NAME:	Shubham Vishwakarma
UID No.	2021700071
BRANCH:	S.Y CSE-DS
BATCH:	D
SUBJECT	Design and Analysis of Algorithms
EXPERIMENT No.	7
Date of Performance	10/04/2023
Date of Submission	14/04/2023

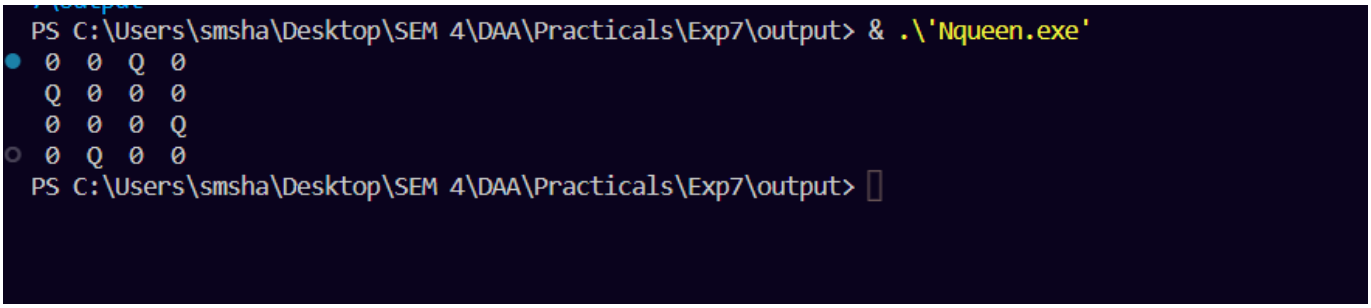
AIM:	**Backtracking (To implement N Queens problem using backtracking.)**
Program 1	
PROBLEM STATEMENT :	Implement the N queen problem for 4x4 chess board.
ALGORITHM/ THEORY:	**Step 1 -** Place the queen row-wise, starting from the left-most cell. **Step 2 -** If all queens are placed then return true and print the solution matrix. **Step 3 -** Else try all columns in the current row. • **Condition 1 -** Check if the queen can be placed safely in this column then mark the current cell [Row, Column] in the solution matrix as 1 and try to check the rest of the problem recursively by placing the queen here leads to a solution or not. • **Condition 2 -** If placing the queen [Row, Column] can lead to the solution return true and print the solution for each queen's position. • **Condition 3 -** If placing the queen cannot lead to the solution then unmark this [row, column] in the solution matrix as 0, BACKTRACK, and go back to condition 1 to try other rows. **Step 4 -** If all the rows have been tried and nothing worked, return false to trigger backtracking.

PROGRAM:

```c
#define N 4
#include <stdbool.h>
#include <stdio.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
        {
            //printf(" %d ", board[i][j]);
            if(board[i][j] == 1)
            {
                printf(" Q ");
            }
            else
            printf(" %d ", board[i][j]);

        }
        printf("\n");
    }
}

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

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

/* A recursive utility function to solve N
Queen problem */
```

```
bool solveNQUtil(int board[N][N], int col)
{
    if (col >= N)
        return true;


    for (int i = 0; i < N; i++) {
        /* Check if the queen can be placed on
        board[i][col] */
        if (isSafe(board, i, col)) {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            if (solveNQUtil(board, col + 1))
                return true;

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

    return false;
}


bool solveNQ()
{
    int board[N][N] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };

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

    printSolution(board);
    return true;
}

// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

RESULT:



CONCLUSION :	• N Queen problem is a classical puzzle that beautifully develops the concept of *Backtracking*. • The time complexity of the brute force backtracking algorithm is O(N×N!)O(N×N!). However, using *bitmasking* the time complexity can be optimized to O(N!)O(N!). • The space complexity irrespective of the approach is O(N2)O(N2) because we need to print a 2-dimensional array as the answer.