Program1: Implement and analyze quicksort algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int partition(int a[], int low, int high)
{
   int pivot=a[low], i=low, j=high+1;
   int temp;
   while(i<j)
   {
        do
        {
          i++;
        }while(pivot>=a[i] && i<high);
        do
        {
          j--;
        }while(pivot<a[j]);
            if(i<j)
            {
              temp=a[i];
              a[i]=a[j];
              a[j]=temp;
            }
    }
    a[low]=a[j];
    a[j]=pivot;
    return j;
  }
  void quick_sort(int a[], int low, int high)
  {
    int s;
    if(low<high)
    {
          s=partition(a,low,high);
```

```
            quick_sort(a,low,s-1);
            quick_sort(a,s+1,high);
        }
    }
    int main()
    {
        int a[10000],n,low,high,i;
        clock_t st, end;
        printf("Enter number of elements\n");
        scanf("%d",&n);
        printf("Random numbers generated are\n");
        for(i=0;i<n;i++)
        {
            a[i]=rand()%100;
            printf("%d\t",a[i]);
        }
        low=0;
        high=n-1;
    st=clock();
    quick_sort(a,low,high);
    end=clock();
    printf("\nSorted array\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
    printf("\nTime required to sort given elements is %f",(float)(end-st)/CLOCKS_PER_SEC);
}
```

OUTPUT

```
Enter number of elements
5
Random numbers generated are
83  86  77  15  93
Sorted array
15  77  83  86  93
Time required to sort given elements is 0.000003
```

Program2:Implement and analyze mergesort algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void simple_merge(int a[],int low, int mid, int high)
{
   int i=low,j=mid+1,k=low,c[10000];
   while(i<=mid && j<=high)
   {
        if(a[i]<a[j])
        {
           c[k]=a[i];
           i++;
           k++;
        }
        else
        {
           c[k]=a[j];
           j++;
           k++;
        }
   }
   while(i<=mid)
        c[k++]=a[i++];
   while(j<=high)
        c[k++]=a[j++];
   for(i=low;i<=high;i++)
        a[i]=c[i];
}
void merge_sort(int a[],int low,int high)
{
   int mid;
   if(low<high)
   {
        mid=(low+high)/2;
        merge_sort(a,low,mid);
```

```
            merge_sort(a,mid+1,high);
            simple_merge(a,low,mid,high);
        }
    }
    int main()
    {
        int a[10000],i=0,n;
        clock_t st,end;
        printf("Enter the value of n\n");
        scanf("%d",&n);
        printf("Random numbers generated are\n");
        for(i=0;i<n;i++)
        {
            a[i]=rand()%100;
            printf("%d\t",a[i]);
        }
      st=clock();
      merge_sort(a,0,n-1);
      end=clock();
      printf("\nAfter Sorting\n");
      for(i=0;i<n;i++)
      printf("%d\t",a[i]);
      printf("\nTime required to sort given elements is %f",(float)(end-st)/CLOCKS_PER_SEC);
    }
```

OUTPUT

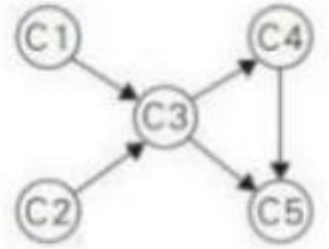
```
Enter the value of n
5
Random numbers generated are
83  86  77  15  93
After Sorting
15  77  83  86  93
Time required to sort given elements is 0.000023
```

Program3: Implement and analyze topological sorting in a given directed graph.

```
#include<stdio.h>
void ts(int a[20][20], int n)
{
 int t[10],vis[10],stack[10],i,j,indeg[10],top=0,ele,k=1;
 for(i=1;i<=n;i++)
 {
   t[i]=0;
   vis[i]=0;
   indeg[i]=0;
 }
 for(i=1;i<=n;i++)
 {
   for(j=1;j<=n;j++)
   {
      if(a[i][j]==1)
      {
         indeg[j]=indeg[j]+1;
      }
   }
 }
   printf("Indegree Array:");
   for(i=1;i<=n;i++)
     printf("%d ",indeg[i]);
   for(i=1;i<=n;i++)
   {
     if(indeg[i]==0)
     {
        stack[++top]=i;
        vis[i]=1;
      }
   }
   while(top>0)
   {
     ele=stack[top--];
     t[k++]=ele;
```

```
    for(j=1;j<=n;j++)
    {
       if(a[ele][j]==1 && vis[j]==0)
       {
          indeg[j]=indeg[j]-1;
          if(indeg[j]==0)
          {
             stack[++top]=j;
             vis[j]=1;
          }
       }
    }
  }
  printf("\nTopological Ordering is:");
  for(i=1;i<=n;i++)
     printf("%d",t[i]);
}
int main()
{
 int n,a[20][20],i,j;
 printf("Enter the number of nodes\n");
 scanf("%d",&n);
 printf("Enter Adjacency matric\n");
 for(i=1;i<=n;i++)
 {
   for(j=1;j<=n;j++)
   {
       scanf("%d",&a[i][j]);
   }
 }
 ts(a,n);
}
```

OUTPUT



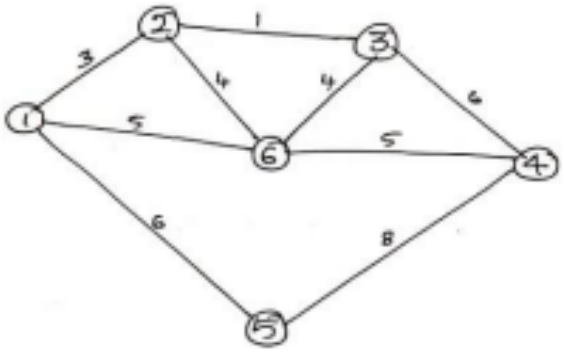

```
Enter the number of nodes
5
Enter Adjacency matric
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
Indegree Array:0 0 2 1 2
Topological Ordering is:21345
```

Program4:Implement and analyze Kruskal`s algorithm and find minimum cost spanning tree of a given connected undirected graph.

```
#include<stdio.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int i)
{
   while(parent[i])
   i=parent[i];
   return i;
}
int uni(int i,int j)
{
   if(i!=j)
   {
      parent[j]=i;
         return 1;
   }
   return 0;
}
int main()
{
   printf("Enter the no. of vertices:\n");
   scanf("%d",&n);
   printf("Enter the cost adjacency matrix:\n");
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         scanf("%d",&cost[i][j]);
         if(cost[i][j]==0)
            cost[i][j]=999;
      }
   }
   for(i=1;i<=n;i++)
   {
```

```
    parent[i]=0;
  }
  printf("The edges of Minimum Cost Spanning Tree are\n");
  while(ne < n)
  {
    for(i=1,min=999;i<=n;i++)
    {
      for(j=1;j<=n;j++)
      {
        if(cost[i][j]<min)
        {
          min=cost[i][j];
          a=u=i;
          b=v=j;
        }
      }
    }
    u=find(u);
    v=find(v);
    if(uni(u,v))
    {
      printf("%d edge (%d,%d) = %d\n",ne++,a,b,min);
      mincost +=min;
    }
    cost[a][b]=cost[b][a]=999;
  }
  printf("Minimum cost = %d\n",mincost);
}
```

OUTPUT




```
Enter the no. of vertices:
6
Enter the cost adjacency matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 0
5 4 4 5 0 0
The edges of Minimum Cost Spanning Tree are
1 edge (2,3) = 1
2 edge (1,2) = 3
3 edge (2,6) = 4
4 edge (4,6) = 5
5 edge (1,5) = 6
Minimum cost = 19
```

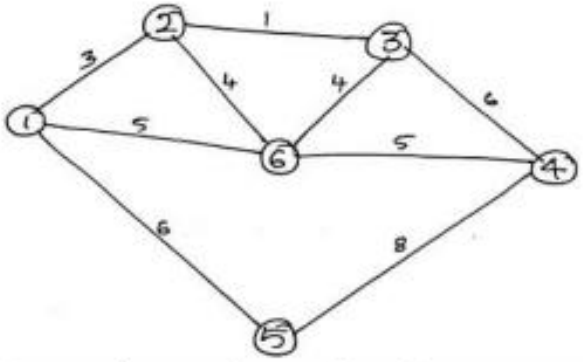
Program 5: Implement and analyze Prim`s algorithm and find minimum cost spanning tree of a given connected undirected graph.

```
#include<stdio.h>
int main()
{
   int n,a[20][20],i,j,min,mincost,u,v,ne,vis[20];
   printf("Enter the number of nodes\n");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
      vis[i]=0;
   printf("Enter the Cost matrix or Adjacency matrix\n");
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         scanf("%d",&a[i][j]);
         if(a[i][j]==0)
         {
            a[i][j]=999;
         }

   }
}
vis[1]=1;
ne=1;
mincost=0;
while(ne<n)
{
   for(i=1,min=999;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         if((a[i][j]<min) && (vis[i]!=0))
         {
            min=a[i][j];
            u=i;
            v=j;
```

```
            }
        }
    }
    if(vis[v]==0)
    {
        printf("Edge %d : (%d %d) cost %d\n", ne,u,v,a[u][v]);
        mincost+=a[u][v];
        ne+=1;
        vis[v]=1;
    }
    a[u][v]=a[v][u]=999;
  }
  printf("Minimum Cost = %d\n",mincost);
}
```

OUTPUT



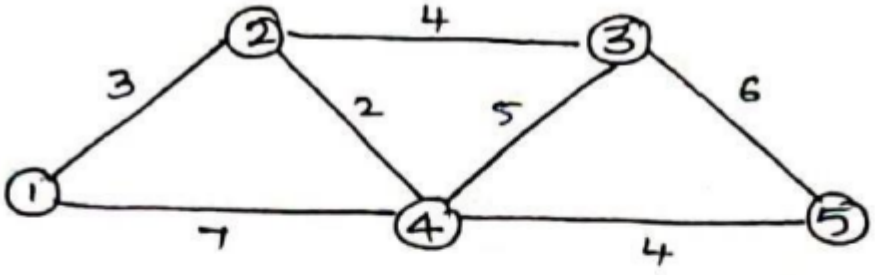

```
Enter the number of nodes
6
Enter the Cost matrix or Adjacency matrix
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 0
5 4 4 5 0 0
Edge 1 : (1 2) cost 3
Edge 2 : (2 3) cost 1
Edge 3 : (2 6) cost 4
Edge 4 : (6 4) cost 5
Edge 5 : (1 5) cost 6
Minimum Cost = 19
```

Program 6: Implement and analyze Dijkstra's algorithm to find the shortest path froma given source.

```
#include<stdio.h>
int main()
{
    int  n,a[20][20],i,j,min,u,v,s[10],d[10],k;
    printf("Enter the number of vertices\n");
    scanf("%d",&n);
    printf("Enter adjacency matrix\n");
    for(i=1;i<=n;i++)
    {
      for(j=1;j<=n;j++)
      {
         scanf("%d",&a[i][j]);
      }
    }
 printf("Enter source vertex\n");
 scanf("%d",&v);
 for(i=1;i<=n;i++)
{
   s[i]=0;
   d[i]=a[v][i];
}
d[v]=0;
s[v]=1;
for(k=2;k<=n;k++)
{
   min=999;
   for(i=1;i<=n;i++)
   {
      if(d[i]<min && s[i]==0)
      {
         min=d[i];
         u=i;
      }
   }
```

```
        s[u]=1;
        for(i=1;i<=n;i++)
        {
           if(s[i]==0)
           {
              if(d[i]>d[u]+a[u][i])
              {
                 d[i]=d[u]+a[u][i];
              }
           }
        }
     }
     for(i=1;i<=n;i++)
     {
        printf("%d --->%d=%d\n",v,i,d[i]);
     }
}
```

OUTPUT




```
Enter the number of vertices
5
Enter adjacency matrix
999 3   999 7   999
3   999 4   2   999
999 4   999 5   6
7   2   5   999 4
999 999 6   4   999
Enter source vertex
1
1---->1=0
1---->2=3
1---->3=7
1---->4=5
1---->5=9
```