1. Write a program to sort a given set of elements using Merge sort method and find the time required to sort the elements.

```
#include<stdio.h>
#define MAX 1000

int count;

void merge(int a[MAX], int low, int mid, int high)
{
      int i, j, k, b[MAX];

      i = low;
      j = mid+1;
      k = low;
      while( (i<=mid) && (j<=high))
      {
            if(a[i] < a[j])
             {
                  b[k++] = a[i++];
                  count++;
             }
            else
             {
                  b[k++] = a[j++];
                  count++;
              }
      }

      while(i <= mid)
         {
            b[k++] = a[i++];
            count++;
         }

      while(j <= high)
          {
             b[k++] = a[j++];
             count++;
            }

      for(i=low; i<=high; i++)
            a[i] = b[i];
}
void mergesort(int a[MAX], int low, int high)
{
      int mid;
      if(low < high)
      {
            mid = (low + high)/2;/*find the mid-point and divide the
array into two halves*/
            mergesort(a,low,mid);//call mergesort for first half
            mergesort(a,mid+1,high);//call mergesort for second half
            merge(a,low,mid,high);// merge the two halves sorted
```

```
        }
    }

    int main()
    {
         int i,j,n,a[MAX],b[MAX],c[MAX];
        int c1,c2,c3;
         printf("\n Enter n: ");
         scanf("%d",&n); //read the number of elements

         printf("\nEnter elements: ");
         for(i=0;i<n;i++)
             scanf("%d",&a[i]);// read the input elements to sort

         count=0;
         mergesort(a,0,n-1);

         printf("\n Sorted elements: \n");
         for(i=0;i<n;i++)
              printf("%d\n",a[i]);//display sorted elements
        printf("\n Number of counts : %d\n",count);
         printf("\n SIZE\t ASC\t DESC\t RAND\n");
        for(i=16; i<550;i=i*2)
          {
              for(j=0;j<i;j++)
               {
                    a[j]=j;
                    b[j]=i-j;
                    c[j]=rand() % i;
               }
           count=0;
           mergesort(a,0,i-1);
           c1=count;
           count=0;
           mergesort(b,0,i-1);
           c2=count;
           count=0;
           mergesort(c,0,i-1);
           c3=count;
           printf("\n %d\t%d\t%d\t%d",i,c1,c2,c3);
         }
        return 0;
}
```

Output

```
Enter n: 5
Enter elements:1 3 7 2 0
Sorted Elements:
0
1
2
3
7
```

For all 3 cases it should be

$n \log n - (n+1)$

2. Write a program to sort a given set of elements using Quick sort method and find the time required to sort the elements.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 1000
int count;

int partition(int a[MAX], int low, int high)
{
      int i, j, key, temp;

      i = low + 1;                  //Initialise lower index i
      j = high;                     //Initialise higher index j
      key = a[low];                 //Make first element as key

      while(1)
      {
            while ((key >= a[i]) && i < high)
               {
                  i++;
                  count++;
               }
            count++
            while(key < a[j])
               {
                  j--;
                  count++;
               }
            count++;
            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;
             }//end if
      }//end while
}//end function

void quicksort(int a[MAX],int low,int high)
{
      int j;
      if(low < high)//If there are more than one elements in the array
      {
            j = partition(a, low, high);
            quicksort(a, low, j-1);     //Sort left subarray
            quicksort(a, j+1, high);    //Sort right subarray
      }
}
```

```
int main()
{
      int n;                    //No. of elements
      int a[MAX],b[MAX],c[MAX];        //Array to store elements
      int i;                    //Index variable
      int c1,c2,c3;
      printf("\nEnter n: ");
      scanf("%d",&n);
      printf("\nEnter elements: \n");
      for(i=0;i<n;i++)
            scanf("%d",&a[i]);

      count=0;
      quicksort(a,0,n-1);

      printf("Sorted elements: \n");
      for(i=0;i<n;i++)
           printf("%d\n",a[i]);
      printf("\n Number of counts : %d\n",count);
      printf("\n SIZE\t ASC\t DESC\t RAND\n");
       for(i=16; i<550;i=i*2)
        {
            for(j=0;j<i;j++)
             {
                  a[j]=j;
                  b[j]=i-j;
                  c[j]=rand() % i;
             }
      count=0;
      quicksort(a,0,i-1);
      c1=count;
      count=0;
      quicksort(b,0,i-1);
      c2=count;
      count=0;
      quicksort(c,0,i-1);
      c3=count;
      printf("\n %d\t%d\t%d\t%d",i,c1,c2,c3);
    }
    return 0;
}
```

Output

```
Enter n: 6
Enter elements:14 5 7 89 2 34
Sorted Elements:
2
5
7
14
34
89
```

ASC & DESC should be $\frac{(n+1)(n+2)}{2} - 3$

$n = 2$ Count = 3

$n = 4$ Count = 12

Rand random it should be ✗ in b/w

3. Write a program to print all the nodes reachable from a given starting node in a graph using Depth First Search method and BFS method. Also check connectivity of the graph. If the graph is not connected, display the number of components in the graph.

```
#include <stdio.h>
#include<stdlib.h>                                  Visited

void dfs(int a[10][10], int n, int v[10], int source)
{
      int i;

      v[source] = 1; //add source to v(indicated source is visited)

      for(i=0; i<n; i++)
            if(a[source][i] == 1 && v[i] == 0)
                  dfs(a,n,v,i);//recursive call for dfs
 }


void bfs(int a[10][10], int n, int v[10], int source)
{
      int q[10], front=0, rear=-1;
      int node, i;

      v[source] = 1;   //add source to v(indicated source is visited)

      q[++rear] = source;
      while(front <= rear)//as long as queue is empty
      {
            node = q[front++];/*delete the next vertex to be explored
from q*/
            for(i=0;i<n;i++)
                  if(a[node][i] == 1 && v[i] == 0)
                  {
                        v[i] = 1;
                        q[++rear] = i;/*insert new vertex to q for
exploration*/
                  }
      }//end while
}//end bfs

int main()
{
      int n,ch;
      int a[10][10];
      int v[10];
      int source;
      int i, j;
      int count = 0;

      printf("Enter no of nodes: ");
      scanf("%d",&n);//read the total number of nodes
```

```
    printf("\n Read Adjacency matrix \n");
    for(i=0;i<n;i++)
          for(j=0;j<n;j++)
                scanf("%d",&a[i][j]);//read the adjacency matrix

    printf("Enter source: ");
    scanf("%d",&source);

    for(i=0;i<n;i++)
          v[i] = 0;
    printf("\n 1. DFS\n 2.BFS\n");
    printf("\n Read Choice :");
    scanf("%d",&ch);
    switch(ch)
   {
   case 1:
    for(i=0;i<n;i++)
      {
          if(v[i] == 0)
             {
               dfs(a,n,v,i);//call dfs method to check connectivity
               count++;
             }
      }
    printf("Result: ");
    if(count == 1)
       printf("Graph is Connected\n");
    else
       printf("Graph is NOT Connected with %d Components\n",count);
   break;
  case 2:
   for(i=0;i<n;i++)
      {
          if(v[i] == 0)
             {
               bfs(a,n,v,i);//call bfs method to check connectivity
               count++;
             }
      }
    printf("Result: ");
    if(count == 1)
       printf("Graph is Connected");
    else
       printf("Graph is NOT Connected with %d Components\n",count);

  break;
default:exit(0);
}
    return 0;
}
```

Output

Run 1:

```
Enter the number of nodes:4
```

```
Read Adjacency Matrix:
0 1 1 0
1 0 0 0
1 0 0 0
0 0 0 0
Enter Source:2
   1. DFS
   2. BFS
Read choice: 1
Result:
Graph is not connected with 2 components
```

Run 2:

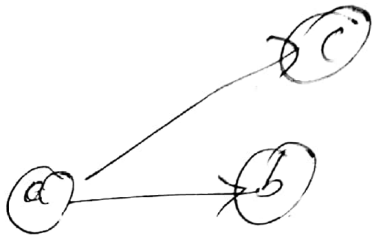
```
Enter the number of nodes:4
Read Adjacency Matrix:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Enter Source:4
   1.DFS
   2.BFS
Read choice:2
Result:
Graph is connected
```

4a. Write a program to obtain the Topological ordering of vertices in a given digraph using Vertices deletion method

```
#include<stdio.h>

int main()
{
      int n;
      int a[10][10];
      int i,j,k,node;
      int in[10]={0};
      int v[10]={0};

      printf("Enter n: ");
      scanf("%d",&n);// read the number of nodes

      printf("Enter Adj matrix: \n");
      for(i=1;i<=n;i++)
      {
            for(j=1;j<=n;j++)
            {
                  scanf("%d", &a[i][j]);

                  if(a[i][j] == 1)//check if vertex has indegree
                        in[j]++;
            }
      }

      printf("\nTopological order: ");
      for(k=1;k<=n;k++)
      {
         for(i=1;i<=n;i++)
            {
                  if(in[i] == 0 && v[i] == 0)
                  {
                        node = i;
                        printf("%5d",node);/*print the node in the
  topological order*/
                        v[node] = 1;
                        break;
                  }
            }

             for(i=1;i<=n;i++)
                  if(a[node][i] == 1)
                        in[i]--;
      }
      printf("\n\n");
}
```

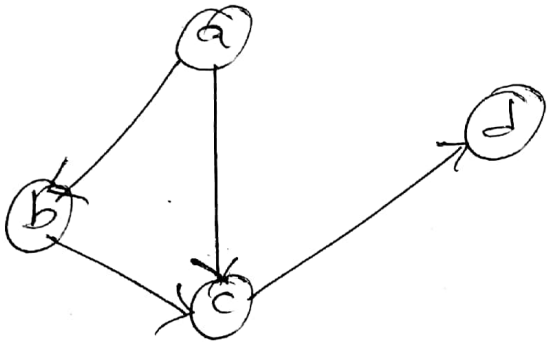
Ouput

```
Enter n:6
Enter Adjacency matrix:
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0

Topological order:1 4 0 2 3 5
```



a c b

a b c



a, 1, 4

b2, 3

c3, 2

d4, 1

d c b a

Ans a b c d

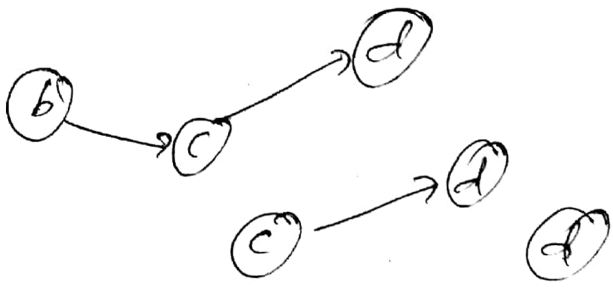
Vertex deleted

0 in(a) = 0

0 X in(b) = X 0

1 X in(c) = X X 0

in(d) = X 0 1



a b c d.

4b. Write a program to obtain the Topological ordering of vertices in a given digraph using DFS method

```
#include<stdio.h>
#include<stdlib.h>
int j=0;pop[10],v[10];

void dfs(int source,int n,int a[10][10])
 {
      int i,k,top=-1,stack[10];
      v[source]=1;
      stack[++top]= source+1;
      while(top!=-1)//check if stack is not empty
        {
            for(k=0;k<n;k++)
             {
                  if( a[source][k] == 1 && v[k] == 1 )
                   {
                    for(i=top; i>=0;i--)
                      if(stack[i] == k+1 )
                        {
                          printf("\n Topological order not possible");
                           exit(0);
                        }
                   }
                 else
                 {
                   if( a[source][k] == 1 && v[k] == 0)
                    {
                       v[k]=1;
                       stack[++top]= k+1;
                       source = k;
                       k=0;
                    }
                 }
             }
        pop[j++]=source+1;
        top --;
        source = stack[top] - 1;
       }
 }

void topo(int n , int a[10][10])
 {
      int i,k;
      for(i=0;i<n;i++)
           v[i]=0;
      for(k=0;k<n;k++)
           if(v[k]== 0)
                dfs(k,n,a);//dfs function call
 }
```

```
int main()
{
    int n,i,j,a[10][10];
    printf("\n Enter the no of Vertices : ");
    scanf("%d",&n);
    printf("\n Enter the Adjacency matrix\n");
    for(i=0;i<n;i++)
       for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    topo(n,a);
    printf("\n The topological ordering is\n");
    for(i=n-1;i>=0;i--)
            printf("%d\t",pop[i]);
 }
```

Output

```
Enter the number of vertices:8
Enter the adjacency matrix

0 1 1 0 1 0 0 0
0 0 1 0 0 1 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0

The topological ordering is
3 0 1 6 5 2 4 7
```

5. Write a program to sort n elements using heap sort.

```
#include<stdio.h>
#define MAX 1000
int count =0;

void heapcon(int a[MAX],int n)
{
  int i,k,v,flag,j;
  for(i=n/2; i>=1; i--)
  {
    k=i;
    v=a[k];
    flag = 0;
    while ( !flag && (2*k<=n) )
    {
      j=2*k;
      if(j<n)
       if(a[j] < a[j+1])
       {
          j=j+1;
       }
      count++;
      if(v>=a[j])
         flag = 1;
      else
      {
        a[k]=a[j];
        k=j;
      }
    }
    a[k]=v;
  }
}

void heapsort(int a[MAX], int n)
{
  int i,j,temp;
  for(i=n;i>=1;i--)
  {
    temp=a[1]; //exchange root with last leaf
    a[1]=a[i];
    a[i]=temp;
    count++;
    heapcon(a,i-1);//construct heap
  }
}

void main()
{
  int a[MAX],b[MAX],c[MAX];
  int n,i,j,c1,c2,c3;
  printf("\n enter the number of elements to be sorted : ");
  scanf("%d",&n);
```

```
printf("\n Enter the elements to be sorted\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
heapcon(a,n);
heapsort(a,n);
printf("\n Elements after sorting\n");
for(i=1;i<=n;i++)
printf("%d ",a[i]);
printf("\n Number of counts : %d\n",count);
printf("\n SIZE\t ASC\t DESC\t RAND\n");
for(i=16; i<550;i=i*2)
     {
          for(j=0;j<i;j++)
           {
                a[j]=j;
                b[j]=i-j;
                c[j]=rand() % i;
           }
         count=0;
         heapcon(a,i-1);
         heapsort(a,i-1);
         c1=count;
         count=0;
         heapcon(b,i-1);
         heapsort(b,i-1);
         c2=count;
         count=0;
         heapcon(c,i-1);
         heapsort(c,i-1);
         c3=count;
         printf("\n %d\t%d\t%d\t%d",i,c1,c2,c3);
     }
  return 0;
}
```

Output

```
Enter the number of elements to be sorted:7
Enter the elements to be sorted:
8 2 0 1 5 7 4
Elements after sorting:
0
1
2
4
5
7
8
```

6a. Write a program to implement Boyer Moore algorithm for String Matching.

```
# include <limits.h>
# include <string.h>
# include <stdio.h>

# define NO_OF_CHARS 256

int max (int a, int b)
{
   return (a > b)? a: b;
 }

void badCharHeuristic( char *str, int size, int badchar[NO_OF_CHARS])
{
    int i;

    for (i = 0; i < NO_OF_CHARS; i++)
         badchar[i] = -1; // Initialize all occurrences as -1

    for (i = 0; i < size; i++)
         badchar[(int) str[i]] = i;/* Fill the actual value of last
occurrence of a character */

}

void search( char *txt,  char *pat)
{
    int m = strlen(pat);//initialize length of pattern to m
    int n = strlen(txt);//initialize length of text to to n

    int badchar[NO_OF_CHARS];

    badCharHeuristic(pat, m, badchar);

    int s = 0;  // s is shift of the pattern with respect to text
    while(s <= (n - m))
    {
        int j = m-1;
           /* Keep reducing index j of pattern while
           characters of pattern and text are
           matching at this shift s */


        while(j >= 0 && pat[j] == txt[s+j])
            j--;
          /* If the pattern is present at current
           shift, then index j will become -1 after
           the above loop */
        if (j < 0)
        {
            printf("\n pattern occurs at shift = %d", s);
```

```
                /* Shift the pattern so that the next
                character in text aligns with the last
                occurrence of it in pattern.
                The condition s+m < n is necessary for
                the case when pattern occurs at the end
                of text */
            s += (s+m < n)? m-badchar[txt[s+m]] : 1;

        }

        else
             /* Shift the pattern so that the bad character
               in text aligns with the last occurrence of
               it in pattern. The max function is used to
               make sure that we get a positive shift.
               We may get a negative shift if the last
               occurrence of bad character in pattern
               is on the right side of the current
               character. */

            s += max(1, j - badchar[txt[s+j]]);
    }

int main()
{
    char txt[20];
    char pat[10];
    printf("\n Enter the Text : ");
    scanf("%s",txt);
    printf("\n Ener the pattern :");
    scanf("%s",pat);
    search(txt, pat);
    return 0;
}
```

Output

```
Enter the Text: ABAABCD

Enter the pattern: ABC


Pattern occurs at shift: 4
```

6b. Write a program to implement Floyd's algorithm for String Matching.

```
#include<stdio.h>

int min(int a,int b)//find minimum of given two values
{
      if(a < b)
        return a;

      else
        return b;
}

void floyd(int n,int d[10][10])
{
       int i,j,k;
       for(k=1;k<=n;k++)
       for(i=1;i<=n;i++)
       for(j=1;j<=n;j++)
          d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
//compute the shortest path for given all pairs of paths
}

int main()
{
      int n,a[10][10];
      int i,j,k;
      printf("Enter the no.of nodes: ");
      scanf("%d",&n);
      printf("\nEnter the adjacency matrix\n");
            for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
            {
                  scanf("%d",&a[i][j]); //read the cost adjacency matrix

            }

       floyd(n,a);//call for Floyd function to compute shortest path
       printf("\n\nThe distance matrix is \n");
       for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            printf("%5d",a[i][j]);//display all pairs shortest distance
            printf("\n");
        }
   return 0;
}
```

Output

```
Enter the number of nodes:4

Enter the adjacency matrix
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

Distance matrix is
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
```

7a. Write a program to implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

#define MAX 150

//Function declarations
int knap(int n,int m);
int big(int a,int b);

//Global variables
int w[MAX];                          //Array to store weights of each item
int p[MAX];                          //Array to store profits of each item
int v[MAX][MAX];       //Optimal solution of 'i' items with 'j' capacity

int big(int a,int b)//compute largest of given two numbers
{
     if (a > b) return a;
       else return b;
}

int knap(int n,int m)
{
     int i, j;
     for(i = 1; i <= n; i++)
     for(j = 1; j <= m; j++)
     {
          if( (j - w[i]) < 0)
               v[i][j] = v[i-1][j];
          else
               v[i][j] = big(v[i-1][j], p[i] + v[i-1][j-w[i]] );
     }

     return v[n][m];
}


int main()
{
     int i, j, profit, n, m;

     printf("\n Enter n (no. of items): ");
     scanf("%d",&n);

     printf("\n Enter the knapsack capacity:");
     scanf("%d",&m);

     printf("\n enter the weights and profits :\n");
     for(i=1;i<=n;i++)
       {
          printf("w[%d] = ",i);
          scanf("%d",&w[i]);
          printf("p[%d] = ",i);
```

```
            scanf("%d",&p[i]);
        }

    for(i=0; i<=n; i++)
            v[i][0]=0;

    for(j=0; j<=m; j++)
            v[0][j]=0;

    profit = knap(n,m);//call knap function to compute profit

    printf("\n goal = %d\n\n",profit);//display profit

    return 0;
}
```

Output

```
Enter n (no. of items): 4

Enter the knapsack capacity: 5

enter the weights and profits:

2 12
1 10
3 20
2 15

goal = 37
```

7b. Write a program to implement binomial coefficient using dynamic programming.

```
#include<stdio.h>

int min(int a, int b)
{
    return (a<b)? a: b;//return minimum of given two numbers
}
int binomialCoeff(int n, int k)
{
    int C[n+1][k+1];//to store binomial coefficients
    int i, j;//index variables such that 0<=i<=n and 0<=j<=k
     for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= min(i, k); j++)
        {
            if (j == 0 || j == i)
               C[i][j] = 1;//fill first column and diagonal items with
1

            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }

    return C[n][k];
}
int main()
{
    int n,k,ans;
    printf("\nEnter the value of n and k \n");
    scanf("%d%d",&n,&k);
    ans= binomialCoeff(n, k)// call to compute binomial coefficient
    printf ("Value of C(%d, %d) is %d ", n, k, ans);
    //display binomial coefficient
    return 0;
}
```

Output

```
Enter the value of n and k
6 3
Value of C(6,3) is 20
```

8. Write a program to find Minimum cost spanning tree of a given undirected graph using Prim's algorithm.

```
#include<stdio.h>

#define INFINITY 999

void prims(int n, int cost[10][10], int source);

int main()
{
      int n;                              //no. of nodes
      int cost[10][10];                   //Adjacency matrix of graph
      int source;                         //source node
      int i, j;                           //index variables

    printf("Enter n (no. of nodes): ");
    scanf("%d",&n);

    printf("Enter cost matrix:\n ");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d",&cost[i][j]);//read the cost adjacency matrix

    printf("Enter Source: ");
    scanf("%d",&source);

    prims(n,cost,source);//prims function call to compute the

    return 0;
}

void prims(int n,int cost[10][10],int source)
{
     int v[10];
     int d[10];
     int i, j;
     int vertex[10];
     int u, least;
     int    sum=0;//initial cost of spanning tree
     for(i=1;i<=n;i++)
     {
         v[i] = 0;
         d[i] = cost[source][i];
         vertex[i] = source;
     }

     v[source] = 1; // add source to v
     for(i=1;i<n;i++)
     {
         least = INFINITY;
         for(j=1; j<=n; j++)
         {
```

```
            if(v[j] == 0 && d[j] < least)
            {
                least = d[j];
                u = j;
            }
        }

        v[u] = 1;//add u to v
        sum += d[u];
        /* add the cost associated with edge to get total cost of
minimum spanning tree */
        printf("%d --> %d = %d Sum =  %d\n\n",vertex[u],u, d[u],sum);

        for(j=1;j<=n;j++)
        {
             if(v[j] == 0 && cost[u][j] < d[j])
             {
                     d[j] = cost[u][j];
                     vertex[j] = u;
             }
        }
     }
    printf("Total cost: %d",sum);//display total cost
}
```

Output

```
Enter n (no. of nodes): 5

Enter cost matrix:

0 3 999 999 6
3 0 1 999 999
999 1 0 6 999
999 999 6 0 8
6 999 999 8 0


Enter source: 2

2--->3 = 1 sum = 1
2--->1 = 3 sum = 4
3--->4 = 6 sum = 10
1--->5 = 6 sum = 16

Total cost: 16
```

9. Write a program to find the shortest path using Dijkstra's algorithm for a weighted connected graph.

```
#include <stdio.h>

#define INFINITY 999

void dijk(int cost[10][10], int n, int source, int v[10], int d[10]);

int main()
{
      int n;                                  //no. of nodes
      int cost[10][10];                       //Adjacency matrix of graph
      int source;                             //source node
      int v[10]; //visited array. keeps track to nodes visited
      int d[10]; //distance array.shortest distance from source node
      int i, j;                               //index variables

      //1. Read no. of nodes
      printf("Enter n: ");
      scanf("%d",&n);

      //2. Read cost adjacency matrix of graph
      printf("Enter  Cost matrix: \n");
      for(i=1; i<=n; i++)
            for(j=1; j<=n; j++)
                  scanf("%d",&cost[i][j]);

      //3. Read source
      printf("\n enter source:");
      scanf("%d",&source);

      //4. Initialise d[] to distance from source to each node
      //Initialise v[] to 0, indicating none of the nodes are visited
      for(i=1; i<=n; i++)
      {
            d[i] = cost[source][i];
            v[i] = 0;
      }

      //5. Call function to compute shortest distance
      dijk(cost, n, source, v, d);

      //6. Print Shortest distance from source to all other nodes
      printf("Shortest distance from source %d\n\n",source);
      for(i=1; i<=n; i++)
            printf("%d --> %d = %d\n\n",source,i,d[i]);

      return 0;
}

//Function to implement dijkstra algorithm
void dijk(int cost[10][10],int n,int source,int v[10],int d[10])
{
```

```
    int least, i, j, u;

    //A. Mark source node as visited
    v[source] = 1;

    //B. From each node find shortest distance to nodes not visited
    for(i=1; i<=n; i++)
    {
          //B1. Assume least as infinity
          least = INFINITY;

          //B2. Find u and d(u) such that d(u) is minimum i.e., Find
          //the next nearest node
          for(j=1; j<=n; j++)
          {
                if(v[j] == 0 && d[j] < least)
                {
                      least = d[j];
                      u = j;
                }
          }

          //B3. Mark u as visited (mark nearest node as visited)
          v[u] = 1;

          //B4. For remaining nodes, find shortest distance through u
          for(j=1; j<=n; j++)
          {
                if(v[j] == 0 && (d[j] > (d[u] + cost[u][j])) )
                      d[j] = d[u] + cost[u][j];

          }
     }//end for outer
}//end function
```

Output

```
Enter n: 5

Enter the cost matrix:
0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 4 6 0

Enter source: 2
```

```
Shortest distance from source 2
2-->1 =3
2-->2 =0
2-->3 =4
2-->4 =2
2-->5 =6
```

10. Write a program to implement n-queens problem.

```
#include <stdio.h>

//Function declarations
void nqueens(int n);
int can_place(int c[10],int r);
void display(int c[10],int r);

//Global variable
int count = 0;

int main()
{
    int n;

     //1. Read no. of queens
    printf("Enter n (no of queens): ");
    scanf("%d",&n);

    //2. Call function if solution exist
    if(n == 2 || n == 3)
       printf("Solution does not exist.");
    else
     {
      nqueens(n);
      printf("Total no. of solutions: %d\n",count);
     }

    return 0;
}

void nqueens(int n)
{
    int r;                      //Contains row no.
    int c[10];                  //Stores queens positions in each row
     int i;

     r = 0;                 //Select first queen (place queen in first row)
     c[r] = -1;             //Initial position of queen

     while(r >= 0)          //As long as there are solutions
     {
         c[r]++;                                //Place queen in r th coloumn

     //verify there is no attack from any of t previous queens placed
         while(c[r] < n && !can_place(c,r))
             c[r]++;

         if(c[r] < n)
         {
             if(r == n-1)       /if all n queens - display
             {
                 printf("Solution %d: ",++count);
```

```
                for(i=0;i<n;i++)
                    printf("%4d",c[i]+1);

                display(c,n);
            }
            else        //else place the next queen in next row
            {
                r++;
                c[r] = -1;
            }
        }
        else
            r--;         //backtracking (go to previous row)
    }
}

//Function to check attack on queen r from 0-(r-1) queens
//return 0: if there is attack, other wise return 1;

int can_place(int c[10],int r)
{
    int i;

    for(i=0; i<r; i++)
    {
        if( (c[i] == c[r]) || (abs(i-r) == abs(c[i] - c[r])) )
            return 0;
    }
    return 1;
}

//Function to create chessboard with queens placed and display
void display(int c[10],int n)
{
     char cb[10][10];
     int i, j;

     for(i=0;i<n;i++)
         for(j=0;j<n;j++)
             cb[i][j] = '-';

     for(i=0;i<n;i++)
         cb[i][c[i]] = 'Q';

     //Display the chess board
     printf("\n\nChessboard: \n");
     for(i=0;i<n;i++)
     {
         for(j=0;j<n;j++)
             printf("%4c",cb[i][j]);
         printf("\n\n");
     }
}
```

Output

Run1

Enter the total number of queens: 4

Total number of solutions: 2

Solution 1:

Chessboard

_Q_ _

_ _ _ Q

Q_ _ _

_ _ Q _

Solution 2

Chessboard

_ _ Q _

Q_ _ _

_ _ _ Q

_ Q _ _

Run2

Enter the number of queens: 3

Solution does not exist