2.Sort a given set of elements using the Heap sort method and determine the time
required to sort the elements. Repeat the experiment for different values of n,
the number of elements in the list to be sorted and plot a graph of the time taken
Versus n.

PROGRAM

```
#include<stdio.h>
#define max 100
void heapify () ;
void heapsort () ;
int maxdel ();
int a[max] , b[max] ,n ;
int main ()
{
      int i ,m;
      printf ( " \n Enter array size : " ) ;
      scanf ( "%d" , &n ) ;
      m = n ;
      printf ( " \n Enter elements : \n " ) ;
      for ( i = 1 ; i <= n ; i++ )
      scanf ( "%d" , &a[i] ) ;
      heapsort () ;
      printf ( " \n The sorted array is : \n " ) ;
      for ( i = 1 ; i <= m ; i++ )
      printf ( "\n%d" , b[i] ) ;
}
void heapsort ()
{
      int i ;
      heapify () ;
      for ( i = n ; i >= 1 ; i-- )
            b[i] = maxdel () ;
}
void heapify ()
{
      int i , e , j ;
      for ( i = n/2 ; i >= 1 ; i-- ) //start from middle of a and move upto 1
      {
            e = a[i] ; //save root of subtree
            j = 2 * i ;   //left child and c+1 is right child
                        while ( j <= n )
            {
                  if ( j < n && a[j] < a[j+1] )
                        j++ ; //pick larger of children
                  if ( e >= a[j] )
                        break;      // is a max heap
                  a[j/2] = a[j] ;
                  j = j * 2 ; //go to the next level
            }
            a[j/2] = e ;
      }
}
```

```
int maxdel ()
{
      int x , j , e , i ;
      if ( n == 0 )     return -1 ;
      x = a[1] ;  //save the maximum element
      e = a[n] ;  //get the last element
       n-- ;
      // Heap the structure again
      i = 1 ;
      j = 2;
      while ( j <= n )
      {
            if ( j < n && a[j] < a[j+1] )
                  j++ ; //Pick larger of two childrean
            if ( e >= a[j] )
                  break ;     //subtree is heap
            a[i] = a[j] ;
            i = j ;
            j = j * 2 ; // go to the next level
      }
      a[i] = e ;
      return x ;
}

==========Input - Output=================
 Enter array size : 5
 Enter elements :
 7
 1
 9
 3
 5
 The sorted array is :
 1
 3
 5
 7
 9
========================================
```

3.Sort a given set of elements using merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

PROGRAM

```
# include <stdio.h>
void Merge(int a[], int low, int mid, int high)
{
    int i, j, k, b[20];
    i=low;
    j=mid+1;
    k=low;
   while ( i<=mid && j<=high )
    {
      if( a[i] <= a[j] )
          b[k++] = a[i++] ;
      else
      b[k++] = a[j++] ;
```

```
   }
    while (i<=mid)
    b[k++] = a[i++] ;
    while (j<=high)
    b[k++] = a[j++] ;
    for(k=low; k<=high; k++)
    a[k] = b[k];
}
void MergeSort(int a[], int low, int high)
{
    int mid;
    if(low >= high)
    return;
    mid = (low+high)/2 ;
    MergeSort(a, low, mid);
    MergeSort(a, mid+1, high);
    Merge(a, low, mid, high);
}
int main()
{
   int n, a[20], k;
   printf("\n Enter how many numbers : ");
   scanf("%d", &n);
   printf("\n Enter %d numbers : \n ", n);
   for(k=1; k<=n; k++)
   scanf("%d", &a[k]);
   MergeSort(a, 1, n);
   printf("\n Sorted numbers are : \n ");
   for(k=1; k<=n; k++)
   printf("%5d", a[k]);
 }
```

```
================Input – Output===============
Enter how many numbers  : 5

Enter 5 numbers   :           99
                         67
                         85
                         12
                         97

Sorted numbers are       12
                         67
                         85
                         97
                         99
=============================================
```

4.Sort a given set of elements using Selection sort and determine the time required
to sort elements. Repeat the experiment for different values of n, the number of
elements in the list to be sorted and plot a graph of the time taken versus n.

PROGRAM

```
#include<stdio.h>
int a[20] , n ;
void selectionsort() ;
int main ()
{
    int i ;
    printf ( " \n Enter size of the array : " ) ;
    scanf ( "%d" , &n ) ;
    printf ( " \n Enter the elements : \n " ) ;
    for ( i = 0 ; i < n ; i++ )
    scanf ( "%d" , &a[i] ) ;
    selectionsort () ;
    printf ( " \n The sorted elements are : " ) ;
    for ( i = 0 ; i < n ; i++ )
      printf ( "\n%d" , a[i] ) ;
}
void selectionsort ()
{
      int i , j , min , temp ;
      for ( i = 0 ; i < n - 1 ; i++ )
      {
            min = i ;
            for ( j = i + 1 ; j < n ; j++ )
            {
                  if ( a[j] < a[min] )
                        min = j ;
            }
            temp = a[i] ;
            a[i] = a[min] ;
            a[min] = temp ;
            }
}
```

==========Input - Output==========

```
Enter size of the array : 5
Enter the elements :     7    1    9    3   5

 The sorted elements are :
 1
```

```
 3
 5
 7
 9
========================================
```

5.Implement 0/1 Knapsack problem using dynamic programming.

PROGRAM

```
#include<stdio.h>
int i,j,n,capacity,w[50],p[50],maxprofit;
int knapsack(int,int);
int maximum(int,int);
int main()
{
printf("\n Enter the no. of objects:");
scanf("%d",&n);
printf("Enter the weights:");
for(i=0;i<n;i++)
scanf("%d",&w[i]);
printf("Enter the profits associated:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter the Knapsack capacity:");
scanf("%d",&capacity);
maxprofit=knapsack(0,capacity);
printf("\n Maximum profit=%d",maxprofit);
}
int knapsack(int i,int c)
{
if(i==n)
return((c<w[n])?0:p[n]);
if(c<w[i])
return knapsack(i+1,c);
else
return maximum(knapsack(i+1,c),knapsack(i+1,c-w[i])+p[i]);
}
int maximum(int x,int y)
{
return((x>y)?x:y);
}
```

```
==============Input – Output================
RUN1
Enter the no. of objects:   3
Enter the weights :   1 2     2
Enter the profits  associated:  18  16    6
Enter the Knapsack capacity:  4
Optimal solution  = 34
========================================
```

6.From a given vertex in a weighted connected graph, find shortest paths to other
Vertices using Dijkstra’s algorithm

PROGRAM

```
#include<stdio.h>
void dijkstra(int,int,int [][],int []);
```

```
int main()
{
int n,v,cost[10][10],dist[10],i,j;
printf("Enter the no. of nodes:");
scanf("%d",&n);
printf("\nEnter 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]=9999;
 }
 printf("Enter the source vertex:");
 scanf("%d",&v);
dijkstra(n,v,cost,dist);
printf("\n Shortest Path from :\n");
for(j=1;j<=n;j++)
if(j!=v)
printf("\n %d--->%d = %d",v,j,dist[j]);
}
void dijkstra(int n,int v,int cost[10][10],int dist[])
{
int count,u,i,w,flag[10],min;
for(i=1;i<=n;i++)
{
flag[i]=0;
dist[i]=cost[v][i];
}
flag[v]=1;
dist[v]=1;
count=2;
while(count<=n)
{
min=9999;
for(w=1;w<=n;w++)
 if(dist[w]<min && !flag[w])
min = dist[w],u=w;
flag[u]=1;
count++;
for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w])&&!flag[w])
dist[w]=dist[u]+cost[u][w];
}
==========Input – Output=====================
      Enter the no. of nodes: 5
Enter the cost adjacency matrix:
0             3           9999        7      9999
3             0           4              2      100
9999       4           0              5      6
7             2           5              0      4
9999       9999     6             4      0
Enter the source vertex:1
Shortest Path from :
 1--->2 = 3
 1--->3 = 7
 1--->4 = 5
```

1--->5 = 9
==========

7.Sort a given set of elements using Quick sort method and determine the time
required sort the elements .Repeat the experiment for different values for
different values of n, the number of elements in the list to be sorted and plot a
graph of the time taken versus n.

PROGRAM

```
#include<stdio.h>
#include<time.h>
void quicksort(int [],int,int);
int partition(int [],int,int);
int main()
{
int i,n,a[20];
clock_t start,end;
printf("Enter the number of elements:");
scanf("%d",&n);
printf("Enter the elements");
for(i=0;i<n;i++)
{
  scanf("%d",&a[i]);
}
start=clock();
quicksort(a,0,n-1);
end=clock();
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n Time taken %f",(end-start)/CLK_TCK);
}
void quicksort(int a[],int low,int high)
{
int j;
if(low<high)
{
```

```
j=partition(a,low,high);
quicksort(a,low,j-1);
quicksort(a,j+1,high);
}
}
int partition(int a[],int low,int high)
{
int i,j,temp,key;
key=a[low];
i=low+1;
j=high;
while(1)
{
while(i<high &&key>=a[i])
i++
while(key<a[j])
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
else
{
temp=a[low];
a[low]=a[j];
a[j]=temp;
return j;
}
}
}
Enter size of the array : 5
Enter the elements :     7    1    9    3   5
 The sorted elements are :
 1  3  5  7  9
```

8.Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's Algorithm.

PROGRAM

```
#include<stdio.h>
int parent[10],min,no_of_edges=1,mincost=0,cost[10][10],a,b,i,j,u,v,n;
int main()
{
printf("\n Enter the no.of vertices:");
scanf("%d",&n);
printf("\n Enter the cost adjacency matrix:");
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;
  }
while(no_of_edges<n)
```

```
{
      for(i=1,min=999;i<=n;i++)
         for(j=1;j<=n;j++)
              if(cost[i][j]<min)
.......
```

2.Sort a given set of elements using the Heap sort method and determine the time
required to sort the elements. Repeat the experiment for different values of n,
the number of elements in the list to be sorted and plot a graph of the time taken
Versus n.

PROGRAM

```
#include<stdio.h>
#define max 100
void heapify () ;
void heapsort () ;
int maxdel ();
int a[max] , b[max] ,n ;
int main ()
{
      int i ,m;
      printf ( " \n Enter array size : " ) ;
      scanf ( "%d" , &n ) ;
      m = n ;
      printf ( " \n Enter elements : \n " ) ;
      for ( i = 1 ; i <= n ; i++ )
      scanf ( "%d" , &a[i] ) ;
      heapsort () ;
      printf ( " \n The sorted array is : \n " ) ;
      for ( i = 1 ; i <= m ; i++ )
      printf ( "\n%d" , b[i] ) ;
}
void heapsort ()
{
      int i ;
      heapify () ;
      for ( i = n ; i >= 1 ; i-- )
            b[i] = maxdel () ;
}
void heapify ()
{
      int i , e , j ;
      for ( i = n/2 ; i >= 1 ; i-- ) //start from middle of a and move upto 1
      {
            e = a[i] ; //save root of subtree
            j = 2 * i ;   //left child and c+1 is right child
                        while ( j <= n )
            {
                  if ( j < n && a[j] < a[j+1] )
                        j++ ; //pick larger of children
                  if ( e >= a[j] )
                        break;      // is a max heap
```

```
                  a[j/2] = a[j] ;
                  j = j * 2 ; //go to the next level
            }
            a[j/2] = e ;
      }
}
int maxdel ()
{     int x , j , e , i ;
      if ( n == 0 )     return -1 ;
      x = a[1] ;  //save the maximum element
      e = a[n] ;  //get the last element
       n-- ;
      // Heap the structure again
      i = 1 ;
      j = 2;
      while ( j <= n )
      {
            if ( j < n && a[j] < a[j+1] )
                  j++ ; //Pick larger of two childrean
            if ( e >= a[j] )
                  break ;     //subtree is heap
            a[i] = a[j] ;
            i = j ;
            j = j * 2 ; // go to the next level
      }
      a[i] = e ;
      return x ;
}
```

```
===========Input - Output==================
 Enter array size : 5
 Enter elements :
 7
 1
 9
 3
 5
 The sorted array is :
 1
 3
 5
 7
 9
=========================================
```

3.Sort a given set of elements using merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

PROGRAM

```
# include <stdio.h>
void Merge(int a[], int low, int mid, int high)
{
    int i, j, k, b[20];
    i=low;
    j=mid+1;
    k=low;
   while ( i<=mid && j<=high )
```

```
    {
      if( a[i] <= a[j] )
          b[k++] = a[i++] ;
      else
      b[k++] = a[j++] ;
   }
    while (i<=mid)      b[k++] = a[i++] ;
    while (j<=high)
    b[k++] = a[j++] ;
    for(k=low; k<=high; k++)
    a[k] = b[k];
}
void MergeSort(int a[], int low, int high)
{
    int mid;
    if(low >= high)
    return;
    mid = (low+high)/2 ;
    MergeSort(a, low, mid);
    MergeSort(a, mid+1, high);
    Merge(a, low, mid, high);
}
int main()
{
   int n, a[20], k;
   printf("\n Enter how many numbers : ");
   scanf("%d", &n);
   printf("\n Enter %d numbers : \n ", n);
   for(k=1; k<=n; k++)
   scanf("%d", &a[k]);
   MergeSort(a, 1, n);
   printf("\n Sorted numbers are : \n ");
   for(k=1; k<=n; k++)
   printf("%5d", a[k]);
 }
```

```
================Input – Output===============
Enter how many numbers  : 5

Enter 5 numbers   :           99
                          67
                          85
                          12
                          97

Sorted numbers are        12
                          67
                          85
                          97
                          99
=============================================
```

4.Sort a given set of elements using Selection sort and determine the time required
to sort elements. Repeat the experiment for different values of n, the number of
elements in the list to be sorted and plot a graph of the time taken versus n.

PROGRAM

```
#include<stdio.h>
int a[20] , n ;
void selectionsort() ;
int main ()
{
    int i ;
    printf ( " \n Enter size of the array : " ) ;
    scanf ( "%d" , &n ) ;
    printf ( " \n Enter the elements : \n " ) ;
    for ( i = 0 ; i < n ; i++ )
    scanf ( "%d" , &a[i] ) ;
    selectionsort () ;
    printf ( " \n The sorted elements are : " ) ;
    for ( i = 0 ; i < n ; i++ )
      printf ( "\n%d" , a[i] ) ;
}
void selectionsort ()
{
      int i , j , min , temp ;
      for ( i = 0 ; i < n - 1 ; i++ )
      {
            min = i ;
            for ( j = i + 1 ; j < n ; j++ )
            {
                  if ( a[j] < a[min] )
                        min = j ;
            }
            temp = a[i] ;
            a[i] = a[min] ;
            a[min] = temp ;
            }
}
==============Input - Output=============
Enter size of the array : 5
Enter the elements :     7    1    9    3   5
```

```
 The sorted elements are :
 1
 3
 5 7
 9
=======================================
```

5.Implement 0/1 Knapsack problem using dynamic programming.

PROGRAM

```
#include<stdio.h>
int i,j,n,capacity,w[50],p[50],maxprofit;
int knapsack(int,int);
int maximum(int,int);
int main()
{
printf("\n Enter the no. of objects:");
scanf("%d",&n);
printf("Enter the weights:");
for(i=0;i<n;i++)
scanf("%d",&w[i]);
printf("Enter the profits associated:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter the Knapsack capacity:");
scanf("%d",&capacity);
maxprofit=knapsack(0,capacity);
printf("\n Maximum profit=%d",maxprofit);
}
int knapsack(int i,int c)
{
if(i==n)
return((c<w[n])?0:p[n]);
if(c<w[i])
return knapsack(i+1,c);
else
return maximum(knapsack(i+1,c),knapsack(i+1,c-w[i])+p[i]);
}
int maximum(int x,int y)
{
return((x>y)?x:y);
}
```

```
==============Input – Output================
RUN1
Enter the no. of objects:   3
Enter the weights :   1 2     2
Enter the profits  associated:  18  16    6
Enter the Knapsack capacity:  4
Optimal solution  = 34
=======================================
```

6.From a given vertex in a weighted connected graph, find shortest paths to other
Vertices using Dijkstra’s algorithm

PROGRAM

```
#include<stdio.h>
void dijkstra(int,int,int [][],int []);
int main()
{int n,v,cost[10][10],dist[10],i,j;
printf("Enter the no. of nodes:");
scanf("%d",&n);
printf("\nEnter 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]=9999;
 }
 printf("Enter the source vertex:");
 scanf("%d",&v);
dijkstra(n,v,cost,dist);
printf("\n Shortest Path from :\n");
for(j=1;j<=n;j++)
if(j!=v)
printf("\n %d--->%d = %d",v,j,dist[j]);
}
void dijkstra(int n,int v,int cost[10][10],int dist[])
{
int count,u,i,w,flag[10],min;
for(i=1;i<=n;i++)
{
flag[i]=0;
dist[i]=cost[v][i];
}
flag[v]=1;
dist[v]=1;
count=2;
while(count<=n)
{
min=9999;
for(w=1;w<=n;w++)
 if(dist[w]<min && !flag[w])
min = dist[w],u=w;
flag[u]=1;
count++;
for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w])&&!flag[w])
dist[w]=dist[u]+cost[u][w];
}
```

```
===========Input – Output=====================
      Enter the no. of nodes: 5
Enter the cost adjacency matrix:
0             3           9999        7      9999
3             0           4              2      100
9999       4           0              5      6
7             2           5              0      4
9999       9999     6             4      0
Enter the source vertex:1
Shortest Path from :
 1--->2 = 3
 1--->3 = 7
```

```
 1--->4 = 5
 1--->5 = 9
==========================================
```

7.Sort a given set of elements using Quick sort method and determine the time
required sort the elements .Repeat the experiment for different values for
different values of n, the number of elements in the list to be sorted and plot a
graph of the time taken versus n.

PROGRAM

```
#include<stdio.h>
#include<time.h>
void quicksort(int [],int,int);
int partition(int [],int,int);
int main()
{
int i,n,a[20];
clock_t start,end;
printf("Enter the number of elements:");
scanf("%d",&n);
printf("Enter the elements");
for(i=0;i<n;i++)
{
  scanf("%d",&a[i]);
}
start=clock();
quicksort(a,0,n-1);
end=clock();
printf("\n The sorted array is:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n Time taken %f",(end-start)/CLK_TCK);
}
void quicksort(int a[],int low,int high)
{
int j;
if(low<high)
{
```

```
j=partition(a,low,high);
quicksort(a,low,j-1);
quicksort(a,j+1,high);
}}
int partition(int a[],int low,int high)
{
int i,j,temp,key;
key=a[low];
i=low+1;
j=high;
while(1)
{
while(i<high &&key>=a[i])
i++
while(key<a[j])
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
else
{
temp=a[low];
a[low]=a[j];
a[j]=temp;
return j;
}
}
}
Enter size of the array : 5
Enter the elements :     7    1    9    3   5
 The sorted elements are :
 1  3  5  7  9
```

8.Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's Algorithm.

PROGRAM

```
#include<stdio.h>
int parent[10],min,no_of_edges=1,mincost=0,cost[10][10],a,b,i,j,u,v,n;
int main()
{
printf(“\n Enter the no.of vertices:”);
scanf(“%d”,&n);
printf(“\n Enter the cost adjacency matrix:”);
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;
  }
while(no_of_edges<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;
              }
while(parent[u])
u=parent[u];
while(parent[v])
v=parent[v];
if(u!=v)
{
no_of_edges++;
printf("\n %d \t Edge \t(%d,%d)=%d",no_of_edges,a,b,min);
mincost+=min;
parent[v]=u;
}
cost[a][b]=cost[b][a]=999;
}
Printf("\n\t Minimum Cost is  = %d",mincost);
}
```

```
===========Input – Output=====================
      Enter the no. of vertices: 5
Enter the cost adjacency matrix:
0             11          9          7         8
11            0           15        14      13
9             15          0          12      14
7             14          12         0       6
8             13          14         6       0

2Edge(4,5)=6
3Edge(1,4)=7
4Edge(1,3)=9
5Edge(1,2)=11

MinimumCost=33
============================================
```

9.a) Print all the nodes from a given starting node in a digraph using breadth first
search method.

b) Check wheather a given graph is connected or not using DFS method.

PROGRAM(Print All the nodes of a digraph using BFS)

```
#include<stdio.h>
int visited[10];
void bfs(int n,int a[10][10],int source)
.............
```



```
{
 int i,q[10],u,t[10][10],k=0;
 int front=0,rear=0;
 visited[source]=1;
 q[rear]=source;
 while(front<=rear)
 {
    u=q[front];
    front=front+1;
    for(i=1;i<n;i++)
    {
      if(a[u][i]==1&&visited[i]==0)
      {
      rear=rear+1;
      q[rear]=i;
      visited[i]=1;
      t[k,0] = u;
      t[k,1] = I
      k++;
       }
     }
  }
}
main()
{
  int n,a[10][10],i,j,source;
              printf("\n ENTER THE NUMBER OF NODES:");
  scanf("%d",&n);
  printf("\n ENTER THE ADJACENCY MATRIX :\n");
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    scanf("%d",&a[i][j]);
     printf("\n ENTER THE SOURCE \n");
      scanf("%d",&source);
  for(i=1;i<=n;i++)
  visited[i]=0;
  bfs(n,a,source);
  for(i=1;i<=n;i++)
  {
     if(visited[i]==0)
     printf("\n THE NODE %d IS NOT REACHABLE ",i);
     else
     printf("\n THE NODE %d IS REACHABLE ",i);
  }
  for(i=0;i<n;i++)
```

```
  for(j=0;j<n;j++)
  printf(“The traversal is %d? %d \n” , t[i][j]);
}
==========Input – Output=====================

Enter the number of vertices: 3
      Enter the cost matrix:
            0 1 1 1 0
0 0 0 0 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

Enter the starting vertex:  1
The nodes reachable from 1 are :   2  3 4 5
The BFS traversal is :
1 2
1 3
1 4
2 5
=========================================
PROGRAM (Graph Connected or not using DFS)

#include<stdio.h>
void Dfs(int);
int adj[10][10];
int Reach[10],n;
void main()
{
 int i,j;
 printf("\n\n\tEnter how many vertices: ");
 scanf("%d",&n);
 printf("\n\n\tEnter 1 if Edge exist else Enter 0 :\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
      printf("\n\t\t\t\tadj[%d][%d]: ",i,j);
      scanf("%d",&adj[i][j]);
  }
 Dfs(1);
 for(i=1;i<=n;i++)
  if(Reach[i]!=1)
  {
      printf("\n\n\tThe Graph is Not Connected........" );
      break;
  }
 if(i>n) printf("\n\n\tThe graph is Connected.......");
}
void Dfs(int vertex)
{
 int i,j;
 Reach[vertex]=1;
 for(i=1;i<=n;i++)
  if((adj[vertex][i]==1||adj[i][vertex]==1)&&Reach[i]!=1)
  {
      Reach[i]=1;
      Dfs(i);
```

```
  }
}
```

10.Find the subset of a given set S={s1,s2,……sn.} of n positive integers whose sum is equal to a given positive integer d. For eg., if S= {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution.

PROGRAM

```
#include<stdio.h>
#define max 10
int s[max] , x[max] ;
int d ;
void sumofsub ( int , int , int ) ;
main ()
{
        int n , sum = 0 ;
        int i ;
        printf ( " \n enter the size of the set : " ) ;
        scanf ( "%d" , &n ) ;
        printf ( " \n Enter the set in increasing order : \n " ) ;
        for ( i = 1 ; i <= n ; i++ )
                scanf ( "%d", &s[i] ) ;
        printf ( " \n enter the value of d : \n " ) ;
                scanf ( "%d" ,  &d ) ;
        for ( i = 1 ; i <= n ; i++ )
                sum = sum + s[i] ;
        if ( sum < d || s[1] > d )
                printf ( " \n no subset possible : " ) ;
        else
                sumofsub ( 0 , 1 , sum ) ;
}
void sumofsub ( int m , int k , int r )
{
        int i ;
        x[k] = 1 ;
        if ( ( m + s[k] ) == d )
        {
                for ( i = 1 ; i <= k ; i++ )
                        if ( x[i] == 1 )
                                printf ( " \t %d " , s[i] ) ;
                printf ( " \n " ) ;
        }
        else
                if ( m + s[k] + s[k+1] <= d )
                sumofsub ( m + s[k] , k + 1 , r - s[k] ) ;
                if ( ( m + r - s[k] >= d ) && ( m + s[k+1] <=d ) )
                {
                        x[k] = 0;
                        sumofsub ( m , k + 1 , r - s[k] ) ;
                }
}
```

==========Input – Output==========

```
Enter number of elements: 7
Enter the elements in ascending order:
1     2     3     4     5     6     7
```

```
Enter the value: 7
Subset 1 = 1   2   4
Subset 2 = 1   6
Subset 2 = 2   5
Subset 2 = 3   4
Subset 2 = 7
=========================================
```

11 a) Implement Horspool algorithm for String Matching.
b) Find the Binomial coefficient using Dynamic programming.

```
PROGRAM(Horospoll String Matching)
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char Table[128];
char Text[25],Pattern[25];
int length;
int HorspoolMatching();
void CreatShiftTable();
void main()
{
 int pos;
 printf("\n\n\n\tEnte the Text String: ");
 scanf(“%s”,Text);
 printf("\n\n\tEnter the Pattern String: ");
scanf(“%s”,Pattern);
 pos=HorspoolMatching();
 if(pos!=-1)
  printf("\n\n\tThe Pattern Exist in the Text and Starts at Position:
%d",pos+1);
 else
  printf("\n\nPattern Doesnot exist in the Text...");
}
int HorspoolMatching()
{
 int i,k,n;
 CreatShiftTable();
 n=strlen(Text);
 i=length-1;
 while(i<=n-1)
 {
  k=0;
  while(k<=length-1&&Pattern[length-1-k]==Text[i-k])
      k=k+1;
  if(k==length)
   return (i-length+1);
  else
   i=i+Table[Text[i]];
 }
 return -1;
}

void CreatShiftTable()
{
 int i,j;
```

```
 length=strlen(Pattern);
 for(i=0;i<128;i++)
  Table[i]=length;
 for(j=0;j<=length-2;j++)
  Table[Pattern[j]]=length-1-j;
}

==========Input – Output=====================
Enter the Text string : Algorithms Lab

Enter the Pattern String : rit

The Pattern Exist in the Text and Starts at Position: 5
```

```
PROGRAM(Binomial Co-efficient Computation)
#include<stdio.h>
#include<math.h>
float BinomialCoeff(int,int);
int Min(int,int);
int C[10][10];
void main()
{
 int n,r;
 int nCr;
 printf("\n\n\tEnter values of n and r to find nCr...");
 printf("\n\n\tEnter the value of n: ");
 scanf("%d",&n);
 printf("\n\n\tEnter the value of r: ");
 scanf("%d",&r);
 nCr=BinomialCoeff(n,r);
 printf("\n\n\n\tThe value of %dC%d is: %d",n,r,nCr);
}
int Min(int a,int b)
{
 if(a>b) return b;
 else return a;
}
float BinomialCoeff(int n,int r)
{
 int i,j;
 for(i=0;i<=n;i++)
  for(j=0;j<=Min(i,r);j++)
  {
      if(j==0||j==i)
       C[i][j]=1;
      else
       C[i][j]=C[i-1][j-1]+C[i-1][j];
  }
 return C[n][r];  }
```

12. Find Minimum cost spanning tree of a given undirected graph using Prim’s
algorithm.

PROGRAM

```
#include<stdio.h>
int a,b,u,v,n,i,j,no_of_edges=1;
int visited[10],min,mincost=0,cost[10][10];
main()
{
printf("Enter the no.of vertices:");
scanf("%d",&n);
printf("Enter the adjacency matrix:");
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=2;i<=n;i++)
visited[1]=1;
printf("\n The Edges of Spanning Tree are:\n");
visited[1]=1;
while(no_of_edges<n)
{
 for(i=1,min=999;i<=n;i++)
 for(j=1;j<=n;j++)
 if(cost[i][j]<min)
 if(visited[i]==0)
 continue;
 else
 {
     min=cost[i][j];
      a=u=i;
      b=v=j;
    }
    if(visited[u]==0 || visited[v]==0)
    {
     no_of_edges++;
     printf("\n%d\tEdge \t(%d,%d)=%d",no_of_edges,a,b,min);
     mincost+=min;
    visited[b]=1;
    }
    cost[a][b]=cost[b][a]=99;
 }
 printf("\n\t Minimum Cost=%d\n",mincost);
}
```

==========Input – Output==========

```
Enter the no.of vertices: 4

Enter the cost adjacency matrix
 0    2     9     1
 2    5     1     11
 7    4     1     22
 11   4     7     2
```

```
The edges of spanning tree are:

1Edge(1,4)=1
2Edge(1,2)=2
3Edge(2,3)=1
Minimum Cost = 4
==========================================
```

13 a) Implement Floyd's Algorithm for the All-Pairs-Shortest paths Problem.

b) Compute the transitive closure of a given directed graph using Warshall's algorithm.

```
PROGRAM (Floyd's)
#include<stdio.h>
int n,a[20][20],i,j,k;
void allpairs(int [][20],int);
int min(int,int);
main()
{
printf("Enter the number of vertices");
scanf("%d",&n);
printf("Enter the cost 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;
}
allpairs(a,n);
for(i=1;i<=n;i++)
a[i][i]=0;
printf("The all pair shotest path matrix is\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
printf("%6d",a[i][j]);
}
void allpairs(int a[][20],int n)
{
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
a[i][j]=min(a[i][j],a[i][k]+a[k][j]);    }
int min(int a,int b)
{
return((a<b)?a:b);
}
=========Input - Output=================
Enter the no.of vertices:4
Enter the cost adjacency matrix:
0     10    0     40
```

```
0     0     0     20
50    0     0     0
0     0     60    0
The all pair shortest path matrix is
0     10    90    30
130   0     80    20
50    60    0     80
110   120   60    0
=======================================
PROGRAM (Warshall's)
#include<stdio.h>
void warshall(int [][10],int);
int main()
{
int c[10][10],i,j,n;
printf("Enter the no.of vertices:\n");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&c[i][j]);
warshall(c,n);
printf("the path matrix is:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%d\t",c[i][j]);
printf("\n");
}
}
void warshall(int p[10][10],int n)
{
int i,j,k;
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
if(p[i][k]==1)
for(j=1;j<=n;j++)
p[i][j]=p[i][j]||p[k][j]);
}
===========Input - Output==================
Enter the no.of vertices:4
Enter the adjacency matrix:
0     1     0     0
0     0     0     1
0     0     0     0
1     0     1     0
The Path Matrix is
1     1     1     1
1     1     1     1
0     0     0     0
1     1     1     1
=======================================
```

14.Implement N queen’s problem using Backtracking.

PROGRAM

```
#include <stdio.h>
#include <math.h>
int count=0,x[100];
main()
{
      int n;
      printf("\t\t\tN QUEEN'S PROBLEM\n");
      printf("\nEnter the number of queens:");
      scanf("%d",&n);
      nqueen(1,n);
      if(count==0)
            printf("\n There is no solution for '%d - Queens'
problem",n);
      else
            printf("\nTotal number of solutions :%d",count);
}
int  place(int k,int i)
{
      int j;
      for(j=1;j<k;j++)
      if((x[j]==i) ||(abs(x[j]-i)==abs(j-k)))   return(0);
      return(1);
}
nqueen(int k,int n)
{
      int i,j,p;
      for(i=1;i<=n;i++)
      if(place(k,i))
      {
            x[k]=i;
            if(k==n)
            {
                  count++;
                  for(j=1;j<=n;j++)
                  {
                        for(p=1;p<=n;p++)
                        if(x[j]==p) printf(" Q ");
                        else        printf(" X ");
                        printf("\n");
                  }
                  }
            else  nqueen(k+1,n);
      }
      printf("\n");
}
```

==========Input - Output==========

```
Enter the number of queens:4
The Solution to Queens Problem is:
```

```
 X  Q  X  X
 X  X  X  Q
 Q  X  X  X
 X  X  Q  X

The Solution to Queens Problem is:
 X  X  Q  X
 Q  X  X  X
 X  X  X  Q
 X  Q  X  X

Total No. of Solutions is : 2
========================================
```