**Batch: B3 Roll No.: 121**

**Experiment No. \_\_\_8\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of N-Queen Problem using Backtracking Algorithm** |

**Objective:** To learn the Backtracking strategy of problem solving for 8-Queens problem

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajasekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algorithms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. [**http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**](http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf)
5. [**http://www.slideshare.net/Tech\_MX/8-queens-problem-using-back-tracking**](http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking)
6. [**http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**](http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html)
7. [**http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**](http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/)
8. [**http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**](http://www.hbmeyer.de/backtrack/achtdamen/eight.htm)
9. **https://github.com/rayenebech/N-Queens/blob/master/Time%20Complexity%20Analysis.md**
10. **https://www.interviewbit.com/blog/8-queens-problem/#:~:text=A%20simple%20bruteforce%20solution%20would,))%2C%20which%20is%20too%20high.**
11. **https://www.codingninjas.com/codestudio/library/n-queen**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

The **N-Queens puzzle** is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem-solving Vs other methods of problem solving, 8- Queens problem and its applications.

**Algorithm N Queens Problem: -**

void NQueens(int k, int n)

// Using backtracking, this procedure prints all possible placements of n queens on an n X n chessboard so that they are nonattacking.

{ for (int i=1; i<=n; i++)

{

if (Place(k, i))

{

x[k] = i;

if (k==n)

for (int j=1;j<=n;j++) Print x[j] ;

else NQueens(k+1, n);

}

}

}

Boolean Place(int k, int i)

// Returns true if a queen can be placed in kth row and ith column. Otherwise it returns false.

// x[] is a global array whose first (k-1) values have been set. abs(r) returns absolute value of r.

{

for (int j=1; j < k; j++)

if ((x[j] == i) // Two in the same column

|| (abs(x[j]-i) == abs(j-k))) // or in the same diagonal

return(false);

return(true);

}

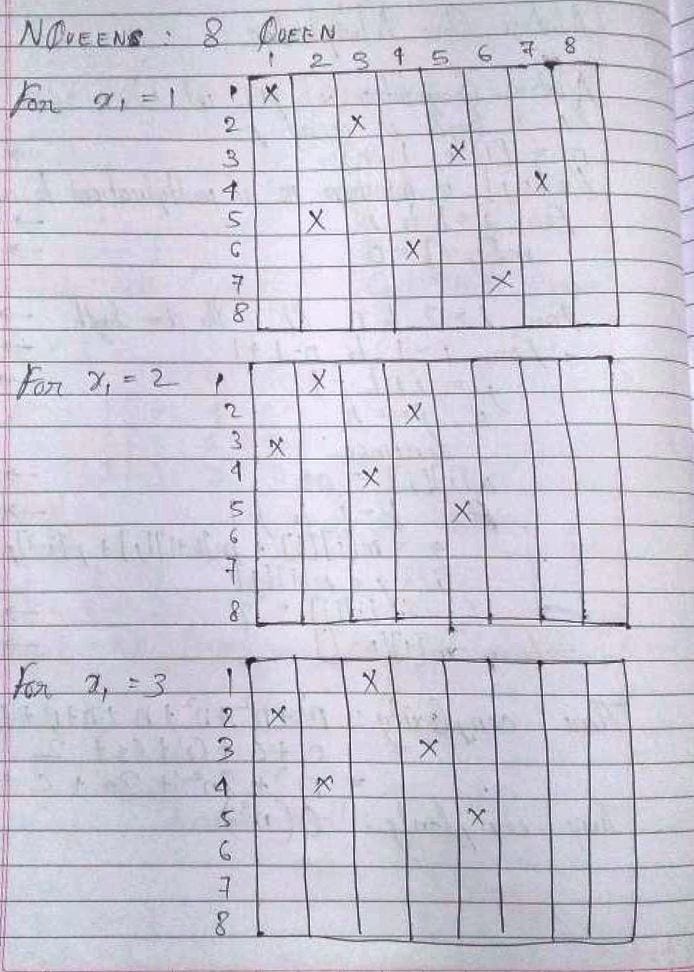
**Example 8-Queens Problem:**

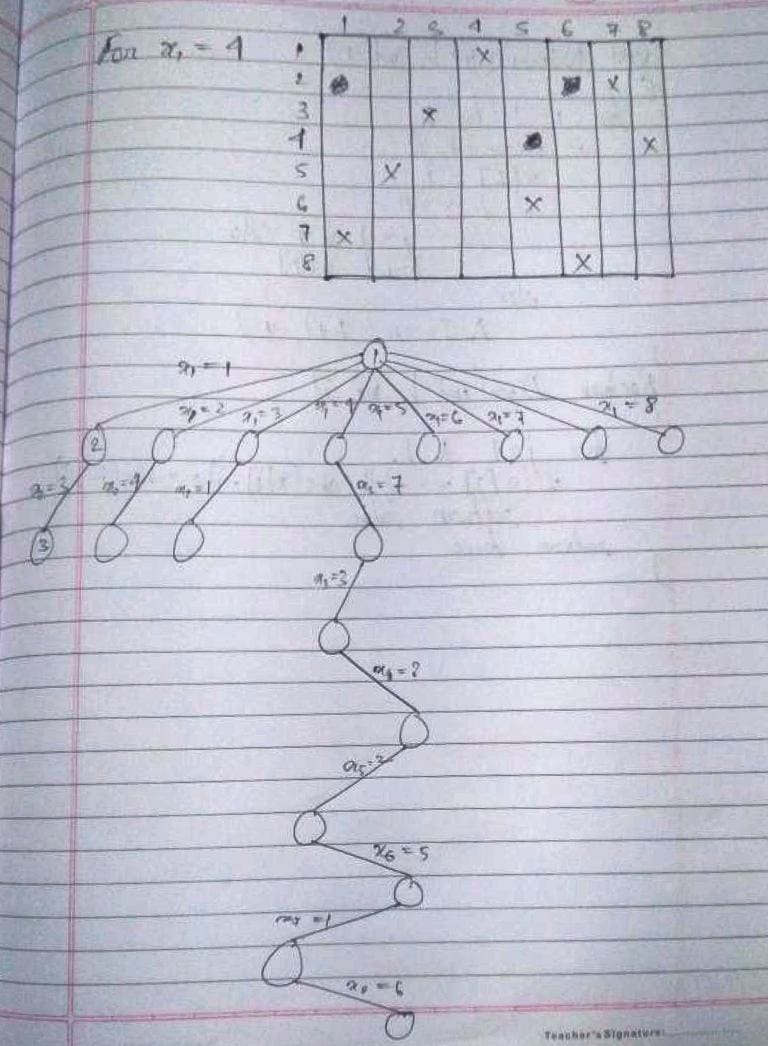
The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other i.e. no two queens share the same row, column, or diagonal.

**Solution Using Backtracking Approach:**

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

**State Space tree for N-Queens (Solution):**

****

****

**Implementation (Code):**

import java.util.\*;

class EXPT\_08\_NQueens\_2 {

    static int counter = 0;

    private static boolean position(int mat[][], int row, int col) {

        // false if two queens are in the same column

        for (int i = 0; i < row; i++) {

            if (mat[i][col] == 1) {

                return false;

            }

        }

        // false if two queens share the same `/` diagonal

        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {

            if (mat[i][j] == 1) {

                return false;

            }

        }

        // false if two queens share the same `\` diagonal

        for (int i = row, j = col; i >= 0 && j < mat.length; i--, j++) {

            if (mat[i][j] == 1) {

                return false;

            }

        }

        // true if none of the above conditions occur

        return true;

    }

    private static void print(int mat[][]) {

        int i = 1;

        for (int row = 0; row < mat.length; row++) {

            System.out.print("\n");

            if (row % 8 == 0) {

                counter++;

                System.out.print("\nSolution " + counter + " :\n");

            }

            for (int col = 0; col < mat[i].length; col++) {

                System.out.print(mat[row][col] + "\t");

            }

            System.out.print("\n");

        }

    }

    private static void place\_nqueens(int mat[][], int row) {

        // if n queens are placed successfully, the solution is printed

        if (row == mat.length) {

            print(mat);

            return;

        }

        // placing queens in the chessboard

        for (int i = 0; i < mat.length; i++) {

            // if no two queens threaten each other

            if (position(mat, row, i)) {

                // place the queen on the current square

                mat[row][i] = 1;

                // recur for the next row

                place\_nqueens(mat, row + 1);

                // backtrack and remove the queen from the current square

                mat[row][i] = 0;

            }

        }

    }

    public static void main(String[] args) {

        // `N × N` chessboard

        Scanner in = new Scanner(System.in);

        System.out.print("\nEnter value of N: ");

        int N = in.nextInt();

        // `mat[][]` keeps track of the position of queens in

        // the current configuration

        int[][] mat = new int[N][N];

        // initialize `mat[][]` by `0`

        for (int i = 0; i < N; i++) {

            // Arrays.fill(mat[i], '-');

            for (int j = 0; j < N; j++) {

                mat[i][j] = 0;

            }

        }

        long start = System.currentTimeMillis();

        place\_nqueens(mat, 0);

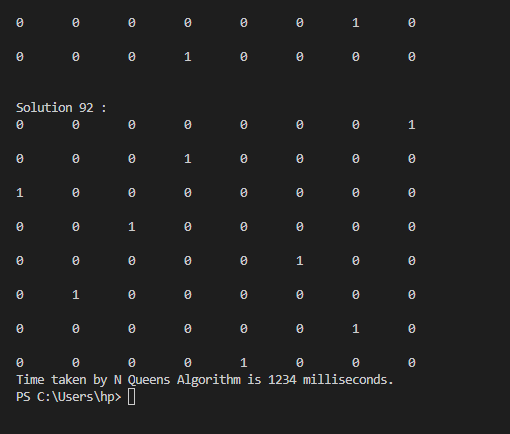
        long end = System.currentTimeMillis();

        System.out.println("Time taken by N Queens Algorithm is " + (end - start) + " milliseconds.");

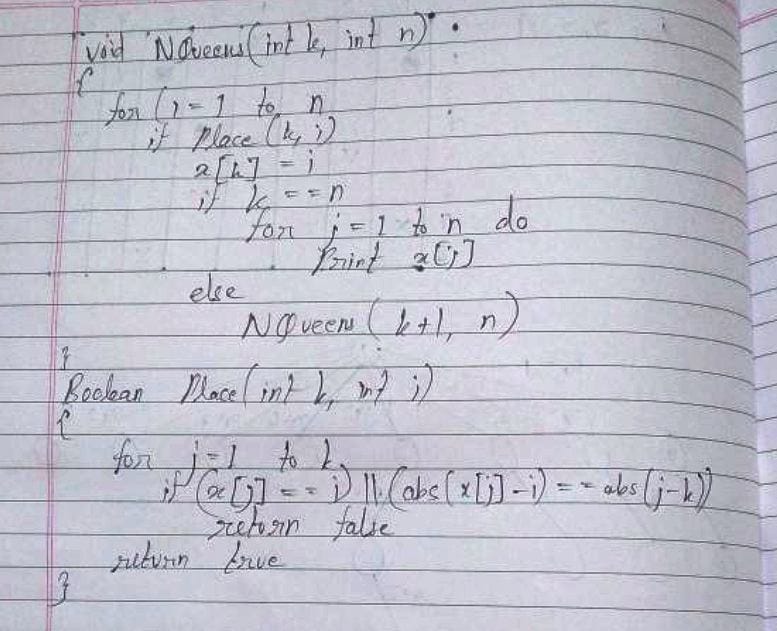
    }

}

**OUTPUT:**

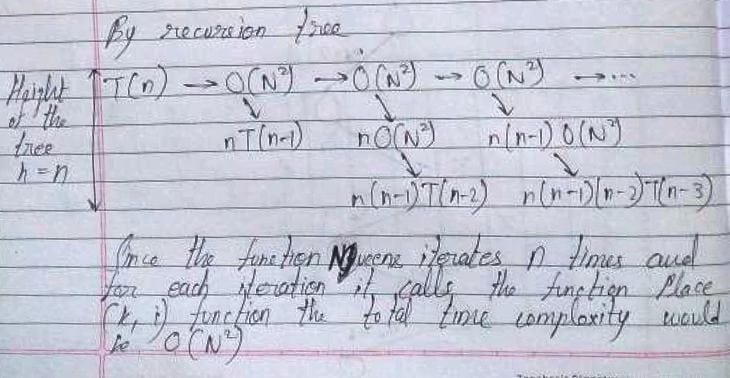


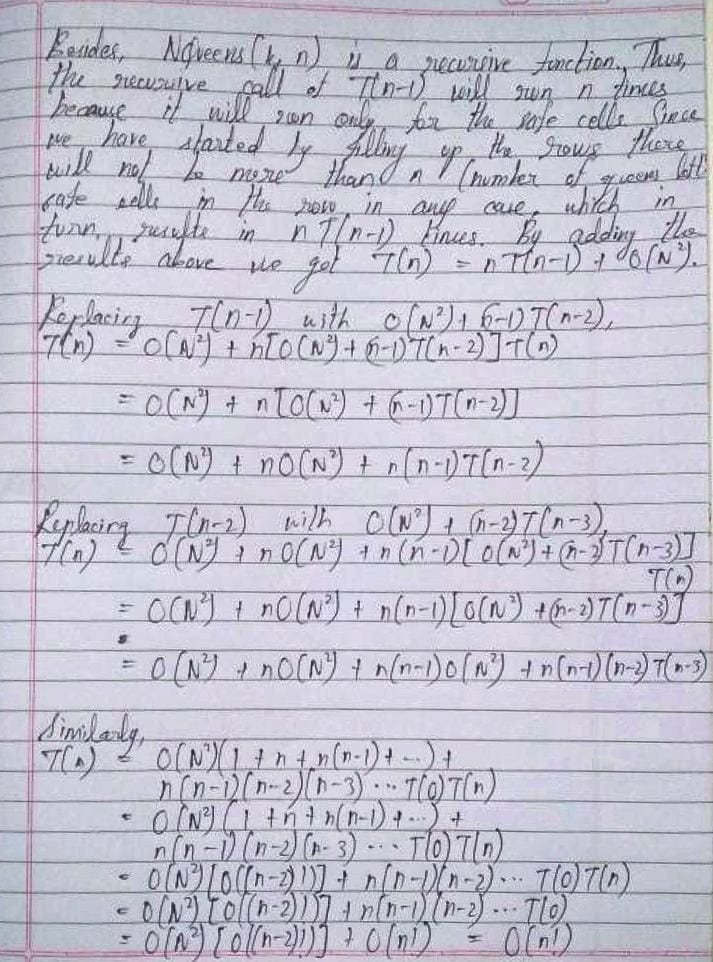
**Algorithm:**



**Analysis of Backtracking solution:**

The space complexity is O(N2). A 2D array of size N rows and N columns is being used and also, because of recursion, the recursive stack will have a linear space requirement. So, overall space complexity is O(N2).

****

****

**CONCLUSION:**

Thus, in this experiment, the concept of Backtracking has been implemented to solve 8 Queens Problem. A simple brute force solution would be to generate all possible chessboards with 8 queens. Accordingly, there would be N2 positions to place the first queen, N2 - 1 positions to place the second queen and so on. The total time complexity in that case would be O(N2N) which is unfeasible for practical application. However, Backtracking solution results in a time complexity of O(n!), which is much more practical. Hence, Backtracking approach should be used to solve N-Queens problem.