

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

NAGARAJ V HALATTI (1BM20CS091)

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 **NAGARAJ V HALATTI (1BM20CS091)**, 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 **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge: **Namratha M**
Designation: Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
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.	13
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.	16
5	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	20
6	Write program to obtain the Topological ordering of vertices in a given digraph.	22
7	Implement Johnson Trotter algorithm to generate permutations.	24
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.	32
10	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	34
11	Implement Warshall's algorithm using dynamic programming	37
12	Implement 0/1 Knapsack problem using dynamic programming.	38
13	Implement All Pair Shortest paths problem using Floyd's algorithm.	41
14	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	42
15	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.	45
16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	47
17	Implement "Sum of Subsets" using Backtracking. "Sum of Subsets" problem: Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d. For	49

	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.	51

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.

Write a recursive program to Solve

a) Towers-of-Hanoi problem b) To find GCD

a) TOWER OF HANOI

```
#include <stdio.h>
void toh(int n,char a,char b,char c)
{
    if(n>0)
    {
        toh(n-1,a,c,b);
        printf("move the disk from %c to %c\n",a,c);
        toh(n-1,b,a,c);
    }
}
int main()
{
    int n;
    char a,b,c;
    printf("Enter the number of disks: ");
    scanf("%d",&n);
    toh(n,'a','b','c');
    return 0;
}
```

OUTPUT:

```
"D:\ENGINEERING\4th sem\ada\lab\tower of honai.exe"
Enter the number of disks: 3
move the disk from a to c
move the disk from a to b
move the disk from c to b
move the disk from a to c
move the disk from b to a
move the disk from b to c
move the disk from a to c

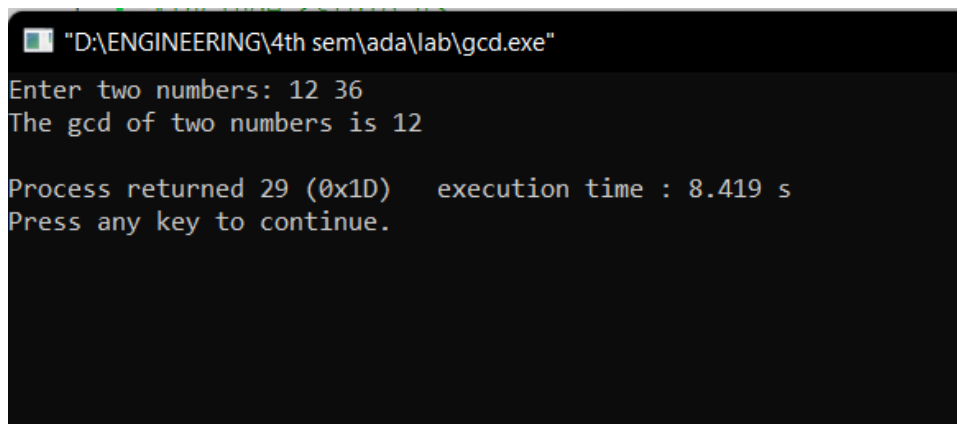
Process returned 0 (0x0)   execution time : 9.336 s
Press any key to continue.
```

b) GREATEST COMMON DIVISOR

```
#include <stdio.h>
```

```
int gcd(int a,int b) {  
    if(b!=0)  
        return gcd(b,a%b);  
    else  
        return a;  
}  
void main()  
{  
    int a,b,c;  
    printf("Enter two numbers: ");  
    scanf("%d %d",&a,&b);  
    c=gcd(a,b);  
    printf("The gcd of two numbers is %d\n",c);  
}
```

OUTPUT:



```
"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"  
Enter two numbers: 12 36  
The gcd of two numbers is 12  
  
Process returned 29 (0x1D)    execution time : 8.419 s  
Press any key to continue.
```

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.

CODE:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
int bin_srch(int [],int,int,int);
int lin_srch(int [],int,int,int);
int n,a[1000000];

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<=7000){
for(i=0;i<n;i++)
a[i]=i;
key=a[n-1];
start=clock();
search_status=bin_srch(a,0,n-1,key);
end=clock();
if(search_status==-1)

printf("\nKey Not Found");
else
printf("\n Key found at position %d",search_status);
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<=7000) {
for(i=0;i<n;i++)
a[i]=i;
key=a[n-1];
```



```
start=clock();
search_status=lin_srch(a,0,n-1,key);
end=clock();
if(search_status==-1)
printf("\nKey Not Found");
else
printf("\n Key found at position %d",search_status);

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) {
for(int j=0;j<1000000;j++);
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) {
for(int j=0;j<10000;j++){ int temp=38/600;}
if(i>high)
return -1;
if(key==a[i])
return i;
else
return lin_srch(a,i+1,high,key);
}
```

OUTPUT:

```
"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"

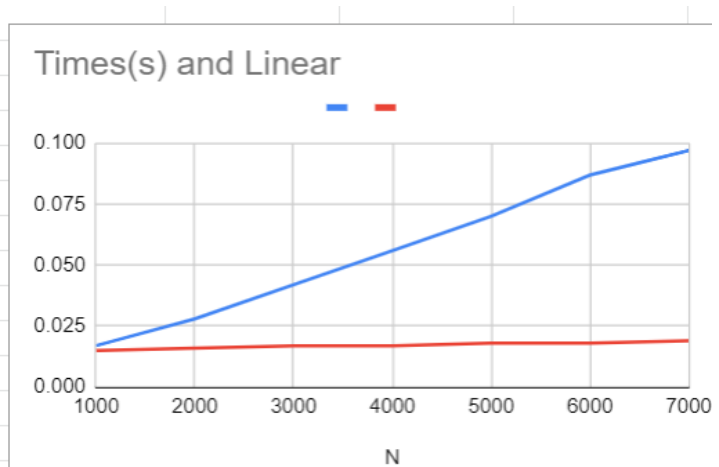
1: Binary search      2: Linear search      3: Exit
Enter your choice:    1

Key found at position 999
Time for n=1000 is 0.001000 Secs
Key found at position 1999
Time for n=2000 is 0.001000 Secs
Key found at position 2999
Time for n=3000 is 0.002000 Secs
Key found at position 3999
Time for n=4000 is 0.001000 Secs
Key found at position 4999
Time for n=5000 is 0.001000 Secs
Key found at position 5999
Time for n=6000 is 0.004000 Secs
Key found at position 6999
Time for n=7000 is 0.001000 Secs
1: Binary search      2: Linear search      3: Exit
Enter your choice:    2

Key found at position 999
Time for n=1000 is 0.002000 Secs
Key found at position 1999
Time for n=2000 is 0.001000 Secs
Key found at position 2999
Time for n=3000 is 0.001000 Secs
Key found at position 3999
Time for n=4000 is 0.019000 Secs
Key found at position 4999
Time for n=5000 is 0.022000 Secs
Key found at position 5999
Time for n=6000 is 0.025000 Secs
Key found at position 6999
Time for n=7000 is 0.029000 Secs
1: Binary search      2: Linear search      3: Exit
Enter your choice:
```

GRAPH:

	Linear	Binary
N	Times(s)	Time(s)
1000	0.017	0.015
2000	0.028	0.016
3000	0.042	0.017
4000	0.056	0.017
5000	0.07	0.018
6000	0.087	0.018
7000	0.097	0.019



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.

CODE:

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

void sort(int x){
    int n=x;
    int a[n],max,i,j,k;
```

```

for(i=0;i<n;i++)
a[i]=i+1;
double start,end;
start = clock();
for(i=0;i<(n-1);i++){
    max=a[i];
    for(j=(i+1);j<n;j++){
        if(max<a[j]){
            max=a[j];
            k=j;
        }
    }
    if(a[i]!=max){
        int temp=a[i];
        a[i]=a[k];
        a[k]=temp;
    }
}
end = clock();
printf("Time taken to sort %d numbers is %f seconds \n",n,(end-
start)/CLOCKS_PER_SEC);

n=n+1000;
}
void main(){
    for(int x=1000;x<=10000;x+=1000){
        sort(x);}
}

```

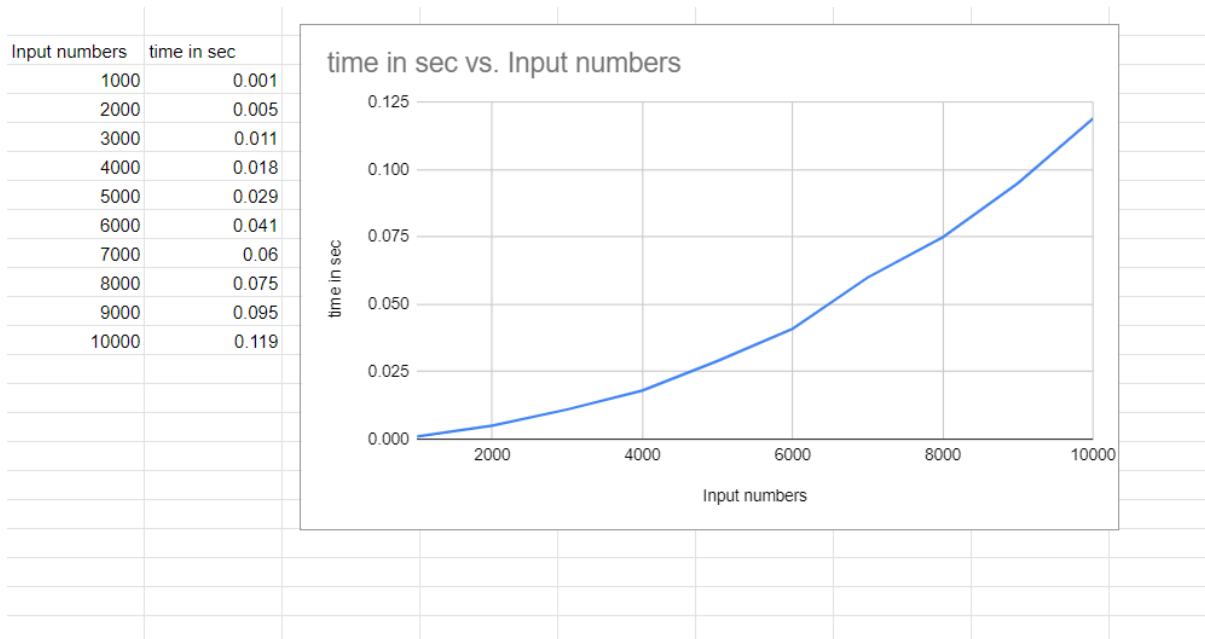
OUTPUT:

```
"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"

Time taken to sort 1000 numbers is 0.001000 seconds
Time taken to sort 2000 numbers is 0.005000 seconds
Time taken to sort 3000 numbers is 0.011000 seconds
Time taken to sort 4000 numbers is 0.018000 seconds
Time taken to sort 5000 numbers is 0.029000 seconds
Time taken to sort 6000 numbers is 0.041000 seconds
Time taken to sort 7000 numbers is 0.060000 seconds
Time taken to sort 8000 numbers is 0.075000 seconds
Time taken to sort 9000 numbers is 0.095000 seconds
Time taken to sort 10000 numbers is 0.119000 seconds

Process returned 54 (0x36)   execution time : 1.210 s
Press any key to continue.
```

GRAPH:



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.**

a) BREADTH FIRST SEARCH

CODE:

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

int a[15][15],n;
void bfs(int);

void 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);
}

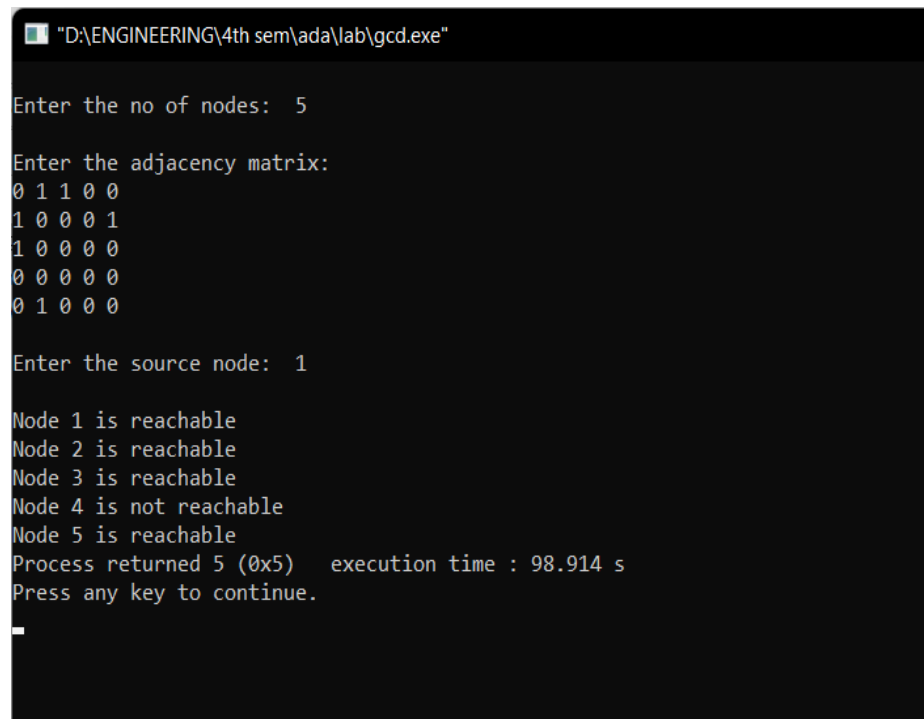
void bfs(int src) {
    int q[15],f=0,r=-1,vis[15],i,j;
    for(j=1;j<=n;j++)
        vis[j]=0;
    vis[src]=1;
    r=r+1;
    q[r]=src;
    while(f<=r) {
        i=q[f];
        f=f+1;
        for(j=1;j<=n;j++)
        {
```

```

    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",j);
    else
        printf("\nNode %d is reachable",j);
}
}

```

OUTPUT:



```

"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"
Enter the no of nodes: 5
Enter the adjacency matrix:
0 1 1 0 0
1 0 0 0 1
1 0 0 0 0
0 0 0 0 0
0 1 0 0 0
Enter the source node: 1
Node 1 is reachable
Node 2 is reachable
Node 3 is reachable
Node 4 is not reachable
Node 5 is reachable
Process returned 5 (0x5)   execution time : 98.914 s
Press any key to continue.

```

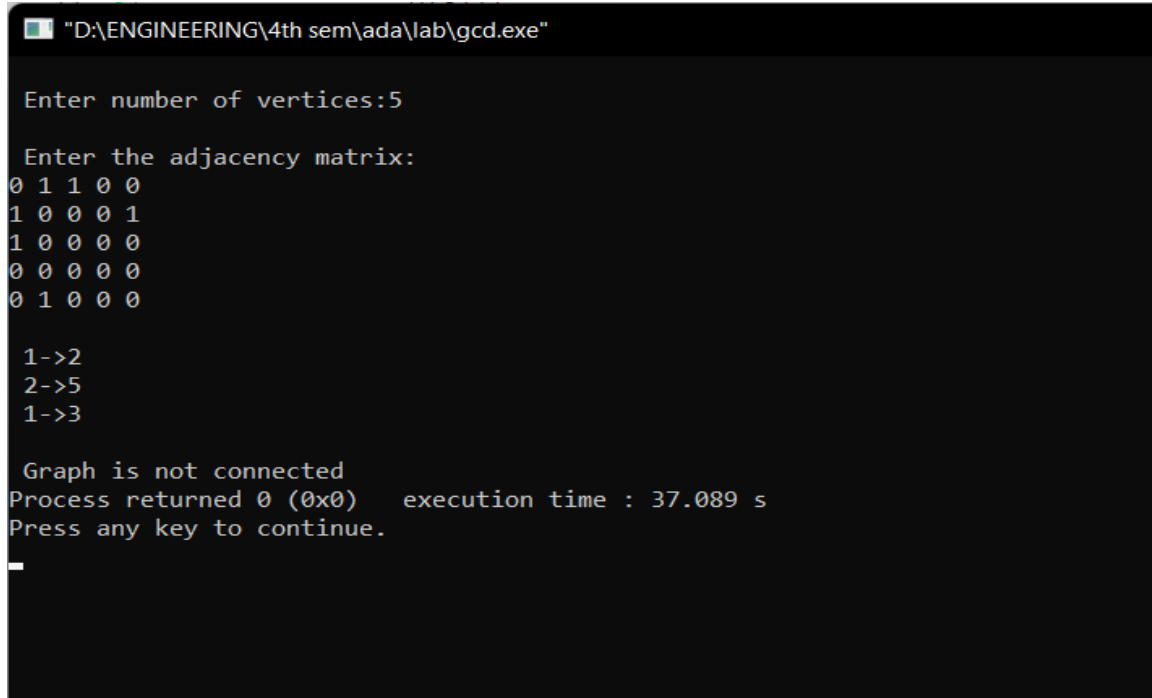

b)DEPTH FIRST SEARCH CODE:

```
#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("\nGraph 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:



```
"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"  
  
Enter number of vertices:5  
  
Enter the adjacency matrix:  
0 1 1 0 0  
1 0 0 0 1  
1 0 0 0 0  
0 0 0 0 0  
0 1 0 0 0  
  
1->2  
2->5  
1->3  
  
Graph is not connected  
Process returned 0 (0x0) execution time : 37.089 s  
Press any key to continue.  
-
```

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

CODE:

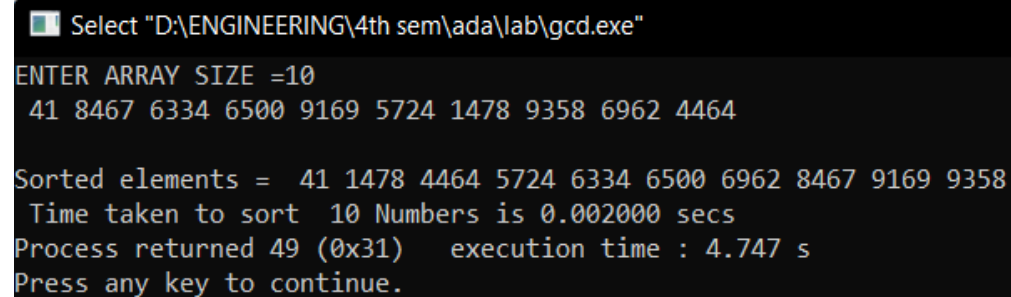
```
#include <math.h>
#include <stdio.h>
#include<stdlib.h>
#include<time.h>

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            for(int k=0;k<100000;k++);
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void main() {
    int i, n;
    clock_t start, end;
    printf("ENTER ARRAY SIZE =");
    scanf("%d", &n);
    int arr[150000];
    for (int j = 0; j < n; j++)
        arr[j] = rand()%10000;
    for (i = 0; i < n; i++)
        printf(" %d", arr[i]);
    printf("\n");
    start = clock();
    insertionSort(arr, n);
    end = clock();
```

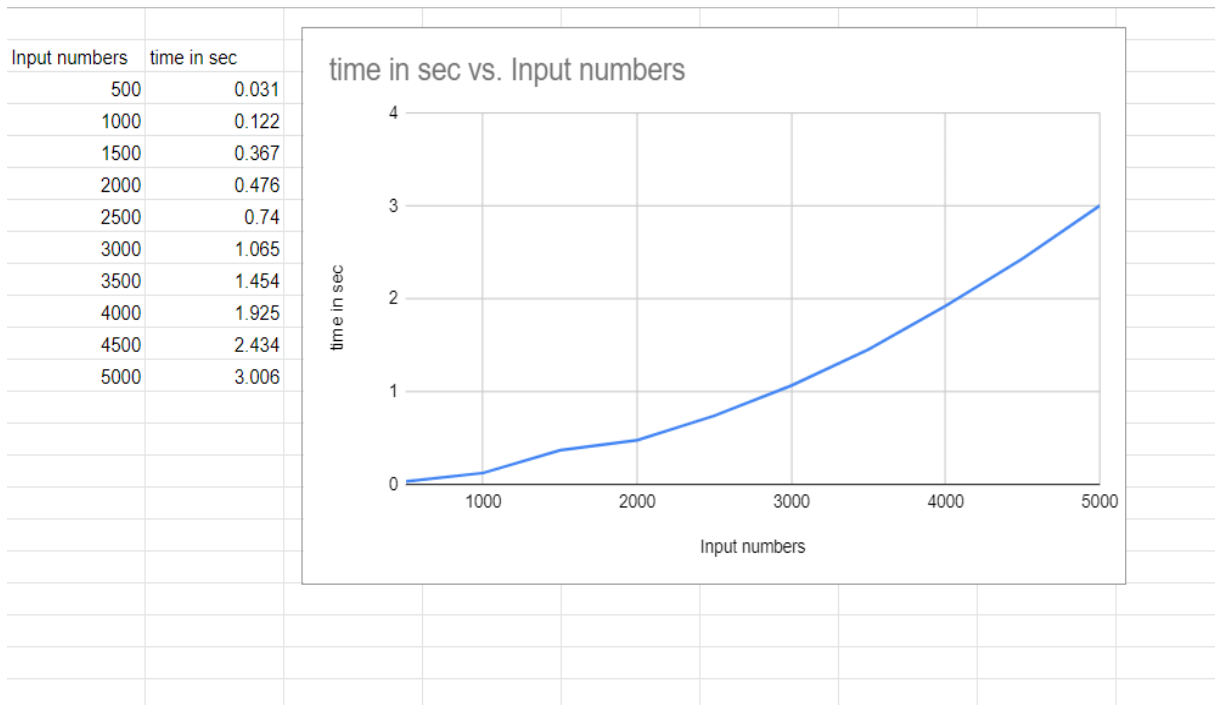
```
printf("\nSorted elements = ");  
for (i = 0; i < n; i++)  
    printf(" %d", arr[i]);  
printf("\n Time taken to sort  %d Numbers is %f secs", n, (((double)(end -  
start)) / CLOCKS_PER_SEC));  
}
```

OUTPUT:



```
Select "D:\ENGINEERING\4th sem\ada\lab\gcd.exe"  
ENTER ARRAY SIZE =10  
41 8467 6334 6500 9169 5724 1478 9358 6962 4464  
  
Sorted elements = 41 1478 4464 5724 6334 6500 6962 8467 9169 9358  
Time taken to sort 10 Numbers is 0.002000 secs  
Process returned 49 (0x31) execution time : 4.747 s  
Press any key to continue.
```

GRAPH:



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

CODE:

```
#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[i][j];
        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;
            }
        }
    }
    printf("Topological order :");
    for(i=0;i<n;i++)
        printf(" %d", t[i]);
}

```

```

void 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);
    getch();
}

```

OUTPUT:

```
C:\Users\mknv7\OneDrive\Documents\C-Workspace\topo_sort\Debug\topo_sort.exe
Enter number of nodes
6
Enter the adjacency matrix
0 1 1 1 0 0
0 0 0 0 1 0
0 0 0 0 1 1
0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 1 0
Topological order : 0 3 2 5 1 4
```

7. Implement Johnson Trotter algorithm to generate permutations.

CODE:

```
#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("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:

```
C:\Users\mkknv7\OneDrive\Documents\C-Workspace\Jhonson-Trotter\Debug\Jhonson-Trotter.exe
Johnson trotter algorithm to find all permutations of given numbers
Enter the number
3
total permutations = 6
All possible permutations are:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.719 (MM:SS.MS)
Press any key to continue...
```

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

CODE:

```
#include<stdio.h>
#include<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[30000],n=500,i;
    while(n<=5000){
        for(i=0;i<n;i++)
        {
            a[i] = rand()%1000;
        }
        start = clock();
        mergesort(a,0,n-1);
        end = clock();
        printf("\n To Sort array of %d numbers ",n);
        printf("required  time is %lf secs",(double)(end-start)/CLOCKS_PER_SEC);
        printf("\n");
        n+=500;
    }
}

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[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];
}
}
```

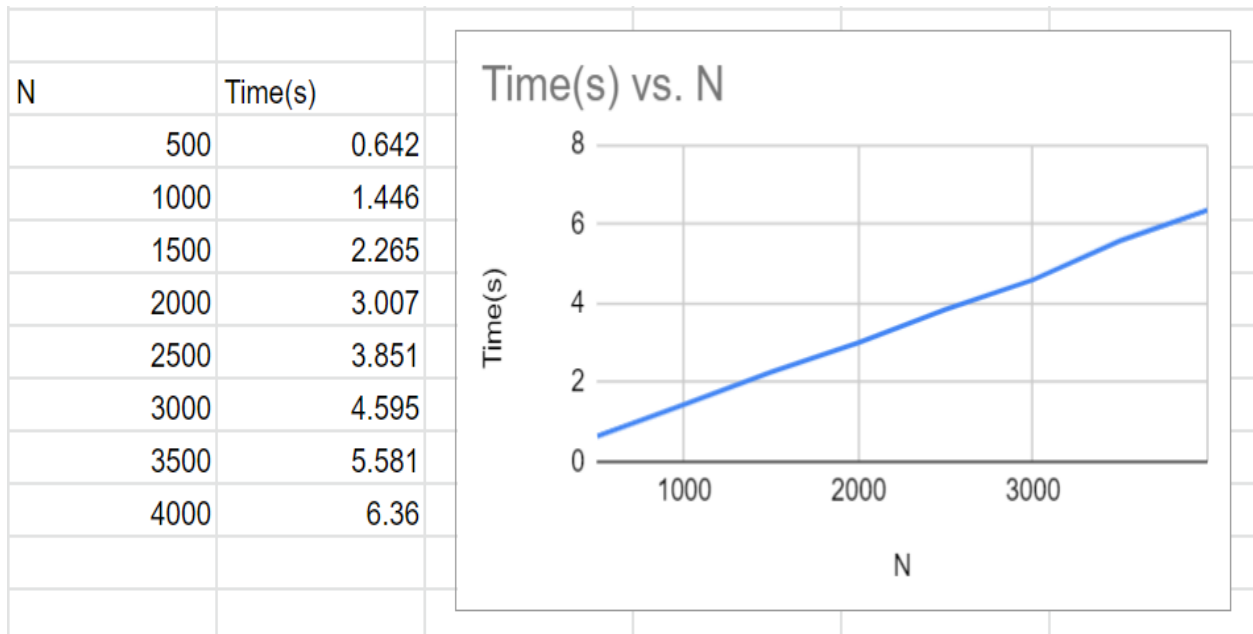
OUTPUT:

"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"

```
To Sort array of 500 numbers required time is 0.190000 secs
To Sort array of 1000 numbers required time is 1.473000 secs
To Sort array of 1500 numbers required time is 1.131000 secs
To Sort array of 2000 numbers required time is 2.813000 secs
To Sort array of 2500 numbers required time is 2.417000 secs
To Sort array of 3000 numbers required time is 2.670000 secs
To Sort array of 3500 numbers required time is 2.855000 secs
To Sort array of 4000 numbers required time is 2.133000 secs
To Sort array of 4500 numbers required time is 4.245000 secs
To Sort array of 5000 numbers required time is 6.587000 secs
```

Process returned 0 (0x0) execution time : 29.450 s
Press any key to continue.

GRAPH:



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

CODE:

```
#include<stdio.h>
#include<time.h>
#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);
    }
}

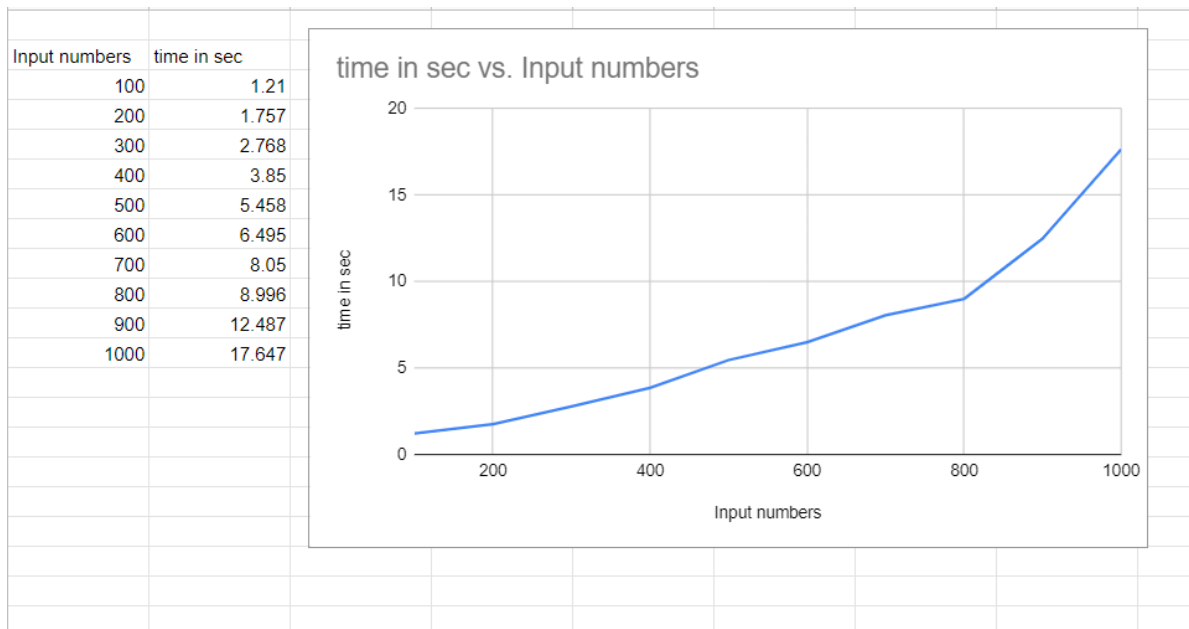
int main()
{
```

```
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()%5000;
}
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:

```
"D:\ENGINEERING\4th sem\ada\lab\gcd.exe"
No. of elements: 100
Order of Sorted elements: 6 37 41 106 141 153 288 292 333 350 436 447 491 537 547 667 705 724 778 890 1101 1118 1299 13
08 1322 1323 1334 1478 1500 1538 1538 1541 1726 1729 1827 1840 1842 1868 1869 1942 1944 1962 2035 2082 2190 2316 2376 23
82 2391 2421 2439 2446 2529 2623 2644 2648 2662 2673 2711 2757 2859 2929 2995 3035 3145 3253 3281 3467 3548 3703 3716 37
23 3756 3805 3811 3902 3931 3942 4040 4084 4169 4264 4358 4370 4393 4464 4604 4626 4629 4664 4718 4741 4771 4827 4894 48
95 4912 4954 4961 4966
Seconds taken 0.713000
Process returned 0 (0x0)   execution time : 3.286 s
Press any key to continue.
```

GRAPH:



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

CODE:

```
#include <stdio.h>
```

```
#include <time.h>
```

```
void 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) {  
    int i, j;  
    for (i = n / 2 - 1; i >= 0; i--)  
        heapify(arr, n, i);  
    for (i = n - 1; i >= 0; i--) {  
        swap(&arr[0], &arr[i]);  
        for(j=0;j<10000000;j++);  
        heapify(arr, i, 0);  
    }  
}
```

```
}
```

```
void printArray(int arr[], int n)
```

```
{
```

```
    int i;
```

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

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

```
    printf("\n");
```

```
}
```

```
int main()
```

```
{
```

```
    clock_t start,end;
```

```
    int arr[1000];
```

```
    int j,n;
```

```
    printf("Enter the number of values to be inserted : \n");
```

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

```
    for(j=0;j<1000;j++)
```

```
        arr[j] = rand()%1000;
```

```
    printf("Before Heap Sort : \n");
```

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

```
        printf("%d ",arr[j]);
```

```
    start = clock();
```

```
    heapSort(arr, n);
```

```
    end = clock();
```

```
    printf("\nSorted array is given in the following way \n");
```

```
    printArray(arr, n);
```

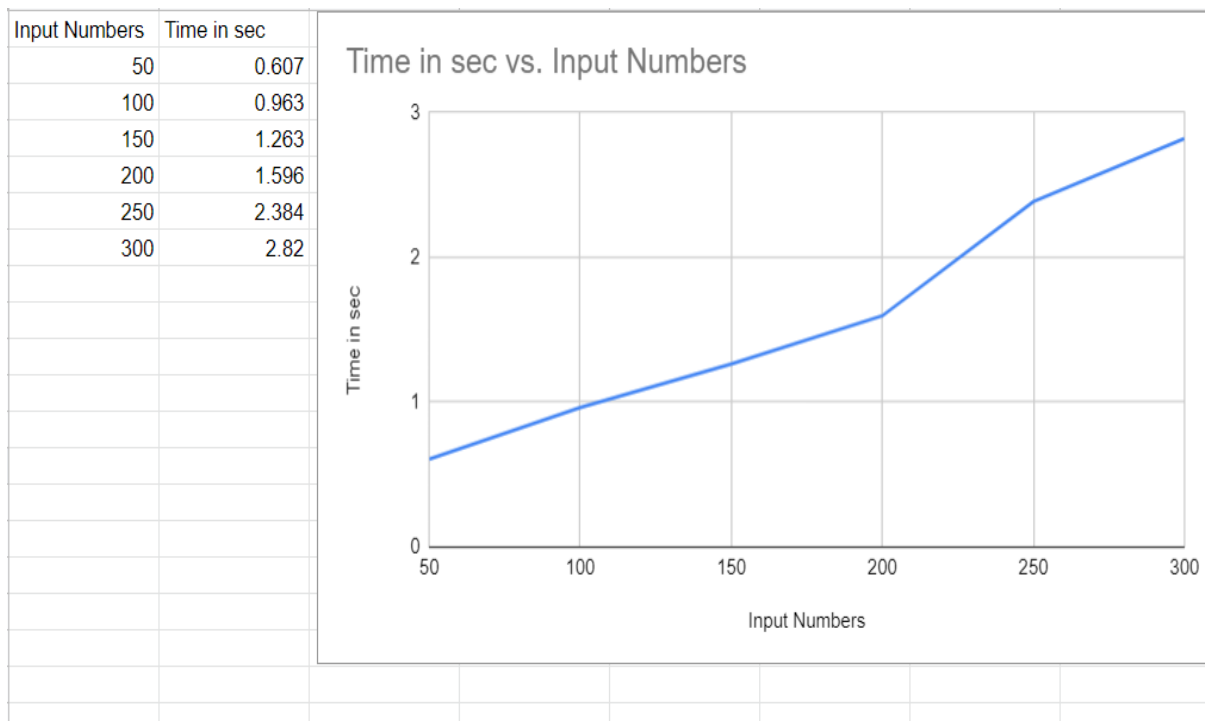
```
    printf("Time taken : %lf", (double)(end-start)/CLOCKS_PER_SEC);
```

```
}
```

OUTPUT:

```
C:\Users\Lenovo\OneDrive\Documents\heapsort.exe
Enter the number of values to be inserted :
20
Before Heap Sort :
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491 995 942 827 436
Sorted array is given in the following way
995 962 961 942 827 827 724 705 500 491 478 467 464 436 358 334 281 169 145 41
Time taken : 0.154000
Process returned 0 (0x0)   execution time : 6.112 s
Press any key to continue.
```

GRAPH:



11.Implement Warshall's algorithm using dynamic programming

CODE:

```
#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");
    }
    int flag = 0;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if (i==j)
                if(a[i][j]==1)
                    flag = 1;
        }
    }
}
```

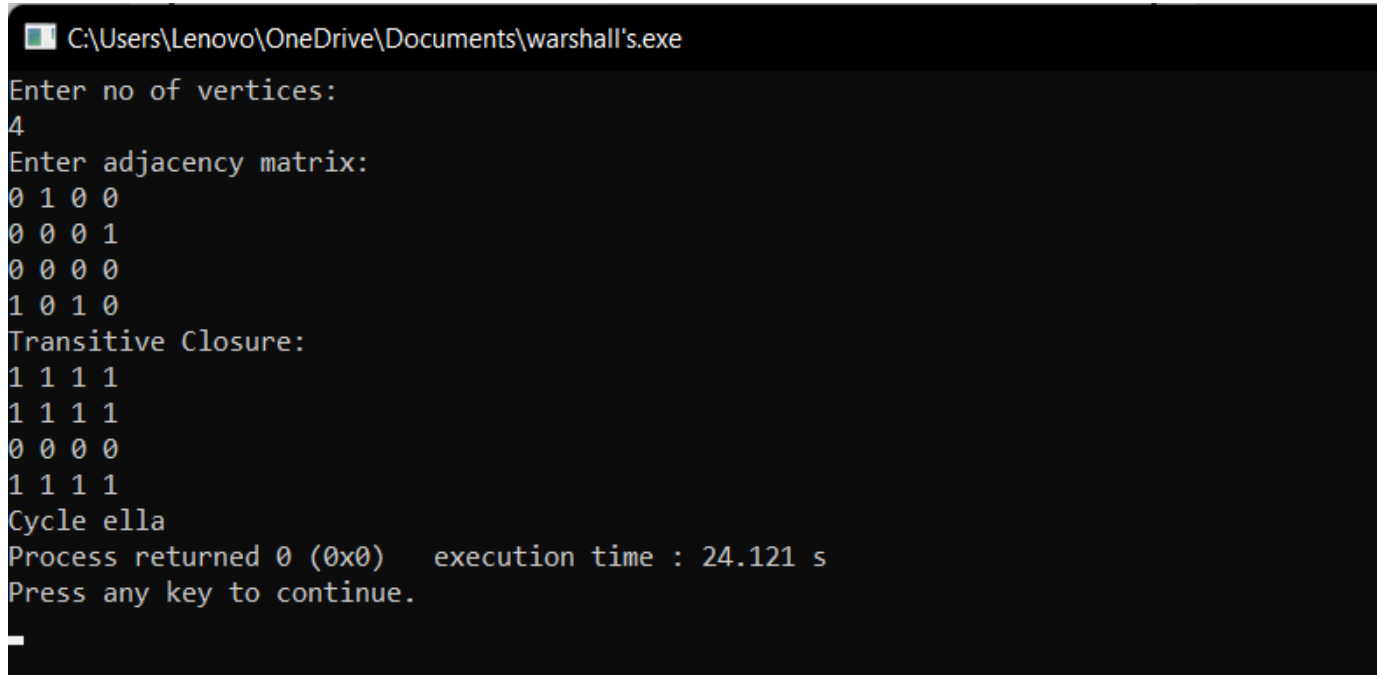
```

    if(flag==0)
        printf("Cycle present");
    else
        printf("No Cycle ");

}

```

OUTPUT:



```

C:\Users\Lenovo\OneDrive\Documents\warshall's.exe
Enter no of vertices:
4
Enter adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Transitive Closure:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
Cycle ella
Process returned 0 (0x0)   execution time : 24.121 s
Press any key to continue.

```

12.Implement 0/1 Knapsack algorithm using dynamic programming

CODE:

```

#include<stdio.h>
#include<conio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]= {0};
int max(int i,int j) {
    return ((i>j)?i:j);
}

```

```

int knap(int i,int j) {
    int value;
    if(v[i][j]<0) {
        if(j<w[i])
            value=knap(i-1,j);
    else
        value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
        v[i][j]=value;
    }
    return(v[i][j]);
}

void main() {
    int profit,count=0;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    printf("Enter the profit and weights of the elements: ");
    for (i=1;i<=n;i++) {
        printf("For item no %d\n",i);
        scanf("%d%d",&p[i],&w[i]);
    }
    printf("Enter the capacity: ");
    scanf("%d",&cap);
    for (i=0;i<=n;i++)
        for (j=0;j<=cap;j++)
            if((i==0) || (j==0))
                v[i][j]=0; else
                v[i][j]=-1;
    profit=knap(n,cap);
    i=n;
    j=cap;
    while(j!=0&& i!=0) {
        if(v[i][j]!=v[i-1][j]) {
            x[i]=1;
            j=j-w[i];
            i--;
        } else
            i--;
    }
    printf("Items included are:\n");
}

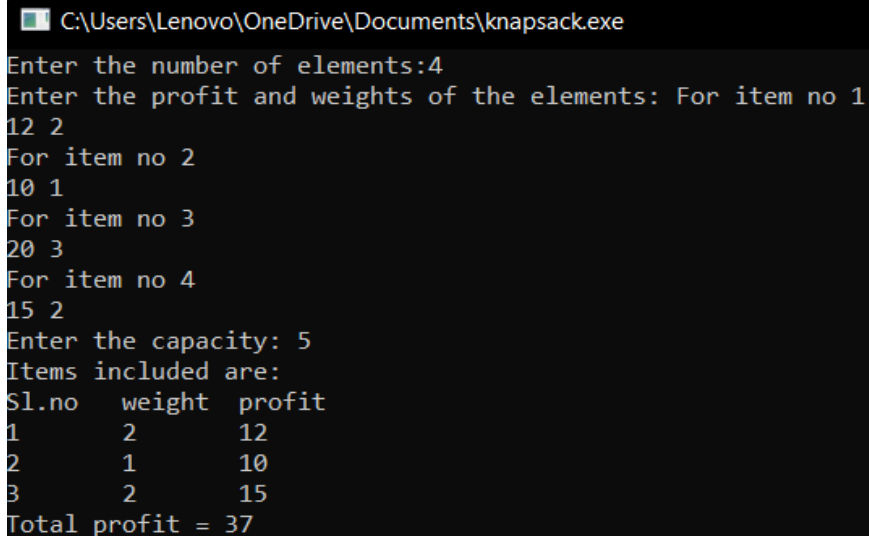
```

```

        printf("Sl.no\tweight\tprofit\n");
        for (i=1;i<=n;i++)
            if(x[i])
                printf("%d\t%d\t%d\n",++count,w[i],p[i]);
        printf("Total profit = %d\n",profit);
        getch();
    }

```

OUTPUT:



```

C:\Users\Lenovo\OneDrive\Documents\knapsack.exe
Enter the number of elements:4
Enter the profit and weights of the elements: For item no 1
12 2
For item no 2
10 1
For item no 3
20 3
For item no 4
15 2
Enter the capacity: 5
Items included are:
Sl.no   weight  profit
1       2      12
2       1      10
3       2      15
Total profit = 37

```

13.Implement All pair Shortest paths problem using Floyd's algorithm

CODE:

```

#include <stdio.h>
int min(int a, int b)
{
    return (a<b)? a: b;
}
void floyd(int a[][100], int n)

```



```

{
    int i, j, k;
    for(k=0; k<n; k++){
        for(i=0; i<n; i++){
            for(j=0; j<n; j++)
                a[i][j] = min(a[i][j], (a[i][k] + a[k][j]));
        }
    }
}

int main()
{
    int n, a[100][100], i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the weighted matrix(enter 99999 for infinity): \n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++)
            scanf("%d", &a[i][j]);
    }

    floyd(a, n);
    printf("All pair shortest paths: \n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++)
            printf("%d \t", a[i][j]);
        printf("\n");
    }

    return 0;
}

```

OUTPUT:

```
C:\Users\Lenovo\OneDrive\Documents\floyd's.exe
Enter the number of vertices: 4
Enter the weighted matrix(enter 99999 for infinity):
11 99999 99999 44
33 44 99999 3
33 55 99999 555
44 3 6 9
All pair shortest paths:
11      47      50      44
33       6       9       3
33      55      64      58
36       3       6       6
Process returned 0 (0x0)   execution time : 70.499 s
Press any key to continue.
```

14. Find Minimum Cost spanning tree of given undirected graph using prim's algorithm

CODE:

```
#include<stdio.h>
int main()
{
    int cost[10][10],visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;
    printf("Enter number of nodes: ");
    scanf("%d",&n);
    printf("Enter cost in form of adjacency matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=1000;
        }
    }
}
```

```

}

visited[1]=1;
while(no_e<n)
{
    min=1000;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(cost[i][j]<min)
            {
                if(visited[i]!=0)
                {
                    min=cost[i][j];
                    a=i;
                    b=j;
                }
            }
        }
    }

    if(visited[b]==0)
    {
        printf("\n%d to %d cost=%d",a,b,min);
        min_cost=min_cost+min;
        no_e++;
    }
    visited[b]=1;
    cost[a][b]=cost[b][a]=1000;
}
printf("\nminimum weight is %d",min_cost);
return 0;
}

```

OUTPUT:

```
C:\Users\Lenovo\OneDrive\Documents\prim's.exe
Enter number of nodes: 5
Enter cost in form of adjacency matrix:
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 2
999 999 999 2 0

1 to 2 cost=1
1 to 4 cost=2
4 to 5 cost=2
4 to 3 cost=3
minimum weight is 8
Process returned 0 (0x0) execution time : 52.657 s
Press any key to continue.
```

15. Find Minimum Cost spanning tree of given undirected graph using kruskal's algorithm

CODE:

```
#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]);
if(c[i][j]==0){
    c[i][j]=9999;
}
}
}
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];
}
}
}

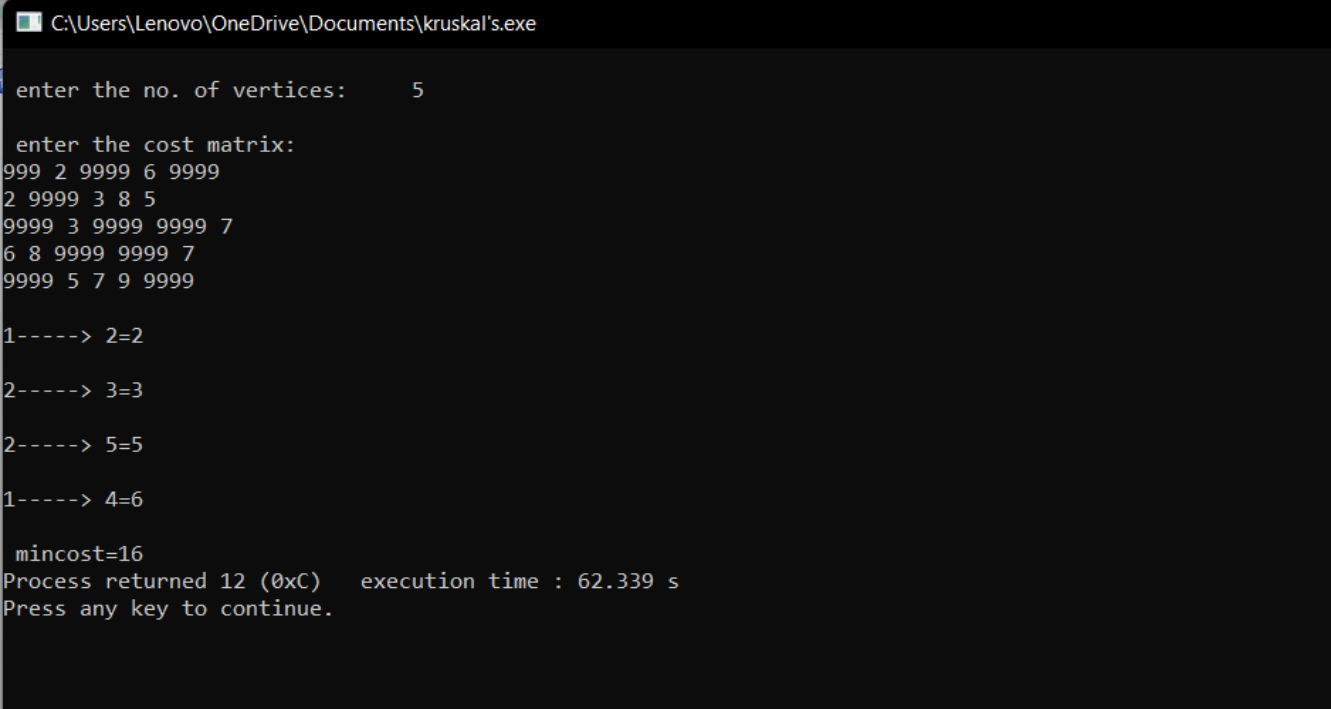
```

```

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;
}
if(u==v){
printf("in cycle %d",u);
}
c[a][b]=c[b][a]=9999;
}
printf("\n mincost=%d",mincost);
}

```

OUTPUT:



```

C:\Users\Lenovo\OneDrive\Documents\kruskal's.exe

enter the no. of vertices:    5

enter the cost matrix:
999 2 9999 6 9999
2 9999 3 8 5
9999 3 9999 9999 7
6 8 9999 9999 7
9999 5 7 9 9999

1-----> 2=2
2-----> 3=3
2-----> 5=5
1-----> 4=6

mincost=16
Process returned 12 (0xC)   execution time : 62.339 s
Press any key to continue.

```

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

CODE:

```
#include<stdio.h>
void dijkstras();
int c[10][10],n,src;
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]);
            if(c[i][j]==0){
                c[i][j]=9999;
            }
        }
    }
    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("\nthe shortest distance is:\n");
for(j=1;j<=n;j++)
{
printf("; \n%d----->%d=%d",src,j,dist[j]);
}
}
```


OUTPUT:

```
C:\Users\Lenovo\OneDrive\Documents\djstra.exe

enter the no of vertices:      5

enter the cost matrix:
0 10 999 30 100
10 0 50 999 999
999 50 0 20 10
30 999 20 0 60
100 999 10 60 0

enter the source node:  1

the shortest distance is:
;
1----->1=0;
1----->2=10;
1----->3=50;
1----->4=30;
1----->5=60
Process returned 5 (0x5)   execution time : 105.731 s
Press any key to continue.
```

17. From implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d . For example, if $S = \{1, 2, 5, 6, 8\}$ and $d = 9$ there are two solutions $\{1, 2, 6\}$ and $\{1, 8\}$. A suitable message is to be displayed if the given problem instance doesn't have a solution.

CODE:

```
#include<stdio.h> 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:

```

/tmp/KyG0ccI2gd.o
Enter the size of the set : 5
Enter the set in increasing order:
1 2 5 6 8
Enter the value of d :
9
Subset: 1 2 6
Subset: 1 8

```

18.Implement “N-Queens Problem” using Backtracking From implement “Sum of Subsets” using Backtracking.

CODE:

```

#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:

```

/tmp/v0T8G6v4Ze.o
Enter the number of Queens

4
Solution #1:
*  Q  *  *
*  *  *  Q
Q  *  *  *
*  *  Q  *

Solution #2:
*  *  Q  *
Q  *  *  *
*  *  *  Q
*  Q  *  *

Total solutions=2

```