Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai) Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Master of Computer Application

Experiment	8
Aim	To implement Backtracking algorithm (8 queens problem)
Objective	1) Understand the problem of placing 8 queens on an 8x8 chessboard such that no two queens threaten each other (8 queens problem). 2) Understand the Time complexity of N queen problem
Name	Durgesh Dilip Mandge
UCID	2023510032
Class	FYMCA
Batch	B
Date of Submission	01/04/24

Algorithm and Explanation of the technique Used	void placeQueens(vector<vector<int>>& board, int row, int col)
	{ if (row == N) { printSolution(board);
	return;
	}
	for (int c = 0; c < N; ++c) { if
	(isSafe(board, row, c)) {
	board[row][c] = 1;
	placeQueens(board, row + 1, c); board[row][c]
	= 0;
	}
	}
	}
	
	void solveNQueens() {
	vector<vector<int>> board(N, vector<int>(N, 0)); placeQueens(board,
	0, 0);
	}
	
	The time complexity of the backtracking solution for the 8 queens problem is O(N!).

Program Code

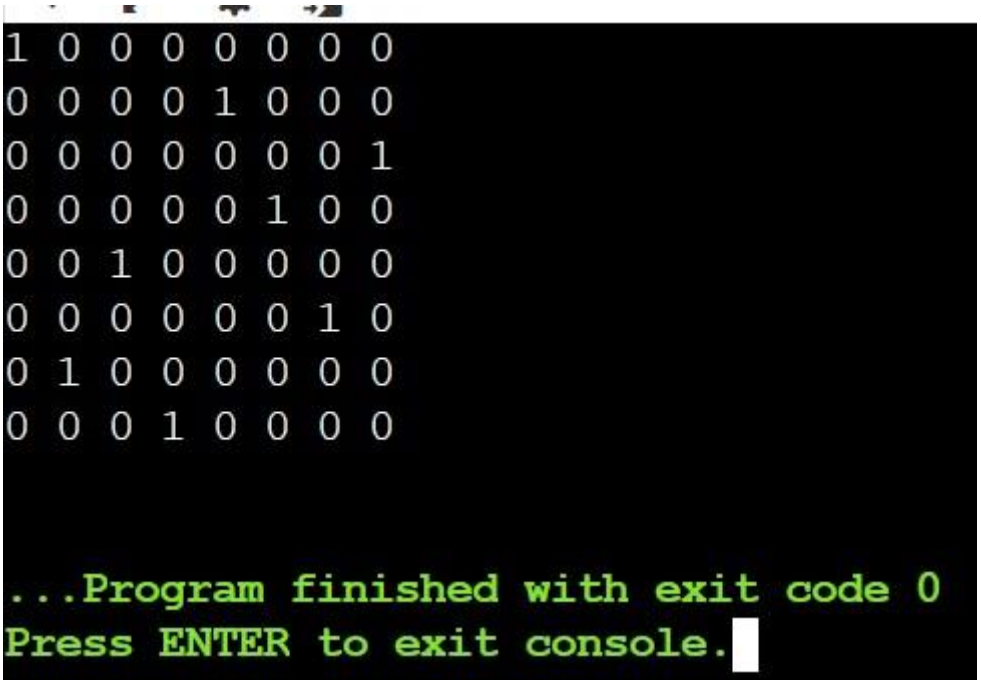
```
public class NQueens {

    static final int N = 8;
    static boolean isSafe(int[][] board, int
row, int col) {
        for (int i = 0; i < row; ++i)
            if (board[i][col] == 1)
                return false;
        for (int i = row, j = col; i >= 0 && j
>= 0; --i, --j)
            if (board[i][j] == 1)
                return false;
        for (int i = row, j = col; i >= 0 && j <
N; --i, ++j)
            if (board[i][j] == 1)
                return false;
        return true;
    }

    static boolean solveNQueens(int[][] board,
int row) {
        if (row == N)
            return true;
        for (int col = 0; col < N; ++col) {
            if (isSafe(board, row, col)) {
                board[row][col] = 1;
                if (solveNQueens(board, row +
1))
                    return true;
                board[row][col] = 0;
            }
        }
        return false;
    }

    static void printSolution(int[][] board) {
        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < N; ++j)
                System.out.print(board[i][j] + "
");
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] board = new int[N][N];
        if (solveNQueens(board, 0))
            printSolution(board);
        else
            System.out.println("Solution does
not exist.");
    }
```

	}
Output	
Justification of the complexity calculated	The time complexity of the backtracking solution for the 8 queens problem is O(N!). In the worst-case scenario, the backtracking algorithm explores all possible permutations of placing 8 queens on an 8x8 chessboard, which is N! (8! in this case). This is because for the first row, you have 8 choices, for the second row you have 7 choices (one queen has been placed in the first row), for the third row you have 6 choices, and so on, resulting in a total of 8! = 40320 possible permutations to check. Therefore, the time complexity of the backtracking solution for the 8 queens problem is O(N!), where N is the size of the chessboard.

Conclusion	The backtracking algorithm for the 8 queens problem offers significant advantages in solving combinatorial optimization challenges, particularly in scenarios where exhaustive search is feasible and efficient. By systematically exploring the solution space and leveraging constraints to prune branches, the algorithm efficiently identifies valid arrangements of queens on the chessboard, ensuring that no two queens threaten each other. This approach finds application in various fields, such as puzzle-solving, constraint satisfaction, and combinatorial optimization tasks, where it enables the rapid identification of solutions while minimizing computational complexity.