# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT on**

# Analysis and Design of Algorithms

***Submitted by***

**G SANJANA HEBBAR(1BM21CS062)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**B.M.S. COLLEGE OF ENGINEERING**
**BENGALURU-560019**
**June-2023 to Sep-2023**
**(Autonomous Institution under VTU)**

# 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 **G SANJANA HEBBAR(1BM21CS062),** who is a 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 academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

**Radhika A D**
Assistant Professor
Department of CSE
BMSCE, Bengaluru

**Dr. Jyothi S Nayak**
Professor and Head
Department of CSE
BMSCE, Bengaluru

# Index Sheet

## Course Outcome

| | |
|---|---|
| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| CO2 | Apply various design techniques for the given problem. |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems. |

# PROGRAM-1

## Q) 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.**

## CODE-

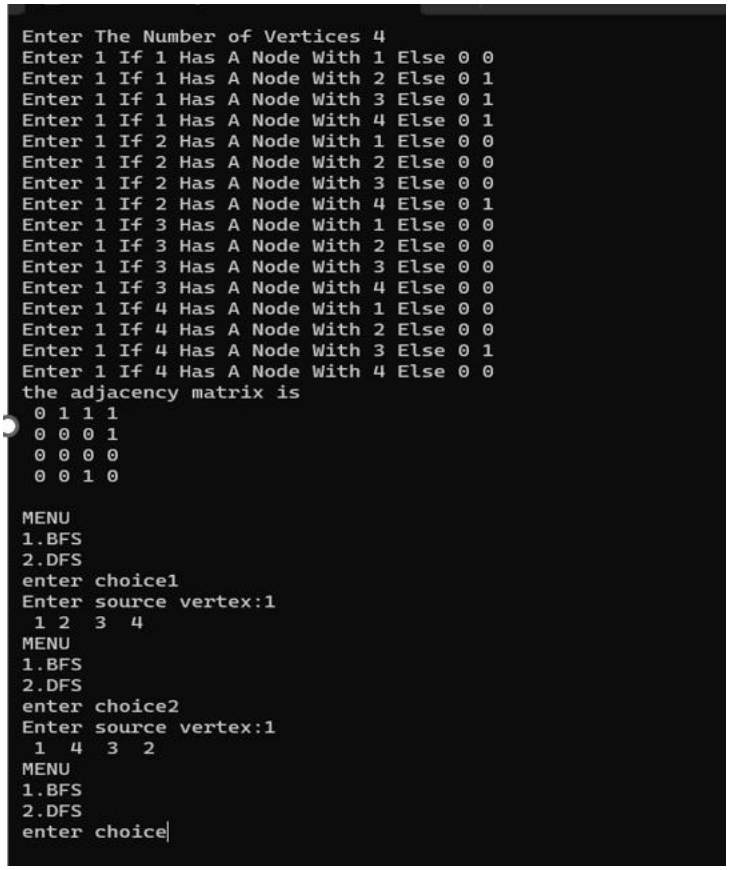
```
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();

void main()
{
int n,i,s,ch,j;
char c,dummy;
printf("ENTER THE NUMBER VERTICES ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
scanf("%d",&a[i][j]);
}
}
printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf(" %d",a[i][j]);
}
printf("\n");
}

do
{
for(i=1;i<=n;i++)
vis[i]=0;
printf("\nMENU");
printf("\n1.B.F.S");
printf("\n2.D.F.S");
printf("\nENTER YOUR CHOICE");
scanf("%d",&ch);
```

```
printf("ENTER THE SOURCE VERTEX :");
scanf("%d",&s);

switch(ch)
{
case 1:bfs(s,n);
break;
case 2:
dfs(s,n);
break;
}
printf("DO U WANT TO CONTINUE(Y/N) ? ");
scanf("%c",&dummy);
scanf("%c",&c);
}while((c=='y')||(c=='Y'));
}


//**************BFS(breadth-first search) code**************//
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
add(i);
vis[i]=1;
}
p=delete();
if(p!=0)
printf(" %d ",p);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}


void add(int item)
{
if(rear==19)
printf("QUEUE FULL");
else
{
if(rear==-1)
{
q[++rear]=item;
front++;
}
```

```
else
q[++rear]=item;
}
}
int delete()
{
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}


//***************DFS(depth-first search) code******************//
void dfs(int s,int n)
{
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(vis[i]==0))
{
push(i);
vis[i]=1;
}
k=pop();
if(k!=0)
printf(" %d ",k);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}
void push(int item)
{
if(top==19)
printf("Stack overflow ");
else
stack[++top]=item;
}
int pop()
{
int k;
if(top==-1)
return(0);
else
{
```

```
k=stack[top--];
return(k);
}
}
```

## OUTPUT-



```
Enter The Number of Vertices 4
Enter 1 If 1 Has A Node With 1 Else 0 0
Enter 1 If 1 Has A Node With 2 Else 0 1
Enter 1 If 1 Has A Node With 3 Else 0 1
Enter 1 If 1 Has A Node With 4 Else 0 1
Enter 1 If 2 Has A Node With 1 Else 0 0
Enter 1 If 2 Has A Node With 2 Else 0 0
Enter 1 If 2 Has A Node With 3 Else 0 0
Enter 1 If 2 Has A Node With 4 Else 0 1
Enter 1 If 3 Has A Node With 1 Else 0 0
Enter 1 If 3 Has A Node With 2 Else 0 0
Enter 1 If 3 Has A Node With 3 Else 0 0
Enter 1 If 3 Has A Node With 4 Else 0 0
Enter 1 If 4 Has A Node With 1 Else 0 0
Enter 1 If 4 Has A Node With 2 Else 0 0
Enter 1 If 4 Has A Node With 3 Else 0 1
Enter 1 If 4 Has A Node With 4 Else 0 0
the adjacency matrix is
 0 1 1 1
 0 0 0 1
 0 0 0 0
 0 0 1 0

MENU
1.BFS
2.DFS
enter choice1
Enter source vertex:1
 1 2  3  4
MENU
1.BFS
2.DFS
enter choice2
Enter source vertex:1
 1  4  3  2
MENU
1.BFS
2.DFS
enter choice
```

## PROGRAM-2

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

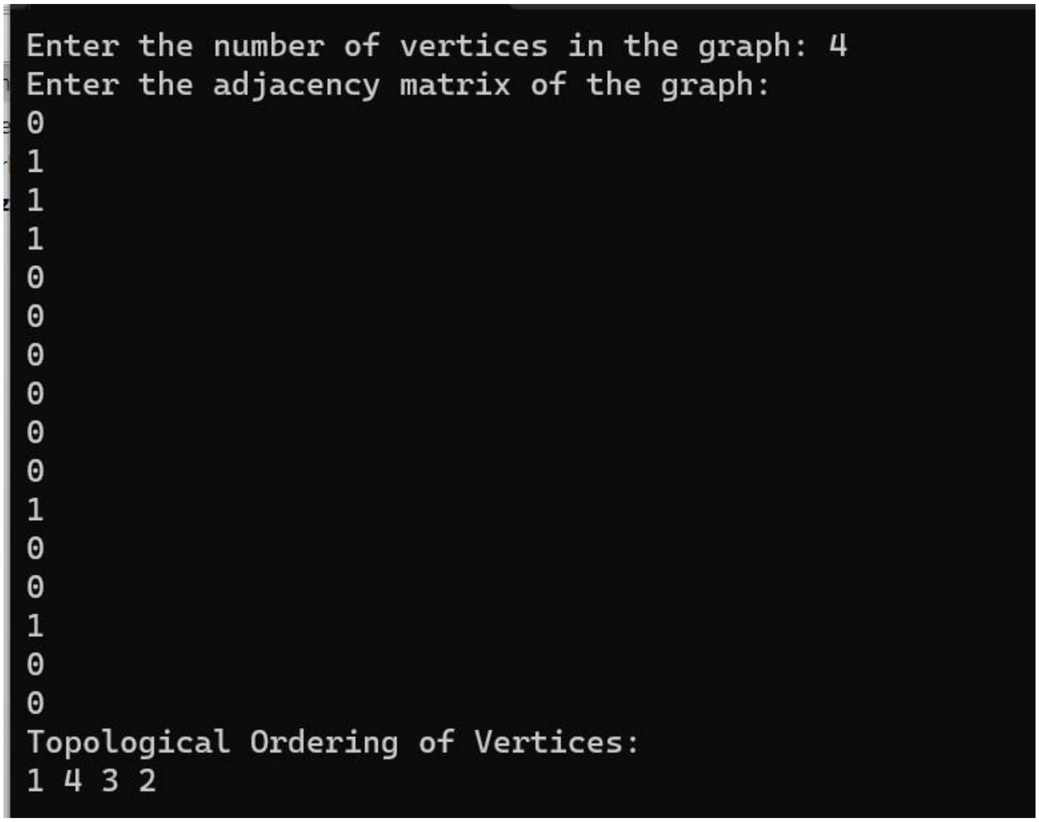
### CODE-

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

#define MAX_VERTICES 100

typedef struct {
    int vertices[MAX_VERTICES];
    int count;
} Stack;


void initialize(Stack* stack) {
   stack->count = 0;
                                          }


int isEmpty(Stack* stack) {
   return (stack->count == 0);
                                          }


void push(Stack* stack, int value) {
    stack->vertices[stack->count++] = value;
}


int pop(Stack* stack) {     if
(isEmpty(stack)) {
        printf("Error: Stack underflow\n");
exit(0);
    }
    return stack->vertices[--stack->count];
}
```

```
void topologicalSortDFS(int vertex, int** graph, int* visited, Stack* stack, int numVertices) {
    visited[vertex] = 1;
     int i;

    for (i = 0; i < numVertices; i++) {         if
(graph[vertex][i] && !visited[i]) {

            topologicalSortDFS(i, graph, visited, stack, numVertices);

        }

    }


    push(stack, vertex + 1);

}


void topologicalSort(int** graph, int numVertices) {
    Stack stack;
    int visited[MAX_VERTICES];
    int i;

    initialize(&stack);

    for (i = 0; i < numVertices; i++) {
        visited[i] = 0;

}

    for (i = 0; i < numVertices; i++) {
        if (!visited[i]) {
            topologicalSortDFS(i, graph, visited, &stack, numVertices);
        }

}

    printf("Topological Ordering of Vertices:\n");
    while (!isEmpty(&stack)) {
        printf("%d ", pop(&stack));

}
    printf("\n");                                   }


int main() {
    int numVertices, i, j;

    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &numVertices);
```

```
  int** graph = (int**)malloc(numVertices * sizeof(int*));
  for (i = 0; i < numVertices; i++) {
    graph[i] = (int*)malloc(numVertices * sizeof(int));

}

  printf("Enter the adjacency matrix of the graph:\n");
  for (i = 0; i < numVertices; i++) {
    for (j = 0; j < numVertices; j++) {
      scanf("%d", &graph[i][j]);
    }

}

  topologicalSort(graph, numVertices);


  return 0;
}
```

**OUTPUT-**



```
Enter the number of vertices in the graph: 4
Enter the adjacency matrix of the graph:
0
1
1
1
0
0
0
0
0
0
1
0
0
1
0
0
Topological Ordering of Vertices:
1 4 3 2
```

## PROGRAM-3

**Q) Implement Johnson Trotter algorithm to generate permutations.**

**CODE-**

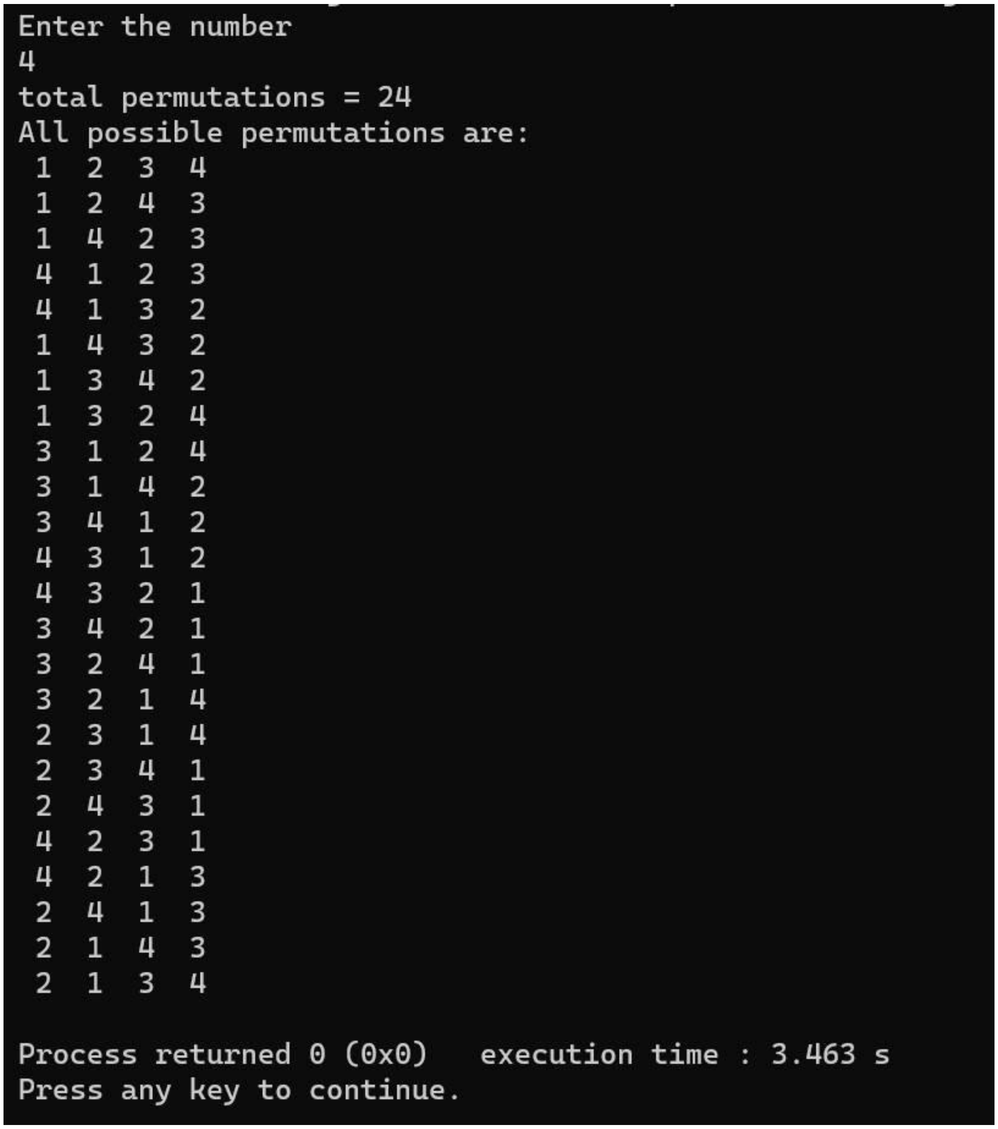
```
#include <stdio.h>
#include <stdlib.h>
int flag = 0;
int swap(int *a,int *b) {
int t = *a; *a = *b;
*b = t;
}
int search(int arr[],int num,int mobile)
 {
int g;
for(g=0;g<num;g++) {
 if(arr[g] == mobile)
return g+1;
 else flag++;
}
return -1;
}
int find_Moblie(int arr[],int d[],int num)
{
int mobile = 0;
int mobile_p = 0;
int i;
for(i=0;i<num;i++)
{
if((d[arr[i]-1] == 0) && i != 0)
{
```

```
if(arr[i]>arr[i-1] && arr[i]>mobile_p)
{
mobile = arr[i];
mobile_p = mobile;
}
else flag++;
}
else if((d[arr[i]-1] == 1) & i != num-1)
{
if(arr[i]>arr[i+1] && arr[i]>mobile_p)
{
mobile = arr[i];
mobile_p = mobile;
}
else
flag++;
}
else flag++; }
if((mobile_p == 0) && (mobile == 0))
return 0; else return mobile;
}
void permutations(int arr[],int d[],int num)
{ int i;
int mobile = find_Moblie(arr,d,num);
int pos = search(arr,num,mobile);
if(d[arr[pos-1]-1]==0)
swap(&arr[pos-1],&arr[pos-2]);
else swap(&arr[pos-1],&arr[pos]);
for(int i=0;i<num;i++)
```

```
{
if(arr[i] > mobile)
{
if(d[arr[i]-1]==0) d[arr[i]-1] = 1;
else
d[arr[i]-1] = 0;
}
}
for(i=0;i<num;i++)
{
printf(" %d ",arr[i]);
} }
int factorial(int k)
{ int f = 1;
int i = 0;
 for(i=1;i<k+1;i++)
 f = f*i;
return f;
}
int main()
{

int num = 0;
int i; int j; int z = 0;
printf("Johnson trotter algorithm to find all permutations of given numbers\n");
printf("Enter the number\n");
 scanf("%d",&num);
 int arr[num],d[num];
z = factorial(num);
```

```
printf("total permutations = %d",z);

printf("\nAll possible permutations are: \n");
for(i=0;i<num;i++)

{ d[i] = 0; arr[i] = i+1;
printf(" %d ",arr[i]);

} printf("\n");

for(j=1;j<z;j++) {

 permutations(arr,d,num);

printf("\n");

}

return 0;

}
```

## OUTPUT-



```
Enter the number
4
total permutations = 24
All possible permutations are:
 1  2  3  4
 1  2  4  3
 1  4  2  3
 4  1  2  3
 4  1  3  2
 1  4  3  2
 1  3  4  2
 1  3  2  4
 3  1  2  4
 3  1  4  2
 3  4  1  2
 4  3  1  2
 4  3  2  1
 3  4  2  1
 3  2  4  1
 3  2  1  4
 2  3  1  4
 2  3  4  1
 2  4  3  1
 4  2  3  1
 4  2  1  3
 2  4  1  3
 2  1  4  3
 2  1  3  4

Process returned 0 (0x0)   execution time : 3.463 s
Press any key to continue.
```

## PROGRAM-4

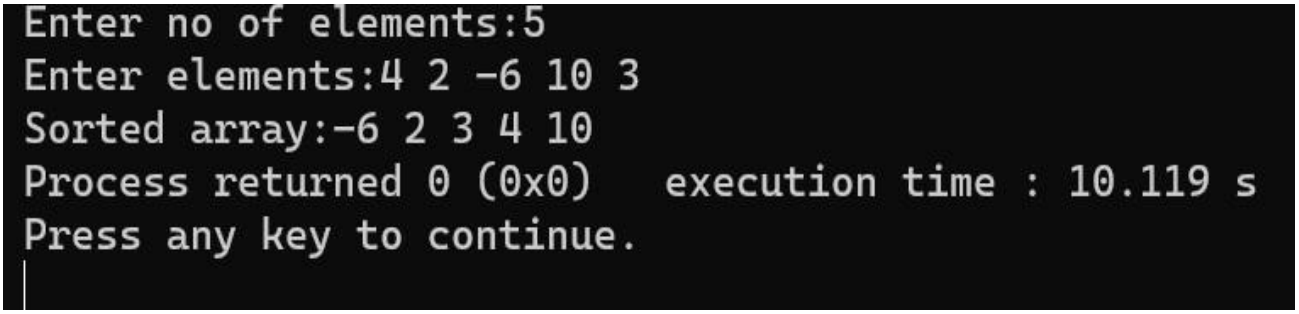
**Q) Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.**

**CODE-**

```
#include <stdio.h>
#include <stdlib.h>
 void merge(int low,int mid,int high,int array[20],int mer[20])
{
    int i = low;     int j = mid+1;
    int k = 0;
  while(i<=mid && j<=high)
    {
        if(array[i]<array[j])
        {
            mer[k] = array[i];
            i++;            k++;
        }        else
        {
            mer[k] = array[j];
            j++;            k++;
        }
    }
  while (i <= mid)
    {
        mer[k] = array[i];
        i++;
         k++;
    }
    while (j <= high)
    {
        mer[k] = array[j];
        j++;         k++;
    }

    for(int i=0;i<k;i++)
    {
        array[low+i] = mer[i];
    }
}

void merge_sort(int low,int high,int array[20],int merged[20])
{
    if(low<high)
    {
        int mid = (low+high)/2;
 merge_sort(low,mid,array,merged);
 merge_sort(mid+1,high,array,merged);
 merge(low,mid,high,array,merged);
    }
}
 int main()
{
 int n,array[30];
 printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter elements:");
```

```
    for(int i=0;i<n;i++)
    {
        scanf("%d",&array[i]);
    }

    int merged[30];
 merge_sort(0,n-1,array,merged);
printf("Sorted array:");

    for(int i=0;i<n;i++)
    {
        printf("%d ",array[i]);
    }
}
```

**OUTPUT-**



```
Enter no of elements:5
Enter elements:4 2 -6 10 3
Sorted array:-6 2 3 4 10
Process returned 0 (0x0)   execution time : 10.119 s
Press any key to continue.
```

## PROGRAM-5

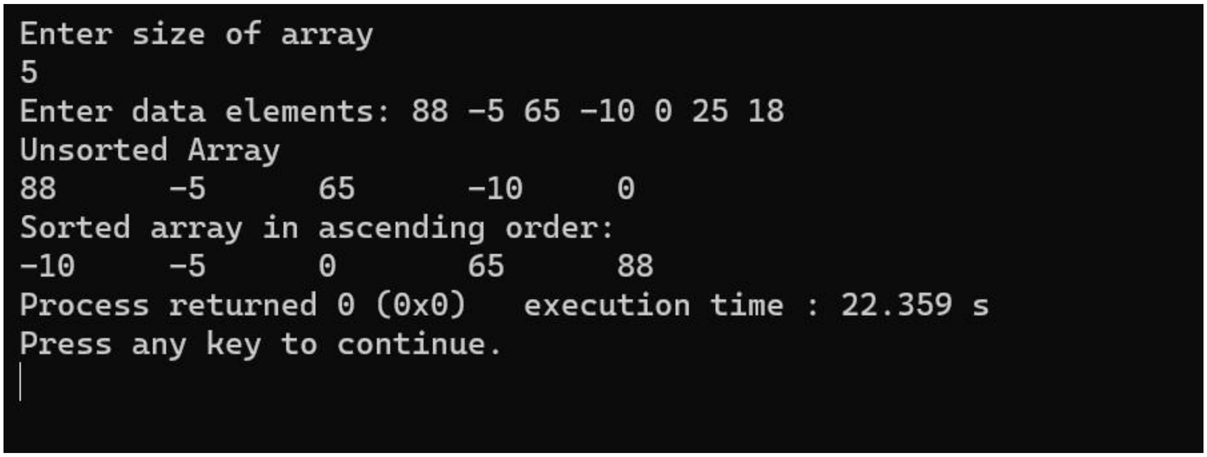
**Q) Sort a given set of N integer elements using Quick Sort technique and compute its time taken**

**CODE-**

```
#include <stdio.h>

 void swap(int *a, int *b) {

 int t = *a;

*a = *b;

*b = t;

}

 int partition(int a[], int l, int h) {

 int pivot = a[l];

 int i = l,

j = h;

 while (i < j) {

 while (a[i] <= pivot && i <= h) {

 i++;    }

  while (a[j] > pivot) {

     j--;    }    if (i < j) {

     swap(&a[i], &a[j]);

    }

  }

  swap(&a[l], &a[j]);
  return j;

}

 void quickSort(int a[], int l, int h) {

 if (l < h) {

  int pi = partition(a, l, h);

  quickSort(a, l, pi - 1);

quickSort(a, pi + 1, h);

  }

}

 int main() {

int a[20], n, i;

 printf("Enter size of array\n");

  scanf("%d", &n);

 printf("Enter data elements: ");

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

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

  }

 printf("Unsorted Array\n");   for (i = 0; i < n; i++) {     printf("%d\t", a[i]);

  }

  quickSort(a, 0, n - 1);


  printf("\nSorted array in ascending order: \n");
```

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

 printf("%d\t", a[i]);

 }

 return 0;

}
```

## OUTPUT



```
Enter size of array
5
Enter data elements: 88 -5 65 -10 0 25 18
Unsorted Array
88      -5      65      -10     0
Sorted array in ascending order:
-10     -5      0       65      88
Process returned 0 (0x0)   execution time : 22.359 s
Press any key to continue.
```

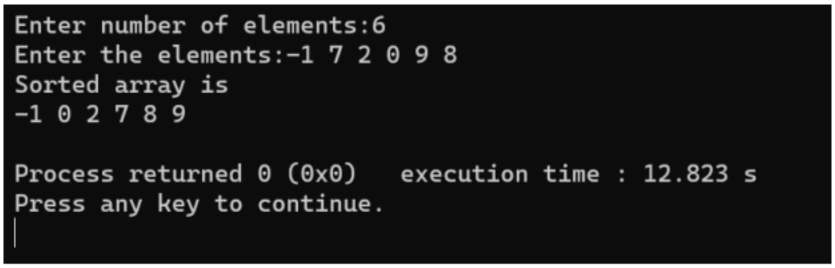
## PROGRAM-6

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

### CODE-

```
#include <stdio.h>

void swap(int* a, int* b)

{

int temp = *a;

*a = *b;

*b = temp;

}

void heapify(int arr[], int N, int i)

{

int largest = i;

int left = 2 * i + 1;

int right = 2 * i + 2;

if (left < N && arr[left] > arr[largest])

largest = left;

if (right < N && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, N, largest);

}

}

void heapSort(int arr[], int N)

{

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);
}
}
void printArray(int arr[], int N)
{
for (int i = 0; i < N; i++)
printf("%d ", arr[i]);
printf("\n");
}
int main()
{
int n;
printf("Enter number of elements:");
scanf("%d",&n);
int arr[n];
printf("Enter the elements:");
for (int i=0;i<n;i++)
{
scanf("%d",&arr[i]);
}
heapSort(arr, n);
printf("Sorted array is\n");
printArray(arr, n);
}
```

## OUTPUT



```
Enter number of elements:6
Enter the elements:-1 7 2 0 9 8
Sorted array is
-1 0 2 7 8 9

Process returned 0 (0x0)   execution time : 12.823 s
Press any key to continue.
```

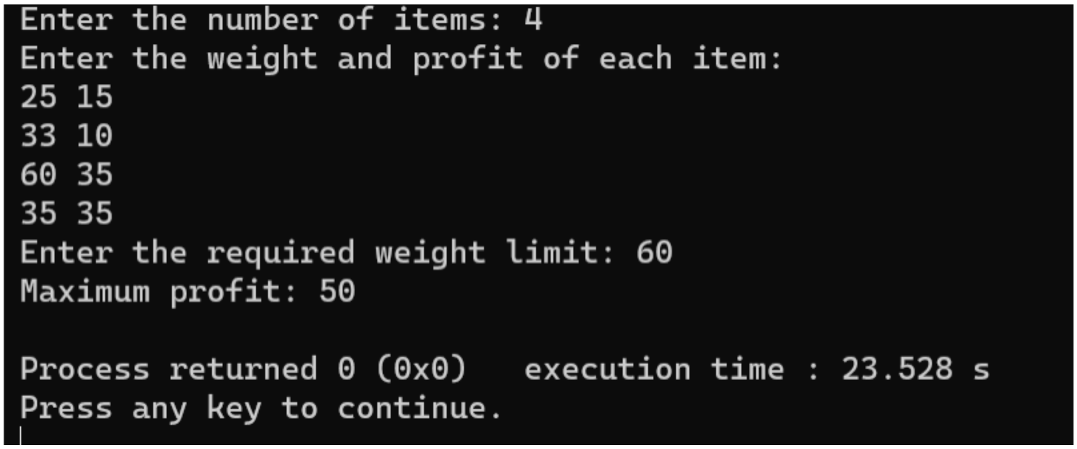
## PROGRAM-7

**Q)Implement 0/1 Knapsack problem using dynamic programming.**

**CODE-**

```
#include <stdio.h>
int knap(int w[], int p[], int n, int ww) {
int v[n+1][ww+1];
 for (int i = 0; i < n + 1; i++) {
 for (int j = 0; j < ww + 1; j++) {
 if (i == 0 || j == 0) {
 v[i][j] = 0;
 continue;
 } else {
 if (w[i - 1] > j) {
 v[i][j] = v[i - 1][j];
 } else {
 if (v[i - 1][j] > (v[i - 1][j - w[i - 1]] + p[i - 1])) {
 v[i][j] = v[i - 1][j];
 } else {
 v[i][j] = v[i - 1][j - w[i - 1]] + p[i - 1];
 }
 }
 }
 }
 }
 int q = v[n][ww];
 return q;
}
int main() {
 int w[10], p[10], n, ww, ans;
printf("Enter the number of items: ");
scanf("%d", &n);
printf("Enter the weight and profit of each item:\n");
for (int i = 0; i < n; i++) {
scanf("%d %d", &w[i], &p[i]);
}
printf("Enter the required weight limit: ");
scanf("%d", &ww);
ans = knap(w, p, n, ww);
printf("Maximum profit: %d\n", ans);
return 0;
}
```

**OUTPUT-**



```
Enter the number of items: 4
Enter the weight and profit of each item:
25 15
33 10
60 35
35 35
Enter the required weight limit: 60
Maximum profit: 50

Process returned 0 (0x0)   execution time : 23.528 s
Press any key to continue.
```

## PROGRAM-8

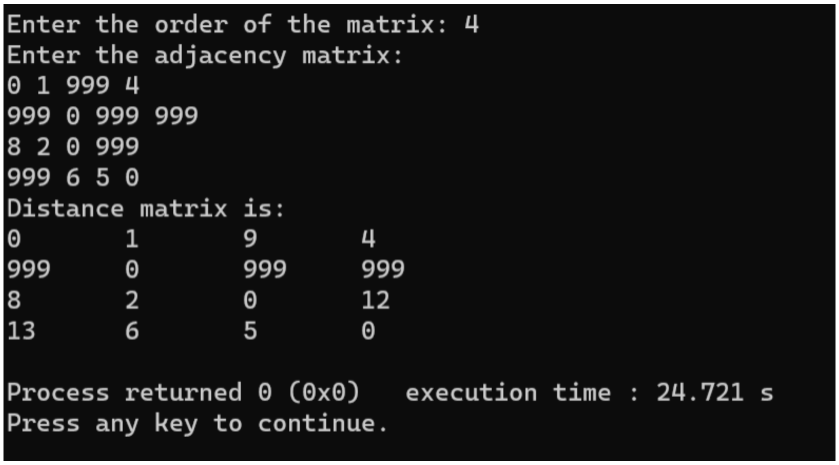
**Q) Implement All Pair Shortest paths problem using Floyd's algorithm.**

**CODE-**

```
#include <stdio.h>
int min(int a, int b) {
if (a < b)
return a;
else
return b;
}
void printm(int n, int d[][10]) {
printf("Distance matrix is:\n");
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
printf("%d\t", d[i][j]);
}
printf("\n");
}
}
void floyd(int n, int a[][10]) {
int d[10][10], i, j, k;
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
d[i][j] = a[i][j];
}
}
for (k = 0; k < n; k++) {
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
```

```
d[i][j] = min(d[i][j], (d[i][k] + d[k][j]));
}
}
}
printm(n, d);
}
int main() {
int a[10][10], i, j, n;
printf("Enter the order of the matrix: ");
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]);
}
}
floyd(n, a);
return 0;
}
```

## OUTPUT



```
Enter the order of the matrix: 4
Enter the adjacency matrix:
0 1 999 4
999 0 999 999
8 2 0 999
999 6 5 0
Distance matrix is:
0       1       9       4
999     0       999     999
8       2       0       12
13      6       5       0

Process returned 0 (0x0)   execution time : 24.721 s
Press any key to continue.
```

## PROGRAM-9

### Q) Find Minimum Cost Spanning Tree of a given undirected graph using Prim/Kruskal's algorithm.

### CODE

**Prim's Algorithm-**

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
int V;
int minKey(int key[], bool mstSet[]) {
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++) {
if (mstSet[v] == false && key[v] < min) {
min = key[v];
min_index = v;
}
}
return min_index;
}
int printMST(int parent[], int graph[V][V]) {
int sum = 0;
printf("Edge \tWeight\n");
for (int i = 1; i < V; i++) {
printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
sum += graph[i][parent[i]];
}
printf("weight=%d\n", sum);
}
void primMST(int graph[V][V]) {
```

```
int parent[V];
int key[V];
bool mstSet[V];
for (int i = 0; i < V; i++) {
key[i] = INT_MAX;
mstSet[i] = false;
}
key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {
int u = minKey(key, mstSet);
mstSet[u] = true;
for (int v = 0; v < V; v++) {
if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {
parent[v] = u;
key[v] = graph[u][v];
}
}
}
printMST(parent, graph);
}
int main() {
printf("Enter the number of vertices: ");
scanf("%d", &V);
int graph[V][V];
printf("Enter the adjacency matrix:\n");
for (int i = 0; i < V; i++) {
for (int j = 0; j < V; j++) {
scanf("%d", &graph[i][j]);
}
}
primMST(graph);
return 0;
}
```

**Krushkal's Algorithm-**

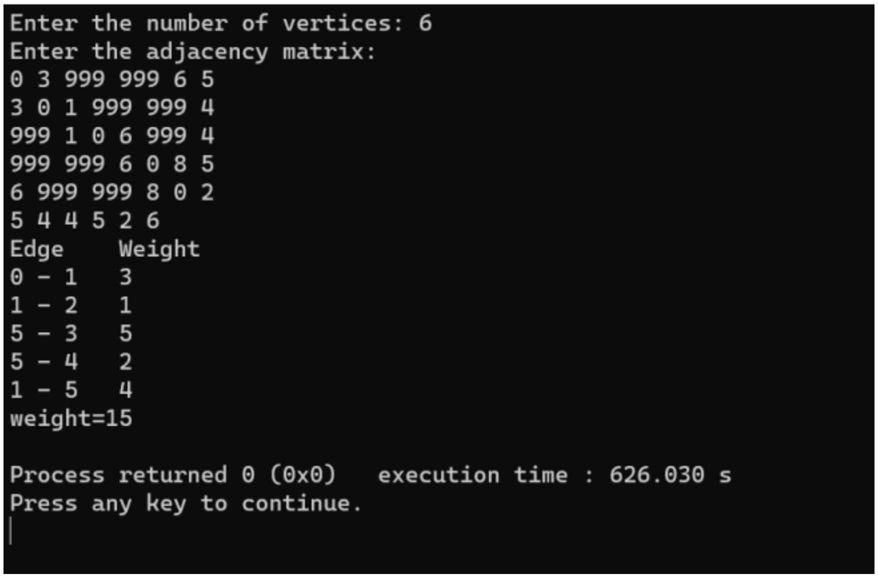
```
#include <stdio.h>
int find(int v, int parent[10])
{
 while (parent[v] != v)
 {
 v = parent[v];
 }
 return v;
}
void union1(int i, int j, int parent[10])
{
 if (i < j)
 parent[j] = i;
 else
 parent[i] = j;
}
void kruskal(int n, int a[10][10])
{
 int count, k, min, sum, i, j, t[10][10], u, v, parent[10];
 count = 0;
 k = 0;
 sum = 0;
 for (i = 0; i < n; i++)
```

```
parent[i] = i;
while (count != n - 1)
{
min = 999;
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
if (a[i][j] < min && a[i][j] != 0)
{
min = a[i][j];
u = i;
v = j;
}
}
}
i = find(u, parent);
j = find(v, parent);
if (i != j)
{
union1(i, j, parent);
t[k][0] = u;
t[k][1] = v;
k++;
count++;
sum = sum + a[u][v];
}
a[u][v] = a[v][u] = 999;
}
```

```
 if (count == n - 1)
 {
 printf("spanning tree\n");
 for (i = 0; i < n - 1; i++)
 {
 printf("%d %d\n", t[i][0], t[i][1]);
 }
 printf("cost of spanning tree=%d\n", sum);
 }
 else
 printf("spanning tree does not exist\n");
}
int main()
{
 int n, i, j, a[10][10];
 printf("enter the 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]);
 }
 kruskal(n, a);
 return 0;
}
```

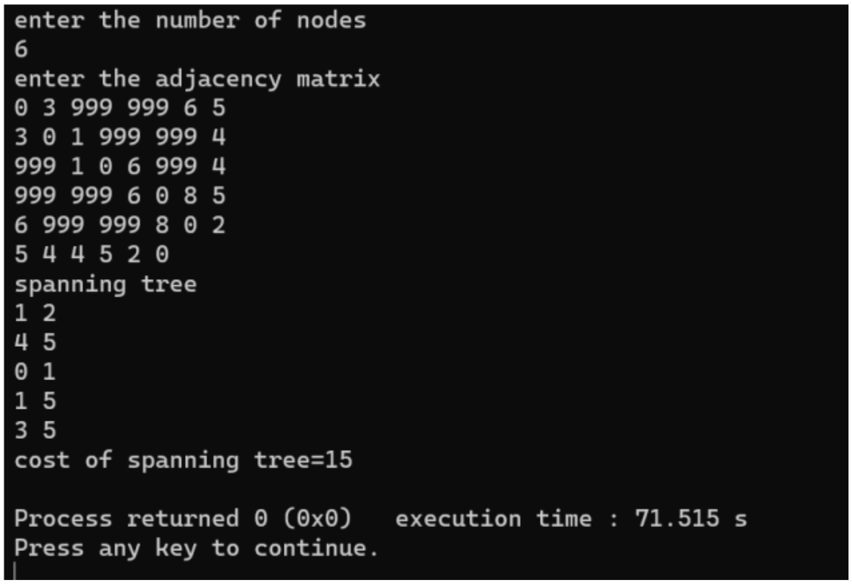
# OUTPUT

## Prim’s Algorithm



```
Enter the number of vertices: 6
Enter the adjacency matrix:
0 3 999 999 6 5
3 0 1 999 999 4
999 1 0 6 999 4
999 999 6 0 8 5
6 999 999 8 0 2
5 4 4 5 2 6
Edge    Weight
0 - 1   3
1 - 2   1
5 - 3   5
5 - 4   2
1 - 5   4
weight=15

Process returned 0 (0x0)   execution time : 626.030 s
Press any key to continue.
```

## Krushkal’s Algorithm



```
enter the number of nodes
6
enter the adjacency matrix
0 3 999 999 6 5
3 0 1 999 999 4
999 1 0 6 999 4
999 999 6 0 8 5
6 999 999 8 0 2
5 4 4 5 2 0
spanning tree
1 2
4 5
0 1
1 5
3 5
cost of spanning tree=15

Process returned 0 (0x0)   execution time : 71.515 s
Press any key to continue.
```

## PROGRAM-10

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

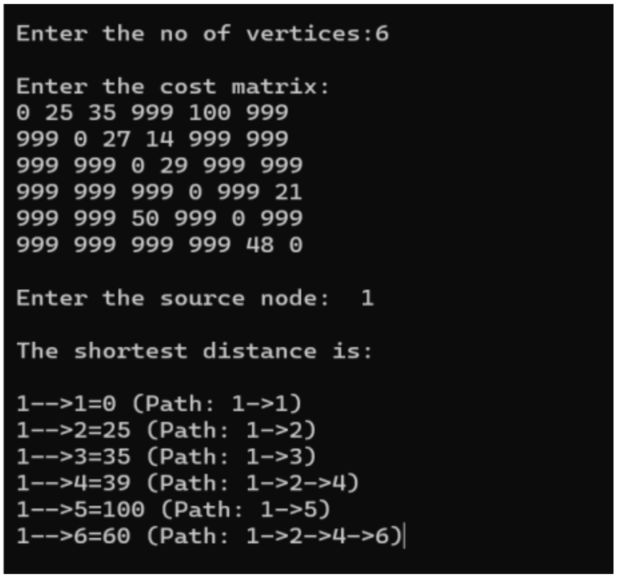
### CODE

```
#include <stdio.h>
#include <conio.h>
void dijkstras();
int c[10][10], n, src;
void printPath(int parent[], int node);
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();
 getch();
}
void dijkstras()
{
```

```
int vis[10], dist[10], parent[10], u, j, count, min;
for (j = 1; j <= n; j++)
{
dist[j] = c[src][j];
parent[j] = src;
}
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];
 parent[j] = u;
 }
 }
 }
 printf("\nThe shortest distance is:\n");
 for (j = 1; j <= n; j++)
 {
 printf("\n%d-->%d=%d (Path: %d", src, j, dist[j], src);
 printPath(parent, j);
 printf(")");
 }
}
void printPath(int parent[], int node)
{
 if (parent[node] == src)
 {
 printf("->%d", node);
 return;
 }
 printPath(parent, parent[node]);
 printf("->%d", node);
}
```

## OUTPUT



```
Enter the no of vertices:6

Enter the cost matrix:
0 25 35 999 100 999
999 0 27 14 999 999
999 999 0 29 999 999
999 999 999 0 999 21
999 999 50 999 0 999
999 999 999 999 48 0

Enter the source node:  1

The shortest distance is:

1-->1=0 (Path: 1->1)
1-->2=25 (Path: 1->2)
1-->3=35 (Path: 1->3)
1-->4=39 (Path: 1->2->4)
1-->5=100 (Path: 1->5)
1-->6=60 (Path: 1->2->4->6)
```

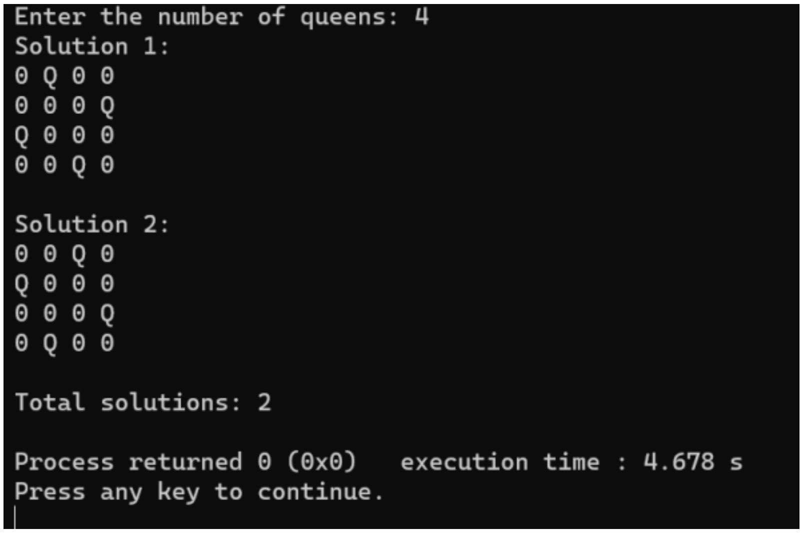
## PROGRAM-11

### Q) Implement “N-Queens Problem” using Backtracking

### CODE

```
#include <stdio.h>
#include <math.h>
int x[20]; // Solution array to store column index of queens
int count = 0;
int place(int k, int i) {
for (int j = 1; j <= k - 1; j++) {
if (x[j] == i || abs(x[j] - i) == abs(j - k)) {
return 0;
}
}
return 1;
}
void nqueens(int k, int n) {
for (int i = 1; i <= n; i++) {
if (place(k, i)) {
x[k] = i;
if (k == n) {
count++;
printf("Solution %d:\n", count);
for (int j = 1; j <= n; j++) {
for (int l = 1; l <= n; l++) {
if (x[j] == l) {
printf("Q ");
} else {
printf("0 ");
}
}
printf("\n");
}
printf("\n");
} else {
nqueens(k + 1, n);
}
}
}
}
int main() {
int n;
printf("Enter the number of queens: ");
scanf("%d", &n);
if (n <= 0) {
printf("Invalid input.\n");
return 1;
}
nqueens(1, n);
if (count == 0) {
printf("No solutions found for %d queens.\n", n);
} else {
printf("Total solutions: %d\n", count);
}
return 0;
}
```

## OUTPUT-



```
Enter the number of queens: 4
Solution 1:
0 Q 0 0
0 0 0 Q
Q 0 0 0
0 0 Q 0

Solution 2:
0 0 Q 0
Q 0 0 0
0 0 0 Q
0 Q 0 0

Total solutions: 2

Process returned 0 (0x0)   execution time : 4.678 s
Press any key to continue.
```