

# **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

"JnanaSangama", Belgaum -590014, Karnataka.



## **LAB REPORT**

**on**

## **ANALYSIS AND DESIGN OF ALGORITHMS**

*Submitted by*

**VENU GOPAL V (1BM21CS414)**

*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 **venu gopal V(1BM21CS414)**, 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 an **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Dr Manjunath D R

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.
<b>1</b>	Write a recursive program to a. Solve Towers-of-Hanoi problem b. To find GCD	<b>1</b>
<b>2</b>	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.	<b>3</b>
<b>3</b>	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.	<b>7</b>
<b>4</b>	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.	<b>10</b>
<b>5</b>	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	<b>14</b>
<b>6</b>	Write program to obtain the Topological ordering of vertices in a given digraph.	<b>16</b>
<b>7</b>	Implement Johnson Trotter algorithm to generate permutations	<b>18</b>
<b>8</b>	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.	<b>22</b>
<b>9</b>	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	<b>25</b>
<b>10</b>	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	
<b>11</b>	Implement Wars hall's algorithm using dynamic programming.	<b>28</b>
<b>12</b>	Implement 0/1 Knapsack problem using dynamic programming.	<b>30</b>
<b>13</b>	Implement All Pair Shortest paths problem using Floyd's algorithm.	<b>34</b>
<b>14</b>	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	<b>37</b>
<b>15</b>	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm	<b>40</b>
<b>16</b>	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	<b>43</b>
<b>17</b>	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.	

<b>18</b>	Implement “N-Queens Problem” using Backtracking.	
-----------	--	--

## Course Outcome

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

## EXPERIMENT-01

Write a recursive program to a. Solve Towers-of-Hanoi problem b. To find GCD

```
#include <stdio.h>

#include<time.h>

#include<stdlib.h>

#include<windows.h>

int gcd(int m,int n){
    if(n==0)
        return m;
    return(n,m%n);
}

void tower(int n,int s,int temp,int d){
    if(n==0)
        return;
    tower(n-1,s,d,temp);
    printf("Move the disc %d from %c to %c\n",n,s,d);
    tower(n-1,temp,s,d);
}

void main(){
    int ch,n1,m,n,result;
    while(1){
        printf("\nEnter the choice:1.Tower of hanoi 2.GCD\n");
        scanf("%d",&ch);
        switch(ch){
        case 1:
            printf("\nEnter the number of discs:\t");
            scanf("%d",&n1);
            tower(n1,'A','B','C');
```

```

        break;
case 2:
    printf("Enter the value of m:\t");
    scanf("%d",&m);
    printf("Enter the value of n:\t");
    scanf("%d",&n);
    result=gcd(m,n);
    printf("GCD is %d",result);
    break;
default:exit(0);
}
}
}

```

```

Enter the choice:1.Tower of hanoi 2.GCD
1
Enter the number of discs:      3
Move the disc 1 from A to C
Move the disc 2 from A to B
Move the disc 1 from C to B
Move the disc 3 from A to C
Move the disc 1 from B to A
Move the disc 2 from B to C
Move the disc 1 from A to C
Enter the choice:1.Tower of hanoi 2.GCD
2
Enter the value of m:   6
Enter the value of n:   4
GCD is 2
Enter the choice:1.Tower of hanoi 2.GCD

```

## EXPERIMENT-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>

int RecursiveLS(int arr[], int key, int index, int n)
{
    int flag = 0;
    if(index >= n)
    { return 0;
    }
    else if (arr[index] == key)
    {
        flag = index + 1;
        return flag;
    }
    else
    { return RecursiveLS(arr, key , index+1, n);
    }
    return flag;
}

int RecursiveBN(int arr[],int key,int index,int n){
int low=0,high=n-1,mid;
while(low<=high){
    mid=(low+high)/2;
    if(key==arr[mid])
        return mid;
    else if(key<arr[mid])
```

```

        high=mid-1;
    else
        low=mid+1;
    }
    return -1;
}

int main()
{
    int n,i,j, key , pos , m = 0, arr[10000],ch;
    clock_t s,e;
    while(1){
        printf("\nEnter the choice:1.Linear search 2.Binary search\n");
        scanf("%d",&ch);
        switch(ch){
            case 1: n=1000;
                while(n<10000){
                    for(i=0;i<n;i++){
                        arr[i];
                    }
                    key=arr[n-100];
                    s=clock();
                    pos= RecursiveLS(arr, key , 0, n);
                    if(pos!=0)
                        printf("\n Element found ");
                    else
                        printf("element not found");
                    for(j=0;j<80000000;j++){
                    }
                    e=clock();
                    printf("\n Time taken for %d elements is %f\n",n,((double)(e-s))/CLK_TCK);

```



```

n=n+1000;
}
break;
case 2: n=1000;
while(n<10000){
for(i=0;i<n;i++){
    arr[i];
}
key=arr[n-100];
s=clock();
pos= RecursiveBN(arr, key , 0, n);;
if(pos!=0)
printf("\n Element found");
else
printf("element not found");
for(j=0;j<80000000;j++){
e=clock();
printf("\n Time taken for %d elements is %f\n",n,((double)(e-s))/CLK_TCK);
n=n+1000;
}
break;
default: exit(0);
}
}
return 0;
}

```

Enter the choice:1.Linear search 2.Binary search  
1

Element found  
Time taken for 1000 elements is 0.161000

Element found  
Time taken for 2000 elements is 0.159000

Element found  
Time taken for 3000 elements is 0.160000

Element found  
Time taken for 4000 elements is 0.161000

Element found  
Time taken for 5000 elements is 0.158000

Element found  
Time taken for 6000 elements is 0.160000

Element found  
Time taken for 7000 elements is 0.158000

Element found  
Time taken for 8000 elements is 0.159000

Element found  
Time taken for 9000 elements is 0.160000

Element found  
Time taken for 10000 elements is 0.162000

Enter the choice:1.Linear search 2.Binary search  
2

Element found  
Time taken for 1000 elements is 0.157000

Element found  
Time taken for 2000 elements is 0.158000

Element found  
Time taken for 3000 elements is 0.159000

Element found  
Time taken for 4000 elements is 0.162000

Element found  
Time taken for 5000 elements is 0.156000

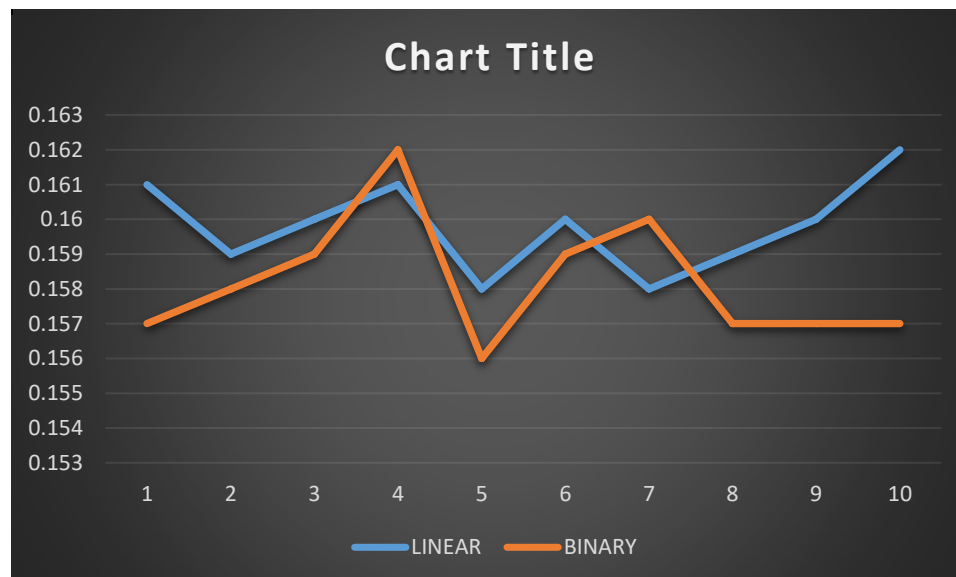
Element found  
Time taken for 6000 elements is 0.159000

Element found  
Time taken for 7000 elements is 0.160000

Element found  
Time taken for 8000 elements is 0.157000

Element found  
Time taken for 9000 elements is 0.157000

Element found  
Time taken for 10000 elements is 0.157000



## EXPERIMENT-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<windows.h>

#include<stdlib.h>

void selection_sort(int n,int array[]){

    int i,j,min,pos,temp;

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

        pos=i;

        min=array[i];

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

            if(array[j]<min){

                pos=j;

                min=array[j];}

        }

        temp=array[i];

        array[i]=array[pos];

        array[pos]=temp;

    }

}

void main(){

int array[10000],n,i,num;

clock_t s,e;

printf("Selection sort!");

n=1000;

while(n<=5000){

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

```

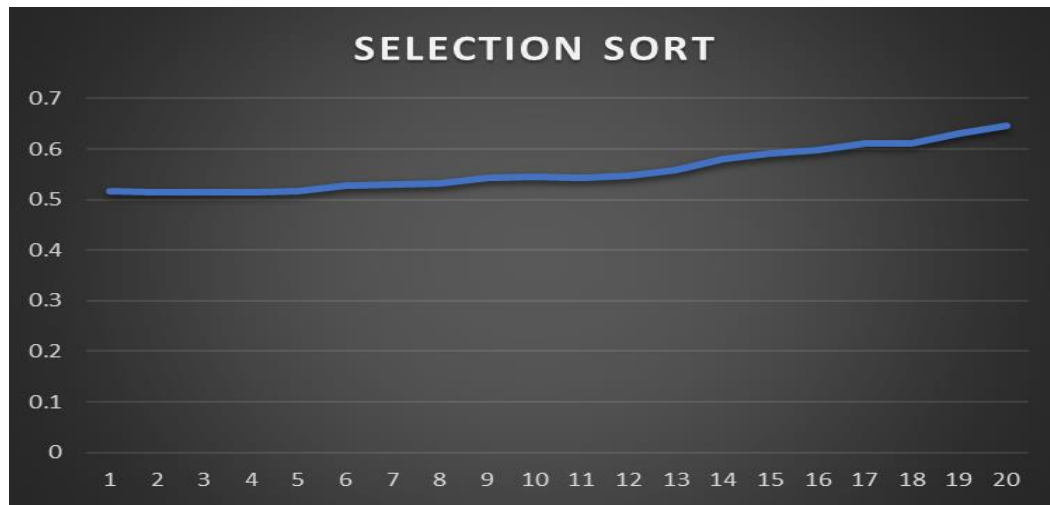
    array[i]=n-i;
}
s=clock();
selection_sort(n,array);
Sleep(500);
e=clock();
printf("\n\nTime taken for n=%d in selection sort is:%f",n,((double)(e-s))/CLK_TCK);
n=n+1000;
}
}

```

```

C:\Users\yathri\Desktop\ADA\SELECTION_SORT.exe
Selection sort!
Time taken for n=500 in selection sort is:0.516000
Time taken for n=1000 in selection sort is:0.515000
Time taken for n=1500 in selection sort is:0.514000
Time taken for n=2000 in selection sort is:0.515000
Time taken for n=2500 in selection sort is:0.517000
Time taken for n=3000 in selection sort is:0.527000
Time taken for n=3500 in selection sort is:0.529000
Time taken for n=4000 in selection sort is:0.531000
Time taken for n=4500 in selection sort is:0.542000
Time taken for n=5000 in selection sort is:0.546000
Time taken for n=5500 in selection sort is:0.544000
Time taken for n=6000 in selection sort is:0.548000
Time taken for n=6500 in selection sort is:0.559000
Time taken for n=7000 in selection sort is:0.580000
Time taken for n=7500 in selection sort is:0.592000
Time taken for n=8000 in selection sort is:0.598000
Time taken for n=8500 in selection sort is:0.611000
Time taken for n=9000 in selection sort is:0.610000
Time taken for n=9500 in selection sort is:0.630000
Time taken for n=10000 in selection sort is:0.645000
Process returned 0 (0x0) execution time : 12.009 s
Press any key to continue.

```



## EXPERIMENT-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.**

### **BFS**

```
#include<stdio.h>

#include<stdlib.h>

void bfs(int a[20][20],int s,int visited[20],int n){

int q[20],front,rear,u,v;

front=rear=0;

q[rear]=s;

visited[s]=1;

while(front<=rear){

u=q[front];

front=front+1;

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

if(visited[v]!=1&&a[u][v]==1){

visited[v]=1;

rear=rear+1;

q[rear]=v;

}

}

}

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

if(visited[v]==0)

printf("Node %d is not reachable\n",v);

else

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

}
```

```

}

void main(){
int n,a[20][20],j,i,visited[20],s;
printf("BFS ");
printf("Enter the number of vertex:\t");
scanf("%d",&n);
printf("Enter adjacency matrix of the graph:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("Enter the source vertex:\t");
scanf("%d",&s);
for(i=1;i<=n;i++){
visited[i]=0;}
bfs(a,s,visited,n);
}

```

### **DFS**

```

#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("\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("\n enter the source node:\t");
scanf("%d",&src);
ans=dfs(src);
if(ans==1)
{
    printf("\n graph is connected\n");
}
else
{
    printf("\n graph 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;
}

```

```

BFS Enter the number of vertex: 4
Enter adjacency matrix of the graph:
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
Enter the source vertex:      1
Node 1 is reachable
Node 2 is reachable
Node 3 is reachable
Node 4 is reachable

Process returned 5 (0x5)   execution time : 17.961 s
Press any key to continue.

```

```

enter the no of nodes: 4

enter the adjacency matrix:
0 1 1 0
1 0 1 0
1 1 0 0
0 0 0 0

enter the source node:  1

graph is not connected

```

## EXPERIMENT-05

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

```
#include<stdio.h>

#include<time.h>

#include<windows.h>

#include<stdlib.h>

int insertion_sort(int a[],int n){

int i,j,item;

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

    item=a[i];

    j=i-1;

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

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

        j=j-1;

    }

    a[j+1]=item;

}

}

void main(){

int a[10000],i,j,n,num,u=0,l=400,result ;

clock_t s,e;

printf("Insertion sort\n");

n=1000;

while(n<=10000){

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

    a[i]=n-i;

}

s=clock();
```

```

result= insertion_sort(a,n);

Sleep(150);

e=clock();

printf("\nTime taken for Insertion_sort where n : %d is: %f\n",n,((double)(e-s))/CLK_TCK);

n=n+1000;

}

}

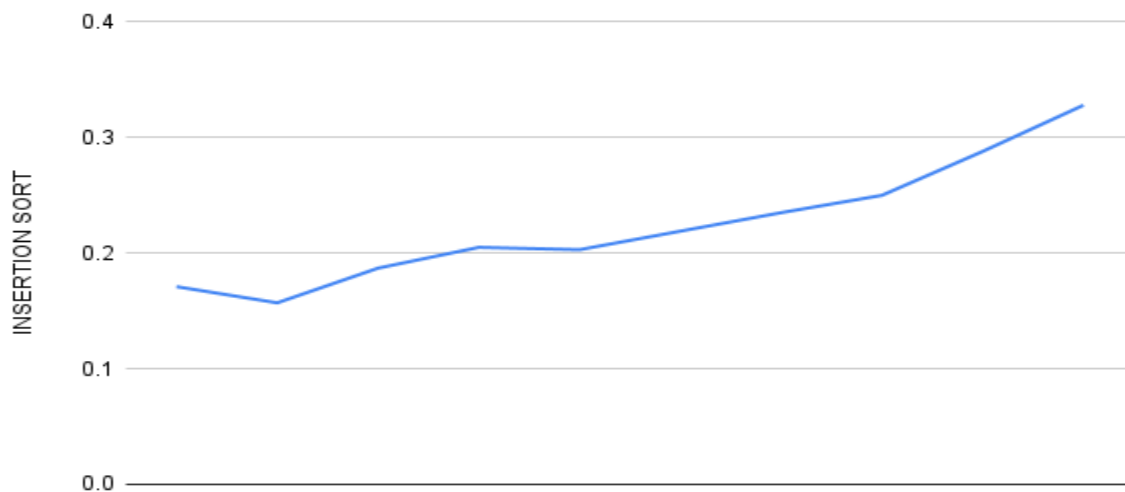
```

```

C:\Users\bmsce\Documents\1BM21CS416\insertion_sort.exe
Insertion sort
Time taken for Insertion_sort where n : 1000 is: 0.171000
Time taken for Insertion_sort where n : 2000 is: 0.172000
Time taken for Insertion_sort where n : 3000 is: 0.172000
Time taken for Insertion_sort where n : 4000 is: 0.203000
Time taken for Insertion_sort where n : 5000 is: 0.219000
Time taken for Insertion_sort where n : 6000 is: 0.219000
Time taken for Insertion_sort where n : 7000 is: 0.234000
Time taken for Insertion_sort where n : 8000 is: 0.266000
Time taken for Insertion_sort where n : 9000 is: 0.281000
Time taken for Insertion_sort where n : 10000 is: 0.297000
Process returned 60 (0x3C)   execution time : 2.687 s
Press any key to continue.

```

## INSERTION SORT



## EXPERIMENT-06

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

```
#include<stdio.h>
#include<conio.h>
void topology(int n,int a[10][10]){
    int j,sum=0,i,top=-1,sequence[10],u,indeg[10],res[10],k=0,v;
    for(j=0;j<n;j++){
        sum=0;
        for(i=0;i<n;i++){
            sum=sum+a[i][j];
        }
        indeg[j]=sum;
    }
    for(i=0;i<n;i++){
        if(indeg[i]==0){
            top=top+1;
            sequence[top]=i;
        }
    }
    while(top!=-1){
        u=sequence[top];
        top=top-1;
        res[k++]=u;
        for(v=0;v<n;v++){
            if(a[u][v]==1){
                indeg[v]=indeg[v]-1;
                if(indeg[v]==0){
                    top=top+1;
                    sequence[top]=v;
                }
            }
        }
    }
}
```

```

    } } }
}

printf("Sequence is:\n");
for(int l=0;l<n;l++){
    printf("%d\t",res[l]);
}
}

void main(){
    int n,a[10][10],i,j;
    printf("Enter the no.of vertex:\t");
    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]);
    topology(n,a);
}

```

```

Enter the no.of vertex: 6
Enter the adjacency matrix:
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0
Sequence is:
1      4      0      2      3      5
Process returned 6 (0x6)   execution time : 65.069 s
Press any key to continue.

```

## EXPERIMENT-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);
int i;
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");
for (int i = 0; i < n; i++)
{
a[i] = i + 1;
printf("%d \n", a[i]);
printf("\n");
}
}

```

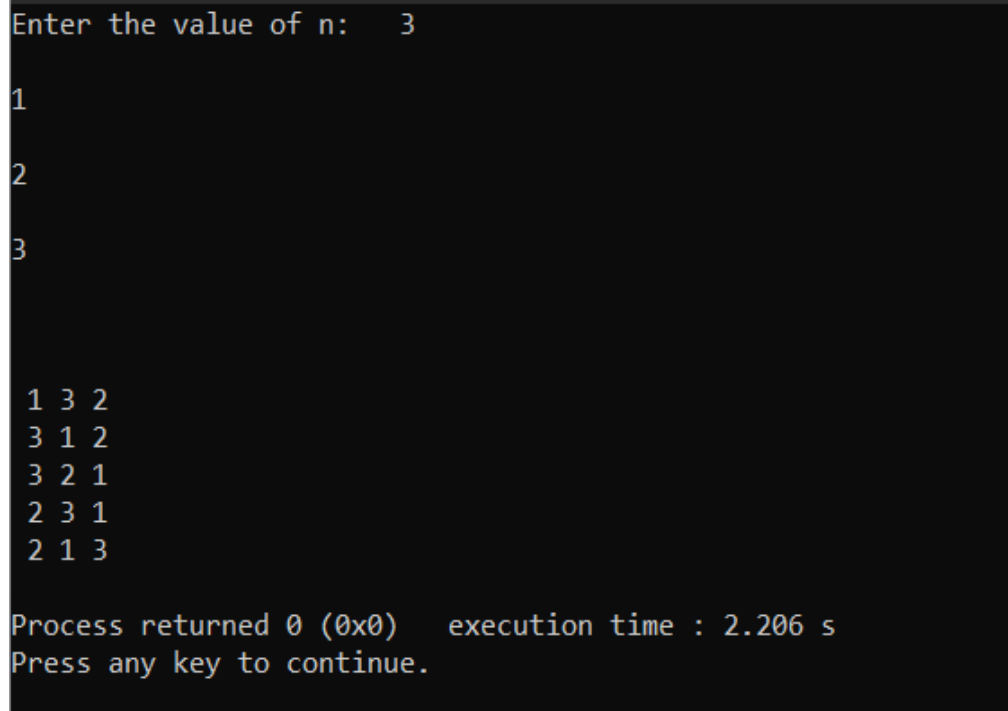


```

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()
{
    printf("Enter the value of n: \t");
    int n;
    scanf("%d",&n);
    printPermutation(n);
    return 0;
}

```



```

Enter the value of n: 3

1
2
3

1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
1 2 3

Process returned 0 (0x0) execution time : 2.206 s
Press any key to continue.

```

## EXPERIMENT-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<conio.h>
#include<time.h>
#include<windows.h>
#include<stdlib.h>
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);
        mergesort(a,low,mid,high);
    }
}
void mergesort(int a[],int low,int mid,int high){
    int i,j,k,c[10000];
    i=low;
    j=mid+1;
    k=low;
    while(i<=mid && j<=high){
        if(a[i]<a[j]){
            c[k]=a[i];
            i=i+1;
            k=k+1;
```

```

    }
    else{
        c[k]=a[j];
        j=j+1;
        k=k+1;
    }
}

if(j>high){
while(i<=mid){
    c[k]=a[i];
    k=k+1;
    i=i+1;
}
}

if(i>mid){
while(j<=high){
    c[k]=a[j];
    k=k+1;
    j=j+1;
}
}

}

void main ()
{
int a[10000],i,j,k,n;
clock_t s,e;
printf("Merge sort\n");
n=1000;
while(n<10000){

```

```

for(i=0;i<n;i++){
    a[i]=n-i;
}
s=clock();
split(a,0,n-1);
Sleep(150);
e=clock();
printf("\nTime taken for Merge sort where n : %d is: %f\n",n,((double)(e-s))/CLK_TCK);
n=n+1000;
}
}

```

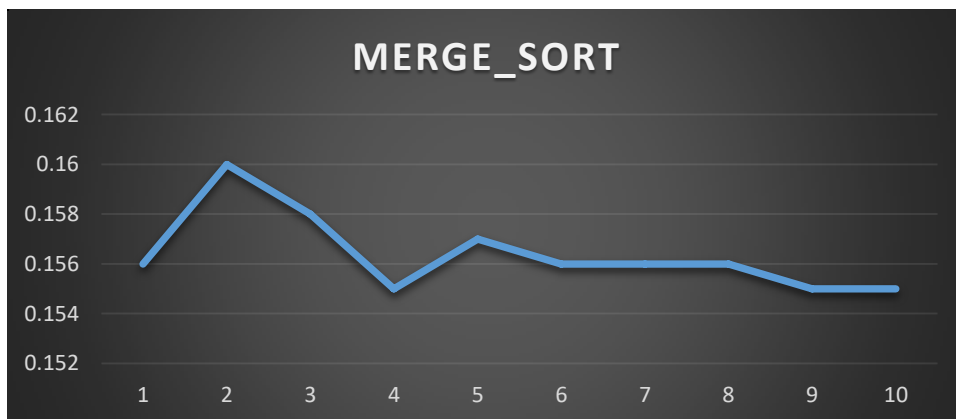
```

Merge sort

Time taken for Merge sort where n : 1000 is: 0.156000
Time taken for Merge sort where n : 2000 is: 0.160000
Time taken for Merge sort where n : 3000 is: 0.158000
Time taken for Merge sort where n : 4000 is: 0.155000
Time taken for Merge sort where n : 5000 is: 0.157000
Time taken for Merge sort where n : 6000 is: 0.156000
Time taken for Merge sort where n : 7000 is: 0.156000
Time taken for Merge sort where n : 8000 is: 0.155000
Time taken for Merge sort where n : 9000 is: 0.155000

Process returned 55 (0x37)   execution time : 1.431 s
Press any key to continue.

```



## EXPERIMENT-09

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

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

void quick_sort(int a[],int low,int high){
    int mid,i,j;
    i=low;
    j=high;
    mid=(low+high)/2;
    if(low>high)
        return;
    quick_sort(a,i,mid-1);
    quick_sort(a,mid+1,j);
}

void main ()
{
    int a[10000],i,j,k,n;
    clock_t s,e;
    printf("Quick sort\n");
    n=1000;
    while(n<=10000){
        for(i=0;i<n;i++){
            a[i]=n-i;
        }
        s=clock();
        quick_sort(a,0,n-1);
```

```
Sleep(150);  
e=clock();  
printf("\nTime taken for Quick sort where n : %d is: %f\n",n,((double)(e-s))/CLK_TCK);  
n=n+1000;  
}  
}
```

```
Quick sort  
Time taken for Quick sort where n : 1000 is: 0.164000  
Time taken for Quick sort where n : 2000 is: 0.157000  
Time taken for Quick sort where n : 3000 is: 0.156000  
Time taken for Quick sort where n : 4000 is: 0.156000  
Time taken for Quick sort where n : 5000 is: 0.155000  
Time taken for Quick sort where n : 6000 is: 0.158000  
Time taken for Quick sort where n : 7000 is: 0.157000  
Time taken for Quick sort where n : 8000 is: 0.157000  
Time taken for Quick sort where n : 9000 is: 0.157000  
Time taken for Quick sort where n : 10000 is: 0.156000  
  
Process returned 56 (0x38)   execution time : 1.601 s  
Press any key to continue.
```



## EXPERIMENT-10

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

## EXPERIMENT-11

### Implement Warshall's algorithm using dynamic programming.

```
#include<stdio.h>

#include<math.h>

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");
}
}

```

```

Enter the number of vertices:4
Enter the number of edges:6
Enter the end vertices of edge 1:1
2
Enter the end vertices of edge 2:2
4
Enter the end vertices of edge 3:4
3
Enter the end vertices of edge 4:1
3
Enter the end vertices of edge 5:4
1
Enter the end vertices of edge 6:3
2
Matrix of input data:
0  1  1  0
0  0  0  1
0  1  0  0
1  0  1  0

Transitive closure:
1  1  1  1
1  1  1  1
1  1  1  1
1  1  1  1

```

## EXPERIMENT-12

### Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

#include<time.h>

void knapsack();

int max(int,int);

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

void main()

{

    clock_t start , end;

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

    scanf("%d",&n);

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

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

    {

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

    }

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

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

    {

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

    }

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

    scanf("%d",&m);

    start=clock();

    knapsack();

    end=clock();

    printf("\nThe time taken to run this knapsack pgm is %f", (((double)(end-start))/CLOCKS_PER_SEC));

}
```

```

void knapsack()
{
    int x[10];
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=m;j++)
        {
            if(i==0 || j==0)
            {
                v[i][j]=0;
            }
            else if(j-w[i]<0)
            {
                v[i][j]=v[i-1][j];
            }
            else
            {
                v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);
            }
        }
    }
    printf("\nthe output is:\n");
    for(i=0;i<=n;i++)
    {
        for(j=0;j<=m;j++)
        {
            printf("%d\t",v[i][j]);
        }
    }
}

```

```

printf("\n\n");
}
printf("\nthe optimal solution is %d",v[n][m]);
printf("\nthe solution vector is:\n");
for(i=n;i>=1;i--)
{
if(v[i][m]!=v[i-1][m])
{
x[i]=1;
m=m-w[i];
}
else
{
x[i]=0;
}
}
for(i=1;i<=n;i++)
{
printf("%d\t",x[i]);
}
}

```

```

int max(int x,int y)
{
if(x>y)
{
return x;
}
else

```

```
{  
    return y;  
}  
}
```

```
enter the no. of items: 3  
enter the weight of the each item:  
1 2 3  
enter the profit of each item:  
15 10 20  
enter the knapsack's capacity: 5  
the output is:  
0 0 0 0 0 0  
  
0 15 15 15 15 15  
  
0 15 15 25 25 25  
  
0 15 15 25 35 35  
  
the optimal solution is 35  
the solution vector is:  
1 0 1  
The time taken to run this knapsack pgm is 0.000061|
```

## EXPERIMENT-13

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

```
#include <stdio.h>

int a[10][10],n;

void floyds()
{
    int i,j,k;
    for(k=1;k<=n;k++)
    {
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
            }
        }
    }
    printf("\nall pair shortest path matrix is:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n\n");
    }
}

int min(int x,int y)
```

```

{
    if(x<y)
    {
        return x;
    }
    else
    {
        return y;
    }
}

void main()
{
    int i,j;
    printf("\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",&a[i][j]);
        }
    }

    floyds();
}

```

```
enter the no. of vertices: 4
enter the cost matrix:
0 10 999 999
999 0 50 999
40 999 0 30
999 20 999 0
all pair shortest path matrix is:
0 10 60 90

90 0 50 80

40 50 0 30

110 20 70 0
```



## EXPERIMENT-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);

```

```
}
```

```
enter the no. of vertices: 4
```

```
enter the cost matrix:
```

```
0 10 999 999
```

```
999 0 50 999
```

```
40 999 0 30
```

```
999 20 999 0
```

```
1----->2=10
```

```
2----->3=50
```

```
3----->4=30
```

```
mincost=90
```

## EXPERIMENT-15

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

```
#include<stdio.h>

void kruskals();

int c[10][10],n;

void main()

{

    int i,j;

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

    scanf("%d",&n);

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

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

    {

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

        {

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

        }

    }

    kruskals();

}

void kruskals()

{

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

    int ne=0,mincost=0;

    int parent[10];

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

    {

        parent[i]=0;
```

```

}
while(ne!=n-1)
{
    min=9999;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(c[i][j]<min)
            {
                min=c[i][j];
                u=a=i;
                v=b=j;
            }
        }
    }
    while(parent[u]!=0)
    {
        u=parent[u];
    }
    while(parent[v]!=0)
    {
        v=parent[v];
    }
    if(u!=v)
    {
        printf("\n%d----->%d=%d\n",a,b,min);
        parent[v]=u;
        ne=ne+1;
    }
}

```

```

    mincost=mincost+min;
}
c[a][b]=c[b][a]=9999;
}
printf("\nmincost=%d",mincost);
}

```

```

enter the no. of vertices: 4
enter the cost matrix:
0 10 999 999
999 0 50 999
40 999 0 30
999 20 999 0
0 10 999 999

999 0 50 999

40 999 0 30

999 20 999 0

1----->2=10

4----->2=20

3----->4=30

mincost=60

```

## EXPERIMENT-16

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

```
#include<stdio.h>

void dijkstras();

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

void main()
{
    int i,j;

    printf("\nEnter the no of vertices:\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]);
        }
    }

    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]);

}

}

```

```

enter the no of vertices: 4
enter the cost matrix:
0 10 999 999
999 0 20 999
999 999 0 30
40 15 999 0
enter the source node: 1
the shortest distance is:

1----->1=0
1----->2=10
1----->3=30
1----->4=60

```

## EXPERIMENT-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 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.

## EXPERIMENT-18

Implement “N-Queens Problem” using Backtracking.