**B.M.S.COLLEGE OF ENGINEERING,**

**BANGALORE-19**

**(Autonomous College under VTU)**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Analysis and Design of Algorithms**

**Lab manual**

**NAME: SUPRIYA.M.LINGDAL**

**USN: 1BM20CS415**

**PROGRAM: BACHELOR OF ENGINEERING**

**SEMESTER: IV**

**SESSION: APRIL-JULY 2021**

**COURSE CODE: 19CS4PCADA**

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

**CREDITS: 4**

**PROGRAM 1**

**WRITE A RECURSIVE PROGRAM TO :**

**A)SOLVE TOWERS-OF-HANOI PROBLEM**

**B)TO FIND GCD**

**Recursive program to find GCD:**

#include <stdio.h>

#include <time.h>

int gcd(int a, int b)

{

if (b == 0)

return a;

return gcd(b, a % b);

}

int main()

{

int a,b;

clock\_t start, end;

double time;

printf("Greatest common divisor\n");

printf("Enter two numbers\n");

scanf("%d%d",&a,&b);

start = clock();

printf("GCD of %d and %d is %d \n", a, b, gcd(a, b));

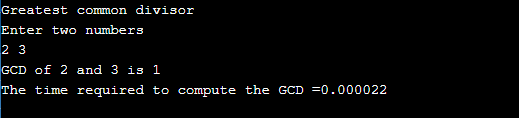
end = clock();

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

printf("The time required to compute the GCD =%f",time);

}

**Output:**



**Recursive program to solve Tower-of-Hanoi Problem**

#include <stdio.h>

#include <time.h>

void TOH(int, char, char, char);

int main()

{

int num;

clock\_t start, end;

double time;

printf("TOWER OF HAN0I PROBLEM\n");

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

scanf("%d", &num);

printf("The sequence of moves involved in the Tower of Hanoi are :\n");

start=clock();

TOH(num, 'A', 'C', 'B');

end = clock();

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

printf("\n\nThe time required to compute the Tower of Hanoi problem is =%f",time);

}

void TOH(int num, char frompeg, char topeg, char auxpeg)

{

if (num == 1)

{

printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);

return;

}

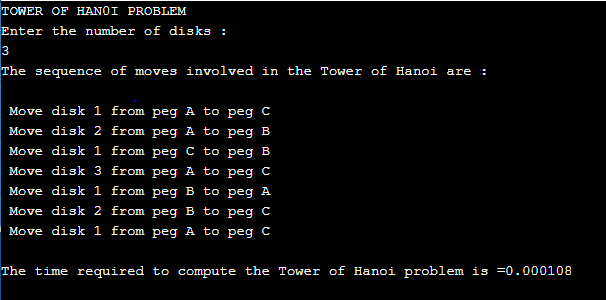
TOH(num - 1, frompeg, auxpeg, topeg);

printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);

TOH(num - 1, auxpeg, topeg, frompeg);

}

**Output:**



**PROGRAM 2**

**Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N**

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int Linear\_Search(int array[10000], int search, int index, int n);

int Binary\_Search(int array[10000], int low,int high, int search);

int main()

{

clock\_t start,end;

double time;

int i,j,a,low, high, mid, n, search, array[10000],choice,loc,res,index;

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

scanf("%d", &n);

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

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

{

array[i] = rand()%100;

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

}

printf("\n Enter the key to be searched: ");

scanf("%d",&search);

while(1){

printf("\nCHOICES\n");

printf("\n 1. Linear search");

printf("\n 2. Binary search");

printf("\n 3. Exit");

printf("\n Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1:

start = clock();

loc = Linear\_Search(array, search, 0, n);

end = clock();

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

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

if (loc != 0)

{

printf("\nElement found in the array at location: %d\n", loc);

}

else

{

printf("\nElement not found in the array!");

}

break;

case 2 : for (i = 0; i < n; ++i)

{

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

{

if (array[i] > array[j])

{

a = array[i];

array[i] = array[j];

array[j] = a;

}

}

}

printf("\nsorted arrray: \n");

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

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

start = clock();

res = Binary\_Search(array, 0, n-1, search);

end = clock();

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

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

if(res == -1){

printf("Element is not found in array");

}

else{

printf("Element is present in the array at location: %d", res);

}

break;

case 3: exit(0);

break;

default: printf("/n Enter valid choice!");

break;

}

}

}

int Linear\_Search(int arr[], int search, int index, int n)

{

int pos = 0;

if(index >= n)

{

return 0;

}

else if (arr[index] == search)

{

pos = index + 1;

return pos;

}

else

{

return Linear\_Search(arr, search, index+1, n);

}

return pos;

}

int Binary\_Search(int arr[], int low, int high, int search)

{

if (high >= low)

{

int mid = low + (high - low)/2;

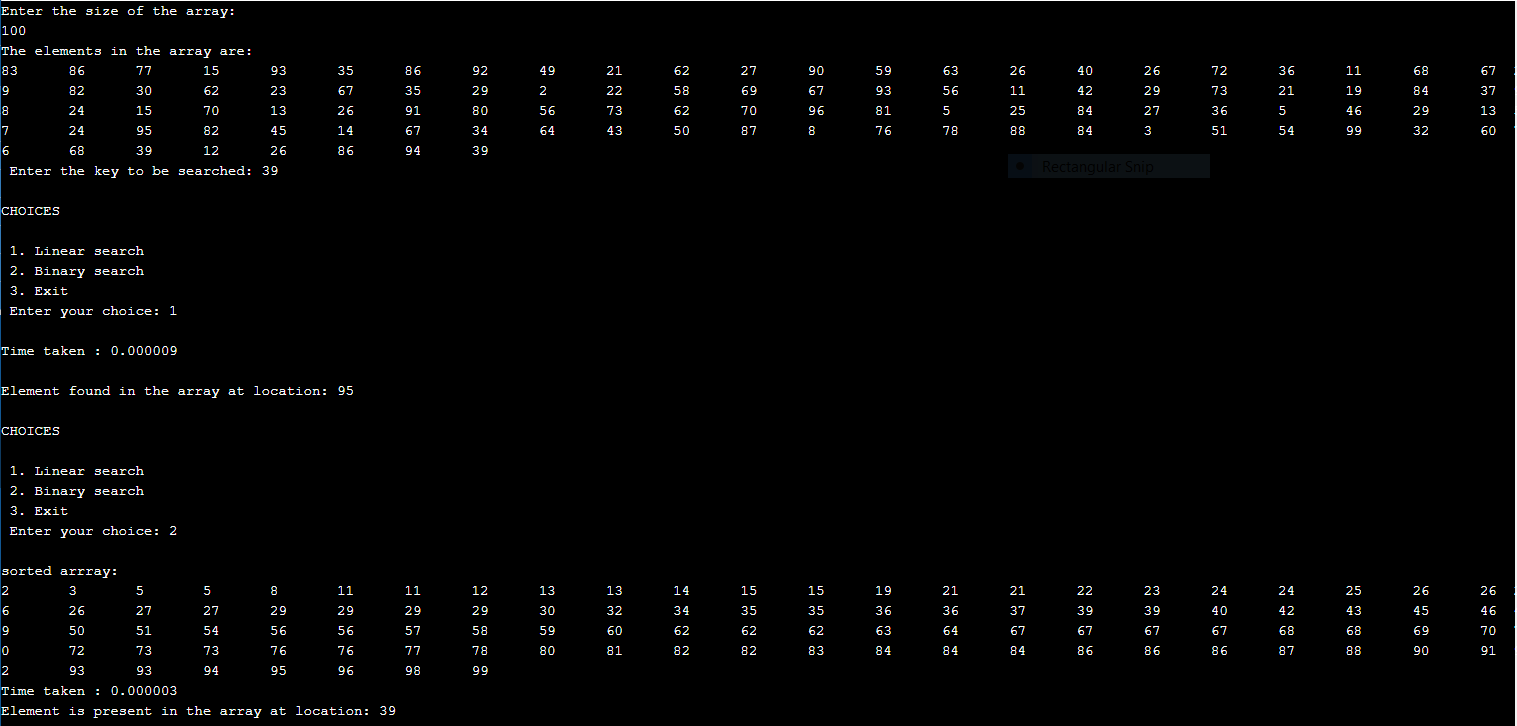
if (arr[mid] == search) return mid;

if (arr[mid] > search) return Binary\_Search(arr, low, mid-1, search);

return Binary\_Search(arr, mid+1, high, search);

} return -1;}

**OUTPUT**



**PROGRAM 3**

**Sort a given set of N integer elements using Selection Sort technique and compute its time taken.Run the program for different values of N and record the time taken to sort.**

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void swap(int arr[], int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

void selectionSort(int arr[], int i, int n)

{

int min = i;

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

{

if (arr[j] < arr[min]) {

min = j;

}

}

swap(arr, min, i);

if (i + 1 < n) {

selectionSort(arr, i + 1, n);

}

}

void printArray(int arr[], int n)

{

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

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

}

}

int main()

{

int i,n,arr[1000];

double time;

clock\_t start,end;

printf("Enter the total number of elements to be sorted:");

scanf("%d",&n);

printf("The elements to be sorted are:\n");

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

{

arr[i] = rand()%100;

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

}

printf("\nThe sorted array is:\n");

start=clock();

selectionSort(arr, 0, n);

printArray(arr, n);

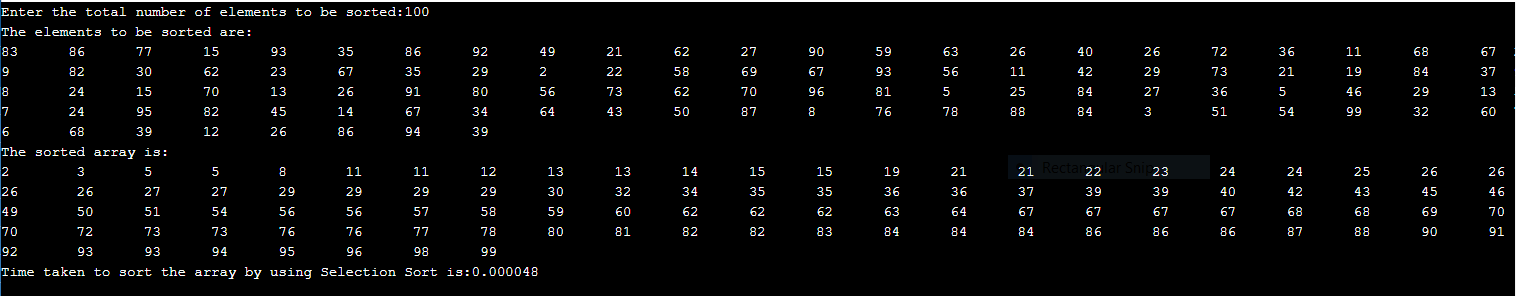
end=clock();

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

printf("\nTime taken to sort the array by using Selection Sort is:%lf\n",time);

}

**Output:**



**Program 4(A)**

**Write program to do the following:**

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

**Program:**

#include<stdio.h>

#include<conio.h>

#include<time.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;

clock\_t start, end;

double time;

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);

start = clock();

bfs(v);

end = clock();

printf("\n The node which are reachable are:\n");

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

if(visited[i])

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

else

printf("\n Bfs is not possible");

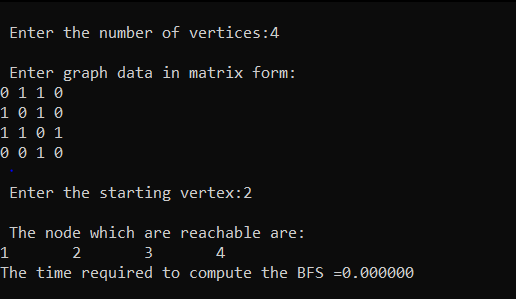
time= ((double) (end - start)) / CLOCKS\_PER\_SEC;

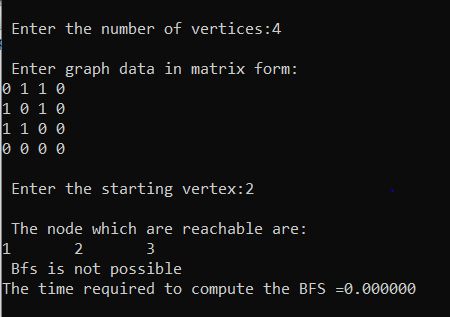
printf("\nThe time required to compute the BFS =%f",time);

getch();

}

**Output:**





**PROGRAM 4(B)**

**Write program to do the following**

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

**Program:**

#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;

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)

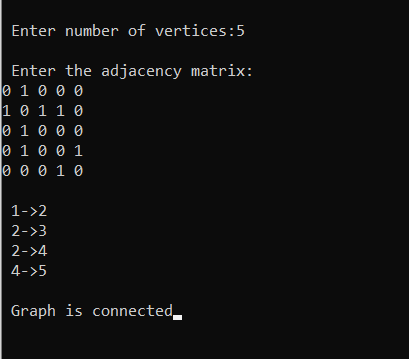
printf("\n Graph is connected"); else

printf("\n Graph is not connected");

getch();

}

**Output:**



**PROGRAM 5**

**Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.**

**CODE:**

#include <stdio.h>

#include<stdlib.h>

#include<time.h>

void InsertionSort(int arr[], int n)

{

if (n <= 1)

return;

InsertionSort( arr, n-1 );

int nth = arr[n-1];

int j = n-2;

while (j >= 0 && arr[j] > nth)

{

arr[j+1] = arr[j];

j--;

}

arr[j+1] = nth;

}

void printArray(int arr[], int n)

{

int i;

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

{

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

}

}

int main()

{

int i,n,arr[1000];

double time;

clock\_t start,end;

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

scanf("%d",&n);

printf("The elements to be sorted are:\n");

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

{

arr[i] = rand()%100;

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

}

printf("\nThe sorted array is:\n");

start=clock();

InsertionSort(arr,n);

printArray(arr, n);

end=clock();

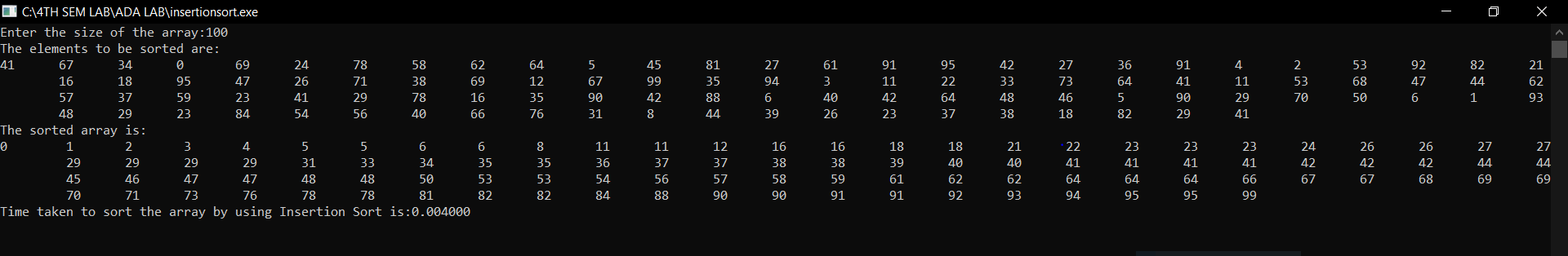
time = ((double)(end - start))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort the array by using Insertion Sort is:%lf\n",time);

getch();

}

**OUTPUT:**



**PROGRAM 6**

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

**CODE:**

#include<stdio.h>

#include<time.h>

void findindegree(int a[10][10],int indegree[10],int n)

{

int i,j,sum;

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

{

sum=0;

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

sum=sum+a[i][j];

indegree[j]=sum;

}

}

void topological(int n,int a[10][10])

{

int i,k,u,v,top,t[100],stack[20],indegree[20];

k=1;

top=-1;

findindegree(a,indegree,n);

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

{

if(indegree[i]==0)

stack[++top]=i;

}

while(top!=-1){

u=stack[top--];

t[k++]=u;

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

{

if(a[u][v]==1)

{

indegree[v]--;

if(indegree[v]==0)

stack[++top]=v;

}}}

printf("Topological sequence is\n");

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

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

}

void main()

{

int a[10][10],i,j,n;

clock\_t start,end;

double time;

printf("Enter the number of nodes:");

scanf("%d",&n);

printf("Enter the adjacency matrix:\n");

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

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

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

}

start=clock();

topological(n,a);

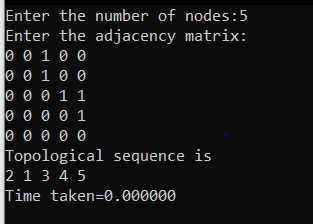
end=clock();

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

printf("\nTime taken=%1f",time);

getch();}

**Output:**



**PROGRAM 7**

**Implement Johnson Trotter algorithm to generate permutations**

**Code:**

#include <stdio.h>

#include<time.h>

int N, i;

int p[10000], q[10000];

int direct[10000];

void Move(int x, int d)

{

int z;

printf("\n");

z = p[q[x]+d];

p[q[x]] = z;

p[q[x]+d] = x;

q[z] = q[x];

q[x] = q[x]+d;

}

void Permutation(int n)

{

int i;

if (n > N){

int i;

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

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

}

else

{

Permutation( n+1 );

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

{

Move(n,direct[n]);

Permutation(n+1);

}

direct[n] = -direct[n];

}

}

int main ()

{

clock\_t start,end;

double time;

printf("Enter the value of N:");

scanf("%d", &N);

printf("\n");

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

{

direct[i] = -1; p[i] = i;

q[i] = i;

}

printf("The permutations generated are:\n");

start = clock();

Permutation(1);

end = clock();

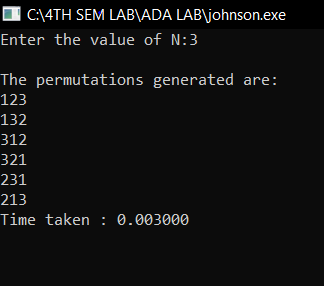
time = ((double)(end - start))/CLOCKS\_PER\_SEC;

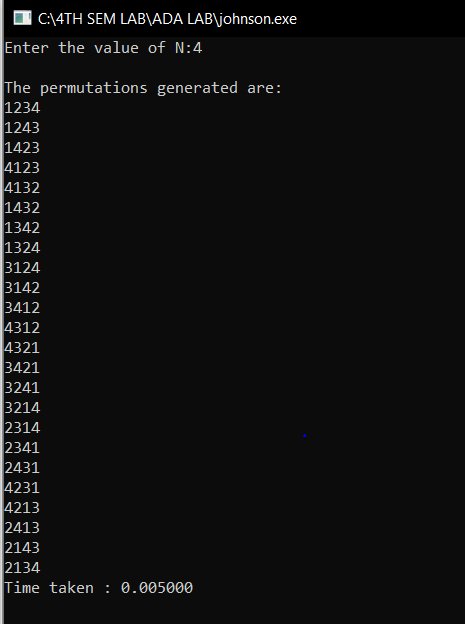
printf("\nTime taken : %lf\n",time);

getch();

}

**Output:**





**PROGRAM 8**

**Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.**

**CODE:**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l + (r - l) / 2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

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

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

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int p = m - l + 1;

int q = r - m;

int L[p], R[q];

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

L[i] = arr[l + i];

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

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < p && j < q)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < p) {

arr[k] = L[i];

i++;

k++;

}

while (j < q) {

arr[k] = R[j];

j++;

k++;

}

}

int main()

{

int i,n,arr[1000];

double time;

clock\_t start,end;

printf("Enter the total number of elements to be sorted:");

scanf("%d",&n);

printf("The elements to be sorted are:\n");

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

{

arr[i] = rand()%100;

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

}

printf("\nThe sorted array is:\n");

start=clock();

mergeSort(arr, 0, n- 1);

printArray(arr,n-1);

end=clock();

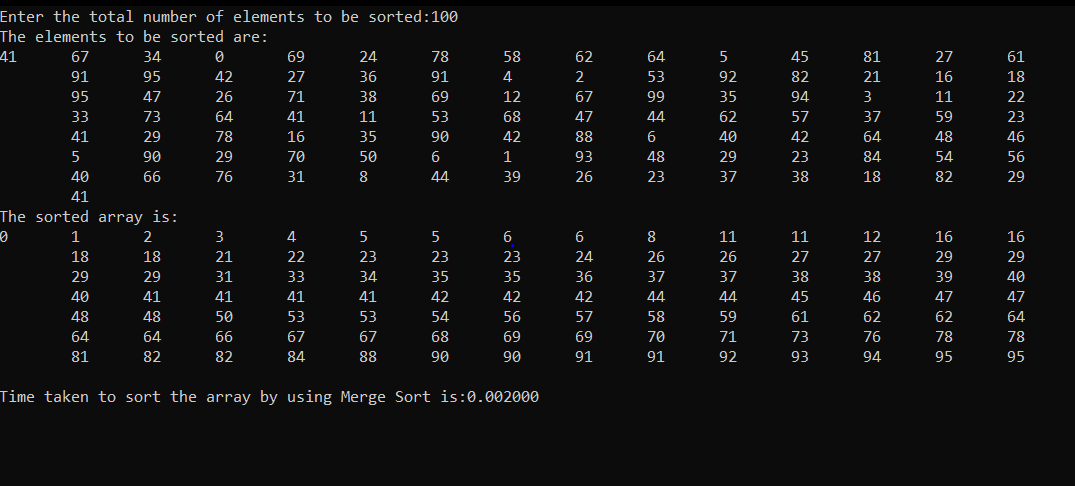
time = ((double)(end - start))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort the array by using Merge Sort is:%lf\n",time);

getch();

}

**OUTPUT:**



**PROGRAM 9**

**Sort a given set of N integer elements using Quick Sort technique and compute its time taken**

**CODE**

#include<stdio.h>

#include<time.h>

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

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int partition(int a[],int lb,int ub)

{

int pivot=a[lb],i=lb,j=ub;

while(i<j)

{

while(a[i]<=pivot)

i++;

while(a[j]>pivot)

j--;

if(i<j)

swap(&a[i],&a[j]);

}

swap(&a[lb],&a[j]);

return j;

}

void sort(int a[],int lb,int ub)

{

if(lb<ub)

{

int loc=partition(a,lb,ub);

sort(a,lb,loc-1);

sort(a,loc+1,ub);

}

}

void printArray(int arr[],int n)

{

int i;

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

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

}

int main()

{

int i,n,arr[1000];

double time;

clock\_t start,end;

printf("Enter the total number of elements to be sorted:");

scanf("%d",&n);

printf("The elements to be sorted are:\n");

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

{

arr[i] = rand()%100;

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

}

printf("\nThe sorted array is:\n");

start=clock();

sort(arr, 0, n-1);

printArray(arr, n);

end=clock();

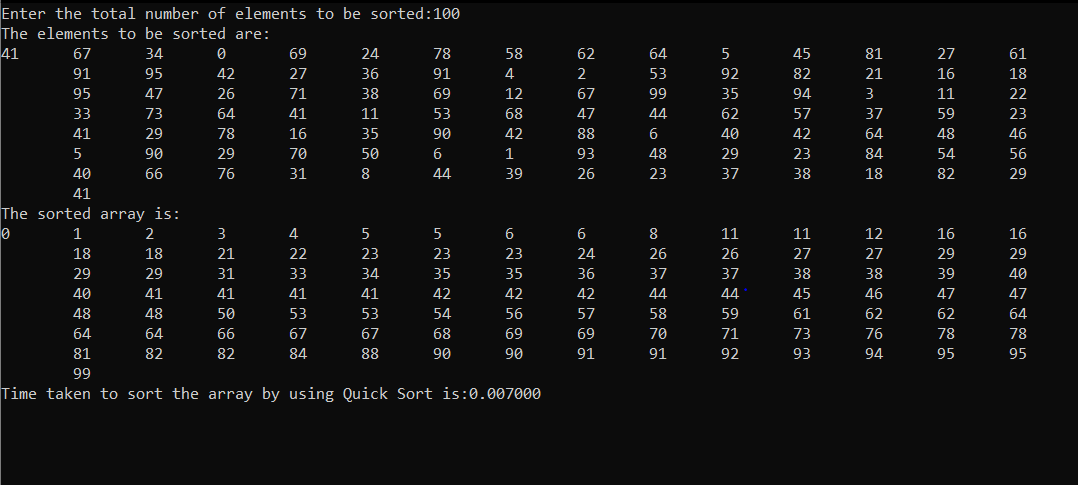
time = ((double)(end - start))/CLOCKS\_PER\_SEC;

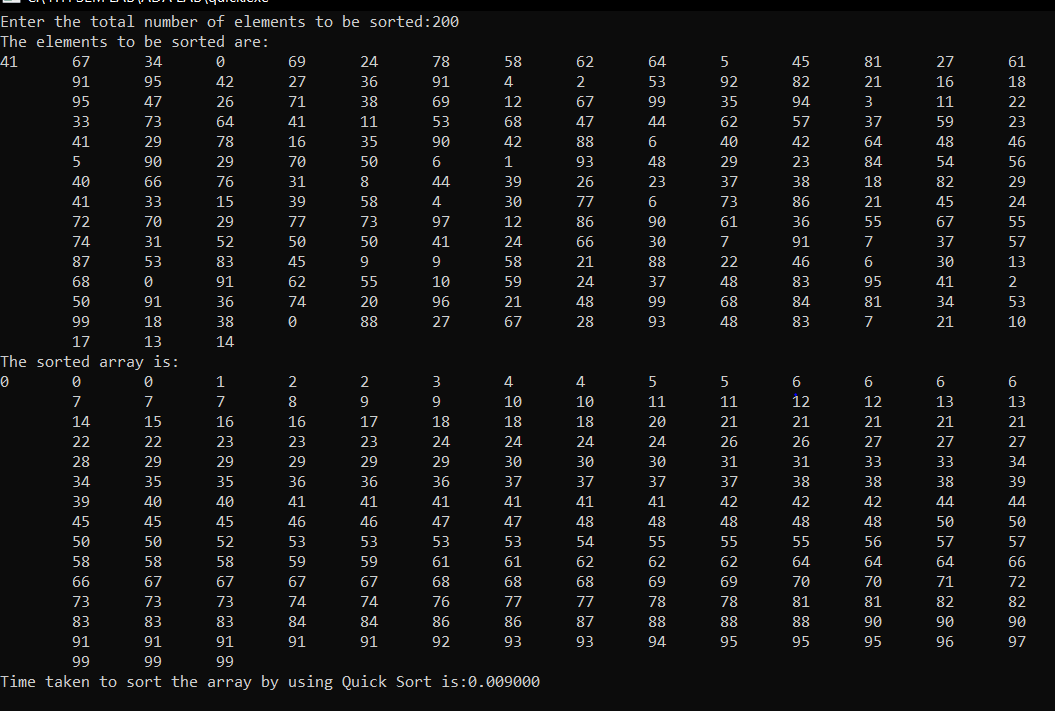
printf("\nTime taken to sort the array by using Quick Sort is:%lf\n",time);

getch();

}

**OUTPUT:**





**PROGRAM 10**

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

**CODE:**

#include <stdio.h>

#include <time.h>

void main()

{

int heap[20], no, i, j, c, root, temp;

clock\_t start, end;

double time;

printf("\n Enter no of elements :");

scanf("%d", &no);

printf("\n Enter the nos : ");

for(i=0;i<no;i++)

{

heap[i] = rand()%100;

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

}

start = clock();

for (i = 1; i < no; i++)

{

c = i;

do

{

root = (c - 1) / 2;

if (heap[root] < heap[c]) /\* to create MAX heap array \*/

{

temp = heap[root];

heap[root] = heap[c];

heap[c] = temp;

}

c = root;

} while (c != 0);

}

end = clock();

printf("\n\nHeap array : ");

for (i = 0; i < no; i++)

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

for (j = no - 1; j >= 0; j--)

{

temp = heap[0];

heap[0] = heap[j]; /\* swap max element with rightmost leaf element \*/

heap[j] = temp;

root = 0;

do

{

c = 2 \* root + 1; /\* left node of root element \*/

if ((heap[c] < heap[c + 1]) && c < j-1)

c++;

if (heap[root]<heap[c] && c<j) /\* again rearrange to max heap array \*/

{

temp = heap[root];

heap[root] = heap[c];

heap[c] = temp;

}

root = c;

} while (c < j);

}

printf("\n\n The sorted array is : ");

for (i = 0; i < no; i++)

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

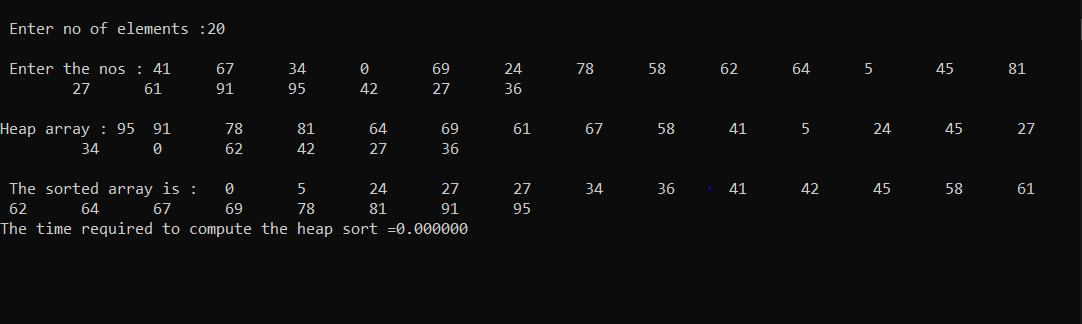
time= ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("\nThe time required to compute the heap sort =%f",time);

getch();

}

**Output:**



**PROGRAM 11**

**Implement Warshall’s algorithm using dynamic programming**

**Code:**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int max(int,int);

void warshal(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]);

}

int max(int a,int b)

{

;

if(a>b)

return(a); else

return(b);

}

void main()

{

int p[10][10]= {999},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 starting and ending vertex of the 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");

}

warshal(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");

}

getch();

}

**Graph:**

**Adjacency matrix: Transitive closure:**

0 0 1 0 0 0 1 0

1 0 0 1 1 1 1 1

0 0 0 0 0 0 0 0

0 1 0 0 1 1 1 1

2→1 , 1→3 2→3

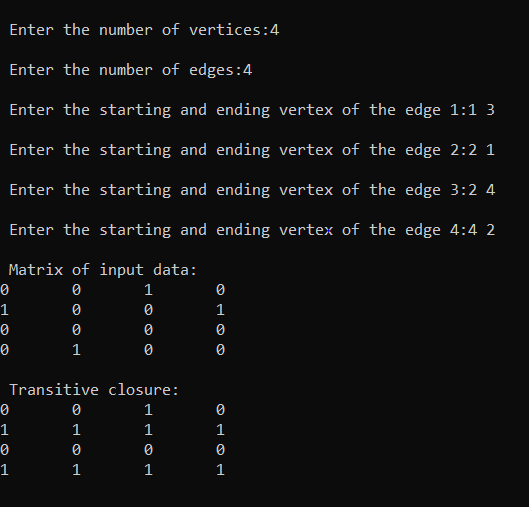
2→4 , 4→2 2→2

4→2 , 2→1 4→11

4→2 , 2→1 , 1→3 4→3

4→2 , 2→4 4→4

**Output:**



**PROGRAM-12**

**Implement 0/1 Knapsack problem using dynamic programming**

**Code:**

#include<stdio.h>

#define MAX 50

int p[MAX],w[MAX],n;

int knapsack(int,int);

int max(int,int);

void main()

{

int m,i,optsoln;

printf("1/0 Knapsack problem\n");

printf("Enter no. of objects: ");

scanf("%d",&n);

printf("\nEnter the weights and profits:\n");

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

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

printf("\nEnter the knapsack capacity:");

scanf("%d",&m);

optsoln=knapsack(1,m);

printf("\nThe optimal soluntion is:%d",optsoln);

getch();

}

int knapsack(int i,int m)

{

if(i==n)

return (w[n]>m) ? 0 : p[n];

if(w[i]>m)

return knapsack(i+1,m);

return max(knapsack(i+1,m),knapsack(i+1,m-w[i])+p[i]);

}

int max(int a,int b)

{

if(a>b)

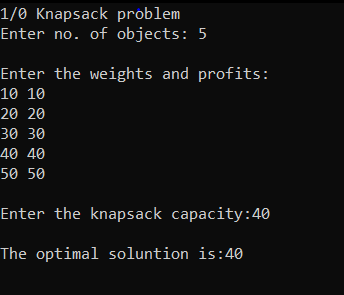
return a;

else

return b;

}

**Output:**



**PROGRAM 13**

**Implement All Pair Shortest paths problem using Floyd’s algorithm.**

**Code:**

#include<stdio.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:");

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 start and 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();

}

**Graph:**

8

3

5

7

2

2

1

**Adjacency matrix:**

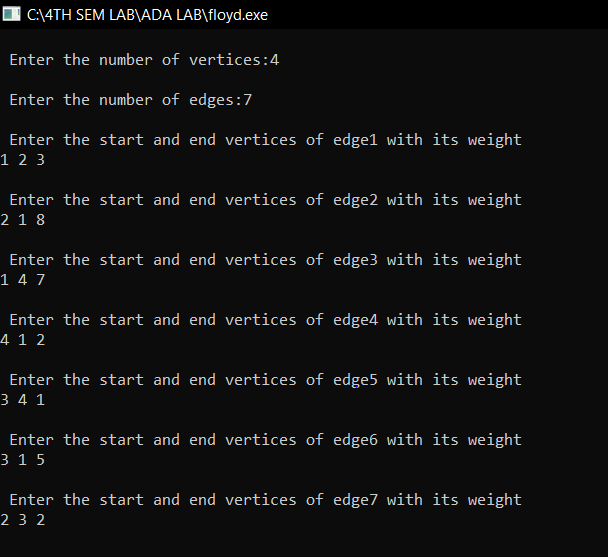
0 3 999 7

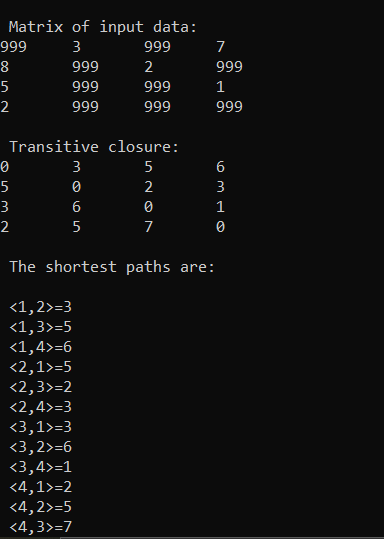
8 0 2 999

5 999 0 1

2 999 999 0

**Output:**





**PROGRAM 14**

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

**Code:**

#include<stdio.h>

#include<conio.h>

#include<time.h>

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

int visited[10]= {0},min,mincost=0,cost[10][10];

void main()

{

double time;

clock\_t start,end;

printf("PRIM'S ALGORITHM\n");

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

scanf("%d",&n);

printf("\n Enter the weighted 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");

start=clock();

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;

}

end=clock();

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

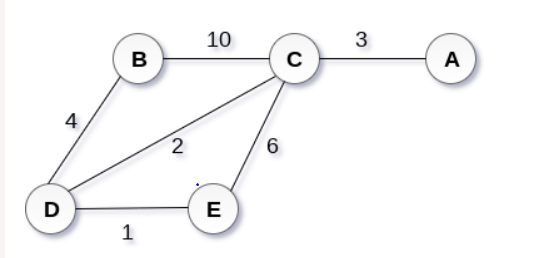
printf("\n Minimun cost=%d\n",mincost);

printf("Time taken to find the minimum cost using the Prims algorithm is:%lf\n",time);

getch();

}

**Graph:**



**Adjacency matrix**

A B C D E

A 0 0 3 0 0

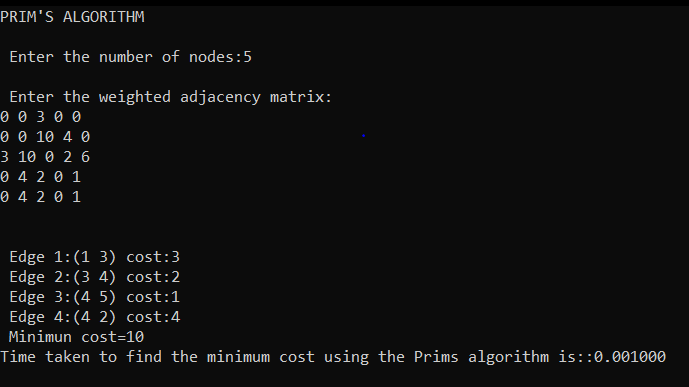
B 0 0 10 4 0

C 3 10 0 2 6

D 0 4 2 0 1

E 0 4 2 0 1

**Output:**



**PROGRAM 15**

**Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.**

**Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<time.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()

{

double time;

clock\_t start,end;

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

printf("\nEnter the no. 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");

start=clock();

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("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

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

}

end=clock();

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

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

printf("Time taken to find the minimum cost using the kruskal's algorithm is:%lf\n",time);

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;

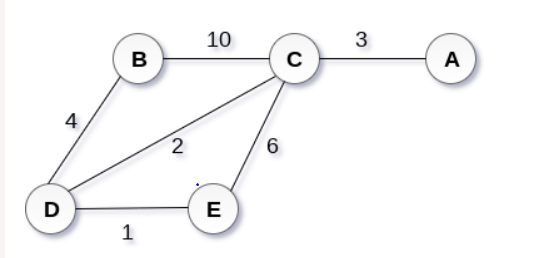
return 1;

}

return 0;

}

**Graph:**



**Adjacency matrix**

A B C D E

A 0 0 3 0 0

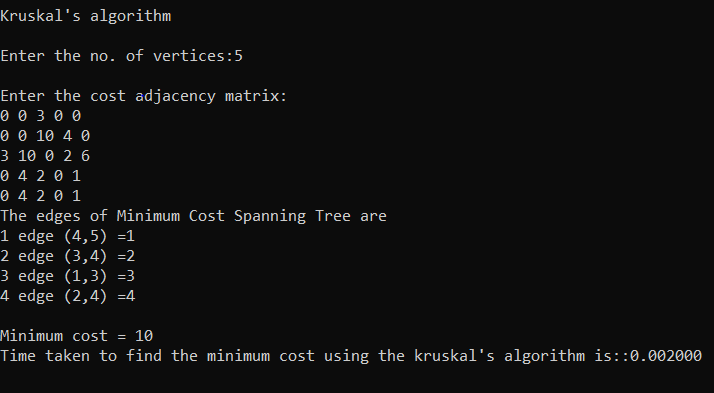
B 0 0 10 4 0

C 3 10 0 2 6

D 0 4 2 0 1

E 0 4 2 0 1

**Output:**



**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 INFINITY 999

#define MAX 10

void dijikstra(int G[MAX][MAX], int n, int startnode);

void main()

{

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

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",&G[i][j]);

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

scanf("%d",&u);

dijikstra(G,n,u);

getch();

}

void dijikstra(int G[MAX][MAX], int n, int startnode)

{

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

int visited[MAX], count, mindistance, nextnode, i,j;

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

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

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

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

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

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1){

mindistance=INFINITY;

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

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

{

mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i]){

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;}

count++;

}

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

if(i!=startnode)

{

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

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

j=i;

do

{

j=pred[j];

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

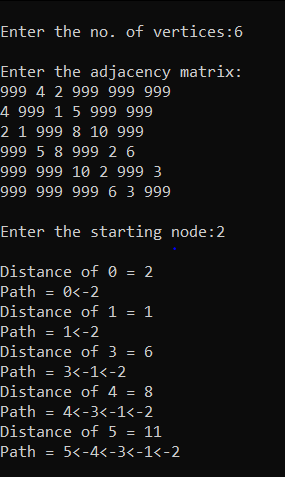
}

while(j!=startnode);

}

}

**Output:**



**Code2:**

#include<stdio.h>

void dij(int,int [20][20],int [20],int [20],int);

void main()

{

int i,j,n,visited[20],source,cost[20][20],d[20];

printf("Enter no. of vertices: ");

scanf("%d",&n);

printf("Enter the cost adjacency matrix\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);

dij(source,cost,visited,d,n);

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

if(i!=source)

printf("\nShortest path from %d to %d is %d",source,i,d[i]);

getch();

}

void dij(int source,int cost[20][20],int visited[20],int d[20],int n)

{

int i,j,min,u,w;

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

{

visited[i]=0;

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

}

visited[source]=1;

d[source]=0;

for(j=2;j<=n;j++)

{

min=999;

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

{

if(!visited[i])

{

if(d[i]<min)

{

min=d[i];

u=i;

}

}

} //for i

visited[u]=1;

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

{

if(cost[u][w]!=999 && visited[w]==0)

{

if(d[w]>cost[u][w]+d[u])

d[w]=cost[u][w]+d[u];

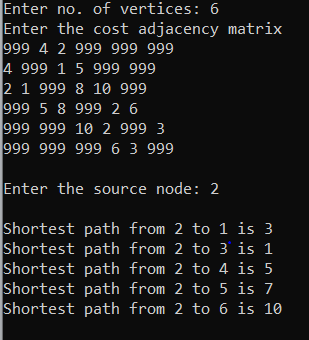
}

} //for w

} // for j

}

**OUTPUT:**



**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. 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.**

**Code:**

#include<stdio.h>

void subset(int,int,int);

int x[10],w[10],d,count=0;

void main()

{

int i,n,sum=0;

printf("Enter the no. of elements: ");

scanf("%d",&n);

printf("\nEnter the elements in ascending order:\n");

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

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

printf("\nEnter the sum: ");

scanf("%d",&d);

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

sum=sum+w[i];

if(sum<d)

{

printf("No solution\n");

getch();

return;

}

subset(0,0,sum);

if(count==0)

{

printf("No solution\n");

getch();

return;

}

getch();

}

void subset(int cs,int k,int r)

{

int i;

x[k]=1;

if(cs+w[k]==d)

{

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

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

if(x[i]==1)

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

}

else

if(cs+w[k]+w[k+1]<=d)

subset(cs+w[k],k+1,r-w[k]);

if(cs+r-w[k]>=d && cs+w[k]<=d)

{

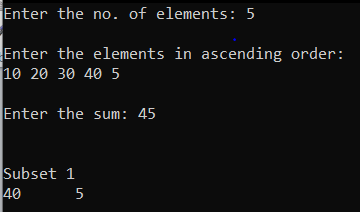
x[k]=0;

subset(cs,k+1,r-w[k]);

}

}

**Output:**



**PROGRAM-18**

**Implement “N-Queens Problem” using Backtracking**

**Code:**

#include<stdio.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);

getch();

}

//function for printing the solution

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"); //queen at i,j position

else

printf("\t-"); //empty slot

}

}}

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:**

