Aim :

N-Queen is the problem of placing ‘N’ chess queens on an NxN chessboard. Design a solution for this problem so that no two queens attack each other.

Note : A queen can attack when an opponent is on the same row/column/diagonal.

Description :

We have been given an NxN chessboard, our aim here is to place it on the chessboard so that no two queens attack each other. Here, we follow the strategy of backtracking since we have the possibility of getting more than one solution, and if we have reached the dead end we go the previous state , place the queen in the proper order and then come back.

Algorithm :

```
Algorithm SolveNQUtil(board,col)
//Here board is NxN array, col is the column in which queen is to be placed.
{
        if(col>=N) then write(solution) return true;
        res=false;
        for i:=0 to N do{
                if(isSafe(board,i,col)) then do{
                        board[i][col]:=1
                        res=solveNQUtil(board,col+1)||res;
                        board[i][col]=0
                }
        }
 return res;
}
Algorithm isSafe(board,row,col)
{
        for i:=0 to col do
                if(board[row][i]) then return false;
        for i:=row,j:=col; i>=0&&j>=0, i--,j—do
                if(board[i][j]) then return false;
        for i:=row,j:=col; j>=0&&i<N, i++,j—do
                if(board[i][j]) then return false;
return true;
}
```

Code :

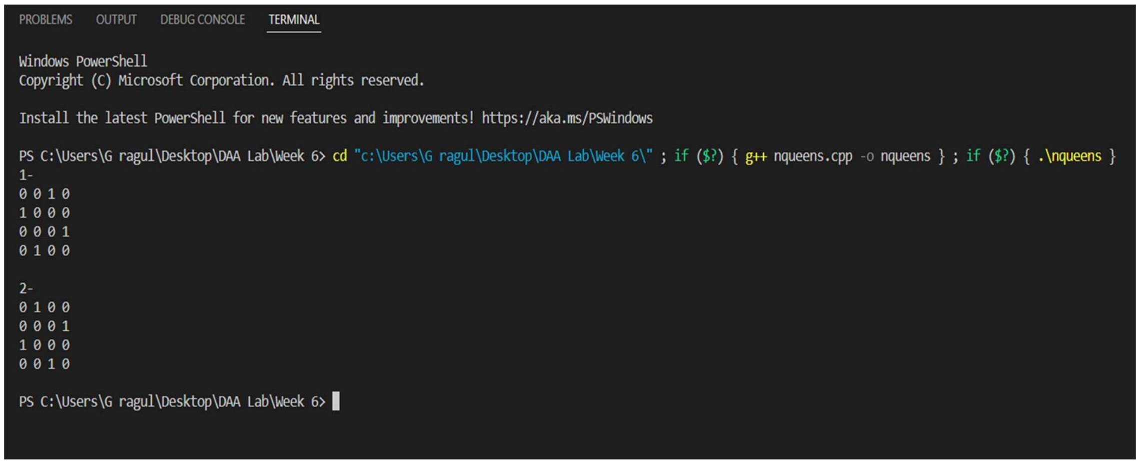
```
#include<bits/stdc++.h>
using namespace std;
#define N 4

void printSolution(int board[N][N])
{
   static int k = 1;
   cout<<k<<"-\n";
   k++;
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
         cout<<board[i][j]<<" ";
         cout<<"\n";
   }

   cout<<"\n";
}

bool isSafe(int board[N][N], int row, int col)
{
   int i, j;


   for (i = 0; i < col; i++)
      if (board[row][i])
         return false;
```

```
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;



    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}



bool solveNQUtil(int board[N][N], int col)
{

    if (col >= N)
    {
        printSolution(board);
        return true;
    }



    bool res = false;
    for (int i = 0; i < N; i++)
    {

        if ( isSafe(board, i, col) )
        {
            board[i][col] = 1;



            res = solveNQUtil(board, col + 1) || res;



            board[i][col] = 0; // BACKTRACK
        }
    }



    return res;
}



void solveNQ()
{
    int board[N][N];
    memset(board, 0, sizeof(board));

    if (solveNQUtil(board, 0) == false)
    {
        printf("Solution does not exist");
        return ;
    }

    return ;
}

// driver program to test above function
int main()
```

```
{
   solveNQ();
   return 0;
}
```

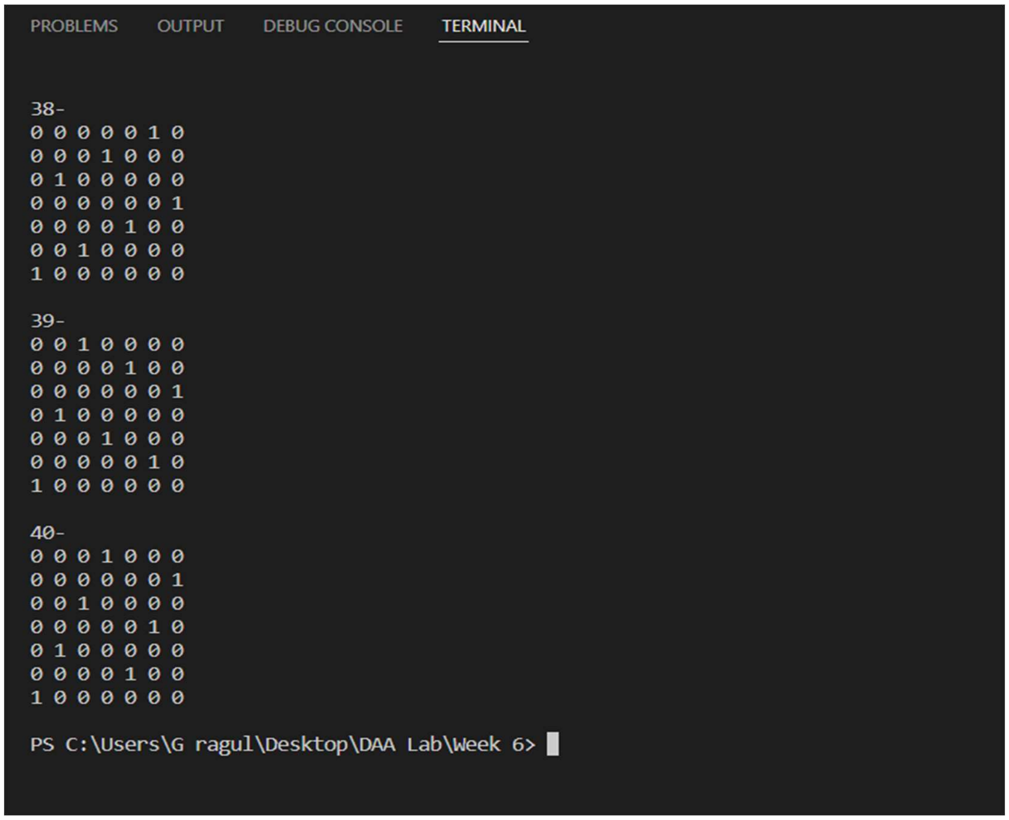
Result Analysis :

For N=4



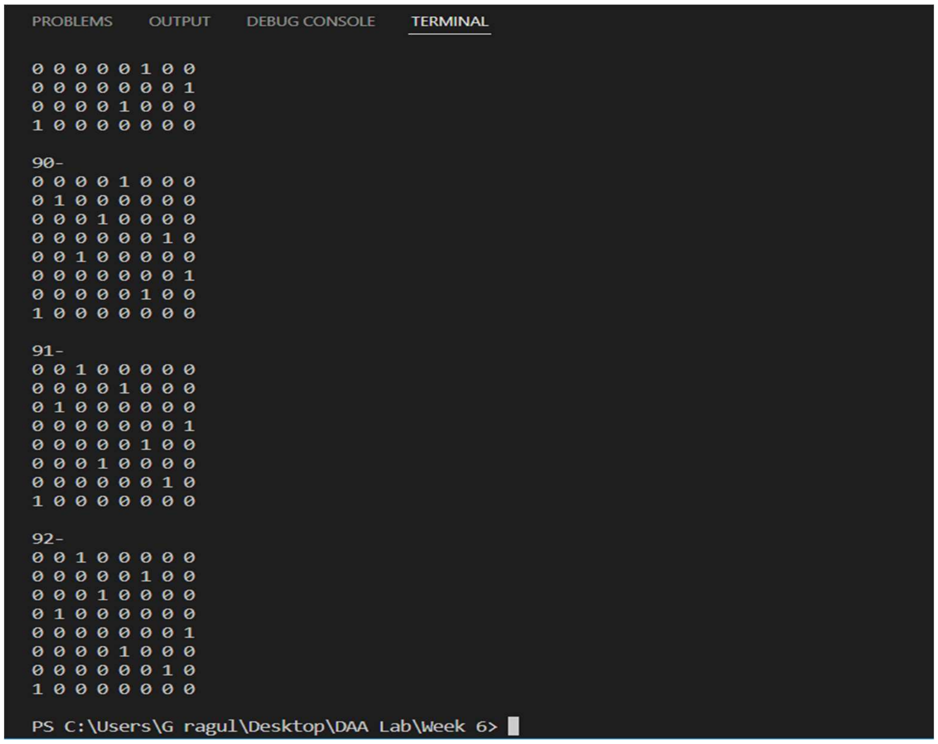
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\G ragul\Desktop\DAA Lab\Week 6> cd "c:\Users\G ragul\Desktop\DAA Lab\Week 6\" ; if ($?) { g++ nqueens.cpp -o nqueens } ; if ($?) { .\nqueens }
1-
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

2-
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

PS C:\Users\G ragul\Desktop\DAA Lab\Week 6> 
```

For N=7;



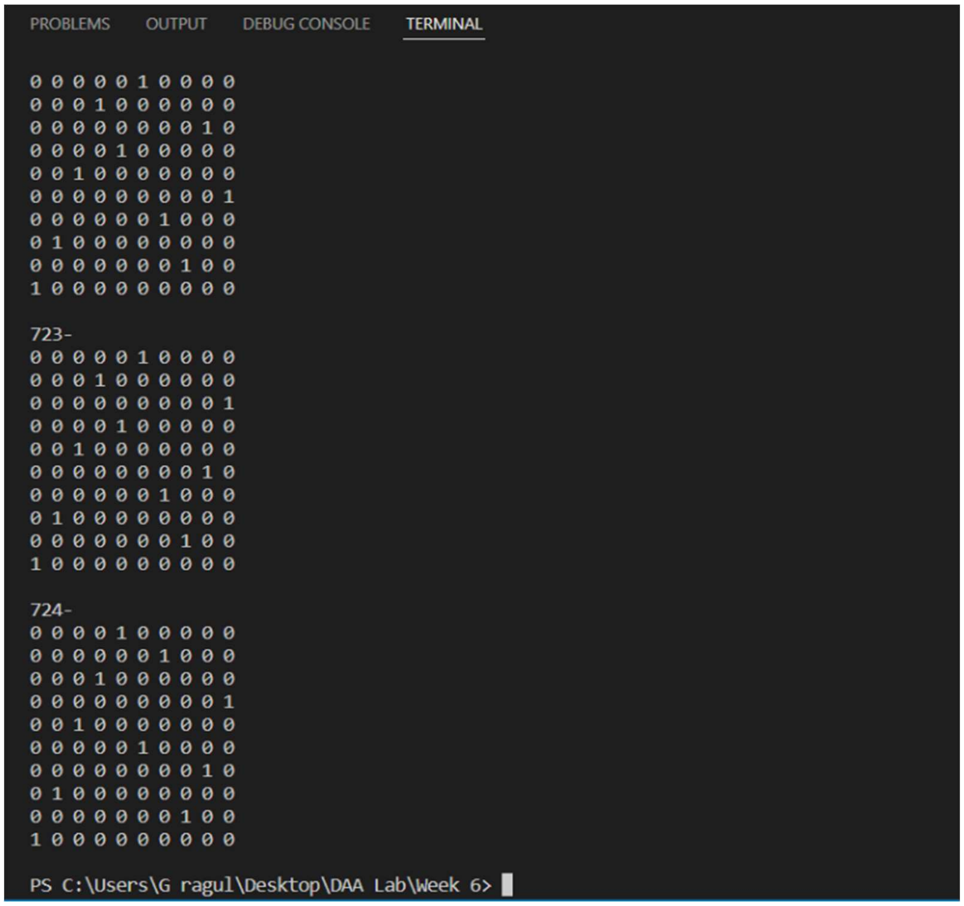
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

38-
0 0 0 0 0 1 0
0 0 0 1 0 0 0
0 1 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 1 0 0
0 0 1 0 0 0 0
1 0 0 0 0 0 0

39-
0 0 1 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 1
0 1 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 1 0
1 0 0 0 0 0 0

40-
0 0 0 1 0 0 0
0 0 0 0 0 0 1
0 0 1 0 0 0 0
0 0 0 0 0 1 0
0 1 0 0 0 0 0
0 0 0 0 1 0 0
1 0 0 0 0 0 0

PS C:\Users\G ragul\Desktop\DAA Lab\Week 6> 
```

For N=8;


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0

90-
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0

91-
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0

92-
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0

PS C:\Users\G ragul\Desktop\DAA Lab\Week 6>
```

For N=10;



```
0 0 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0

723-
0 0 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0

724-
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0

PS C:\Users\G ragul\Desktop\DAA Lab\Week 6>
```

Time Complexity :

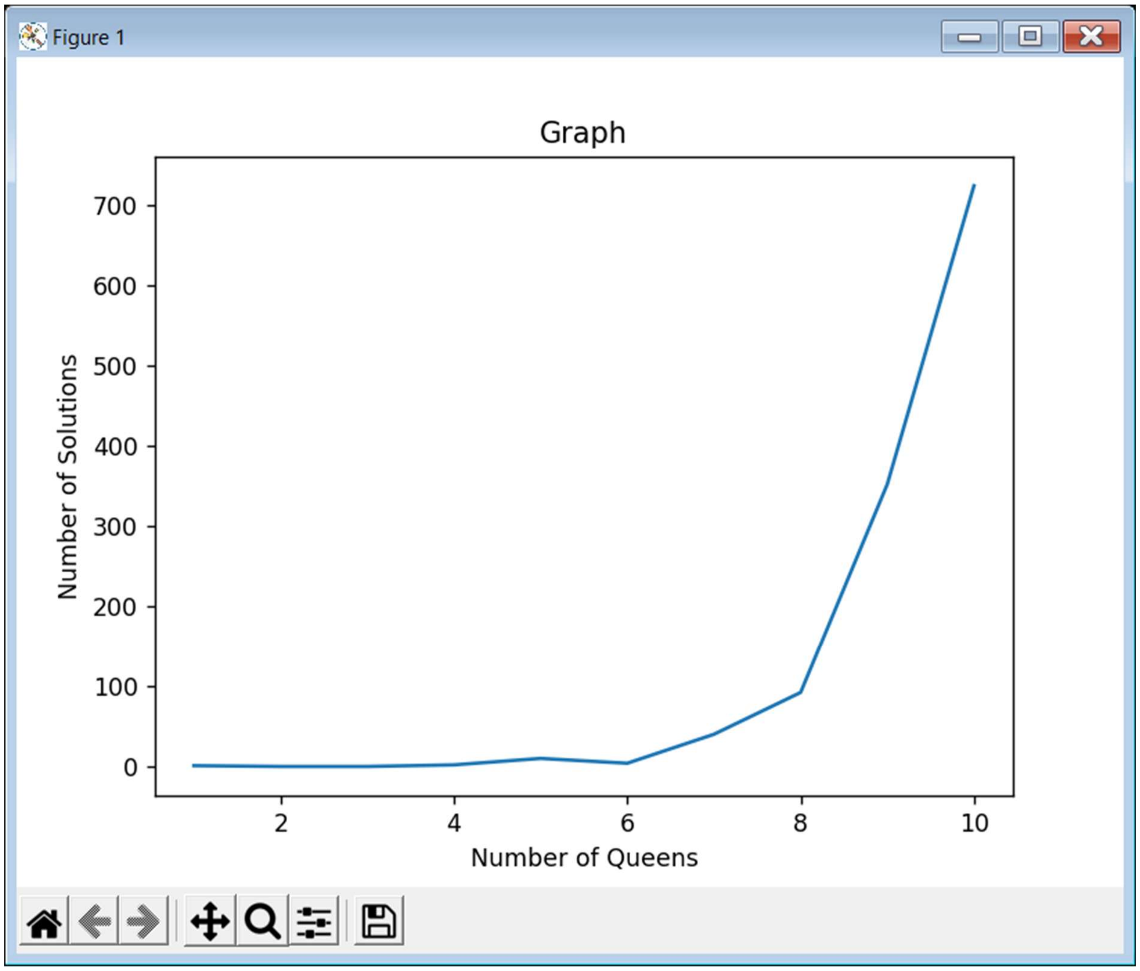
The recurrence relation obtained is T(n)=n*T(n+1)

on solving we get the time complexity as O(n!).

Space Complexity :

Since the chessboard occupies NxN array , space complexity is O(n^2)

Graph :



Conclusion :

As the number of queens increases, the time complexity of the code increases, space remains constant and the number of solution increases.

References :

https://tutorialspoint.dev/algorithm/backtracking-algorithms/printing-solutions-n-queen-problem