LAB MANUAL
On
Algorithms Laboratory
For MCA 3rd Semester of VTU

Instructions to the Readers:

1. This document is a useful material for the 3rd Semester MCA students (2016 Scheme – 16MCA38) of Visvesvaraya Technological University, Belagavi, Karnatka.
2. Though these programs are compiled and verified on Turbo C compiler, they can be executed on different C compilers with little/no modifications.
3. Additional explanation for the program is provided, wherever necessary. However, it is advised to know the theory concepts by referring to the respective subject notes (available in the website).
4. Clarifications/suggestions are welcome!!

1. Implement recursive Binary Search and Linear Search. Determine the time required to search an element. Repeat the experiment for different values of *n*, the number of elements in the list to be searched and plot a graph of the time taken versus n.

Explanation:

Functions for linear search and binary search have been developed. But, as binary search requires sorted elements, a function for sorting the elements also has been included in this program.

To compute the time taken by the algorithm, we make use of built-in data type ***clock_t*** which is available in the header file ***time.h***. Two variables of type *clock_t* are required to get the starting time and ending time of the algorithm. The built-in function ***clock()*** gives the current time of the system in terms of *number of clock-ticks*. So, to compute the time elapsed for the working of an algorithm, the different between starting time and ending time has to be considered and it must be divided by ***CLK_TCK*.**

As the processors are very fast, the time computed will be negligibly small and hence it may result in zero always. Hence, a ***delay*** is enforced wherever the basic operation is expected to execute.

Normally, time complexity can be better understood with a huge value of input size. Hence, instead of reading the array elements from the keyboard, it is suggested to generate random numbers.

Also, students are advised to run the program for different values of input and to note down the time taken.

Theoretically,

the time complexity of linear search = ***n***

the time complexity of binary search = $\log_2 n$

For a comparative study, it is better to draw the graph as per theoretical time complexity as well as the actual time computed through the program.

(NOTE: The above explanation holds good for Program 2, 3 and 7).

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

int binary(int item, int a[],int low,int high)
{
        int mid;

        mid=(low+high)/2;

        delay(10);

        if(low>high)
                return 0;
        if(a[mid]==item)
                return mid+1;
        else if(a[mid]>item)
                return binary(item,a,low,mid-1);
        else
                return binary(item,a,mid+1,high);
}

int linear(int item, int n, int a[])
{
        int i;
        for(i=0;i<n;i++)
        {
                delay(1);
                if(a[i]==item)
                        return i+1;
        }
```

```
	return 0;
}

void sort(int a[], int n)
{
	int i, j, temp;
	for(i=0;i<n-1;i++)
		for(j=i+1;j<n;j++)
			if(a[i]>a[j])
			{
				temp=a[i];
				a[i]=a[j];
				a[j]=temp;
			}
}

void main()
{
	int a[10000],n,pos,item, i, opt;
	clock_t start, end;
	double time;
	clrscr();

	printf("Enter the array size:");
	scanf("%d",&n);

	printf("\nElements are:\n");
	for(i=0;i<n;i++)
	{
		a[i]=(int)rand()%1000;
		printf("%d\t", a[i]);
	}

	for(;;)
	{
		printf("\n1. Binary Search \n2. Linear Search \n3. Exit");
		printf("\nEnter your option:");
		scanf("%d",&opt);

		switch(opt)
		{
			case 1:
				sort(a,n);
				printf("\nSorted list:\n");
```

```
                        for(i=0;i<n;i++)
                                printf("%d\t",a[i]);
                        printf("Enter the key element:");
                        scanf("%d",&item);
                        start=clock();
                        pos=binary(item,a,0,n-1);
                        end=clock();
                        break;
                case 2:
                        printf ("Enter the Element to be searched:\n");
                        scanf ("%d",&item);
                        start=clock();
                        pos=linear(item,n,a);
                        end=clock();
                        break;
                default : exit(0);
        }
        if(pos==0)
                printf("Item not found\n");
        else
                printf("Item found at the position %d",pos);

        time=(end-start)/CLK_TCK;
        printf("\n Time taken = %f", time);
    }
}
```

2. Sort a given set of elements using the Insertion 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.

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<math.h>

void main()
{
        int a[1000],n,i,item,j;
        clock_t start, end;
        float time;
```

```
        clrscr();
        printf("Enter the size of the array:");
        scanf("%d",&n);

        printf("\nRandomly generated array elements are:\n");
        for(i=0;i<n;i++)
        {
                a[i]=(int)rand()%100;
                printf("%d  ",a[i]);
        }

        start=clock();

        for(i=1;i<n;i++)
        {
                item=a[ i ];
                delay(10);

                for( j=i-1; j>=0 && item<a[ j ]; j--)
                        a[ j+1]=a[ j ];

                a[ j+1]=item;
        }

        end=clock();

        printf("\nSorted list is:\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);

        time= (float)(end-start)/CLK_TCK;
        printf("\nTime taken = %f", time);
        getch();
}
```

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.

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

```
void Merge(int b[1000], int c[1000],int a[1000],int p, int q)
{
        int i=0,j=0,k=0;

        while(i<p && j<q)
        {
                if(b[i]<c[j])
                {
                        a[k]=b[i];
                        i++;
                }
                else
                {
                        a[k]=c[j];
                        j++;
                }
                k++;
                delay(1);
        }

        while(i<p)
        {
                a[k]=b[i];
                i++;
                k++;
                delay(1);
        }
        while(j<q)
        {
                a[k]=c[j];
                j++;
                k++;
                delay(1);
        }

}

void MergeSort(int a[1000],int n)
{
        int b[1000],c[1000],i,j,p,q;

        if(n>1)
        {
```

```
                for(i=0;i<n/2;i++)
                        b[i]=a[i];
                p=i;

                for(i=n/2,j=0;i<n;i++,j++)
                        c[j]=a[i];
                q=j;

                MergeSort(b,p);
                MergeSort(c,q);
                Merge(b,c,a,p,q);
        }
        return;
}

void main()
{
        int a[1000],n,i;
        clock_t s, e;
        double t;

        clrscr();
        printf("Enter the size of array:");
        scanf("%d",&n);

        printf("\nThe elements are:");
        for(i=0;i<n;i++)
        {
                a[i]=(int)rand()/1000;
                printf("%d\t", a[i]);
        }

        s=clock();
        MergeSort(a,n);
        e=clock();

        printf("\nSorted list is:\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);
        printf("\nTime taken = %f", (e-s)/CLK_TCK);
        getch();
}
```

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

NOTE: The following program is developed for solving a topological ordering problem using source removal method. For this program, the graph should be **directed acyclic graph.** Hence, give the input adjacency matrix appropriately.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n, count =0, am[10][10], indeg[10], flag[10], i, j, k;
        clrscr();

        printf("Enter number of vertices:");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                indeg[i]=0;
                flag[i]=0;
        }

        printf("\nEnter adjacency matrix:\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&am[i][j]);

        printf("\nMatrix is :\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%d\t",am[i][j]);
                printf("\n");
        }

        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        indeg[i] += am[j][i];

        printf("\nThe topological ordering is:\n");
        while(count<n)
        {
                for(k=0;k<n;k++)
                        if((indeg[k]==0) && (flag[k]==0))
```

```
                {
                        printf("%d\n",k);
                        flag[k]=1;
                        count++;
                        for(i=0;i<n;i++)
                                if(am[k][i]==1)
                                        indeg[i]--;
                }
        }
        getch();
}
```

Output:

```
Enter number of vertices: 6
Enter adjacency matrix:
        0 1 1 0 0 0
        0 0 0 1 0 0
        0 0 0 0 1 0
        0 0 1 0 0 1
        0 0 0 0 0 0
        0 0 0 0 1 0

The topological ordering is:
        0       1       3       5       2       4
```

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

Method 1 uses recursive technique as given below. Alternative method applies the formula as per the theory chapter Dynamic Programming and it generates the table for all the values and weights. Students can opt any method as per their convenience.

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

int knapsack(int i, int cap, int n, int w[10],int v[10])
{
        int x, y;

        if (i==n)
                return ((cap<w[n])?0:v[n]);

        if(cap<w[ i ])
                return knapsack(i+1, cap, n, w, v);
```

```
        x=knapsack(i+1, cap, n, w, v);
        y=knapsack(i+1, cap-w[i], n, w, v) + v[ i ];

        return ((x>=y)?x:y);
}


void main()
{
        int n, i, w[10],v[10],cap, maxprofit;
        clrscr();

        printf("Enter number of items:");
        scanf("%d",&n);
        printf("\nEnter respective weights:");
        for(i=0;i<n;i++)
                scanf("%d",&w[i]);

        printf("\nEnter respective profits/values:");
        for(i=0;i<n;i++)
                scanf("%d",&v[i]);

        printf("\nEnter capacity:");
        scanf("%d",&cap);

        maxprofit=knapsack(0,cap,n,w,v);
        printf("\nMaximum profit is: %d",maxprofit);

        getch();
}
```

Output:

```
        Enter number of items:3

        Enter respective weights:
        3       1       2

        Enter respective profits/values:
        25      20      40

        Enter capacity: 4
        Maximum profit is: 60
```

Alternative Method for solving 0/1 Knapsack Problem:

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

int knapsack(int cap, int n, int w[10], int v[10], int V[10][10])
{
	int i, j;

	for(i=0;i<=n;i++)
		for(j=0;j<=cap;j++)
		{
			if(i==0)
				V[i][j]=0;
			else if(j==0)
				V[i][j]=0;
			else
			{
				if(j-w[i]<0)
					V[i][j]=V[i-1][j];
				else
					V[i][j]=(V[i-1][j]> v[i]+V[i-1][j-w[i]])? V[i-1][j]: v[i]+V[i-1][j-w[i]];
			}
		}
	return V[n][cap];
}

void main()
{
	int n, i,j,count=0, w[10],v[10],cap, maxprofit, V[10][10], sol[10];

	clrscr();

	printf("Enter number of items:");
	scanf("%d",&n);
	printf("\nEnter respective weights:");
	for(i=1;i<=n;i++)
		scanf("%d",&w[i]);

	printf("\nEnter respective profits/values:");
	for(i=1;i<=n;i++)
		scanf("%d",&v[i]);

	printf("\nEnter capacity:");
```

```
        scanf("%d",&cap);

        maxprofit=knapsack(cap,n,w,v, V);

        printf("\nThe table would be: \n");
        for(i=0;i<=n;i++)
        {
                for(j=0;j<=cap;j++)
                        printf("%d\t",V[i][j]);
                printf("\n");
        }

        j=n;
        i=0;
        while(cap>=0)
        {
                if(V[j][cap]!=V[j-1][cap])
                {
                        sol[i++]=j;
                        cap=cap-w[j];
                }
                else
                {
                        sol[i++]=j-1;
                        cap=cap-w[j-1];
                }
                count++;
                j--;
                if(j-1==0)
                        break;
        }

        printf("\nMaximum profit is: %d",maxprofit);
        printf("\nThe solution set is : ");
        for(i=count-1;i>=0;i--)
                printf("%d \t",sol[i]);

        getch();
}
```

6. 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>

void dijkstra(int n, int v, int cost[10][10],int dist[10])
{
        int count, u, i, w, visited[10], min;

        for(i=0;i<n;i++)
        {
                visited[i]=0;
                dist[i]=cost[v][i];
        }

        visited[v]=1;
        dist[v]=1;
        count=2;

        while(count<=n)
        {
                min=999;
                for(w=0;w<n;w++)
                        if((dist[w]<min) && (visited[w]!=1))
                        {
                                min=dist[w];
                                u=w;
                        }

                visited[u]=1;
                count++;
                for(w=0;w<n;w++)
                        if((dist[u]+cost[u][w]<dist[w]) && (visited[w]!=1))
                                dist[w]=dist[u]+cost[u][w];
        }
}

void main()
{
        int n, v, cost[10][10], dist[10], i, j;
        clrscr();

        printf("Enter number of vertices:");
        scanf("%d",&n);
```

```
        printf("\nEnter cost matrix (for infinity, enter 999):\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&cost[i][j]);

        printf("\nEnter source vertex:");
        scanf("%d",&v);

        dijkstra(n,v,cost,dist);

        printf("\nShortest path from \n");
        for(i=0;i<n;i++)
                if(i!=v)
                        printf("\n%d -> %d = %d", v, i, dist[i]);

        getch();
}
```

Output:

```
Enter number of vertices:7

Enter cost matrix (for infinity, enter 999):
0       2       999     3       999     999     999
2       0       9       999     1       4       999
999     9       0       999     999     3       999
3       999     999     0       5       999     7
999     1       999     5       0       999     4
999     4       3       999     999     0       6
999     999     999     7       4       6       0

Enter source vertex: 0

Shortest path from
0 -> 1 = 2
0 -> 2 = 9
0 -> 3 = 3
0 -> 4 = 3
0 -> 5 = 6
0 -> 6 = 7
```

7. Sort a given set of elements using Quick 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.

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

int quick_sort(int x[], int low, int high)
{
        int pos;
        if (low < high)
        {
                pos = partition(x,low,high);
                quick_sort(x,low,pos-1);
                quick_sort(x,pos+1,high);
        }
        return;
}

int partition(int x[], int low, int high)
{
        int key, temp, true = 1;
        int left, right;

        key = x[low];
        left = low +1;
        right = high;

        while(true)
        {
                while ((left < high) && (key >= x[left]))
                {
                        left++;
                        delay(1);
                }
                while(key < x[right])
                {
                        right--;
                        delay(1);
                }
```

```
                if(left < right)
                {
                        temp = x[left];
                        x[left] = x[right];
                        x[right] = temp;
                }
                else
                {
                        temp = x[low];
                        x[low] = x[right];
                        x[right] = temp;
                        return(right);
                }
        }
        return 0;
}

void main()
{
        int a[1000],n,i,low,high;
        clock_t s, e;
        float time;
        clrscr();

        printf("Enter array size\n");
        scanf("%d",&n);

        printf("\nElements are:\n");

        for(i=0;i<n;i++)
        {
                a[i]=(int)rand()%1000;
                printf("%d\t", a[i]);
        }

        low = 0;
        high = n-1;

        s=clock();
        quick_sort(a,low,high);
        e=clock();

        printf("The sorted list is \n");
        for(i=0;i<n;i++)
```

```
            printf("%d\t",a[i]);

        printf("\nTime taken =%f", (e-s)/CLK_TCK);

        getch();
}
```

Output:

```
Enter array size: 6
Enter the elements
23      1       67      -9      45      34

The sorted list is
-9      1       23      34      45      67

Time taken = 0.054912
```

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

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

void main()
{
    int n, v, u,cost[10][10], parent[10]={0}, i, j;
    int count=1, mincost=0, min, a, b;

    clrscr();

    printf("Enter number of vertices:");
    scanf("%d",&n);
    printf("\nEnter 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]=999;
        }

    while(count<n)
    {
        min=999;
```

```
        for(i=1;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)
        {
                count++;
                printf("\nEdge(%d, %d) = %d", a, b, min);
                mincost+=min;
                parent[v]=u;
        }

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

Output:

```
Enter number of vertices: 7

Enter cost matrix (for infinity, enter 999):
0       2       999     3       999     999     999
2       0       9       999     1       4       999
999     9       0       999     999     3       999
3       999     999     0       5       999     7
999     1       999     5       0       999     4
999     4       3       999     999     0       6
999     999     999     7       4       6       0

Edge(2, 5) = 1
Edge(1, 2) = 2
Edge(1, 4) = 3
Edge(3, 6) = 3
Edge(2, 6) = 4
```

```
Edge(5, 7) = 4
Minimum cost = 17
```

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

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

void dfs(int n, int a[10][10], int u, int visited[])
{
        int v;

        visited[u]=1;
        for(v=0;v<n;v++)
                if((a[u][v]==1)&& (visited[v]==0))
                        dfs(n,a,v,visited);
}

void main()
{
        int n, i, j, a[10][10], visited[10],flag, connected;

        clrscr();
        printf("Enter number of vertices:");
        scanf("%d",&n);
        printf("\nEnter adjacency matrix:\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);

        connected=0;

        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        visited[j]=0;

                dfs(n,a, i, visited);

                flag=0;

                for(j=0;j<n;j++)
                        if(visited[ j]==0)
                                flag=1;
```

```
            if(flag==0)
                connected=1;
        }

        if(connected==1)
            printf("\nGraph is connected");
        else
            printf("\nGraph is not connected");

        getch();
    }
```

Output 1:

```
Enter number of vertices: 6
Enter adjacency matrix:
0 1 0 1 0 0
1 0 1 1 0 0
0 1 0 1 0 0
1 1 1 0 0 0
0 0 0 0 0 1
0 0 0 0 1 0

Graph is not connected
```

Output 2:

```
Enter number of vertices: 3
Enter adjacency matrix:
0 1 1
1 0 1
1 1 0
Graph is connected
```

10. Find a subset of a given set S = {s1,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<stdio.h>
#include<conio.h>

void subset(int n, int d, int w[])
{
    int s=0, k=1, i, x[10];
```

```
    for(i=1;i<=n;i++)
        x[i]=0;

    x[k]=1;

    while(1)
    {
        if(k<=n && x[k]==1)
        {
            if(s+w[k]==d)
            {
                printf("\nSolution is:\n");
                for(i=1;i<=n;i++)
                    if(x[i]==1)
                        printf("%d ",w[i]);

                x[k]=0;
            }
            else if(s+w[k]<d)
                s=s+w[k];


            else
                x[k]=0;
        }
        else
        {
            k--;
            while(k>0 && x[k]==0)
                k--;
            if(k==0)
                break;

            x[k]=0;
            s=s-w[k];
        }
        k++;
        x[k]=1;
    }
}

void main()
{
    int n, i, d, w[10], sum=0;
```

```
        clrscr();

        printf("Enter value of n:");
        scanf("%d",&n);

        printf("\nEnter the set in ascending order:\n");
        for(i=1;i<=n;i++)
                scanf("%d",&w[i]);

        printf("\nEnter maximum value d:");
        scanf("%d",&d);

        for(i=1;i<=n;i++)
                sum+=w[i];

        if(sum<d || w[1]>d)
                printf("\nNo solution !!!");
        else
                subset(n, d, w);

        getch();
    }
```

Output:

```
Enter value of n: 5
Enter the set in ascending order:
2       4       6       10      15
Enter maximum value d: 12
Solution is:  2     4    6
Solution is:  2    10
```

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

Program: Horspool String Matching Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 100

void ShiftTable(char pattern[MAX], char table[MAX],int m)
{
```

```
        int i, x;

        for(i=0;i<MAX;i++)
                table[i]=m;

        for(i=0;i<m-1;i++)
        {
                x=pattern[i]-'0';
                table[x]=m-1-i;
        }
}

int horspool(char pattern[MAX],char text[MAX],char table[MAX], int m, int n)
{
        int index, i, k;

        ShiftTable(pattern, table, m);

        for(i=m-1;i<=n-1;)
        {
                k=0;
                while((k<=m-1) && (pattern[m-1-k]==text[i-k]))
                        k++;

                if(k==m)
                        return i-m+1;
                else
                {
                        index = text[i]-'0';
                        i=i+table[index];
                }
        }
        return -1;
}

void main()
{
        char text[MAX],pattern[MAX],table[MAX];
        int n, m,pos;

        clrscr();
        printf("Enter text:");
        gets(text);
        printf("\nEnter pattern:");
```

```
        gets(pattern);

        n=strlen(text);
        m=strlen(pattern);

        pos=horspool(pattern, text, table, m, n);

        if(pos==-1)
                printf("\nPattern not found!!!");
        else
                printf("\nPattern found at position %d",pos);

        getch();
}
```

Output 1:

```
Enter text:
        RNS Institute of Technology is located at Channasandra
Enter pattern:
        is located

Pattern found at position 28
```

Output 2:

```
Enter text:
        Hello, How are you?
Enter pattern:
        how

Pattern not found!!!
```

Program: Find the Binomial Coefficient using Dynamic Programming.

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

int binomial(int n, int k)
{
        int i, j, c[10][10];

        for(i=0;i<=n;i++)
                for(j=0;j<=k;j++)
```

```
                        if(j==0 || i==j)
                                c[i][j]=1;
                        else
                                c[i][j]=c[i-1][j-1]+c[i-1][j];

                return c[n][k];
        }

        void main()
        {
                int n, k, b;
                clrscr();

                printf("Enter value of n:");
                scanf("%d",&n);
                printf("\nEnter value of k (<=n):");
                scanf("%d",&k);

                if(k<=n)
                {
                        b=binomial(n,k);
                        printf("\n C(%d, %d)= %d", n, k, b);
                }
                else
                        printf("\n !!! Can not compute Binomial...n may be less than k !!!");
        }
```

Output 1:

```
Enter value of n: 6
Enter value of k (<=n):4
 C(6, 4)= 15
```

Output 2:

```
Enter value of n: 7
Enter value of k (<=n):2
 C(7, 2)= 21
```

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

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

```
void main()
{
	int n, v, u,cost[10][10], visited[10]={0}, i, j;
	int count=1, mincost=0, min, a, b;

	printf("Enter number of vertices:");
	scanf("%d",&n);
	printf("\nEnter cost matrix (For infinity, put 999):\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("\nThe edges of spanning tree are: \n");

	while(count<n)
	{
		min=999;
		for(i=1;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)
		{
			count++;
			printf("\nEdge(%d, %d) = %d", a, b, min);
			mincost+=min;
			visited[b]=1;
		}

		cost[a][b]=cost[b][a]=999;
```

```
        }

        printf("\nMinimum cost = %d", mincost);
}
```

Output:

```
Enter number of vertices: 6

Enter cost matrix (For infinity, put 999):
0       2       999     3       999     999
2       0       9       999     1       4
999     9       0       999     999     3
3       999     999     0       5       999
999     1       999     5       0       2
999     4       3       999     2       0

The edges of spanning tree are:

Edge(1, 2) = 2
Edge(2, 5) = 1
Edge(5, 6) = 2
Edge(1, 4) = 3
Edge(6, 3) = 3
Minimum cost = 11
```

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: All pair shortest path problem using Floyd's algorithm.

```
#include<stdio.h>
#include<conio.h>
void floyd(int cm[10][10], int n)
{
        int i, j, k;

        for(k=0;k<n;k++)
                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                                if((cm[i][k]+cm[k][j])<cm[i][j])
                                        cm[i][j] = cm[i][k]+cm[k][j];
}
```

```
void main()
{
        int n, i, j, cm[10][10];
        clrscr();

        printf("Enter number of vertices:");
        scanf("%d",&n);

        printf("\nEnter adjacency matrix (for infinity, put 9999):\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&cm[i][j]);

        floyd(cm,n);

        printf("\nThe all pair shortest path is :\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%d\t",cm[i][j]);
                printf("\n");
        }
        getch();
}
```

Output:

```
Enter number of vertices:4

Enter adjacency matrix (for infinity, put 999):
0       5       999     999
999     0       3       999
999     999     0       4
1       8       999     0

The all pair shortest path is :
0       5       8       12
8       0       3       7
5       10      0       4
1       6       9       0
```

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

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

void warshall(int a[10][10], int n)
{
        int i, j, k;

        for(k=0;k<n;k++)
                for(i=0;i<n;i++)
                        if(a[i][k]==1)
                                for(j=0;j<n;j++)
                                        a[i][j] = a[i][j] || a[k][j];

}

void main()
{
        int n, i, j, a[10][10];
        clrscr();

        printf("Enter number of vertices:");
        scanf("%d",&n);

        printf("\nEnter adjacency matrix :\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);

        warshall(a,n);

        printf("\nThe transitive closure is :\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%d\t",a[i][j]);
                printf("\n");
        }
        getch();
}
```

Output:

```
Enter number of vertices:4
```

```
Enter adjacency matrix :
0       1       0       1
0       0       1       0
0       0       0       1
0       1       0       0

The transitive closure is :
0       1       1       1
0       1       1       1
0       1       1       1
0       1       1       1
```

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

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

int place(int m[50], int k)
{
        int i;
        for(i=0;i<k;i++)
                if((m[i]==m[k])||(abs(m[i]-m[k])==abs(i-k)))
                        return 0;

        return 1;
}

void display(int m[50],int n)
{
        int i, j;
        int s[10][10]={0};

        for(i=0;i<n;i++)
                s[i][m[i]]=1;

        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        if(s[i][j])
                                printf("Q\t");
                        else
```

```
                                        printf("x\t");

                        printf("\n");
                }
                getch();
                exit(1);

        }

        void main()
        {
                int m[50], s[50][50], n, k;
                clrscr();

                printf("Enter number of Queens:");
                scanf("%d",&n);

                printf("\nThe solution for the problem is:\n");
                n--;



                for(m[0]=0, k=0;k>=0; m[k]+=1)
                {
                        while((m[k]<=n) && (!place(m,k)))
                                m[k]+=1;

                        if(m[k]<=n)
                                if(k==n)
                                        display(m,n+1);
                                else
                                {
                                        k++;
                                        m[k]=-1;
                                }
                        else
                                k--;
                }

                getch();
        }
```

Output:

Enter number of Queens:4

The solution for the problem is:

```
x  Q  x  x
x  x  x  Q
Q  x  x  x
x  x  Q  x
```

Enter number of Queens:5

The solution for the problem is:

```
Q  x  x  x  x
x  x  Q  x  x
x  x  x  x  Q
x  Q  x  x  x
x  x  x  Q  x
```