

ISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

KOTTURU AMARNATH (1BM20CS074)

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 **KOTTURU AMARNATH (1BM20CS074)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Dr. Rajeshwari B S
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	
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.	
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.	
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.	
5	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	
6	Write program to obtain the Topological ordering of vertices in a given digraph.	
7	Implement Johnson Trotter algorithm to generate permutations.	
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.	
9	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	
10	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	
11	Implement Warshall's algorithm using dynamic programming	
12	Implement 0/1 Knapsack problem using dynamic programming.	
13	Implement All Pair Shortest paths problem using Floyd's algorithm.	
14	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	
15	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.	
16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	
17	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.	
18	Implement “N-Queens Problem” using Backtracking.	

Course Outcome

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

1. Write a recursive program to Solve

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

Program:

a)

```
#include<stdio.h>
void TOH(int n,char S,char T,char D){
    if(n==1)
        printf("move disk 1 from %c to %c \n",S,D);
    else{
        TOH(n-1,S,D,T);
        printf("move disk %d from %c to %c\n",n,S,D);
        TOH(n-1,T,S,D);
    }
}
int main(){
    int n;
    printf("Enter no of disks:");
    scanf("%d",&n);
    TOH(n,'S','T','D');
}
```

Result:

```
Enter no of disks:3
move disk 1 from S to D
move disk 2 from S to T
move disk 1 from D to T
move disk 3 from S to D
move disk 1 from T to S
move disk 2 from T to D
move disk 1 from S to D

...
... Program finished with exit code 0
Press ENTER to exit console.[]
```

b)

```
#include<stdio.h>

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

int main()
{
    int n1, n2, result;
    printf("Enter two numbers: ");
    scanf("%d %d",&n1,&n2);
    result = gcd(n1,n2);
    printf("GCD of %d and %d = %d",n1,n2,result);
    return 0;
}
```

Result:

```
Enter two numbers: 36 48
GCD of 36 and 48 = 12

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
int n,a[10000];
int bin_srch(int a[],int low,int high,int key) {
    int mid;
    if(low>high)
        return -1;
    mid=(low+high)/2;
    if(key==a[mid])
        return mid;
    if(key<a[mid])
        return bin_srch(a,low,mid-1,key);
    else
        return bin_srch(a,mid+1,high,key);
}
```

```
int lin_srch(int a[],int i,int high,int key) {
    if(i>high)
```

```
    return -1;  
  
    if(key==a[i])  
        return i;  
  
    else  
        return lin_srch(a,i+1,high,key);  
}
```

```
int main()  
{  
    int ch,key,search_status,temp;  
    clock_t end,start;  
    unsigned long int i, j;  
    while(1)  
    {  
        printf("\n1: Binary search\t 2: Linear search\t 3: Exit\n");  
        printf("\nEnter your choice:\t");  
        scanf("%d",&ch);  
        switch(ch)  
        {  
            case 1:  
                n=1000;  
                while(n<=5000)
```

```
{  
for(i=0;i<n;i++)  
{  
//a[i]=random(1000);  
a[i]=i; //Inserting numbers in Ascending order  
}  
key=a[n-1]; //Last element of the array  
start=clock();  
//bub_sort(a,n); //Sorting numbers in Ascending order using  
Bubble sort  
search_status=bin_srch(a,0,n-1,key);  
if(search_status== -1)  
printf("\nKey Not Found");  
else  
printf("\n Key found at position %d",search_status);  
//Dummy loop to create delay  
for(j=0;j<500000;j++){ temp=38/600;}  
end=clock();  
printf("\nTime for n=%d is %f Secs",n,(((double)(end-  
start))/CLOCKS_PER_SEC));  
n=n+1000;  
}  
break;
```

```
case      2:  
n=1000;  
while(n<=5000)  
{  
    for(i=0;i<n;i++)  
    {  
        //a[i]=random(10000);  
        a[i]=i;  
    }  
    key=a[n-1]; //Last element of the array  
    start=clock();  
    search_status=lin_srch(a,0,n-1,key);  
    if(search_status== -1)  
        printf("\nKey Not Found");  
    else  
        printf("\n Key found at position %d",search_status);  
    //Dummy loop to create delay  
    for(j=0;j<500000;j++){ temp=38/600;}  
    end=clock();  
    printf("\nTime for n=%d is %f Secs",n,(((double)(end-  
start))/CLOCKS_PER_SEC));  
    n=n+1000;
```

```
}

break;

default:

    exit(0);

}

getchar();

}

}
```

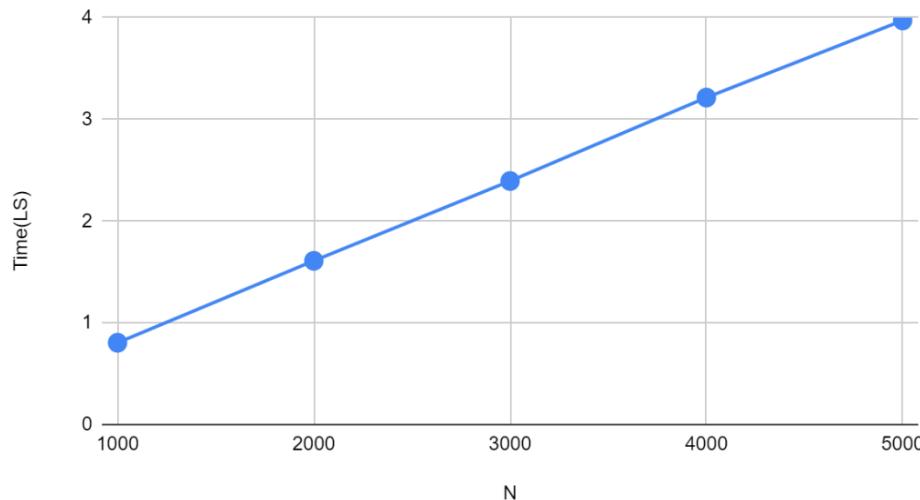
Result:

```
enter the choice: 1.linear search 2.binary search1

search element is found
time for n=1000 is 0.804261 secs
search element is found
time for n=2000 is 1.609531 secs
search element is found
time for n=3000 is 2.393621 secs
search element is found
time for n=4000 is 3.213841 secs
search element is found
time for n=5000 is 3.972152 secs

...Program finished with exit code 0
Press ENTER to exit console.
```

Time(LS) vs. N

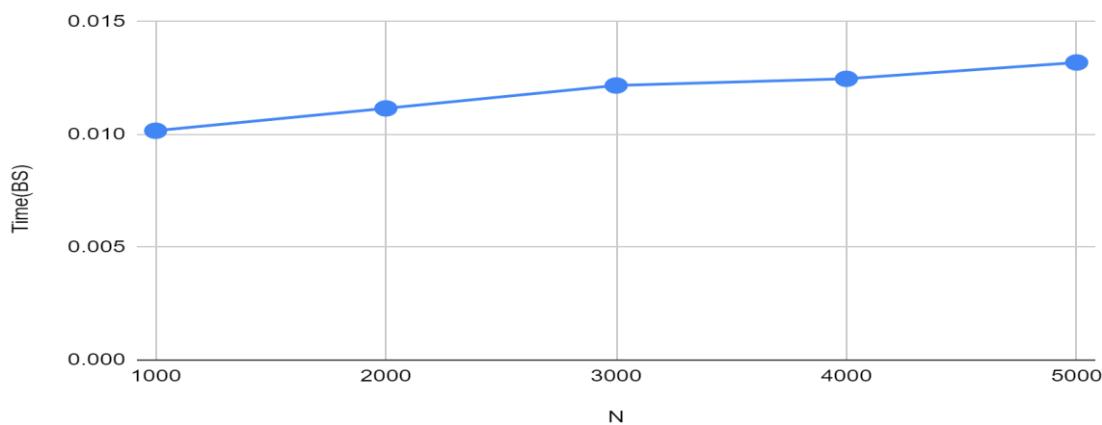


```
enter the choice: 1.linear search 2.binary search2

search element is found
time for n=1000 is 0.010167 secs
search element is found
time for n=2000 is 0.011159 secs
search element is found
time for n=3000 is 0.012174 secs
search element is found
time for n=4000 is 0.012470 secs
search element is found
time for n=5000 is 0.013195 secs

...Program finished with exit code 0
Press ENTER to exit console.
```

Time(BS) vs. N



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

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

void delay(){
    long n;
    for(n=0;n<10;n++){
        int a = 10/10;
    }
}

void selectionsort(int arr[],int length){
    int i,j;
    for(i=0;i<length-1;i++){
        int min=i;
        for(j=i+1;j<length;j++){
            if(arr[j]>arr[min]){
                min=j;
                delay();
            }
        }
    }
    {
        int temp=arr[min];
```

```
        arr[min]=arr[i];
        arr[i]=temp;
    }
}
}

int main()
{
    int arr[15000],n=1000,i;
    double start,end;

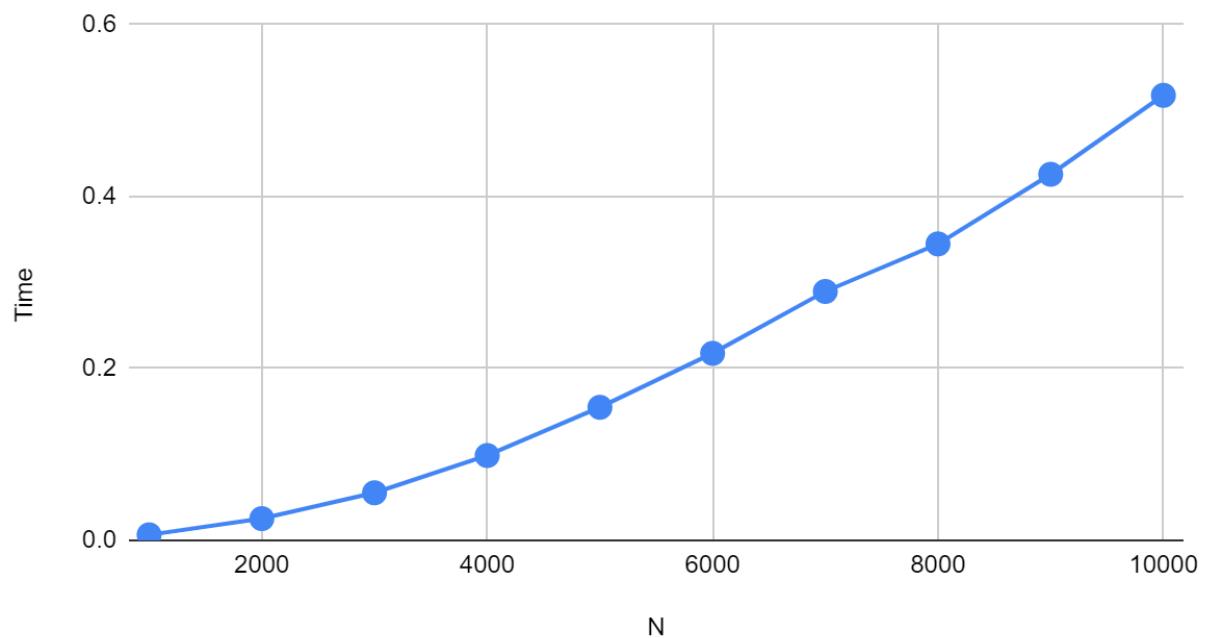
    while(n<=10000){
        for(i=0;i<n;i++){
            arr[i]=i;
        }
        start = clock();
        selectionsort(arr,n);
        end=clock();
        printf("n=%d time= %f \n",n,(end-start)/CLOCKS_PER_SEC);
        n=n+1000;
    }
}
```

Result:

```
n=1000  time= 0.006374
n=2000  time= 0.025264
n=3000  time= 0.055312
n=4000  time= 0.098661
n=5000  time= 0.154757
n=6000  time= 0.217534
n=7000  time= 0.289528
n=8000  time= 0.344848
n=9000  time= 0.425852
n=10000  time= 0.517693
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

Time vs. N



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)

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

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

```
int a[10][10],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[10],f=0,r=-1,vis[10],i,j;
    for(j=1;j<=n;j++)
        vis[j]=0;
    vis[src]=1;
    r=r+1;
    q[r]=src;
    while(f<=r) {
        i=q[f];
        f=f+1;
        for(j=1;j<=n;j++)
        {
            if(a[i][j]==1&&vis[j]!=1) {
                vis[j]=1;
                r=r+1;
                q[r]=j;
            }
        }
    }
}
```

```
}
```

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

```
if(vis[j]!=1)
```

```
printf("\nnode %d is not reachable\n",j);
```

```
else
```

```
{
```

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

```
}
```

```
}
```

```
}
```

b)

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

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

```
int a[10][10],n,vis[10];
```

```
int dfs(int);
```



```
void main()
```

```
{
```

```
int i,j,src,ans;
```

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

```
{
```

```
vis[j]=0;
```

```
}

printf("\nenter the no of nodes:\t");
scanf("%d",&n);

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

printf("\nenter the source node:\t");
scanf("%d",&src);

ans=dfs(src);

if(ans==1)
{
    printf("\ngraph is connected\n");
}
else
{
    printf("\ngragh is not connected\n");
}
getch();
```

```
}

int dfs(int src)

{
    int j;
    vis[src]=1;
    for(j=1;j<=n;j++)
    {
        if(a[src][j]==1&&vis[j]!=1)
        {
            dfs(j);
        }
    }
    for(j=1;j<=n;j++)
    {
        if(vis[j]!=1)
        {
            return 0;
        }
    }
    return 1;
}
```

Result:

```
enter the no of nodes: 4

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

enter the source node: 1

node 1 is reachable
node 2 is reachable
node 3 is reachable
node 4 is reachable

...Program finished with exit code 0
Press ENTER to exit console.[]
```

```
enter the no of nodes: 4

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

enter the source node: 1

graph is connected

...Program finished with exit code 0
Press ENTER to exit console.
```

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

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

void delay(){
    long n;
    for(n=0;n<1000;n++){
        int a = 10/10;
    }
}

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

```
    }

}

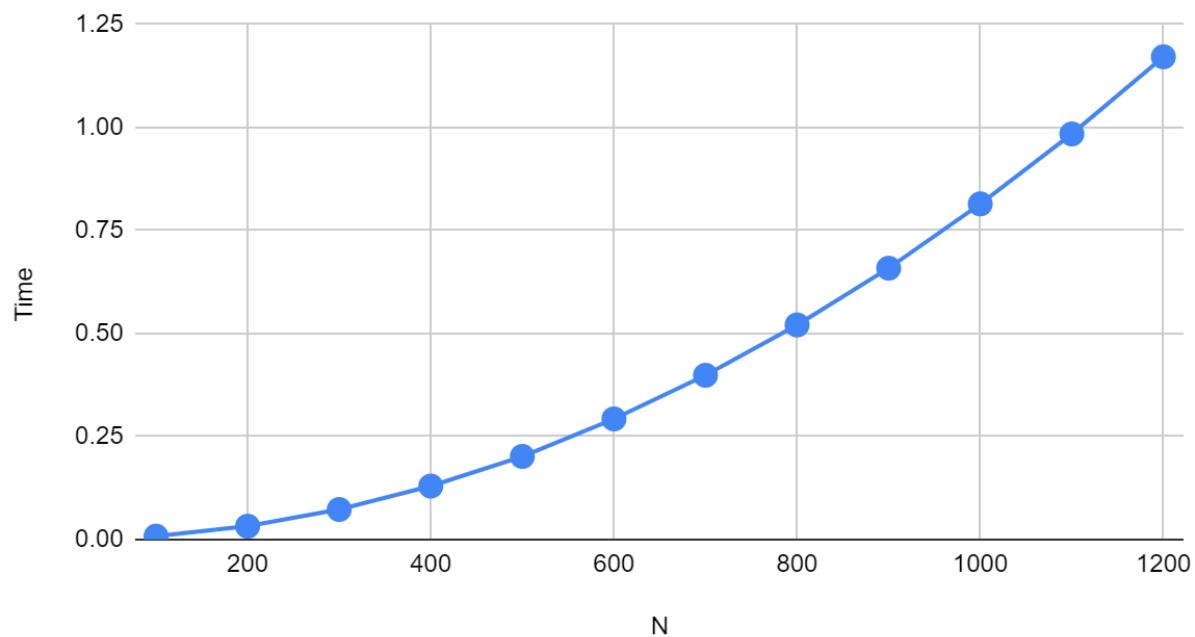
int main()
{
    int arr[1500],n=100,i;
    double start,end;
    while(n<=1200){
        for(i=0;i<n;i++){
            arr[i]=i;
        }
        start = clock();
        insertionSort(arr, n);
        end=clock();
        printf("n=%d time= %f \n",n,(end-start)/CLOCKS_PER_SEC);
        n=n+100;
    }
    return 0;
}
```

Result:

```
n=100  time= 0.008087
n=200  time= 0.032288
n=300  time= 0.072864
n=400  time= 0.129651
n=500  time= 0.201637
n=600  time= 0.292635
n=700  time= 0.398545
n=800  time= 0.520654
n=900  time= 0.658422
n=1000  time= 0.814551
n=1100  time= 0.984286
n=1200  time= 1.171455
```

```
...Program finished with exit code 0
Press ENTER to exit console. □
```

Time vs. N



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

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

```
s[++top]=i;  
}  
}  
k=0;  
while(top!=-1)  
{  
    u=s[top--];  
    t[k++]=u;  
    for(v=0;v<n;v++)  
    {  
        if(a[u][v]==1)  
        {  
            indeg[v]=indeg[v]-1;  
            if(indeg[v]==0)  
                s[++top]=v;  
        }  
    }  
    for(i=0;i<n;i++)  
    {  
        printf("%d\n", t[i]);  
    }
```

```
}

void main()

{

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

    printf("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:

```
Enter number of nodes
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
1
4
0
2
3
5

...Program finished with exit code 0
Press ENTER to exit console. █
```

7. Implement Johnson Trotter algorithm to generate permutations.

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

int flag = 0;

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

int search(int arr[],int num,int mobile)
{
    int g;
    for(g=0;g<num;g++)
    {
        if(arr[g] == mobile)
        {
            return g+1;
        }
    }
}
```

```
        else
        {
            flag++;
        }
    }

    return -1;
}

int find_Moblie(int arr[],int d[],int num)
{
    int mobile = 0;
    int mobile_p = 0;
    int i;
    for(i=0;i<num;i++)
    {
        if((d[arr[i]-1] == 0) && i != 0)
        {
            if(arr[i]>arr[i-1] && arr[i]>mobile_p)
            {
                mobile = arr[i];
                mobile_p = mobile;
            }
        }
    }
}
```

```
        else
        {
            flag++;
        }
    }

else if((d[arr[i]-1] == 1) & i != num-1)
{
    if(arr[i]>arr[i+1] && arr[i]>mobile_p)
    {
        mobile = arr[i];
        mobile_p = mobile;
    }
    else
    {
        flag++;
    }
}
else
{
    flag++;
}
}
```

```
if((mobile_p == 0) && (mobile == 0))

    return 0;

else

    return mobile;

}

void permutations(int arr[],int d[],int num)

{

    int i;

    int mobile = find_Moblie(arr,d,num);

    int pos = search(arr,num,mobile);

    if(d[arr[pos-1]-1]==0)

        swap(&arr[pos-1],&arr[pos-2]);

    else

        swap(&arr[pos-1],&arr[pos]);

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

    {

        if(arr[i] > mobile)

            if(d[arr[i]-1]==0)

                d[arr[i]-1] = 1;

            else
```

```
d[arr[i]-1] = 0;  
}  
}  
for(i=0;i<num;i++)  
{  
    printf(" %d ",arr[i]);  
}  
}
```

```
int factorial(int k)  
{  
    int f = 1;  
    int i = 0;  
    for(i=1;i<k+1;i++)  
    {  
        f = f*i;  
    }  
    return f;  
}  
int main()  
{  
    int num = 0;
```

```
int i;
int j;
int z = 0;
printf("Johnson trotter algorithm to find all permutations of given
numbers \n");
printf("Enter the number\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:

```
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 finished with exit code 0
Press ENTER to exit console.
```

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

```
#include<stdio.h>
#include<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[3000],n,i;
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter array elements:");
for(i=0;i<n;i++)
a[i] = rand()%1000;
start = clock();
mergesort(a,0,n-1);
end = clock();
printf("\nSorted array is :");
for(i=0;i<n;i++)
printf("%d ",a[i]);
```

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

return 0;

}

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[3000];

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];

}
```

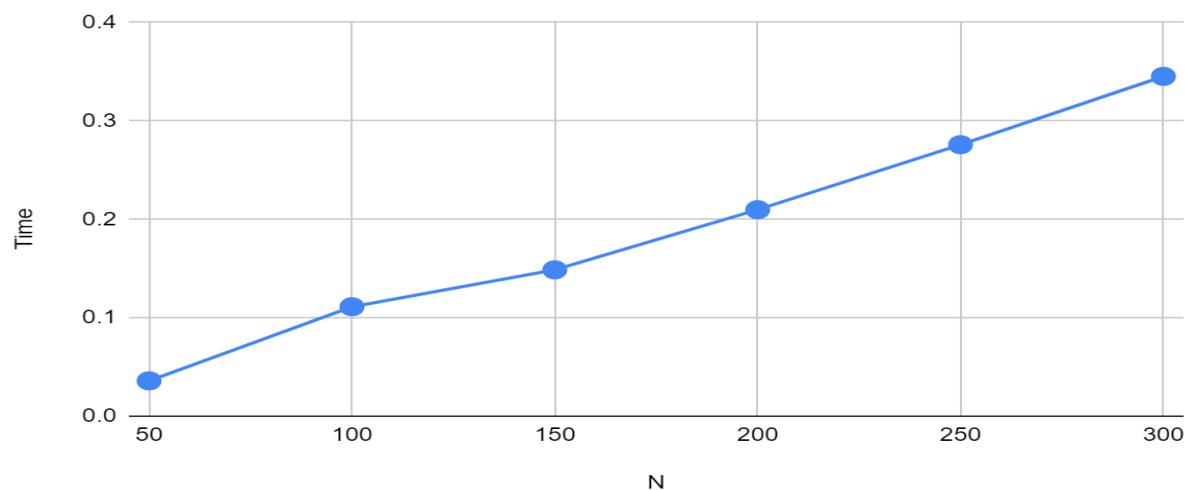
Output:

```
Enter no of elements:50
Enter array elements:
Sorted array is :11 22 27 42 58 59 67 69 123 135 167 172 198 211 229 315 324 335 362 368 370 373 383 386 393 421 421 426 429 456 492 530
537 540 567 649 690 736 763 777 782 784 793 802 862 886 915 919 926 929
Seconds taken 0.035865

...Program finished with exit code 0
Press ENTER to exit console.[]
```

N	Time
50	0.035865
100	0.111199
150	0.148658
200	0.209777
250	0.275837
300	0.345228

Time vs. N



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

```
#include<stdio.h>
#include<time.h>
#include<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++)
    {
        scanf("%d",&number[i]);
    }
    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);

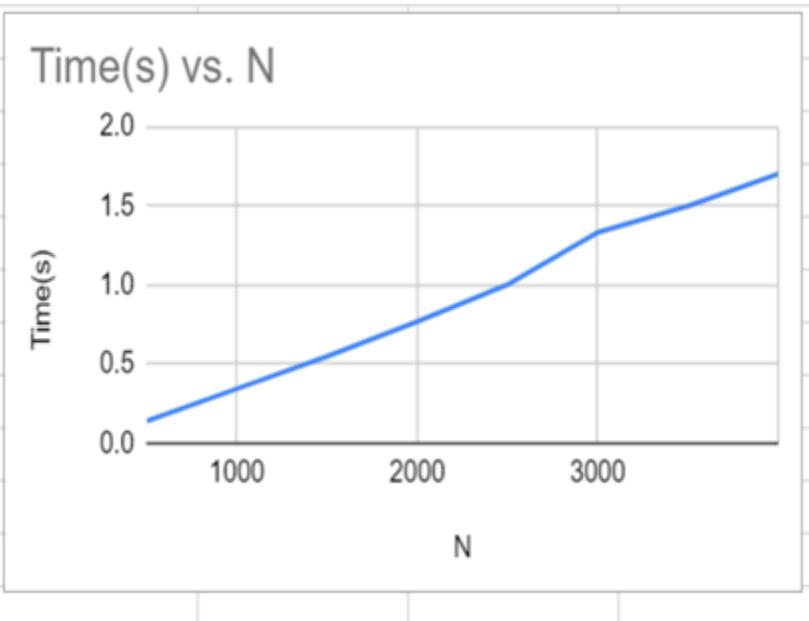
    return 0;
}
```

OUTPUT:

```
C:\Users\mknv7\OneDrive\Documents\C-Workspace\Quick_sort\Debug\Quick_sort.exe
Sorted elements 500: Seconds taken 0.141000
Sorted elements 1000: Seconds taken 0.345000
Sorted elements 1500: Seconds taken 0.549000
Sorted elements 2000: Seconds taken 0.770000
Sorted elements 2500: Seconds taken 1.002000
Sorted elements 3000: Seconds taken 1.334000
Sorted elements 3500: Seconds taken 1.499000
Sorted elements 4000: Seconds taken 1.702000
```

GRAPH:

N	Time(s)
500	0.141
1000	0.345
1500	0.549
2000	0.77
2500	1.002
3000	1.334
3500	1.499
4000	1.702



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

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
void swap(int *,int *);
void heapify(int [],int,int);
void heapSort(int[], int);
int main()
{
    int a[15000], n, i, j, ch;
    clock_t start, end;
    while (1)
    {
        printf("\n1:FOR MANUAL ENTRY");
        printf("\n2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM
RANGE 500 TO 5500");
        printf("\n3:EXIT");
        printf("\nEnter YOUR CHOICE:");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("\nEnter NUMBER OF ARRAY ELEMENTS: ");
                scanf("%d", &n);
                printf("Enter ARRAY ELEMENTS: ");
                for (i = 0; i < n; i++)
                {
                    scanf("%d", &a[i]);
                }
                start = clock();
            case 2:
                end = clock();
                printf("Time taken: %f", (double)(end - start) / CLOCKS_PER_SEC);
                break;
            case 3:
                exit(0);
        }
    }
}
```

```

heapSort(a, n);
end = clock();
printf("\nSORTED ARRAY IS: ");
for (i = n-1; i >= 0; i--)
    printf("%d\t", a[i]);
printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n,
(((double)(end - start)) / CLOCKS_PER_SEC));
break;
case 2:
n = 500;
while (n <= 5500)
{
    for (i = 0; i < n; i++)
    {
        //a[i]=rand()%n;
        a[i] = n - i;
    }
    start = clock();
    heapSort(a, n);
    end = clock();
    printf("\n TIME TAKEN TO SORT %d NUMBERS IS %f SECS", n,
(((double)(end - start)) / CLOCKS_PER_SEC));
    n = n + 1000;
}
break;
case 3:
exit(0);
}
getchar();
}
}

void swap(int *a, int *b)

```

```
{  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}  
void heapify(int arr[], int n, int i)  
{  
    int temp;  
    for (int j = 0; j < 50000; j++)  
        temp = 38 / 600;  
    int largest = i;  
    int left = 2 * i + 1;  
    int right = 2 * i + 2;  
    if (left < n && arr[left] > arr[largest])  
        largest = left;  
    if (right < n && arr[right] > arr[largest])  
        largest = right;  
    if (largest != i)  
    {  
        swap(&arr[i], &arr[largest]);  
        heapify(arr, n, largest);  
    }  
}  
void heapSort(int arr[], int n)  
{  
    for (int i = n / 2 - 1; i >= 0; i--)  
        heapify(arr, n, i);  
    for (int i = n - 1; i >= 0; i--)  
    {  
        swap(&arr[0], &arr[i]);  
        heapify(arr, i, 0);  
    }  
}
```

Output:

```
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 5500
3:EXIT
ENTER YOUR CHOICE:2

TIME TAKEN TO SORT 500 NUMBERS IS 0.270000 SECS
TIME TAKEN TO SORT 1500 NUMBERS IS 0.978000 SECS
TIME TAKEN TO SORT 2500 NUMBERS IS 1.751000 SECS
TIME TAKEN TO SORT 3500 NUMBERS IS 2.580000 SECS
TIME TAKEN TO SORT 4500 NUMBERS IS 3.436000 SECS
TIME TAKEN TO SORT 5500 NUMBERS IS 4.394000 SECS
1:FOR MANUAL ENTRY
2:DISPLAY TIME TAKEN TO SORT ELEMENTS FROM RANGE 500 TO 5500
3:EXIT
ENTER YOUR CHOICE:1

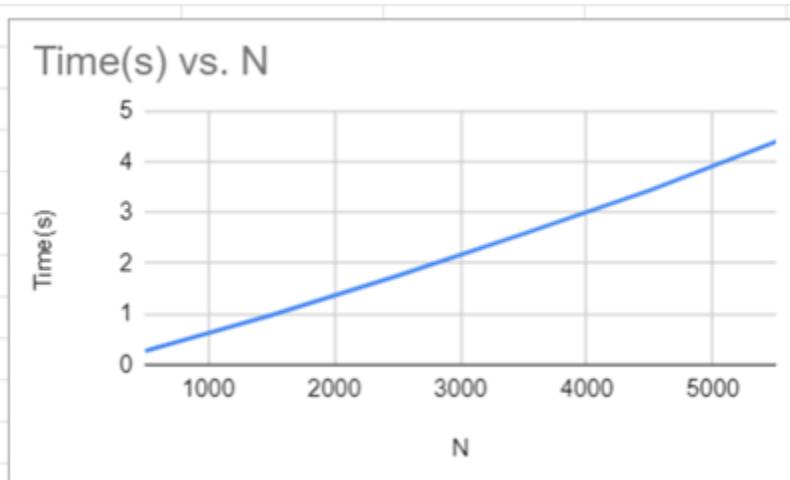
ENTER NUMBER OF ARRAY ELEMENTS: 4

ENTER ARRAY ELEMENTS: 32 8 90 19

SORTED ARRAY IS: 90      32      19      8
TIME TAKEN TO SORT 4 NUMBERS IS 0.000000 SECS
```

GRAPH:

N	Time(s)
500	0.27
1500	0.978
2500	1.751
3500	2.58
4500	3.436
5500	4.394



11. Implement warshall's algorithm using dynamic programming.

```
#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");
    }
}
```

OUTPUT:

```
[Select C:\Users\mknv7\OneDrive\Documents\C-Workspace\Warshalls-algo\Debug]
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

===== Program exited with exit code: 0 =====
Time elapsed: 000:32.141 (MM:SS.MS)
Press any key to continue...
```

12. Implement 0/1 Knapsack algorithm using dynamic programming.

```
#include<stdio.h>
#include<conio.h>
void knapsack();
int max(int,int);
int i,j,n,m,p[10],w[10],v[10][10];
void main()
{
    clrscr();
    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);
    knapsack();
    getch();
}
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;
}

```

OUTPUT:

```
enter the no. of items: 4
enter the weight of the each item:
2 1 3 2
enter the profit of each item:
12 10 20 15
enter the knapsack's capacity: 5
the output is:
0      0      0      0      0      0
0      0      12     12     12     12
0      10     12     22     22     22
0      10     12     22     30     32
0      10     15     25     30     37

the optimal solution is 37
the solution vector is:
1      1      0      1
...Program finished with exit code 255
Press ENTER to exit console.
```

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

```
#include<stdio.h>
int n;
void display(int dist[][n]);
void floyd (int graph[][n])
{
    int dist[n][n], i, j, k;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            dist[i][j] = graph[i][j];

    for (k = 0; k < n; k++)
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];

    display(dist);
}

void display(int dist[][n]) {
    printf ("DISTANCE MATRIX \n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (dist[i][j] == 99)
                printf("99 ");
            else
                printf ("%d ", dist[i][j]);
        }
    }
}
```

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

int main()
{
    printf("ENTER ORDER OF MATRIX \n");
    scanf("%d",&n);
    int graph[n][n];
    printf("ENTER ELEMENTS OF MATRIX and 99 FOR INFINITY\n");
    for(int i = 0;i < n;i++)
        for(int j = 0;j < n; j++)
            scanf("%d",&graph[i][j]);
    floyd(graph);
    return 0;
}
```

OUTPUT:

```
C:\Users\mknv7\OneDrive\Documents\C-Workspace\Floyd's_Algorithm\Debug\Floyd's_Algorithm.exe
ENTER ORDER OF MATRIX
4
ENTER ELEMENTS OF MATRIX and 99 FOR INFINITY
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
DISTANCE MATRIX
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

==== Program exited with exit code: 0 ====
Time elapsed: 000:47.968 (MM:SS.MS)
Press any key to continue...
```

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

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void prims();
int c[10][10],n;
void main()
{
    int i,j;
    printf("\nEnter the no. of vertices:\t");
    scanf("%d",&n);
    printf("\nEnter the cost matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    prims();
}

void prims()
{
    int i,j,u,v,min;
    int ne=0,mincost=0;
    int elec[10];
    for(i=1;i<=n;i++)
    {
        elec[i]=0;
    }
```

```
}

elec[1]=1;
while(ne!=n-1)
{
    min=9999;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(elec[i]==1)
            {
                if(c[i][j]<min)
                {
                    min=c[i][j];
                    u=i;
                    v=j;
                }
            }
        }
    }

if(elec[v]!=1)
{
    printf("\n%d---->%d=%d\n",u,v,min);
    elec[v]=1;
    ne=ne+1;
    mincost=mincost+min;
}
c[u][v]=c[v][u]=9999;
}
printf("\nmincost=%d",mincost);
}
```

OUTPUT:

```
C:\Users\mknv7\OneDrive\Documents\C-Workspace\prim's-algo\Debug\prim's-algo.exe
PRIMS ALGORITHM

Enter the no. of vertices: 6

Enter the cost matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 2
5 4 4 5 2 0

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

mincost=15
==== Program exited with exit code: 12 ====
Time elapsed: 000:03.406 (MM:SS.MS)
Press any key to continue...
```

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

```
#include<stdio.h>
void kruskals();
int c[10][10],n;
void main()
{
    int i,j;
    printf("\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;
        }
    }
    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);
}
```

OUTPUT:

```
C:\Users\mknv7\OneDrive\Documents\C-Workspace\Kruskals-algo\Debug\Kruskals-algo.exe
KRUSKALS ALGORITHM

Enter the no. of vertices: 6

Enter the cost matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 2
5 4 4 5 2 0

2---->3=1

5---->6=2

1---->2=3

2---->6=4

4---->6=5

mincost=15
==== Program exited with exit code: 11 ====
Time elapsed: 000:08.953 (MM:SS.MS)
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.

```
#include <stdio.h>
void dijkstras();
int c[10][10], n, src;
void main()
{
    int i, j;
    printf("\nEnter the no of vertices: ");
    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: ");
    scanf("%d", &src);
    dijkstras();
}
void dijkstras()
{
    int vis[10], dist[10], u, j, count, min;
```

```
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\mknv7\OneDrive\Documents\C-Workspace\djikstras\Debug\djikstras.exe
DIJKSTRAS ALGORITHM

Enter the no of vertices: 6

enter the cost matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 2
5 4 4 5 2 0

Enter the source node: 1

the shortest distance is:

1---->1=0
1---->2=3
1---->3=4
1---->4=10
1---->5=6
1---->6=5
```

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.

```
#include <stdio.h>
#include <conio.h>
int count, w[10], d, x[10];
void subset(int cs, int k, int r)
{
    int i;
    x[k] = 1;
    if (cs + w[k] == d)
    {
        printf("\nSubset solution = %d\n", ++count);
        for (i = 0; i <= k; i++)
        {
            if (x[i] == 1)
                printf("%d ", w[i]);
        }
    }
    else if (cs + w[k] + w[k + 1] <= d)
```

```
subset(cs + w[k], k + 1, r - w[k]);

if ((cs + r - w[k] >= d) && (cs + w[k + 1]) <= d)

{

    x[k] = 0;

    subset(cs, k + 1, r - w[k]);

}

}

void main()

{

    int sum = 0, i, n;

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

    scanf("%d", &n);

    printf("Enter the elements in ascending order\n");

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

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



    printf("Enter the required sum\n");

    scanf("%d", &d);

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

        sum += w[i];

    if (sum < d)
```

```
{  
    printf("No solution exists\n");  
    return;  
}  
  
printf("The solution is\n");  
count = 0;  
subset(0, 0, sum);  
getch();  
}
```

OUTPUT:-

```
Enter the number of elements  
5  
Enter the elements in ascending order  
1  
2  
5  
6  
8  
Enter the required sum  
9  
The solution is  
  
Subset solution = 1  
1 2 6  
Subset solution = 2  
1 8  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

18) Implement “N-Queens Problem” using Backtracking.

```
#include<stdio.h>
#include<conio.h>
void nqueens(int n)
{
    int k,x[20],count=0;
    k=1;
    x[k]=0;
    while(k!=0)
    {
        x[k]++;
        while(place(x,k)!=1 && x[k]<=n)
            x[k]++;
        if(x[k]<=n)
        {
            if(k==n)
            {
                printf("\nSolution is %d\n", ++count);
                printf("Queen\tPosition\n");
                for(k=1;k<=n;k++)
                    printf("%d\t%d\n", k,x[k]);
            }
        }
    }
}
```

```
        }

        else

        {

            k++;

            x[k]=0;

        }

    }

    else

        k--;

}

}

int place(int x[], int k)

{

    int i;

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

    {

        if(i+x[i]==k+x[k] || i-x[i]==k-x[k] || x[i]==x[k])

            return 0;

    }

    return 1;

}

void main()
```

```
{  
    int n;  
  
    printf("Enter the number of Queens\n");  
  
    scanf("%d", &n);  
  
    nqueens(n);  
}
```

OUTPUT:-

```
Enter the number of Queens  
4  
  
Solution is 1  
Queen          Position  
1              2  
2              4  
3              1  
4              3  
  
Solution is 2  
Queen          Position  
1              3  
2              1  
3              4  
4              2  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```