**Topic: Backtracking**

**Theory:** In many applications of the backtrack method, the desired solution is expressible as an n-tuple *(x1,...,Xn),* where the x*i* are chosen from some finite set Si. Often the problem to be solved calls for finding one vector that maximizes (or minimizes or satisfies) a *criterion function P(x1,…..* . , *xn). Sometime*s it seeks all vectors that satisfy *P.* For example, sorting the array of integers in. *a[1* : n] is a problem whose solution is expressible by an *n- tuple, w*here x*i* is the index in *a* of the ith smallest element. The criterion function P is the inequality *a[xi]* ≤ *a[xi+1]* for 1 ≤ i < *n.* The set *Si* is finite and includes the integers 1 through *n.* Though sorting is not usually one of the problems solved by backtracking, it is one example of a familiar problem whose solution can be formulated as an n-tuple.

**Control abstraction**:

void Backtrack( int k )

// This is a schema that describes the backtracking process //using recursion. On entering, the first k-1 values x[1], x[2], //…., x[k-1] of the solution vector x[1:n] have been //assigned. x[] and n are global.

{

for (each x[k] such that x[k] Є T(x[1], …, x[k-1])

{

if (Bk (x[1], x[2], …, x[k]))

{

if (x[1], x[2], …, x[k] is a path to an answer node)

output x[1:k];

if (k < n) Backtrack(k+1);

}

}

}

**Batch: A1 Roll No.: 1911004**

**Experiment No. 8**

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

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

|  |
| --- |
| **Title: Implementation of 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.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,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**

**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

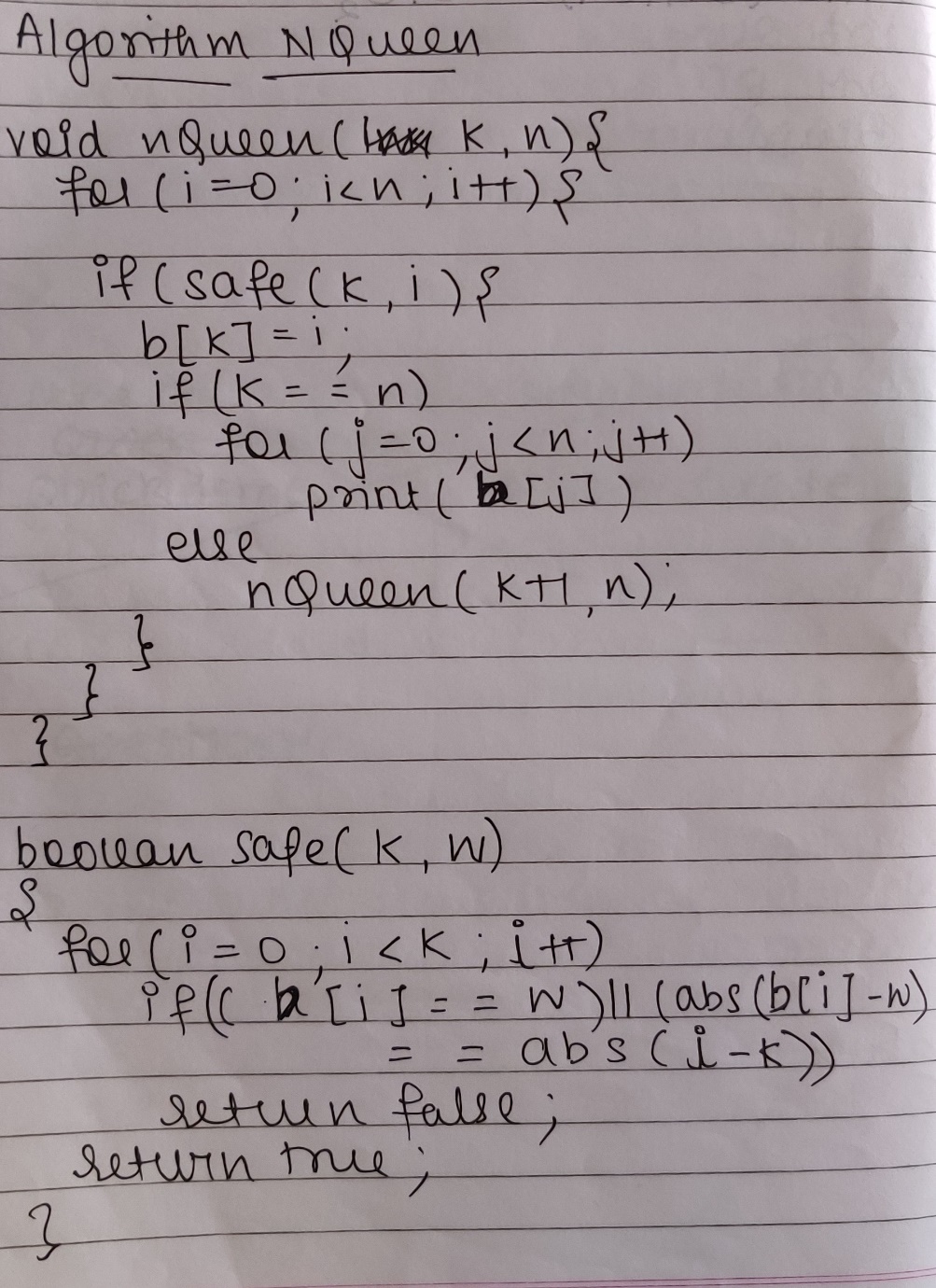
|| (abs(x[j]-i) == abs(j-k))) // or in the same diagonal

return(false);

return(true);

}

**Algorithm:**



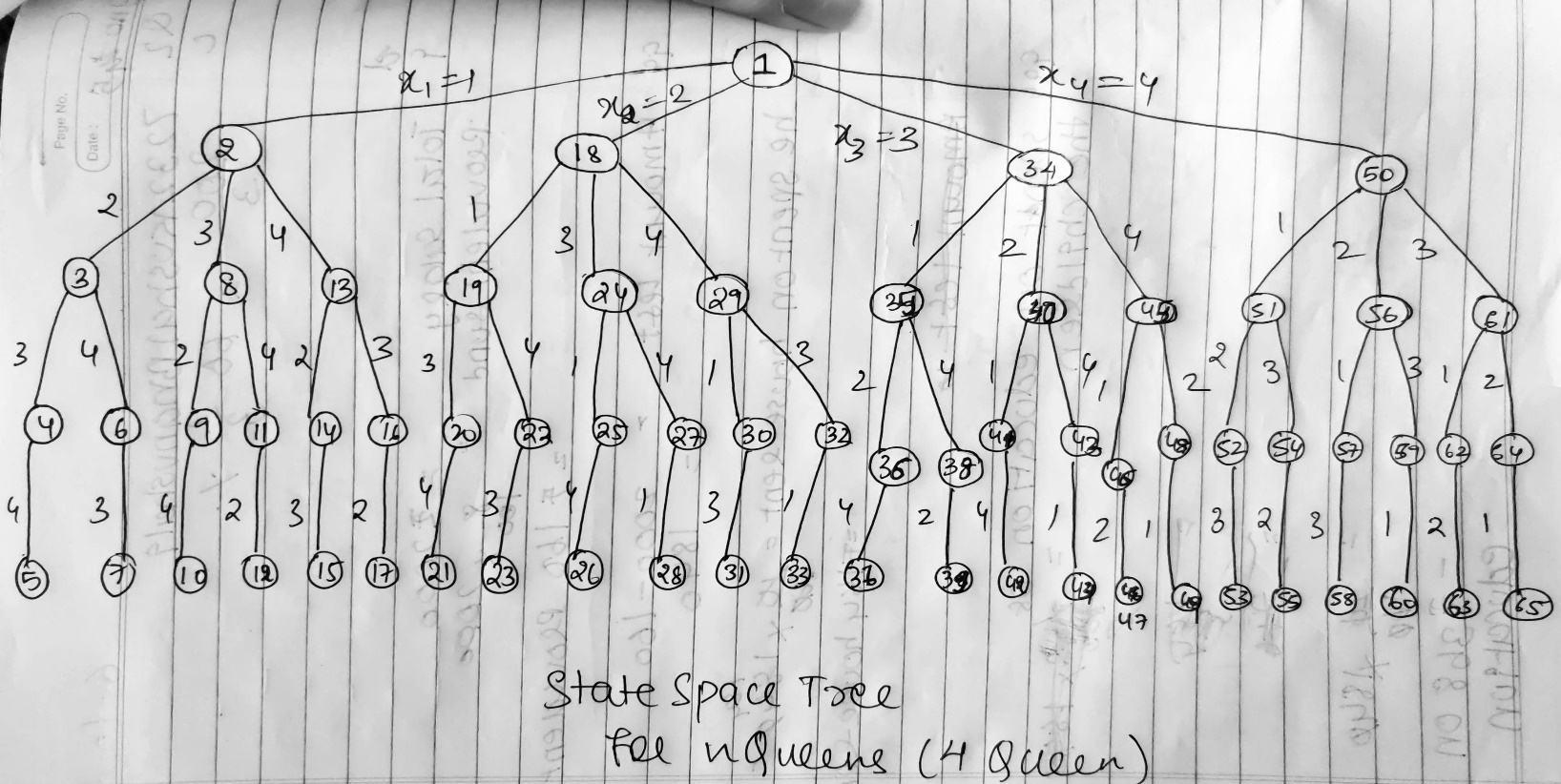
**Example 8-Queens Problem:**

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 nQueen {

public static int board[][],count=0;

public static void main(String[] args) {

Scanner ob = new Scanner(System.in);

System.out.println("N Queens Problem");

System.out.print("Enter no of Queens ");

int n = ob.nextInt();

board = new int[n][n];

nQueens(0);

System.out.println(" ");

}

public static void ptr(int n){

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

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

if(board[i][j]==0)

System.out.print("\_ ");

else

System.out.print("Q ");

}

System.out.println("");

}

}

public static boolean nQueens(int c){

boolean flag = true;

if(c>=board.length){

System.out.println("\nSoln no "+(++count)+" for "+board.length+" Queens\n");

System.out.println("------------------");

ptr(board.length);

System.out.println("\n------------------");

return !flag;//for diff soln

}

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

if(safePlace(i,c)==true){

board[i][c]=1;

System.out.println("");

ptr(board.length);

if(nQueens(c+1)==true){

return flag;

}

board[i][c]=0;

}

}

return (!flag);

}

public static boolean safePlace(int r,int c){

boolean flag = true;

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

if (board[r][j]==1)

return (!flag);

}

for (int i=r,j=c;i>=0 && j>=0;i--,j--){

if (board[i][j]==1)

return (!flag);

}

for (int i=r,j=c;i<board.length && j>=0;i++,j--){

if (board[i][j]==1)

return (!flag);

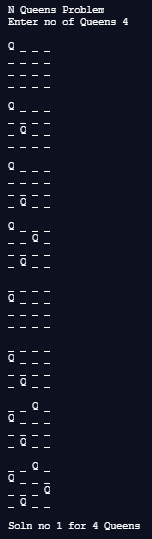
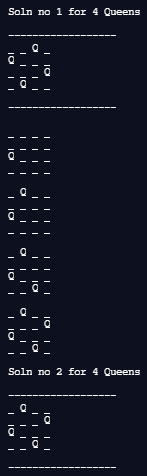
}

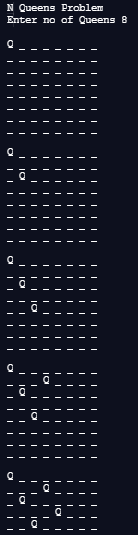
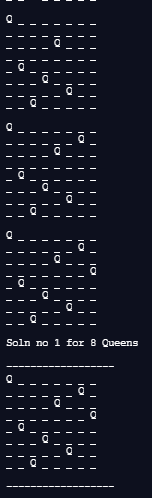
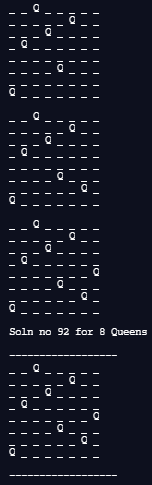
return flag;

}

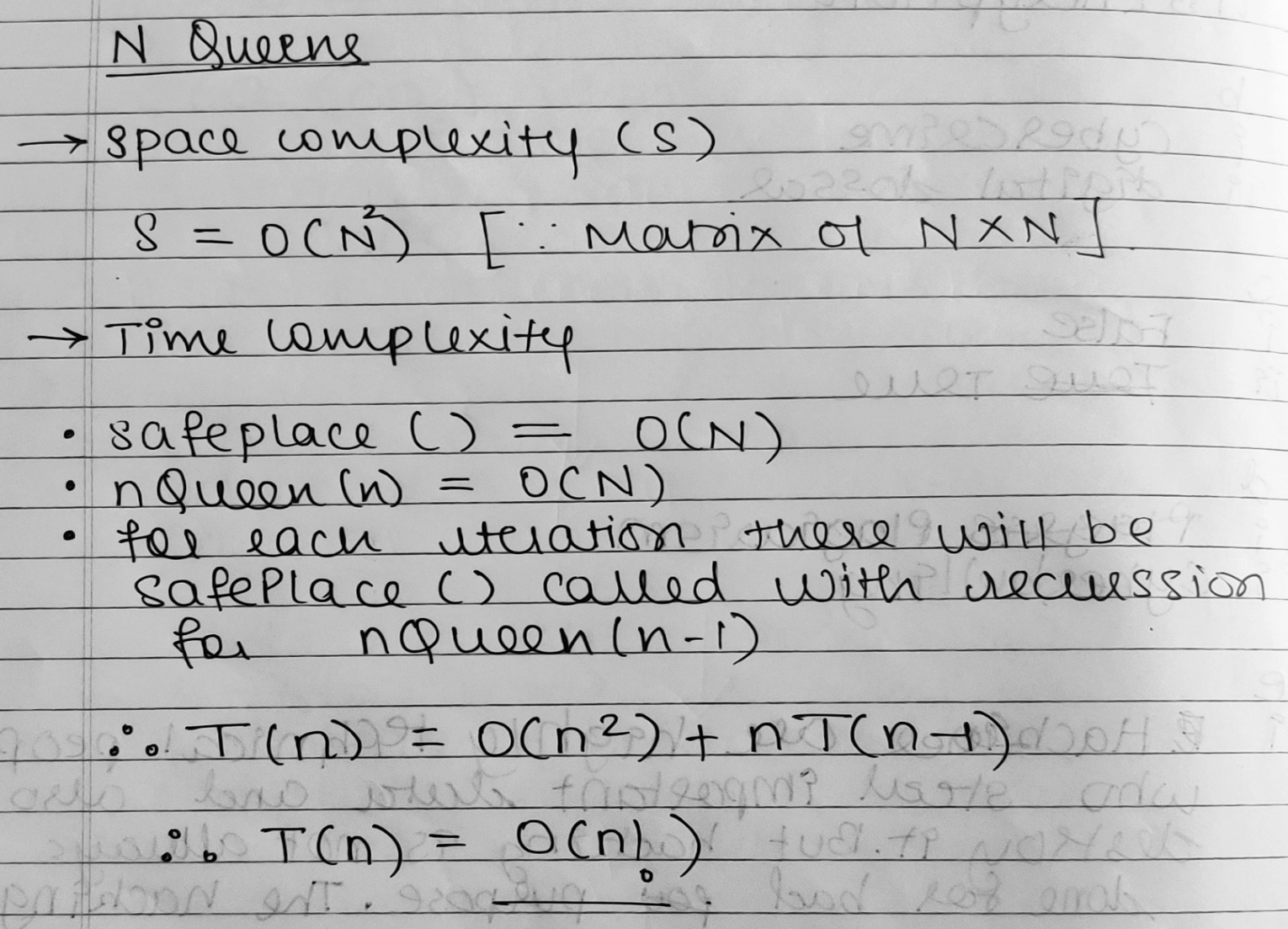
}

**OUTPUT:**

  for 4 Queens

  …  for 8 Queens

**Analysis of Backtracking solution:**



**Conclusion:**

We successfully understood & implemented N Queens problem using Backtracking; thus obtained correct solutions for placing N Queens with backtracking steps in java.