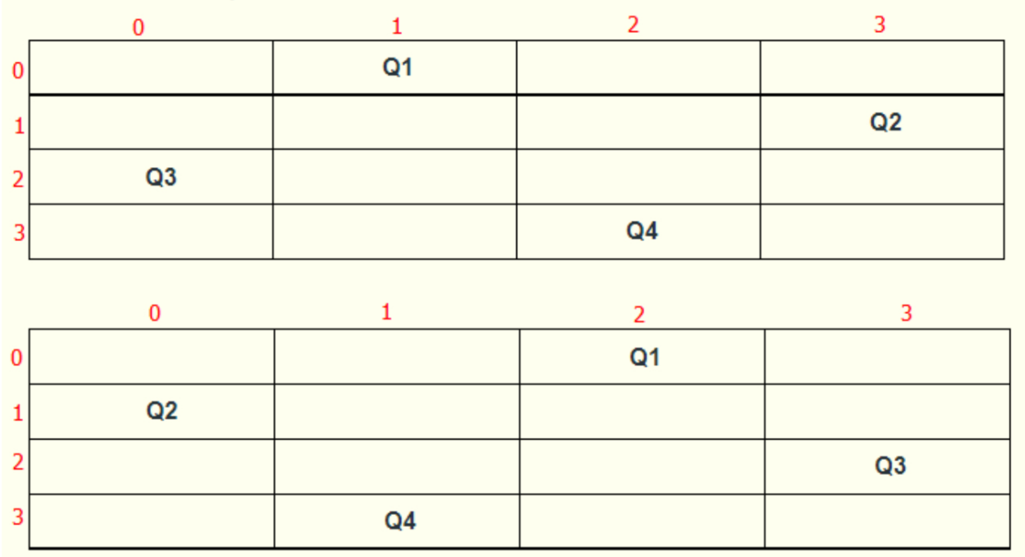
S.No: 12	Exp. Name: ***N Queens Problem***	**Date:2025-09-10**

Aim:

N-**queens** problem is the problem of placing **N** queens on an **N x N** square chessboard (grid) such that no two queens attack each other.
Given an integer **N** return the count of all distinct solutions to the **N queens problem**.
Each solution contains a distinct board configuration of the N queens.
You can see two possibilities for N = 4:



You can see in 4 x 4 chessboard their are 2 possibilities where Queens are placed in safe states

Sample test case 1:
Input:
4 // First line of input is the size of the Chess board (square grid
Output:
2 // No of ways possible to place queens in safe state
Sample test case 1:
Input:
2 // First line of input is the size of the Chess board (square grid
Output:
0 // No of ways possible to place queens in safe state

You just have to complete the function **nQueen() with parameter passed n represents the n x n size of the chess board and function returns the total no of solutions feasible to place queens in safe state, and if the their solution is no solution exists then return 0.**

Constraints to be followed:
2 ≤ n ≤ 10

Instructions: To run your custom test cases strictly follow the input and output layout as mentioned in the visible sample test cases.

Source Code:

nQueenProblem.c

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

#define MAX 10
```

```c
int solutionCount = 0;

// Check if we can place queen at board[row][col]
bool isSafe(int board[MAX][MAX], int row, int col, int n) {
    int i, j;

    // Check column above
    for (i = 0; i < row; i++)
        if (board[i][col])
            return false;

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

    // Check upper right diagonal
    for (i = row, j = col; i >= 0 && j < n; i--, j++)
        if (board[i][j])
            return false;

    return true;
}

void solveNQ(int board[MAX][MAX], int row, int n) {
    if (row == n) {
        solutionCount++;
        return;
    }

    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col, n)) {
            board[row][col] = 1;
            solveNQ(board, row + 1, n);
            board[row][col] = 0; // backtrack
        }
    }
}

int nQueen(int n) {
    int board[MAX][MAX] = {0};
    solutionCount = 0;
    solveNQ(board, 0, n);
    return solutionCount;
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", nQueen(n));
    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
2
`0`

Test Case - 2
User Output
3
`0`