Design and Analysis of Algorithms

Subject Code : 10CSL47

Lab Manual

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 <conio.h>
# include <time.h>

void Exch(int *p, int *q)
{
                        int temp = *p;
                        *p = *q;
                        *q = temp;
 }

void QuickSort(int a[], int low, int high)
{
                        int i, j, key, k;
                        if(low>=high)
                                return;
                        key=low; i=low+1; j=high;
                        while(i<=j)
                        {
                                while ( a[i] <= a[key] ) i=i+1;
                                while ( a[j] > a[key] )  j=j-1;
                                if(i<j) Exch(&a[i], &a[j]);
                        }
                        Exch(&a[j], &a[key]);
                        QuickSort(a, low, j-1);
                        QuickSort(a, j+1, high);
}

void main()
{
                        int n, a[1000],k;
```

```
    clock_t st,et;
    double ts;
    clrscr();
    printf("\n Enter How many Numbers: ");
    scanf("%d", &n);
    printf("\nThe Random Numbers are:\n");
    for(k=1; k<=n; k++)
       {
          a[k]=rand();
          printf("%d\t",a[k]);
       }
    st=clock();
    QuickSort(a, 1, n);
    et=clock();
    ts=(double)(et-st)/CLOCKS_PER_SEC;
    printf("\nSorted Numbers are: \n ");
    for(k=1; k<=n; k++)
          printf("%d\t", a[k]);
    printf("\nThe time taken is %e",ts);
    getch();
}
```

Output:

```
 Enter How many Numbers: 90

The Random Numbers are:
346     130     10982   1090    11656   7117    17595   6415    22948   31126
9004    14558   3571    22879   18492   1360    5412    26721   22463   25047
27119   31441   7190    13985   31214   27509   30252   26571   14779   19816
21681   19651   17995   23593   3734    13310   3979    21995   15561   16092
18489   11288   28466   8664    5892    13863   22766   5364    17639   21151
20427   100     25795   8812    15108   12666   12347   19042   19774   9169
5589    26383   9666    10941   13390   7878    13565   1779    16190   32233
53      13429   2285    2422    8333    31937   11636   13268   6460    6458
6936    8160    24842   29142   29667   24115   15116   17418   1156    4279

Sorted Numbers are:
 53     100     130     346     1090    1156    1360    1779    2285    2422
3571    3734    3979    4279    5364    5412    5589    5892    6415    6458
6460    6936    7117    7190    7878    8160    8333    8664    8812    9004
9169    9666    10941   10982   11288   11636   11656   12347   12666   13268
13310   13390   13429   13565   13863   13985   14558   14779   15108   15116
15561   16092   16190   17418   17595   17639   17995   18489   18492   19042
19651   19774   19816   20427   21151   21681   21995   22463   22766   22879
22948   23593   24115   24842   25047   25795   26383   26571   26721   27119
27509   28466   29142   29667   30252   31126   31214   31441   31937   32233

The time taken is 0.000000e+00
```

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 <conio.h>
#include<time.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);
}
void main()
{
```

```
	int n, a[2000],k;
	clock_t st,et;
	double ts;
	clrscr();
	printf("\n Enter How many Numbers:");
	scanf("%d", &n);
	printf("\nThe Random Numbers are:\n");
	for(k=1; k<=n; k++)
	{
		a[k]=rand();
		printf("%d\t", a[k]);
	}
	st=clock();
	MergeSort(a, 1, n);
	et=clock();
	ts=(double)(et-st)/CLOCKS_PER_SEC;
	printf("\n Sorted Numbers are : \n ");
	for(k=1; k<=n; k++)
		printf("%d\t", a[k]);
	printf("\nThe time taken is %e",ts);
	getch();
}
```

Output:

```
 Enter How many Numbers:15

The Random Numbers are:
346     130     10982   1090    11656   7117    17595   6415    22948   31126
9004    14558   3571    22879   18492
 Sorted Numbers are :
 130    346     1090    3571    6415    7117    9004    10982   11656   14558
17595   18492   22879   22948   31126
The time taken is 0.000000e+00_
```

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

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n,indegre[10];
void find_indegre()
{       int j,i,sum;
        for(j=0;j<n;j++)
        {
                sum=0;
                for(i=0;i<n;i++)
                        sum+=a[i][j];
                indegre[j]=sum;
        }
}
void topology()
{
        int i,u,v,t[10],s[10],top=-1,k=0;
        find_indegre();
        for(i=0;i<n;i++)
        {
                if(indegre[i]==0) s[++top]=i;
        }
        while(top!=-1)
        {
                u=s[top--];
                t[k++]=u;
                for(v=0;v<n;v++)
                {
                        if(a[u][v]==1)
                        {
                                indegre[v]--;
                                if(indegre[v]==0) s[++top]=v;
                        }
                }
        }
        printf("The topological Sequence is:\n");
        for(i=0;i<n;i++)
                printf("%d  ",t[i]);
```

```
}
void main()
{
	int i,j;
	clrscr();
	printf("Enter number of jobs:");
	scanf("%d",&n);
	printf("\nEnter the adjacency matrix:\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			scanf("%d",&a[i][j]);
	}
	topology();
	getch();
}
```

Output:

```
Enter number of jobs:6

Enter the 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
The topological Sequence is:
1  4  0  2  3  5
```

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

```
# include <stdio.h>
# include <conio.h>
int n,a[10][10],p[10][10];
void path()
{
	int i,j,k;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			p[i][j]=a[i][j];
	for(k=0;k<n;k++)
		for(i=0;i<n;i++)
			for(j=0;j<n;j++)
				if(p[i][k]==1&&p[k][j]==1) p[i][j]=1;
}
void main()
{
	int i,j;
	clrscr();
	printf("Enter the number of nodes:");
	scanf("%d",&n);
	printf("\nEnter the adjacency matrix:\n");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&a[i][j]);
	path();
	printf("\nThe path matrix is showm below\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			printf("%d ",p[i][j]);
		printf("\n");
	}
	getch();
}
```

Output:

```
Enter the number of nodes:4

Enter the adjacency matrix:
0       1       0       0
0       0       1       0
0       0       0       1
0       0       0       0

The path matrix is showm below
0  1  1  1
0  0  1  1
0  0  0  1
0  0  0  0
```

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

```
#include<stdio.h>
#include<conio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(int i,int j)
{
	return ((i>j)?i:j);
}
int knap(int i,int j)
{
	int value;
	if(v[i][j]<0)
	{
		if(j<w[i])
			value=knap(i-1,j);
		else
			value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
		v[i][j]=value;
	}
	return(v[i][j]);
}
void main()
{
	int profit,count=0;
	clrscr();
	printf("\nEnter the number of elements\n");
	scanf("%d",&n);
	printf("Enter the profit and weights of the elements\n");
	for(i=1;i<=n;i++)
	{
		printf("For item no %d\n",i);
		scanf("%d%d",&p[i],&w[i]);
	}
	printf("\nEnter the capacity \n");
	scanf("%d",&cap);
	for(i=0;i<=n;i++)
		for(j=0;j<=cap;j++)
			if((i==0)||(j==0))
```

```
                        v[i][j]=0;
                else
                        v[i][j]=-1;
        profit=knap(n,cap);
        i=n;
        j=cap;
        while(j!=0&&i!=0)
        {
                if(v[i][j]!=v[i-1][j])
                {
                        x[i]=1;
                        j=j-w[i];
                        i--;
                }
                else
                        i--;
        }
        printf("Items included are\n");
        printf("Sl.no\tweight\tprofit\n");
        for(i=1;i<=n;i++)
                if(x[i])
                        printf("%d\t%d\t%d\n",++count,w[i],p[i]);
        printf("Total profit = %d\n",profit);
        getch();
}
```

Output:

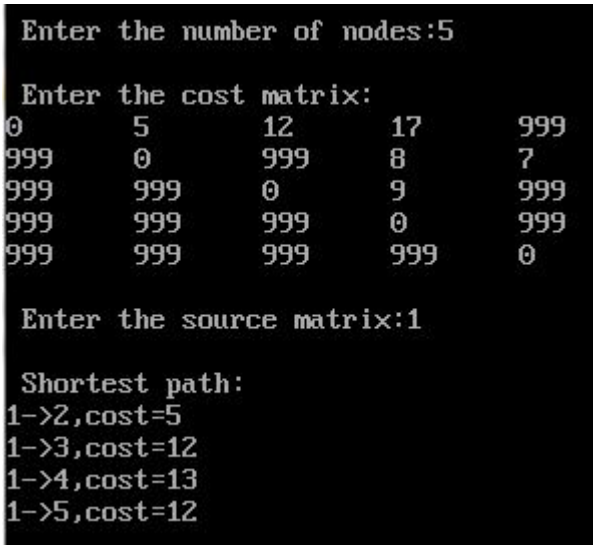
```
Enter the number of elements
3
Enter the profit and weights of the elements
For item no 1
10      30
For item no 2
20      15
For item no 3
30      50

Enter the capacity
45
Items included are
Sl.no   weight  profit
1       30      10
2       15      20
Total profit = 30
```

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

```
#include<stdio.h>
#include<conio.h>
#define infinity 999
void dij(int n,int v,int cost[10][10],int dist[100])
{
	int i,u,count,w,flag[10],min;
	for(i=1;i<=n;i++)
		flag[i]=0,dist[i]=cost[v][i];
	count=2;
	while(count<=n)
	{
		min=99;
		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];
	}
}

void main()
{
	int n,v,i,j,cost[10][10],dist[10];
	clrscr();
	printf("\n Enter the number of nodes:");
	scanf("%d",&n);
	printf("\n Enter the cost 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]=infinity;
```

```
        }
    printf("\n Enter the source matrix:");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\n Shortest path:\n");
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);
    getch();
}
```

Output:



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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
	clrscr();
	printf("\n\n\tImplementation of Kruskal's algorithm\n\n");
	printf("\nEnter the no. of vertices\n");
	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]=999;
		}
	}
	printf("\nThe edges of Minimum Cost Spanning Tree are\n\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("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
                mincost +=min;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
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;
}
```

Output:

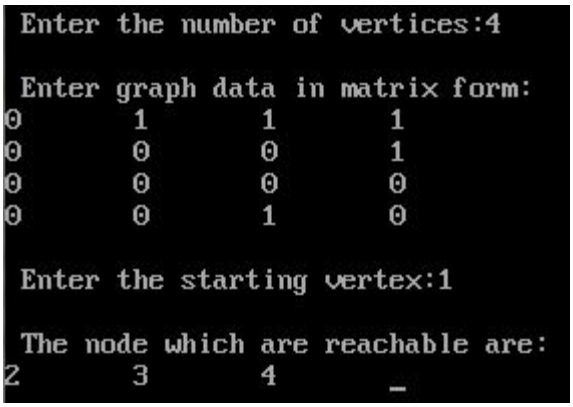
```
        Implementation of Kruskal's algorithm

Enter the no. of vertices
4

Enter the cost adjacency matrix
0       20      10      50
20      0       60      999
10      60      0       40
50      999     40      0

The edges of Minimum Cost Spanning Tree are

1 edge (1,3) =10

2 edge (1,2) =20

3 edge (3,4) =40

        Minimum cost = 70
```

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

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
 for(i=1;i<=n;i++)
  if(a[v][i] && !visited[i])
   q[++r]=i;
 if(f<=r)
 {
  visited[q[f]]=1;
  bfs(q[f++]);
 }
}
void main()
{
 int v;
 clrscr();
 printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  q[i]=0;
  visited[i]=0;
 }
 printf("\n Enter graph data in matrix form:\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
   scanf("%d",&a[i][j]);
 printf("\n Enter the starting vertex:");
 scanf("%d",&v);
 bfs(v);
 printf("\n The node which are reachable are:\n");
 for(i=1;i<=n;i++)
  if(visited[i])
   printf("%d\t",i);
```

```
 getch();
}
```

Output:

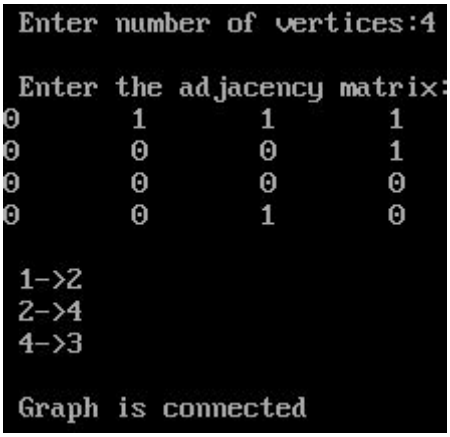


7. b. Check whether a given graph is connected or not using DFS method.

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
	int i;
	reach[v]=1;
	for(i=1;i<=n;i++)
	if(a[v][i] && !reach[i])
	{
		printf("\n %d->%d",v,i);
		dfs(i);
	}
}
void main()
{
	int i,j,count=0;
	clrscr();
	printf("\n Enter number of vertices:");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	{
		reach[i]=0;
		for(j=1;j<=n;j++)
			a[i][j]=0;
	}
	printf("\n Enter the adjacency matrix:\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
			scanf("%d",&a[i][j]);
	dfs(1);
	printf("\n");
	for(i=1;i<=n;i++)
	{
		if(reach[i])
		count++;
```

```
    }
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
getch();
}
```

Output:



8. Find a subset of a given set S = {s_1,s_2,.....,s_n} 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<stdio.h>
#include<conio.h>
int s[10] , x[10],d ;
void sumofsub ( int , int , int ) ;
void main ()
{
      int n , sum = 0 ;
      int i ;
      clrscr () ;
      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 ) ;
      getch () ;
}
void sumofsub ( int m , int k , int r )
{
      int i=1 ;
      x[k] = 1 ;
      if ( ( m + s[k] ) == d )
      {
            printf("Subset:");
            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] ) ;
            }
}
```

Output:

```
Enter the size of the set : 5

Enter the set in increasing order:
1       2       5       6       8

Enter the value of d :
9
Subset: 1       2       6
Subset: 1       8
```

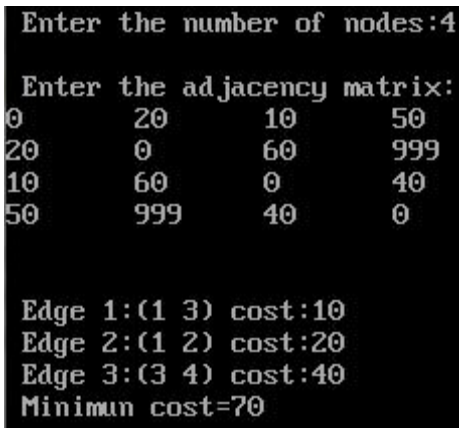
9. Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

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

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
	clrscr();
	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",&cost[i][j]);
			if(cost[i][j]==0)
			cost[i][j]=999;
		}
	visited[1]=1;
	printf("\n");
	while(ne<n)
	{
		for(i=1,min=999;i<=n;i++)
			for(j=1;j<=n;j++)
				if(cost[i][j]<min)
					if(visited[i]!=0)
					{
						min=cost[i][j];
						a=u=i;
						b=v=j;
					}
		if(visited[u]==0 || visited[v]==0)
		{
			printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
			mincost+=min;
			visited[b]=1;
		}
```

```
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

Output:



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<conio.h>
int min(int,int);
void floyds(int p[10][10],int n)
{
	int i,j,k;
	for(k=1;k<=n;k++)
		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				if(i==j)
					p[i][j]=0;
				else
					p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b)
{
	if(a<b)
		return(a);
	else
		return(b);
}
void main()
{
	int p[10][10],w,n,e,u,v,i,j;;
	clrscr();
	printf("\n Enter the number of vertices:");
	scanf("%d",&n);
	printf("\n Enter the number of edges:\n");
	scanf("%d",&e);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
			p[i][j]=999;
	}
	for(i=1;i<=e;i++)
```

```
    {
        printf("\n Enter the end vertices of edge%d with its weight \n",i);
        scanf("%d%d%d",&u,&v,&w);
        p[u][v]=w;
    }
    printf("\n Matrix of input data:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
            printf("%d \t",p[i][j]);
        printf("\n");
    }
    floyds(p,n);
    printf("\n Transitive closure:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
            printf("%d \t",p[i][j]);
        printf("\n");
    }
    printf("\n The shortest paths are:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            if(i!=j)
                printf("\n <%d,%d>=%d",i,j,p[i][j]);
        }
    getch();
}
```

Output:

```
 Enter the number of vertices:4

 Enter the number of edges:
5

 Enter the end vertices of edge1 with its weight
1       3       3

 Enter the end vertices of edge2 with its weight
2       1       2

 Enter the end vertices of edge3 with its weight
3       2       7

 Enter the end vertices of edge4 with its weight
3       4       1

 Enter the end vertices of edge5 with its weight
4       1       6_
```

```
999     999     3       999
2       999     999     999
999     7       999     1
6       999     999     999

 Transitive closure:
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0

 The shortest paths are:

 <1,2>=10
 <1,3>=3
 <1,4>=4
 <2,1>=2
 <2,3>=5
 <2,4>=6
 <3,1>=7
 <3,2>=7
 <3,4>=1
 <4,1>=6
 <4,2>=16
 <4,3>=9
```

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

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a[30],count=0;
int place(int pos)
{
	int i;
	for(i=1;i<pos;i++)
	{
		if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
			return 0;
	}
	return 1;
}
void print_sol(int n)
{
	int i,j;
	count++;
	printf("\n\nSolution #%d:\n",count);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(a[i]==j)
				printf("Q\t");
			else
				printf("*\t");
		}
		printf("\n");
	}
}
void queen(int n)
{
	int k=1;
	a[k]=0;
	while(k!=0)
	{
```

```
            a[k]=a[k]+1;
            while((a[k]<=n)&&!place(k))
                  a[k]++;
            if(a[k]<=n)
            {
                  if(k==n)
                        print_sol(n);
                  else
                  {
                        k++;
                        a[k]=0;
                  }
            }
            else
                  k--;
      }
}
void main()
{
      int i,n;
      clrscr();
      printf("Enter the number of Queens\n");
      scanf("%d",&n);
      queen(n);
      printf("\nTotal solutions=%d",count);
      getch();
}
```

Output:

```
Enter the number of Queens
4


Solution #1:
*       Q       *       *
*       *       *       Q
Q       *       *       *
*       *       Q       *


Solution #2:
*       *       Q       *
Q       *       *       *
*       *       *       Q
*       Q       *       *

Total solutions=2_
```