**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**KANKSHIT PRAKASH BHOJANE (1BM20CS068)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



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

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

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

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **KANKSHIT PRAKASH BHOJANE(1BM20CS0668),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

**Dr.Nagarathna N**          **Dr. Jyothi S Nayak** Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | Write a recursive program to Solve  **a)** Towers-of-Hanoi problem **b)** To find GCD |  |
| **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. |  |
| **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. |  |
| **4** | Write program to do the following:  **a)** Print all the nodes reachable from a given starting node in a digraph using BFS method.  **b)** Check whether a given graph is connected or not using DFS method. |  |
| **5** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. |  |
| **6** | Write program to obtain the Topological ordering of vertices in a given digraph. |  |
| **7** | Implement Johnson Trotter algorithm to generate permutations. |  |
| **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. |  |
| **9** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. |  |
| **10** | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. |  |
| **11** | Implement Warshall’s algorithm using dynamic programming |  |
| **12** | Implement 0/1 Knapsack problem using dynamic programming. |  |
| **13** | Implement All Pair Shortest paths problem using Floyd’s algorithm. |  |
| **14** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |  |
| **15** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm. |  |
| **16** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. |  |
| **17** | 1. 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. |  |
| **18** | Implement “N-Queens Problem” using Backtracking. |  |

**Course Outcome**

|  |  |
| --- | --- |
| **CO1** | Ability to **analyze** time complexity of Recursive and Non-Recursive algorithms using asymptotic notations. |
| **CO2** | Ability to **design** efficient algorithms using various design techniques. |
| **CO3** | Ability to **apply** the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| **CO4** | Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find time efficiency. |

**1 . Write a recursive program to**

**a. Solve Towers-of-Hanoi problem**

**a) #include<stdio.h>**

**#include<conio.h>**

**#include<math.h>**

**void hanoi(int x, char from, char to, char aux)**

**{**

**if(x==1)**

**printf("Move Disk From %c to %c\n",from,to);**

**else**

**{**

**hanoi(x-1,from,aux,to);**

**printf("Move Disk From %c to %c\n",from,to);**

**hanoi(x-1,aux,to,from);**

**} }**

**int main()**

**{**

**int disk;**

**int moves;**

**printf("Enter the number of disks:");**

**scanf("%d",&disk);**

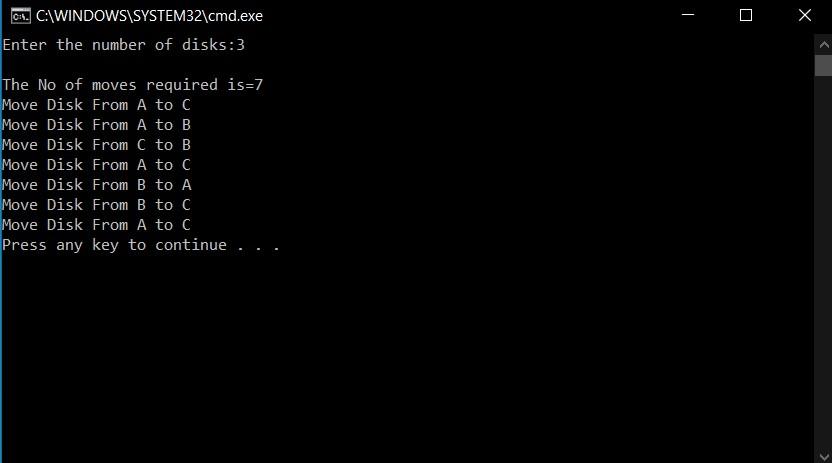
**moves=pow(2,disk)-1;**

**printf("\nThe No of moves required is=%d \n",moves);**

**hanoi(disk,'A','C','B');**

**return 0; }**

**OUTPUT-**



**b)To find GCD**

**#include <stdio.h>**

**int hcf(int x, int y);**

**int main()**

**{**

**int x;**

**int y;**

**printf("Enter two positive integers: ");**

**scanf("%d %d", &x, &y);**

**printf("G.C.D of %d and %d is %d.", x, y, hcf(x,y));**

**return 0;**

**}**

**int hcf(int x, int y)**

**{**

**if (y != 0)**

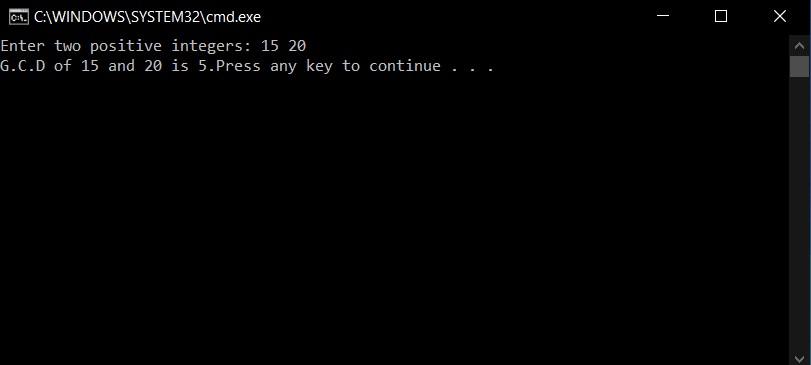
**return hcf(y,x%y);**

**else**

**return x;**

**}**

**OUTPUT -**



**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.**

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h> /\* To recognise exit function when compiling with gcc\*/**

**int bin\_srch(int [],int,int,int);**

**int lin\_srch(int [],int,int,int);**

**int n,a[10000];**

**int main()**

**{**

**int ch,key,search\_status,temp;**

**clock\_t end,start;**

**unsigned long int i, j;**

**while(1)**

**{**

**printf("\n1: Binary search\t 2: Linear search\t 3: Exit\n");**

**printf("\nEnter your choice:\t");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**n=1000;**

**while(n<=5000)**

**{**

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

**{**

**//a[i]=random(1000);**

**a[i]=i; //Inserting numbers in Ascending order**

**}**

**key=a[n-1]; //Last element of the array**

**start=clock();**

**search\_status=bin\_srch(a,0,n-1,key);**

**if(search\_status==-1)**

**printf("\nKey Not Found");**

**else**

**printf("\n Key found at position %d",search\_status);**

**//Dummy loop to create delay**

**for(j=0;j<500000;j++){ temp=38/600;}**

**end=clock();**

**printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS\_PER\_SEC));**

**n=n+1000;**

**}**

**break;**

**case 2:**

**n=1000;**

**while(n<=5000)**

**{**

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

**{**

**//a[i]=random(10000);**

**a[i]=i;**

**}**

**key=a[n-1]; //Last element of the aray**

**start=clock();**

**search\_status=lin\_srch(a,0,n-1,key);**

**if(search\_status==-1)**

**printf("\nKey Not Found");**

**else**

**printf("\n Key found at position %d",search\_status);**

**//Dummy loop to create delay**

**for(j=0;j<500000;j++){ temp=38/600;}**

**end=clock();**

**printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS\_PER\_SEC));**

**n=n+1000;**

**}**

**break;**

**default:**

**exit(0);**

**}**

**getchar();**

**}**

**}**

**int bin\_srch(int a[],int low,int high,int key) {**

**int mid;**

**if(low>high)**

**return -1;**

**mid=(low+high)/2;**

**if(key==a[mid])**

**return mid;**

**if(key<a[mid])**

**return bin\_srch(a,low,mid-1,key);**

**else**

**return bin\_srch(a,mid+1,high,key);**

**}**

**int lin\_srch(int a[],int i,int high,int key) {**

**if(i>high)**

**return -1;**

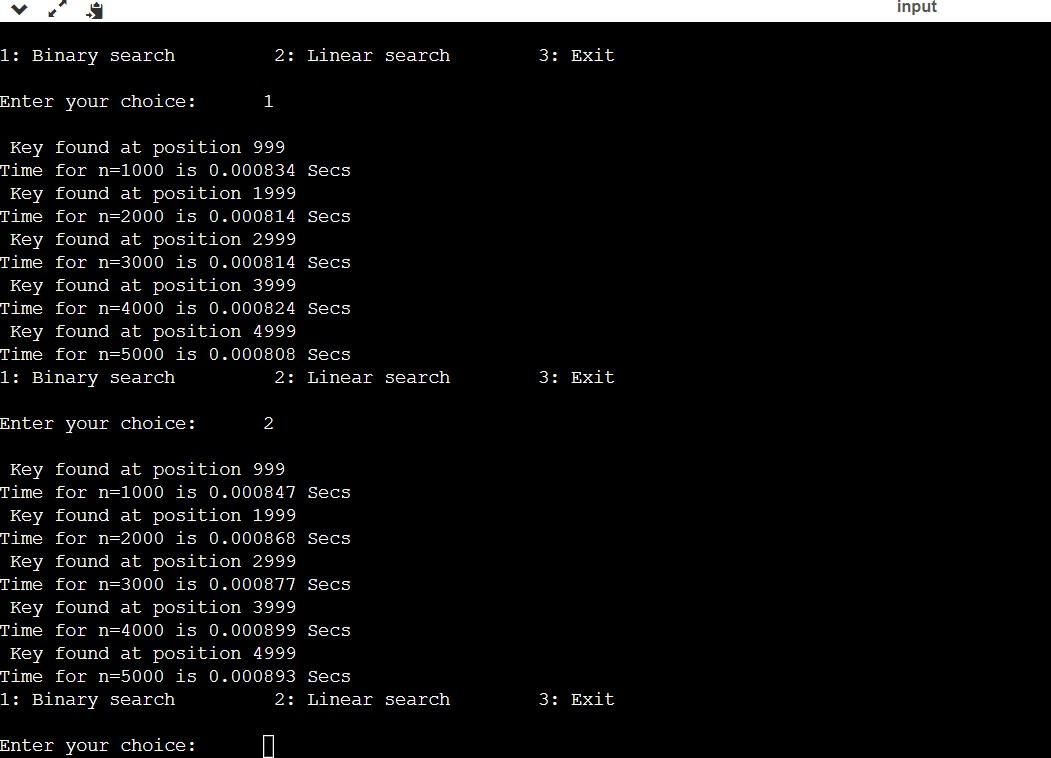
**if(key==a[i])**

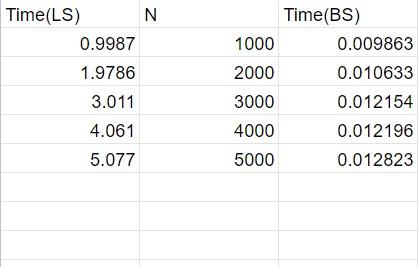
**return i;**

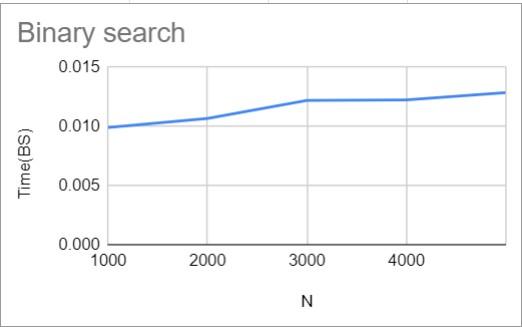
**else**

**return lin\_srch(a,i+1,high,key); }**

**OUTPUT -**

****





**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.**

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h>**

**/\* To recognise exit function when compiling with gcc\*/**

**void selsort(int n,int a[]);**

**int main()**

**{**

**int a[15000],n,i,j,ch,temp;**

**clock\_t start,end;**

**while(1)**

**{**

**printf("\n1:For manual entry of N value and array elements");**

**printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");**

**printf("\n3:To exit");**

**printf("\nEnter your choice:");**

**scanf("%d", &ch);**

**switch(ch)**

**{**

**case 1: printf("\nEnter the number of elements: ");**

**scanf("%d",&n);**

**printf("\nEnter array elements: ");**

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

**{**

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

**}**

**start=clock();**

**selsort(n,a);**

**end=clock();**

**printf("\nSorted array is: ");**

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

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

**printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));**

**break;**

**case 2:**

**n=500;**

**while(n<=14500) {**

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

**{**

**//a[i]=random(1000);**

**a[i]=n-i;**

**}**

**start=clock();**

**selsort(n,a);**

**//Dummy loop to create delay**

**for(j=0;j<500000;j++){ temp=38/600;}**

**end=clock();**

**printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));**

**n=n+1000;**

**}**

**break;**

**case 3: exit(0);**

**}**

**return 0;**

**}**

**}**

**void selsort(int n,int a[])**

**{**

**int i,j,t,small,pos;**

**for(i=0;i<n-1;i++)**

**{**

**pos=i;**

**small=a[i];**

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

**{ if(a[j]<small)**

**{ small=a[j];**

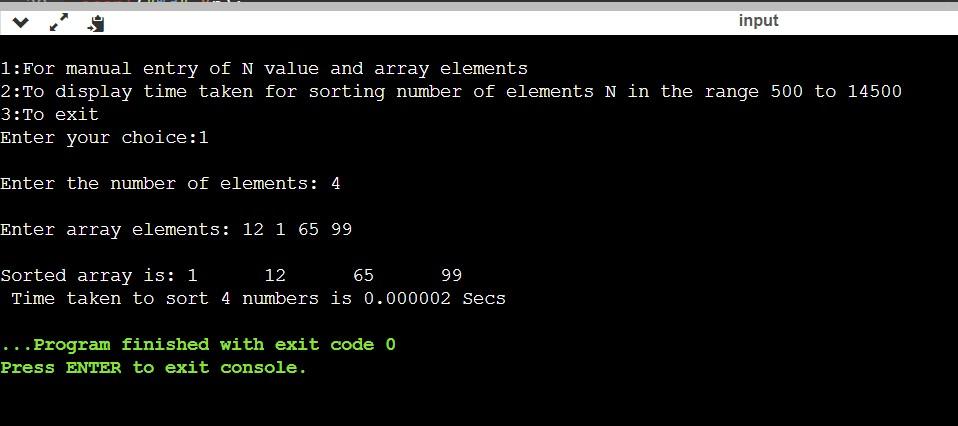
**pos=j; } }**

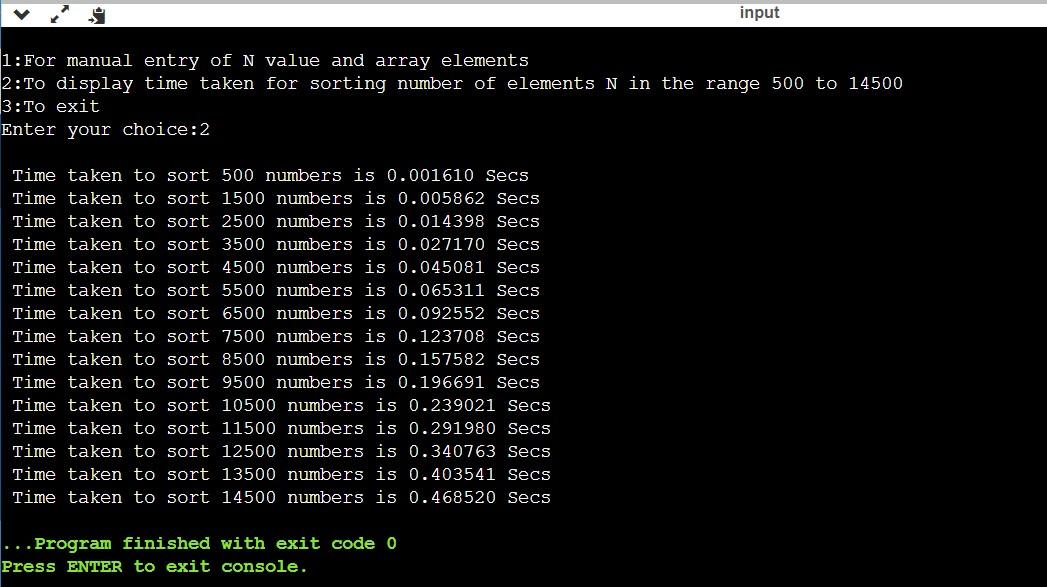
**t=a[i];**

**a[i]=a[pos];**

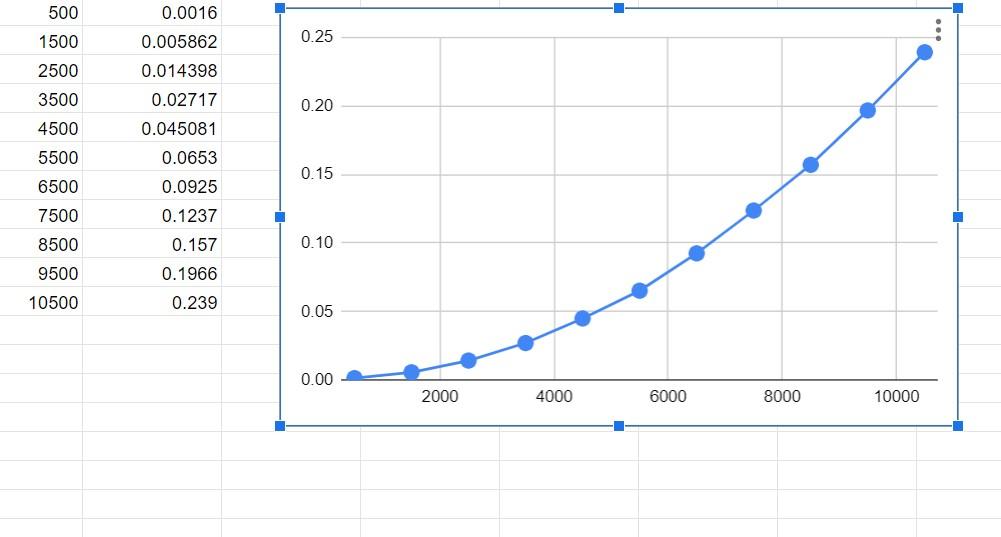
**a[pos]=t; } }**

**OUTPUT-**





N T



**4.Write program to do the following: a) Print all the nodes reachable from a given starting node in a digraph using BFS method.**

**#include<stdio.h>**

**#include<conio.h>**

**int a[10][10],n;**

**void bfs(int);**

**int main()**

**{**

**int i,j,src;**

**printf("\nenter the no of nodes:\t");**

**scanf("%d",&n);**

**printf("\nenter the adjacency matrix:\n");**

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

**{**

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

**{**

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

**} }**

**printf("\nenter the source node:\t");**

**scanf("%d",&src);**

**bfs(src);**

**return 0;**

**}**

**void bfs(int src) {**

**int q[10],f=0,r=-1,vis[10],i,j;**

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

**vis[j]=0;**

**vis[src]=1;**

**r=r+1;**

**q[r]=src;**

**while(f<=r) {**

**i=q[f];**

**f=f+1;**

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

**{**

**if(a[i][j]==1&&vis[j]!=1) {**

**vis[j]=1;**

**r=r+1;**

**q[r]=j;**

**}**

**}**

**}**

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

**if(vis[j]!=1)**

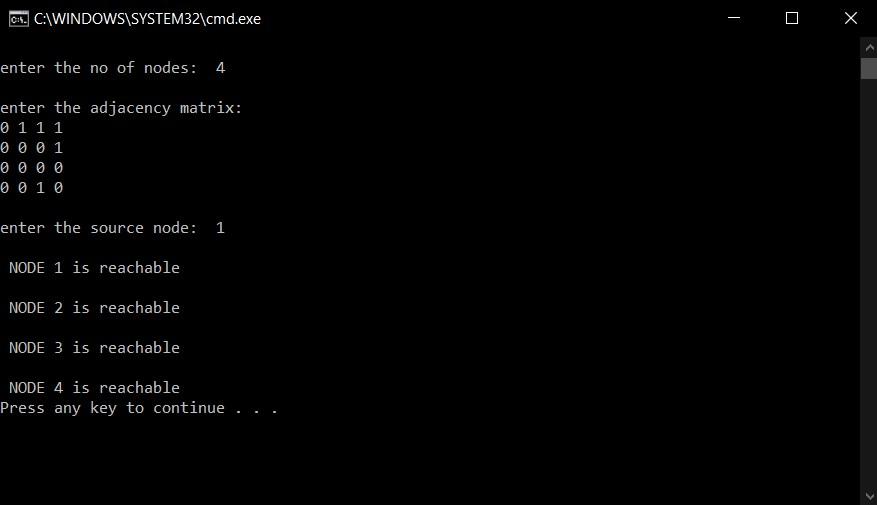
**printf("\n NODE %d is not reachable\n",j);**

**else**

**{ printf("\n NODE %d is reachable\n",j);**

**} } }**

**OUTPUT -**



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

**#include<stdio.h>**

**#include<conio.h>**

**int a[10][10],n,vis[10];**

**int dfs(int);**

**int main()**

**{**

**int i,j,src,ans;**

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

**{**

**vis[j]=0;**

**}**

**printf("\nenter the no of nodes:\t");**

**scanf("%d",&n);**

**printf("\nenter the adjacency matrix:\n");**

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

**{**

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

**{**

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

**} }**

**printf("\nenter the source node:\t");**

**scanf("%d",&src);**

**ans=dfs(src);**

**if(ans==1)**

**{**

**printf("\ngraph is connected\n");**

**}**

**else**

**{ printf("\ngragh is not connected\n"); }**

**return 0;**

**}**

**int dfs(int src)**

**{**

**int j;**

**vis[src]=1;**

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

**{**

**if(a[src][j]==1&&vis[j]!=1)**

**{**

**dfs(j);**

**} }**

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

**{**

**if(vis[j]!=1)**

**{**

**return 0;**

**} }**

**return 1; }**

**OUTPUT-**



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

**#include <math.h>**

**#include <stdio.h>**

**#include <time.h>**

**void delay(){**

**long n;**

**for(n=0;n<1000;n++){**

**int a = 10/10;**

**}**

**}**

**void insertionSort(int arr[], int n)**

**{**

**int i, val, j;**

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

**val = arr[i];**

**j = i - 1;**

**while (j >= 0 && arr[j] > val) {**

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

**j --;**

**delay();**

**}**

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

**}**

**}**

**int main()**

**{ int arr[1500],n=100,i;**

**double start,end;**

**while(n<=1200){**

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

**arr[i]=i;**

**}**

**start = clock();**

**insertionSort(arr, n);**

**end=clock();**

**printf("n=%d time= %f \n",n,(end-start)/CLOCKS\_PER\_SEC);**

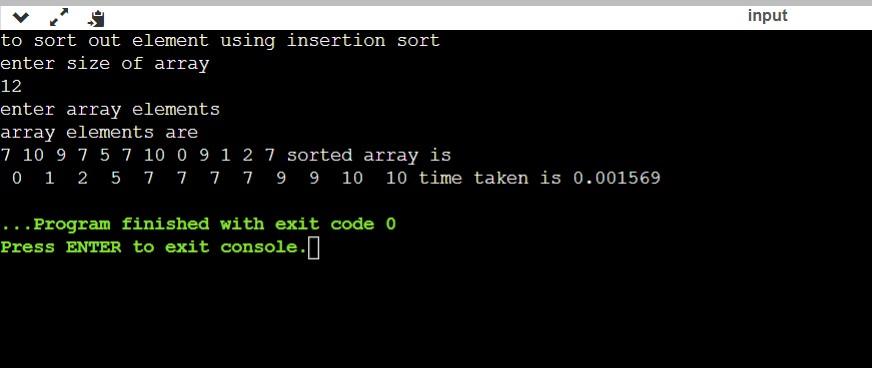
**n=n+100;**

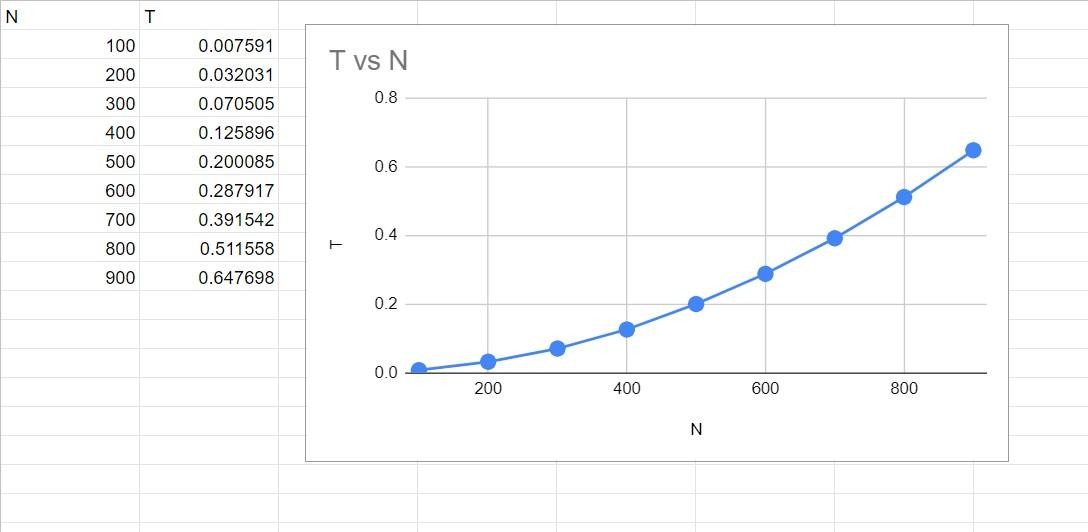
**}**

**return 0;**

**}**

**OUTPUT-**

****

****

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

**#include<stdio.h>**

**#include<conio.h>**

**void source\_removal(int n, int a[10][10])**

**{**

**int i,j,k,u,v,top,s[10],t[10],indeg[10],sum;**

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

**{**

**sum=0;**

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

**{**

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

**}**

**indeg[i]=sum;**

**}**

**top=-1;**

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

**{**

**if(indeg[i]==0)**

**{**

**s[++top]=i;**

**}**

**}**

**k=0;**

**while(top!=-1)**

**{**

**u=s[top--];**

**t[k++]=u;**

**for(v=0;v<n;v++)**

**{**

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

**{**

**indeg[v]=indeg[v]-1;**

**if(indeg[v]==0)**

**s[++top]=v;**

**}**

**}**

**}**

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

**{**

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

**}**

**}**

**int main()**

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

**printf("Enter number 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", &a[i][j]);**

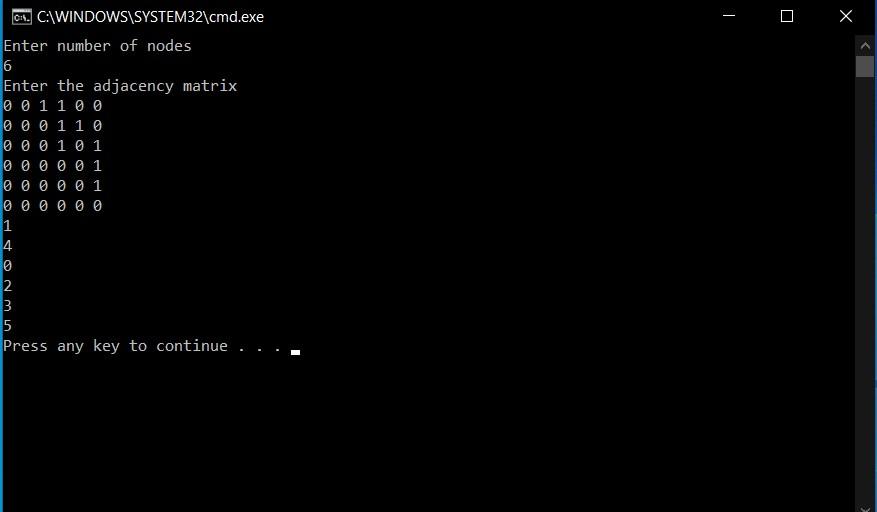
**} }**

**source\_removal(n,a);**

**return 0;**

**}**

**OUTPUT-**



**7.Implement Johnson Trotter algorithm to generate permutations**

**#include <stdio.h>**

**#include <stdlib.h>**

**int flag = 0;**

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

**{**

**int t = \*a;**

**\*a = \*b;**

**\*b = t;**

**}**

**int search(int arr[],int num,int mobile)**

**{**

**int g;**

**for(g=0;g<num;g++)**

**{**

**if(arr[g] == mobile)**

**{**

**return g+1;**

**}**

**else**

**{**

**flag++;**

**}**

**}**

**return -1;**

**}**

**int find\_Moblie(int arr[],int d[],int num)**

**{**

**int mobile = 0;**

**int mobile\_p = 0;**

**int i;**

**for(i=0;i<num;i++)**

**{**

**if((d[arr[i]-1] == 0) && i != 0)**

**{**

**if(arr[i]>arr[i-1] && arr[i]>mobile\_p)**

**{**

**mobile = arr[i];**

**mobile\_p = mobile;**

**}**

**else**

**{**

**flag++ ; }**

**}**

**else if((d[arr[i]-1] == 1) & i != num-1)**

**{**

**if(arr[i]>arr[i+1] && arr[i]>mobile\_p)**

**{**

**mobile = arr[i];**

**mobile\_p = mobile;**

**}**

**else**

**{**

**flag++;**

**}**

**}**

**else**

**{**

**flag++;**

**}**

**}**

**if((mobile\_p == 0) && (mobile == 0))**

**return 0;**

**else**

**return mobile;**

**}**

**void permutations(int arr[],int d[],int num)**

**{**

**int i;**

**int mobile = find\_Moblie(arr,d,num);**

**int pos = search(arr,num,mobile);**

**if(d[arr[pos-1]-1]==0)**

**swap(&arr[pos-1],&arr[pos-2]);**

**else**

**swap(&arr[pos-1],&arr[pos]);**

**for(int i=0;i<num;i++)**

**{**

**if(arr[i] > mobile)**

**{**

**if(d[arr[i]-1]==0)**

**d[arr[i]-1] = 1;**

**else**

**d[arr[i]-1] = 0;**

**}**

**}**

**for(i=0;i<num;i++)**

**{**

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

**}**

**}**

**int factorial(int k)**

**{**

**int f = 1;**

**int i = 0;**

**for(i=1;i<k+1;i++)**

**{**

**f = f\*i;**

**}**

**return f;**

**}**

**int main()**

**{**

**int num = 0;**

**int i;**

**int j;**

**int z = 0;**

**printf("Johnson trotter algorithm to find all permutations of given numbers \n");**

**printf("Enter the number\n");**

**scanf("%d",&num);**

**int arr[num],d[num];**

**z = factorial(num);**

**printf("The total permutations are %d",z);**

**printf("\nAll possible permutations are: \n");**

**for(i=0;i<num;i++)**

**{**

**d[i] = 0;**

**arr[i] = i+1;**

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

**}**

**printf("\n");**

**for(j=1;j<z;j++)**

**{**

**permutations(arr,d,num);**

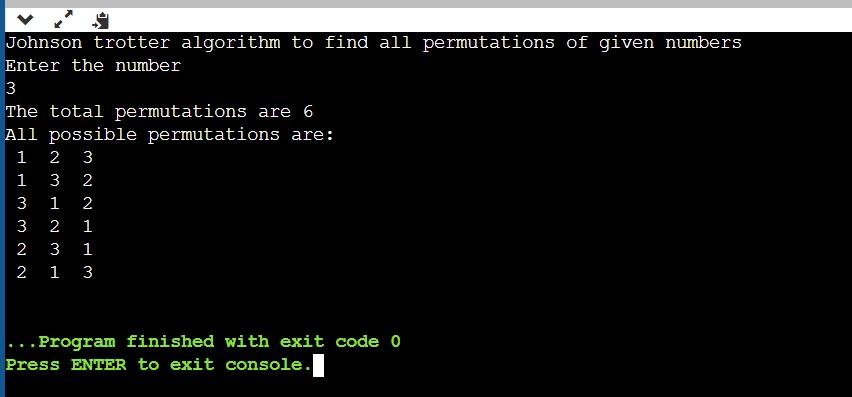
**printf("\n");**

**}**

**return 0;**

**}**

**OUTPUT -**

****

**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.**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**void mergesort(int a[],int i,int j);**

**void merge(int a[],int i1,int j1,int i2,int j2);**

**int main()**

**{**

**clock\_t start,end;**

**int a[3000],n,i;**

**printf("Enter no of elements:");**

**scanf("%d",&n);**

**printf("Enter array elements:");**

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

**a[i] = rand()%1000;**

**start = clock();**

**mergesort(a,0,n-1);**

**end = clock();**

**printf("\nSorted array is :");**

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

**printf("%d ",a[i]);**

**printf("\nSeconds taken %lf",(double)(end-start)/CLOCKS\_PER\_SEC);**

**return 0;**

**}**

**void mergesort(int a[],int i,int j)**

**{**

**int mid;**

**if(i<j)**

**{**

**mid=(i+j)/2;**

**mergesort(a,i,mid);**

**mergesort(a,mid+1,j);**

**merge(a,i,mid,mid+1,j);**

**}**

**}**

**void merge(int a[],int i1,int j1,int i2,int j2)**

**{**

**int temp[3000];**

**int i,j,k;**

**i=i1;**

**j=i2;**

**k=0;**

**while(i<=j1 && j<=j2)**

**{for(int j=0;j<100000;j++);**

**if(a[i]<a[j])**

**temp[k++]=a[i++];**

**else**

**temp[k++]=a[j++];**

**}**

**while(i<=j1)**

**temp[k++]=a[i++];**

**while(j<=j2)**

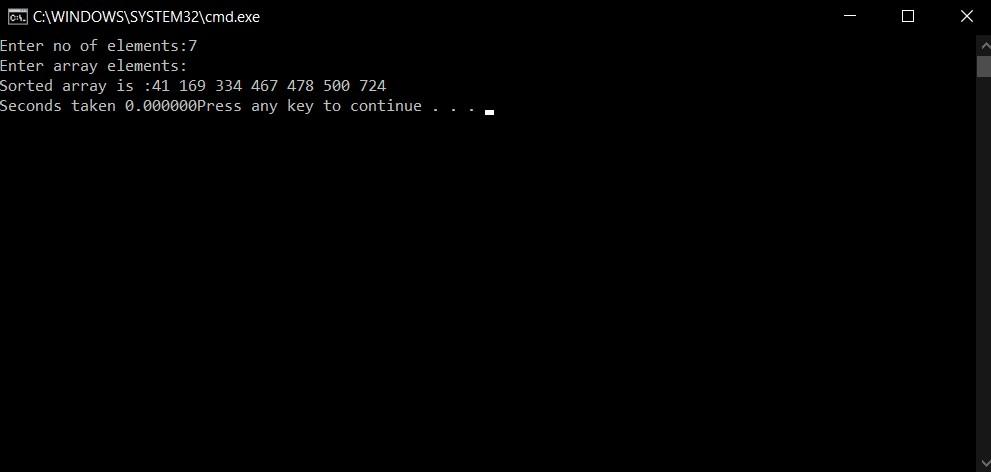
**temp[k++]=a[j++];**

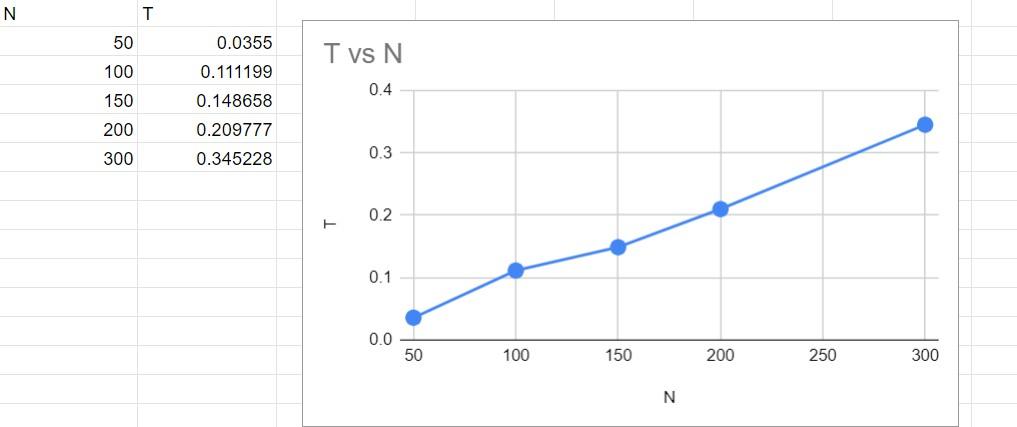
**for(i=i1,j=0;i<=j2;i++,j++)**

**a[i]=temp[j];**

**}**

**OUTPUT-**





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

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h>**

**void quicksort(int number[5000],int first,int last){**

**int i, j, pivot, temp;**

**if(first<last){**

**pivot=first;**

**i=first;**

**j=last;**

**while(i<j){**

**for(int x=0;x<100000;x++);**

**while(number[i]<=number[pivot]&&i<last)**

**i++;**

**while(number[j]>number[pivot])**

**j--;**

**if(i<j){**

**temp=number[i];**

**number[i]=number[j];**

**number[j]=temp;**

**}**

**}**

**temp=number[pivot];**

**number[pivot]=number[j];**

**number[j]=temp;**

**quicksort(number,first,j-1);**

**quicksort(number,j+1,last);**

**}**

**}**

**int main(){**

**clock\_t start,end;**

**int i, count, number[5000];**

**printf("No. of elements: ");**

**scanf("%d",&count);**

**printf("Enter %d elements: ", count);**

**for(i=0;i<count;i++)**

**number[i] = rand()%1000;**

**start = clock();**

**quicksort(number,0,count-1);**

**end = clock();**

**printf("Order of Sorted elements: ");**

**for(i=0;i<count;i++)**

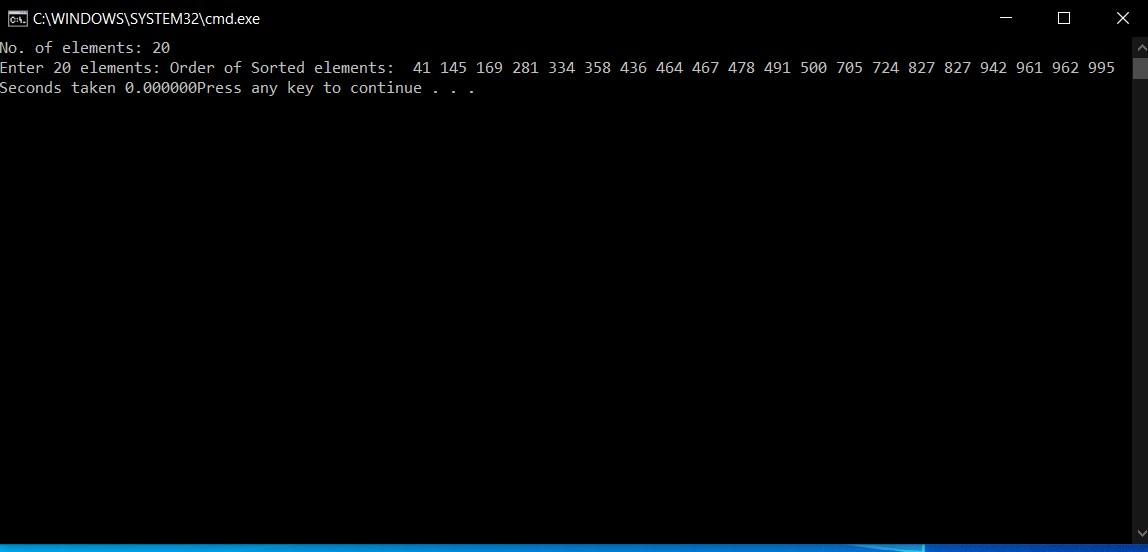
**printf(" %d",number[i]);**

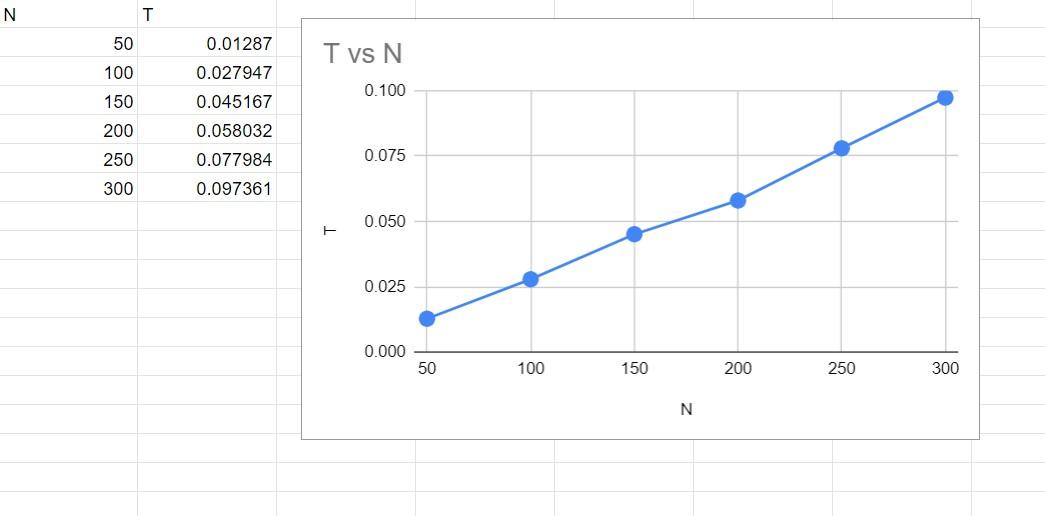
**printf("\nSeconds taken %lf",(double)(end-start)/CLOCKS\_PER\_SEC);**

**return 0;**

**}**

**OUTPUT-**





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

**#include <stdio.h>**

**#include <time.h>**

**#include <stdlib.h>**

**#include <math.h>**

**void swap(int \*,int \*);**

**void heapify(int [],int,int);**

**void heapSort(int[], int);**

**int main()**

**{**

**int a[15000], n, i, j, ch, temp;**

**clock\_t start, end;**

**while (1)**

**{**

**printf("\n1:FOR MANUAL ENTRY");**

**printf("\n2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 15000");**

**printf("\n3:EXIT");**

**printf("\nENTER YOUR CHOICE:");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**printf("\nENTER NUMBER OF ARRAY ELEMENTS: ");**

**scanf("%d", &n);**

**printf("\nENTER ARRAY ELEMENTS: ");**

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

**{**

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

**}**

**start = clock();**

**heapSort(a, n);**

**end = clock();**

**printf("\nSORTED ARRAY IS: ");**

**for (i = n-1; i >= 0; i--)**

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

**printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n, (((double)(end - start)) / CLOCKS\_PER\_SEC));**

**break;**

**case 2:**

**n = 500;**

**while (n <= 14500)**

**{**

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

**{**

**//a[i]=rand()%n;**

**a[i] = n - i;**

**}**

**start = clock();**

**heapSort(a, n);**

**for (j = 0; j < 500000; j++)**

**{**

**temp = 38 / 600;**

**}**

**end = clock();**

**printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n, (((double)(end - start)) / CLOCKS\_PER\_SEC));**

**n = n + 1000;**

**}**

**break;**

**case 3:**

**exit(0);**

**}**

**getchar();**

**}**

**}**

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

**{**

**int temp = \*a;**

**\*a = \*b;**

**\*b = temp;**

**}**

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

**{**

**int largest = i;**

**int left = 2 \* i + 1;**

**int right = 2 \* i + 2;**

**if (left < n && arr[left] > arr[largest])**

**largest = left;**

**if (right < n && arr[right] > arr[largest])**

**largest = right;**

**if (largest != i)**

**{**

**swap(&arr[i], &arr[largest]);**

**heapify(arr, n, largest);**

**}**

**}**

**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--)**

**{**

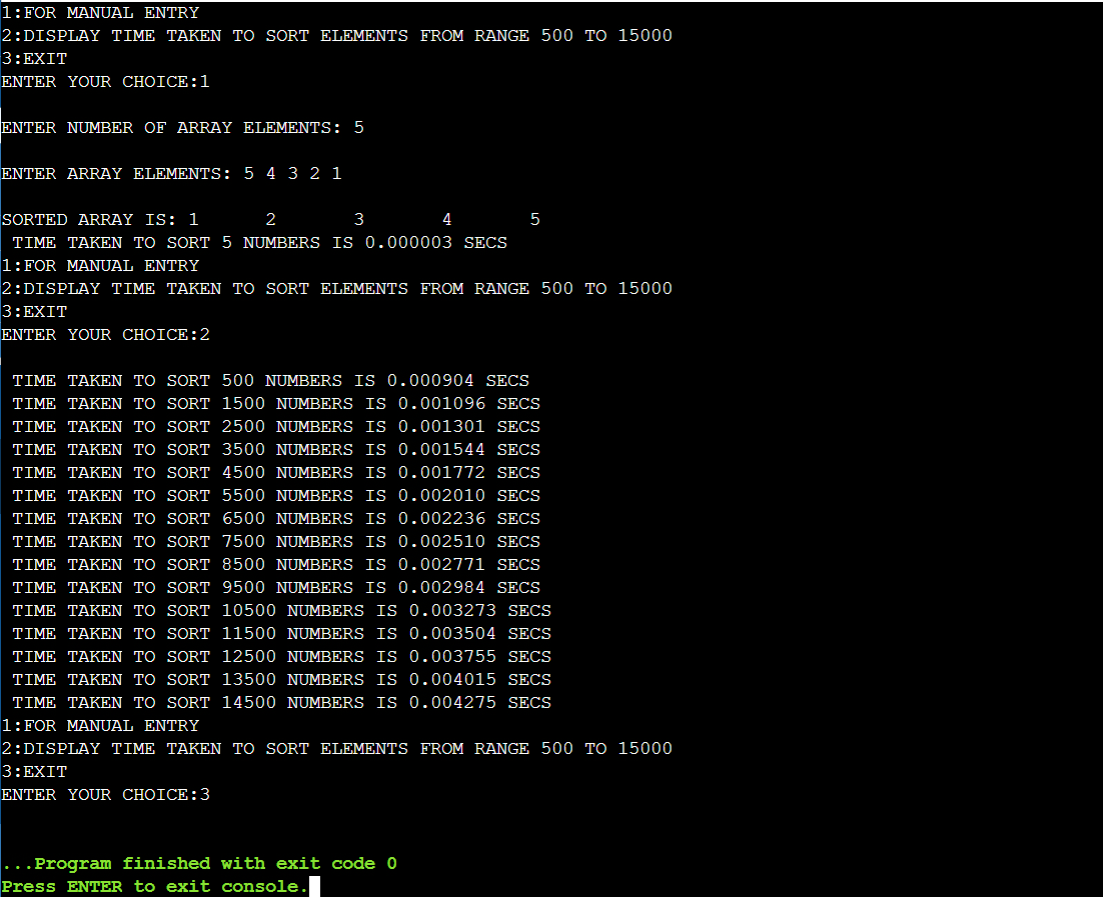
**swap(&arr[0], &arr[i]);**

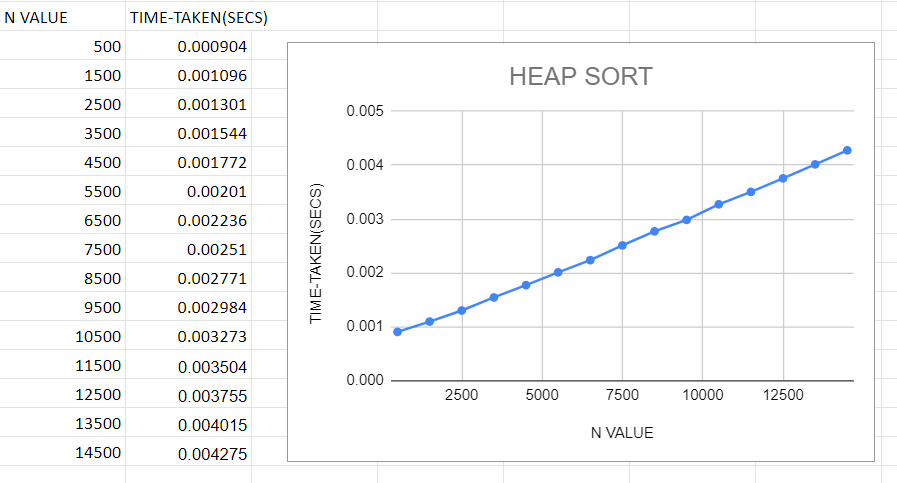
**heapify(arr, i, 0);**

**}**

**}**

**OUTPUT-**





**11) Implement Warshall’s algorithm using dynamic programming.**

**#include<stdio.h>**

**int a[30][30];**

**void warshall(int n){**

**for(int k=1;k<=n;k++)**

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

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

**a[i][j]=a[i][j]|| (a[i][k] && a[k][j]);**

**}**

**int main(){**

**int n;**

**printf("Enter no of vertices: \n");**

**scanf("%d",&n);**

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

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

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

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

**warshall(n);**

**printf("Transitive Closure: \n");**

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

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

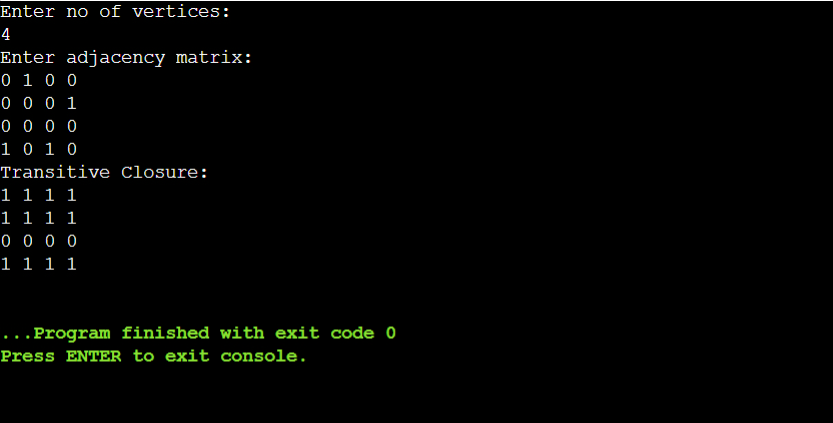
**printf("%d ",a[i][j]);**

**printf("\n");**

**}**

**}**

**OUTPUT-**



**12) Implement 0/1 Knapsack problem using dynamic programming.**

**#include<stdio.h>**

**#include<conio.h>**

**void knapsack();**

**int max(int,int);**

**int i,j,n,m,p[10],w[10],v[10][10];**

**void main()**

**{**

**printf("\nenter the no. of items:\t");**

**scanf("%d",&n);**

**printf("\nenter the weight of the each item:\n");**

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

**{**

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

**}**

**printf("\nenter the profit of each item:\n");**

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

**{**

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

**}**

**printf("\nenter the knapsack's capacity:\t");**

**scanf("%d",&m);**

**knapsack();**

**getch();**

**}**

**void knapsack()**

**{**

**int x[10];**

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

**{**

**for(j=0;j<=m;j++)**

**{**

**if(i==0||j==0)**

**{**

**v[i][j]=0;**

**}**

**else if(j-w[i]<0)**

**{**

**v[i][j]=v[i-1][j];**

**}**

**else**

**{**

**v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);**

**}**

**}**

**}**

**printf("\nthe output is:\n");**

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

**{**

**for(j=0;j<=m;j++)**

**{**

**printf("%d\t",v[i][j]);**

**}**

**printf("\n\n");**

**}**

**printf("\nthe optimal solution is %d",v[n][m]);**

**printf("\nthe solution vector is:\n");**

**for(i=n;i>=1;i--)**

**{**

**if(v[i][m]!=v[i-1][m])**

**{**

**x[i]=1;**

**m=m-w[i];**

**}**

**else**

**{**

**x[i]=0;**

**}**

**}**

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

**{**

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

**}**

**}**

**int max(int x,int y)**

**{**

**if(x>y)**

**{**

**return x;**

**}**

**else**

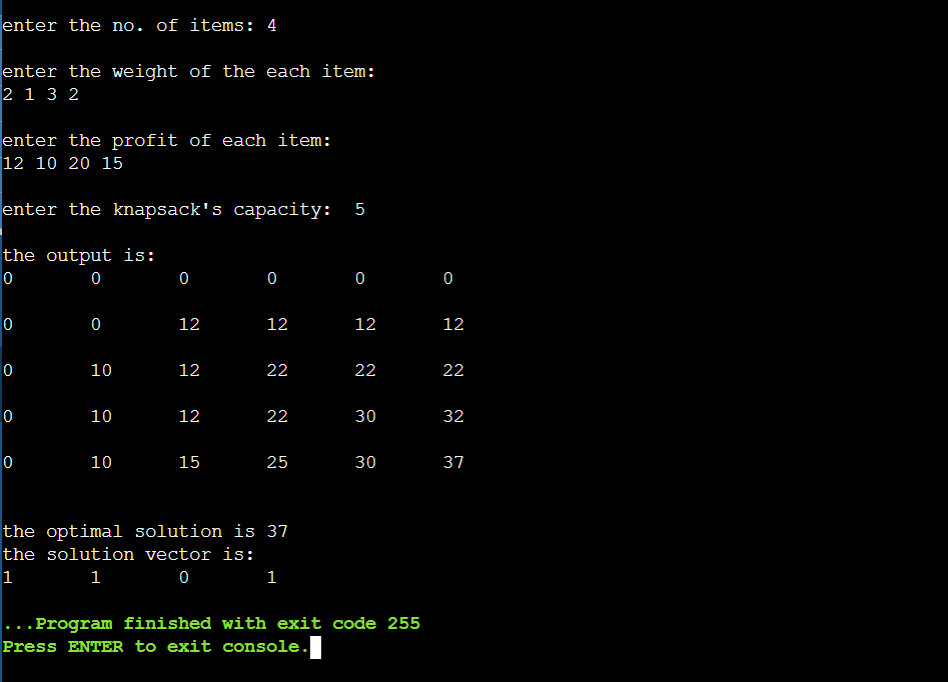
**{**

**return y;**

**}**

**}**

**OUTPUT-**

****

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

**#include<stdio.h>**

**int n;**

**void display(int dist[][n]);**

**void floyd (int graph[][n])**

**{**

**int dist[n][n], i, j, k;**

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

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

**dist[i][j] = graph[i][j];**

**for (k = 0; k < n; k++)**

**{**

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

**{**

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

**{**

**if (dist[i][k] + dist[k][j] < dist[i][j])**

**dist[i][j] = dist[i][k] + dist[k][j];**

**}**

**}**

**}**

**display(dist);**

**}**

**void display(int dist[][n])**

**{**

**printf ("DISTANCE MATRIX \n");**

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

**{**

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

**{**

**if (dist[i][j] == 99)**

**printf("99 ");**

**else**

**printf ("%d ", dist[i][j]);**

**}**

**printf("\n");**

**}**

**}**

**int main()**

**{ printf("ENTER ORDER OF MATRIX \n");**

**scanf("%d",&n);**

**int graph[n][n];**

**printf("ENTER ELEMENTS OF MATRIX and 99 FOR INFINITY\n");**

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

**{**

**for(int j = 0;j < n; j++) {**

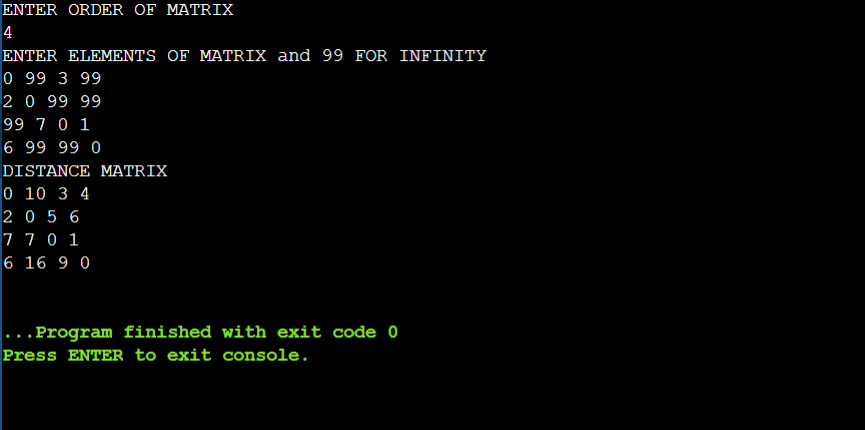
**scanf("%d",&graph[i][j]);**

**} }**

**floyd(graph);**

**return 0;**

**}**



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

**#include<stdio.h>**

**void prims();**

**int c[10][10],n;**

**void main()**

**{**

**int i,j;**

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

**scanf("%d",&n);**

**printf("\nenter the cost matrix:\n");**

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

**{**

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

**{**

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

**}**

**}**

**prims();**

**}**

**void prims()**

**{**

**int i,j,u,v,min;**

**int ne=0,mincost=0;**

**int elec[10];**

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

**{**

**elec[i]=0;**

**}**

**elec[1]=1;**

**while(ne!=n-1)**

**{**

**min=9999;**

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

**{**

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

**{**

**if(elec[i]==1)**

**{**

**if(c[i][j]<min)**

**{**

**min=c[i][j];**

**u=i;**

**v=j;**

**} } } }**

**if(elec[v]!=1) {**

**printf("\n%d----->%d=%d\n",u,v,min);**

**elec[v]=1;**

**ne=ne+1;**

**mincost=mincost+min; }**

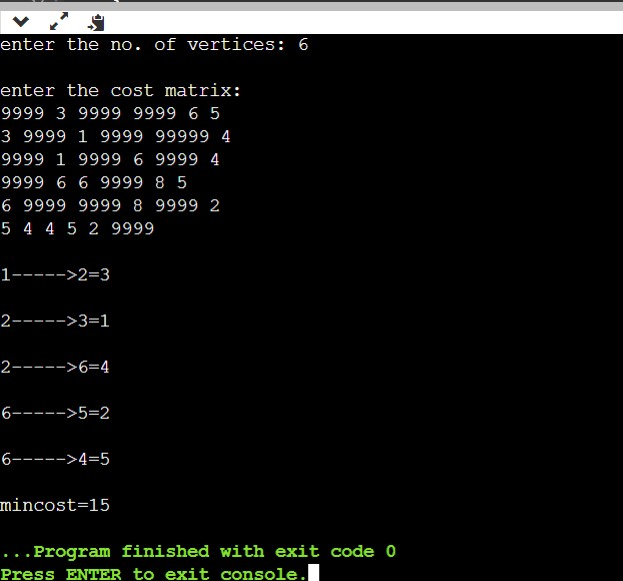
**c[u][v]=c[v][u]=9999;**

**}**

**printf("\nmincost=%d",mincost);**

**}**

**OUTPUT-**

****

**15) Find Minimum Cost Spanning Tree of an undirected graph using Kruskals algorithm**

**#include<stdio.h>**

**void kruskals();**

**int c[10][10],n;**

**void main()**

**{**

**int i,j;**

**printf("\nenter the no. of vertices:\t");**

**scanf("%d",&n);**

**printf("\nenter the cost matrix:\n");**

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

**{**

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

**{**

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

**}**

**}**

**kruskals();**

**}**

**void kruskals()**

**{**

**int i,j,u,v,a,b,min;**

**int ne=0,mincost=0;**

**int parent[10];**

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

**{**

**parent[i]=0;**

**}**

**while(ne!=n-1)**

**{**

**min=9999;**

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

**{**

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

**{**

**if(c[i][j]<min)**

**{**

**min=c[i][j];**

**u=a=i;**

**v=b=j;**

**}**

**}**

**}**

**while(parent[u]!=0)**

**{**

**u=parent[u];**

**}**

**while(parent[v]!=0)**

**{**

**v=parent[v];**

**}**

**if(u!=v)**

**{**

**printf("\n%d----->%d=%d\n",a,b,min);**

**parent[v]=u;**

**ne=ne+1;**

**mincost=mincost+min;**

**}**

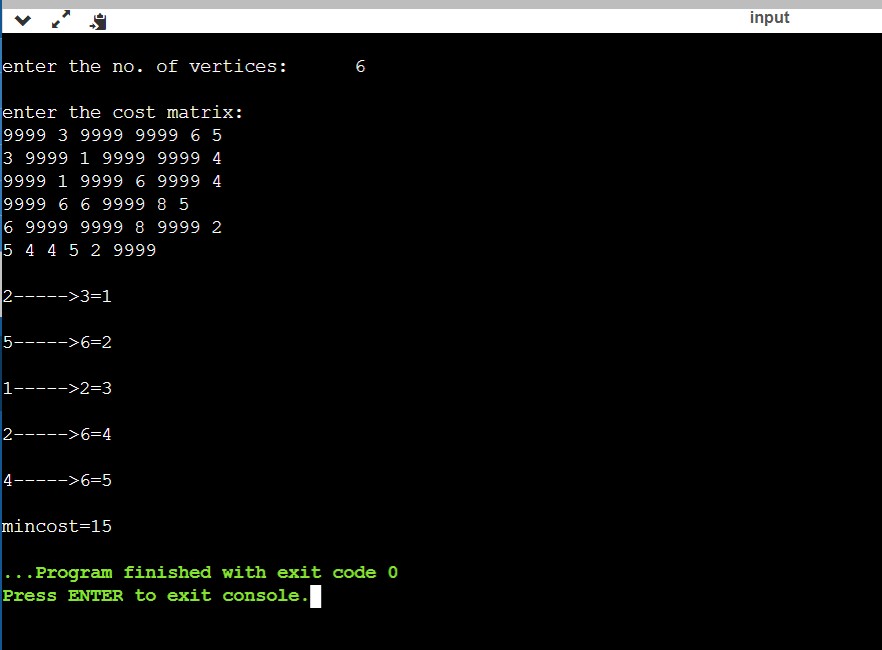
**c[a][b]=c[b][a]=9999;**

**}**

**printf("\nmincost=%d",mincost);**

**}**

**OUTPUT -**

****

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

**#include<stdio.h>**

**void dijkstras();**

**int c[10][10],n,src;**

**void main()**

**{**

**int i,j;**

**printf("\nenter the no of vertices: ");**

**scanf("%d",&n);**

**printf("\nenter the cost matrix:\n");**

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

**{**

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

**{**

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

**}**

**}**

**printf("\nenter the source node: ");**

**scanf("%d",&src);**

**dijkstras();**

**}**

**void dijkstras()**

**{**

**int vis[10],dist[10],u,j,count,min;**

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

**{**

**dist[j]=c[src][j];**

**}**

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

**{**

**vis[j]=0;**

**}**

**dist[src]=0;**

**vis[src]=1;**

**count=1;**

**while(count!=n)**

**{**

**min=9999;**

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

**{**

**if(dist[j]<min&&vis[j]!=1)**

**{**

**min=dist[j];**

**u=j;**

**}**

**}**

**vis[u]=1;**

**count++;**

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

**{**

**if(min+c[u][j]<dist[j]&&vis[j]!=1)**

**{**

**dist[j]=min+c[u][j];**

**}**

**}**

**}**

**printf("\nthe shortest distance is:\n");**

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

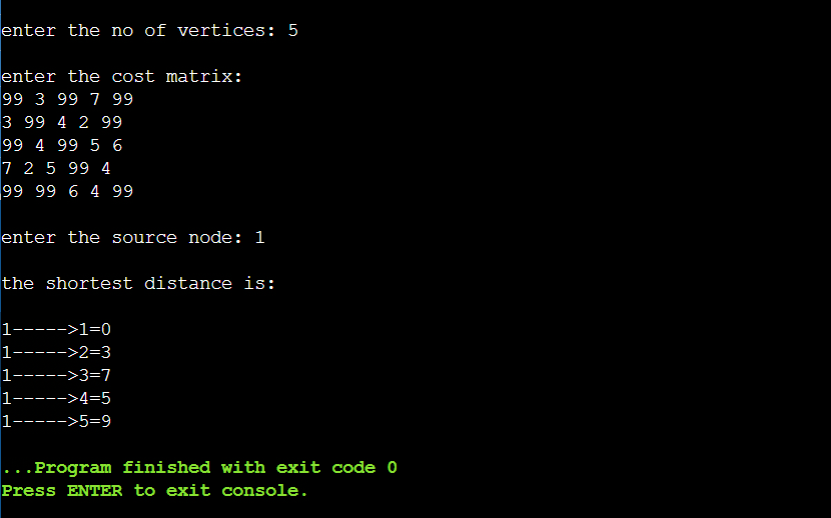
**{**

**printf("\n%d----->%d=%d",src,j,dist[j]);**

**}**

**}**

**OUTPUT -**



**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.**

**#include <stdio.h>**

**#include <conio.h>**

**int count, w[10], d, x[10];**

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

**{**

**int i;**

**x[k] = 1;**

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

**{**

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

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

**{**

**if (x[i] == 1)**

**printf("%d ", 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 + 1]) <= d)**

**{**

**x[k] = 0;**

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

**}**

**}**

**void main()**

**{**

**int sum = 0, i, n;**

**printf("Enter the number of elements\n");**

**scanf("%d", &n);**

**printf("Enter the elements in ascending order\n");**

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

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

**printf("Enter the required sum\n");**

**scanf("%d", &d);**

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

**sum += w[i];**

**if (sum < d)**

**{**

**printf("No solution exists\n");**

**return;**

**}**

**printf("The solution is\n");**

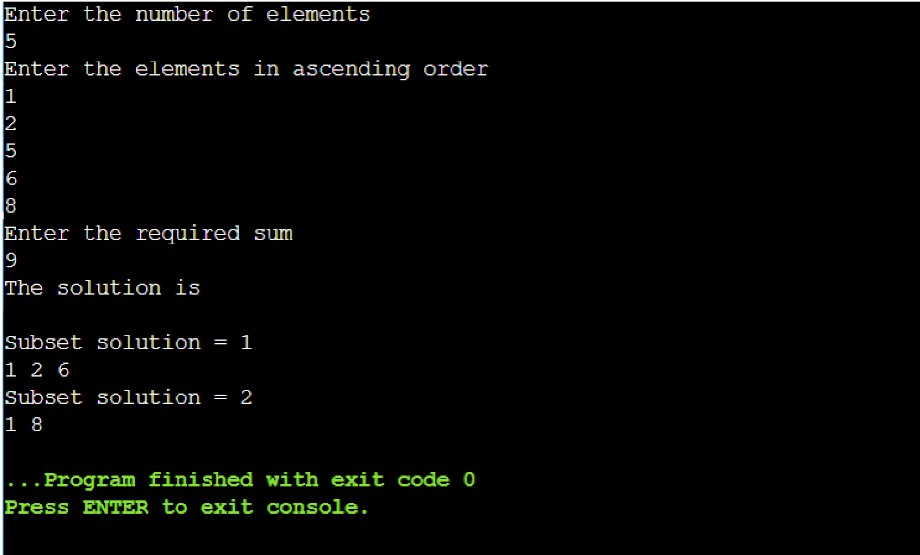
**count = 0;**

**subset(0, 0, sum);**

**getch();**

**}**

**OUTPUT -**

****

**18)Implement N-Queens problem using backtracking**

**#include<stdio.h>**

**#include<conio.h>**

**void nqueens(int n)**

**{**

**int k,x[20],count=0;**

**k=1;**

**x[k]=0;**

**while(k!=0)**

**{**

**x[k]++;**

**while(place(x,k)!=1 && x[k]<=n)**

**x[k]++;**

**if(x[k]<=n)**

**{**

**if(k==n)**

**{**

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

**printf("Queen\t\tPosition\n");**

**for(k=1;k<=n;k++)**

**printf("%d\t\t%d\n", k,x[k]);**

**}**

**else**

**{**

**k++;**

**x[k]=0;**

**}**

**}**

**else**

**k--;**

**}**

**}**

**int place(int x[], int k)**

**{**

**int i;**

**for(i=1;i<=k-1;i++)**

**{**

**if(i+x[i]==k+x[k]||i-x[i]==k-x[k]||x[i]==x[k])**

**return 0;**

**}**

**return 1;**

**}**

**void main()**

**{**

**int n;**

**printf("Enter the number of Queens\n");**

**scanf("%d", &n);**

**nqueens(n);**

**}**

**OUTPUT-**

