**B. M. S. College of Engineering**

**Bull Temple Road, Bangalore-560019**

(Affiliated to Visvesvaraya Technological

University, Belgaum)



**DEPARTMENT:** COMPUTER SCIENCE AND ENGINEERING

**PROGRAM:** UG

**NAME:** RAMYA RAMESH

**USN:** 1BM19CS227

**SECTION:** 4-‘D’

**COURSE NAME:** Analysis and Design of Algorithms

**COURSE CODE:** 19CS4PCADA

NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 10: Sort a given set of N integer elements using Heap Sort technique and compute its time taken**

1. RANDOM GENERATED:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int n, int i) {

int large = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[large])

large = l;

if (r < n && arr[r] > arr[large])

large = r;

if (large != i) {

swap(&arr[i], &arr[large]);

heapify(arr, n, large);

}

}

void HeapSort(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

int main() {

clock\_t start,end;

double time;

int arr[100000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

for (i = 0; i < n; i++)

{

arr[i] = rand()%100;

printf("%d ",arr[i]);

}

start = clock();

HeapSort(arr, n);

end = clock();

printf("\nSorted array is \n");

for (i = 0; i < n; i++)

{

printf("%d ", arr[i]);

}

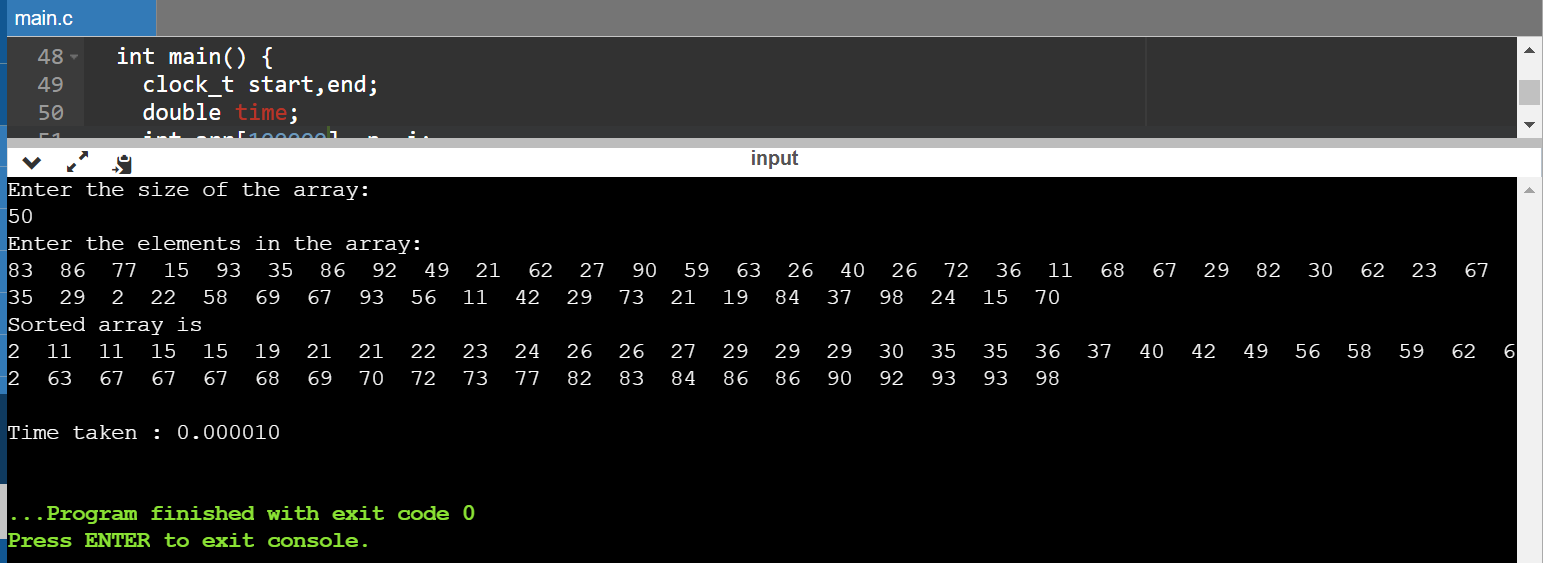
printf("\n");

time = ((double)(end - start))/CLOCKS\_PER\_SEC;

printf("\nTime taken : %lf\n",time);

}

OUTPUT:



|  |  |
| --- | --- |
| N | Time |
| 50 | 0.000010 |
| 100 | 0.000019 |
| 200 | 0.000042 |
| 500 | 0.000127 |
| 1000 | 0.000228 |
| 5000 | 0.001334 |
| 10000 | 0.003058 |
| 15000 | 0.004706 |
| 20000 | 0.007041 |
| 25000 | 0.008934 |

1. MANUAL (USER INPUT):

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int n, int i) {

int large = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[large])

large = l;

if (r < n && arr[r] > arr[large])

large = r;

if (large != i) {

swap(&arr[i], &arr[large]);

heapify(arr, n, large);

}

}

void HeapSort(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

int main() {

clock\_t start,end;

double time;

int arr[100],n,i;

printf("Enter the size of the array:\n");

scanf("%d",&n);

printf("Enter the elements in the array:\n");

for (i = 0; i < n; i++)

{

scanf("%d",&arr[i]);

}

start = clock();

HeapSort(arr, n);

end = clock();

printf("\nSorted array is \n");

for (i = 0; i < n; i++)

{

printf("%d ", arr[i]);

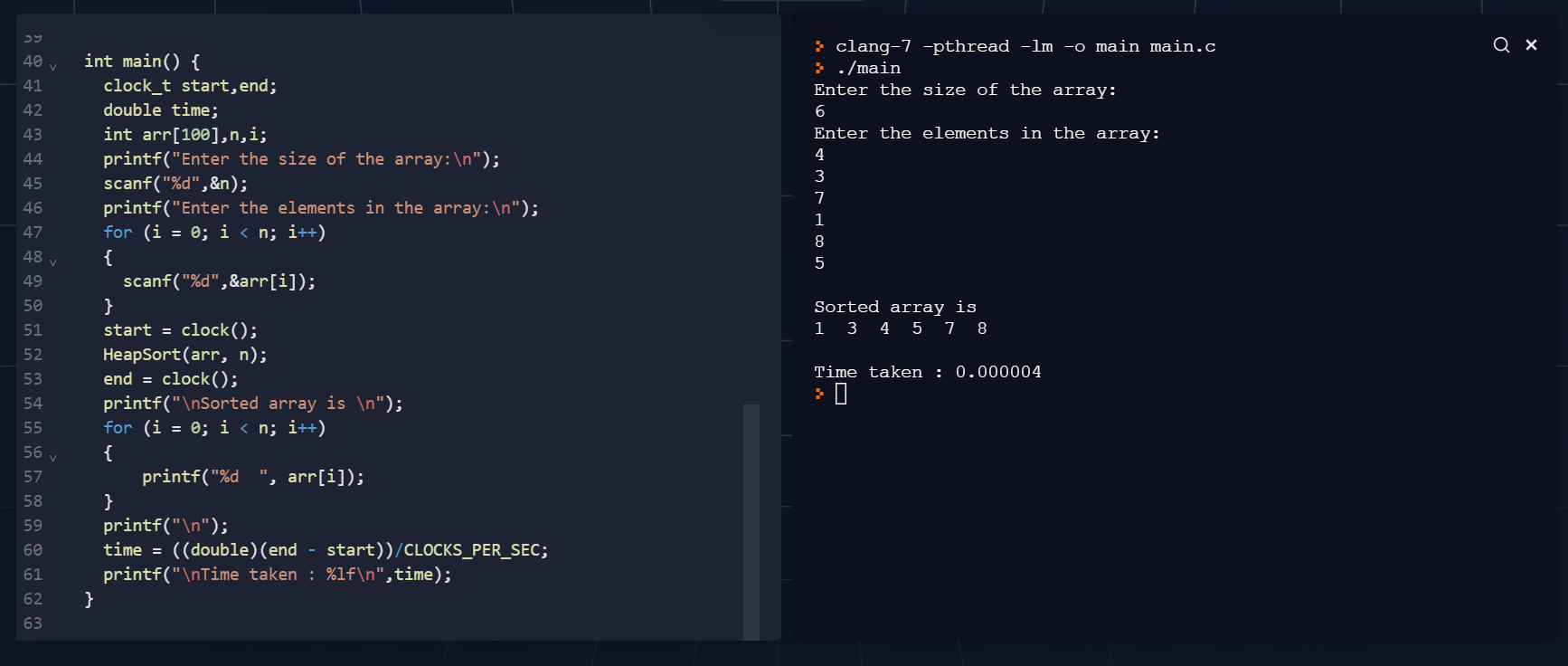
}

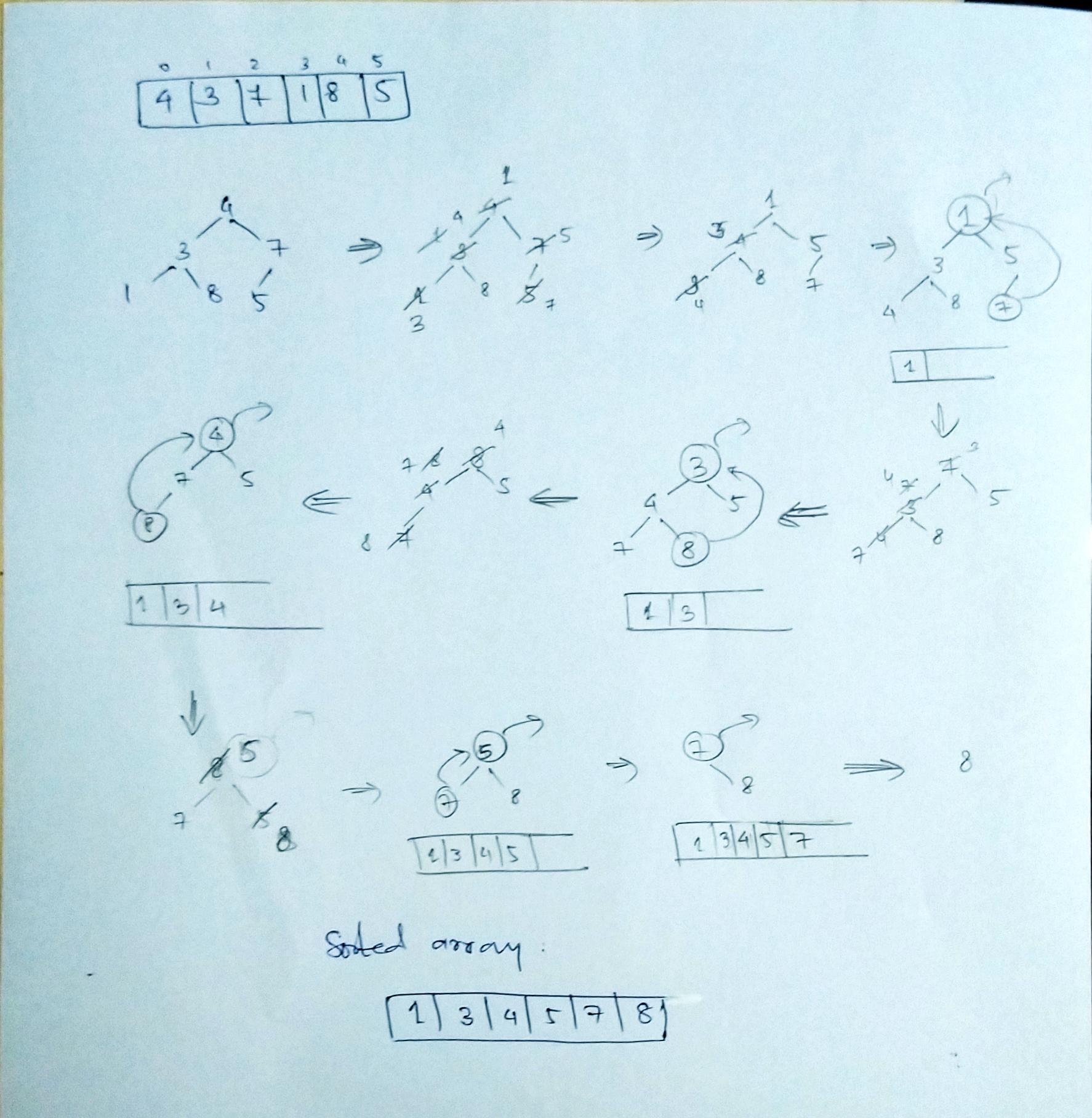
printf("\n");

time = ((double)(end - start))/CLOCKS\_PER\_SEC;

printf("\nTime taken : %lf\n",time);

}





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 11: Implementation of Warshall’s algorithm using dynamic programming**

CODE:

#include<stdio.h>

#include<math.h>

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

void warshall(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]=max(p[i][j],p[i][k]&&p[k][j]);

}

void main() {

int p[10][10]= {

0

}

,n,e,u,v,i,j;

printf("\n Enter the number of vertices:");

scanf("%d",&n);

printf("\n Enter the number of edges:");

scanf("%d",&e);

for (i=1;i<=e;i++) {

printf("\n Enter the end vertices of edge %d:",i);

scanf("%d%d",&u,&v);

p[u][v]=1;

}

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");

}

warshall(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]);

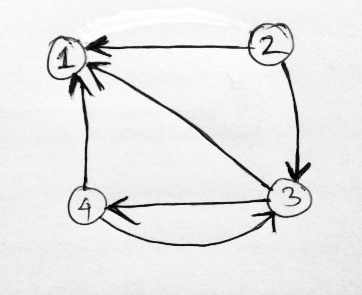
printf("\n");

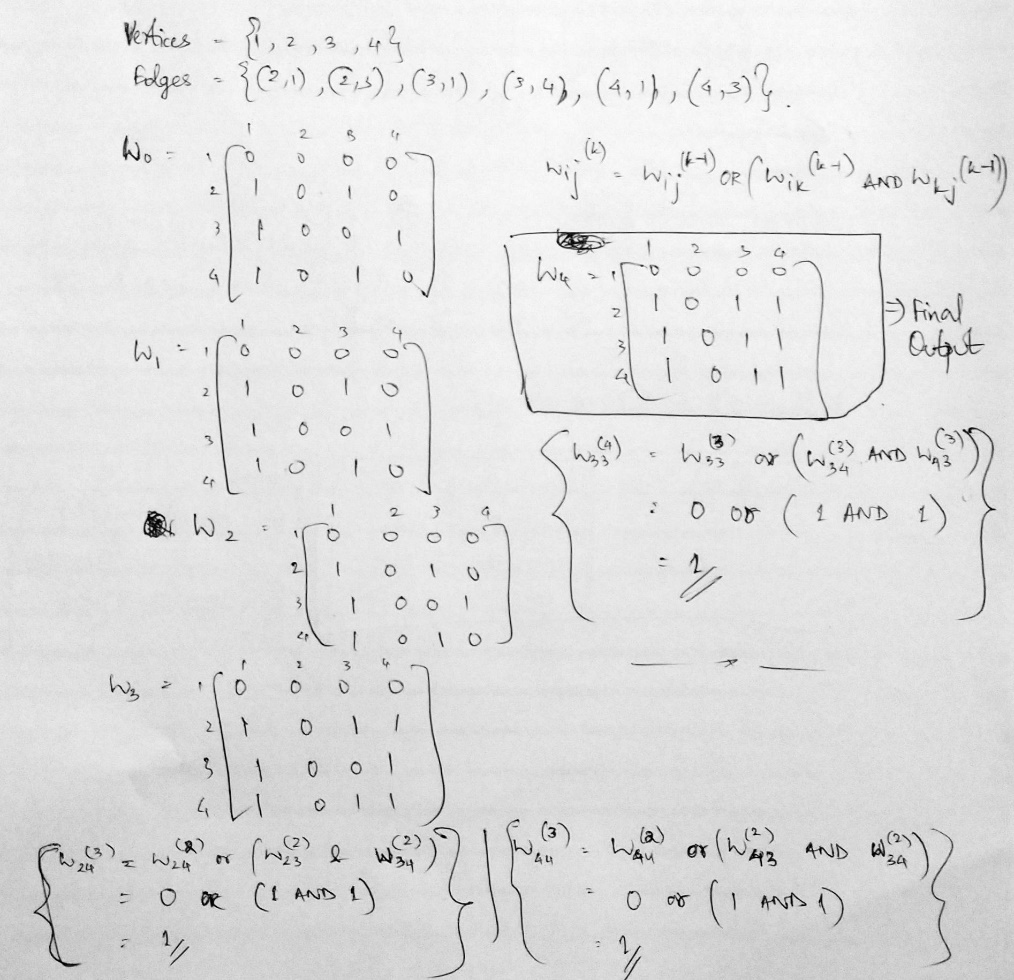
}

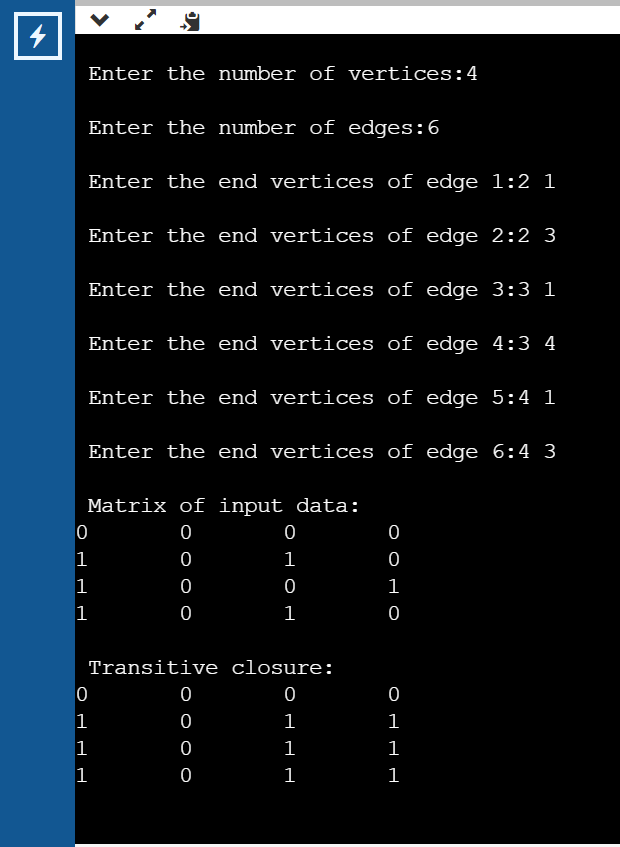
}

OUTPUT:

Graph:







NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 12: Implement 0/1 Knapsack problem using dynamic programming**

CODE:

#include<stdio.h>

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

int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

int K[n+1][W+1];

for (i = 0; i <= n; i++)

{

for (w = 0; w <= W; w++)

{

if (i==0 || w==0)

K[i][w] = 0;

else if (wt[i-1] <= w)

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

else

K[i][w] = K[i-1][w];

}

}

return K[n][W];

}

int main()

{

int i, n, val[100], wt[100], W;

printf("\nEnter number of items:\n");

scanf("%d", &n);

printf("\nEnter value and weight of items:\n");

for(i = 0;i < n; ++i){

scanf("%d%d", &val[i], &wt[i]);

}

printf("\nEnter size of knapsack:\n");

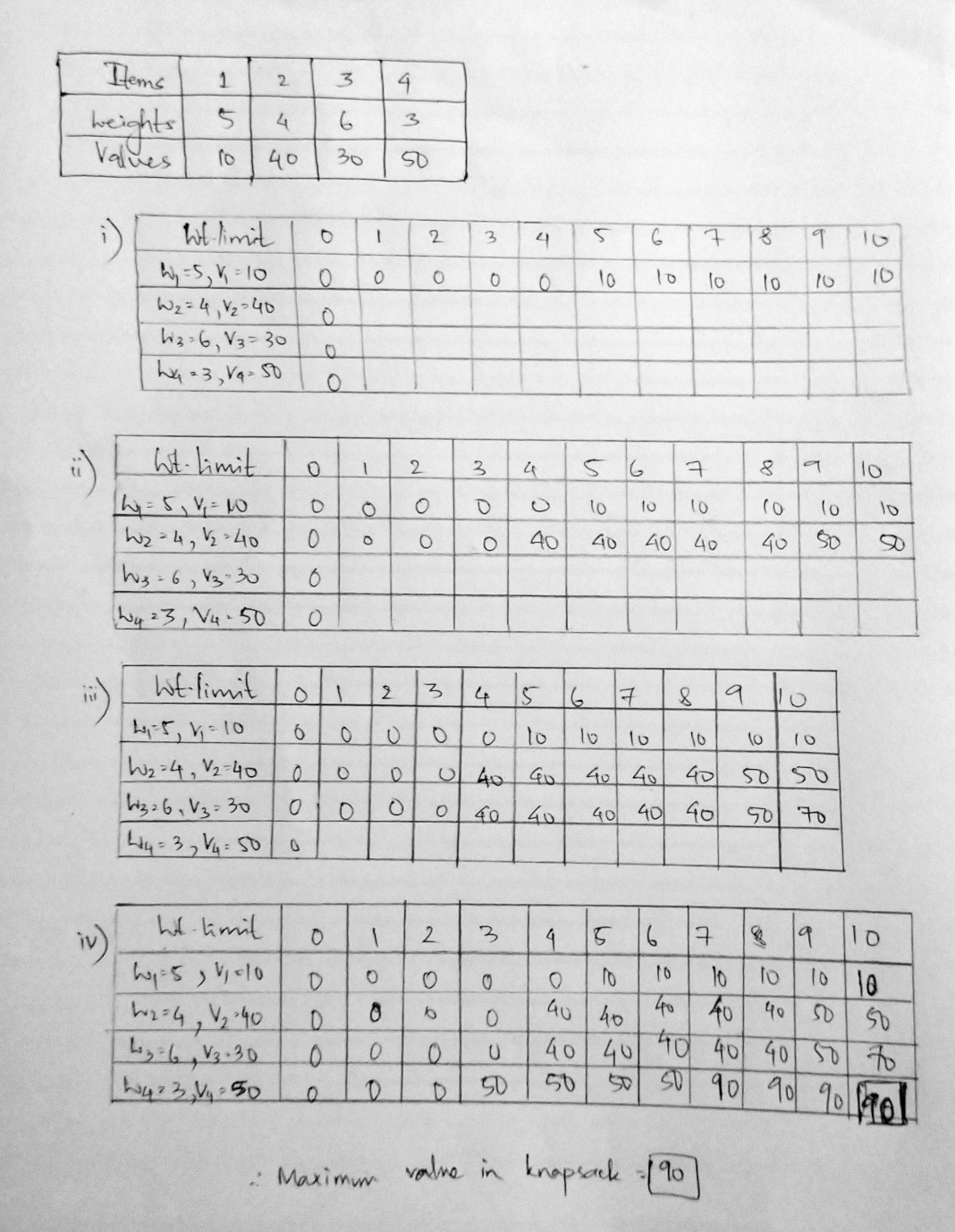
scanf("%d", &W);

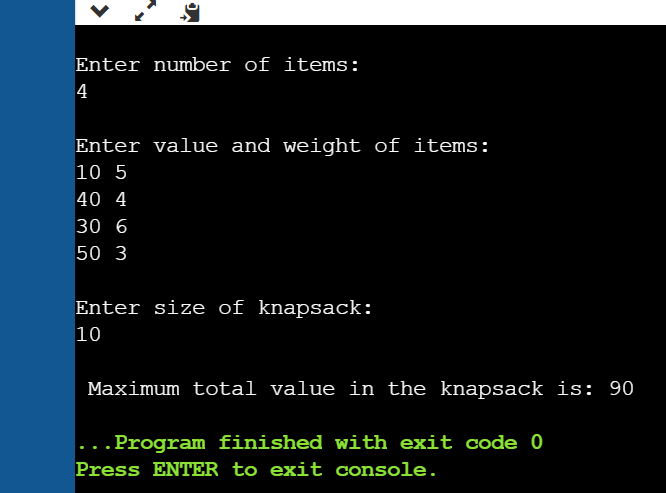
printf("\n Maximum total value in the knapsack is: %d", knapSack(W, wt, val, n));

return 0;

}

OUTPUT:





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 13: Implementation of All Pair Shortest paths problem using Floyd’s algorithm.**

CODE:

#include<stdio.h>

#include<math.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;

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]=99999;

}

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:\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 The shortest paths are:\n");

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

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

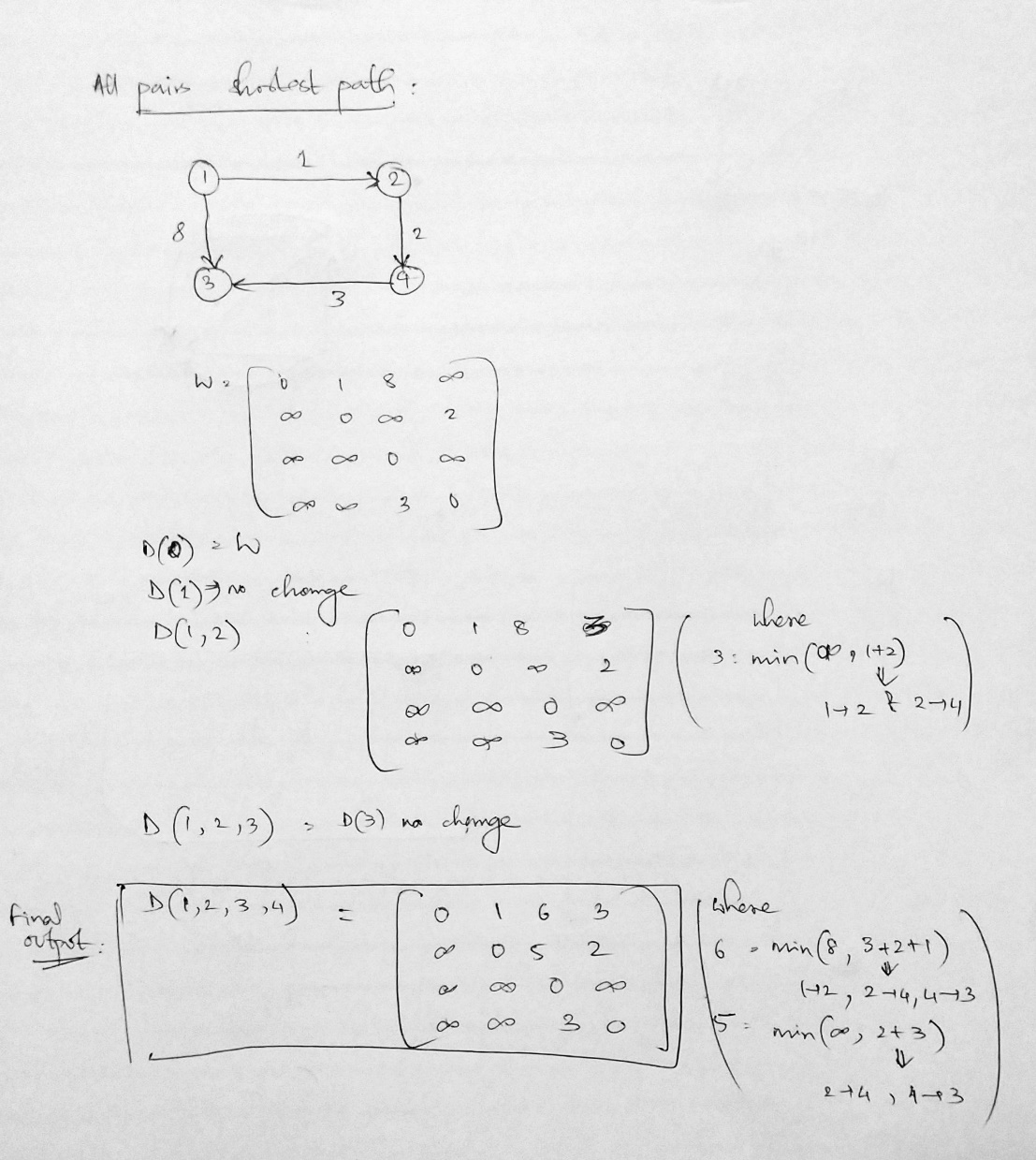
if(i!=j)

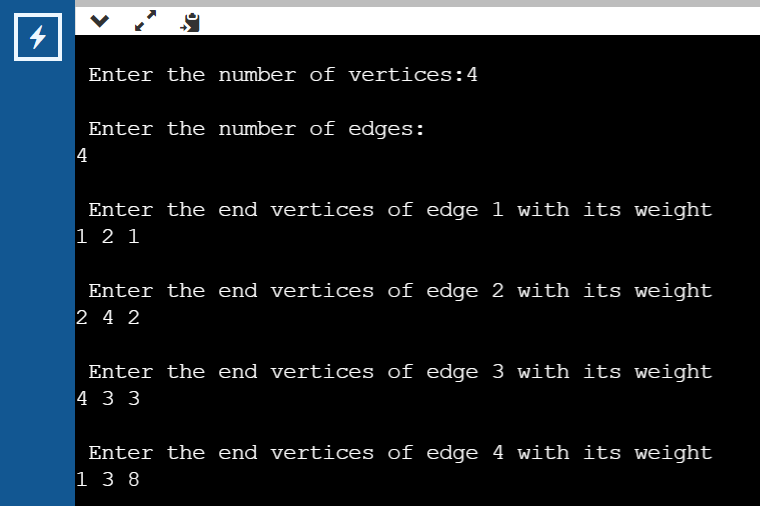
printf("\n <%d,%d> = %d",i,j,p[i][j]);

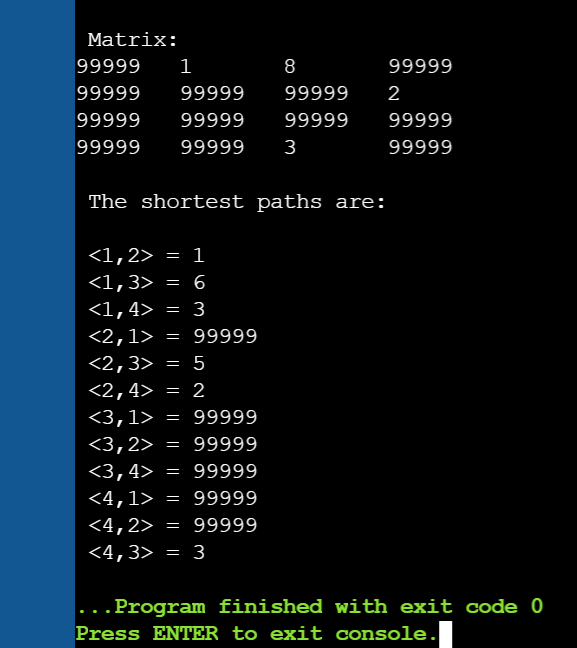
}

}

OUTPUT:







NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 14: Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

CODE:

#include<stdio.h>

int a,b,u,v,n,i,j,count=1;

int visited[10]= {

0

}

,min,mincost=0,cost[10][10];

void main() {

printf("\n\tPrim's algorithm\n");

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(count<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",count++,a,b,min);

mincost+=min;

visited[b]=1;

}

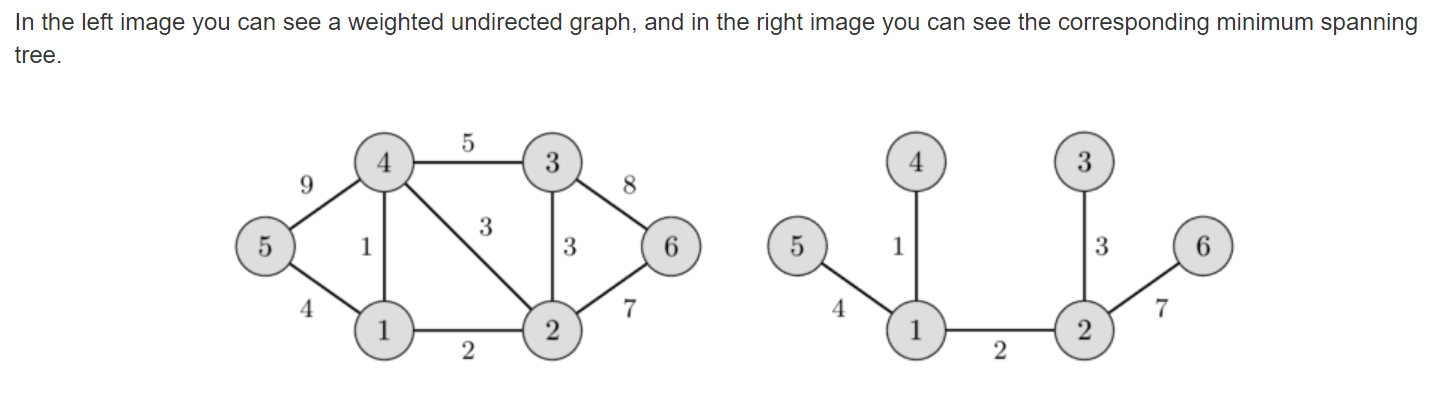
cost[a][b]=cost[b][a]=999;

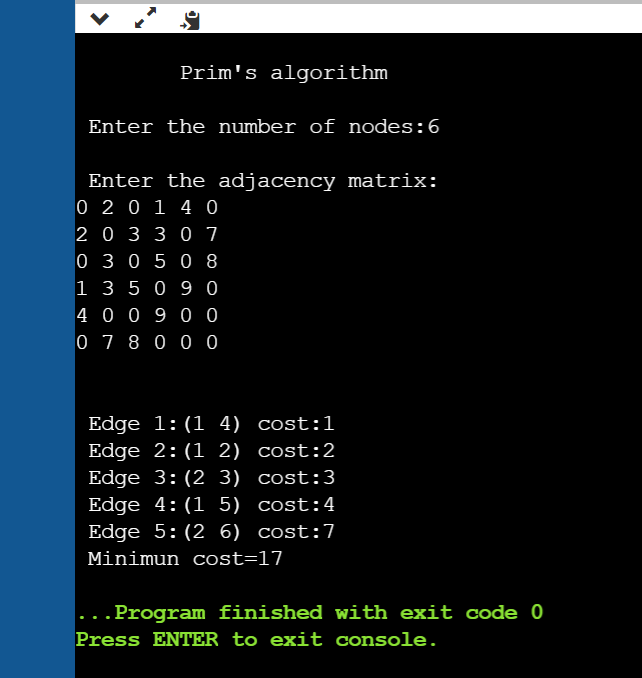
}

printf("\n Minimun cost=%d",mincost);

}

OUTPUT:





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 15: Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.**

CODE:

#include<stdio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,count=1;

int min,mincost=0,cost[10][10],parent[10];

int find(int);

int uni(int,int);

void main()

{

printf("\n\tKruskal's algorithm\n");

printf("\nEnter the number of vertices:");

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("The edges of Minimum Cost Spanning Tree are\n");

while(count < 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 Edge %d:(%d %d) cost:%d",count++,a,b,min);

mincost +=min;

}

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

}

printf("\n\tMinimum cost = %d\n",mincost);

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

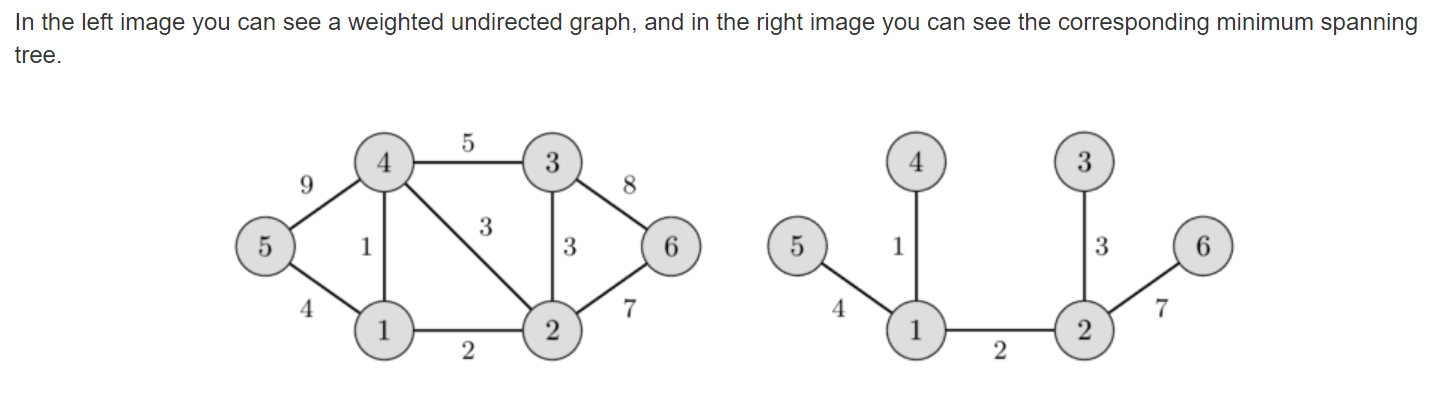
return 1;

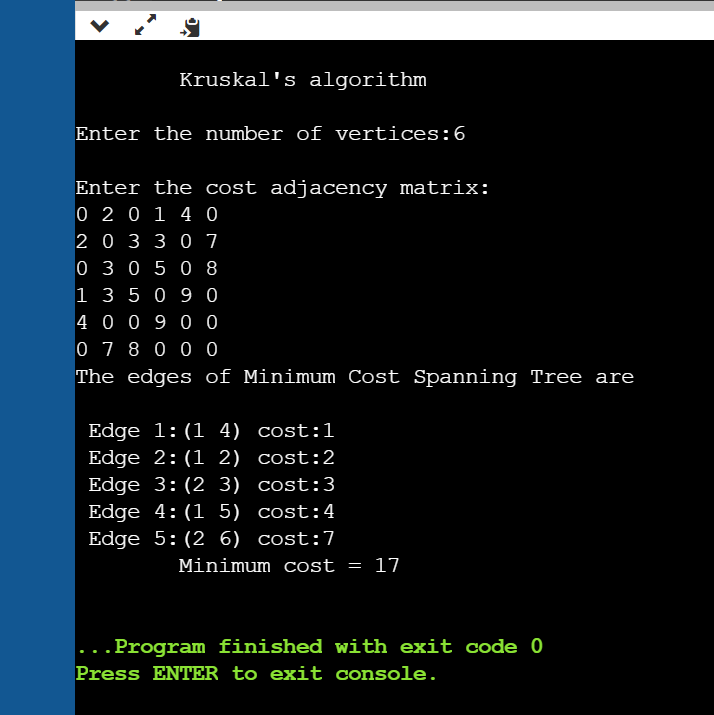
}

return 0;

}

OUTPUT:





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 16: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.**

CODE:

#include<stdio.h>

#define INF 999

#define MAX 10

void dijkstra(int A[MAX][MAX], int n, int start);

void main(){

int A[MAX][MAX], i, j, n, u;

printf("\n\tDijkstra's algorithm\n");

printf("\nEnter the no. of vertices:: ");

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]);

printf("\nEnter the starting node:: ");

scanf("%d", &u);

dijkstra(A,n,u);

}

void dijkstra(int A[MAX][MAX], int n, int start)

{

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindist, next, i,j;

for(i=0;i < n;i++)

for(j=0;j < n;j++)

if(A[i][j]==0)

cost[i][j]=INF;

else

cost[i][j]=A[i][j];

for(i=0;i< n;i++)

{

distance[i]=cost[start][i];

pred[i]=start;

visited[i]=0;

}

distance[start]=0;

visited[start]=1;

count=1;

while(count < n-1){

mindist=INF;

for(i=0;i < n;i++)

if(distance[i] < mindist &&!visited[i])

{

mindist=distance[i];

next=i;

}

visited[next]=1;

for(i=0;i < n;i++)

if(!visited[i])

if(mindist+cost[next][i] < distance[i])

{

distance[i]=mindist+cost[next][i];

pred[i]=next;

}

count++;

}

for(i=0;i < n;i++)

if(i!=start)

{

printf("\nDistance of %d = %d", i, distance[i]);

printf("\nPath = %d", i);

j=i;

do

{

j=pred[j];

printf(" <-%d", j);

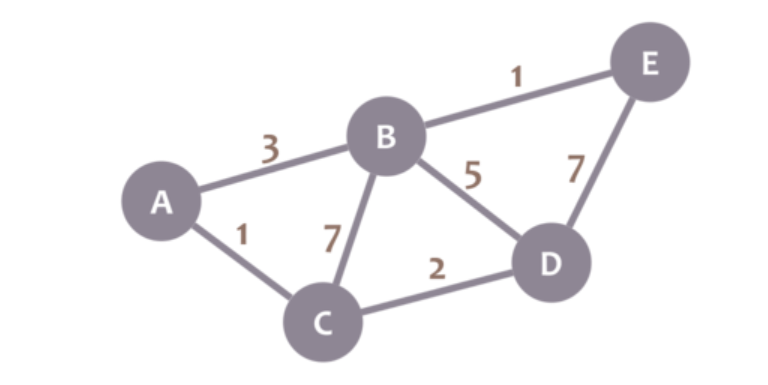
}

while(j!=start);

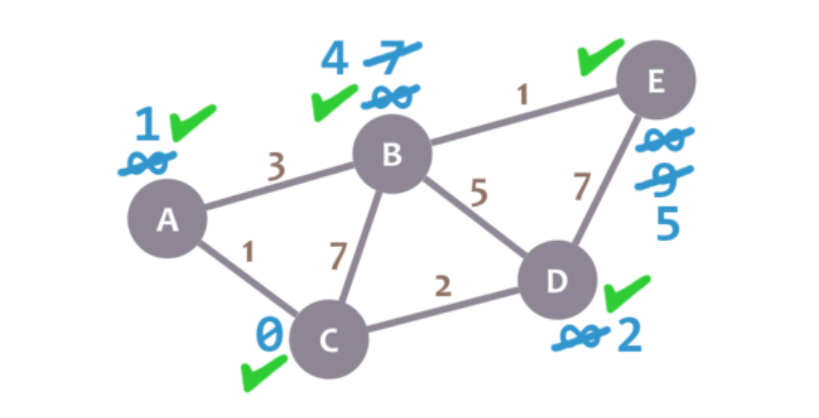
}

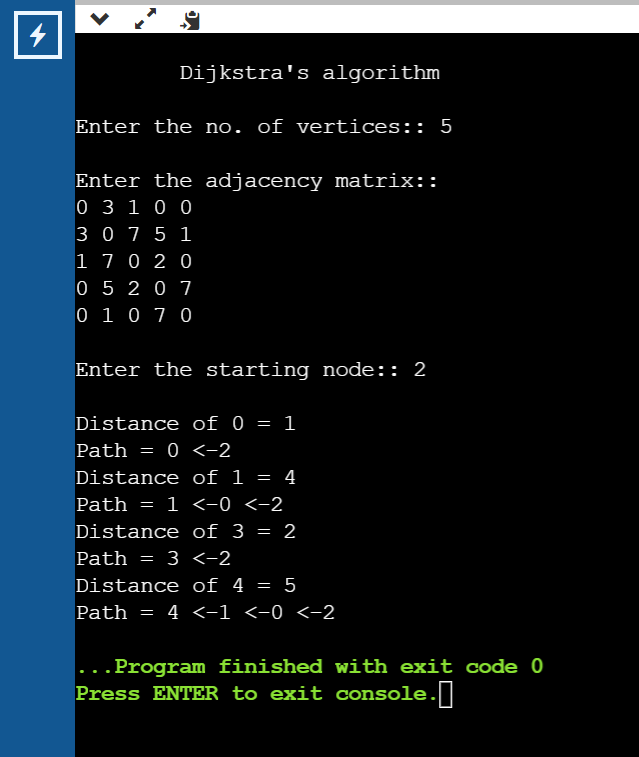
}

OUTPUT:



**Shortest paths starting from Vertex C =**





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 17: Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: 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**

CODE:

#include<stdio.h>

int s[10] , x[10],d ;

void sumofsub ( int , int , int ) ;

void main ()

{

int n , sum = 0 ;

int i ;

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 ) ;

}

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 ) )

{

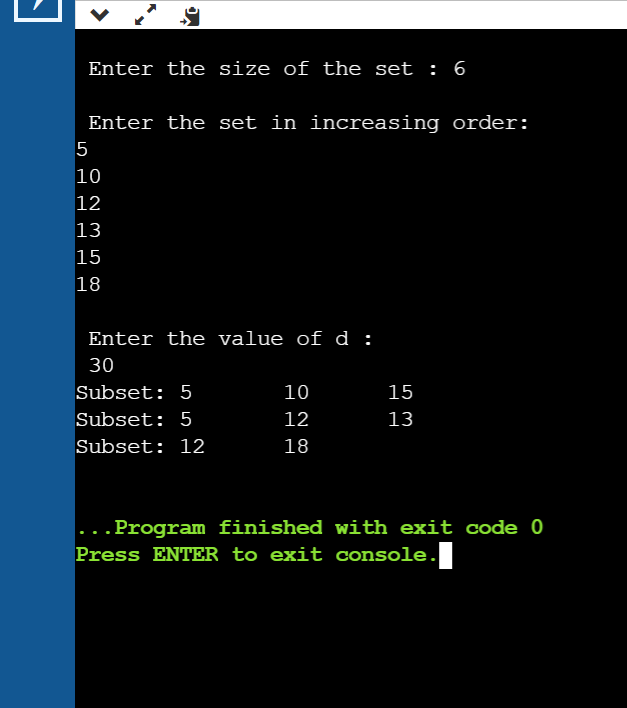
x[k] = 0;

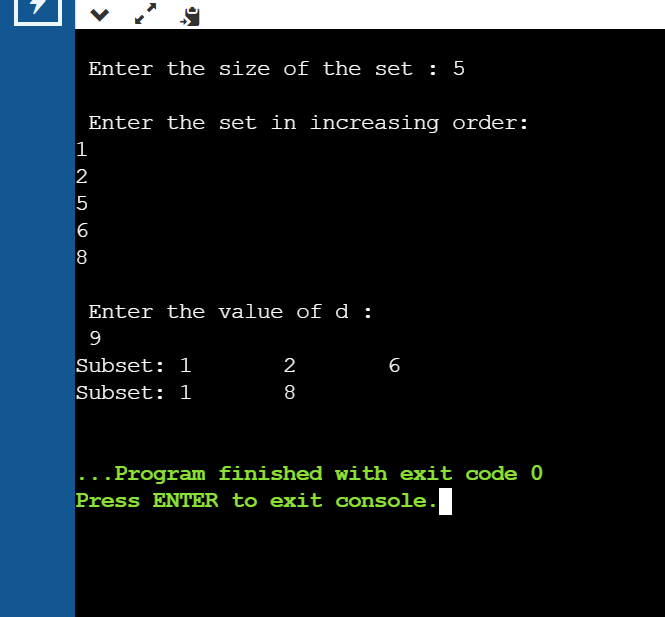
sumofsub ( m , k + 1 , r - s[k] ) ;

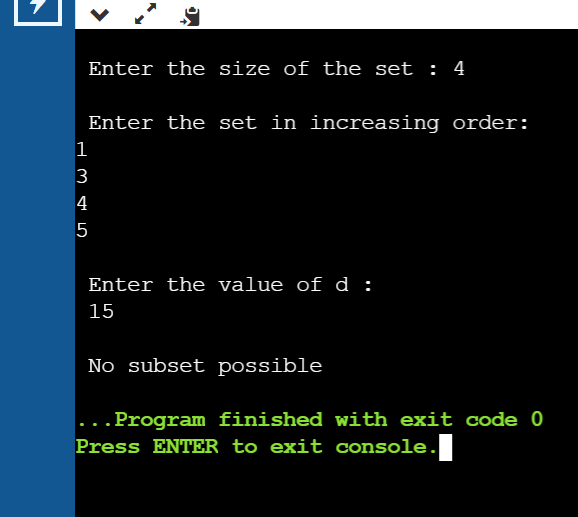
}

}

OUTPUT:







NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 18: Implement “N-Queens Problem” using Backtracking.**

CODE:

#include<stdio.h>

#include <stdlib.h>

#include<math.h>

int board[20],count;

int main()

{

int n,i,j;

void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");

printf("\n\nEnter number of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

void print(int n)

{

int i,j;

printf("\n\nSolution %d:\n\n",++count);

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

printf("\t%d",i);

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

{

printf("\n\n%d",i);

for(j=1;j<=n;++j) //for nxn board

{

if(board[i]==j)

printf("\tQ");

else

printf("\t-");

}

}

}

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

if(board[i]==column)

return 0;

else

if(abs(board[i]-column)==abs(i-row))

return 0;

}

return 1;

}

void queen(int row,int n)

{

int column;

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

{

if(place(row,column))

{

board[row]=column;

if(row==n)

print(n);

else

queen(row+1,n);

}

}

}

OUTPUT:

