**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**VARUN URS M S (1BM20CS182)**

***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 **VARUN URS M S (1BM18CS182),** 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.

Name of the Lab-In charge:

  BASAVARAJ JAKKALI             **Dr. Jyothi S Nayak**

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 | **5** |
| **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. | **7** |
| **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. | **11** |
| **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. | **15** |
| **5** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. | **19** |
| **6** | Write program to obtain the Topological ordering of vertices in a given digraph. | **23** |
| **7** | Implement Johnson Trotter algorithm to generate permutations. | **25** |
| **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. | **29** |
| **9** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | **34** |
| **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. |

**PROGRAM 1**

Write a recursive program to Solve

a) Towers - of - Hanoi problem

#include<stdio.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);

}

}

void main(){

int disk;

int moves;

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

scanf("%d",&disk);

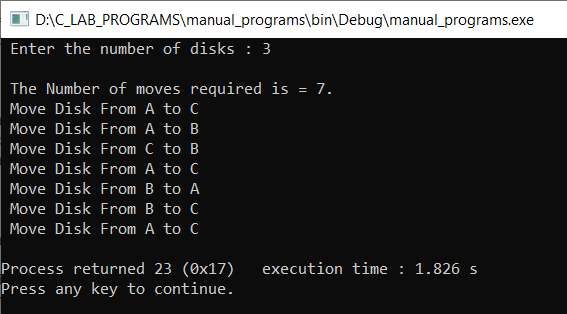
moves=pow(2,disk)-1;

printf("\n The Number of moves required is = %d. \n",moves);

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

}

**OUTPUT : -**



b) To find GCD

#include <stdio.h>

int hcf(int n1, int n2);

int main()

{

int n1, n2;

printf(" Enter two positive integers : ");

scanf("%d %d", &n1,&n2);

printf(" G.C.D of %d and %d is %d.", n1, n2, hcf(n1,n2));

return 0;

}

int hcf(int n1, int n2)

{

if (n2 != 0)

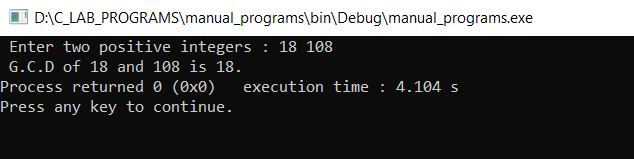
return hcf(n2, n1 % n2);

else

return n1;

}

**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.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

int n,a[10000];

int bin\_search(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\_search(a,low,mid-1,key);

else

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

}

int lin\_search(int a[],int i,int high,int key)

{

if(i > high)

return -1;

if(key == a[i])

return i;

else

return lin\_search(a,i+1,high,key);

}

void main()

{

int ch,key,search\_status,temp;

clock\_t start,end;

unsigned long int i,j;

while(1)

{

printf("\n 1.Binary search \t 2. Linear search \t 3.Exit\n");

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

scanf("%d",&ch);

switch(ch)

{

case 1:

n = 1000;

while(n<=5000)

{

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

a[i] = i;

key = a[n-1];

start = clock();

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

if(search\_status == -1)

printf("\n Key not found ");

else

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

for(j=0;j<500000000;j++)

temp = 38/600;

end = clock();

printf("\n Time 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] = i;

key = a[n-1];

start = clock();

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

if(search\_status == -1)

printf("\n Key not found ");

else

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

for(j=0;j<500000000;j++)

temp = 38/600;

end = clock();

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

n += 1000;

}

break;

default:

exit(0);

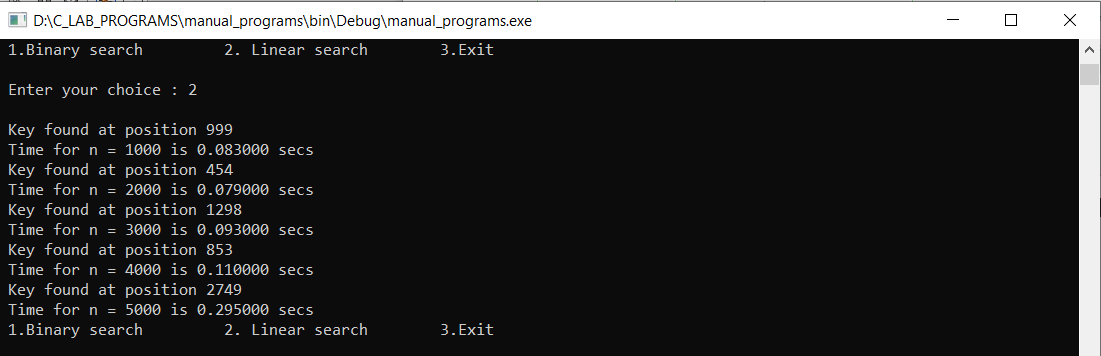
}

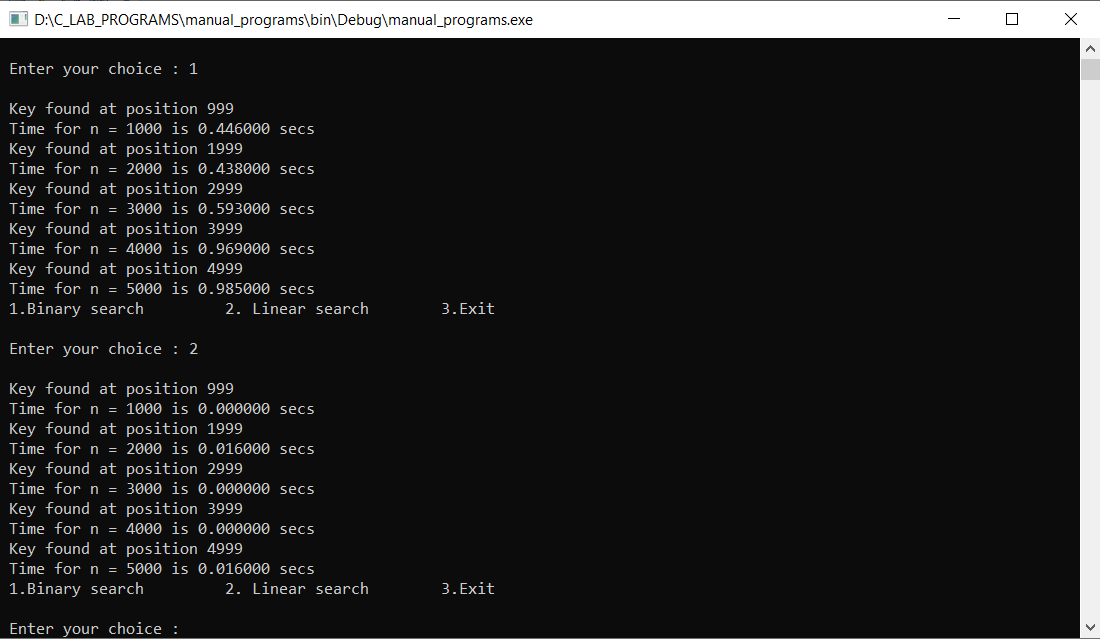
getchar();

}

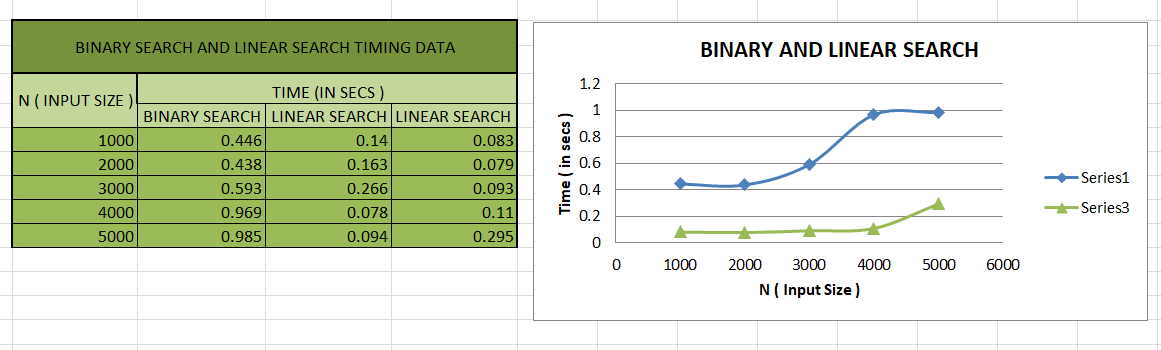
}

**OUTPUT : -**





**GRAPH : -**



**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.

#include<stdio.h>

#include<time’s>

#include<stdlib.h>

void selSort(int n,int a[]);

void main()

{

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

clock\_t start,end;

while(1)

{

printf("\n 1. For manual entry of N value and array elements ");

printf("\n 2. To display time taken for sorting number of elements N in the range 500 to 14500 ");

printf("\n 3. To exit ");

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

scanf("%d",&ch);

switch(ch)

{

case 1 :  printf("\n Enter the number of elements : ");

scanf("%d",&n);

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

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

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

start = clock();

selSort(n,a);

end = clock();

printf("\n Sorted array is : ");

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

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

printf("\n Time taken to sort %d elements is %1.10f seconds. \n",n , (((double)(end - start))/CLOCKS\_PER\_SEC));

break;

case 2 :  n = 500;

while(n <= 14500)

{

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

a[i] = rand()%1000;

start = clock();

selSort(n,a);

for(j=0;j<15000000;j++)

temp = 38/600;

end = clock();

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

n = n + 1000;

}

break;

case 3 : exit(0);

}

getchar();

}

}

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];

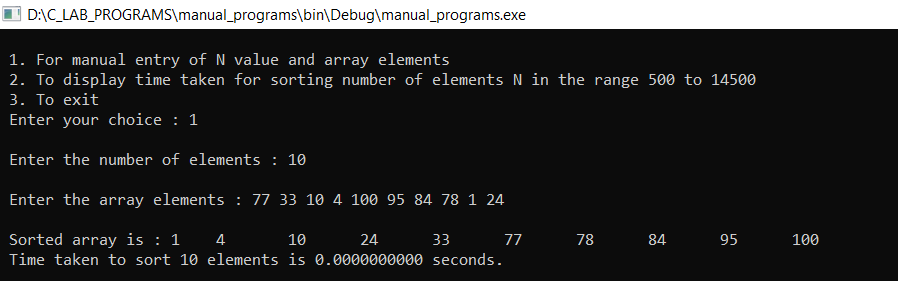
a[i] = a[pos];

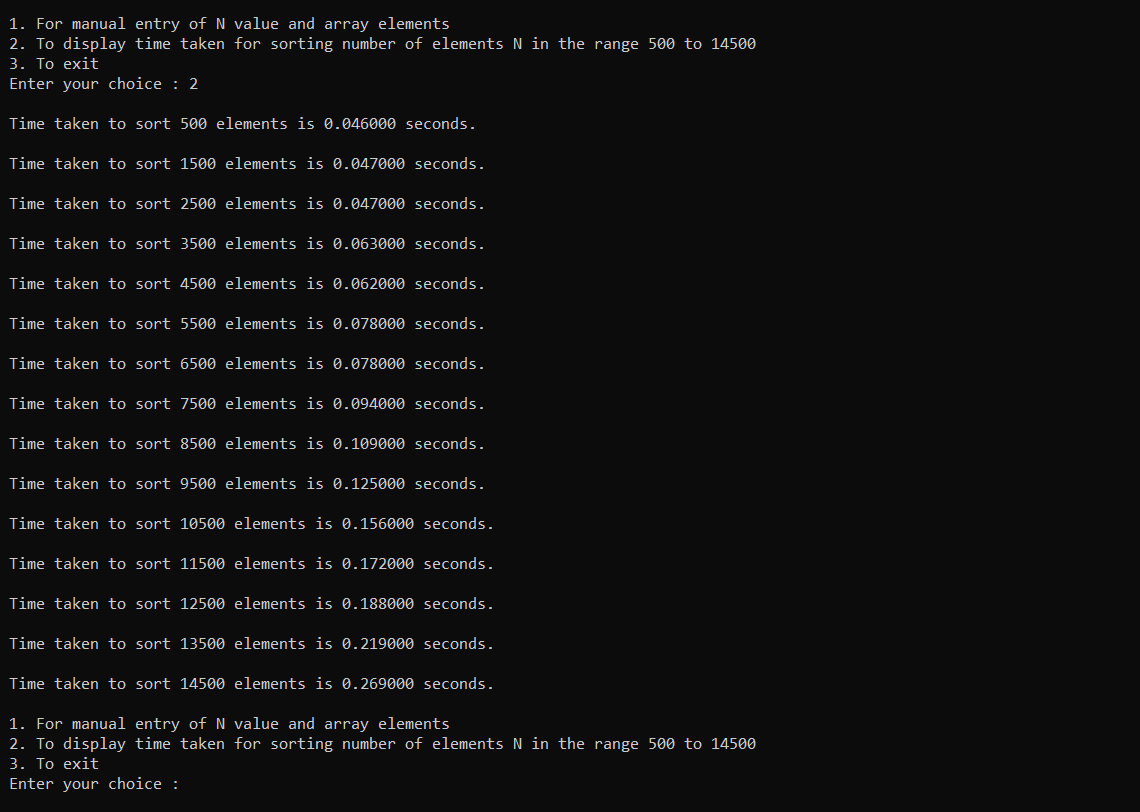
a[pos] = t;

}

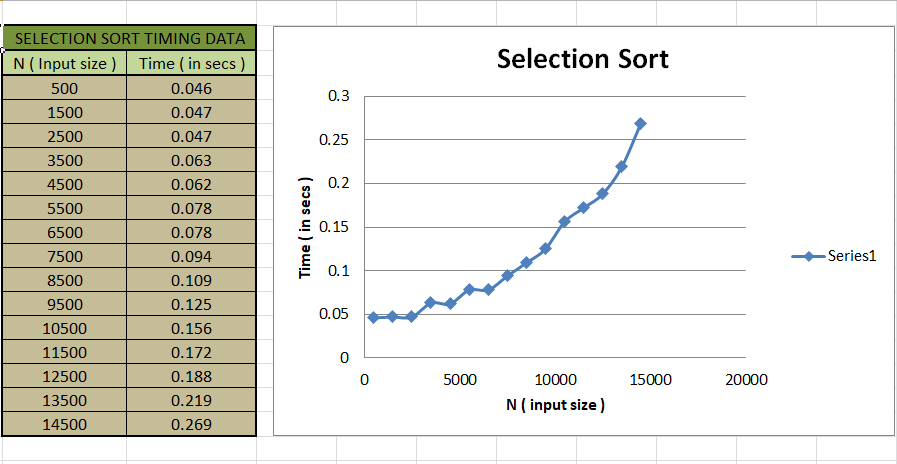
}

**OUTPUT : -**

****

****

**GRAPH : -**

****

**PROGRAM 4**

Write program to do the following:

1. 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);

void main()

{

int i,j,src;

printf("\n Enter the no 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",&a[i][j]);

}

}

printf("\n Enter the source node : ");

scanf("%d",&src);

bfs(src);

getch();

}

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

{

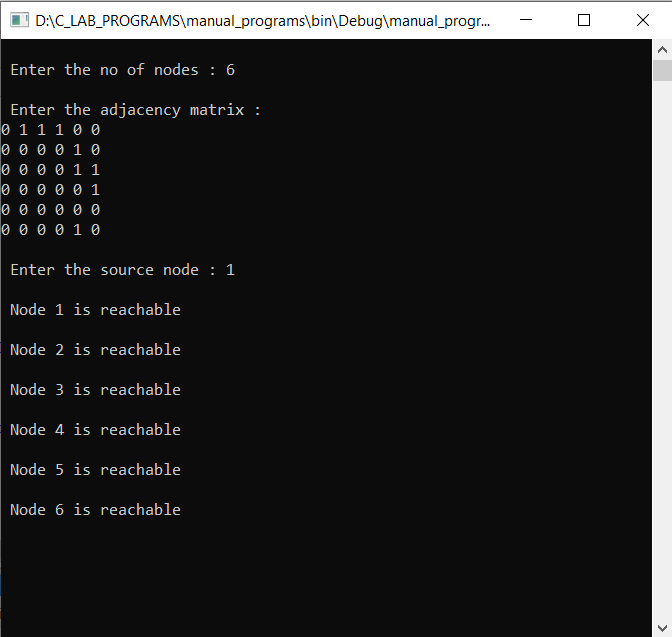
printf("\n Node %d is reachable\n",j);

}

}

}

**OUTPUT : -**



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

#include<stdio.h>

#include<conio.h>

int a[10][10];

int n,vis[10];

int dfs(int);

void main()

{

int i,j,src,ans;

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

{

vis[j]=0;

}

printf("\n Enter the no 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",&a[i][j]);

}

}

printf("\n Enter the source node : ");

scanf("%d",&src);

ans = dfs(src);

if(ans==1)

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

else

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

}

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;

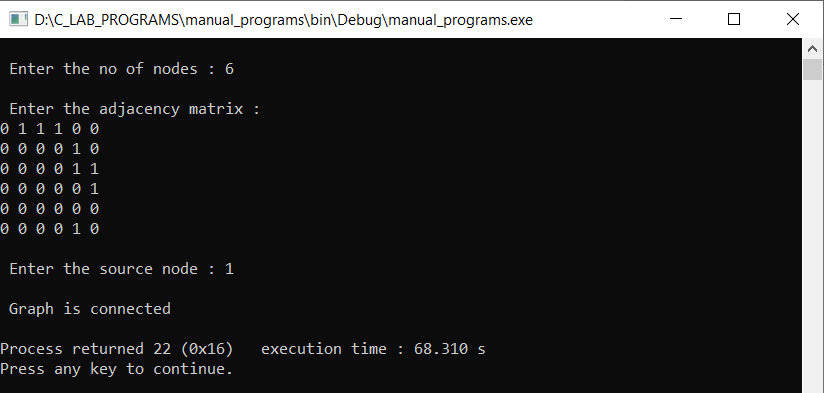
}

}

return 1;

}

**OUTPUT : -**



**PROGRAM 5**

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

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void insertionSort(int n,int a[]);

void main()

{

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

clock\_t start,end;

while(1)

{

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

printf("\n 1. For manual entry of N value and array elements ");

printf("\n 2. To display time taken for sorting number of elements N in the range 1000 to 15000 ");

printf("\n 3. To exit ");

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

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

scanf("%d",&ch);

switch(ch)

{

case 1 :  printf("\n Enter the number of elements : ");

scanf("%d",&n);

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

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

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

start = clock();

insertionSort(n,a);

end = clock();

printf("\n Sorted array is : ");

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

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

printf("\n Time taken to sort %d elements is %1.10f seconds. \n",n , (((double)(end - start))/CLOCKS\_PER\_SEC));

break;

case 2 :  n = 1000;

while(n <= 15000)

{

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

a[i] = rand()%1000;

start = clock();

insertionSort(n,a);

for(j=0;j<50000000;j++)

temp = 38/600;

end = clock();

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

n = n + 1000;

}

break;

case 3 : exit(0);

}

getchar();

}

}

void insertionSort(int n,int a[]){

int i,j,key = 0;

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

key = a[i];

j = i - 1;

while(j >= 0 && a[j] > key){

a[j + 1] = a[j];

j--;

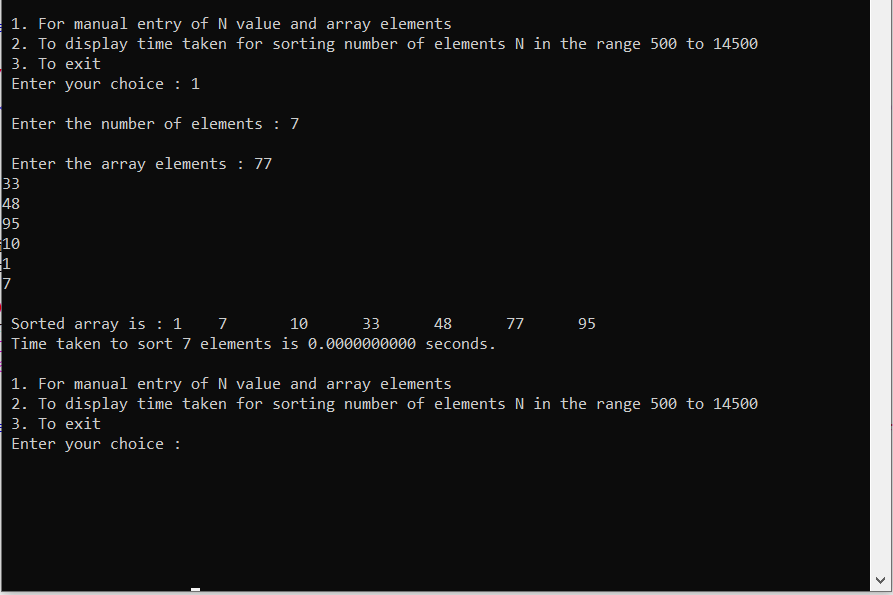
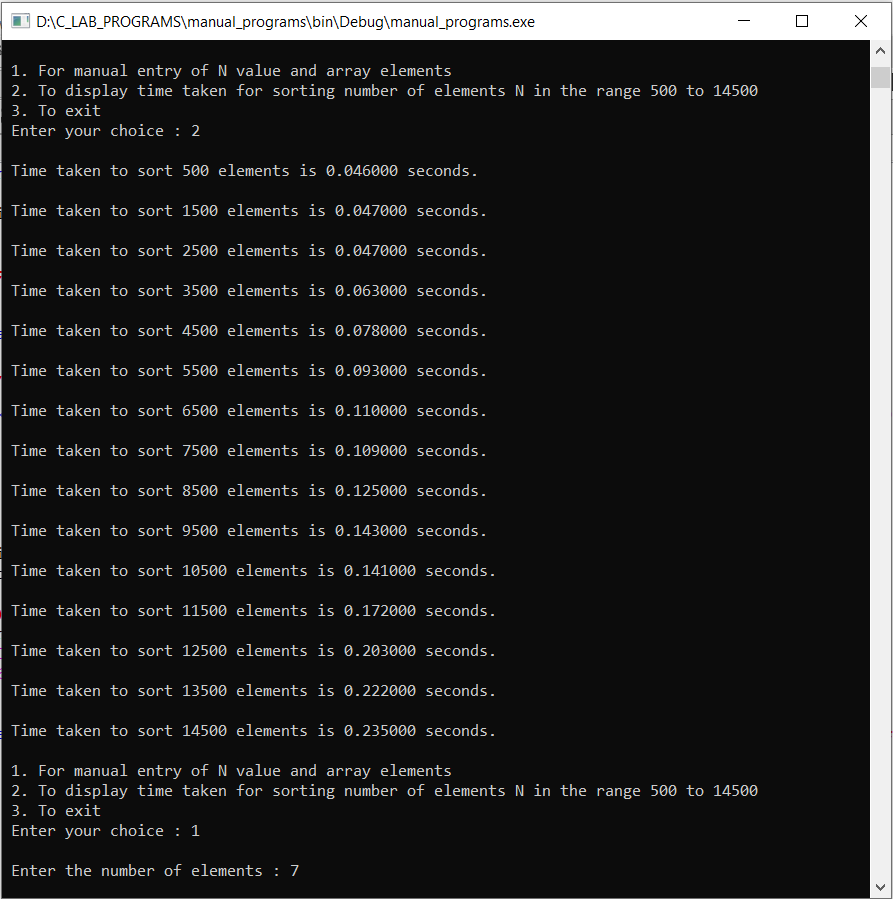
}

a[j + 1] = key;

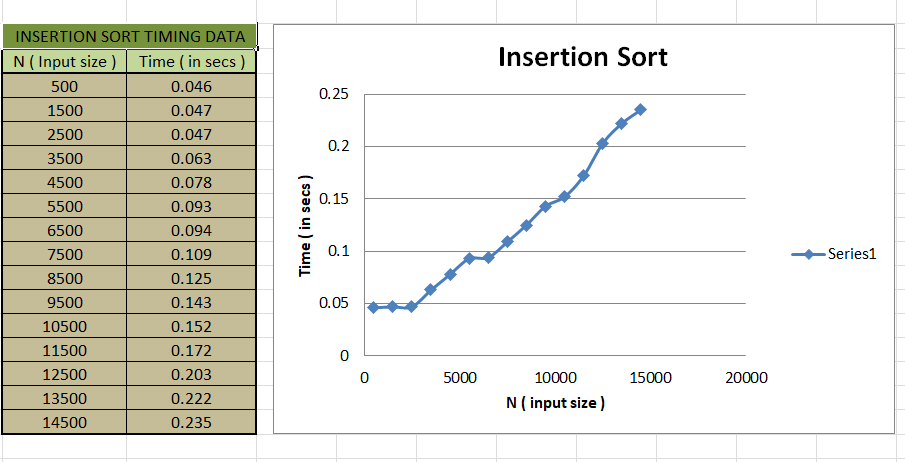
}

}

**OUTPUT : -**

****

**GRAPH : -**

****

**PROGRAM 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]);

}

}

void main()

{

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

printf("\nEnter number of nodes : ");

scanf(" %d", &n);

printf("\nEnter the adjacency matrix\n");

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

{

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

{

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

}

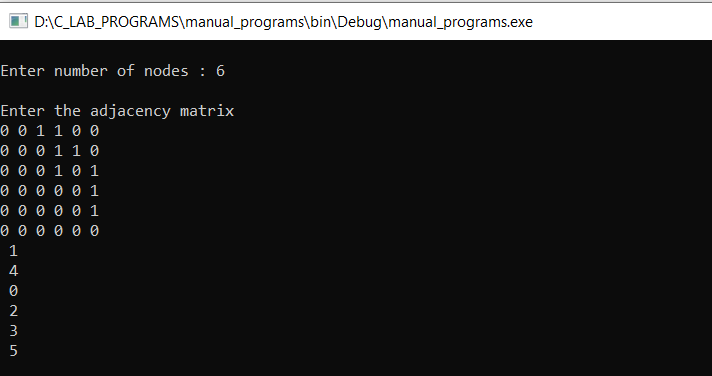
}

source\_removal(n,a);

getch();

}

**OUTPUT : -**



**PROGRAM 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 : ");

scanf("%d",&num);

int arr[num],d[num];

z = factorial(num);

printf("Total permutations = %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);

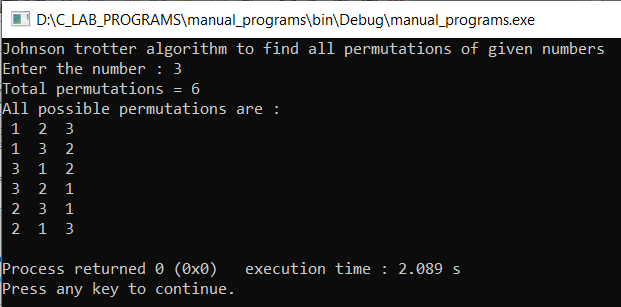
printf("\n");

}

return 0;

}

**OUTPUT : -**



**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.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void split(int[],int,int);

void combine(int[],int,int,int,int);

void main()

{

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

clock\_t start,end;

while(1)

{

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

printf("\n 1 For manual entry of N value and array elements");

printf("\n 2 To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n 3 To exit");

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

printf("\n\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();

split(a,0,n-1);

end=clock();

printf("\nSorted array is : ");

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

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

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

break;

case 2:

n=500;

while(n<=14500)

{

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

{

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

}

start=clock();

split(a,0,n-1);

//Dummy loop to create delay

for(j=0; j<98754777; j++)

{

temp=38/60;

}

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 split(int a[],int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

split(a,low,mid);

split(a,mid+1,high);

combine(a,low,mid,mid+1,high);

}

}

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

{

int temp[30000];

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++)

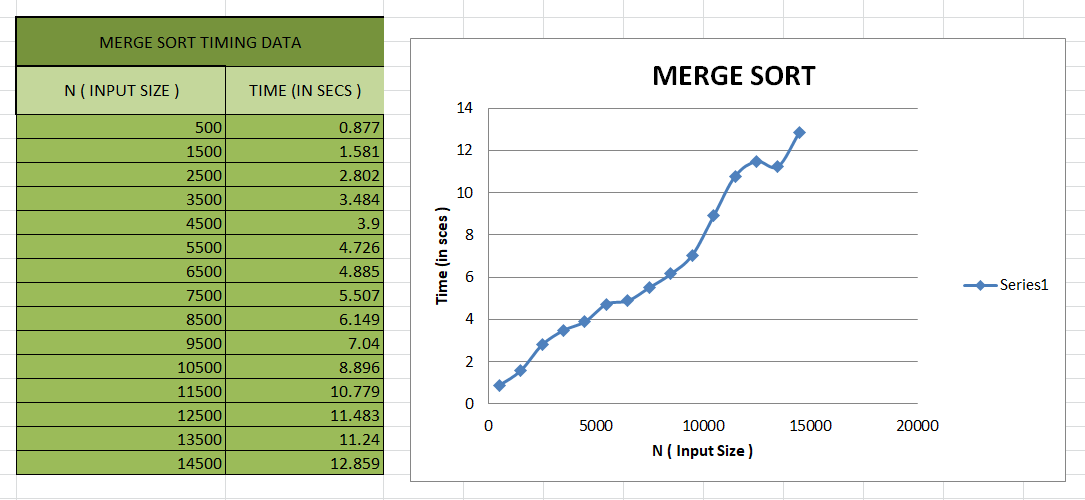
{

a[i]=temp[j];

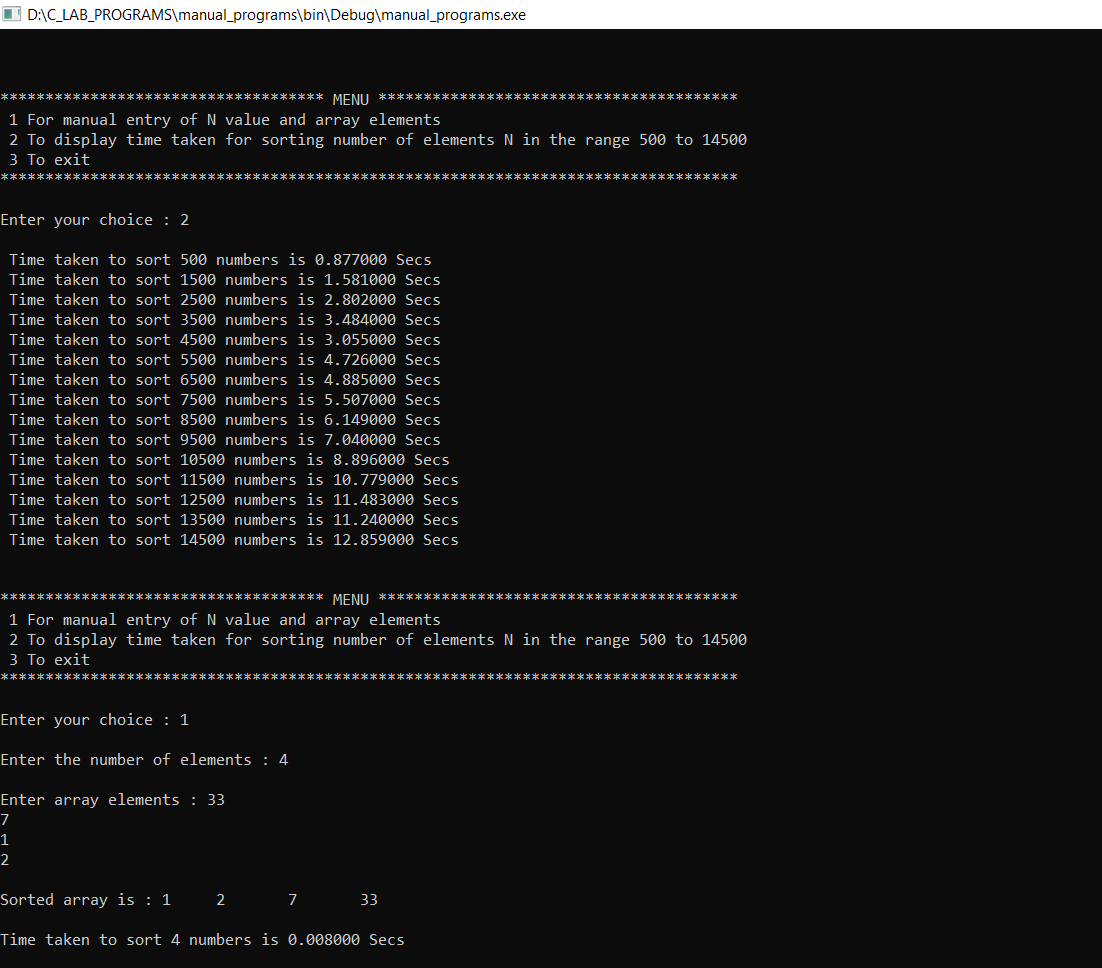
}

}

**GRAPH : -**



**OUTPUT : -**



**PROGRAM 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<math.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<10000000; 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);

}

}

void main(){

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

clock\_t start,end;

while(1){

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

printf("\n 1. For manual entry of N value and array elements ");

printf("\n 2. To display time taken for sorting number of elements N in the range 1000 to 15000 ");

printf("\n 3. To exit ");

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

scanf("%d",&ch);

switch(ch){

case 1 :  printf("\n Enter the number of elements : ");

scanf("%d",&n);

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

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

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

start = clock();

quicksort(a,0,n-1);

end = clock();

printf("\n Sorted array is : ");

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

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

printf("\n Time taken to sort %d elements is %1.10f seconds. \n",n , (((double)(end - start))/CLOCKS\_PER\_SEC));

break;

case 2 :  n = 100;

while(n <= 1500)

{

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

a[i] = rand()%1000;

start = clock();

quicksort(a,0,n-1);

for(j=0;j<5000;j++)

temp = 38/600;

end = clock();

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

n = n + 100;

}

break;

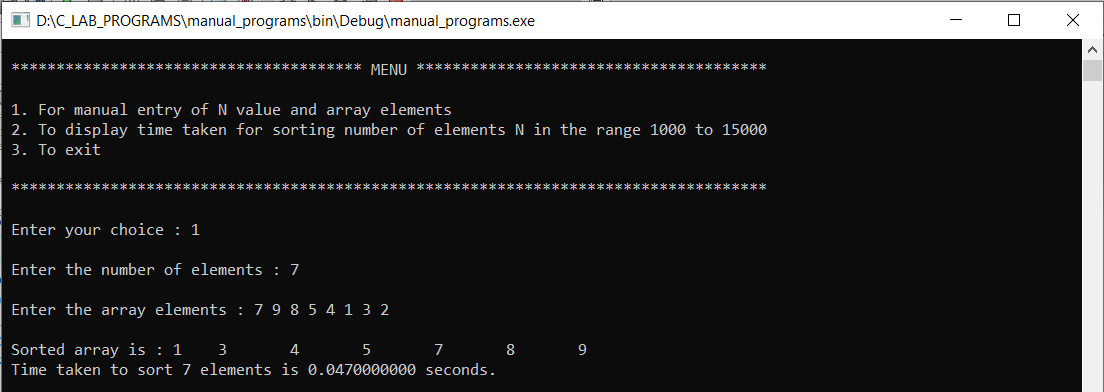
case 3 : exit(0);

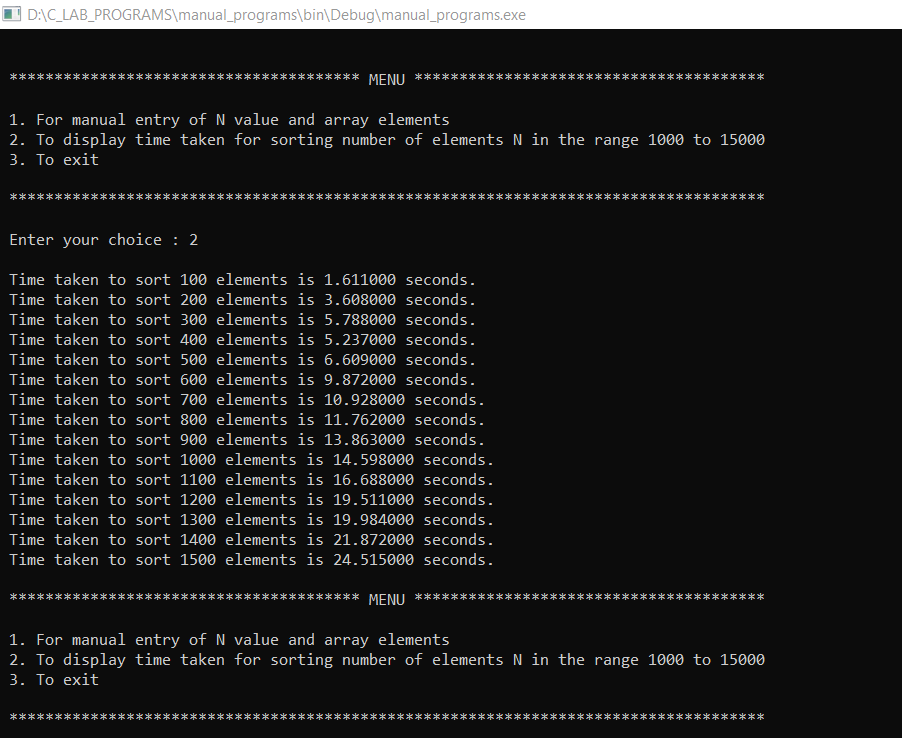
}

}

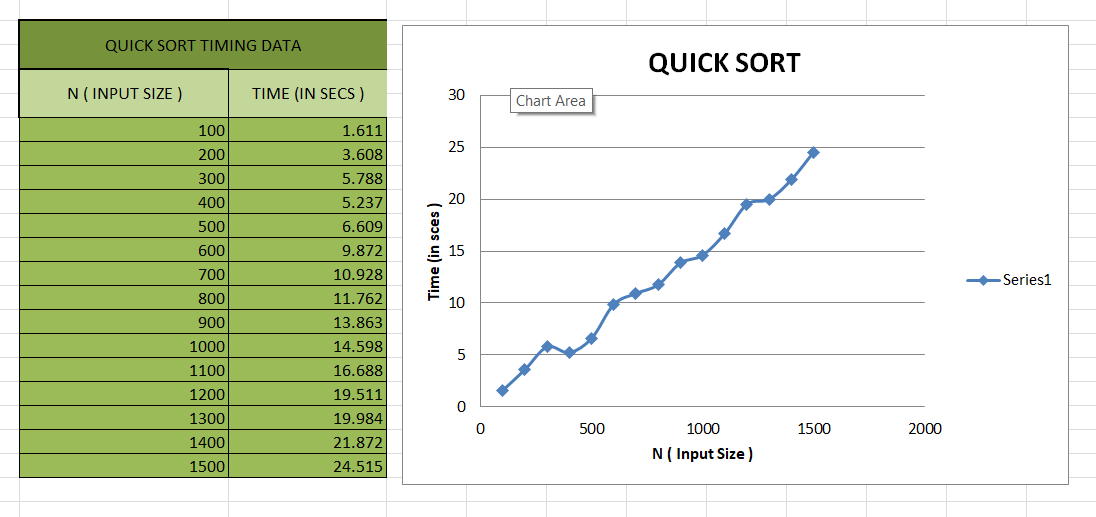
}

**OUTPUT : -**





**GRAPH : -**



**PROGRAM 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>

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

void heapify(int arr[], int N, int i);

void heapSort(int arr[], int N);

void main()

{

    int a[15000],n,i,j,ch,temp;

    clock\_t start,end;

    while(1)

    {

        printf("\n 1. For manual entry of N value and array elements ");

        printf("\n 2. To display time taken for sorting number of elements N in the range 500 to 14500 ");

        printf("\n 3. To exit ");

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

        scanf("%d",&ch);

        switch(ch)

        {

        case 1 :  printf("\n Enter the number of elements : ");

                  scanf("%d",&n);

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

                  for(i=0;i<n;i++)

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

                  start = clock();

                  heapSort(a,n);

                  end = clock();

                  printf("\n Sorted array is : ");

                  for(i=0;i<n;i++)

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

                  printf("\n Time taken to sort %d elements is %1.10f seconds.",n , (((double)(end - start))/CLOCKS\_PER\_SEC));

                  break;

        case 2 :  n = 500;

                  while(n <= 14500)

                  {

                    for(i=0;i<n;i++)

                        a[i] = rand()%1000;

                    start = clock();

                    heapSort(a,n);

                    for(j=0;j<95000000;j++)

                      temp = 38/600;

                    end = clock();

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

                    n = n + 500;

                  }

                  break;

        case 3 : exit(0);

        }

        getchar();

    }

}

// Function to swap the position of two elements

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

{

  int temp = \*a;

  \*a = \*b;

  \*b = temp;

}

// To heapify a subtree rooted with node i

// which is an index in arr[].

// n is size of heap

void heapify(int arr[], int N, int i)

{

  // Find largest among root, left child and right child

  // Initialize largest as root

  int largest = i;

  // left = 2\*i + 1

  int left = 2 \* i + 1;

  // right = 2\*i + 2

  int right = 2 \* i + 2;

  // If left child is larger than root

  if (left < N && arr[left] > arr[largest])

    largest = left;

  // If right child is larger than largest

  // so far

  if (right < N && arr[right] > arr[largest])

    largest = right;

  // Swap and continue heapifying if root is not largest

  // If largest is not root

  if (largest != i) {

    swap(&arr[i], &arr[largest]);

    // Recursively heapify the affected

    // sub-tree

    heapify(arr, N, largest);

  }

}

// Main function to do heap sort

void heapSort(int arr[], int N)

{

  // Build max heap

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

    heapify(arr, N, i);

  // Heap sort

  for (int i = N - 1; i >= 0; i--) {

    swap(&arr[0], &arr[i]);

    // Heapify root element to get highest element at

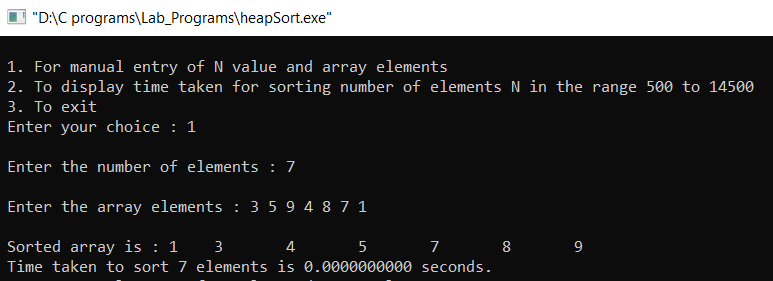
    // root again

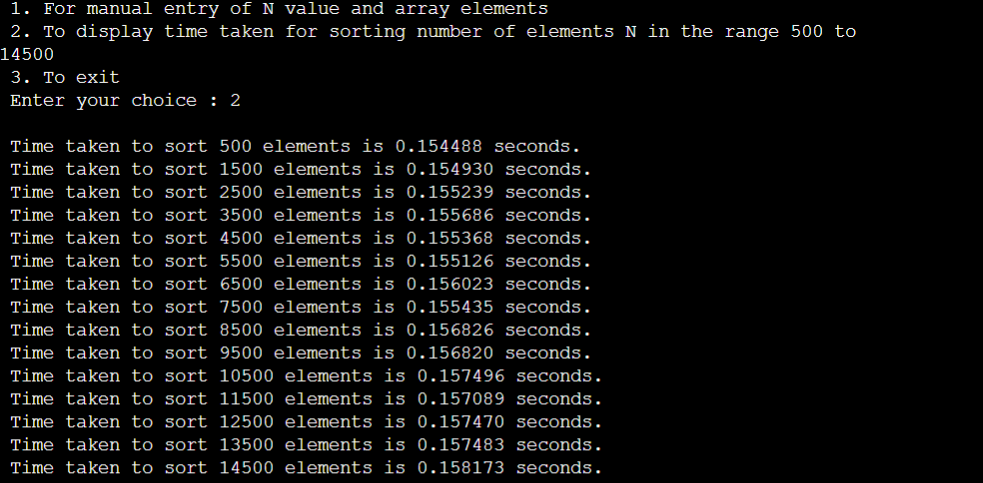
    heapify(arr, i, 0);

  }

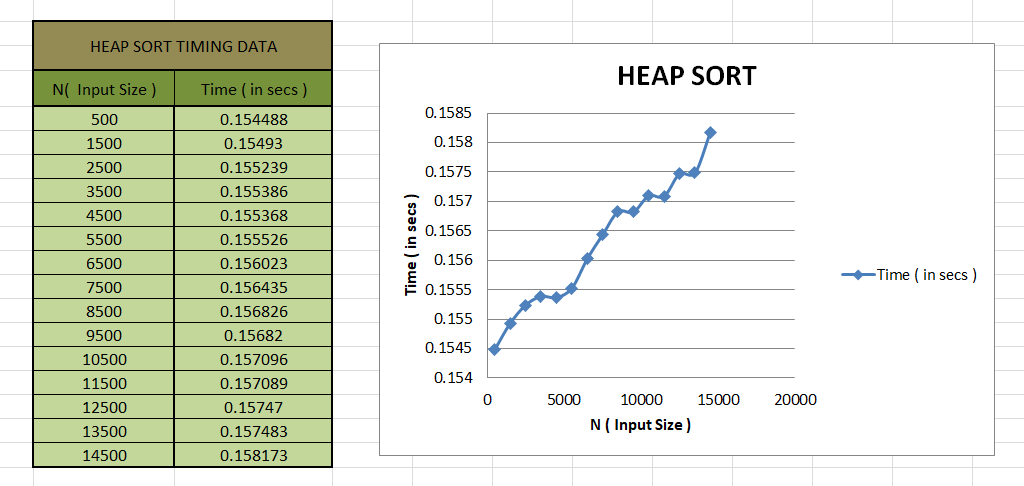
}

**OUTPUT : -**





**GRAPH : -**



**PROGRAM 11**

Implement Warshall’s algorithm using dynamic programming.

#include <stdio.h>

#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

void floydWarshall(int graph[][nV]) {

  int matrix[nV][nV], i, j, k;

  for (i = 0; i < nV; i++)

    for (j = 0; j < nV; j++)

      matrix[i][j] = graph[i][j];

  for (k = 0; k < nV; k++) {

    for (i = 0; i < nV; i++) {

      for (j = 0; j < nV; j++) {

        if (matrix[i][k] + matrix[k][j] < matrix[i][j])

          matrix[i][j] = matrix[i][k] + matrix[k][j];

      }

    }

  }

  printMatrix(matrix);

}

void printMatrix(int matrix[][nV]) {

int i,j;

  for ( i = 0; i < nV; i++) {

    for (j = 0; j < nV; j++) {

      if (matrix[i][j] == INF)

        printf("%4s", "INF");

      else

        printf("%4d", matrix[i][j]);

    }

    printf("\n");

  }

}

int main() {

  int graph[nV][nV] = {{0, 3, INF, 5},

             {2, 0, INF, 4},

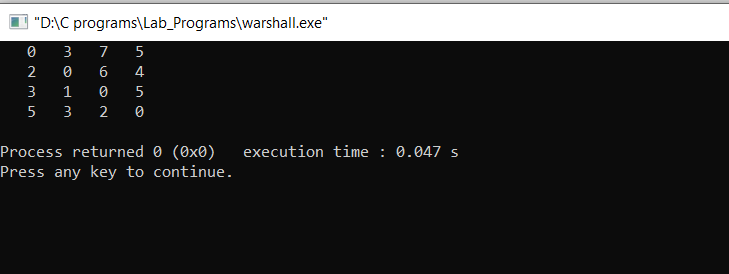
             {INF, 1, 0, INF},

             {INF, INF, 2, 0}};

  floydWarshall(graph);

}

**OUTPUT : -**



**PROGRAM 12**

Implement 0/1 Knapsack problem using dynamic programming.

#include<stdio.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 : ");

    scanf(" %d",&n);

    printf(" \nEnter the weight of the each item : ");

    for(i=1; i<=n; i++){

        scanf(" %d",&w[i]);

    }

    printf(" \nEnter the profit of each item : ");

    for(i=1; i<=n; i++){

        scanf(" %d",&p[i]);

    }

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

    scanf(" %d",&m);

    knapsack();

}

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){

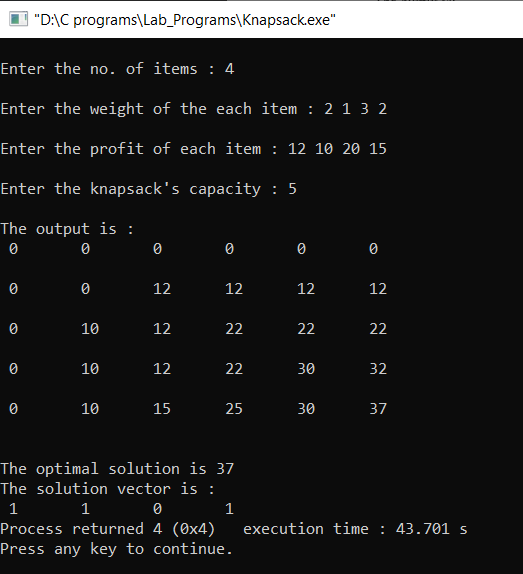
    if(x> y)

        return x;

    return y;

}

**OUTPUT : -**



**PROGRAM 13**

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

#include<stdio.h>

int a[10][10],n;

void floyds()

{

    int i,j,k;

    for(k=1; k<=n; k++)

    {

        for(i=1; i<=n; i++)

        {

            for(j=1; j<=n; j++)

            {

                a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

            }

        }

    }

    printf(" \n All pair shortest path matrix is : \n");

    for(i=1; i<=n; i++)

    {

        for(j=1; j<=n; j++)

        {

            printf(" %d\t",a[i][j]);

        }

        printf(" \n\n");

    }

}

int min(int x,int y)

{

    if(x< y)

    {

        return x;

    }

    else

    {

        return y;

    }

}

void main()

{

    int i,j;

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

    scanf(" %d",&n);

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

    for(i=1; i<=n; i++)

    {

        for(j=1; j<=n; j++)

        {

            scanf(" %d",&a[i][j]);

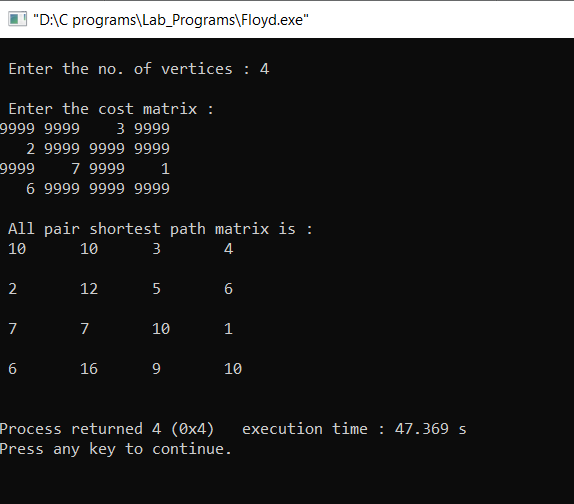
        }

    }

    floyds();

}

**OUTPUT : -**



**PROGRAM 14**

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

#include<stdio.h>

#include<conio.h>

#include<process.h>

void prims();

int c[10][10],n;

void main()

{

    int i,j;

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

    scanf(" %d",&n);

    printf("\n Enter 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;

        }

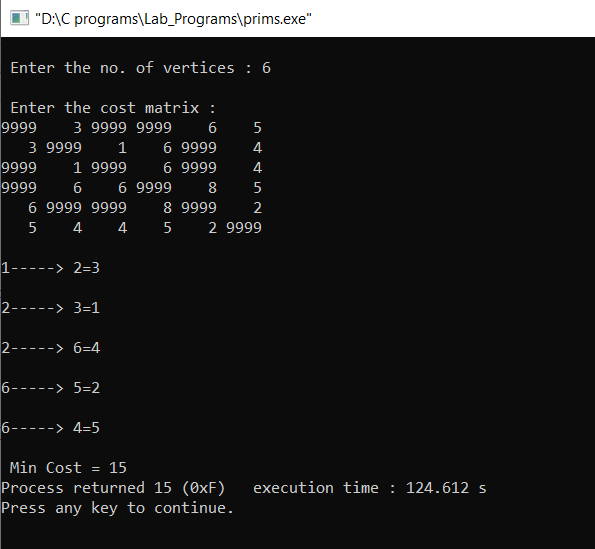
        c[u][v]=c[v][u]=9999;

    }

    printf("\n Min Cost = %d",mincost);

}

**OUTPUT : -**



**PROGRAM 15**

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

#include<stdio.h>

#include<conio.h>

void kruskals();

int c[10][10],n;

void main()

{

    int i,j;

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

    scanf("%d",&n);

    printf("\n Enter 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;

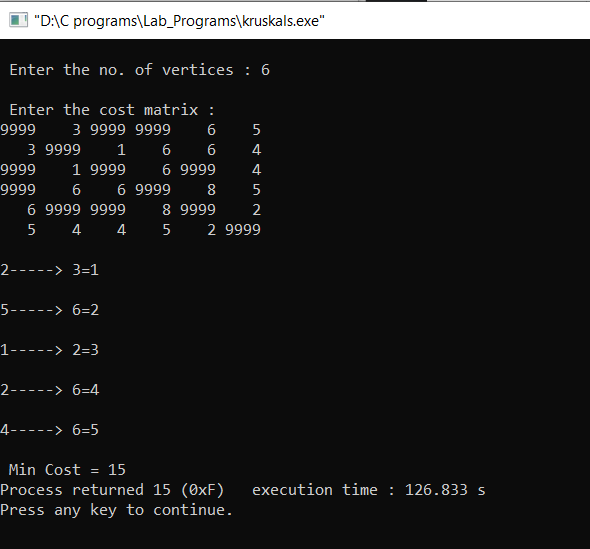
        }

        c[a][b]=c[b][a]=9999;

    }

    printf("\n Min Cost = %d",mincost);

}

**OUTPUT : -**

**PROGRAM 16**

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

#include<stdio.h>

#include<conio.h>

void dijkstras();

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

void main()

{

    int i,j;

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

    scanf("%d",&n);

    printf("\n Enter 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:\t");

    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("\n The shortest distance is : \n");

    for(j=1; j<=n; j++)

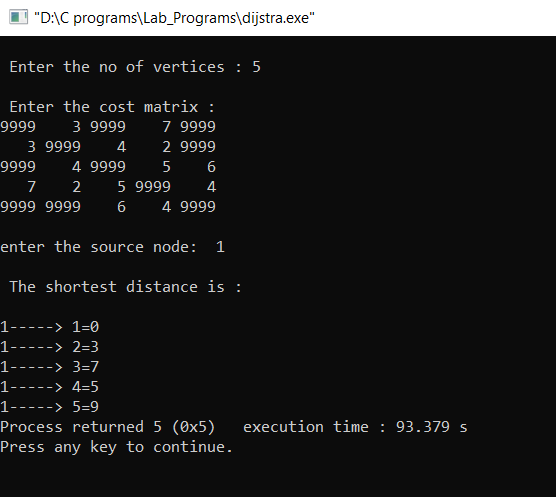
    {

        printf("\n%d-----> %d=%d",src,j,dist[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.

#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("\nEnter the number of elements : ");

    scanf(" %d", &n);

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

    for(i=0; i< n; i++)

        scanf(" %d", &w[i]);

    printf("\nEnter the required sum : ");

    scanf(" %d",&d);

    for(i=0; i< n; i++)

        sum+=w[i];

    if(sum< d)

    {

        printf(" No solution exists\n");

        return;

    }

    printf("\nThe solution is : ");

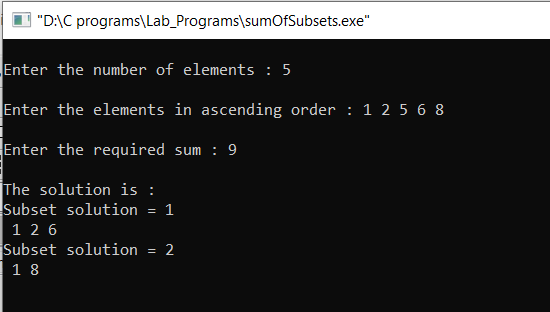
    count=0;

    subset(0,0,sum);

    getch();

}

**OUTPUT : -**



**PROGRAM 18**

Implement “N-Queens Problem” using Backtracking.

#include<stdio.h>

#include<conio.h>

void nqueens(int n);

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

void main()

{

    int n;

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

    scanf("%d",&n);

    nqueens(n);

}

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;

}

**OUTPUT : -**

