

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

PRAJWAL PATIL (1BM20CS108)

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 **PRAJWAL PATIL (1BM20CS108)**, 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.

Name of the Lab-In charge:

REKHA G 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:

```
✕ Output

Enter no of disks:>>>3
(1,3)
(1,2)
(3,2)
(1,3)
(2,1)
(2,3)
(1,3)
time = 0.004288

Process Finished.
```

b)

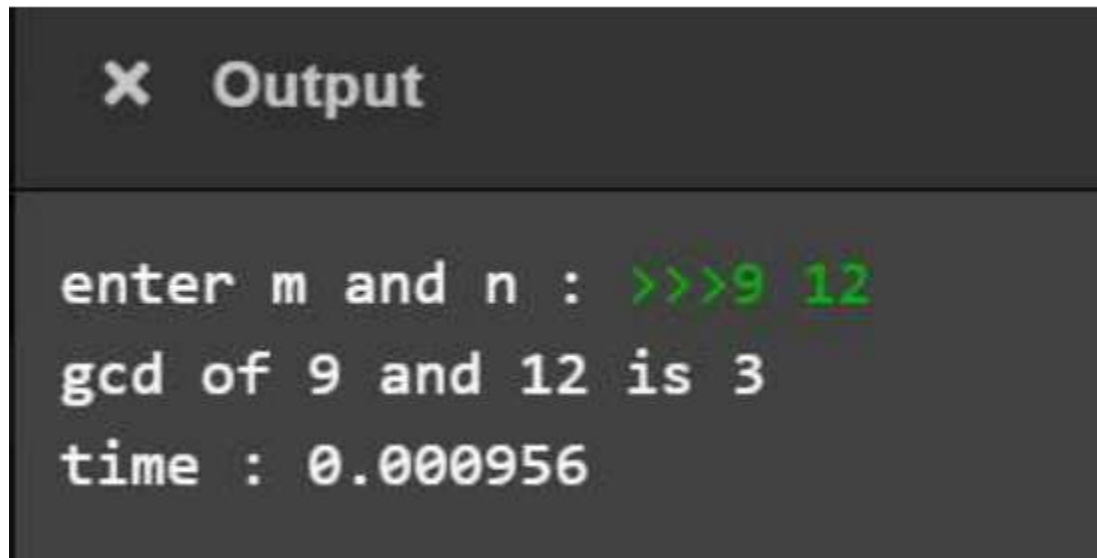
```
#include<stdio.h>

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:



```
X Output  
  
enter m and n : >>>9 12  
gcd of 9 and 12 is 3  
time : 0.000956
```

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.

Program:

```
#include<stdio.h>

#include<time.h>

int binSearch(int a[], int ele, int start,
int end) {
}

int mid = (start+end)/2;
int pos;
if(a[mid] == ele) return mid;
if(a[mid]>ele)
pos = binSearch( a, ele, start, mid-1);
else
pos = binSearch( a, ele, mid+1, end);
double d;
for(d=0;d<9990099;d++) {
d++;
d--;
}
return pos;
```



```

}
void main() {
    int arr[10], n, item, pos;
    clock_t start, end;
    printf("enter number of elements :
");
    scanf("%d", &n);
    printf("enter elements : ");
    for(int i=0; i<n ; i++)
        scanf("%d", &arr[i]);
    printf("enter the element to search :
");
    scanf("%d", &item);
    start = clock();
    pos=binSearch(arr, item, 0, n);
    end= clock();
    printf("element found at %d\n",
pos);
    printf("time: %f", (double)(endstart)/CLOCKS_PER_SEC);
}

```

Result:

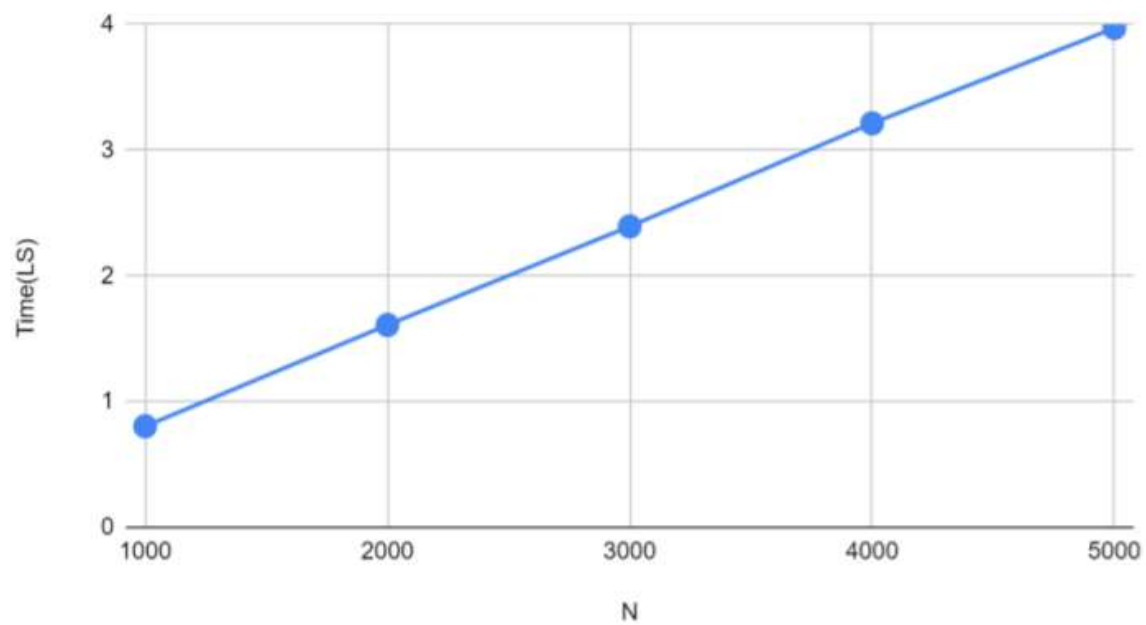
X Output

```
enter number of elements : >>>6  
enter elements : >>>2 4 6 9 10 12  
enter the element to search : >>>6  
element found at 2  
time: 0.058191
```

Process Finished.

```
>>>
```

Time(LS) vs. N



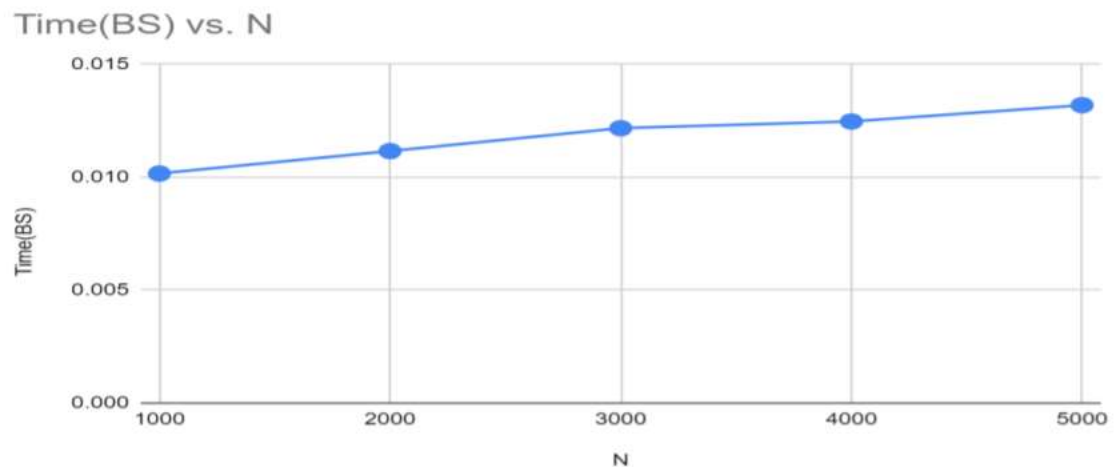
linear search

```
int searchElement(int arr[], int size, int x)
{
    size--;
    if (size < 0) {
        return -1;
    }
    if (arr[size] == x) {
        return size;
    }
    return searchElement(arr, size, x);
}

void main() {
    int arr[10], n, item, pos;
    clock_t start, end;
    printf("enter number of elements : ");
    scanf("%d", &n);
    printf("enter elements : ");
    for(int i=0; i<n ; i++)
        scanf("%d", &arr[i]);
    printf("enter the element to search : ");
    scanf("%d", &item);
```

```
start = clock();  
pos=searchElement(arr, item, 0, n);  
end= clock();  
printf("element found at %d\n", pos);  
printf("time: %f", (double)(endstart)/CLOCKS_PER_SEC);  
}
```

```
X Output  
  
enter number of elements : >>>7  
enter elements : >>>4 6 3 8 2 9 1  
enter the element to search : >>>3  
element found at 2  
time: 0.000000  
  
Process Finished.  
>>>
```



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.

PROGRAM:

```
#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;
    }
}

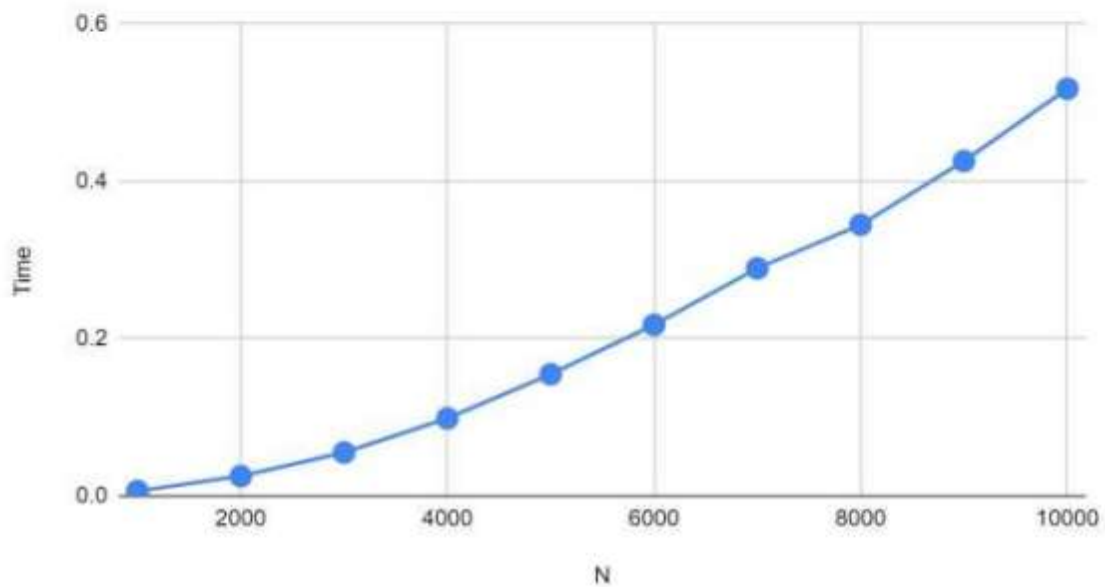
```

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("\ngraph is not connected\n");  
}
```

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

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:

✕ Output

```
enter the size of array:>>>6
```

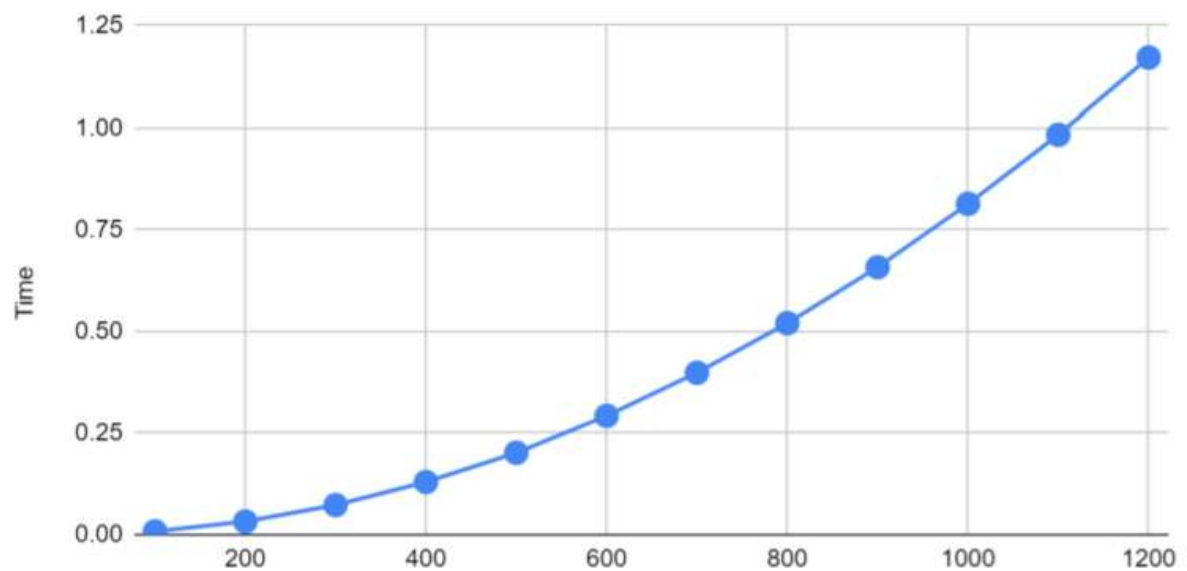
```
before SORTING : 479, 665, 154, 269, 501, 998,
```

```
after SORTING : 8, 154, 269, 479, 501, 665,  
TIME TAKEN : 0.061206
```

```
Process Finished.
```

```
>>>
```

Time vs. N



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

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;
    printf("Enter the no of vertices:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++)
    {
        printf("Enter row %d\n",i+1);
        for(j=0; j<n; j++)
            scanf("%d",&a[i][j]);
    }
    for(i=0; i<n; i++)
    {
        indeg[i]=0;
        flag[i]=0;
    }
    for(i=0; i<n; i++)
```

```

for(j=0; j<n; j++)
indeg[i]=indeg[i]+a[j][i];
printf("\nThe topological order is:");
while(count<n)
{
for(k=0; k<n; k++)
{
if((indeg[k]==0) && (flag[k]==0))
{
printf("%d ",(k+1));
flag [k]=1;
}
for(i=0; i<n; i++)
{
if(a[i][k]==1)
indeg[k]--;
}
}
count++;
}
}

```

RESULT:

Enter the no of vertices:

4

Enter the adjacency matrix:

Enter row 1

0 1 1 0

Enter row 2

0 0 0 1

Enter row 3

0 0 0 1

Enter row 4

0 0 0 0

The topological order is:1 2 3 4

...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:

```
× Output

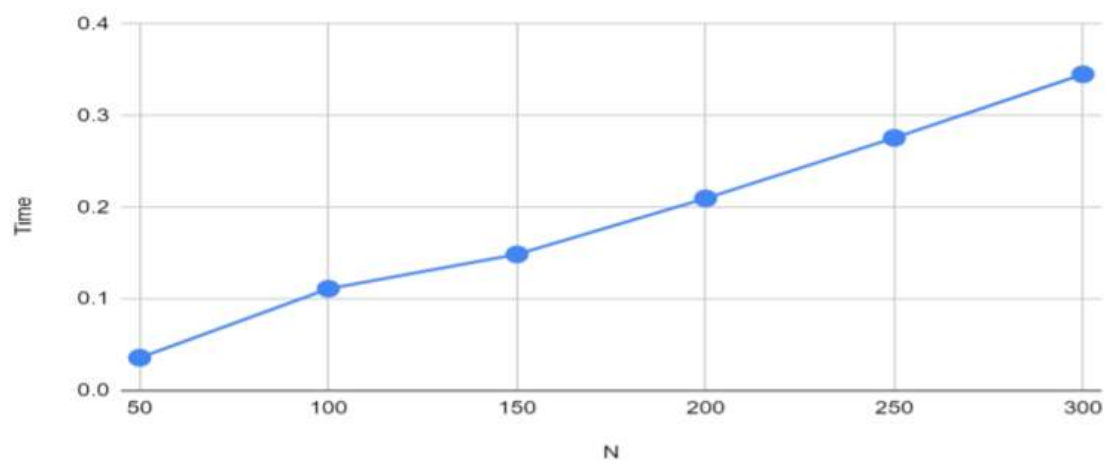
Enter no of elements:>>>9
Enter array elements:>>>5 3 2 6 8 2 0 8 1

Sorted array is :0 1 2 2 3 5 6 8 8 time taken : 0.844277

Process Finished.
```

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.

Program:

```
#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<100000;x++);
while(number[i]<=number[pivot]&& i<last)
i++;
while(number[j]>number[pivot])
j--;
if(i<j){
temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last);
}
```

```

}
int main(){
    clock_t start,end;
    int i, count, number[5000];
    printf("No. of elements: ");
    scanf("%d",&count);
    printf("Enter %d elements: ", count);
    for(i=0;i<count;i++)
        number[i] = rand()%1000;
    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:

✕ Output

enter the size of array:>>>10

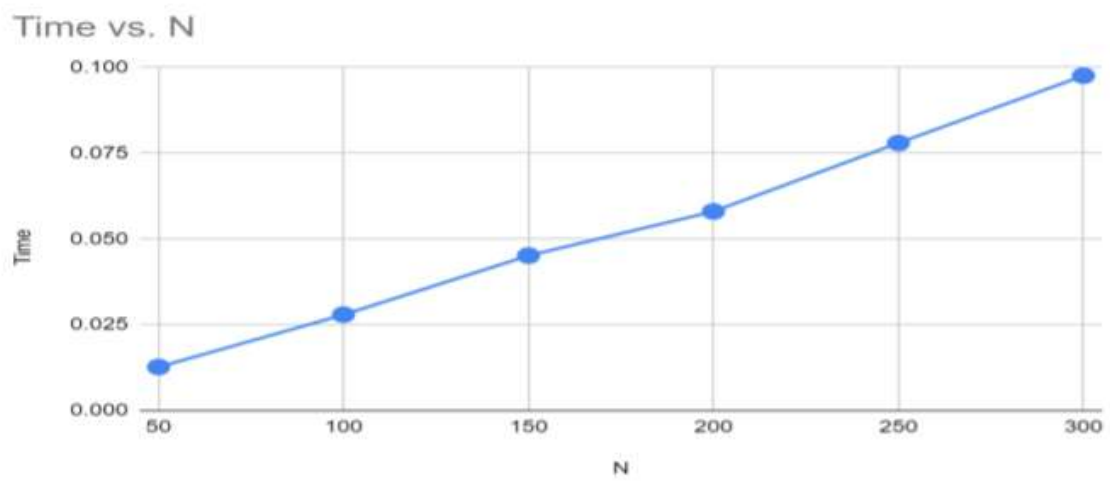
numbers : 83, 86, 77, 15, 93, 35, 86, 92, 49, 21,

Order of Sorted elements: 15 21 35 49 77 83 86 86 92 93

Time taken:0.000000

Process Finished.

N	Time
50	0.012683
100	0.027947
150	0.045167
200	0.058032
250	0.077984
300	0.097606



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

Program :

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>
void swap(int *,int *);
void heapify(int [],int,int);
void heapSort(int[], int);

int main()
{
    int a[15000],n,i,j,ch,temp;
    clock_t start,end;
    while(1)
    {
        printf("\n 1: For manual entry of N values and array elements:");
        printf("\n 2: To display time taken for sorting number of elements
N in the range 500 to 14500:");
        printf("\n 3: To exit");
        printf("\n Enter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: printf("\n Enter the number of elements:");
                    scanf("%d",&n);
                    printf("\n Enter array elements:");
                    for(i=0;i<n;i++)
                    {
```

```

        scanf("%d",&a[i]);
    }
    start=clock();
    heapSort(a, n);
    end=clock();
    printf("\n Sorted array is:");
    for(i=n-1;i>=0;i--){
        printf("%d\t",a[i]);
    }
    printf("\n Time taken to sort %d numbers is %f
secs",n,((double)(end-start)/CLOCKS_PER_SEC));
    break;
    case 2:
        n=500;
        while(n<=14500){
            for(i=0;i<n;i++){
                a[i]=n-i;

            }
            start=clock();
            heapSort(a, n);
            for(j=0;j<50000000;j++){
                temp=38/600;

            }
            end=clock();
            printf("\n Time taken to sort %d numbers is %f
secs",n,((double)(end-start)/CLOCKS_PER_SEC));
            n=n+1000;

        }

```

```

        break;
    case 3: exit(0);
    }

}

}

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

void heapify(int arr[], int n, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i >= 0; i--)

```

```

    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

```

Output :

```

1: For manual entry of N values and array elements:
2: To display time taken for sorting number of elements N in the range 500 to 14500
   :
3: To exit
Enter your choice:1
Enter the number of elements:5
Enter array elements:20 31 10 46 78
Sorted array is:78 46 31 20 10
Time taken to sort 5 numbers is 0.000003 secs
1: For manual entry of N values and array elements:
2: To display time taken for sorting number of elements N in the range 500 to 14500
   :|
3: To exit
Enter your choice:2
Time taken to sort 500 numbers is 0.105851 secs
Time taken to sort 1500 numbers is 0.103846 secs
Time taken to sort 2500 numbers is 0.103909 secs
Time taken to sort 3500 numbers is 0.105498 secs
Time taken to sort 4500 numbers is 0.104747 secs
Time taken to sort 5500 numbers is 0.106133 secs
Time taken to sort 6500 numbers is 0.105619 secs
Time taken to sort 7500 numbers is 0.105099 secs
Time taken to sort 8500 numbers is 0.105469 secs
Time taken to sort 9500 numbers is 0.105425 secs
Time taken to sort 10500 numbers is 0.106843 secs

```

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

Program:

```
#include<stdio.h>

int w[10], p[9], n;

int maxim(int x, int y) {
    x>y? x : y;
}

int knap(int i, int m) {
    if(i == n) return w[i] > m ? 0 : p[i];

    if(w[i] > m) return knap(i+1, m);
    return
    maxim( knap(i+1, m), knap(i+1, m-w[i]) + p[i]);
}

void main() {
    int i, m;
    printf("number of items");
    scanf("%d", &n);

    printf("enter weight and value for each item:\n");
    for(i=1; i<=n ; i++)
        scanf("%d %d", &w[i], &p[i] );

    printf("enter capacity of knapsack ");
    scanf("%d", &m);
```

```
int maxprofit = knap(1, m);  
printf(" max profit : %d", maxprofit );  
}
```

Output:

```
× Output  
  
number of items>>>4  
enter weight and value for each item:  
>>>7 42  
>>>3 12  
>>>4 40  
>>>5 25  
enter capacity of knapsack >>>11  
max profit : 82  
  
Process Finished.  
>>>
```

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

Program:

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

```
int n;
```

```
int a[20][20];
```

```
void floyd(int a[20][20])
```

```
{
```

```
    for(int k=0;k<n;k++)
```

```
    {
```

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

```
        {
```

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

```
                a[i][j]=min(a[i][j],(a[i][k]+a[k][j]));
```

```
        }
```

```
    }
```

```
}
```

```
int main()
```

```
{
```

```
    printf("Enter the no. of vertices");
```

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

```
    printf("Enter the weighted matrix ( enter 99999 for infinity)");
```

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

```
    {
```

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

```
        {
```

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



```

    }
}
floyd(a);
printf("All Pair Shortest paths :");
for(int i=0;i<n;i++)
{
    for(int j=0;j<n;j++)
    {
        printf("%d, ", a[i][j]);

    }
    printf("\n");
}
return 0;
}

```

Output:

```

enter the no. of vertices: 4
enter the cost matrix:
11 999 999 44
33 44 999 3
33 55 999 555
44 3 6 9
all pair shortest path matrix is:
11 47 50 44

33 6 9 3

33 55 64 58

36 3 6 6
|

```

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

Program:

```
#include<stdio.h>
#define inf 999

int prim(int cost[10][10], int source, int n ) {
    int parent[10], visited[10], cmp[10], min, sum=0;
    int i, j, u, v;
    for(i=1 ; i<=n ; i++) {
        visited[i] =0;
        parent[i]= source;
        cmp[i] = cost[source][i];
    }
    visited[source]= 1;
    for( i=1 ; i<n ; i++) {    //check
        min= inf ;
        for( j = 1 ; j<=n ; j++) {
            if(!visited[j] && cmp[j] < min ) {    //check j not !=i
                min=cmp[j];
                u=j;
            }
        }
        visited[u]=1;
        sum += cmp[u];
        printf("");
        for( v=1 ; v<=n ; v++)
            if(!visited[v] && cmp[v] > cost[u][v]) {
                cmp[v] = cost[u][v];
                parent[v] = u;
            }
    }
}
```

```

    }
}
return sum;
}

void main () {
    int a [10] [10] , n, i, j, m, source;
    printf ("\n Enter the number of vertices: ");
    scanf ( "%d" , &n) ;
    printf ("\nEnter the cost matrix: 0 -self loop & 999 - no edge \n" );
    for (j=1; j<=n; j++)
        for (i=1; i<=n; i++)
            scanf ( "&d", &a [i] [j] ) ;
    printf ( " \n Enter the source :");
    scanf ( "%d" , &source) ;
    m=prim (a, source, n) ;
    printf ( " \n\n Cost =%d" , m) ;
}

```

Output:

```

enter the no. of vertices: 4
enter the cost matrix:
1 0 0 1
0 2 3 4
0 3 2 4
1 2 0 3
1----->2=0

1----->3=0

1----->4=1

mincost=1|

```

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

Program:

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

```
#define inf 999
```

```
int cost[20][20], parentkru[20], t[20][2];
```

```
int find (int v) {
```

```
    while (parentkru [v] ) {
```

```
        v=parentkru [v] ;
```

```
    }
```

```
    return v;
```

```
}
```

```
void unionkru(int i, int j) {
```

```
    parentkru[j] = i;
```

```
}
```

```
void kruskal(int n) {
```

```
    int min, i, j, k, u, v;
```

```
    int res1, res2, sum = 0;
```

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

```
        min = inf;
```

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

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

```
                if(i == j)
```

```
                    continue;
```

```

        if(cost[i][j] < min){

            u = find(i);
            v = find(j);

            if(u != v){
                res1 = i;
                res2 = j;
                min = cost[i][j];
            }
        }
    }
}

unionkru(res1, find(res2));
t[k][1]=res1;
t[k][2]=res2;
sum += min;
}

printf("cost mst is %d", sum);

}

void main() {
    int i , j,n;

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

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

```

```
printf ( "\\n Enter the cost adjacent matrix 0- for self edge and 999-if  
no edge \\n" ) ;  
for (i=1; i<=n; i++)  
    for (j=1; j<=n; j+ +)  
        scanf ( "*d" , &cost [i] [j]) ;  
kruskal (n) ;  
}
```

Output:

```
/tmp/1d1WsEvHht.o  
enter the no. of vertices: 4  
enter the cost matrix:  
1 0 0 1  
2 0 0 2  
2 3 4 9  
1 3 4 5  
1-----> 2=0  
  
1-----> 3=0  
  
1-----> 4=1  
  
mincost=1
```

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

Program:

```
#include<stdio.h>
#include<conio.h>
#define inf 999
void dijkstra(int cost[20][20], int n, int source, int distance) {
    int i, j, min, sum=0, u , v;
    int parent[20];
    for(i=1;i<=n i++) {
        distance[i] = cost[source][i];
        parent[i] = source;
    }
    visited[source] = 1;
    for(int i=0; i<=n; i++) {
        min = inf;
        for(j = 0; j<=n ;j++){
            if(!visited[j] && distance[j] < min) {
                min = distance[j];
                u=j;
            }
        }
        visited[u]=1;
        sum += distance[u];
        for(j = 1; j<=n ; j++){
            if(!visited[j] && distance[u]+cost[u][v] < distance[j]) {
                distance[j] = distance[u] + cost[u][j];
            }
        }
    }
}
```

```

}
void main ( ) {
    int i , j , source, sum;
    int n, cost [10] [10] , distance [10];
    printf ( " \n Enter how many nodes :") ;
    scanf ( "%d" , &n) ;
    printf ("\n Cost Matrix\n Enter 999 for no edge\n") ;
    for (j=1; j<=n; j+ +)
        for (i=1; i<=n; i+ +)
            scanf ( "%d" , &cost [i] [j]) ;
    printf ( "Enter the source node :") ;
    scanf ("%d", &source) ;
    dijkstra (cost , n, source, distance) ;
    for (i=1 ; i<=n; i++)
        printf ( " \n SHORTEST DISTANCE FROM &d TO &d IS d\n",source, i,
distance [i] ) ;
}

```

OUTPUT:

```

/tmp/RHJFW6kRtE.o
Enter the number of nodes:5
Enter the cost matrix:
2 3 4 999 999
999 4 5 999 999
3 45 999 2 5
999 5 6 7 999
5 6 8 999 999
Enter the source matrix:1
Shortest path:
1->2,cost=4002
1->3,cost=4004
1->4,cost=4995
1->5,cost=4995

```


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.

Program:

```
#include<conio.h>
#include<stdio.h>
#define MAX 10
int s [MAX], x[MAX];

int d;
void sumofsub (int p, int k, int r) {

    int i;
    x[k] = 1;
    if ((p + s[k]) == d)

        for(i=1; i<=k; i++) {
            if (x[i] == 1)
                printf("%d ", s[i]);
            printf("\n");
        }
    else
        if(p+ s[k] + s[k+1] <= d)
            sumofsub(p + s[k], k+1, r-s[k]);
    if ((p + r - s[k] >= d) && (p + s[k+1] <= d)) {
        x [k]=0;
        sumofsub(p, k+1, r-s[k]);
    }
}
```

```
}  
}
```

```
void main () {  
    int i, n, sum=0 ;  
  
    printf ( " \n Enter max. number : " ) ;  
    scanf ( "%d" , &n) ;  
  
    printf ( " \n Enter the set in increasing order : \n" ) ;  
    for (i=1; i<=n; i++)  
        scanf ( "%d" , &s[i]) ;  
  
    printf ( " \n Enter the max. subset value : " );  
    scanf ( "%d", &d) ;  
  
    for (i=1; i<=n; i+ +)  
        sum=sum+ s[i] ;  
    if (sum < d || s[1] >d )  
        printf ( " \n No subset possible" ) ;  
    else  
        sumof sub ( 0 , 1, sum) ;  
}
```

Output:

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

18.Implement “N-Queens Problem” using Backtracking

Program:

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

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

```
int board[20],count;
```

```
int main()
```

```
{
```

```
int n,i,j;
```

```
void queen(int row,int n);
```

```
printf(" - N Queens Problem Using Backtracking -");
```

```
printf("\n\nEnter number of Queens:");
```

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

```
queen(1,n);
```

```
return 0;
```

```
}
```

```
void print(int n)
```

```
{
```

```
int i,j;
```

```
printf("\n\nSolution %d:\n\n",++count);
```

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

```
printf("\t%d",i);
```

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

```

{
    printf("\n\n%d",i);
    for(j=1;j<=n;++j)
    {
        if(board[i]==j)
            printf("\tQ");
        else
            printf("\t-");
    }
}
}

```

```

int place(int row,int column)
{
    int i;
    for(i=1;i<=row-1;++i)
    {
        if(board[i]==column)
            return 0;
        else
            if(abs(board[i]-column)==abs(i-row))
                return 0;
    }
    return 1;
}

```

```

void queen(int row, int n)
{
    int column;
    for (column=1; column <=n;++column)
    {

```

```
if(place(row, column))
{
    board[row]=column;
    if(row==n)
        print(n);
    else
        queen(row+1,n);
}
}
}
```

Output:

```
/tmp/v0T8G6v42e.o
Enter the number of Queens

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

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

Total solutions=2
```