**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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



**LAB REPORT**

**On**

# ANALYSIS AND DESIGN OF ALGORITHMS

***Submitted by***

**NIMISHAMBA.G S(1BM21CS406)**

***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 NIMISHAMBA**.G.S(1BM21CS406),** who is a 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 - (19CS34PCADA)** work prescribed for the said degree.

**REKHA.G S Dr. Jyothi S Nayak**

Associate Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

## Index Sheet

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| **01** | **Write a recursive program to**  **a. Solve Towers-of-Hanoi problem b. To find GCD** | **05-07** |
| **02** | **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.** | **08-13** |
| **03** | **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**. | **14-18** |
| **04** | **Write program to do the following:**   1. **Print all the nodes reachable from a given starting node in a digraph using BFS method.** 2. **Check whether a given graph is connected or not using DFS method.** | **19-24** |
| **05** | **Sort a given set of N integer elements using Insertion Sort**  **technique and compute its time taken.** | **25-27** |
| **06** | **Write program to obtain the Topological ordering of vertices**  **in a given digraph.** | **28-30** |
| **07** | **Implement Johnson Trotter algorithm to generate**  **permutations.** | **31-36** |
| **08** | **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.** | **37-43** |
| **09** | **Sort a given set of N integer elements using Quick Sort**  **technique and compute its time taken.** | **44-47** |
| **10** | **Sort a given set of N integer elements using Heap Sort technique and compute its time taken.** | **48-50** |
| **11** | **Implement Warshall’s algorithm using dynamic programming**. | **50-53** |
| **12** | **Implement 0/1 Knapsack problem using dynamic programming.** | **53-54** |
| **13** | **Implement All Pair Shortest paths problem using Floyd’s algorithm.** | **54-56** |
| **14** | **Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm** | **56-58** |
| **15** | **Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm** | **58-59** |
| **16** | **From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm** | **59-61** |
| **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.** | **61-63** |
| **18** | **Implement “N-Queens Problem” using Backtracking** | **63-64** |

**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. |

# LAB PROGRAM-01

## Write a recursive program to

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

## To find GCD

1. #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);

}

}

void main( )

{

int disk; int moves; clrscr();

printf("Enter the number of disks you want to play with:"); scanf("%d",&disk);

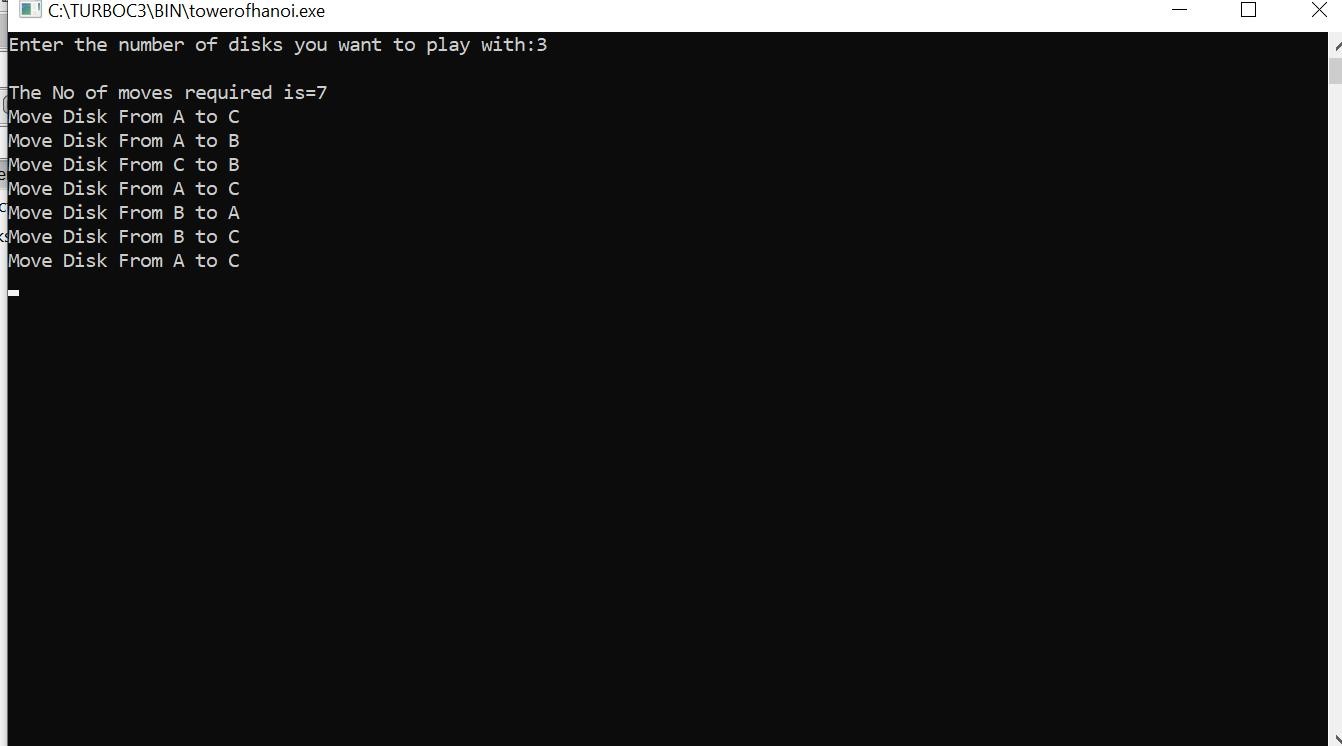
moves=pow(2,disk)-1;

printf("\nThe No of moves required is=%d \n",moves); hanoi(disk,'A','C','B');

getch( );

}

## OUTPUT:



1. #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)

{

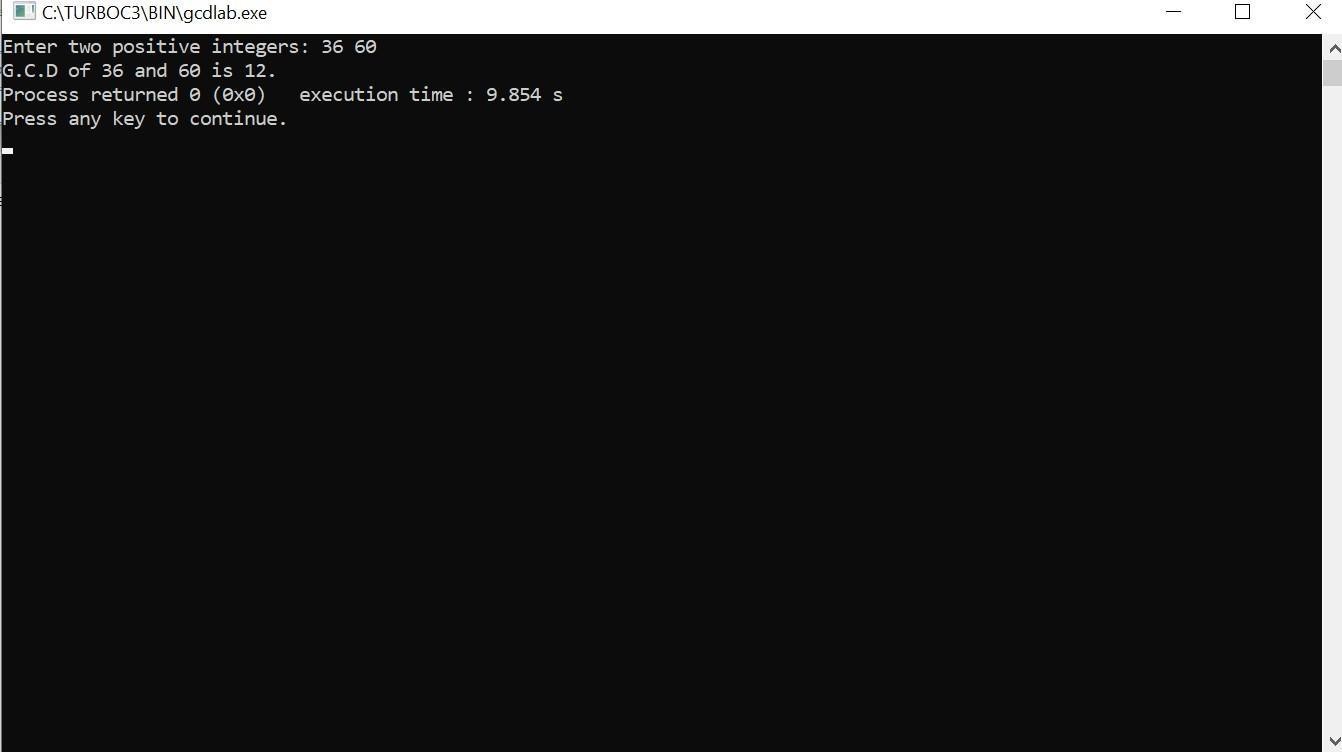
if (n2 != 0)

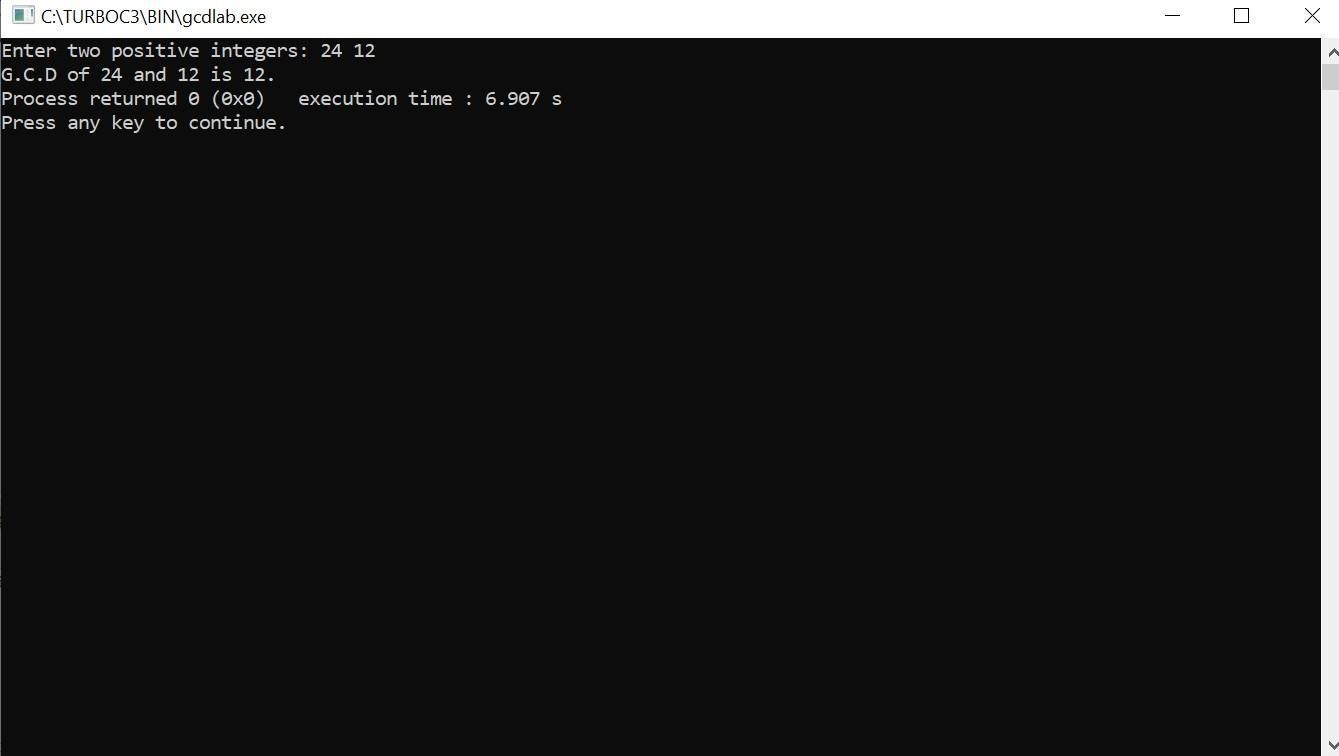
return hcf(n2, n1%n2); else

return n1;

}

**OUTPUT:**





# LAB PROGRAM-02

## 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); void bub\_sort(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; //Insering numbers in Ascending order

}

key=a[n-1]; //Last element of the aray start=clock();

//bub\_sort(a,n); //Sorting numbers in Ascending order using Bubble sort

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();

}

}

void bub\_sort(int a[],int n)

{

int i,j,temp; for(i=0;i<=n-2;i++)

{

for(j=0;j<=n-2-i;j++)

{

if(a[j]>a[j+1])

{

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

}

}

}

}

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

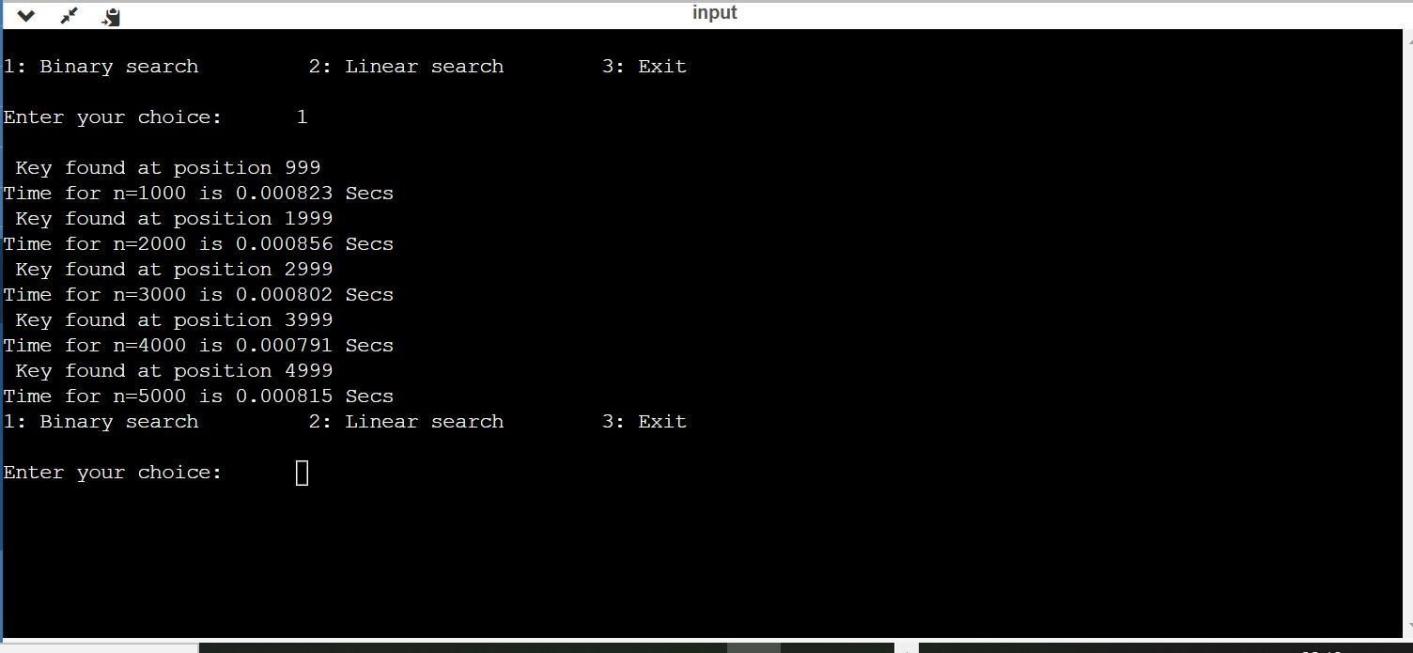
{

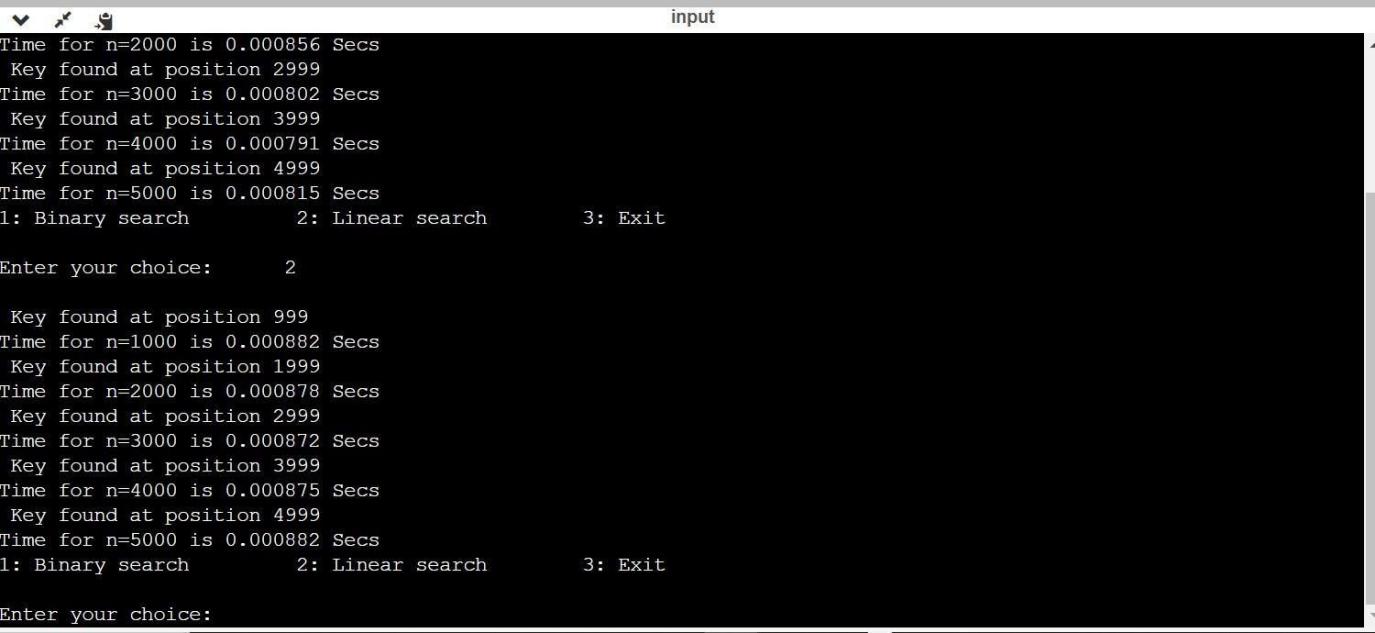
return lin\_srch(a,i+1,high,key);

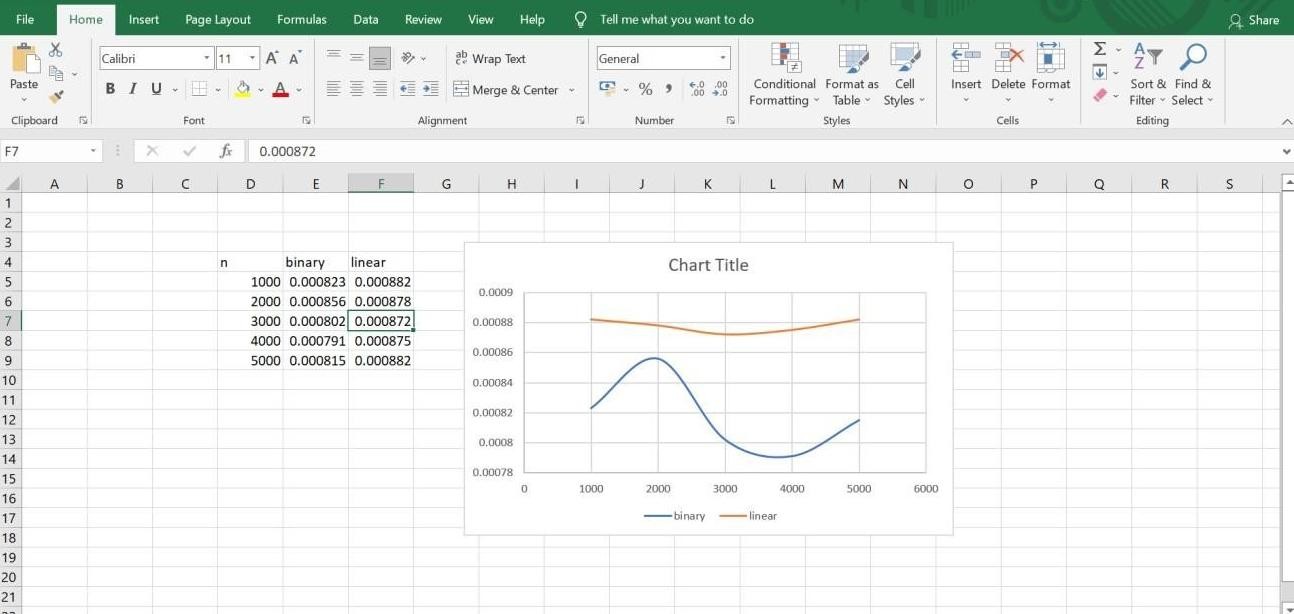
}

}

**OUTPUT:**







# LAB PROGRAM-03

## 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>

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 1000 to 10000");

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=1000;

while(n<=10000)

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

{

//a[i]=random(1000); a[i]=n-i;

}

start=clock(); selsort(n,a);

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(); 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;

}

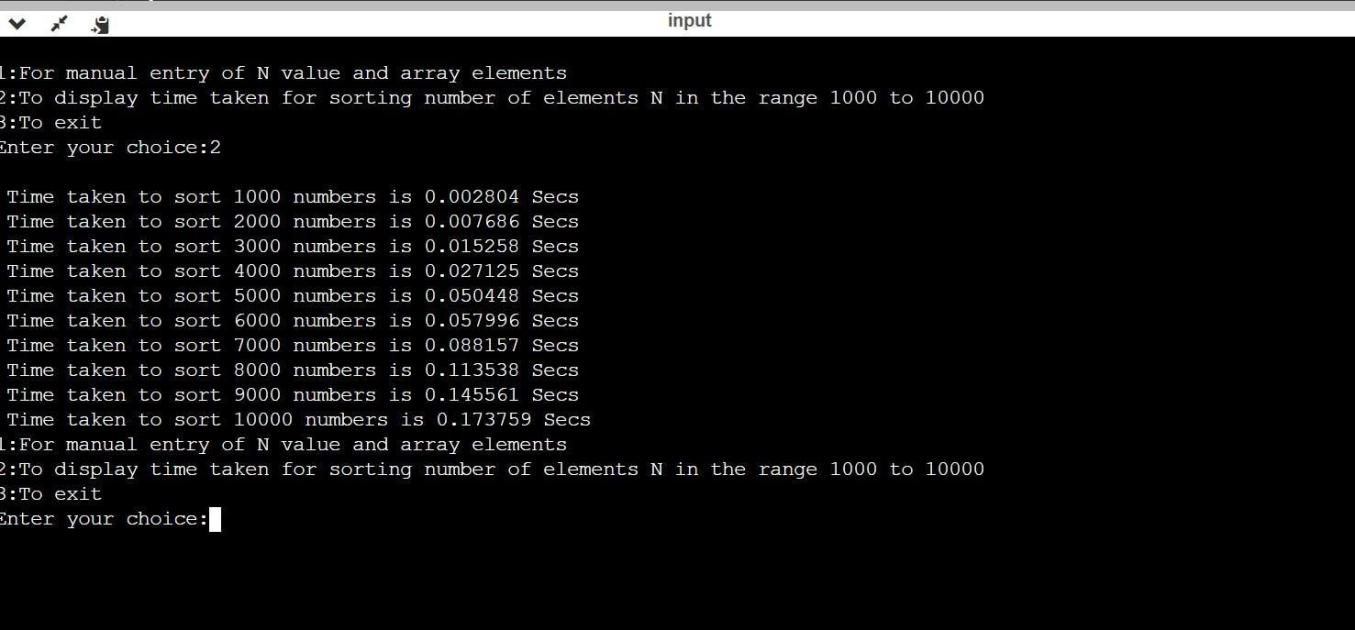
}

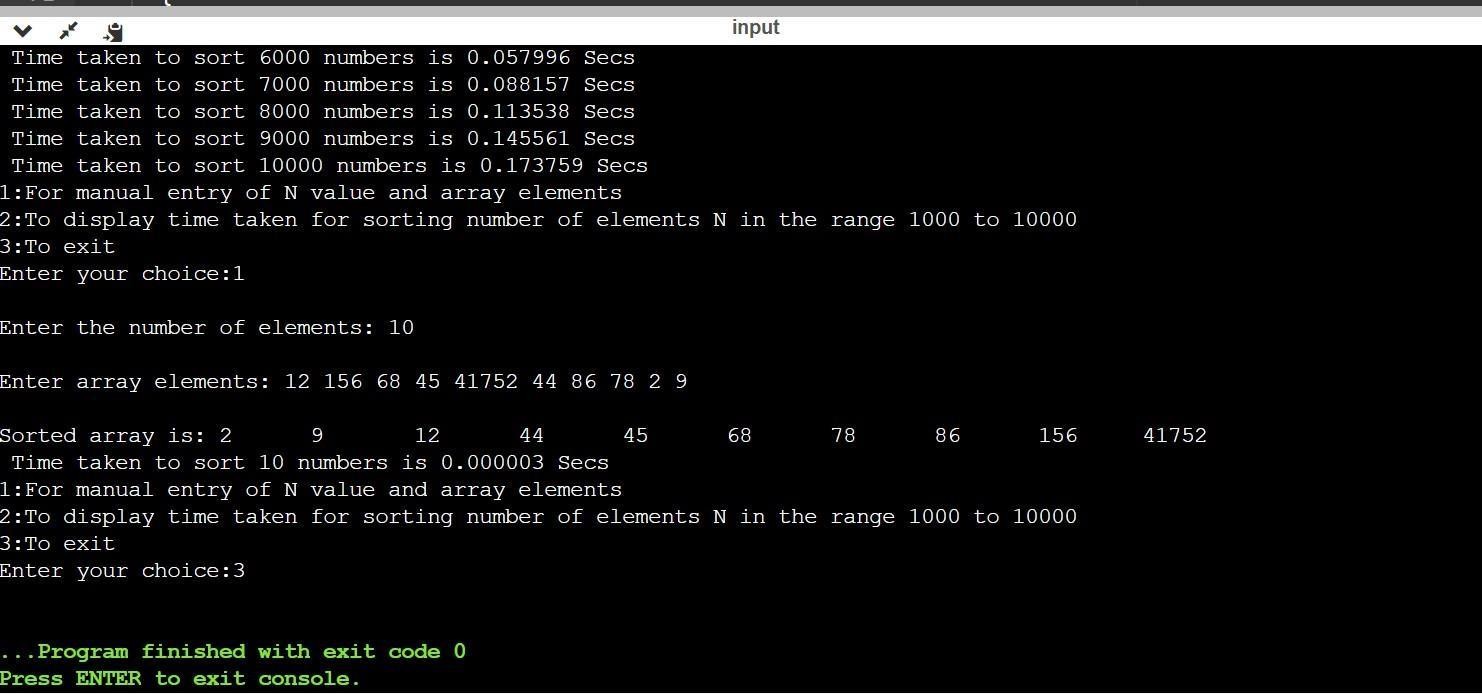
t=a[i]; a[i]=a[pos]; a[pos]=t;

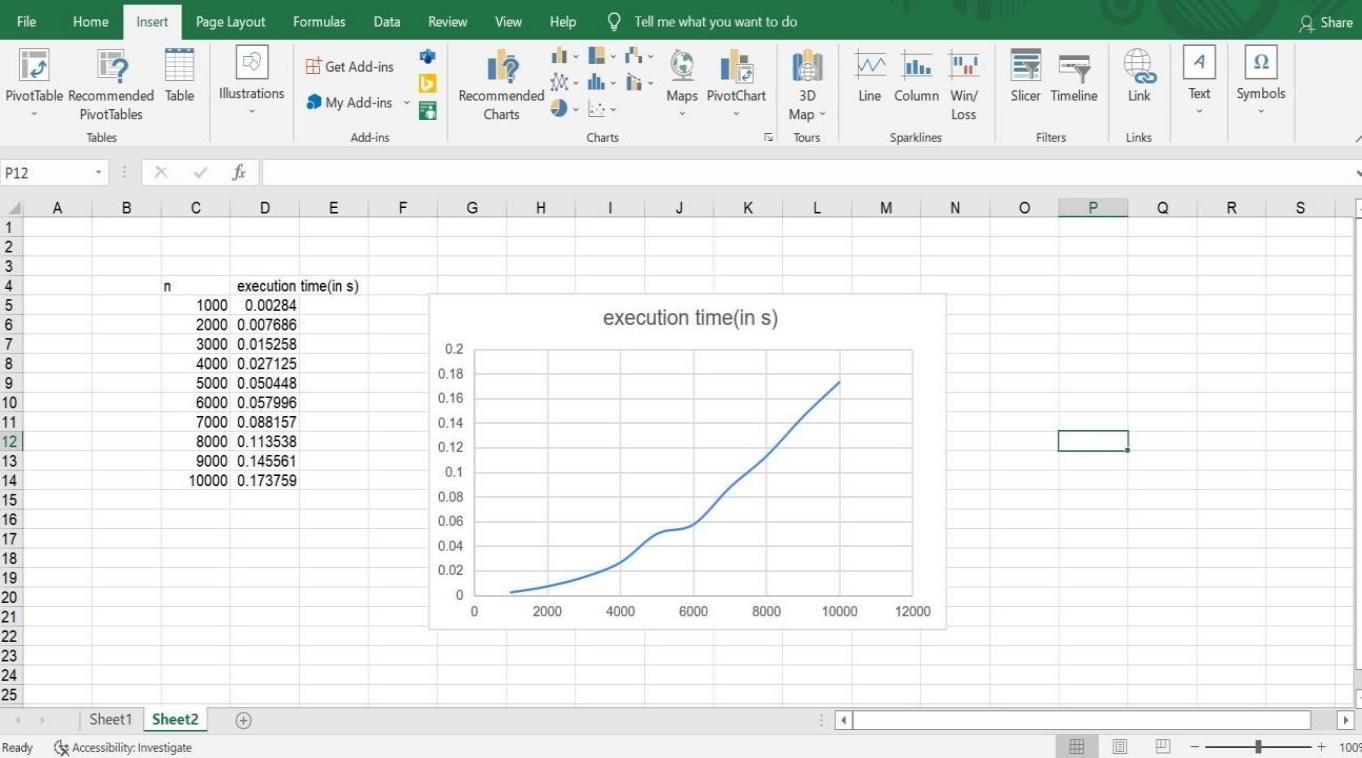
}

}

**OUTPUT:**







# LAB PROGRAM-04

## Write program to do the following:

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

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

1. #include<stdio.h> #include<conio.h> int a[10][10],n; void bfs(int);

int main()

{

int i,j,src;

printf("\n enter the no of nodes:\t"); 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("\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("\nnode %d is not reachable\n",j);

}

else

{

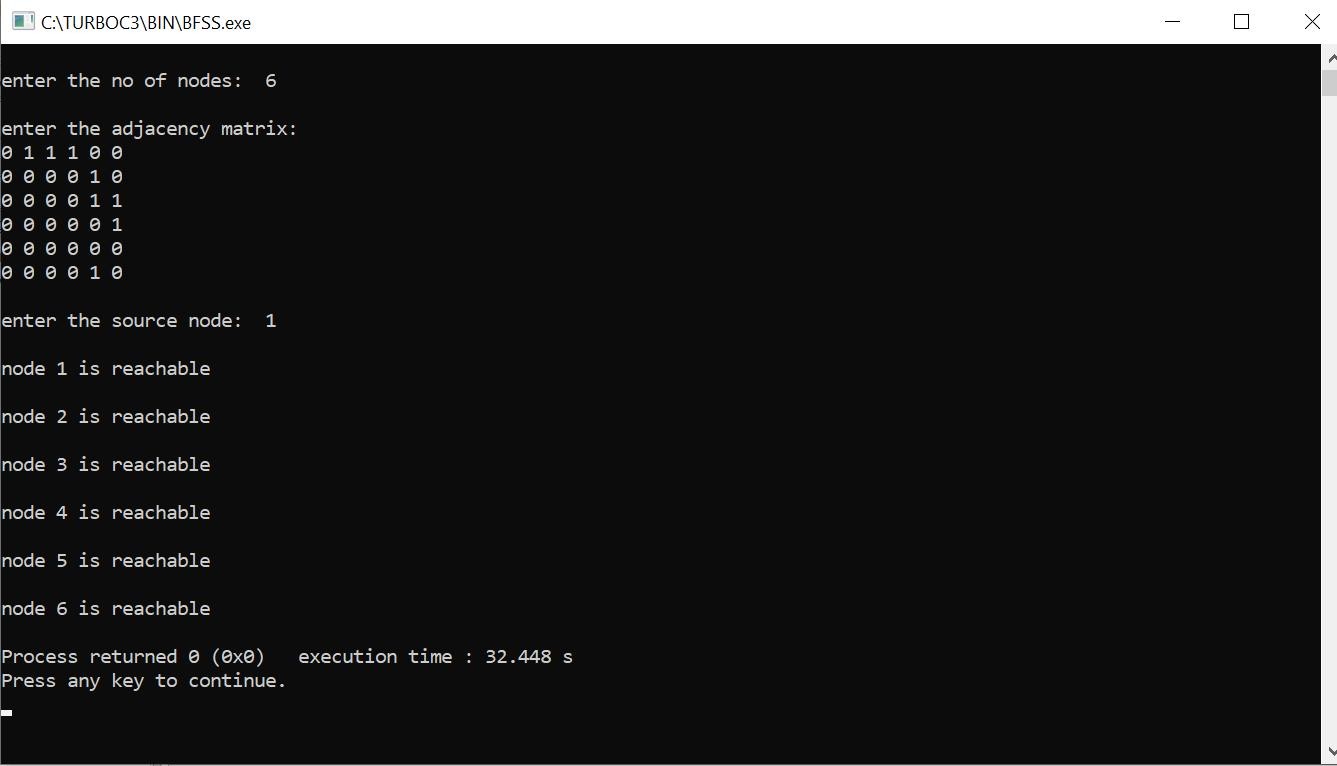
printf("\nnode %d is reachable\n",j);

}

}

}

## OUTPUT:



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

int a[10][10],n,vis[10]; int dfs(int);

void 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");

}

getch();

}

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;

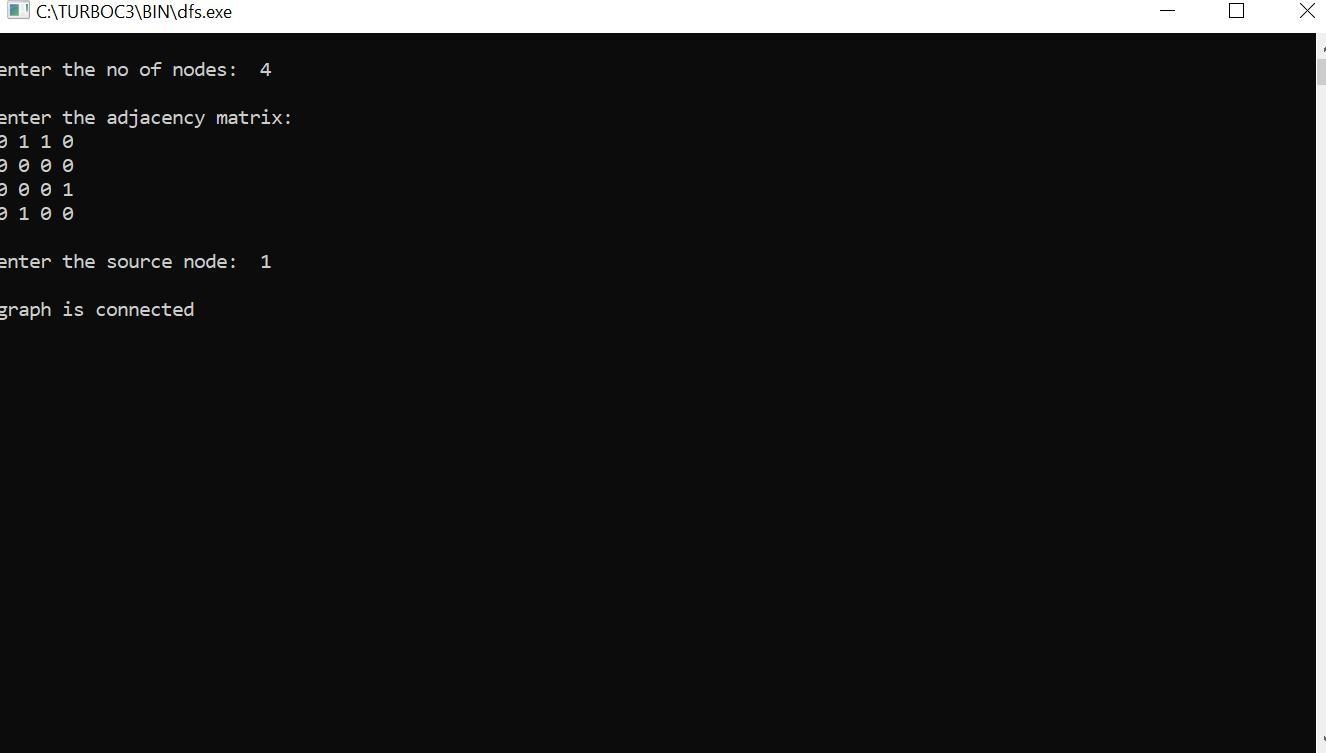
}

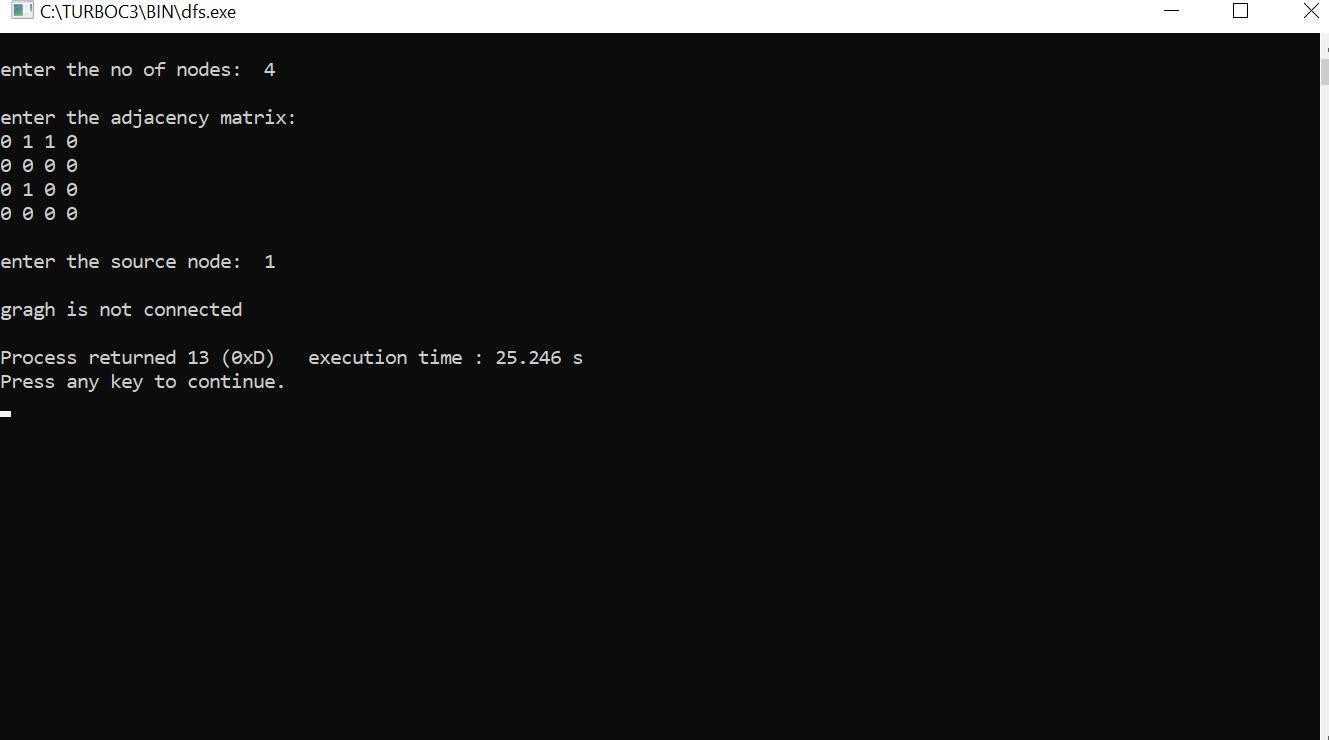
}

return 1;

}

**OUTPUT:**





# LAB PROGRAM-05

## Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator.

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

void insertionsort(int n,int a[])

{

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

{

val=a[i]; j=i-1;

while(j>=0 && a[j]>val)

{

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

j--;

}

a[j+1]=val;

}

}

void main()

{

clock\_t start,end;

int a[15500],i,j,temp; int n=100; while(n<1300)

{

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

{

a[i]=n-i;

}

start=clock(); insertionsort(n,a); for(j=0; j<500000; j++)

{

temp=38/600;

}

end=clock();

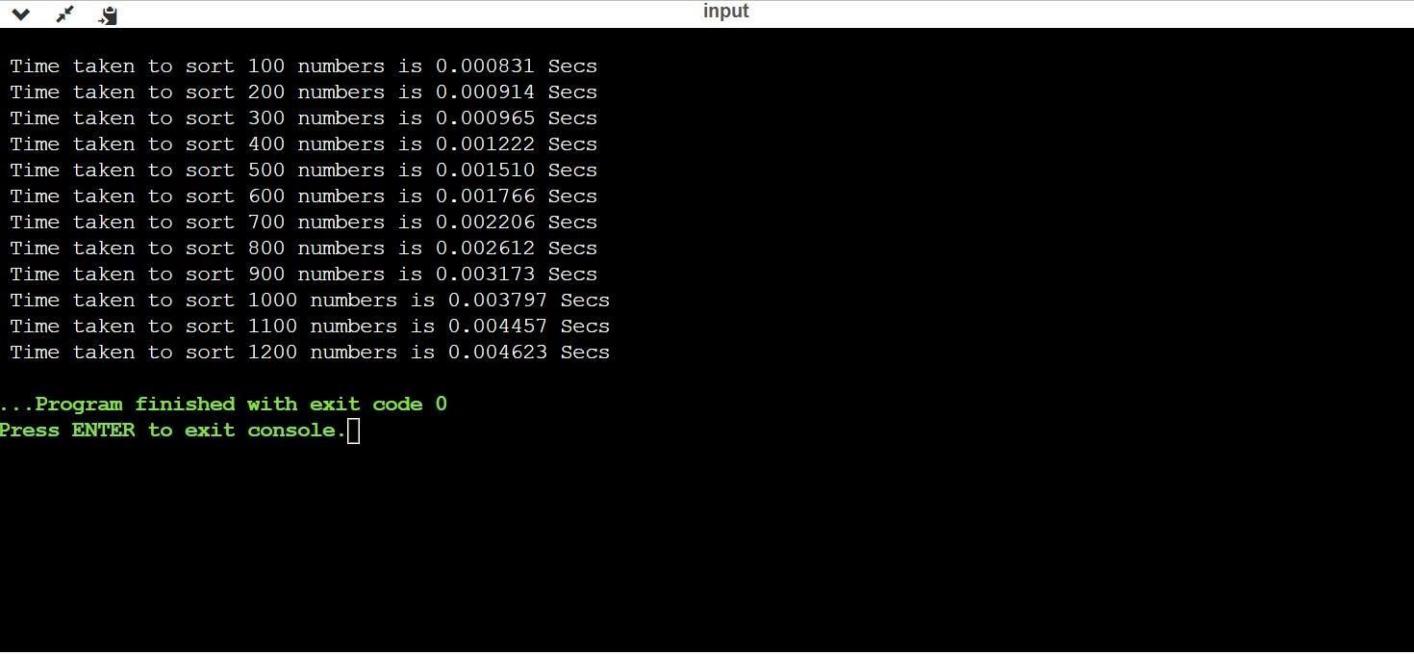
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

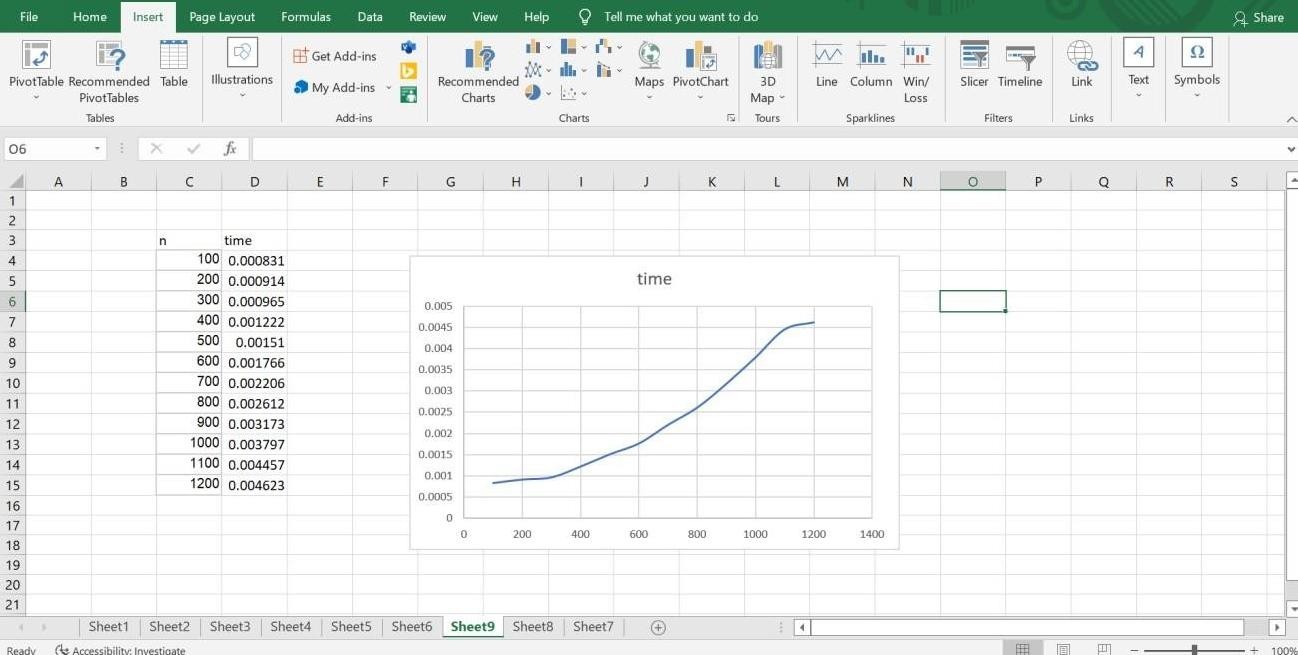
n=n+100;

}

}

**Output:**





# LAB PROGRAM-06

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

#include<stdio.h> void dfs(int);

int a[10][10],n,e[10],vis[10],j=0;

int main()

{

int m, u, v, i;

printf("Enter number of vertices : "); scanf("%d",&n);

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

{

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

{

a[i][j] = 0;

}

}

printf("Enter number of edges : "); scanf("%d",&m);

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

{

printf("Enter an edge : "); scanf("%d%d",&u,&v);

a[u][v] = 1;

}

for(i=1;i<=n;i++) vis[i] = 0;

j=0;

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

{

if(vis[i] == 0)

dfs(i);

}

printf("Topological order : "); for(i=n-1; i>=0;i--)

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

return 0;

}

void dfs(int v)

{

int i; vis[v] = 1;

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

{

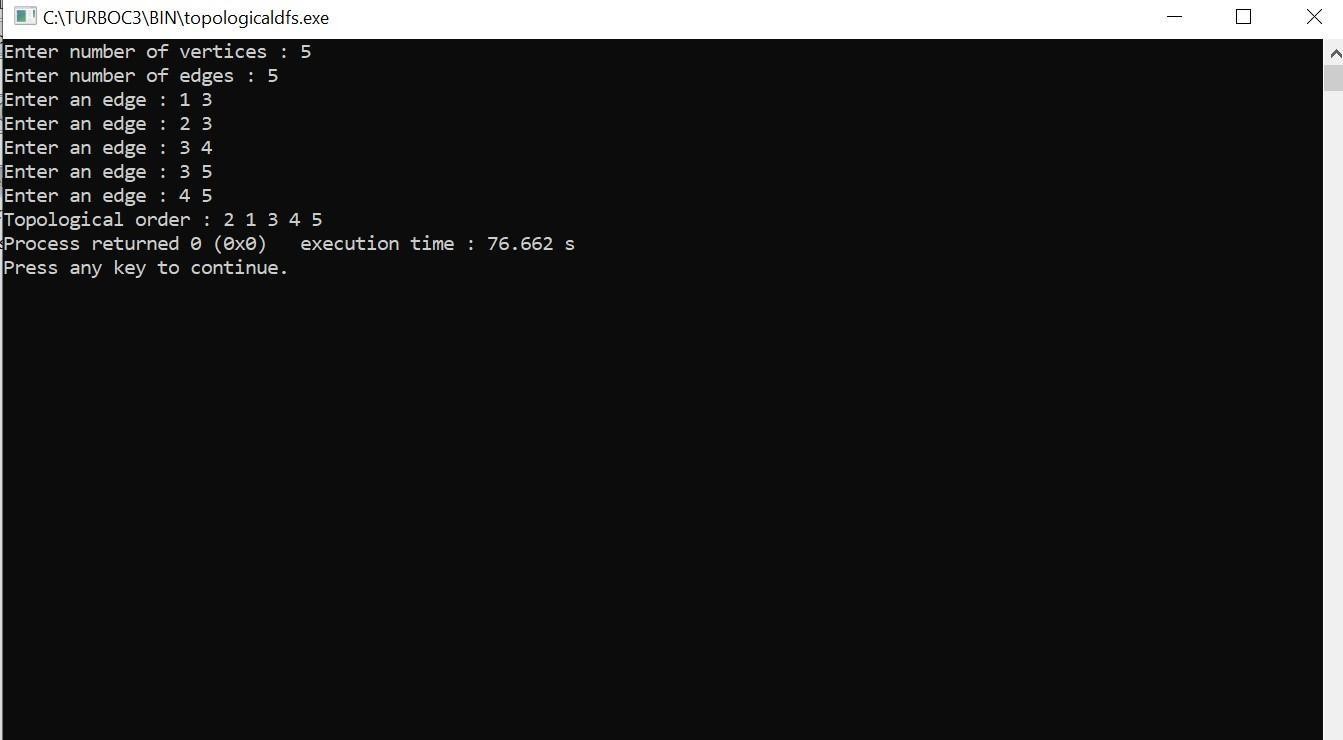
if(a[v][i] == 1 && vis[i] == 0) dfs(i);

}

e[j++] = v;

}

**Output:**



# LAB PROGRAM-07

## Implement Johnson Trotter algorithm to generate permutations.

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

int LEFT\_TO\_RIGHT = 1; int RIGHT\_TO\_LEFT = 0;

int searchArr(int a[], int n, int mobile)

{for (int i = 0; i < n; i++) if (a[i] == mobile) return i + 1;

}

int getMobile(int a[], int dir[], int n)

{int mobile\_prev = 0, mobile = 0; for (int i = 0; i < n; i++) {

if (dir[a[i]-1] == RIGHT\_TO\_LEFT && i!=0)

{if (a[i] > a[i-1] && a[i] > mobile\_prev)

{ mobile = a[i]; mobile\_prev = mobile;

}

}

if (dir[a[i]-1] == LEFT\_TO\_RIGHT && i!=n-1) {

if (a[i] > a[i+1] && a[i] > mobile\_prev)

{

mobile = a[i]; mobile\_prev = mobile;

}

}

}

if (mobile == 0 && mobile\_prev == 0) return 0;

else

return mobile;

}

int printOnePerm(int a[], int dir[], int n)

{

int mobile = getMobile(a, dir, n); int pos = searchArr(a, n, mobile);

if (dir[a[pos - 1] - 1] == RIGHT\_TO\_LEFT)

{

printf("\n"); int temp;

temp = a[pos-1] ;

a[pos-1] = a[pos-2]; a[pos-2]= temp;

}

else if (dir[a[pos - 1] - 1] == LEFT\_TO\_RIGHT)

{

printf("\n"); int temp;

temp = a[pos] ; a[pos] = a[pos-1]; a[pos-1]= temp;

}

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

{

if (a[i] > mobile)

{

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT) dir[a[i] - 1] = RIGHT\_TO\_LEFT;

else if (dir[a[i] - 1] == RIGHT\_TO\_LEFT) dir[a[i] - 1] = LEFT\_TO\_RIGHT;

}

}

for (int i = 0; i < n; i++) printf(" %d", a[i]);

}

int fact(int n)

{

int res = 1; int i;

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

return res;

}

void printPermutation(int n)

{

int a[n]; int dir[n];

printf("\n");

printf("\n");

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

{

a[i] = i + 1;

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

}

printf("\n");

for (int i = 0; i < n; i++) dir[i] = RIGHT\_TO\_LEFT;

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

printOnePerm(a, dir, n); printf("\n");

}

int main()

{

int n;

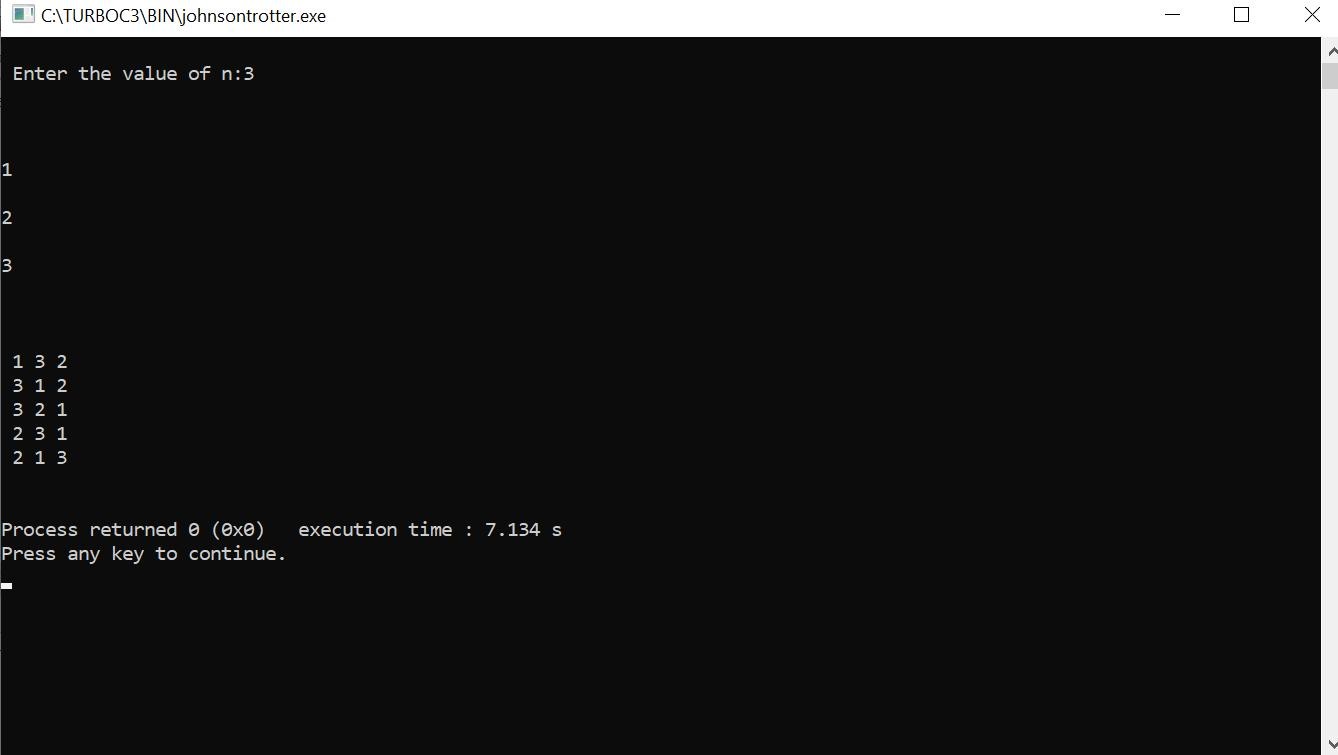
printf("\n Enter the value of n:N"); scanf("%d",&n);

printf("\n"); printPermutation(n); printf("\n");

return 0;

}

**Output:**



# LAB PROGRAM-08

## 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); void 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(); split(a,0,n-1); 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]=n-i;

}

start=clock(); split(a,0,n-1);

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 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,high);

}

}

void combine(int a[],int low,int mid,int high)

{

int c[15000],i,j,k; i=k=low; j=mid+1;

while(i<=mid &&j<=high)

{

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

{

c[k]=a[i];

++k;

++i;

}

else

{

c[k]=a[j];

++k;

++j;

}

}

if(i>mid)

{

while(j<=high)

{

c[k]=a[j];

++k;

++j;

}

}

if(j>high)

{

while(i<=mid)

{

c[k]=a[i];

++k;

++i;

}

}

for(i=low;i<=high;i++)

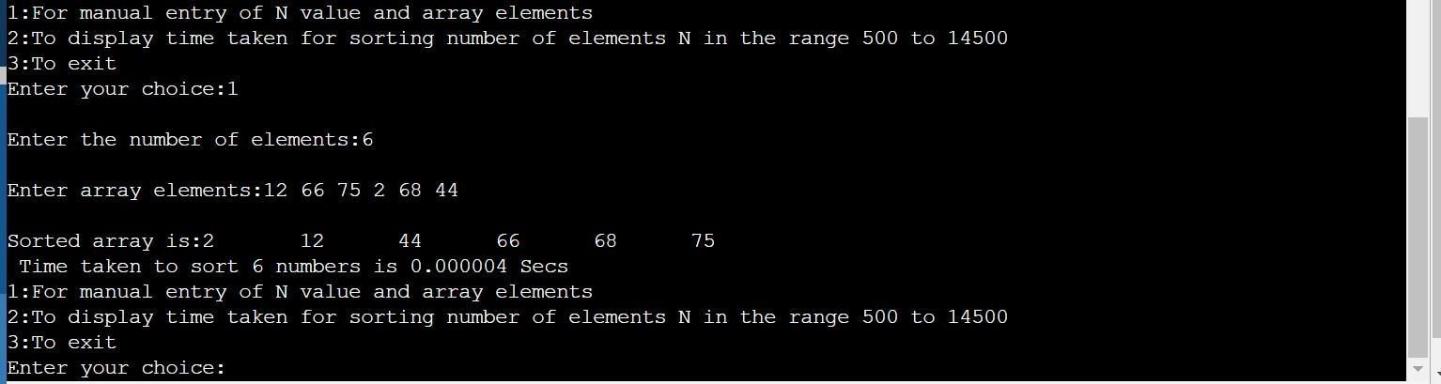
{

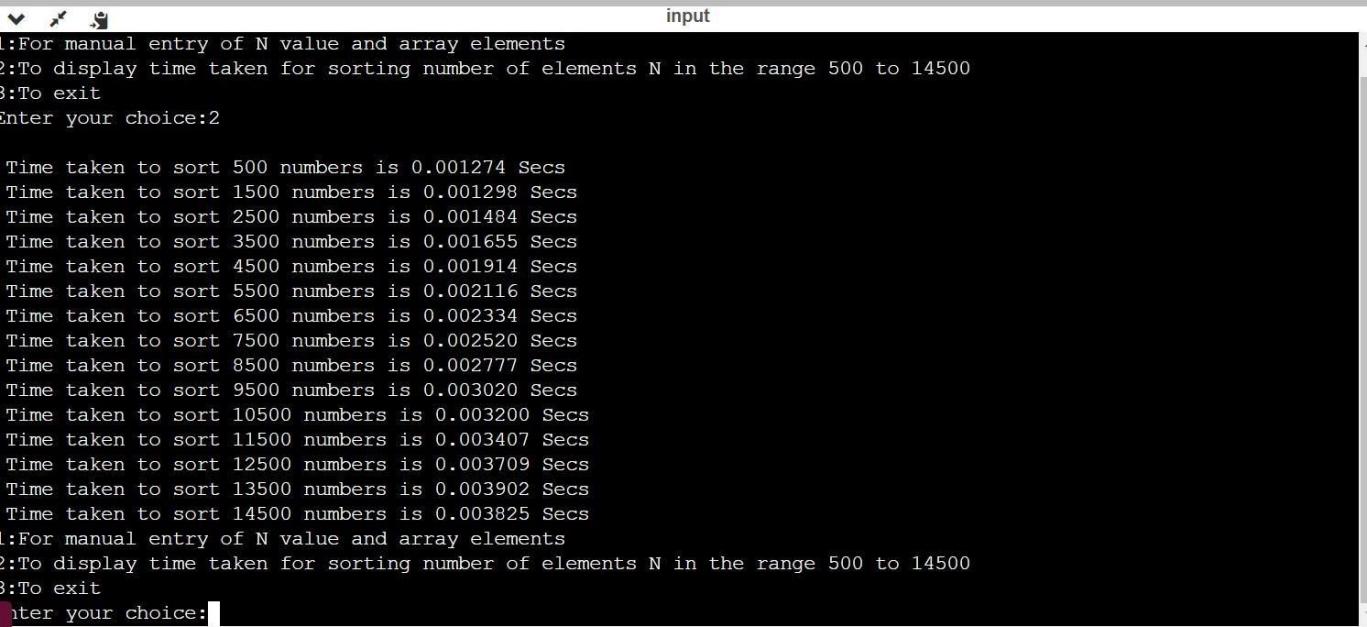
a[i]=c[i];

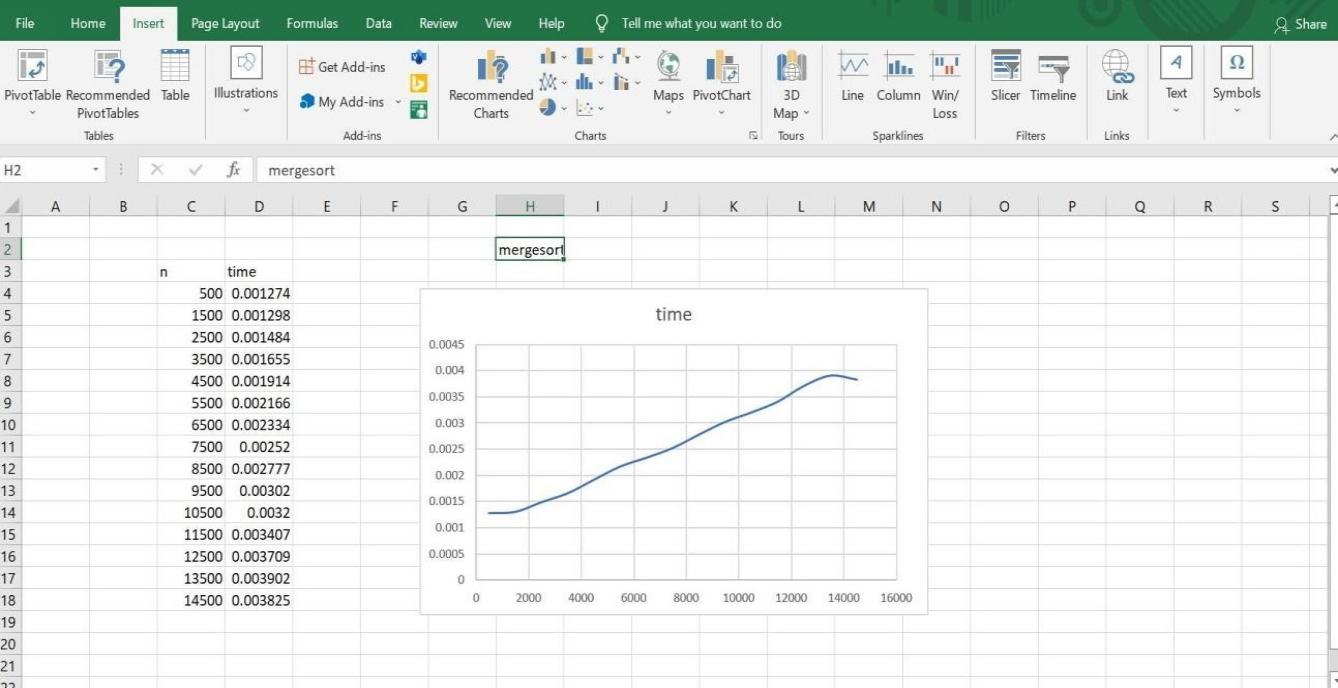
}

}

**Output:**







# LAB PROGRAM-09

## 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> #define MAXINT 2000 void delay(int n)

{

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

}

}

void quickSort(int number[],int first,int last){int i,j,pivot,temp;

if(first<last){ pivot=first; i=first; j=last;

while(i<j){ while(number[i]<=number[pivot]& &i<last){i++;

}

while(number[j]>number[pivot]&&j>first){ 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()

{

clock\_t start,end; int i,datasize=1; long int n=10000; int \*a;

while(datasize<=20){

a=(int \*)calloc(n,sizeof(int)); if(a==NULL){

printf("Insufficiant Memory"); exit(0);

}

for(i=0;i<=n-

1;i++){ a[i]=rand()%MAXI NT;

}

start=clock(); quickSort(a,0,n-1); end=clock(); free(a);

if((end-start)!=0){ printf("\n%d\t%f",n,(double)(end-start)/CLK\_TCK); datasize++;

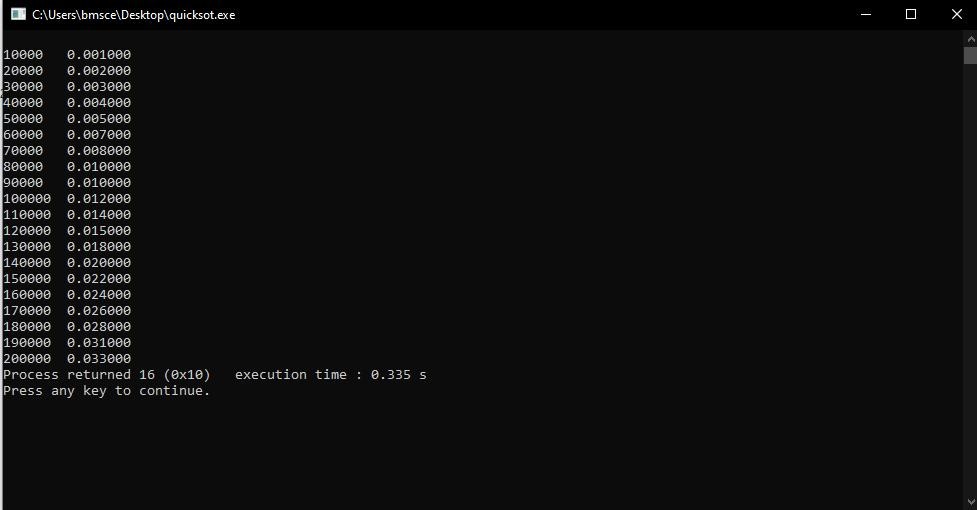
} n+=10000;

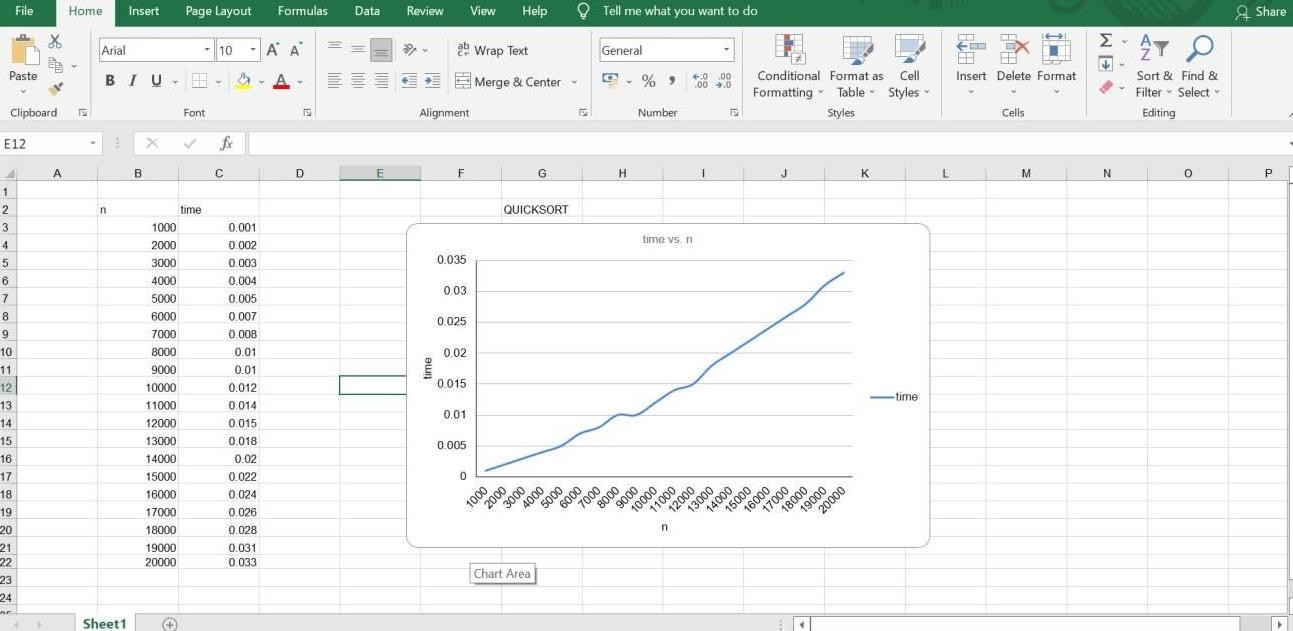
}

return;

}

# Output:





# LAB 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>

#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("\n 1: For manual entry of N values 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 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=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]=n-i;

}

start=clock();

heapSort(a, n);

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

}

}

}

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--)

{

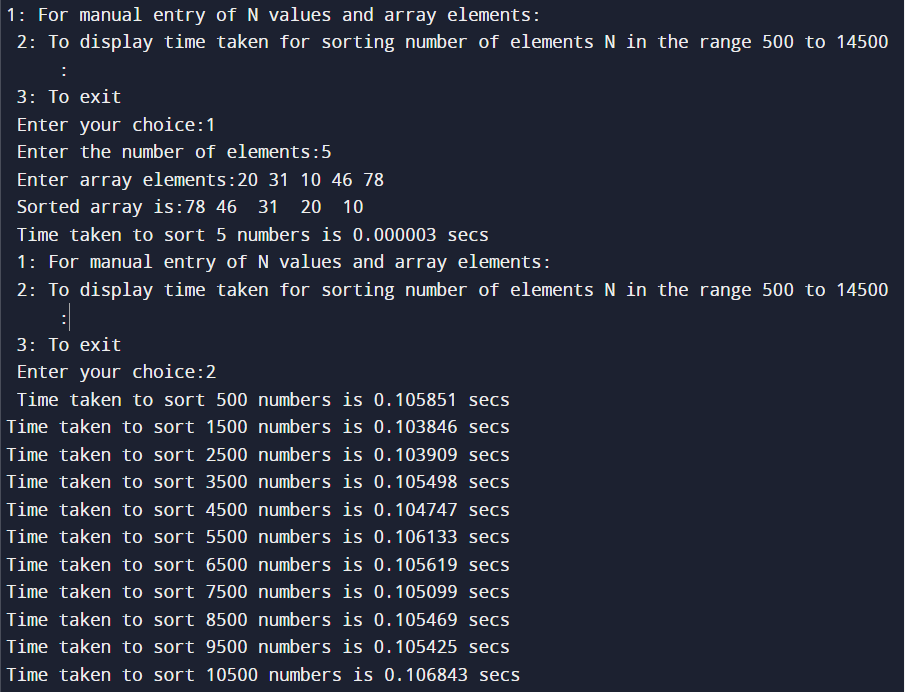
swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

**OUTPUT:**

****

# LAB PROGRAM-11

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

#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]= {

0

}

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

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

scanf("%d",&n);

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

scanf("%d",&e);

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

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

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

p[u][v]=1;

}

printf("\n Matrix of input data: \n");

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

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

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

printf("\n");

}

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]);

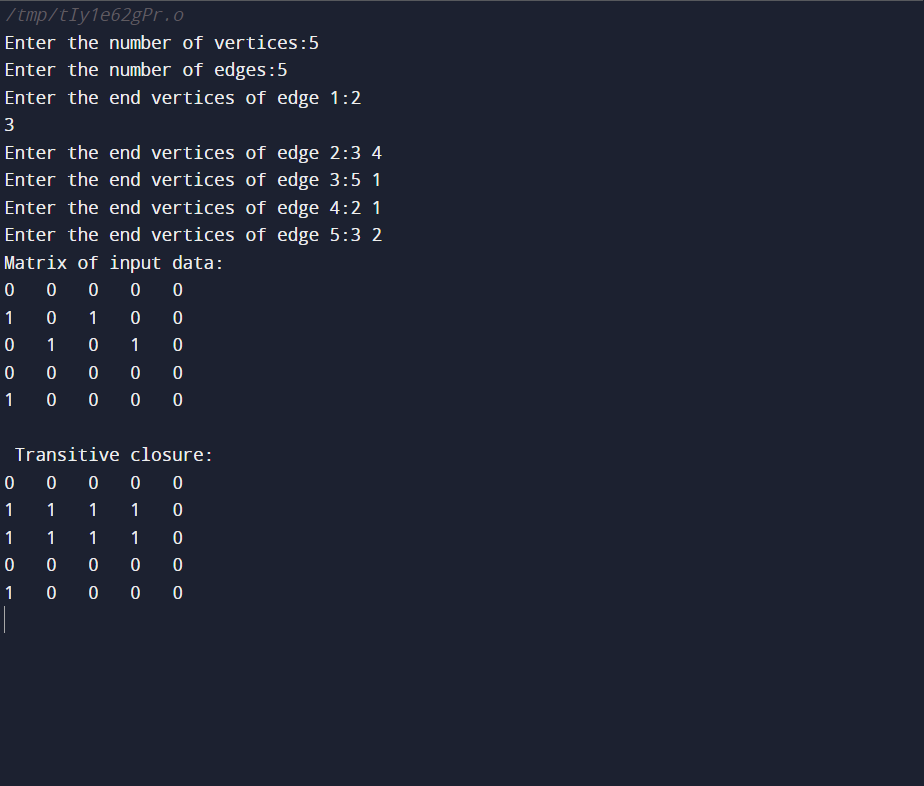
printf("\n");

}

getch();

}

**OUTPUT:**

****

# LAB 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("\n enter the no. of items:\t");

scanf("%d",&n);

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

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

{

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

}

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

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

{

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

}

printf("\n enter the knapsack's capacity:\t ");

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("\n the 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

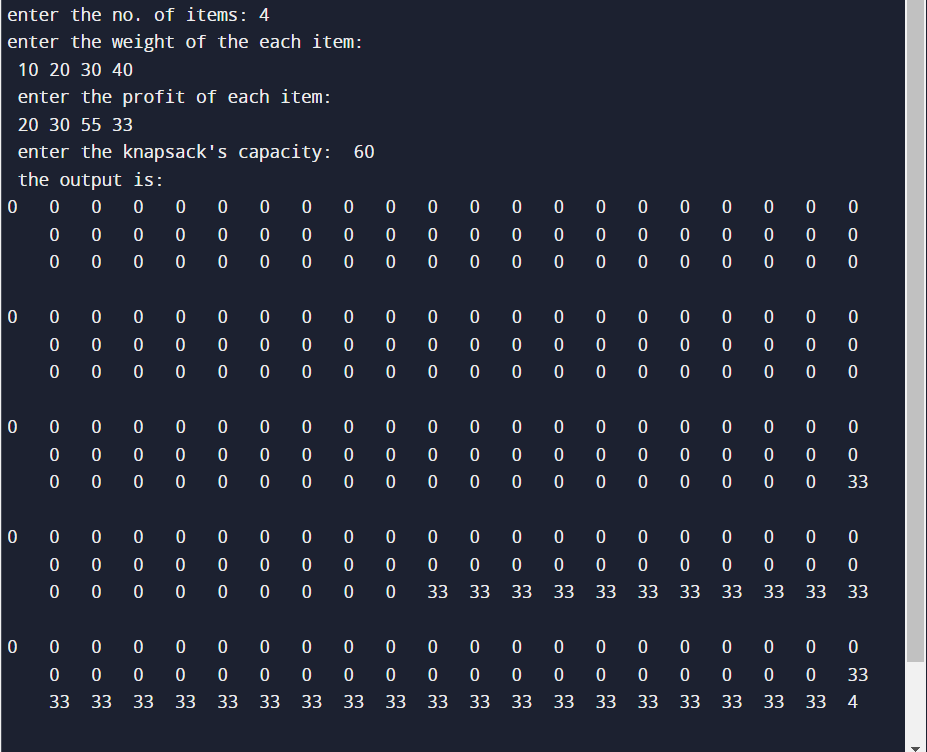
{

return y;

}

}

**OUTPUT:**

****

# LAB PROGRAM-13

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

#include<stdio.h>

int a[10][10],n;

void floyds();

int min(int,int);

void main()

{

int i,j;

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

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();

}

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

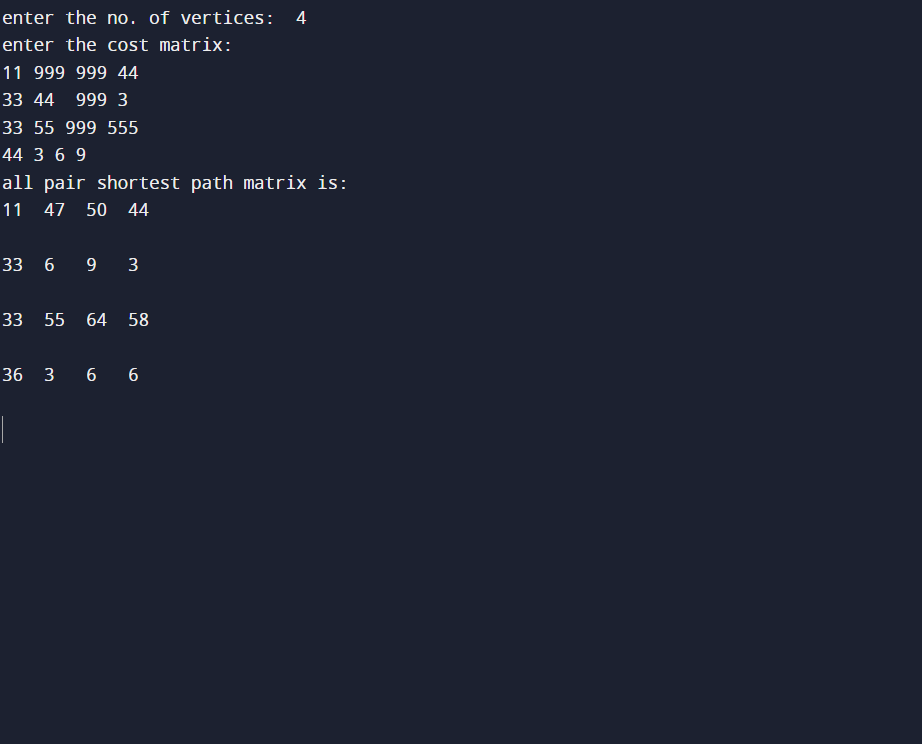
{

return y;

}

}

**OUTPUT:**

****

# LAB PROGRAM-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:\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]);

}

}

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;

}

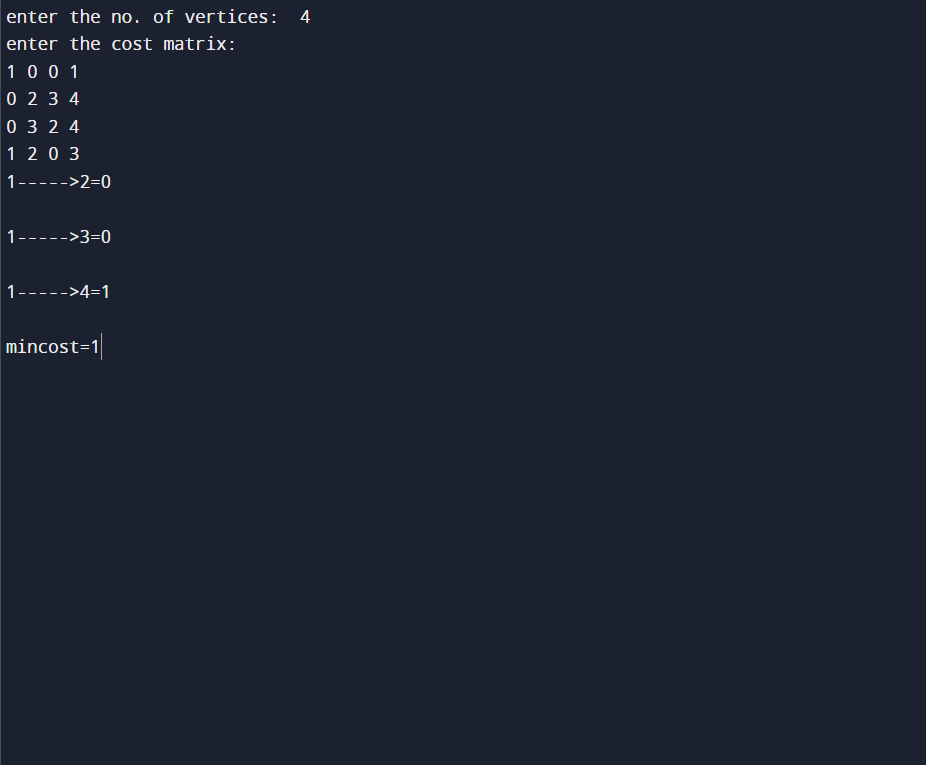
c[u][v]=c[v][u]=9999;

}

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

}

**OUTPUT:**

****

# LAB PROGRAM-15

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

#include<stdio.h>

void kruskals();

int c[10][10],n;

void main()

{

int i,j;

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

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;

}

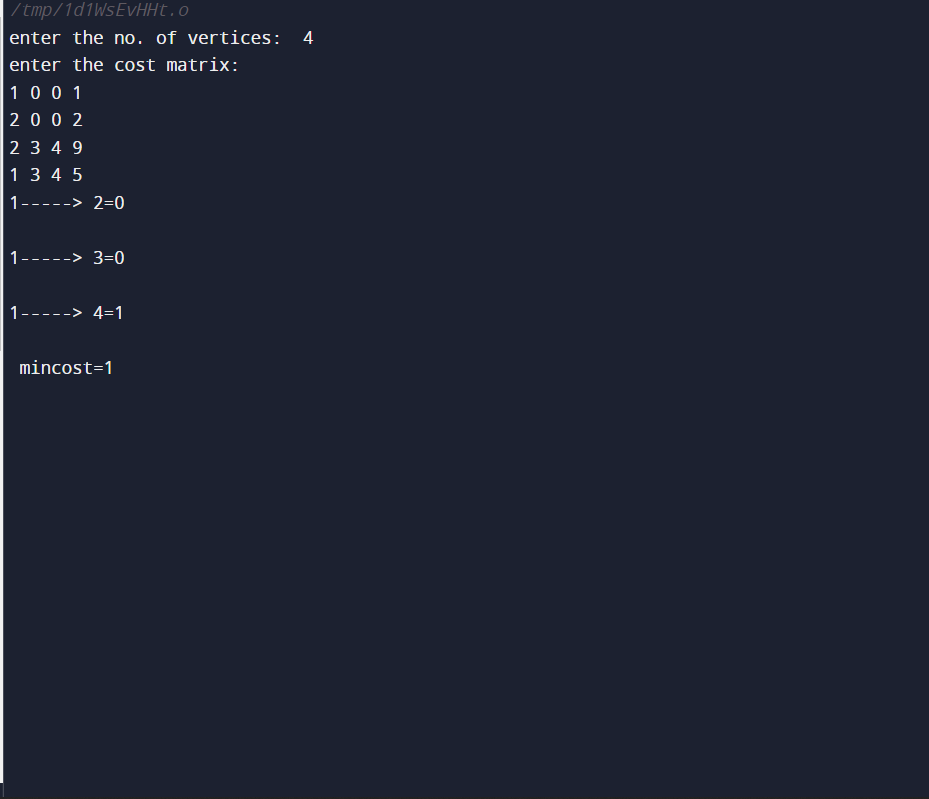
c[a][b]=c[b][a]=9999;

}

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

}

**OUTPUT:**

****

# LAB PROGRAM-16

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

#include<stdio.h>

#define infinity 999

void dij(int n,int v,int cost[10][10],int dist[100])

{

int i,u,count,w,flag[10],min;

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

flag[i]=0,dist[i]=cost[v][i];

count=2;

while(count<=n)

{

min=99;

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

if(dist[w])

min=dist[w],u=w;

flag[u]=1;

count++;

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

if(dist[u]+cost[u][w])

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

}

}

void main()

{

int n,v,i,j,cost[10][10],dist[10];

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

scanf("%d",&n);

printf("\n Enter the cost 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]=infinity;

}

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

scanf("%d",&v);

dij(n,v,cost,dist);

printf("\n Shortest path:\n");

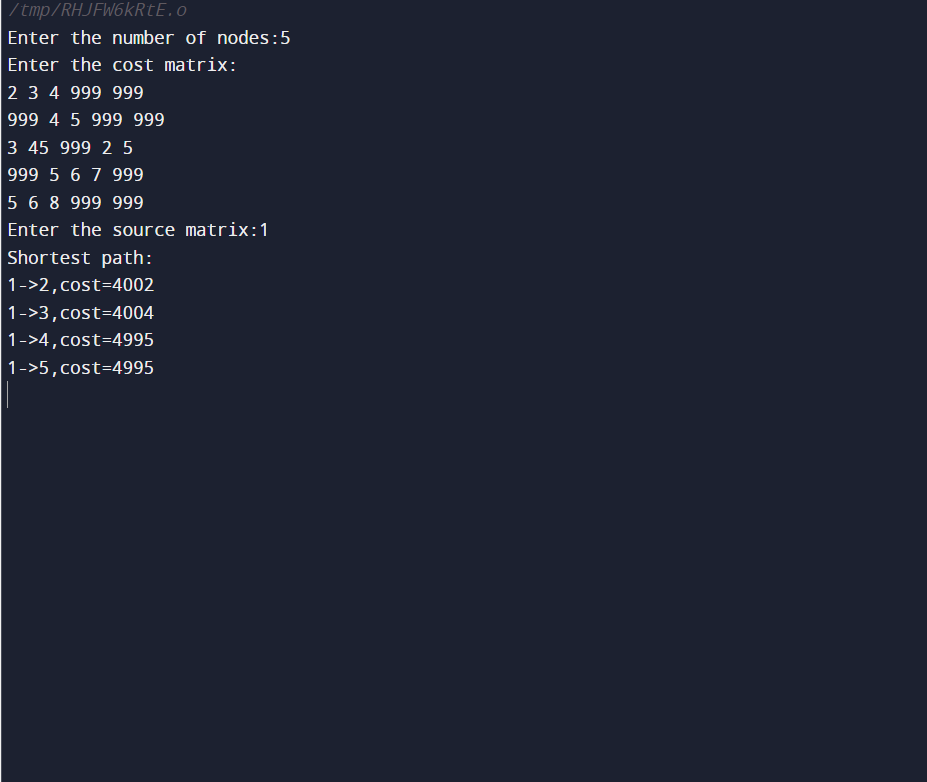
for(i=1;i<=n;i++)

if(i!=v)

printf("%d->%d,cost=%d\n",v,i,dist[i]);

}

**OUTPUT:**

****

# LAB 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>

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

void sumofsub ( int , int , int ) ;

void main ()

{

int n , sum = 0 ;

int i ;

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

scanf ( "%d" , &n ) ;

printf ( " \n Enter the set in increasing order:\n" ) ;

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

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

printf ( " \n Enter the value of d : \n " ) ;

scanf ( "%d" , &d ) ;

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

sum = sum + s[i] ;

if ( sum < d || s[1] > d )

printf ( " \n No subset possible : " ) ;

else

sumofsub ( 0 , 1 , sum ) ;

}

void sumofsub ( int m , int k , int r )

{

int i=1 ;

x[k] = 1 ;

if ( ( m + s[k] ) == d )

{

printf("Subset:");

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

if ( x[i] == 1 )

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

printf ( "\n" ) ;

}

else

if ( m + s[k] + s[k+1] <= d )

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

if ( ( m + r - s[k] >= d ) && ( m + s[k+1] <=d ) )

{

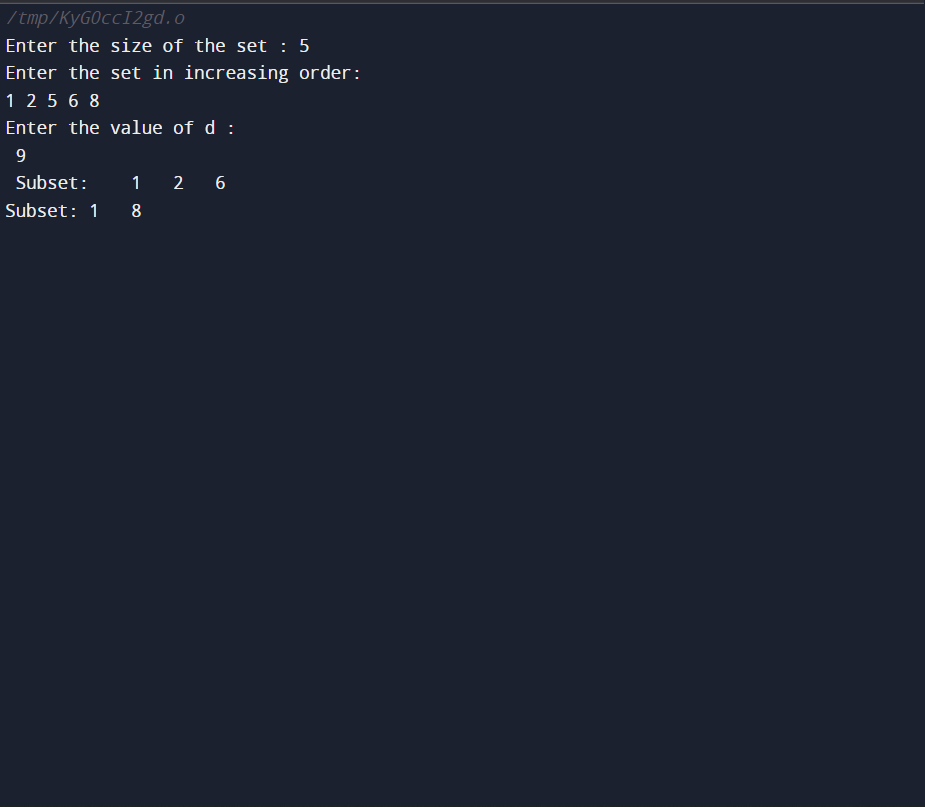
x[k] = 0;

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

}

}

**OUTPUT:**

****

# LAB PROGRAM-18

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

**#include<stdio.h>**

**#include<math.h>**

**int a[30],count=0;**

**int place(int pos) {**

**int i;**

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

**if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))**

**return 0;**

**}**

**return 1;**

**}**

**void print\_sol(int n) {**

**int i,j;**

**count++;**

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

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

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

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

**printf("Q\t"); else**

**printf("\*\t");**

**}**

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

**}**

**}**

**void queen(int n) {**

**int k=1;**

**a[k]=0;**

**while(k!=0) {**

**a[k]=a[k]+1;**

**while((a[k]<=n)&&!place(k))**

**a[k]++;**

**if(a[k]<=n) {**

**if(k==n)**

**print\_sol(n); else {**

**k++;**

**a[k]=0;**

**}**

**} else**

**k--;**

**}**

**}**

**void main() {**

**int i,n;**

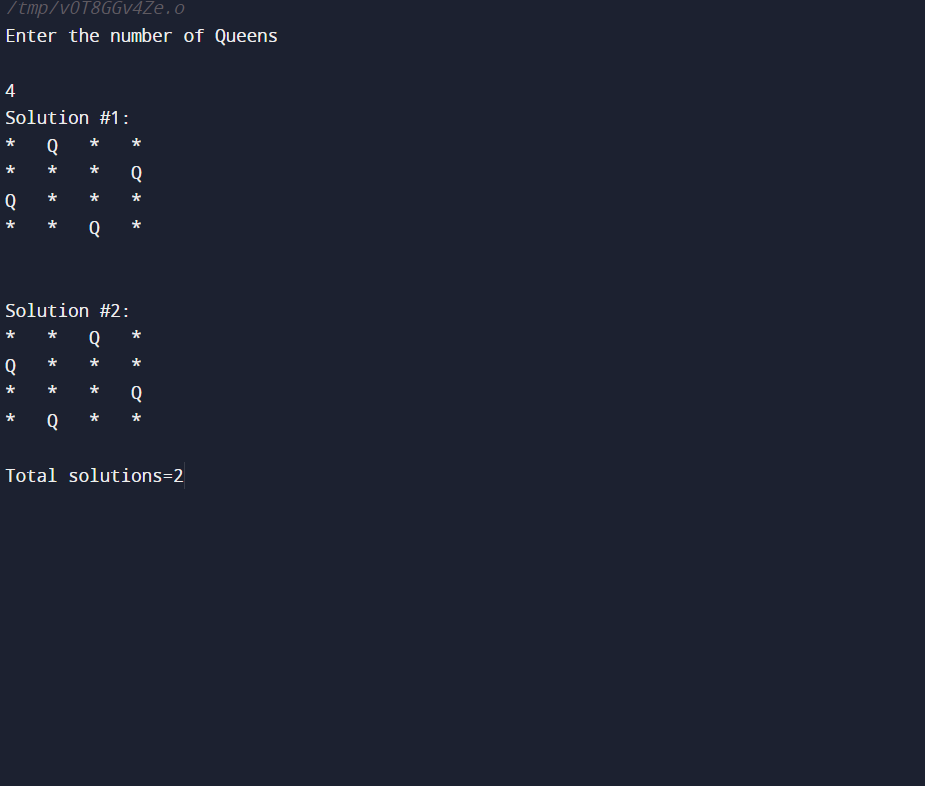
**printf("Enter the number of Queens\n");**

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

**queen(n);**

**printf("\nTotal solutions=%d",count);**

**}**

**OUTPUT:**