JAIN (Deemed-to-be-University)

SCHOOL OF COMPUTER SCIENCE AND IT

DEPARTMENT OF BACHELOR OF COMPUTER APPLICATIONS

V Semester (General)

A lab Manual on:

Analysis and Design of Algorithms

(16BCA5CD12L)

Manual Prepared by:

Bhavana Gowda D M

Asst.Professor, Dept.of BCA

School of CS & IT, Jain(Deemed-to-be University)

ANALYSIS AND DESIGN OF ALGORITHM LABORATORY

Subject Code-16BCA5CD12L **Course :BCA**

Sem/Specialisation : 5th/General **Teaching Hours: 15**

List Of Programs

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

2 Write a program to sort a given set of elements using the Merge sort method and determine the time required to sort elements. The elements can be generated using the random number generator.

3 Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method.

4 Write a program to check whether a given graph is connected or not using DFS method.

5 Write a program to obtain the Topological ordering of vertices in a given digraph.

6 Write a program to find shortest paths to other vertices from a given vertex in a weighted connected graph, using Single Source Shortest path algorithm.

7 Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

8 Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal algorithm.

9 Write a program to sort a given set of elements by implementing Radix Sort.

10 Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.

11 Write a program to implement All-Pairs Shortest Paths Problem using Floyd's algorithm.

12 Write a program to compute Binomial Co-Efficient using Dynamic Programming.

Program : 01

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

Algorithm

Algorithm QuickSort(a,low,high)

//Purpose :Sort the given array using quicksort

//Inputs: low: The position of first element in array a

high: The position of the last element of array a

a: It is an array consisting of unsorted elements

//Output a:It is an array consisting of sorted elements

if (low>high) **return**

//No elements to partition

k<--partition(a,low,high)

//Divide the array into two parts

QuickSort(a,low,k-1)

//Sort the left part of the array

QuickSort(a,k+1,high)   //Sort the right part of the array

"**algorithm QuickSort Ends here"**

<u>Algorithm partition(a,low,high)</u>

//Purpose :Divide the array inyo two parts such that elements towards left part of pivot element are<=pivot element // and elements towards right of key are>=pivot element

//Inputs: low: The position of first element in array a

//high: The position of the last element of array a

//A : It is an array consisting of unsorted elements

key<-a[low]
i<-low
j<-high+1

while(i<=j)

do i<-i+1 **while**(key>=a[i])

do j<-j-1 **while**(key<a[j])

end while

If (i<j) exchange(a[low],a[j])

return j **//End of the algorithm Partition**

Implementation

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
int partition(int a[],int low,int high)
	{
		int i,j,temp,key;
		key=a[low];
		i=low+1;
		j=high;
		while(1)
		{
			while(i<high && key>=a[i])
			i++;
			while(key<a[j])
			j--;
			if(i<j)
				{
					temp=a[i];
					a[i]=a[j];
					a[j]=temp;
				}
		   else
			{
				temp=a[low];
				a[low]=a[j];
				a[j]=temp;
				return j;
```

```
                  }
            }
      }
void quicksort(int a[],int low,int high)
{
  int j;
  if(low<high)

      {
            j=partition(a,low,high);
            quicksort(a,low,j-1);
            quicksort(a,j+1,high);
      }
}
void main()
{
  int i,n,a[20];
 float duration;
  clock_t start,end;
  clrscr();
  do{
  printf("\nEnter the no. of elements:\n");
  scanf("%d",&n);
  } while(n>10)
 printf("Random numbers are \n");
 for(i=0;i<n;i++)
{
      a[i]=rand()%100
      printf("%d",a[i]);
```

```
}
        start=clock();
        quicksort(a,0,n-1);
        delay(100);
        end=clock();
        printf("\nSorted elements are:\n");
        for(i=0;i<n;i++)
        printf("%d\n",a[i]);
        duration=(end-start)/CLK_TCK;
        printf("Time taken is in ms: %f",duration);
        getch();
}
```

Output:

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

Program 2:

Write a program to sort a given set of elements using the Merge sort method and determine the time required to sort elements. The elements can be generated using the random number generator.

Algorithm

Algorithm MergeSort(a,low,high)

//Purpose :Sort the given array between lower bound and upper bound

//Inputs: a is an array consisting of unsorted elements with low and high as lower bound and

// upper bound

//Output a:It is an array consisting of sorted elements

if (low>high) **return** //No elements to partition mid<-

(low+high)/2 //Divide the array into two parts

MergeSort(a,low,mid) //Sort the left part of the array

MergeSort(a,mid+1,high)//Sort the right part of the array

SimpleMerge(a,low,mid,high) // Merge the left part and

right part //**End of the algorithm MergeSort**

//Purpose: Merge two sorted arrays where the first array starts from low to mid and the second

// starts from mid+1 to high

//Input : a is sorted from the index position low to mid

// a is sorted from index position mid+1 to

high //Output : a is sorted from index low to high

i<-low

```
J<- mid+1
k<-low
while(i<=mid and j<=high)
    if(a[i]<a[j]) then
        c[k]<-a[i]    //Copy the lowest elements from first part of a to
        i<-i+1        c //Point to next item in the left part of a //Pont to
        k<-k+1        next item in C
    else
        c[k]<-a[j]    //Copy the lowest elements from second part of a to
        j<-j+1        c //Point to next item in the right part of a //Pont to
        k<-k+1        next item in C
    end if
end while
while(i<=mid)                 //Copy the remaining items from left part of a to c
    c[k]<-a[i]
    k<-k+1,i<-i+1
end while
while(j<=high)          // Copy the remaining items from right part of a to c
    c[k]<-a[j]
    k<-k+1,j<-j+1
end while
for i=low to high // Copy the elements from c to a a[i]<-
c[i]
end for    //End of Algorithm SimpleMerge
```

Implementation of Merge Sort

```
#include<stdio.h>
void ms(int a[ ],int low,int high);
main()
{
int a[100],n,i;
printf("\n enter the num of elements\n");
scanf("%d",&n);
printf("enter elements before sorting\n");
for(i=0;i<n;i++)
        {
           a[i]=rand()%100;
           printf("%d\n",a[i]);
        }
ms(a,0,n-1);
printf("array elements after sorting are\t");
for(i=0;i<n;i++)
printf("%d\n",a[i]);
getch();
return 0;
}
void sm(int a[],int low,int mid,int high)
{
int i=low,j=mid+1,k=low,c[100];
```

```
while(i<=mid && j<=high)
        {
                if(a[i]<a[j])
                {
                        c[k++]=a[i++];
                }
        else
                {
                        c[k++]=a[j++];
                }
        }
while(i<=mid)
{
        c[k++]=a[i++];
}
while(j<=high)
{
        c[k++]=a[j++];
}
for(i=low;i<=high;i++)
{
        a[i]=c[i];
}
}
```

```
void ms(int a[],int low,int high)
{
        int mid,i;
        if(low<high)
        {
                mid=(low+high)/2;
                {
                        ms(a,low,mid);
                        ms(a,mid+1,high);
                        sm(a,low,mid,high);
                }
        }
}
```

Output

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

Program 3:

Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method.

Algorithm BFS(a,n,source,T)

//Purpose: Traverse the graph from the given source node in BFS

//Input: a-adjacency matrix of the given graph

// n-the number of nodes in the graph

// source-from where the traversal is initiated

//Output:

// (u,v)-the nodes v reachable from u are stored in vector T

for i<-0 to n-1 **do**

s[i]=0

end for

f<-r<-0

q[r]<-source

s[source]<-1

k<-0

while(f<=r)

u<-q[r]

f<-f+1

for every v adjacent **to** u **do**

if v is not visited

s[v]<-1

```
                r<-r+1
                q[r]<-v
                T[k,1]<-u
                T[k,2]<-v,k<-k+1
        end if
    end for
end while    //End of algorithm BFS
```

Implementation

```
#include<stdio.h>
 #include<conio.h>
 int visited[10];
 void bfs(int n,int a[10][10],int source)
 {
        int i,g[10],u;
        int front=1,rear=1;
        visited[source]=1;

        while(front<=rear)
               {
                      u=g[front];
                      front=front+1;
                      for(i=1;i<=n;i++)
                      if(a[u][i]==1 && visited[i]==0)

                             {
                                    rear=rear+1;
                                    g[rear]=i;
                                    visited[i]=1;

                             }
               }
 }

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

```
clrscr();
printf("\n Enter the no. of nodes:");
scanf("%d",&n);
printf("\n Enter the adjacency matrix:");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter the source:");
scanf("%d",&source);
for(i=1;i<=n;i++)
visited[i]=0;
bfs(n,a,source);
for(i=1;i<=n;i++)
        {
                if(visited[i]==0)
                printf("\n The node %d is not reachable.",i); else
                printf("\n The node %d is reachable.",i);
        }
getch();}
```

Output:

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

Program 4 :

Write a program to check whether a given graph is connected or not using DFS method.

Algorithm

Algorithm DFS(u,n,a)

//Purpose: To obtain the sequence of jobs to be executed resulting in topological order

//Input :

// u-From where the DFS traversal start

// n-the number of vertices in the graph

// a-adjacency matrix of the given graph

//Global variables:

// s-to know what are the nodes visited and what are the nodes that are not visited

// j-index variable to store the vertices(only those nodes which are dead ends or those nodes

// whose nodes are completely explored

// res-an array which hold the order in which the vertices are popped

//Output:

// res-indicates the vertices in reverse order that are to be

executed Step 1:[Visit the vertex u]

S[u]<-1

Step 2:[Traverse deeper into the graph till we get the dead end or till all vertices are visited

for v<-0 to n-1 do

if(a[u][v]=1 ands[v]=0) **then**

DFS(v,n,a)

end if

end for

Step 3:[store the dead vertex or which is completely explored]

j<-j+1

res[j]<-u

Step4:[Finished]

return

//End of DFS Algorithm

Implementation

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

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);
}

void main()

{
	int n,a[10][10],i,j,source,count=0;
	clrscr();
	printf("\n Enter the no. of nodes:");
	scanf("%d",&n);
	printf("\n Enter the cost matrix,0-no edge and 1-if edge:\n");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	scanf("%d",&a[i][j]);
	printf("\n Enter the source vertex:");
	scanf("%d",&source);
	for(i=1;i<=n;i++)
	visited[i]=0;
	dfs(n,a,source);
```

```
        if(visited[i])
        count=count+1;
        if(count==n)
        printf("\n Graph is connected.");
        else
        printf("\n Graph is not connected.");
        getch();
}
```

Output:

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

<u>Program 5:</u>

Write a program to obtain the Topological ordering of vertices in a given digraph.

Algorithm DFS(u,n,a)

//Purpose: To obtain the sequence of jobs to be executed resulting in topological order

//Input :

// u-From where the DFS traversal start

// n-the number of vertices in the graph

// a-adjacency matrix of the given graph

//Global variables:

// s-to know what are the nodes visited and what are the nodes that are not visited

// j-index variable to store the vertices(only those nodes which are dead ends or those nodes

// whose nodes are completely explored

// res-an array which hold the order in which the vertices are popped

//Output:

// res-indicates the vertices in reverse order that are to be

executed Step 1:[Visit the vertex u]

S[u]<-1

Step 2:[Traverse deeper into the graph till we get the dead end or till all vertices are visited

for v<-0 to n-1 do

if(a[u][v]=1 ands[v]=0) **then**

DFS(v,n,a)

end if

end for

Step 3:[store the dead vertex or which is completely explored]

j<-j+1

res[j]<-u

Step4:[Finished]

return //End of DFS Algorithm

<u>Algorithm topological_order(a,n)</u>

////Purpose: To obtain the sequence of jobs to be executed resulting in topological order

//Input :

// n-the number of vertices in the graph

// a-adjacency matrix of the given graph

//Global variables:

// s-to know what are the nodes visited and what are the nodes that are not visited

// j-index variable to store the vertices(only those nodes which are dead ends or those nodes

// whose nodes are completely explored

// res-an array which hold the order in which the vertices are popped

//Output:

// res-indicates the vertices in reverse order that are to be executed

Step 1:[Initialization to indicate that no vertex has been visited]

for i<-0 **to** n-1 **do**

s[i]<-0

end for

j<-0

Step 2:[process each vertex in the graph]

for u<-0 **to** n-1 **do**

if(s[u]=0) call DFS(u,n,a)

end for

Step 3:[Output the topological sequence by printing in the revere order of popped sequence]

for i<-n-1**to** 0 **do**

print res[i]

end for

Step4:[Finished]

return **//End of Topological ordering Algorithm**

Implementation

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

int res[20],s[20],j=0;

void dfs(int u,int n,int cost[20][20])
{
        int v;
        s[u]=1;
        for(v=0;v<n;v++)
                {
                        if(cost[u][v]==1&&s[v]==0)
                                {
                                        dfs(v,n,cost);
                                }
                }
        res[j++]=u;
}

void depth_first_traversal(int n,int a[20][20])

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

                {
                        if(s[i]==0)
                        dfs(i,n,a);
                }
}
```

```
void main()
{
        int i,j,k,n,cost[20][20];
        clrscr();
        printf("enter the no of nodes\n");
        scanf("%d",&n);

        printf("enter the adjacency matrix\n");
        for(i=0;i<n;i++)
                {
                        for(j=0;j<n;j++)
                        scanf("%d",&cost[i][j]);
                }
 depth_first_traversal(n,cost);

 printf("topological sequence is:\n");
 for(i=n-1;i>=0;i--)
 printf("%d\t",res[i]);

 getch();
}
```

Output:

……………………………………………………………………………………….

……………………………………………………………………………………….

……………………………………………………………………………………….

Program 6:

Write a program to find shortest paths to other vertices from a given vertex in a weighted connected graph,using Single Source Shortest path algorithm.

Algorithm

Algorithm Dijkstra(n,w,source,destination,d,p)

//Purpose: To compute the shortest distance and shortest path from given source to

// destination

//Input: n-no of vertices in the graph

// w-Cost adjacency matrix with values>=0

// sorce-source vertex

// destination-destination vertex

//Output: d-shortest distance between source to all nodes

// p-shortest path from source to destination

// s-gives the nodes that are so far visited and the nodes that are not visited

for i<-0 to n-1 **do**

d[i]=cost[source][i]

p[i]=source

s[i]=0

end for

s[source]=1

//add source to s

for i<-0 to n-1 **do**

find u and d[u] such that d[u] is minimum and u є

v-s add u to s

if(u=destination) break;

for every v є v-s do (i.e,for v=0 to n-1)

if(d[u]+w[u,v]<d[v]

d[v]=d[u]+w[u,v]

p[v]=u

end if

end for

end for

//End of Dijkstra's algorithm

Implementation

```
#include<stdio.h>

#include<conio.h>
#define INFINITY 999

void dijkstra(int cost[10][10],int n,int source,int distance[10])
        {

                int visited[10],min,u,i,j;
                for(i=1;i<=n;i++)
                {
                        distance[i]=cost[source][i];
                        visited[i]=0;
                }

   visited[source]=1;
   for(i=1;i<=n;i++)

   {
        min=INFINITY;
        for(j=1;j<=n;j++)
        if(visited[j]==0 && distance[j]<min)

        {
           min=distance[j];
           u=j;
        }

        visited[u]=1;
         for(j=1;j<=n;j++)
        if(visited[j]==0 && (distance[u]+cost[u][j])<distance[j])


                {
                        distance[j]=distance[u]+cost[u][j];
                }
        }
```

```
}

void main()
{

   int n,cost[10][10],distance[10];
   int i,j,source,num;
   clrscr();
   printf("\nEnter the no. of nodes:");
   scanf("%d",&n);
   printf("\nCost matrix\nEnter 999 for no
     edge:\n");

   for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    scanf("%d",&cost[i][j]);
   printf("\nEnter the source node:");
   scanf("%d",&source);

   dijkstra(cost,n,source,distance);

       for(i=1;i<=n;i++)
   printf("\nShortest distance from %d to %d
   is %d\n",source,i,distance[i]); getch();
}
```

Output:

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

Program 7:
Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

Algorithm

Algorithm Prim(n,w)

//Purpose: To Compute the minimum spanning tree using Kruskal's algorithm

//Input: n-Number of vertices in the graph // w-Cost adjacency matrix //Output: The spanning tree if exists

```
Step1:[Obtain an edge with least cost from the adjacency
    matrix] min<-999;
    source<-1
    for i<-1 to n do
        for j<-1 to n do
            if(a[i][j]!=0 && a[i][j]<=min)
                min<-a[i][j]
                source<-i;
            end if
        end for
    end for
Step2:[initialization to find minimum spanning tree]
    for i<-1 to n do
        s[i]<-0
        d[i]<-w[source,i]
        p[i]<-source
    end for
```

Step3:[find minimum spanning tree]

s[source]<-1

sum<-0; k<-0

for i<-1 to n **do**

find u and d[u] such that d[u] is minimum and

uєV-S Add u to s

Add cost to selected edge to get total cost of minimum spanning

tree **for** every vєv-s **do**

if(w[u,v]<d[v]

d[v]<-w[u,v]

p[v]<-u

end if

end for

end for

if(sum≥999)

write "Spanning Tree does not exist"

else

write "Spanning tree exists and print the minimum spanning tree is"

end if

Step4:[Finished]

return   **//End of Prim's algorithm**

Implementation

```
#include<stdio.h>
 #include<conio.h>
 #define INFINITY 999
 int prim(int cost[10][10],int source,int n)
        {
  I             int i,j,visited[10],vertex[10],cmp[10];
                int min,u,v,sum=0;
                for(i=1;i<=n;i++)

                {
                        vertex[i]=source;
                        visited[i]=0;
                        cmp[i]=cost[source][i];
                }

        visited[source]=1;
        for(i=1;i<=n-1;i++)

                {

                        min=INFINITY;
                        for(j=1;j<=n;j++)
                        if(!visited[j] && cmp[j]<min)

                        {

                                min=cmp[j];
                                u=j;
                        }

         visited[u]=1;
         sum=sum+cmp[u];
          printf("\n %d->%d sum=%d",vertex[u],u,sum);
                        for(v=1;v<=n;v++)
                        if(!visited[v] && cost[u][v]<cmp[v])

                        {
```

```
                    cmp[v]=cost[u][v];
                    vertex[v]=u;
                }

            }

    return sum;
}

void main()

{

    int a[10][10],n,i,j,m,source;
    clrscr();
    printf("\n Enter the no. of vertices:");
    scanf("%d",&n);
    printf("\n Enter the cost matrix, 0-self loop and 999-no edge:\n");
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
     scanf("%d",&a[i][j]);
    printf("\n Enter the source:");
    scanf("%d",&source);
    m=prim(a,source,n);
    printf("\n\n Cost=%d",m);
    getch();
}
```

Output:

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

………………………………………………………………………………………………

Program 8 :

Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal algorithm

```
Algorithm Kruskal(n,m,E)
//Purpose:    To Compute the minimum spanning tree using Kruskal's algorithm
//Input:              n-Number of vertices in the graph
//            m-Number of edges in the graph
//            E-edge list consisting of set of edges along with equivalent weights
//Output:     The spanning tree
count<-0
k<-0
sum<-0
//Create forest with n vertices
for i<-1 to n do
         parent[i]<-i
end for
while(count!=n-1 and E!=ø)
        select an edge (u,v) with least cost
        j<-find(u,parent)   //find the root for the vertex u
        j<-find(v,parent)   //find the root for the vertex v
if(i!=j)                            //if the roots of vertex u and v are different

        t[k][0]<-u          //Select the edge(u,v) as the edge ofMSt
        t[k][1]<-v
        k++
        count++             //Update number of edges selected for MST
        sum<-sum+cost(u,v)       //Update the cost of MST
        union(i,j,parent);  //Merge the two trees with roots i and j
end if
//delete the edge (u,v)from the list
end while
if(count!=n-1)
        write("Spanning tree does not exist")
        return
end if
```

write (“the spanning tree is shown below
for i<-1 to n-1 **do**
    write(t[i][0],t[i][1])
end for
write(“cost of spanning tree is” sum)
//End of kruskal algorithm

Implementation

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 999
#define max 100

int parent[max],cost[max][max],t[max][2];

int find(int v)

    {
        while(parent[v])

        {

            v=parent[v];
        }

        return v;
    }

void union1(int i,int j)

    {
        parent[j]=i;
    }

void kruskal(int n)
```

```
{
        int i,j,k,u,v,res1,res2,sum=0,mincost;
        for(k=1;k<n;k++)

        {
                mincost=INFINITY;
                for(i=1;i<n;i++)
                {
                        for(j=1;j<=n;j++)

                        {
                                if(i==j)continue;
                                if(cost[i][j]<mincost)

                                {
                                        u=find(i);

                                        v=find(j);
                                        if(u!=v)

                                        {
                                                res1=i;
                                                res2=j;
                                                mincost=cost[i][j];
                                        }
                                }
                        }
                }

        union1(res1,find(res2));
        t[k][1]=res1;
        t[k][2]=res2;
        sum=sum+mincost;
}
```

```
  printf("\n Cost of spanning tree
  is %d\n",sum); printf("\n Edges of spanning
  tree are:\n");
     for(i=1;i<n;i++)
  printf("%d->%d\n",t[i][1],t[i][2]);
}


void main()

{
        int i,j,n;
        clrscr();
        printf("\n Enter the no. of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        parent[i]=0;
        printf("\n Enter the cost matrix,0-self edge and 999-no edge:\n");

        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&cost[i][j]);

        kruskal(n);

  getch();

}
```

Output:

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

……………………………………………………………………………………………

Program 9:

Write a program to sort a given set of elements by implementing Radix Sort

Implementation

```
#include <stdio.h>

#include <conio.h>
int largest (int arr[], int n);
void radix_sort (int arr[], int n);
void main ()
{
        int arr[10], i, n , j ,k;
        clrscr ();
        printf("\n Enter the number of elements in the array :
        " ); scanf("%d", &n);
        printf("\n Enter the elements of the array");
   for(i = 0;i < n;i++)
   {
       printf("\n arr[%d] = ", i);
       scanf ("%d", &arr [i]);
   }
         radix_sort (arr, n);
         printf ("\n The sorted array is: \n" );
    for(i = 0;i < n;i++)
    {
          printf ("%d\t", arr [i]);
    }
          getch ();
}
        int largest (int arr[], int n)
        {
             int large=arr[0], i;
              for(i = 1;i < n;i++)
              {
                 if(arr[i]> large)
                  {
                     large = arr[i];
```

```
                }

        }
                return large;
        }
void radix_sort (int arr[], int n)
 {
        int bucket [10] [10], bucket_count [10];
        int i, j , k , remainder, NOP=0, divisor=1, large,
        pass; large = largest (arr, n);
    while (large > 0)
     {
          NOP++;
              large=large/10;
         for(pass = 0;pass < NOP;pass++)
         {
                    /*Initialize the buckets */
             for(i = 0;i < 10;i++)
              {
                    bucket_count[i]=0;
              }
                   for(i = 0;i < n;i++)
                    {
  /* sort the numbers according to the digit at the place specified by
                       pass */ remainder= (arr[i]/divisor)%10;
                       bucket [remainder] [bucket_count[remainder]] =
                       arr[i]; bucket_count[remainder] += 1;
                    }
         }
              /* collect the numbers after PASS pass
            */ i=0;
       for(k=0;k < 10;k++)
       {
            for(j=0;j< bucket_count[k];j++)
              {
                   arr [i] = bucket [k] [j];
                   i++;
              }
```

```
        }
            divisor =divisor*10;
      }
  }
```

Output :

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

Program 10 :

Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm

Algorithm

```
Algorithm warshall(n,a,p)

//Purpose :To Compute transitive closure(path matrix)

//Inputs: Adjacency matrix a of size n x n

//Output : Transitive Closure(path matrix) of size n

x n Step 1:[Make a copy of the adjacency matrix]

    for i<-0 to n-1 do

        for j<-0 to n-1 do

            p[i,j]=a[i,j]

        end for

    end for

Step 2:[Find transitive closure(path matrix)] for k<-0 to n-1 do
        for i<-0 to n-1 do

            for j<-0 to n-1 do

                if(p[i,j]=0 and(if(p[i,k]=1 and p[k,j]=1))

                    then p[i,j]=1

                end if

            end for

        end for

    end for

Step 3:[Finished]   Return      //End of Warshall Algorithm
```

Implementation

```
#include<stdio.h>
#include<conio.h>
 void warsh(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++)
  p[i][j]=p[i][j]||(p[i][k] && p[k][j]);
}
 void main()
{
   int a[10][10],n,i,j;
   clrscr();
   printf("\n Enter the no. of vertices:");
   scanf("%d",&n);
   printf("\n Enter the cost matrix, 0-self loop and 1-for
   edge\n"); for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
          scanf("%d",&a[i][j]);
   warsh(a,n);
   printf("\n Resultant matrix is:\n");
   for(i=1;i<=n;i++)
   {
       for(j=1;j<=n;j++)
           printf("%d\t",a[i][j]);
           printf("\n");
   }
   getch();
}
```

Output:

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

Program 11:

Write a program to implement All-Pairs Shortest Paths Problem using Floyd's algorithm.

Algorithm

```
Algorithm Flyod(n,cost,D)
//Purpose :To Compute all pair shortest distance
matrix //Inputs: Adjacency matrix a of size n x n
//Output : Shortest distance matrix of size n
    x n for i<-1 to n do
        for j<-1 to n do
            D[i,j]=cost{I,j]
        end for
    end for
for k<-1 to n do
    for i<-1 to n do
        for j<-1 to n do
            D[I,j]=min(D[i,j],D[i,k]+D[k,j])
        end for
    end for
end for
return
//end of Algorithm Floyd
```

Implementation

```
#include<stdio.h>
#include<conio.h>
#include<omp.h>
#define INFINITY 999

int min(int a,int b)
{
  return a<b?a:b;
}
void floyd(int w[10][10],int n)
{
 int i,j,k;

#pragma omp parallel for private(i, j, k) shared(w)
 for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
          w[i][j]=min(w[i][j],w[i][k]+w[k][j]);
}
void main()
{
  int a[10][10],n,i,j;
  double startTime,endTime;
  printf("\n Enter the no. of vertices:");
  scanf("%d",&n);
  printf("\n Enter the cost matrix, 0-self loop and 999-no
  edge\n"); for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
          scanf("%d",&a[i][j]);
  startTime=omp_get_wtime();
  floyd(a,n);
  endTime = omp_get_wtime();
  printf("\n Shortest path matrix:\n");
  for(i=1;i<=n;i++)
  {
     for(j=1;j<=n;j++)
```

```
            printf("%d\t",a[i][j]);
            printf("\n");
        }
            printf("Time taken is %10.9f\n",(double)(endTime-startTime));

        getch();
    }
```

Output

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

<u>Program 12:</u>

Write a program to compute Binomial Co-Efficient using Dynamic programming.

```
#include<stdio.h>

//      Prototype of a utility function that returns minimum of two
integers int min(int a, int b);

// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
   int C[n+1][k+1];
   int i, j;

   //   Caculate value of Binomial Coefficient in bottom up
   manner for (i = 0; i <= n; i++)
   {
      for (j = 0; j <= min(i, k); j++)
      {
         // Base Cases
         if (j == 0 || j == i)
            C[i][j] = 1;

         //     Calculate value using previosly stored
         values else
            C[i][j] = C[i-1][j-1] + C[i-1][j];
      }
   }

   return C[n][k];
}

//      A utility function to return minimum of two
integers int min(int a, int b)
{
   return (a<b)? a: b;
}

/* Drier program to test above function*/
int main()
```

```
{
    int n = 5, k = 2;
    printf ("Value of C(%d, %d) is %d ", n, k, binomialCoeff(n,k) );
    return 0;
}
```

Output :

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………