

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## Analysis and Design Of Algorithms LAB REPORT

(19CS4PCADA)

*Submitted by*

**RAVI SAJJANAR (1BM19CS127)**

*Under the Guidance of*

**Dr. Nagarathna N  
Professor, BMSCE**

*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**

**Mar-2021 to Jun-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 Assignment work entitled “**Database Management System**” carried out by **RAVI SAJJANAR (1BM19CS127)** who is the bonafide students 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 Visveswaraiiah Technological University, Belgaum during the year 2021-2022. The Lab report has been approved as it satisfies the academic requirements in respect of **Analysis and Design Of Algorithms (19CS4PADA) LAB** work prescribed for the said degree.

Signature of the Guide Dr.  
Dr.Nagarathna N  
Professor  
BMSCE, Bengaluru

Signature of the HOD  
Dr. Umadevi V  
Associate Prof.& Head, Dept. of CSE  
BMSCE, Bengaluru

External Viva

Name of the Examiner

Signature with date

1. \_\_\_\_\_

\_\_\_\_\_

2. \_\_\_\_\_

\_\_\_\_\_

## INDEX

S.NO	PROGRAM	PAGE
01	Write a recursive program to a) Solve Towers-of-Hanoi problem b) To find GCD	4
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.	8
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.	15
04	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.	18
05	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	23
06	Write program to obtain the Topological ordering of vertices in a given digraph.	26
07	Implement Johnson Trotter algorithm to generate permutations.	31
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.	35
09	Sort a given set of N integer elements using Quick Sort technique and compute its time taken	39

## PROGRAM 1:

Write a recursive program to

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

```
// Tower of hennoi
#include<stdio.h>

int count=1;

void TowerOfHanoi(int n,char src,char temp,char des){
    if(n==1){
        printf("%d. Move Disk %d from %c to %c\n",count++,n,src,des);
        return;
    }

    TowerOfHanoi(n-1,src,des,temp);
    printf("%d. Move Disk %d from %c to %c\n",count++,n,src,des);
    TowerOfHanoi(n-1,temp,src,des);
}

void main(){
    int n=0;
    printf("Enetr the number of diskess:\t");
    scanf("%d",&n);

    TowerOfHanoi(n,'S','T','D');
}
```

D:\codes\LAB1B.exe

```
Enetr the number of disks:      3
1. Move Disk 1 from S to D
2. Move Disk 2 from S to T
3. Move Disk 1 from D to T
4. Move Disk 3 from S to D
5. Move Disk 1 from T to S
6. Move Disk 2 from T to D
7. Move Disk 1 from S to D

-----
Process exited after 2.9 seconds with return value 0
Press any key to continue . . .
```

// Iterative GCD

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

```
int gcd(int a,int b){
```

```
    int r;
```

```
    while (b!=0){
```

```
        r=a%b;
```

```
        a=b;
```

```
        b=r;
```

```
    }
```

```
    return a;
```

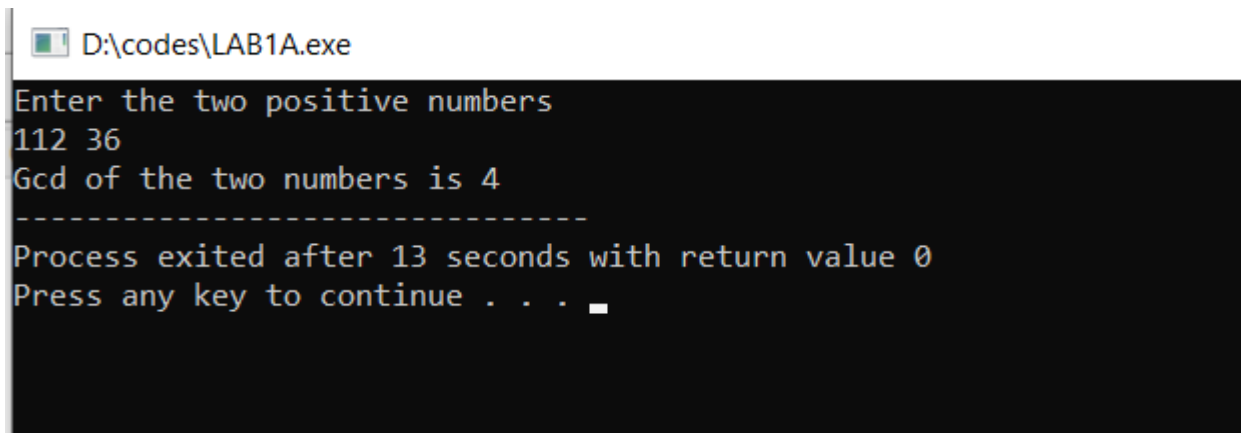
```
}
```

```
int main()
```

```
{
```

```
    int a,b,res;
```

```
printf("Enter the two positive numbers\n");  
scanf("%d%d",&a,&b);  
res=gcd(a,b);  
printf("Gcd of the two numbers is %d",res);  
return 0;  
}
```



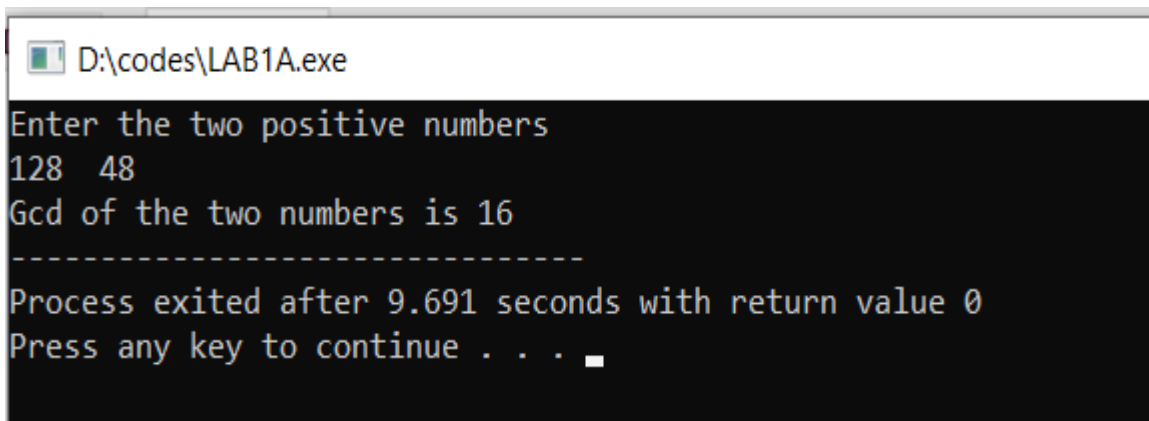
```
D:\codes\LAB1A.exe  
Enter the two positive numbers  
112 36  
Gcd of the two numbers is 4  
-----  
Process exited after 13 seconds with return value 0  
Press any key to continue . . .
```

```
//Recursive GCD
```

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

```
int gcd(int a,int b){  
    if (b==0) return a;  
    else return gcd(b,(a%b));  
}
```

```
int main()  
{  
    int a,b,res;  
    printf("Enter the two positive numbers\n");  
    scanf("%d%d",&a,&b);  
    res=gcd(a,b);  
    printf("Gcd of the two numbers is %d",res);  
    return 0;  
}
```



```
D:\codes\LAB1A.exe  
Enter the two positive numbers  
128 48  
Gcd of the two numbers is 16  
-----  
Process exited after 9.691 seconds with return value 0  
Press any key to continue . . .
```

## **PROGRAM 2: Linear & Binary Search**

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.

### **Linear Search**

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

int arr[100000];

int linear_search(int arr[], int key,int i,int n)
{ int a,b;
  for(a=0;a<1000;a++)
  {
    for(b=0;b<10000;b++)
    {

    }
  }

  if(i>n)
  return -1;
  if(arr[i]==key)
  return i;
  else
  {
    linear_search(arr,key,i+1,n);
  }
}

int main()
```



```

{
    int key,res,n,i;
    time_t start,end;

    printf("Enter the no of elements \n");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        arr[i]=rand();
    }

    printf("The elements are :\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",arr[i]);
    }
    printf("\n");

    printf("Enter element to be searched\n");
    scanf("%d",&key);


    start=time(NULL);
    res=linear_search(arr,key,0,n);
    end=time(NULL);

    if(res==-1)
        printf("Element not found in linear search\n");
    else
        printf("Element found in linear search at pos %d\n",res);

    printf("The time taken is %.10f",difftime(end,start)/CLOCKS_PER_SEC);

    return 0;
}

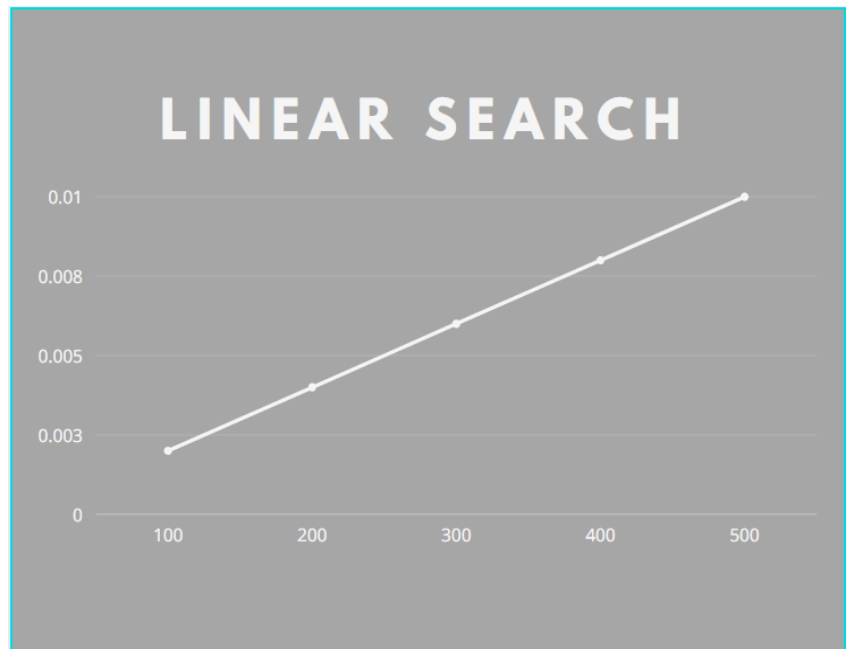
```

```

C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 2 ( Recursive Binary and Linear Search)\linearsearchrecursive.exe
Enter the no of elements
100
The elements are :
41      18467   6334    26500   19169   15724   11478   29358   26962   24464   5705    28145   23281   16827   9961
      491      2995    11942   4827    5436    32391   14604   3902    153     292     12382   17421   18716   19718
      19895   5447    21726   14771   11538   1869    19912   25667   26299   17035   9894    28703   23811   31322
      30333   17673   4664    15141   7711    28253   6868    25547   27644   32662   32757   20037   12859   8723
      9741    27529   778     12316   3035    22190   1842    288     30106   9040    8942    19264   22648   27446
      23805   15890   6729    24370   15350   15006   31101   24393   3548    19629   12623   24084   19954   18756
      11840   4966    7376    13931   26308   16944   32439   24626   11323   5537    21538   16118   2082    22929
      16541
Enter element to be searched
1
Element not found in linear search
The time taken is 0.0020000000
-----
Process exited after 4.752 seconds with return value 0
Press any key to continue . . .

```

Linear Search Array size	Time in sec
100	0.002
200	0.004
300	0.006
400	0.008
500	0.010



# Binary Search

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

int arr[100000];

int binarySearch(int arr[], int l, int r, int x)
{
    int a,b;
    for(a=0;a<1000;a++)
    {
        for(b=0;b<10000;b++)
        {

        }
    }
    if (r >= l)
    {
        int mid = l + (r - l)/2;

        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);

        return binarySearch(arr, mid+1, r, x);
    }
    return -1;
}

void sort(int arr[],int n)
{
    int temp;
    int i,j;
    for(i=0;i<n;i++)
```

```

    {
        for(j=i+1;j<n;j++)
        {
            if(arr[i]>=arr[j])
            {
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
        }
    }
}

```

```

int main()
{
    int key,res,n,i;
    time_t start,end;

    printf("Enter the no of elements \n");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        arr[i]=rand();
    }

    start=time(NULL);
    sort(arr,n);
    printf("The elements in sorted array are :\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",arr[i]);
    }
    printf("\n");

    printf("Enter element to be searched\n");

```

```

scanf("%d",&key);

res=binarySearch(arr,0,n-1,key);
end=time(NULL);

if(res== -1)
    printf("Element not found in binary search\n");
else
    printf("Element found in binary search at pos %d\n",res);

printf("The time taken is %.10f",difftime(end,start)/CLOCKS_PER_SEC);

return 0;
}

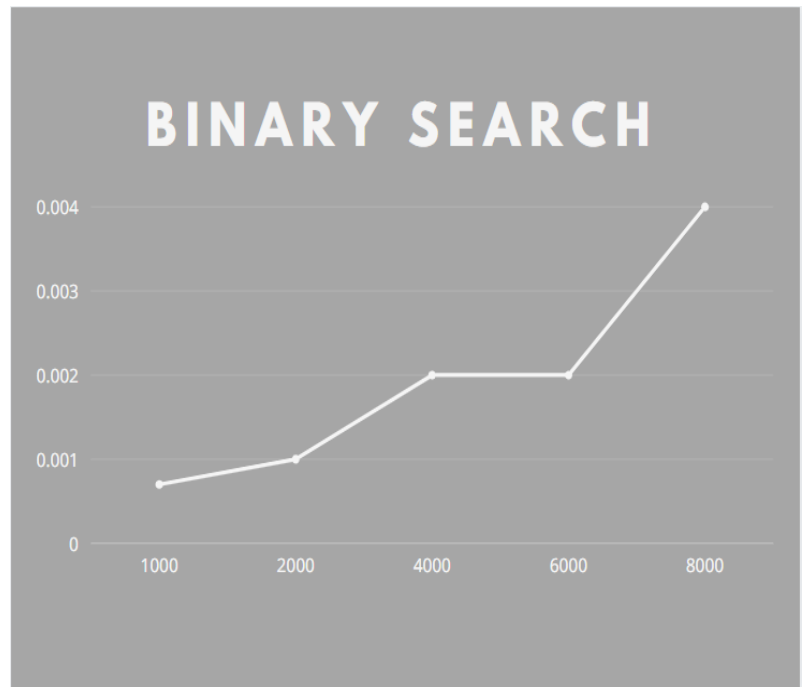
```

```

C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 2 ( Recursive Binary and Linear Search)\binarysearchrecursive.exe
Enter the no of elements
100
The elements in sorted array are :
41      153      288      292      491      778      1842      1869      2082      2995      3035      3548      3902      4664      4827
4966     5436     5447     5537     5705     6334     6729     6868     7376     7711     8723     8942     9040     9741
9894     9961     11323    11478    11538    11840    11942    12316    12382    12623    12859    13931    14604    14771
15006    15141    15350    15724    15890    16118    16541    16827    16944    17035    17421    17673    18467    18716
18756    19169    19264    19629    19718    19895    19912    19954    20037    21538    21726    22190    22648    22929
23281    23805    23811    24084    24370    24393    24464    24626    25547    25667    26299    26308    26500    26962
27446    27529    27644    28145    28253    28703    29358    30106    30333    31101    31322    32391    32439    32662
32757
Enter element to be searched
32757
Element found in binary search at pos 99
The time taken is 0.0050000000
-----
Process exited after 7.106 seconds with return value 0
Press any key to continue . . .

```

Binary Search Array size	Time in sec
1000	0.0007
2000	0.001
4000	0.002
6000	0.002
8000	0.004



### PROGRAM 3: Selection Sort

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 <stdlib.h>
#include <time.h>
int arr[1000000];

int swap(int arr[], int index1, int index2)
{
    int temp = arr[index2];
    arr[index2] = arr[index1];
    arr[index1] = temp;
}

int FindMin(int arr[], int n, int i)
{
    int minindex = i, temp;

    for (int j = i; j < n; j++)
    {
        if (arr[j] < arr[minindex])
        {
            minindex = j;
        }
    }
    return minindex;
}

void SelectionSort(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int min = FindMin(arr, n, i);
```

```

        swap(arr, min, i);
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    time_t start, end;
    int n;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        arr[i] = rand();
    }

    start = time(NULL);
    SelectionSort(arr, n);
    end = time(NULL);

    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

    printf("The time taken is %.10f\n", difftime(end, start) / CLOCKS_PER_SEC);
    return 0;
}

```



C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 3 ( Selection Sort and bubble sort )\selectionSort.exe

```
Enter the no of elements
20000
The array is sorted
The time taken is 0.0010000000

-----
Process exited after 9.846 seconds with return value 0
Press any key to continue . . .
```

Bubble Sort Array size	Time in sec
20000	0.001
40000	0.002
60000	0.004
80000	0.007
100000	0.011



## PROGRAM 4: BFS & DFS

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.

### BFS:

```
#include <stdio.h>
#include <conio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v)
{
    for (i = 1; i <= n; i++)
        if (a[v][i] && !visited[i])
            q[++r] = i;
    if (f <= r)
    {
        visited[q[f]] = 1;
        bfs(q[f++]);
    }
}

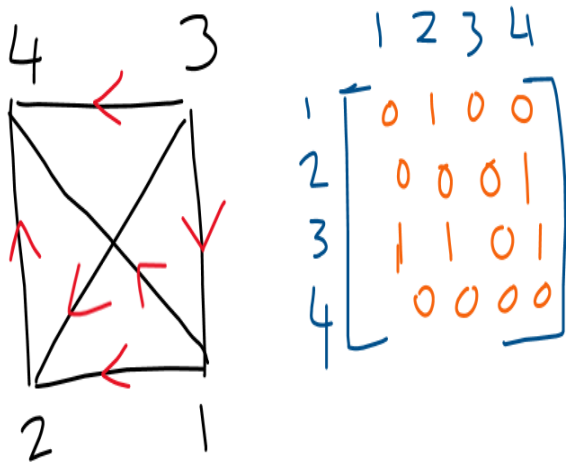
void main()
{
    int v;

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

```

for (i = 1; i <= n; i++)
{
    q[i] = 0;
    visited[i] = 0;
}
printf("\n Enter graph data in matrix form:\n");
for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        scanf("%d", &a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
printf("\n The node which are reachable are:\n");
for (i = 1; i <= n; i++)
    if (visited[i])
        printf("%d\t", i);
getch();
}

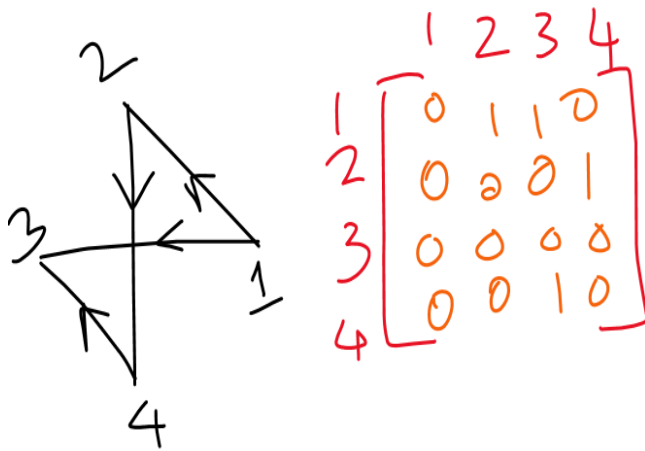
```



```

C:\softwares_sajjan\c code\lab_4_ada.exe
enter number of vertices
4
Enter adjacency matrix
0 1 0 1
0 0 0 1
1 1 0 1
0 0 0 0
enter starting vertex
1
The nodes reachable from src are
124
-----
Process exited after 39.12 seconds with return value 0
Press any key to continue . . .

```



```
C:\softwares_sajjan\c code\lab_4_ada.exe
enter number of vertices
4
Enter adjacency matrix
0 1 1 0
0 0 0 1
0 0 0 0
0 0 1 0
enter starting vertex
1
The nodes reachable from src are
1234
-----
Process exited after 35.31 seconds with return value 0
Press any key to continue . . .
```

## DFS:

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

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

```
int a[20][20], reach[20], n;
```

```
void dfs(int v)
```

```
{
    int i;
    reach[v] = 1;
    for (i = 1; i <= n; i++)
        if (a[v][i] && !reach[i])
        {
            printf("\n %d->%d", v, i);
            dfs(i);
        }
}
```

```
void main()
```

```
{
    int i, j, count = 0;
```

```

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

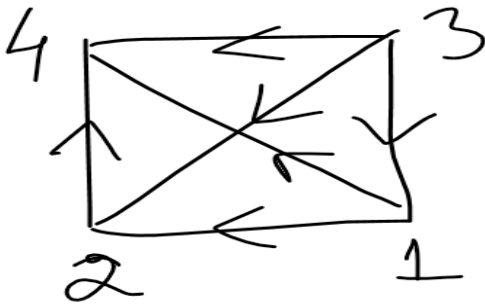
for (i = 1; i <= n; i++)
{
    reach[i] = 0;
    for (j = 1; j <= n; j++)
        a[i][j] = 0;
}

printf("\n Enter the adjacency matrix:\n");
for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        scanf("%d", &a[i][j]);

dfs(1);
printf("\n");
for (i = 1; i <= n; i++)
{
    if (reach[i])
        count++;
}

if (count == n)
    printf("\n Graph is connected");
else
    printf("\n Graph is not connected");
getch();
}

```



$$\begin{matrix} & 1 & 2 & 3 & 4 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 4 ( BFS and DFS)\ConnectedDFS.exe

```

Enter number of vertices:4

Enter the adjacency matrix:
0 1 0 1
0 0 0 1
1 1 0 1
0 0 0 0

1->2
2->4

Graph is not connected_

```

C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 4 ( BFS and DFS)\ConnectedDFS.exe

```

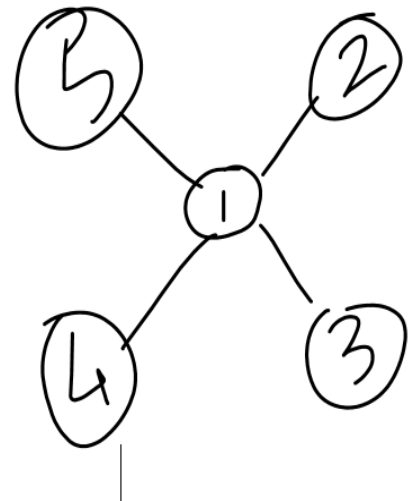
Enter number of vertices:5

Enter the adjacency matrix:
0 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0

1->2
1->3
1->4
1->5

Graph is connected

```



## PROGRAM 5: Insertion Sort

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>

int arr[1000000];
void insetionSort(int arr[], int n)
{
    int i,j,curr;
    for ( i = 1; i < n; i++)
    {
        curr = arr[i];
        j = i - 1;
        while (j >= 0 && curr < arr[j])
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = curr;
    }
}

void printArray(int arr[], int n)
{
    int i;
```

```

        for (i = 0; i < n; i++)
            printf("%d ", arr[i]);
        printf("\n");
    }

int main()
{
    time_t start, end;
    int n;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    int i;
    for ( i = 0; i < n; i++)
    {
        arr[i] = rand();
    }
    start = time(NULL);
    insetionSort(arr, n);
    end = time(NULL);

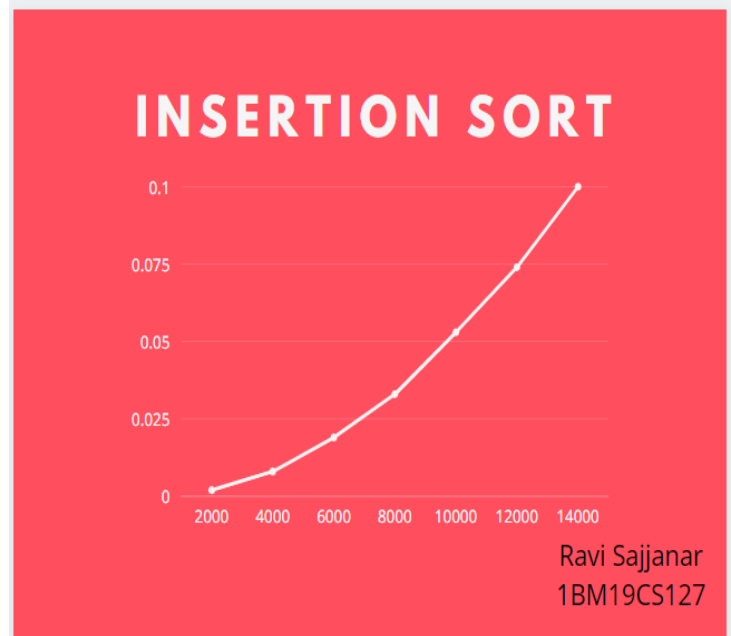
    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

    printf("The time taken is %.10f\n", difftime(end, start) / CLOCKS_PER_SEC);
    return 0;
}

```



Array Size	Time (in Sec)
2000	0.002
4000	0.008
6000	0.019
8000	0.033
10000	0.053
12000	0.074
14000	0.100



## PROGRAM 6: Topological Ordering

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

```
#include<stdio.h>
int temp[10],k=0;

void topo(int n,int indegree[10],int a[10][10])
{
    int i,j;

    for(i=1;i<=n;i++)
    {
        if(indegree[i]==0)
        {
            indegree[i]=1;
            temp[++k]=i;
            for(j=1;j<=n;j++)
            {
                if(a[i][j]==1&&indegree[j]!=-1)
                    indegree[j]--;
            }
            i=0;
        }
    }
}

void main()
{
    int i,j,n,indegree[10],a[10][10];
    printf("enter the number of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        indegree[i]=0;

    printf("\n enter the adjacency matrix\n");
    for(i=1;i<=n;i++)
```

```

for(j=1;j<=n;j++)
{
    scanf("%d",&a[i][j]);
    if(a[i][j]==1)
        indegree[j]++;
}

topo(n,indegree,a);

if(k!=n)
    printf("topological ordering is not possible\n");

else
{
    printf("\n topological ordering is :\n");
    for(i=1;i<=k;i++)
        printf("v%d\t",temp[i]);
}
}

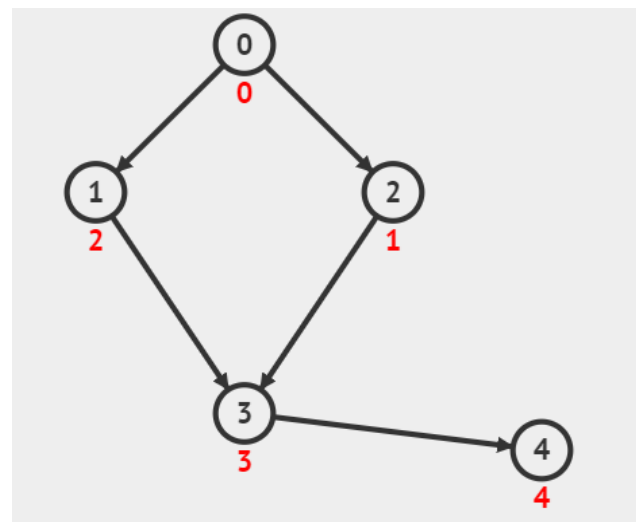
```

```

D:\codes\ADA Lab6.exe
enter the number of vertices:5
enter the adjacency matrix
0 1 1 0 0
0 0 0 1 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0

topological ordering is :
v0    v1    v2    v3    v4
-----
Process exited after 34.63 seconds with return value 0
Press any key to continue . . .

```



## Using DFS Technique

```
#include<stdio.h>

int res[10],top=0,s[10];
void dfs(int v,int n,int a[10][10]){
    s[v]=1;
    for(int i=1;i<=n;i++){
        if(s[i]==0&&a[v][i]==1) dfs(i,n,a);
    }
    top++;
    res[top]=v;
}

void topo(int n,int a[10][10])
{
    for(int i=1;i<=n;i++){
        s[i]=0;
    }
    top=0;
    for(int i=1;i<=n;i++){
        if(s[i]==0)dfs(i,n,a);
    }

}

void main()
{
    int i,j,n,vertices[10],a[10][10];
    printf("enter the number of vertices:");
    scanf("%d",&n);
```

```

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

printf("\n enter the adjacency matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
    scanf("%d",&a[i][j]);

}

topo(n,a);

if(top!=n)
printf("topological ordering is not possible\n");

else
{
    printf("\n topological ordering is :\n");
    for(int i=n;i>0;i--){
        printf("v%d\t",res[i]);
    }
}
}

```

```
enter the number of vertices:5
```

```
enter the adjacency matrix
```

```
0 1 1 0 0
0 0 0 1 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0
```

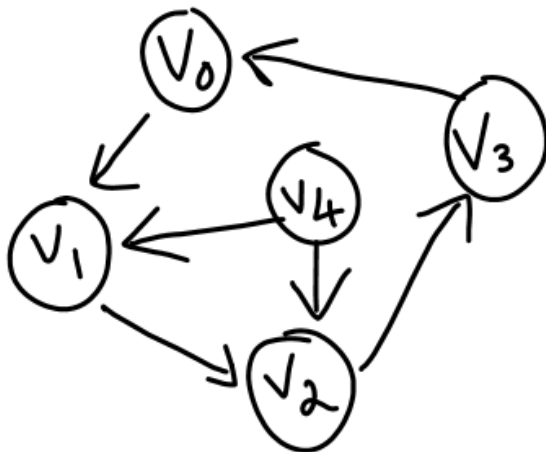
```
topological ordering is :
```

```
v1      v3      v2      v4      v5
```

```
-----
```

```
Process exited after 23.07 seconds with return value 0
```

```
Press any key to continue . . .
```



	$v_0$	$v_1$	$v_2$	$v_3$	$v_4$
$v_0$	0	1	0	0	0
$v_1$	0	0	1	0	0
$v_2$	0	0	0	1	0
$v_3$	1	0	0	0	0
$v_4$	0	1	1	0	0

## PROGRAM 7: Johnson Trotter

Implement Johnson Trotter algorithm to generate permutations.

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

```
int NN, i, count=0;
```

```
int p[100], pi[100];
```

```
int dir[100];
```

```
void PrintPerm()
```

```
{
```

```
    int i;
```

```
    count = count + 1;
```

```
    printf( "[%2d] ", count );
```

```
    for (i=1; i <= NN; ++i)
```

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

```
}
```

```
void PrintTrans( int x, int y )
```

```
{
```

```
    // printf( " (%d %d)", x, y );
```

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

```
}
```

```
void Move( int x, int d )
```

```

{
    int z;
    PrintTrans( pi[x], pi[x]+d );
    z = p[pi[x]+d];
    p[pi[x]] = z;
    p[pi[x]+d] = x;
    pi[z] = pi[x];
    pi[x] = pi[x]+d;
}

```

```

void Perm ( int n )
{
    int i;
    if (n > NN)
        PrintPerm();
    else
    {
        Perm( n+1 );
        for (i=1; i<=n-1; ++i)
        {
            Move( n, dir[n] );
            Perm( n+1 );
        }
        dir[n] = -dir[n];
    }
}

int main ()

```



```
{  
    printf( "Enter n: " );  
    scanf( "%d", &NN );  
    printf( "\n" );  
    for (i=1; i<=NN; ++i)  
    {  
        dir[i] = -1; p[i] = i;  
        pi[i] = i;  
    }  
    Perm ( 1 );  
    printf( "\n" );  
}
```

```
D:\codes\LAB7.exe
Enter n: 4
[ 1] 1234
[ 2] 1243
[ 3] 1423
[ 4] 4123
[ 5] 4132
[ 6] 1432
[ 7] 1342
[ 8] 1324
[ 9] 3124
[10] 3142
[11] 3412
[12] 4312
[13] 4321
[14] 3421
[15] 3241
[16] 3214
[17] 2314
[18] 2341
[19] 2431
[20] 4231
[21] 4213
[22] 2413
[23] 2143
[24] 2134

-----
Process exited after 2.394 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM 8: Merge Sort

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

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

int arr[1000000];
void merge(int arr[], int p, int q, int r)
{
    int n1 = q - p + 1;
    int n2 = r - q;

    int L[n1], M[n2];
    int i,j,k;
    for ( i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for ( j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];

    i = 0;
    j = 0;
    k = p;

    while (i < n1 && j < n2)
    {
        if (L[i] <= M[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {

```

```

        arr[k] = M[j];
        j++;
    }
    k++;
}

while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

while (j < n2)
{
    arr[k] = M[j];
    j++;
    k++;
}
}

void mergeSort(int arr[], int l, int r)
{
    int i,j;
    for( i=0;i<80;i++)
    {
        for( j=0;j<40 ; j++)
        {

        }
    }
    if (l < r)
    {

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

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
    }
}

```

```

        merge(arr, l, m, r);
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    time_t start, end;
    int n,i;
    srand(time(0));
    printf("Enter the no of elements \n");
    scanf("%d", &n);

    for ( i = 0; i < n; i++)
    {
        arr[i] = rand();
    }

    start = time(NULL);
    mergeSort(arr,0,n-1);
    end = time(NULL);

    printf("The array is sorted\n");
    // printf("The sorted array is: \n");
    // printArray(arr, n);

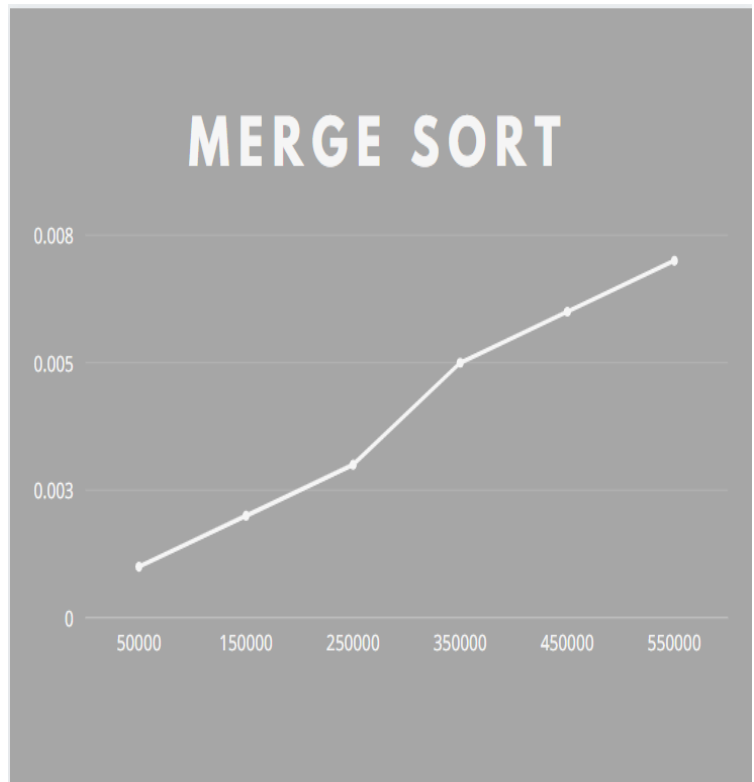
    printf("The time taken is %.10f\n", difftime(end, start) / CLOCKS_PER_SEC);
    return 0;
}

```

```
C:\Users\ravis\Desktop\IRET\Labs\ADA\ADA-Lab-master\Lab Prog 8 ( Merge Sort )\mergeSort.exe
Enter the no of elements
50000
The array is sorted
The time taken is 0.0010000000

-----
Process exited after 3.004 seconds with return value 0
Press any key to continue . . .
```

Merge Sort Array size	Time in sec
50000	0.001
150000	0.002
250000	0.003
350000	0.005
450000	0.006
550000	0.007



## PROGRAM 9: Quick Sort

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

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

void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high- 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```

    }
}
int main()
{
    int arr[100000],size,i;
    clock_t timereq;
    float cpu_time;

    //printf("Enter size\n");
    //scanf("%d",&size);
    for(size=25000;size<=100000;size=size+10000){
        i=0;

        srand(time(NULL));
        for(int i=0;i<size;i++){
            arr[i]=rand()%100;
            // printf("%d\t",arr[i]);
        }
        printf("\n");


        timereq= clock();
        quickSort(arr, 0, size-1);
        timereq=clock()-timereq;
        cpu_time=((float)(timereq))/CLK_TCK;
        printf("Check: ");
        for(i=0;arr[i]<=arr[i+1] && i<size ;i++);
        if(size-1 == i)
            printf("All the elements are Sorted\n");
        else
            printf("Elements are NOT Sorted\n");

        printf("Time taken to sort an arr[%d] is %f seconds\n ",size,cpu_time);

    }
    return 0;
}

```



 D:\codes\quicksort.exe

Note: All the elements are Sorted

Time taken in seconds for arr 25000: 0.007000

Note: All the elements are Sorted

Time taken in seconds for arr 35000: 0.013000

Note: All the elements are Sorted

Time taken in seconds for arr 45000: 0.021000

Note: All the elements are Sorted

Time taken in seconds for arr 55000: 0.031000

Note: All the elements are Sorted

Time taken in seconds for arr 65000: 0.042000

Note: All the elements are Sorted

Time taken in seconds for arr 75000: 0.055000

Note: All the elements are Sorted

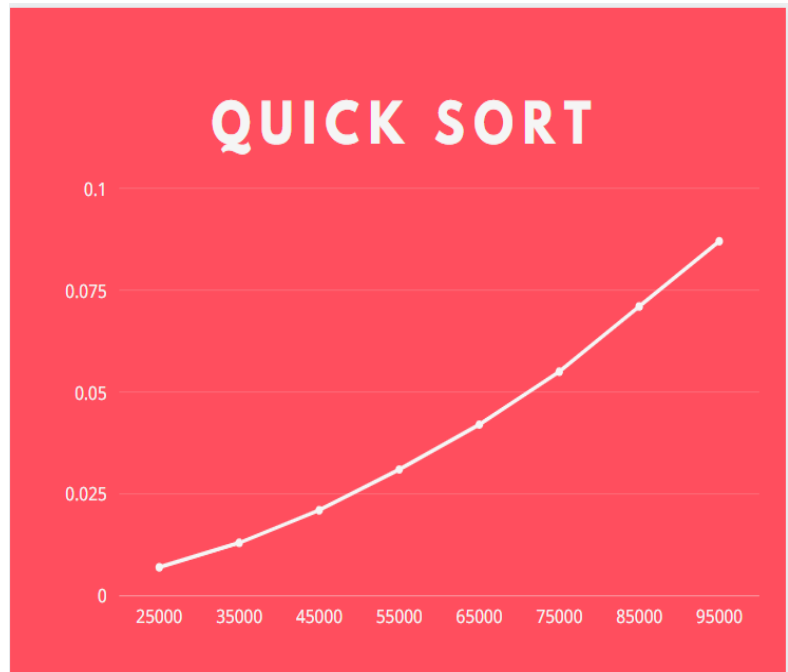
Time taken in seconds for arr 85000: 0.071000

Note: All the elements are Sorted

Time taken in seconds for arr 95000: 0.087000

-----  
Process exited after 3.772 seconds with return value 0  
Press any key to continue . . . █

25000	0.007
35000	0.013
45000	0.021
55000	0.031
65000	0.042
75000	0.055
85000	0.071
95000	0.087



## **PROGRAM 10: Heap Sort**

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

## **PROGRAM 11: Warshall's algorithm**

Implement Warshall's algorithm using dynamic programming.

## **PROGRAM 12: Knapsack problem**

Implement 0/1 Knapsack problem using dynamic programming.

## **PROGRAM 13: Floyd's algorithm**

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

### **PROGRAM 14: Prim's algorithm.**

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

## **PROGRAM 15: Kruskals algorithm**

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



## **PROGRAM 16: Dijkstra's algorithm**

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

## **PROGRAM 17: Sum of Subsets” problem**

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.

## **PROGRAM 18: N-Queens Problem**

Implement “N-Queens Problem” using Backtracking.