1. **Sort a given set of elements using the Quicksort 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. The elements can be read from a file or can be generated using the random number generator.**

```
#include<stdio.h>
#include<sys/time.h>
void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
void generate_random(int a[],int n)
{   int i;
    srand(time(0));
    for(i=0;i<n;i++)
            a[i]=rand()%1000;
}
int Partition(int a[10],int l,int h)
{
    int i,j,p;
    i=l;j=h+1; p=l;
    do{
            do{
           i=i+1;
            }while(a[i]<a[p]);
            do{
                    j=j-1;
```

```
                    }while(a[j]>a[p]);
    swap(&a[i],&a[j]);
    }while(i<=j);
    swap(&a[i],&a[j]);
    swap(&a[l],&a[j]);
    return j;
}
int Quicksort(int a[10],int m,int n)
{
    int s;
if(m<n+1)
{
s=Partition(a,m,n);
Quicksort(a,m,s-1);
Quicksort(a,s+1,n);
return a;
}
}

int main()
{
    int a[100000],i,ch,n;
    struct timeval t;
    double start,end;
    FILE *fp;
    printf("Enter the type of operation\n");
    printf("1-Randomly generate numbers and quicksort\n");
    printf("2-Enter the number of values to generate and sort\n");
    scanf("%d",&ch);
    switch(ch)
    {case 1:
```

```
  fp=fopen("quicksort.txt","w");
  for(i=10000;i<100000;i=i+5000)
  {
        generate_random(a,i);
        gettimeofday(&t,NULL);
        start=t.tv_sec+(t.tv_usec/1000000.0);
        Quicksort(a,0,i-1);
        gettimeofday(&t,NULL);

        end=t.tv_sec+(t.tv_usec/1000000.0);
        printf("%d\t%lf\n",i,end-start);
        fprintf(fp,"%d\t%lf\n",i,end-start);
 }
 fclose(fp); break;
 case 2:printf("Enter the number of values to be generated\n");
  scanf("%d",&i);
  generate_random(a,i);
  gettimeofday(&t,NULL);
        start=t.tv_sec+(t.tv_usec/1000000.0);
        Quicksort(a,0,i-1);
        gettimeofday(&t,NULL);
        end=t.tv_sec+(t.tv_usec/1000000.0);
        printf("%d\t%lf\n",i,end-start);
        printf("Sorted numbers are\n");
        printf("The sorted array is\n");
        for(n=0;n<i;n++)
        printf("%d\t",a[n]);
        break;
default: printf("Invalid choice\n");
  }
  return 0;         }
```

2. **Using OpenMP, implement a parallelized Merge Sort algorithm to sort a given set of elements 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. The elements can be read from a file or can be generated using the random number generator.**

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>
void simplemerge(int a[],int low,int mid,int high)
{
   int i,j,k,c[10000];
   i=low;
   j=mid+1;
   k=low;
   int tid;
   omp_set_num_threads(5);
   {
      #pragma omp parallel
      tid=omp_get_thread_num();
      #pragma omp while
      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];
      i++;
      k++;
   }
   while(j<=high)
   {
      c[k]=a[j];
      j++;
      k++;
   }
   for(k=low;k<=high;k++)
   a[k]=c[k];
}
void merge(int a[],int low,int high)
{
   int mid;
   if(low<high)
   {
      mid=(low+high)/2;
      merge(a,low,mid);
      merge(a,mid+1,high);
      simplemerge(a,low,mid,high);
```

```
    }
}
void getnumber(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
    a[i]=rand()%10000;
}
int main()
{
    FILE *fp;
    int a[10000],i;
    struct timeval tv;
    double start,end,elapse;
    fp=fopen("m.txt","w");
    printf("Num.   time");
    for(i=0;i<9999;i+=100)
    {
        getnumber(a,i);
        gettimeofday(&tv,NULL);
        start=tv.tv_sec+(tv.tv_usec/100000.0);
        merge(a,0,i-1);
        gettimeofday(&tv,NULL);
        end=tv.tv_sec+(tv.tv_usec/100000.0);
        elapse=end-start;
      if(elapse>=0)
                fprintf(fp,"%d\t%lf\n",i,elapse);
     }
    fclose(fp);
   system("gnuplot");
    return 0;           }
```

3. i. Obtain the Topological ordering of vertices in a given digraph.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int array[20][20],v,i,j,count=0;
    int job[10],z,sum,indegree[20],flag;

    printf("Enter the number of vertices\n");
    scanf("%d",&v);

    printf("Enter the adjacency matrix\n");
    for(i=0;i<v;i++)
    {
        for(j=0;j<v;j++)
        {
            scanf("%d",&array[i][j]);
        }
    }
    while(count<v)
    {
        for(i=0;i<v;i++)
        {
            sum=0;
            if(indegree[i]!=-1)
            {
                for(j=0;j<v;j++)
                {
                    sum+=array[j][i];
                }
```

```
                        indegree[i]=sum;
                }
        }
  flag=0;
   for(i=0;i<v;i++)
   {
        if(indegree[i]==0)
        {
                z=i;
                flag=1;
                break;
        }
   }
   if(!flag)
   {
        printf("The above graph cannot be sorting using topology sort");
        exit(0);
   }
        for(j=0;j<v;j++)
                array[z][j]=0;
   indegree[z]=-1;
   job[count]=z+1;
   count++;
   }

   printf("The Topological order is \n");
   for(i=0;i<count;i++)
        printf("%d\t",job[i]);
   printf("\n");
   return 0;
}
```

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

```
#include<iostream>
using namespace std;
int a[10][10],n;
void warshall()
{
	int i,j,k;
	for(k=1;k<=n;k++)
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	if(a[i][j]!=1)
	{
		if(a[i][k]==1 && a[k][j]==1)
		a[i][j]=1;
	}
}
int main()
{
	int i,j;
	cout<<"Enter no of nodes : ";
	cin>>n;
	cout<<"Enter adjacency matrix : \n";
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
		cin>>a[i][j];
	warshall();
	cout<<"Transitive closure :\n";
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
```

```
            cout<<a[i][j]<<"\t";
            cout<<endl;
        }
        return 0;
}
```

4. Implement 0/1 Knapsack problem using Dynamic Programming.

```
#include <stdio.h>
#include <stdlib.h>
int max(int a,int b)
{
	if(a>b)
		return a;
	else
		return b;
}
void Knapsack(int table[10][10],int weight[10],int value[10],int items,int W)
{
	int i,j,count=0,Knapsack[items];
	for(i=0;i<=items;i++)
		table[i][0]=0;
	for(j=0;j<=W;j++)
		table[0][j]=0;
	for(i=1;i<=items;i++)
	{
		for(j=1;j<=W;j++)
		{
			if(j<weight[i])
				table[i][j]=table[i-1][j];
			else
				table[i][j]=max(table[i-1][j],(table[i-1][j-weight[i]]+value[i]));
		}
	}
	printf("Profit Table\n");
	for(i=0;i<=items;i++)
	{
		for(j=0;j<=W;j++)
```

```
                {
                        printf("%d\t",table[i][j]);
                }
                printf("\n");
        }
        i=items;
        j=W;
        while(i>0 && j>0)
        {
                if(table[i][j]!=table[i-1][j])
                {
                        Knapsack[count++]=i;
                        j=j-weight[i];
                }
                i=i-1;
        }
        if(!count)
                printf("The Knapsack Is Empty\n");
        else
        {
                printf("Elements of the most valuable subset are ");
                for(i=count-1;i>=0;i--)
                        printf("%d ",Knapsack[i]);
        }
        printf("\n");
        printf("Value is %d",table[items][W]);
}
int main(void)
{
        int table[10][10],weight[10],value[10],items,i,W;
```

```
        printf("Enter number of items\n");
        scanf("%d",&items);

        printf("Enter weight of the items\n");
        for(i=1;i<=items;i++)
                scanf("%d",&weight[i]);
        printf("Enter value of the items\n");
        for(i=1;i<=items;i++)
                scanf("%d",&value[i]);
        printf("Enter the Knapsack capacity\n");
        scanf("%d",&W);
        Knapsack(table,weight,value,items,W);
        return 0;
}
```

5. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
#include<iostream>
#define INFINITY 999
using namespace std;
void dijkstra(int n,int source,int cost[10][10],int distance[])
{
    int i,v,u,visited[10],min,count=2;
    for(i=1;i<=n;i++)
    visited[i]=0,distance[i]=cost[source][i];
    while(count<=n)
    {
            min=INFINITY;
            for(u=1;u<=n;u++)
                    if(distance[u]<min && !visited[u])
            min=distance[u],v=u;
            visited[v]=1;
            count++;
            for(u=1;u<=n;u++)
                    if((distance[v]+cost[v][u]<distance[u]) && !visited[u])
            distance[u]=distance[v]+cost[v][u];
    }
}
int main()
{
    int n,source,i,j,cost[10][10],distance[10];
    cout<<"Enter the number of nodes : ";
    cin>>n;
    cout<<"Enter the cost matrix :\n";
    for(i=1;i<=n;i++)
```

```
    for(j=1;j<=n;j++)
    {
            cin>>cost[i][j];
            if(cost[i][j]==0) cost[i][j]=INFINITY;
    }
    cout<<"Enter the source : ";
    cin>>source;
    dijkstra(n,source,cost,distance);
    cout<<"Shortest path from given source are :\n";
    for(i=1;i<=n;i++)
            if(i!=source)
                    cout<<source<<" --> "<<i<<" : cost = "<<distance[i]<<endl;
    return 0;
}
```

6. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
#include<iostream>

using namespace std;

void kruskal(int cost[10][10], int n)

{

        int parent[10],i,j,a,b,u,v,min,count=1,sum=0;

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

                parent[i]=0;

        while(count!=n)

        {

                min=9999;

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

                        for(j=1;j<=n;j++)

                                if(cost[i][j]<min)

                {

                        min=cost[i][j];

                        u=a=i;

                        v=b=j;

                }

                while(parent[u]!=0)
```

```
            u=parent[u];
        while(parent[v]!=0)
            v=parent[v];
        if(u!=v)
        {
            count++;
            sum=sum+cost[a][b];
            cout<<"\nEdge ("<<a<<","<<b<<") : "<<cost[a][b];
            parent[v]=u;
        }
        cost[a][b]=cost[b][a]=9999;
    }
    cout<<"\nWeight of minimum spanning tree = "<<sum<<endl;
}
int main()
{
    int cost[10][10],i,j,n;
    cout<<"\nEnter the number of vertices :  ";
    cin>>n;
    cout<<"\nEnter the cost matrix : \n";
    for(i=1;i<=n;i++)
```

```
            for(j=1;j<=n;j++)
                cin>>cost[i][j];
        kruskal(cost,n);
        return 0;
}
```

7. i. Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
#include<iostream>
using namespace std;
int visited[10];
void bfs(int n,int a[10][10],int source)
{
    int i,q[10],u,front=1,rear=1;
    visited[source]=1;
    q[rear]=source;
    while(front<=rear)
    {
            u=q[front];
            front++;
            for(i=1;i<=n;i++)
                    if(a[u][i]==1 && visited[i]==0)
                    {
                            rear++;
                            q[rear]=i;
                            visited[i]=1;
                    }
    }
}
int main()
{
    int n,a[10][10],i,j,source;
    cout<<"Enter the number of nodes : ";
    cin>>n;
    cout<<"Enter the adjacency matrix :\n";
    for(i=1;i<=n;i++)
```

```
            for(j=1;j<=n;j++)
                    cin>>a[i][j];
    cout<<"Enter the source : ";
    cin>>source;
    for(i=1;i<=n;i++)
            visited[i]=0;
    bfs(n,a,source);
    for(i=1;i<=n;i++)
    {
            if(visited[i]==0)
                    cout<<"The node "<<i<<" is NOT reachable.\n";
            else
                    cout<<"The node "<<i<<" is reachable.\n";
    }
    return 0;
}
```

ii. Check whether a given graph is connected or not using DFS method.

```
#include<iostream>
using namespace std;
int visited[10];
void dfs(int n,int a[10][10],int source)
{
        int i;
        visited[source]=1;
        for(i=1;i<=n;i++)
                if(a[source][i]==1 && visited[i]==0)
                        dfs(n,a,i);
}
int main()
{
        int i,j,source,n,a[10][10],count=0;
        cout<<"Enter the number of nodes : ";
        cin>>n;
        cout<<"Enter the adjacency matrix :\n";
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
```

```
                cin>>a[i][j];
        for(source=1;source<=n;source++)
        {
                for(i=1;i<=n;i++)
                        visited[i]=0;
                dfs(n,a,source);
        }
        for(i=1;i<=n;i++)
                if(visited[i]) count++;
        if(count==n)
                cout<<"The graph is connected.\n";
        else
                cout<<"The graph is NOT connected.\n";
        return 0;
}
```

8. **Find a subset of a given set S = {sl, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, 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.**

```
#include <iostream>
#include <stdlib.h>
#define MAX 20
using namespace std;
int stk[MAX];
int set[MAX];
int size;
int top = -1;
int c=1;
void push(int x)
{
   if(top==MAX-1)
   {
      cout<<"No more place";
      exit(0);
   }
   else
      stk[++top]=x;
}
void pop()
{
   if(top<0)
   {
      cout<<"No elements";
      exit(0);
   }
```

```
   else
      top--;
}
void display()
{
   cout<<"Solution "<<c<<":";
   c++;
    for (int i=0; i<=top; i++)
            cout << stk[i] << " ";
   cout<<endl;
}
int subset(int pos, int sum)
{
    int i;
    static int foundSoln = 0;
    if(sum>0)
    {
            for(i=pos;i<=size;i++)
            {
                    push(set[i]);
                    subset(i+1,sum-set[i]);
                    pop();
            }
    }
    if(sum==0)
    {
            display();
            foundSoln = 1;
    }
    return foundSoln;
}
```

```
int main()
{
    int i,sum;
    cout<<"Enter the maximum number of elements:";
    cin>>size;
    cout<<"Enter the elements:\n";
    for(i=1;i<=size;i++)
    cin>>set[i];
    cout<<"Enter the required sum:";
    cin>>sum;
    cout<<"\n";
    if (!subset(1, sum))
            cout<<"No solution";
    return 0;
}
```

9. Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

```
#include <iostream>
#include<iomanip>
#define MAX 10
using namespace std;
int path[MAX];
static int k=0;
int count=0;
int perm[120][7];
int tourcost[120];
void swap(int *x,int *y)
{
	int temp;
	temp=*x;
	*x=*y;
	*y=temp;
}
void DFS(int c,int v[MAX],int g[MAX][MAX],int n)
{
	int i;
	v[c]=1;
	path[k++]=c;
	for(i=0;i<n;i++)
	{
		if(g[c][i] && !v[i])
		{
			DFS(i,v,g,n);
```

```
            }
        }
}
void permute(int a[],int i,int n)
{
        int j,k;
        if(i==n)
        {
                for(k=0;k<=n;k++)
                        perm[count][k+1]=a[k];
                count++;
        }
        else
        {
                for(j=i;j<=n;j++)
                {
                        swap((a+i),(a+j));
                        permute(a,i+1,n);
                        swap((a+i),(a+j));
                }
        }
}
int apptsp(int n,int cost[MAX][MAX])
{
        int i,j,u,v,min,excost=0;
        int sum,k,t[MAX][2],p[MAX],d[MAX],s[MAX],tree[MAX][MAX];
        int source,count;
        int visited[MAX]={0};
        min=999;
        source=0;
        for(i=0;i<n;i++)
```

```
{
        for(j=0;j<n;j++)
        {
                if(cost[i][j]!=0&&cost[i][j]<=min)
                {
                        min=cost[i][j];
                        source=i;
                }
        }
}
for(i=0;i<n;i++)
{
        d[i]=cost[source][i];
        s[i]=0;
        p[i]=source;
}
s[source]=1;
sum=0;
k=0;
count=0;
while(count!=n-1)
{
        min=999;
        u=-1;
        for(j=0;j<n;j++)
        {
                if(s[j]==0)
                {
                        if(d[j]<=min)
                        {
                                min=d[j];
```

```
                    u=j;
                }
            }
        }
        t[k][0]=u;
        t[k][1]=p[u];
        k++;
        count++;
        sum+=cost[u][p[u]];
        s[u]=1;
        for(v=0;v<n;v++)
        {
            if(s[v]==0&&cost[u][v]<d[v])
            {
                d[v]=cost[u][v];
                p[v]=u;
            }
        }
}
for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
            tree[i][j]=0;
        }
}
if(sum>=999)
        cout<<"\nNo min cost span tree\n";
else
{
        for(i=0;i<k;i++)
```

```
            {
                tree[t[i][0]][t[i][1]]=tree[t[i][1]][t[i][0]]=1;
            }
        }
        DFS(0,visited,tree,n);
        cout<<"\n\nApproximate cost tour is"<<endl;
        for(i=0;i<=k;i++)
        {
            cout<<path[i]<<" -> ";
            excost+=cost[path[i]][path[i+1]];
        }
        cout<<path[0];
        excost+=cost[path[i]][path[0]];
        cout<<"\nCost(Approx):"<<excost;
        return excost;
}
int main(void)
{
        int a[MAX][MAX],city[10],fact=1;
    int n;
    cout<<"Enter number of cities:\n";
    cin>>n;
    cout<<"Enter an adjacency matrix:\n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
                cin>>a[i][j];
        }
    }
    int numofcities=n;
```

```
    int intercities=n-1,i,j;
    int mct=999,mctindex,appmct;
    for(i=0;i<n;i++)
            city[i]=i+1;
    permute(city,0,intercities-1);
    for(i=n-1;i>=1;i--)
            fact=fact*i;
    for(i=0;i<fact;i++)
    {
        for(j=0;j<n;j++)
        {
                tourcost[i]+=a[perm[i][j]][perm[i][j+1]];
        }
        if(mct>tourcost[i])
        {
                mct=tourcost[i];
                mctindex=i;
         }
     }
     cout<<"The exact cost tour is:\n";
     for(i=0;i<numofcities;i++)
          cout<<perm[mctindex][i]<<" -> ";
     cout<<perm[mctindex][i];
     cout<<"\nCost(Exact): "<<mct;
     appmct=apptsp(numofcities,a);
     cout<<"\n\nError: "<<appmct-mct;
     cout<<"\nAccuracy ratio: "<<(float)appmct/mct;
     cout<<"\nApproximate tour is ";
     cout<<(((float)appmct/mct)-1)*100<<"% longer than optimal tour\n";
     return 0;
}
```

10. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
#include<iostream>
using namespace std;
int c[10][10],n;
void prims()
{
    int visited[10],i,j,u,v,min,count=1,sum=0;
    for(i=1;i<=n;i++)
            visited[i]=0;
    visited[1]=1;
    while(count!=n)
    {
            min=9999;
            for(i=1;i<=n;i++)
                    for(j=1;j<=n;j++)
                            if(visited[i]==1)
                                    if(c[i][j]<min)
                                    {
                                            min=c[i][j];
                                            u=i;
                                            v=j;
                                    }
            if(visited[v]!=1)
            {
                    cout<<"\nEdge ("<<u<<","<<v<<") : "<<min;
                    visited[v]=1;
                    count++;
                    sum=sum+min;
            }
```

```
            c[u][v]=c[v][u]=9999;
    }
    cout<<"\nWeight of minimum spanning tree = "<<sum<<endl;
}
int main()
{
    int i,j;
    cout<<"Enter number of vertices : ";
    cin>>n;
    cout<<"Enter cost matrix : \n";
    for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                    cin>>c[i][j];
    prims();
}
```

11. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>
double SerialFloyd(int graph[10][10],int ver,double dstart,double dend)
{
    int i,j,k,iaFloyd[ver][ver];
    struct timeval tv;
            gettimeofday(&tv,NULL);
    dstart=tv.tv_sec+(tv.tv_usec/1000000.0);
    for(i=0;i<ver;i++)
    {
            for(j=0;j<ver;j++)
            {
                    iaFloyd[i][j]=graph[i][j];
            }
    }
    for(k=1;k<=ver;k++)
    {
            for(i=0;i<ver;i++)
            {
                    for(j=0;j<ver;j++)
                    {

    iaFloyd[i][j]=(iaFloyd[i][j]<(iaFloyd[i][k]+iaFloyd[k][j]))?(iaFloyd[i][j])          :
    (iaFloyd[i][k]+iaFloyd[k][j]);
                    }
            }
```

```
    }
    gettimeofday(&tv,NULL);
    dend=tv.tv_sec+(tv.tv_usec/1000000.0);
    return dend-dstart;
}
double ParallelFloyd(int graph[10][10],int ver,double dstart,double dend)
{
    int i,j,k,iaFloyd[ver][ver];
    struct timeval tv;
    gettimeofday(&tv,NULL);
    dstart=tv.tv_sec+(tv.tv_usec/1000000.0);
    for(i=0;i<ver;i++)
    {
            for(j=0;j<ver;j++)
            {
                    iaFloyd[i][j]=graph[i][j];
            }
    }
    omp_set_num_threads(20);
    #pragma omp parallel for private(k) shared(iaFloyd,ver)
    for(k=1;k<=ver;k++)
    {
            for(i=0;i<ver;i++)
            {
                    for(j=0;j<ver;j++)
                    {
                    iaFloyd[i][j]=(iaFloyd[i][j]<(iaFloyd[i][k]+iaFloyd[k][j]))?(iaFloyd[i][j]) :
                    (iaFloyd[i][k]+iaFloyd[k][j]);
                    }
            }
    }
```

```
    gettimeofday(&tv,NULL);
    dend=tv.tv_sec+(tv.tv_usec/1000000.0);

    return dend-dstart;
}
int main(void)
{
    int graph[10][10],ver,i,j;
    double dstart,dend,TimeS,TimeP,Factor;
    printf("Enter the number of vertices\n");
    scanf("%d",&ver);
    printf("Enter the adjacency matrix\n");
    for(i=0;i<ver;i++)
    {
            for(j=0;j<ver;j++)
            {
                    scanf("%d",&graph[i][j]);
            }
    }
    TimeS=SerialFloyd(graph,ver,dstart,dend);
    TimeP=ParallelFloyd(graph,ver,dstart,dend);
    printf("Time Taken By Serial Floyd = %f\n",TimeS);
    printf("Time Taken By Parallel Floyd = %f\n",TimeP);
    Factor=(TimeP/TimeS)*100;
    printf("Parallel Floyd is %f percent times faster than Serial Floyd\n",Factor);
    return 0;
}
```

12. Implement N Queen's problem using Back Tracking.

```
#include <stdio.h>
#include <stdlib.h>
int x[10];
int place(int k,int i)
{
    int j;
    for(j=1;j<=k-1;j++)
        if(x[j]==i || abs(x[j]-i)==abs(j-k))
            return 0;
    return 1;
}
void display(int n)
{
    int k,i,j;
    char cb[n][n];
    for(k=1;k<=n;k++)
        cb[k][x[k]]='Q';
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(j!=x[i])
            cb[i][j]='-';
        }
    }
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
```

```
                printf("%c\t",cb[i][j]);
            printf("\n");
    }
    printf("\n\n");
}
void NQueens(int k,int n)
{
    int i;
    for(i=1;i<=n;i++)
            if(place(k,i))
            {
                    x[k]=i;
                    if(k==n)
                            display(n);
                    else
                            NQueens(k+1,n);
            }
}
int main(void)
{
    int n,k=1;

    printf("Enter the dimensions of the chessboard\n");
    scanf("%d",&n);
    if(n==2 || n==3)
    {
     printf("No solution\n");
     exit(0);
    }
    NQueens(k,n);
    return 0;       }
```