# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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

**LAB REPORT**
**on**

# Analysis and Design of Algorithms

***Submitted by***

**YASHASVINI M R (1BM21CS252)**

***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-2023 to July-2023**

# 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 **YASHASVINI M R (1BM21CS252),** 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 academic semester May-2023 to July-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 | Dr. Jyothi S Nayak |
| Assistant Professor | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | 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. |

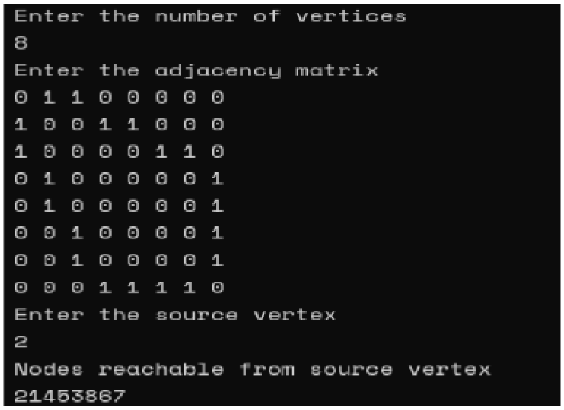
**1.Write program to do the following:**

**a. Print all the nodes reachable from a given starting node in a digraph using BFS method.**

```
#include<stdio.h>
void bfs(int);
int a[10][10],vis[10],n;

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

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

    }

    vis[i]=0;
  }

  printf("Enter the source vertex\n");
  scanf("%d",&src);
  printf("Nodes reachable from source vertex\n");
  bfs(src);

}

void bfs(int v)
{
   int q[10],f=1,r=1,u,i;
   q[r]=v;
   vis[v]=1;
   while(f<=r)
   {
     u=q[f];
```

```
        printf("%d",u);
        for(i=1;i<=n;i++)
        {
            if(a[u][i]==1 && vis[i]==0)
            {
               vis[i]=1;
               r=r+1;
               q[r]=i;
            }
        }
        f=f+1;
    }
}
```

OUTPUT



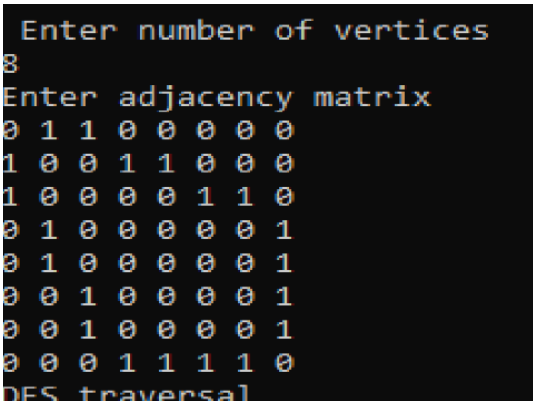
```
Enter the number of vertices
8
Enter the adjacency matrix
0 1 1 0 0 0 0 0
1 0 0 1 1 0 0 0
1 0 0 0 0 1 1 0
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 0 1 1 1 1 0
Enter the source vertex
2
Nodes reachable from source vertex
21453867
```

### b. Check whether a given graph is connected or not using DFS method.

```
#include
#include
int a[10][10], n, vis[10];
int dfs();
int main()
{
          int i, j;
          printf("\n Enter number of vertices\n");
          scanf("%d",&n);
          printf("Enter adjacency matrix\n");
          for(i=1;i<=n;i++)
          {
                    for(j=1;j<=n;j++)
                    {
```

```
                    scanf("%d",&a[i][j]);
                }
        }
        for(i=1;i<=n;i++)
        vis[i]=0;
        printf("DFS traversal\n");
        for(i=1;i<=n;i++)
        {
                if(vis[i]==0)
                dfs(i);
        }
        return 0;
}
int dfs(int v)
{
        int i; vis[v]=1;
        printf("%d",v);
        for(i=1;i<=n;i++)
        {
                if(a[v][i]==1&& vis[i]==0)
                dfs(i);
        }
        return 0;
}
```

OUTPUT



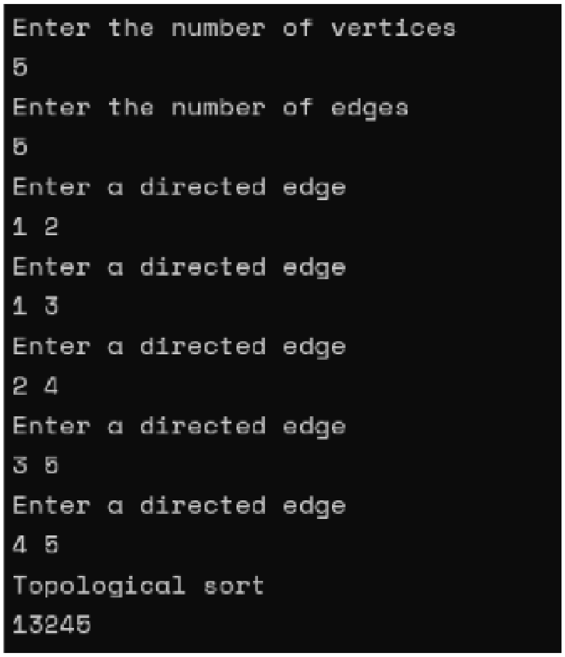
```
 Enter number of vertices
8
Enter adjacency matrix
0 1 1 0 0 0 0 0
1 0 0 1 1 0 0 0
1 0 0 0 0 1 1 0
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 0 1 1 1 1 0
DFS traversal
```

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

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

void dfs(int);
int a[10][10],vis[10],exp[10],n,j,m;

void main()
{
  int i,x,y;
  printf("Enter the number of vertices\n");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=n;j++)
    {
         a[i][j]=0;
    }
    vis[i]=0;
  }
   printf("Enter the number of edges\n");
   scanf("%d",&m);
   for(i=1;i<=m;i++)
   {
     printf("Enter a directed edge\n");
     scanf("%d %d",&x,&y);
     a[x][y]=1;
   }
   j=0;
   for(i=1;i<=n;i++)
   {
     if(vis[i]==0)
          dfs(i);
   }
   printf("Topological sort\n");
   for(i=n-1;i>=0;i--)
   {
     printf("%d",exp[i]);
   }
```

```
 getch();
}

void dfs(int v)
{
  int i;
  vis[v]=1;
  for(i=1;i<=n;i++)
  {
    if(a[v][i]==1 && vis[i]==0)
    dfs(i);
  }
  exp[j++]=v;
}
```

OUTPUT



```
Enter the number of vertices
5
Enter the number of edges
5
Enter a directed edge
1 2
Enter a directed edge
1 3
Enter a directed edge
2 4
Enter a directed edge
3 5
Enter a directed edge
4 5
Topological sort
13245
```

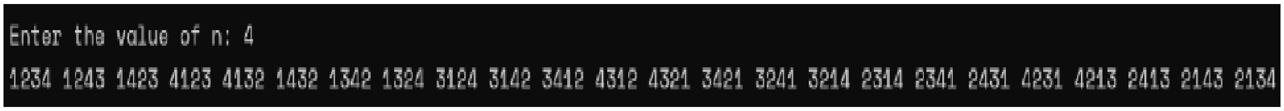
### 3. Implement Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>

#include <stdbool.h>



#define RIGHT_TO_LEFT false

#define LEFT_TO_RIGHT true



int searchArr(int a[], int n, int mobile)

{

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

      if (a[i] == mobile)

         return i + 1;

   return -1;

}



int getMobile(int a[], bool dir[], int n)

{

   int mobile_prev = 0, mobile = 0;

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

   {

      if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0)

      {

         if (a[i] > a[i - 1] && a[i] > mobile_prev)

         {

            mobile = a[i];
```

```
                mobile_prev = mobile;
            }
        }

        if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1)
        {
            if (a[i] > a[i + 1] && a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    if (mobile == 0 && mobile_prev == 0)
        return 0;
    else
    return mobile;
}

void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
```

```
}

void printOnePerm(int a[], bool dir[], int n)
{
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT)
        swap(&a[pos - 1], &a[pos - 2]);
    else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
        swap(&a[pos], &a[pos - 1]);

    for (int i = 0; i < n; i++)
    {
        if (a[i] > mobile)
        {
            if (dir[a[i] - 1] == LEFT_TO_RIGHT)
                dir[a[i] - 1] = RIGHT_TO_LEFT;
            else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
                dir[a[i] - 1] = LEFT_TO_RIGHT;
        }
    }

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

int fact(int n)
{
    int res = 1;
    for (int i = 1; i <= n; i++)
        res = res * i;
    return res;
}

void printPermutation(int n)
{
    int a[n];
    bool dir[n];

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

    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;
    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}


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

OUTPUT



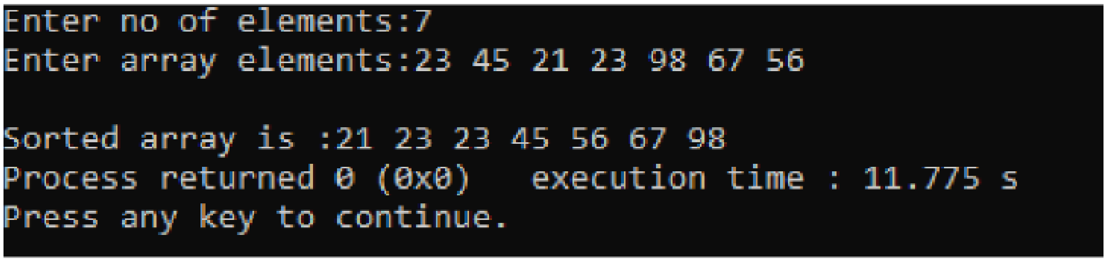
```
Enter the value of n: 4
1234 1243 1423 4123 4132 1432 1342 1324 3124 3142 3412 4312 4321 3421 3241 3214 2314 2341 2431 4231 4213 2413 2143 2134
```

**4. 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.**

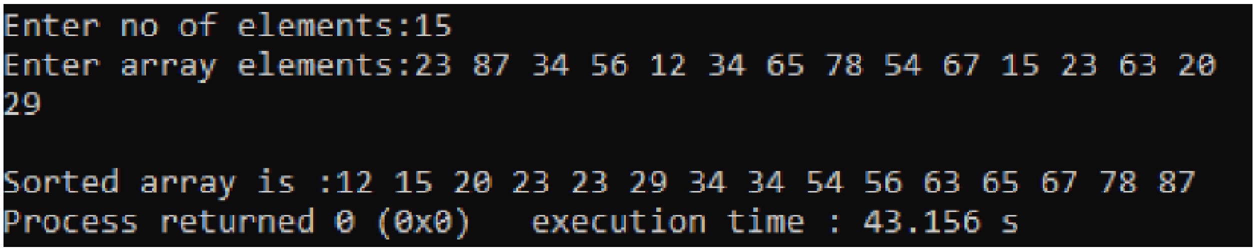
```
#include<stdio.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
   int a[30],n,i;
   printf("Enter no of elements:");
   scanf("%d",&n);
   printf("Enter array elements:");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   mergesort(a,0,n-1);
   printf("\nSorted array is :");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);
   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[50];
   int i,j,k;
   i=i1;
```

```
    j=i2;
    k=0;
    while(i<=j1 && j<=j2)
    {
        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

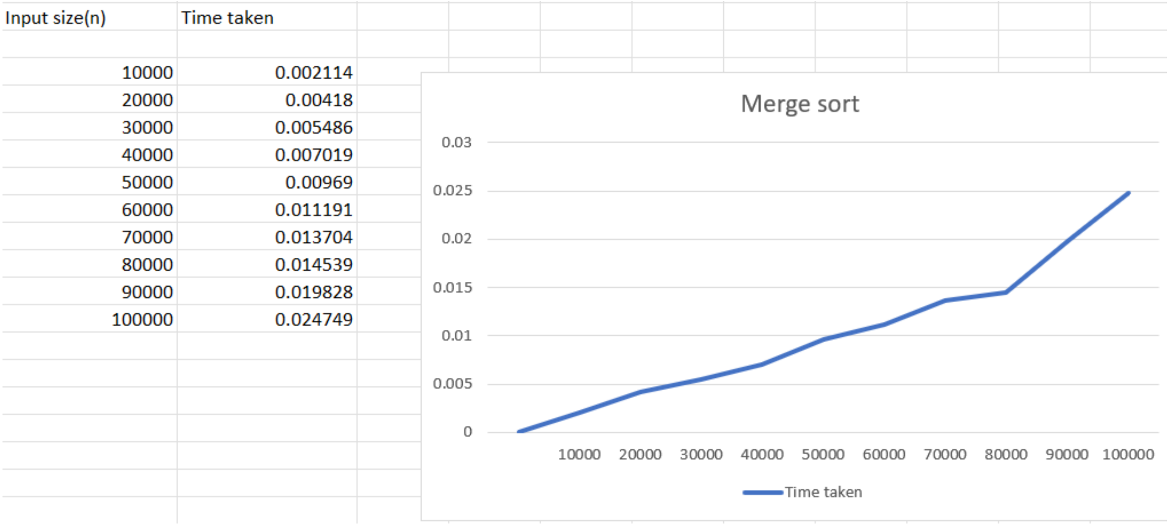


```
Enter no of elements:7
Enter array elements:23 45 21 23 98 67 56

Sorted array is :21 23 23 45 56 67 98
Process returned 0 (0x0)   execution time : 11.775 s
Press any key to continue.
```



```
Enter no of elements:15
Enter array elements:23 87 34 56 12 34 65 78 54 67 15 23 63 20
29

Sorted array is :12 15 20 23 23 29 34 34 54 56 63 65 67 78 87
Process returned 0 (0x0)   execution time : 43.156 s
```

GRAPH



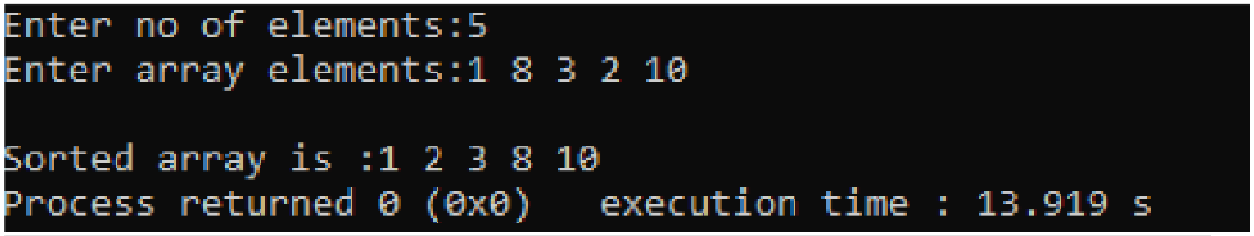
| Input size(n) | Time taken |
|---|---|
| 10000 | 0.002114 |
| 20000 | 0.00418 |
| 30000 | 0.005486 |
| 40000 | 0.007019 |
| 50000 | 0.00969 |
| 60000 | 0.011191 |
| 70000 | 0.013704 |
| 80000 | 0.014539 |
| 90000 | 0.019828 |
| 100000 | 0.024749 |

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

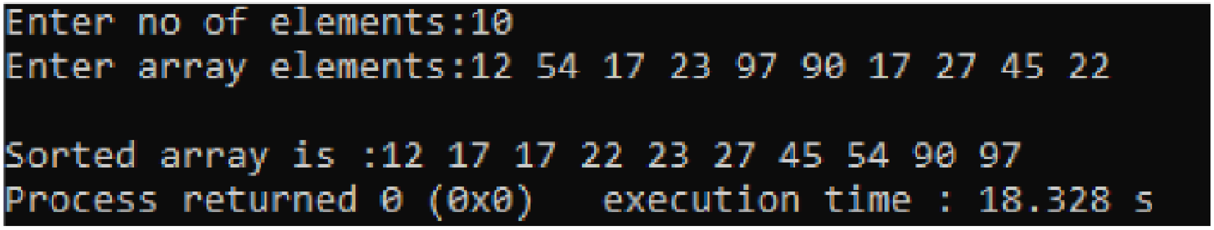
```
#include<stdio.h>

void qsort(int a[], int low, int high)
{
   int mid;
   if(low<high)
   {
      mid=partition(a,low,high);
      qsort(a,low,mid-1);
      qsort(a,mid+1, high);
   }
}
int partition(int a[],int low, int high)
{
   int i,j,temp, pivot;
   pivot=a[low];
   i=low+1;
   j=high;
   while(i<=j)
   {
      while(a[i]<=pivot)
         i++;
      while(a[j]>pivot)
         j--;
      if(i<j)
      {
         temp=a[i];
         a[i]=a[j];
         a[j]=temp;
      }
   }
   temp=a[low];
   a[low]=a[j];
   a[j]=temp;
   return j;
}
```

```
int main()
{
   int a[30],n,i;
   printf("Enter no of elements:");
   scanf("%d",&n);
   printf("Enter array elements:");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   qsort(a,0,n-1);
   printf("\nSorted array is :");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);
   return 0;
}
```

OUTPUT

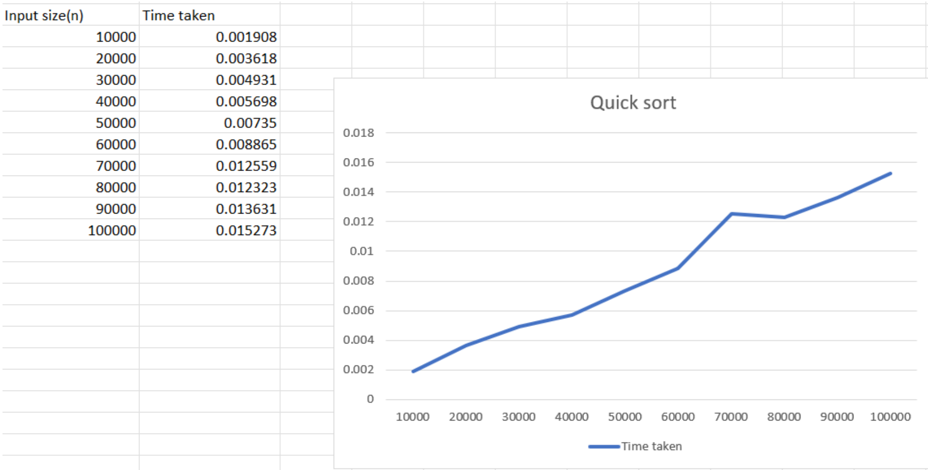


```
Enter no of elements:5
Enter array elements:1 8 3 2 10

Sorted array is :1 2 3 8 10
Process returned 0 (0x0)   execution time : 13.919 s
```



```
Enter no of elements:10
Enter array elements:12 54 17 23 97 90 17 27 45 22

Sorted array is :12 17 17 22 23 27 45 54 90 97
Process returned 0 (0x0)   execution time : 18.328 s
```

GRAPH



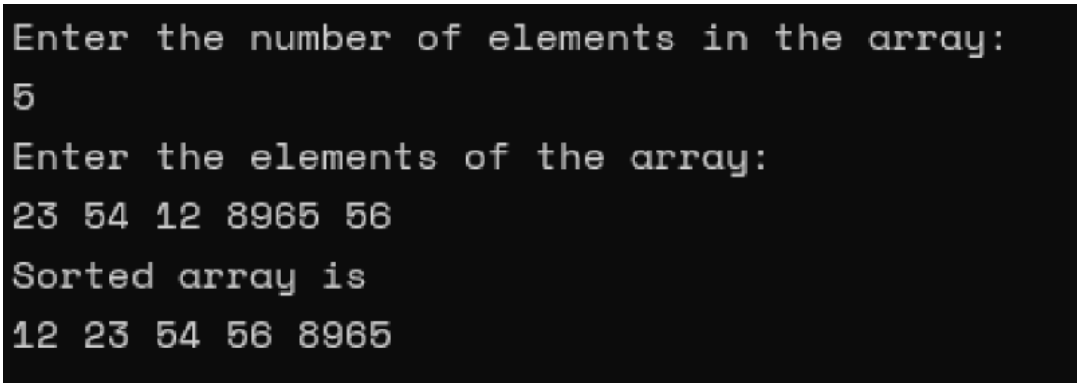
| Input size(n) | Time taken |
|---|---|
| 10000 | 0.001908 |
| 20000 | 0.003618 |
| 30000 | 0.004931 |
| 40000 | 0.005698 |
| 50000 | 0.00735 |
| 60000 | 0.008865 |
| 70000 | 0.012559 |
| 80000 | 0.012323 |
| 90000 | 0.013631 |
| 100000 | 0.015273 |

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

```
#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);
   }
}
```

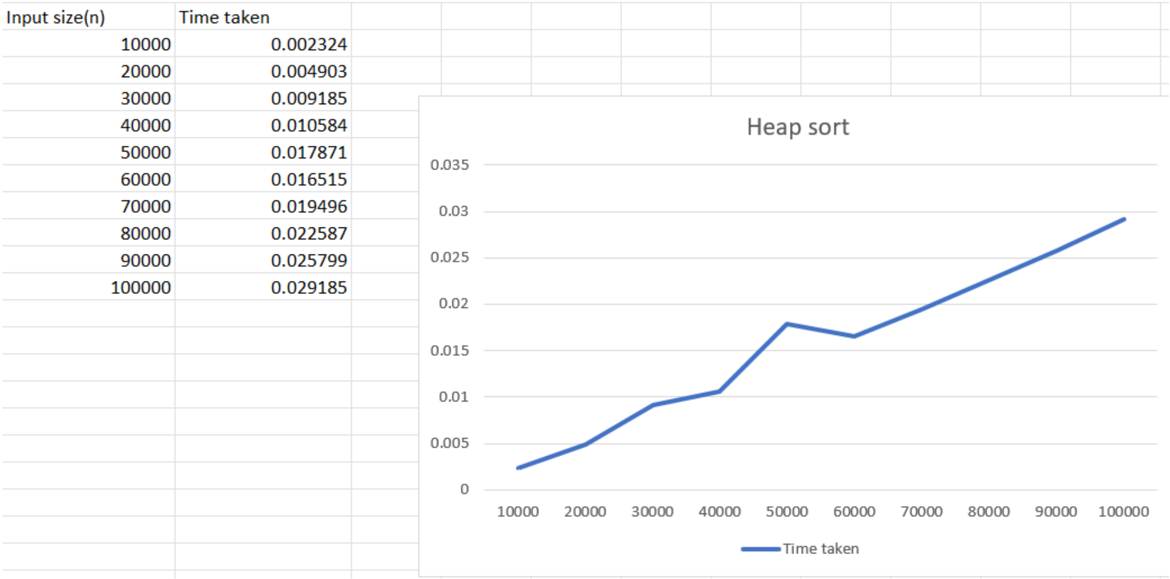
```
void printArray(int arr[], int N)
{
    for (int i = 0; i < N; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

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

    heapSort(arr, N);
    printf("Sorted array is\n");
    printArray(arr, N);
}
```

OUTPUT



```
Enter the number of elements in the array:
5
Enter the elements of the array:
23 54 12 8965 56
Sorted array is
12 23 54 56 8965
```

GRAPH

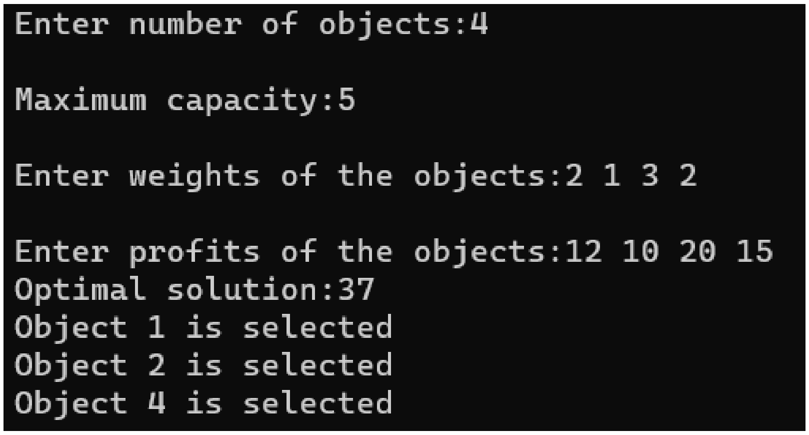


| Input size(n) | Time taken |
|---|---|
| 10000 | 0.002324 |
| 20000 | 0.004903 |
| 30000 | 0.009185 |
| 40000 | 0.010584 |
| 50000 | 0.017871 |
| 60000 | 0.016515 |
| 70000 | 0.019496 |
| 80000 | 0.022587 |
| 90000 | 0.025799 |
| 100000 | 0.029185 |

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

```
#include<stdio.h>
#include<conio.h>
int n,m,w[10],p[10],v[10][10],x[10];
void knapsack()
{
   int i,j;
   for(i=0;i<=n;i++)
   {
      for(j=0;j<=m;j++)
      {
         if(i==0||j==0)
            v[i][j]=0;
         if(w[i]>j)
            v[i][j]=v[i-1][j];
         else
            v[i][j]=(v[i-1][j]>v[i-1][j-w[i]]+p[i])?v[i-1][j]:v[i-1][j-w[i]]+p[i];
      }
   }
   object_selected();
}
void object_selected()
{
   int i,j;
   printf("Optimal solution:%d\n",v[n][m]);
   for(i=1;i<=n;i++)
      x[i]=0;
      i=n;
      j=m;
   while(i!=0 && j!=0)
   {
      if(v[i][j]!=v[i-1][j])
      {
         x[i]=1;
         j=j-w[i];
      }
      i--;
   }
   for(i=1;i<=n;i++)
   {
        if(x[i]==1)
        {
            printf("Object %d is selected\n",i);
        }
    }
}
void main()
{
    int i;
    printf("Enter number of objects:");
    scanf("%d",&n);
    printf("\nMaximum capacity:");
    scanf("%d",&m);
    printf("\nEnter weights of the objects:");
    for(i=1;i<=n;i++)
    {
        scanf("%d",&w[i]);
    }
    printf("\nEnter profits of the objects:");
    for(i=1;i<=n;i++)
    {
        scanf("%d",&p[i]);
    }
    knapsack();
}
```

OUTPUT

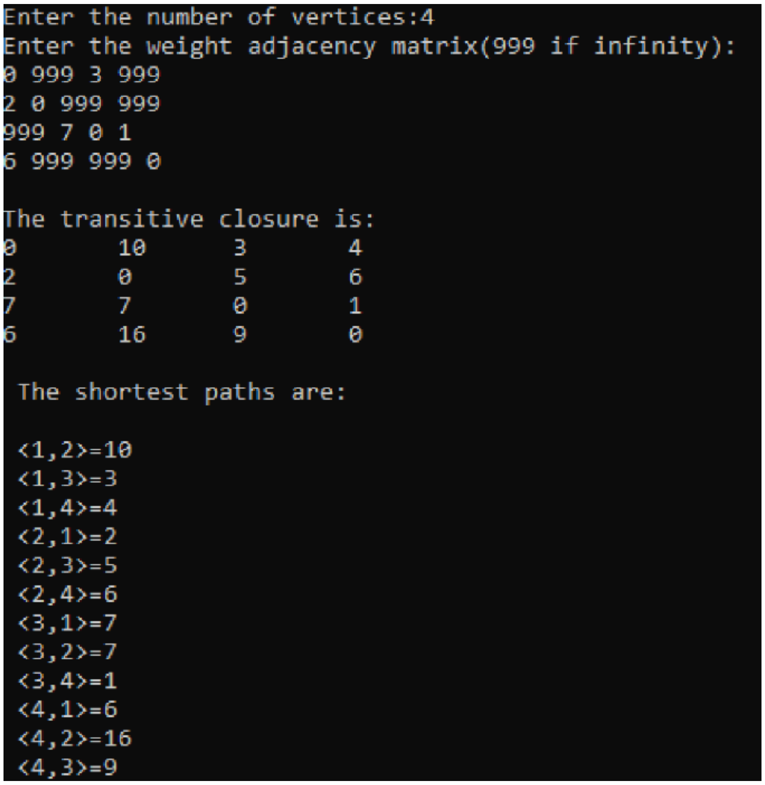


```
Enter number of objects:4

Maximum capacity:5

Enter weights of the objects:2 1 3 2

Enter profits of the objects:12 10 20 15
Optimal solution:37
Object 1 is selected
Object 2 is selected
Object 4 is selected
```

## 8. Implement All Pair Shortest paths problem using Floyd’s algorithm.

```
#include<stdio.h>
#include<conio.h>
int c[10][10],d[10][10],n;
void floyd()
{
   int i,j,k;
   for(i=1;i<=n;i++)
   {

      for(j=1;j<=n;j++)
      {
         d[i][j]=c[i][j];
      }
   }
   for(k=1;k<=n;k++)
   {
      for(i=1;i<=n;i++)
      {
         for(j=1;j<=n;j++)
         {
            d[i][j]=(d[i][j]<d[i][k]+d[k][j])?d[i][j]:(d[i][k]+d[k][j]);
         }
      }

   }
}
void main()
{
   int i,j;
   printf("Enter the number of vertices:");
   scanf("%d",&n);
   printf("Enter the weight adjacency matrix(999 if infinity):\n");
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         scanf("%d",&c[i][j]);
      }
   }
   floyd();
   printf("\nThe transitive closure is:\n");
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         printf("%d\t",d[i][j]);
      }
      printf("\n");
   }
   printf("\n The shortest paths are:\n");
         for (i=1;i<=n;i++)
           for (j=1;j<=n;j++) {
                  if(i!=j&&d[i][j]!=999)
                     printf("\n <%d,%d>=%d",i,j,d[i][j]);
         }
}
```

OUTPUT



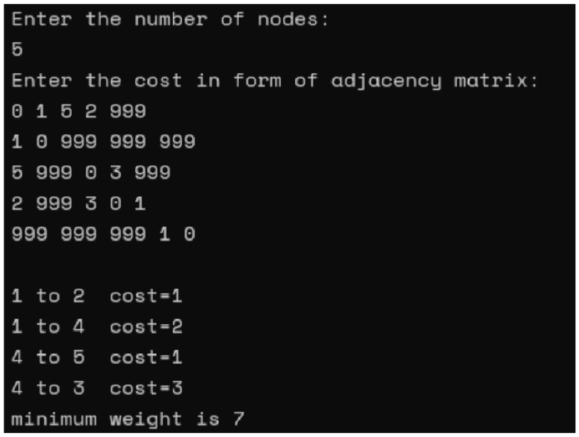
```
Enter the number of vertices:4
Enter the weight adjacency matrix(999 if infinity):
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

The transitive closure is:
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0

 The shortest paths are:

 <1,2>=10
 <1,3>=3
 <1,4>=4
 <2,1>=2
 <2,3>=5
 <2,4>=6
 <3,1>=7
 <3,2>=7
 <3,4>=1
 <4,1>=6
 <4,2>=16
 <4,3>=9
```

**9. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm.**

PRIM’S

```
#include<stdio.h>
int main()
{
    int cost[10][10],visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;
    printf("Enter the number of nodes:\n");
    scanf("%d",&n);
    printf("Enter the cost in form of adjacency matrix:\n");

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

            if(cost[i][j]==0)
              cost[i][j]=1000;
        }
    }


    visited[1]=1;
    while(no_e<n)
    {
        min=1000;

        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(cost[i][j]<min)
                {
                    if(visited[i]!=0)
                    {
                        min=cost[i][j];
                        a=i;
                        b=j;
                    }
                }
            }
        }

        if(visited[b]==0)
        {
            printf("\n%d to %d  cost=%d",a,b,min);
            min_cost=min_cost+min;
            no_e++;
        }
        visited[b]=1;

        cost[a][b]=cost[b][a]=1000;
    }
    printf("\nminimum weight is %d",min_cost);
    return 0;
}
```

OUTPUT



```
Enter the number of nodes:
5
Enter the cost in form of adjacency matrix:
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 1
999 999 999 1 0

1 to 2  cost=1
1 to 4  cost=2
4 to 5  cost=1
4 to 3  cost=3
minimum weight is 7
```

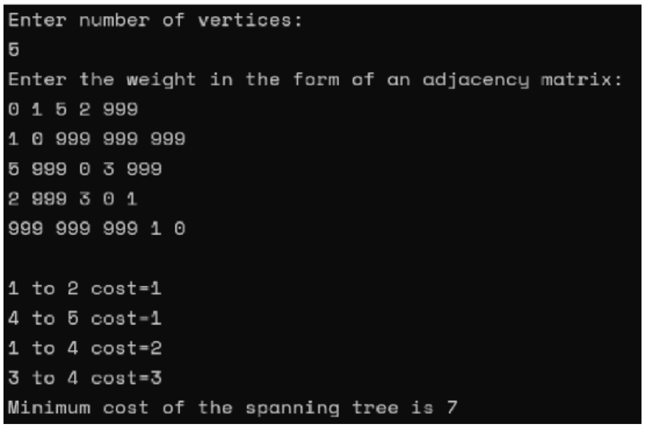
KRUSKAL’S

```
#include<stdio.h>
int parent[10]={0};
int find_parent(int);
int is_cyclic(int,int);
int main()
{
    int cost[10][10],min_cost=0,min,i,j,n,no_e=1,a,b,u,v,x;
    printf("Enter number of vertices:\n");
```

```
scanf("%d",&n);
printf("Enter the weight in the form of an adjacency matrix:\n");

for(i=1;i<=n;i++)
{
   for(j=1;j<=n;j++)
   {
      scanf("%d",&cost[i][j]);
      if(cost[i][j]==0)
        cost[i][j]=999;
   }
}


while(no_e<n)
{
   min=999;

   for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
      {
         if(cost[i][j]<min)
         {
            min=cost[i][j];
            a=u=i;
            b=v=j;
         }
      }
   }

   u=find_parent(u);
   v=find_parent(v);
   x=is_cyclic(u,v);
   if(x==1)
   {
      printf("\n%d to %d cost=%d",a,b,min);
      no_e++;
      min_cost+=min;
   }
```

```
        cost[a][b]=cost[b][a]=999;
    }
    printf("\nMinimum cost of the spanning tree is %d",min_cost);
    return 0;
}


int find_parent(int a)
{
    while(parent[a]!=0)
      a=parent[a];
    return a;
}


int is_cyclic(int a ,int b)
{
    if(a!=b)
    {
        parent[b]=a;
        return 1;
    }
    return 0;
}
```

OUTPUT



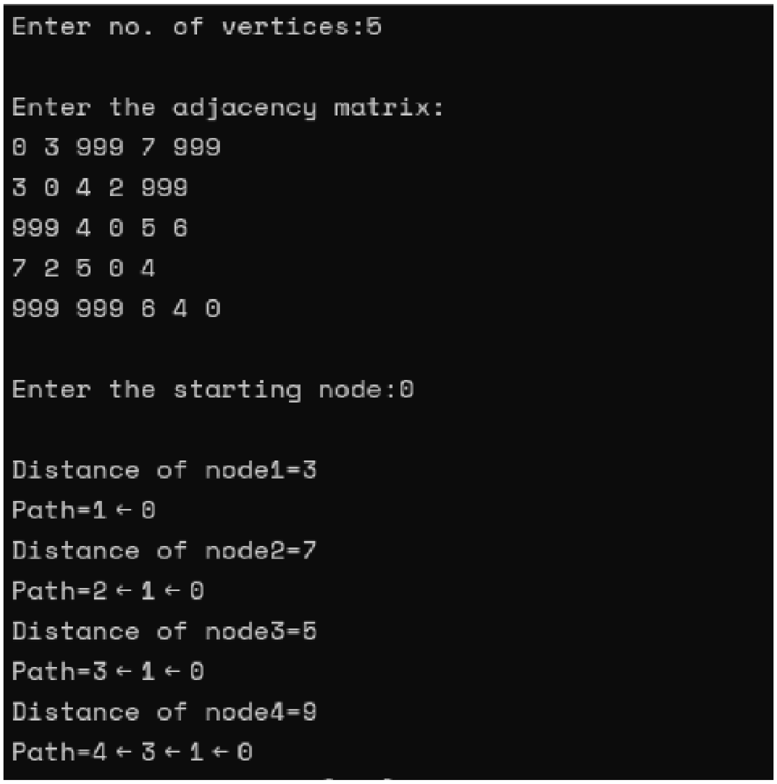
```
Enter number of vertices:
5
Enter the weight in the form of an adjacency matrix:
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 1
999 999 999 1 0

1 to 2 cost=1
4 to 5 cost=1
1 to 4 cost=2
3 to 4 cost=3
Minimum cost of the spanning tree is 7
```

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

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
    int G[MAX][MAX],i,j,n,u;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    scanf("%d",&G[i][j]);
    printf("\nEnter the starting node:");
    scanf("%d",&u);
    dijkstra(G,n,u);
    return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

    int cost[MAX][MAX],distance[MAX],pred[MAX];
    int visited[MAX],count,mindistance,nextnode,i,j;
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    if(G[i][j]==0)
    cost[i][j]=INFINITY;
    else
    cost[i][j]=G[i][j];
    for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
```

```
   }
   distance[startnode]=0;
   visited[startnode]=1;
   count=1;
   while(count<n-1)
   {
      mindistance=INFINITY;
      for(i=0;i<n;i++)
      if(distance[i]<mindistance&&!visited[i])
      {
         mindistance=distance[i];
         nextnode=i;
      }
      visited[nextnode]=1;
      for(i=0;i<n;i++)
         if(!visited[i])
         if(mindistance+cost[nextnode][i]<distance[i])
         {
            distance[i]=mindistance+cost[nextnode][i];
            pred[i]=nextnode;
         }
         count++;
      }
      for(i=0;i<n;i++)
      if(i!=startnode)
      {
         printf("\nDistance of node%d=%d",i,distance[i]);
         printf("\nPath=%d",i);
         j=i;
         do
         {
            j=pred[j];
            printf("<-%d",j);
      }
      while(j!=startnode);
   }
}
```

OUTPUT



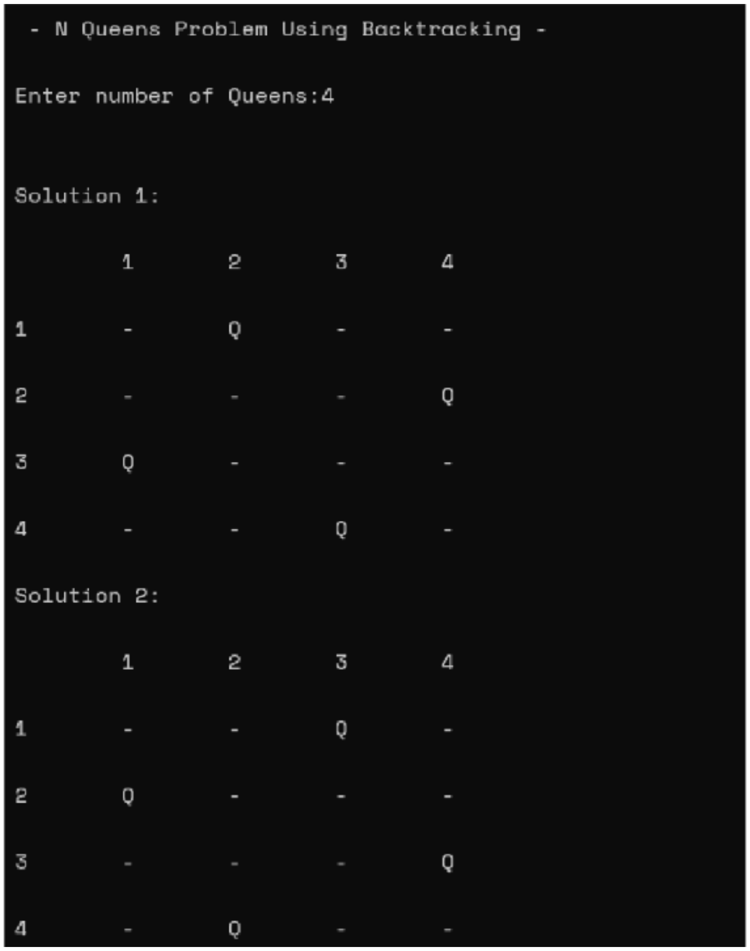
```
Enter no. of vertices:5

Enter the adjacency matrix:
0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 6 4 0

Enter the starting node:0

Distance of node1=3
Path=1←0
Distance of node2=7
Path=2←1←0
Distance of node3=5
Path=3←1←0
Distance of node4=9
Path=4←3←1←0
```

## 11.Implement “N-Queens Problem” using Backtracking.

```
#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;
}

// function for printing the solution
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) // for nxn board
    {
      if (board[i] == j)
        printf("\tQ"); // queen at i,j position
      else
        printf("\t-"); // empty slot
    }
  }
}
```

```
/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row, int column)
{
  int i;
  for (i = 1; i <= row - 1; ++i)
  {
    // checking column and digonal conflicts
    if (board[i] == column)
      return 0;
    else if (abs(board[i] - column) == abs(i - row))
      return 0;
  }

  return 1; // no conflicts
}

// function to check for proper positioning of queen
void queen(int row, int n)
{
  int column;
  for (column = 1; column <= n; ++column)
  {
    if (place(row, column))
    {
      board[row] = column; // no conflicts so place queen
      if (row == n)        // dead end
        print(n);          // printing the board configuration
      else                 // try queen with next position
        queen(row + 1, n);
    }
  }
}
```

OUTPUT



```
 - N Queens Problem Using Backtracking -

Enter number of Queens:4


Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```