Batch: B3 Roll No.: 16010121110 **Experiment No. ___9___** **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**
5. **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**
7. **http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

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 k^th row and i^th 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);
 }
```

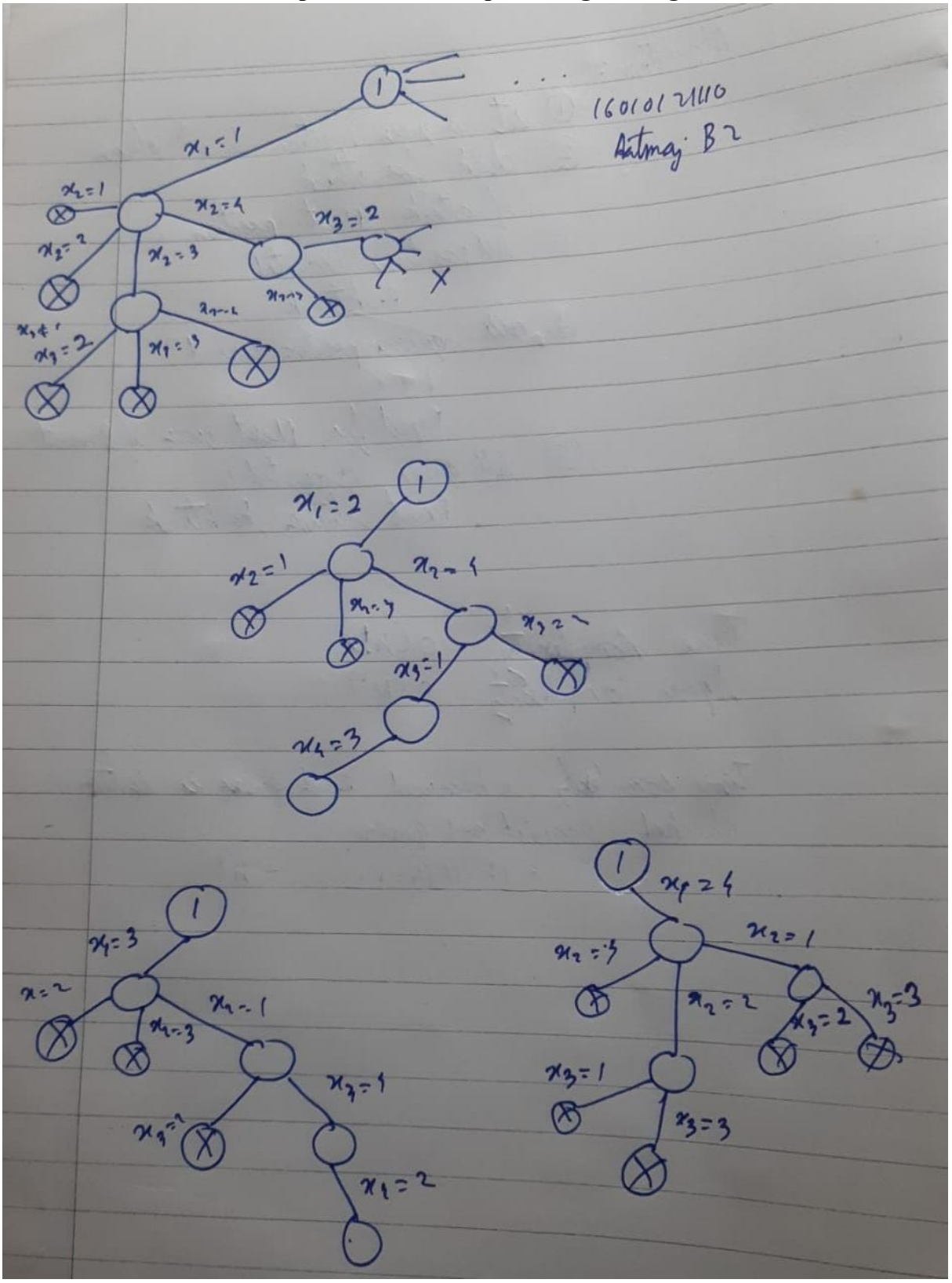
Example 8-Queens Problem:

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):


160101 21110
Aatmaj B 2

Implementation (Code):

```
/**********************************************************************
***************

Need to put value of N

**********************************************************************
***************/
public class queen
{
	public static void main(String[] args) {
		System.out.println("Hello World");
		int N=8;
		 int[][] positions = new int[N][N];

		boolean a = place(0,positions);

		//printPos(positions);
	}
	public static boolean place(int queenNo,int[][] positions){
	   boolean val=false;
      for(int row=0;row<positions[0].length;row++){//iterate row

			if(checkQueen(queenNo,positions,row)==true){
			 //place queen
			    positions[row][queenNo]=1; //column is queenNo

			    if(queenNo==positions[0].length-1){
			       //end of recursion
			   printPos(positions);
			   positions[row][queenNo]=0;
				return true;
			    }
			    //recur
			    if(place(queenNo+1,positions)==false){
			       //failed
				positions[row][queenNo]=0;
				continue; //look for next position
```

```
                    }
                    else{
                val = true;//valid position
                positions[row][queenNo]=0;
                continue; //still look for next position

                    }
                }
            }
        if(val==true){
            return true;
        }
            //backtrack
            return false;
        }
        public static boolean checkQueen(int column,int [][] positions, int
row){


if(checkRow(column,positions,row)&&checkdiaLeft(column,positions,row)&&
checkdiaRight(column,positions,row)){
                return true;
            }
            return false;

        }
        public static boolean checkRow(int column,int [][] positions, int row){
        for(int i=0;i<positions[0].length;i++){ //iterate columns
            if(positions[row][i]==1){

                return false;
            }
        }
        return true;
    }
    public static boolean checkdiaLeft(int column,int [][] positions, int row){
        for(int i=0;i<positions[0].length;i++){ //iterate row
        for(int j=0;j<positions[0].length;j++){ //iterate columns
          if(i-j==row-column){

            if(positions[i][j]==1 ){
```

```
            return false;
          }
         }
       }

   }
    return true;
   }

    public static boolean checkdiaRight(int column,int [][] positions, int row){
      for(int i=0;i<positions[0].length;i++){ //iterate row
      for(int j=0;j<positions[0].length;j++){ //iterate columns
        if(i+j==row+column){

          if(positions[i][j]==1 ){

            return false;
          }
         }
       }

   }
    return true;
   }


   public static void printPos(int [][] positions){
       for(int i=0;i<positions[0].length;i++){ //iterate columns
           for(int j=0;j<positions[0].length;j++){ //iterate rows
          System.out.print(positions[i][j]+",");
       }
       System.out.println();
       }
       System.out.println("_____________________");
   }
}
```

OUTPUT:

0,0,1,0,

1,0,0,0,

0,0,0,1,

0,1,0,0,

0,1,0,0,

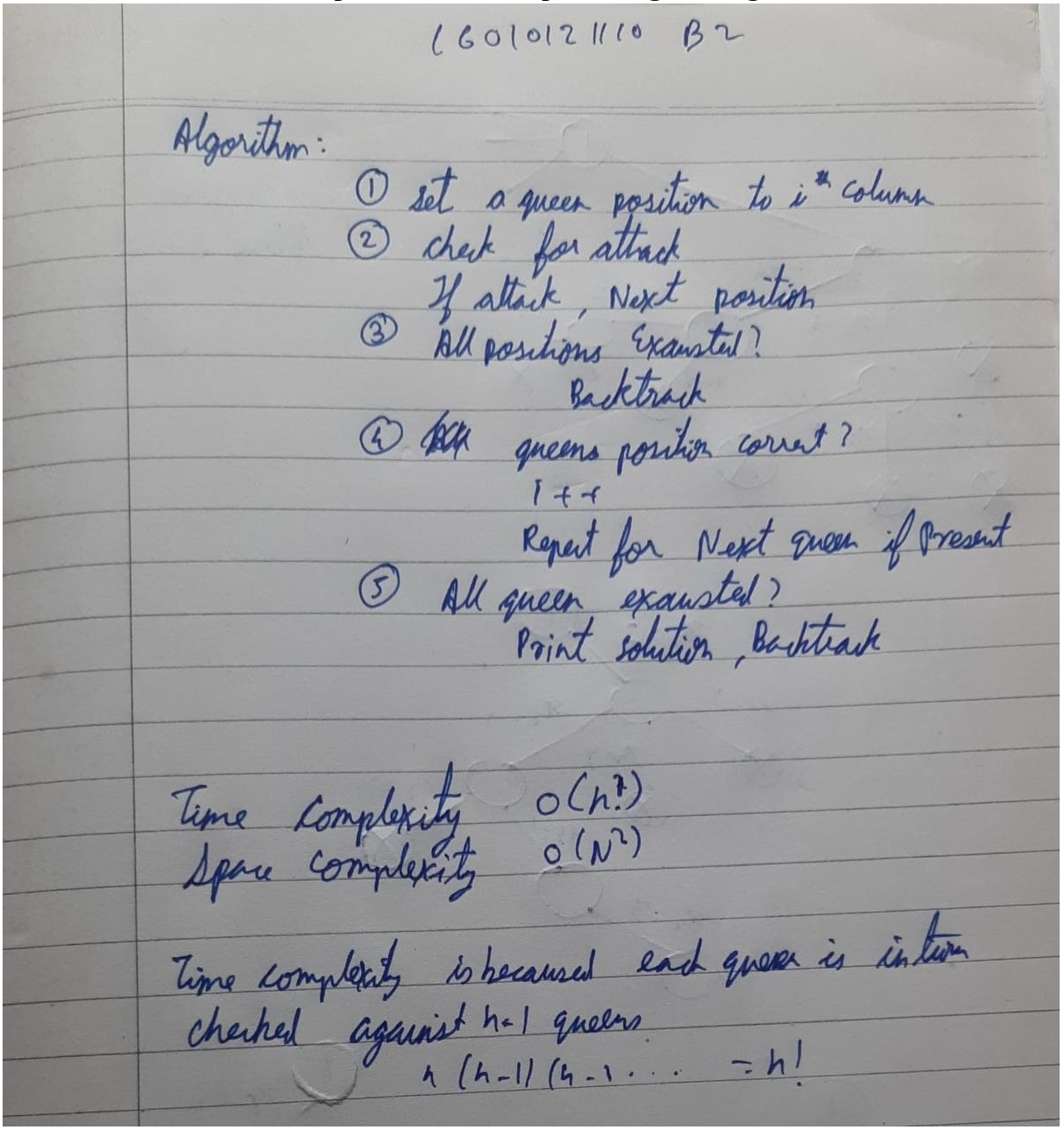
0,0,0,1,

1,0,0,0,

0,0,1,0,

Algorithm:

Analysis of Backtracking solution:



CONCLUSION:

Thus we have understood the N queen algorithm and implemented it. This problem is a standard example of backtracking and is used in various applications like VLSI testing, traffic control, parallel memory storage schemes, and deadlock prevention.