UNIVERSITY OF ASIA PACIFIC

Department of CSE

ASSIGNMENT

Course Code:CSE_208

Course Title:Data Structures and Algorithm 2 – Lab

Submitted by,

Name:Md. Mashrur Rahman Masfi

ID:23101182

Section:D1

Submitted to,

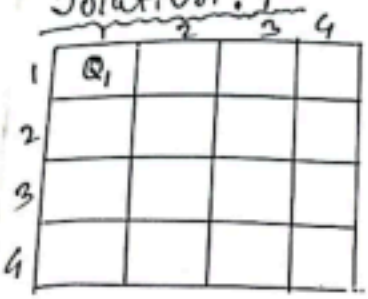
Fabliha Haque

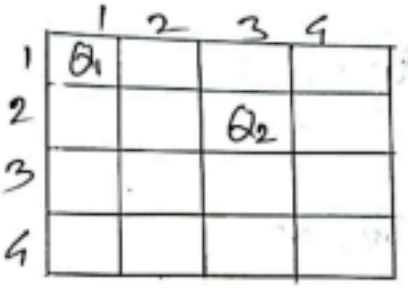
Lecturer,

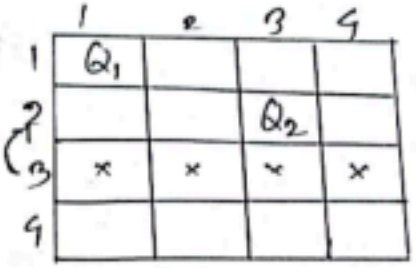
UAP

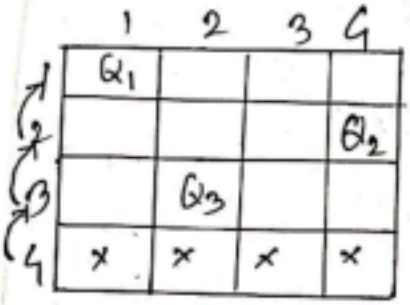
Simulation of N queens Problem

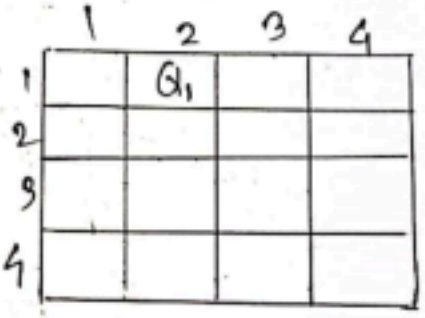
Solution: 1

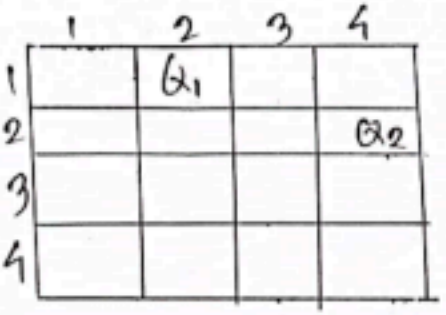


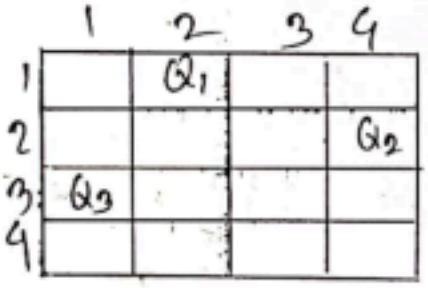


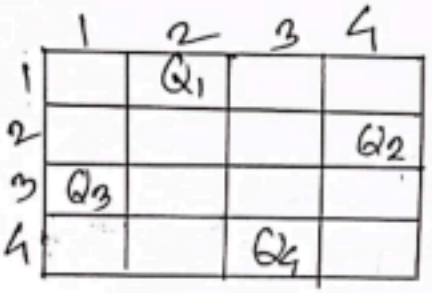




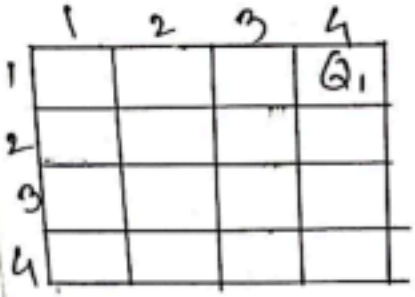


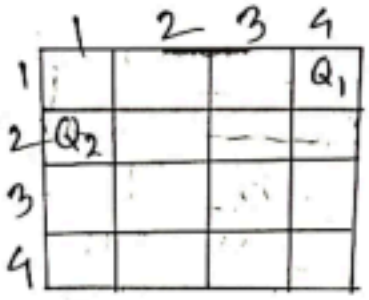


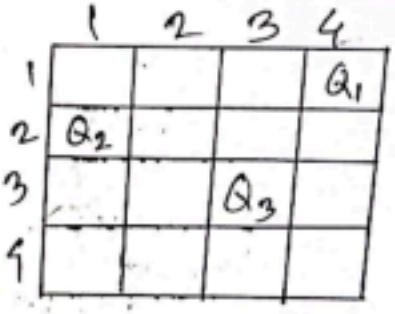


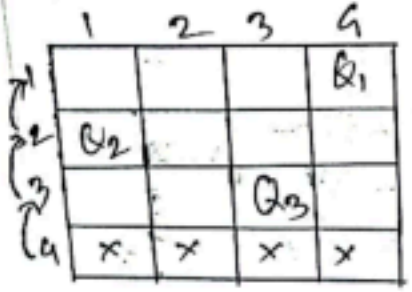


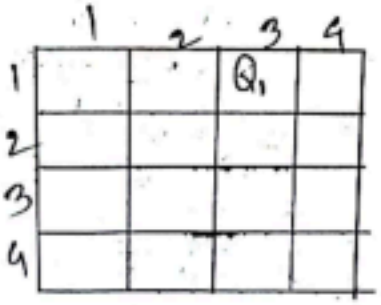
Solution: 2

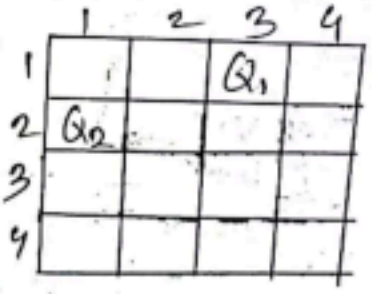


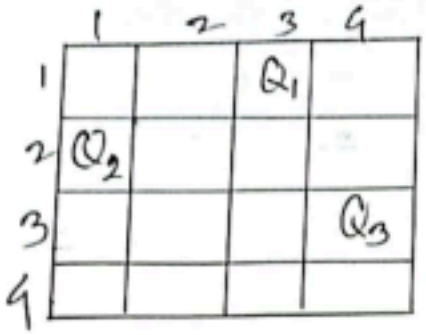


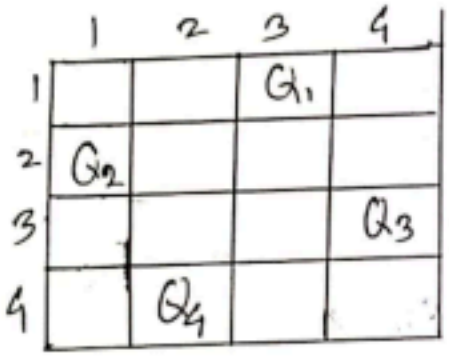












C Code for N-queen problem

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

#define MAX 20  // Maximum board size (20x20)

// Function to check if placing a queen at board[row][col] is
safe
bool isSafe(char board[MAX][MAX], int row, int col, int n) {
   // Check the same column above
   for (int i = 0; i < row; i++) {
      if (board[i][col] == 'Q') {
         return false;
      }
   }

   // Check the upper-left diagonal
   for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
      if (board[i][j] == 'Q') {
         return false;
      }
   }
```

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

    // If no conflicts, it's safe to place a queen
    return true;
}

// Function to print the current board configuration
void printBoard(char board[MAX][MAX], int n) {
    for (int i = 0; i < n; i++) {       // Loop through each row
        for (int j = 0; j < n; j++) {    // Loop through each column
            printf("%c", board[i][j]);   // Print either 'Q' or '.'
        }
        printf("\n"); // Move to next line after each row
    }
    printf("\n");    // Extra line after each complete solution
}

// Recursive function to solve the N-Queens problem using
backtracking
void solve(int row, int n, char board[MAX][MAX], int* count) {
```

```
    // If all queens are placed successfully
    if (row == n) {
        printBoard(board, n); // Print the current board
        (*count)++;          // Increment the solution count
        return;
    }

    // Try placing a queen in each column of the current row
    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col, n)) {
            board[row][col] = 'Q';          // Place queen
            solve(row + 1, n, board, count); // Recur for next row
            board[row][col] = '.';          // Backtrack and remove
queen
        }
    }
}

int main() {
    int n;
    printf("Input board size (n): ");
    scanf("%d", &n); // User input for board size

    char board[MAX][MAX]; // 2D array to represent the
chessboard
```

```
    // Initialize the board with all empty cells ('.')
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            board[i][j] = '.';
        }
    }

    int solutionCount = 0; // Variable to store total number of
solutions

    solve(0, n, board, &solutionCount); // Start solving from the
first row

    printf("Total Solutions: %d\n", solutionCount);

    return 0;
}
```