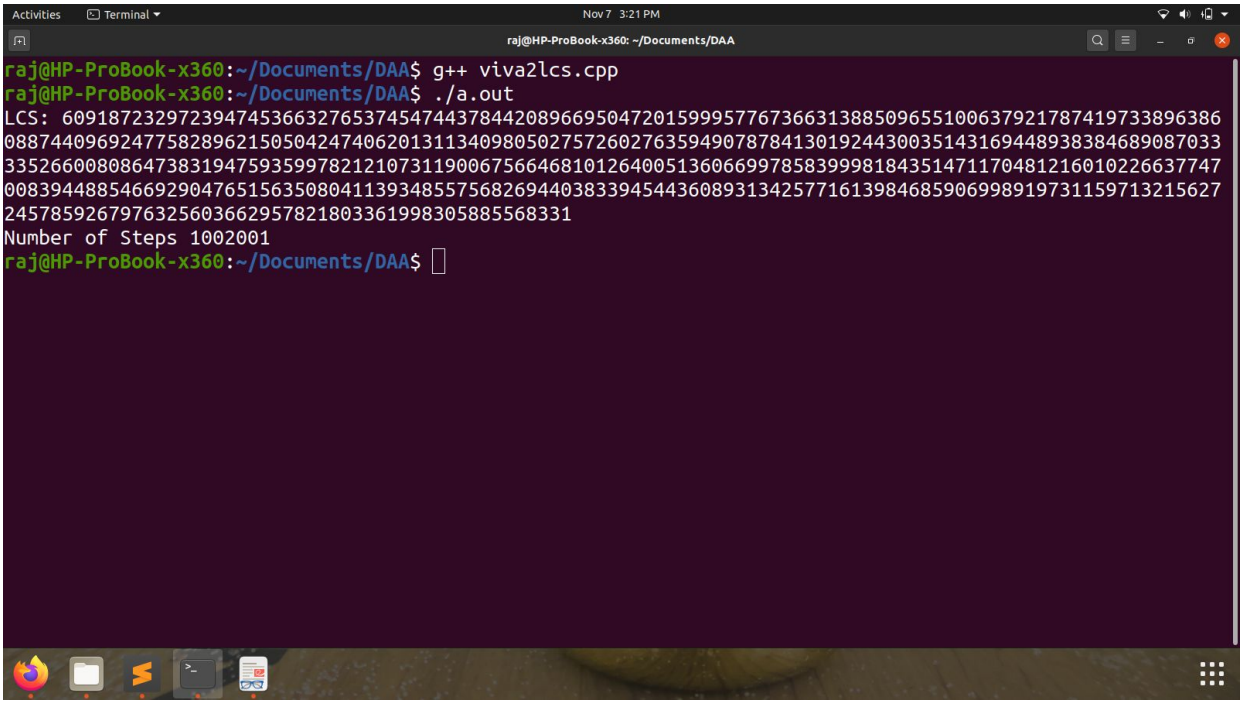
[
3675356291270936062618792023759228973612931947845036106320615547656937452547443078688431492068926649]
[
3744062688897839217990804232919305913391392197218909100633047299712341482911632755667640257176267594]
LCS: 376297321790232931931978010630479723414892664 9
Number of Steps 10201



For Length 1000- **// for(i=0; i< 1000;++i)**

LCS:
6091872329723947453663276537454744378442089669504720159995 7
7673663138850965510063792178741973389638608874409692477582 8
9621505042474062013113409805027572602763594907878413019244 3
0035143169448938384689087033335266008086473831947593599782 1

2107311900675664681012640051360669978583999818435147117048121601022663774700839448854669290476515635080411393485575682
6944038339454436089313425771613984685906998919731159713215627
2724578592679763256036629578218033619983058855683311
Number of Steps 1002001



For string length of 1000 it was not possible to print string S1 & S2 so i commented that line and printed the LCS only.

2. **Count and compare the number of steps used in Prim's and Kruskal's MST algorithms for the input used in lab exercises.**

Code of Prim's Algo-

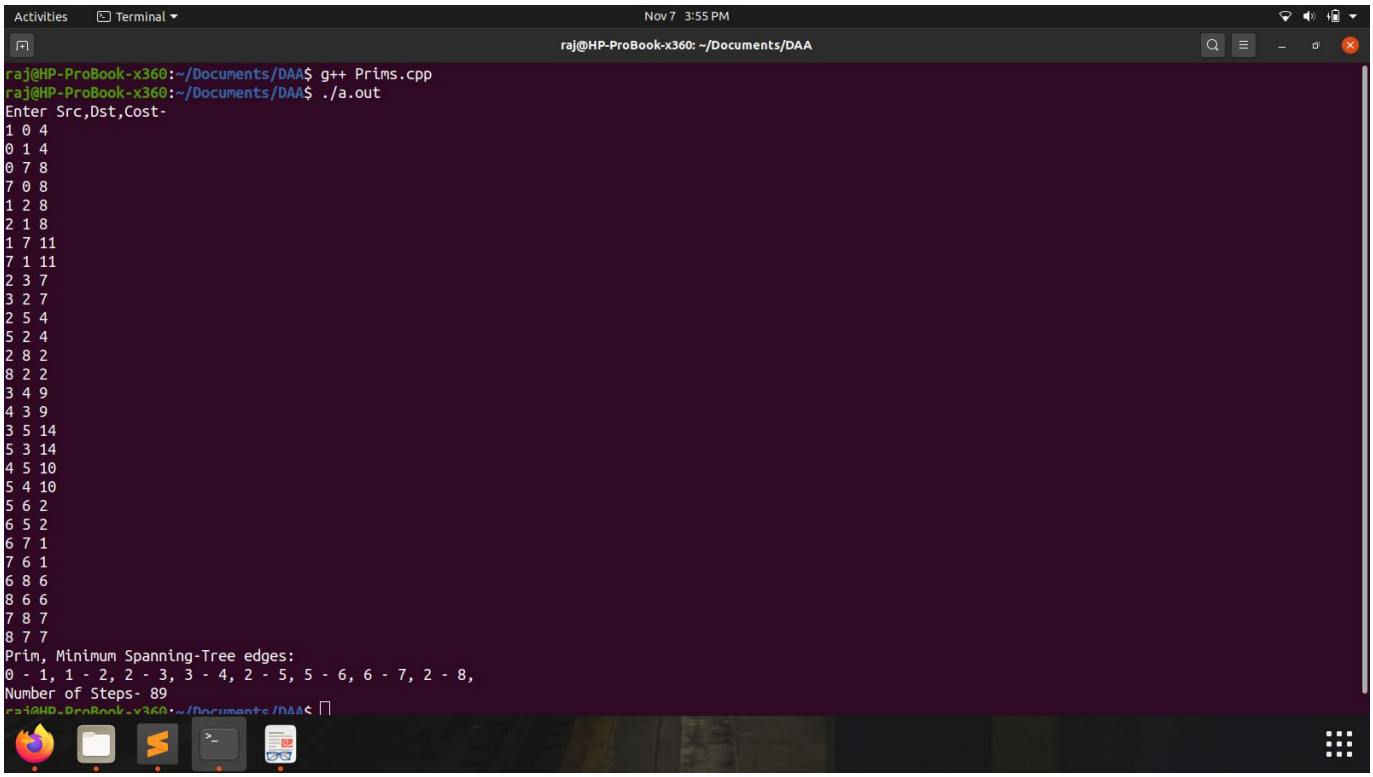
```
#include<bits/stdc++.h>
using namespace std;
class  vertice
{
public:int v,count=0;
    list< pair<int,int> >*adj;
     vertice(int ver)
     {
         v=ver;
         adj = new list< pair<int,int> > [v];
     }
    void add(int s,int d,int c)
    {
          adj[s].push_back(make_pair(d,c));
          adj[d].push_back(make_pair(s,c));
    }
    void prim(int src)
    {
          priority_queue< pair<int,int> , vector< pair<int,int> >,greater<
pair<int,int> > >q;
          int key[v],parent[v],include[v];
          memset(include,0,sizeof(include));
          for(int i=0;i<v;i++)
          {
                ++count;
                key[i]=INT_MAX;
                parent[i]=-1;
          }
          int mincost=0;
          q.push(make_pair(0,src));
```

```
    key[src]=0;
    int j=0;
    while(!q.empty())
    {
        ++count;
        if(j==v)
            break;
        pair<int ,int> p;
        p=q.top();
        q.pop();
        include[p.second]=1;
        j++;
        mincost+=p.first;
        list< pair<int,int> >::iterator i;
        for(i=adj[p.second].begin();i!=adj[p.second].end();i++)
        {
            ++count;
            int v=(*i).first;
            int c=(*i).second;
            if(include[v]==0 && key[v] > c)
            {
                ++count;
                key[v]=c;
                q.push(make_pair(key[v],v));
                parent[v]=p.second;
            }
        }
    }
    cout<<"Prim, Minimum Spanning-Tree edges: \n";
    for(int i=1;i<v;i++)
        cout<<parent[i]<<" - "<<i<<", ";
    cout<<endl;
}
```

```
};
int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(0);
    int v=9,e=28,src=0;
    vertice g(v);
    cout<<"Enter Src,Dst,Cost- \n";
    for(int i=0;i<e;i++)
    {
        int s,d,c;
        cin>>s>>d>>c;
        g.add(s,d,c);
    }
    g.prim(src);
    cout<<"Number of Steps- "<<g.count<<endl;
}
```

Output-

```
Prim, Minimum Spanning-Tree edges:
0 - 1, 1 - 2, 2 - 3, 3 - 4, 2 - 5, 5 - 6, 6 - 7, 2 - 8,
Number of Steps- 89
```


Activities
Terminal
Nov 7 3:55 PM
raj@HP-ProBook-x360: ~/Documents/DAA
raj@HP-ProBook-x360:~/Documents/DAA$ g++ Prims.cpp
raj@HP-ProBook-x360:~/Documents/DAA$ ./a.out
Enter Src,Dst,Cost-
1 0 4
0 1 4
0 7 8
7 0 8
1 2 8
2 1 8
1 7 11
7 1 11
2 3 7
3 2 7
2 5 4
5 2 4
2 8 2
8 2 2
3 4 9
4 3 9
3 5 14
5 3 14
4 5 10
5 4 10
5 6 2
6 5 2
6 7 1
7 6 1
6 8 6
8 6 6
7 8 7
8 7 7
Prim, Minimum Spanning-Tree edges:
0 - 1, 1 - 2, 2 - 3, 3 - 4, 2 - 5, 5 - 6, 6 - 7, 2 - 8,
Number of Steps- 89

Code of Kruskal Algo-

```
#include<bits/stdc++.h>
using namespace std;
typedef  pair<int, int> iPair;
struct Graph
{
        int V, E,count=0;
        vector< pair<int, iPair> > edges;
        Graph(int V, int E)
        {
        this->V = V;
        this->E = E;
        }
        void addEdge(int u, int v, int w)
        {
        edges.push_back({w, {u, v}});
        }
        int kruskalMST();
};
struct DisjointSets
{
        int *parent, *rnk,count2=0;
        int n;
        DisjointSets(int n)
        {
        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];
        for (int i = 0; i <= n; i++)
        {
        rnk[i] = 0;
        parent[i] = i;
        }
```

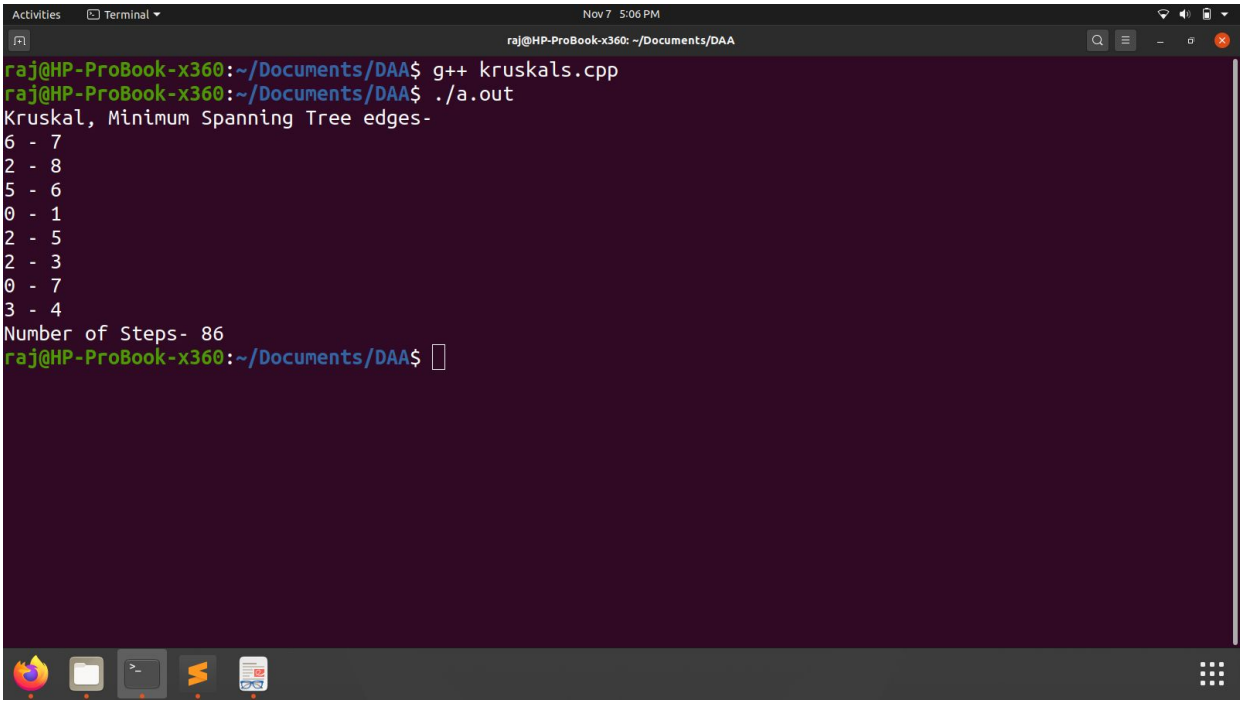
```
        }
        int find(int u)
        {
        if (u != parent[u])
        parent[u] = find(parent[u]);
        return parent[u];
        }

        void merge(int x, int y)
        {
        x = find(x), y = find(y);
        if (rnk[x] > rnk[y])
        parent[y] = x;
        else // If rnk[x] <= rnk[y]
        parent[x] = y;

        if (rnk[x] == rnk[y])
        rnk[y]++;
        }
};
int Graph::kruskalMST()
{
        int mst_wt = 0;
        sort(edges.begin(), edges.end());
        DisjointSets ds(V);
        vector< pair<int, iPair> >::iterator it;
        for (it=edges.begin(); it!=edges.end(); it++)
        {
         ++count;
        int u = it->second.first;
        int v = it->second.second;

        int set_u = ds.find(u);
```

```
        int set_v = ds.find(v);
        if (set_u != set_v)
        {
                ++count;
        cout << u << " - " << v << endl;
        mst_wt += it->first;
        ds.merge(set_u, set_v);
        }
        }

        return mst_wt;
}
int main()
{

        int V = 9, E = 28;
        Graph g(V, E);
        DisjointSets h();
        g.addEdge(0, 1, 4);
        g.addEdge(1, 0, 4);
        g.addEdge(0, 7, 8);
        g.addEdge(7, 0, 8);
        g.addEdge(1, 2, 8);
        g.addEdge(2, 1, 8);
        g.addEdge(1, 7, 11);
        g.addEdge(7, 1, 11);
        g.addEdge(2, 3, 7);
        g.addEdge(3, 2, 7);
        g.addEdge(2, 8, 2);
        g.addEdge(8, 2, 2);
        g.addEdge(2, 5, 4);
        g.addEdge(5, 2, 4);
        g.addEdge(3, 4, 9);
```

```
        g.addEdge(4, 3, 9);
        g.addEdge(3, 5, 14);
        g.addEdge(5, 3, 14);
        g.addEdge(4, 5, 10);
        g.addEdge(5, 4, 10);
        g.addEdge(5, 6, 2);
        g.addEdge(6, 5, 2);
        g.addEdge(6, 7, 1);
        g.addEdge(7, 6, 1);
        g.addEdge(6, 8, 6);
        g.addEdge(8, 6, 6);
        g.addEdge(7, 8, 7);
        g.addEdge(8, 7, 7);

        cout << "Kruskal, Minimum Spanning Tree edges- \n";
        int mst_wt = g.kruskalMST();
        cout<<"Number of Steps- "<<g.count<<endl;
        return 0;
}
```

Output-

```
Kruskal, Minimum Spanning Tree edges-
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4
Number of Steps- 86
```



Time Complexity of Both Prim's and Kruskal algorithm is O(ElogV), where

E = No of Edges and V = No of Vertices

So here

E = 28 & V = 9

Value of O(ElogV) = 28 * log(9)

= 28 * 3.16992500144

= 88.48 ≈ 89

Prim's Algorithm Count = 89

Kruskal's Algorithm Count = 86 ≈ 89 (Close)