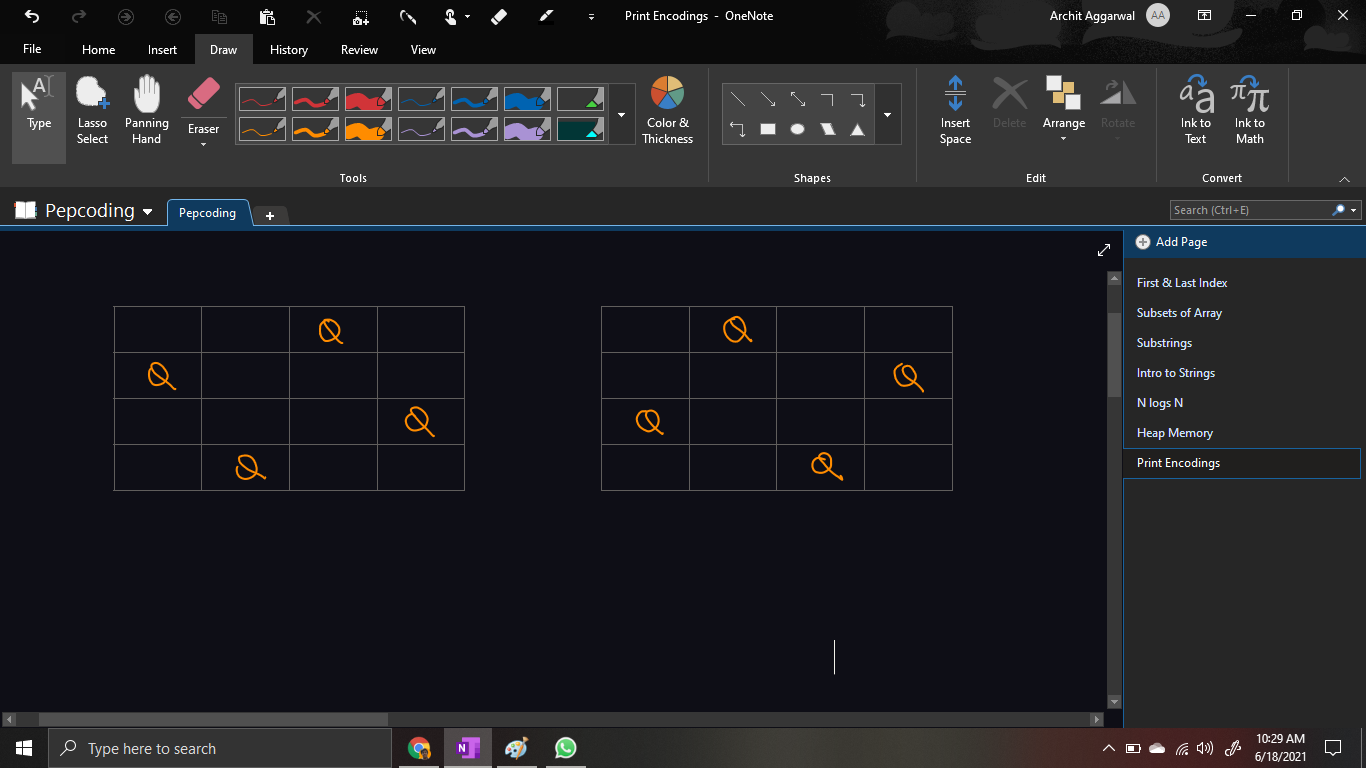
Dear reader, welcome to the combined articles on the problems named **‘**[**N Queens**](https://www.pepcoding.com/resources/online-java-foundation/recursion-backtracking/n-queens-official/ojquestion)**’.** This is a very nice problem which will act as a building block of the backtracking concept.

***Problem Statement:*** The N Queen is the problem of placing N queens on an N×N chessboard so that no two queens attack each other.

Queen can kill other queens in all of the 8 directions possible (left, right, up, down, and all the 4 diagonals).

Given the value of N, you have to print all the valid chess board configurations possible.

Example) For N = 4, there are two valid configurations:



If you are unable to get the question completely, then please refer to the [question video](https://www.youtube.com/watch?v=prZJ0hA43NU).

*Note*: If you have not tried enough to come up with logic, then we recommend you to first spend an hour or so doing it, else read only the logic used, take it as a hint and try the problem again with the same logic.

***(Backtracking) Solution:***

We have to keep in mind the following 4 conditions:

1. **No 2 Queens can be in the same row**
2. **No 2 Queens can be in the same column**
3. **No 2 Queens can be in the left diagonal**
4. **No 2 Queens can be in the right diagonal**

Since,

* No 2 queens can be in the same row
* There are n queens and n rows

It implies that **each row will contain exactly one queen**.

Thus, we will consider taking a decision for a given ***row*** at a given depth of recursion tree, i.e. we will try to place a queen in the ith row at the ith height/level of the recursive tree. (We will place the queen in 1st row at first level, 2nd row at second level and so on).

But, what will be the decisions to take at ith level (for the ith row). It will be to place the queen in which ***column***. In the ith row, we will try to place the queen in any of the columns.

Till now, we have discovered:

* **Level (node in recursion tree) corresponds to choosing a row in grid**
* **Decision (edge in recursion tree) corresponds to choosing a column in the given row.**

But, how is writing conditions for nodes different from edges? We will try to learn the concept of **node-pre, node-post, edge-pre, and edge-post,** using the following code snippet:

public static void printNQueens(int[][] chess, String qsf, int row){

if(/\* Base Case \*/){

System.out.println(qsf + ".");

return;

}

// Node - Pre Statement

for(int col = 0; col < chess.length; col++)

// Decisions or Edges -> Choosing Column for given row

{

// Edge - Pre Statement

printNQueens(chess, qsf + row + "-" + col + ", ", row + 1);

// Next recursive call -> Higher Depth -> Higher row

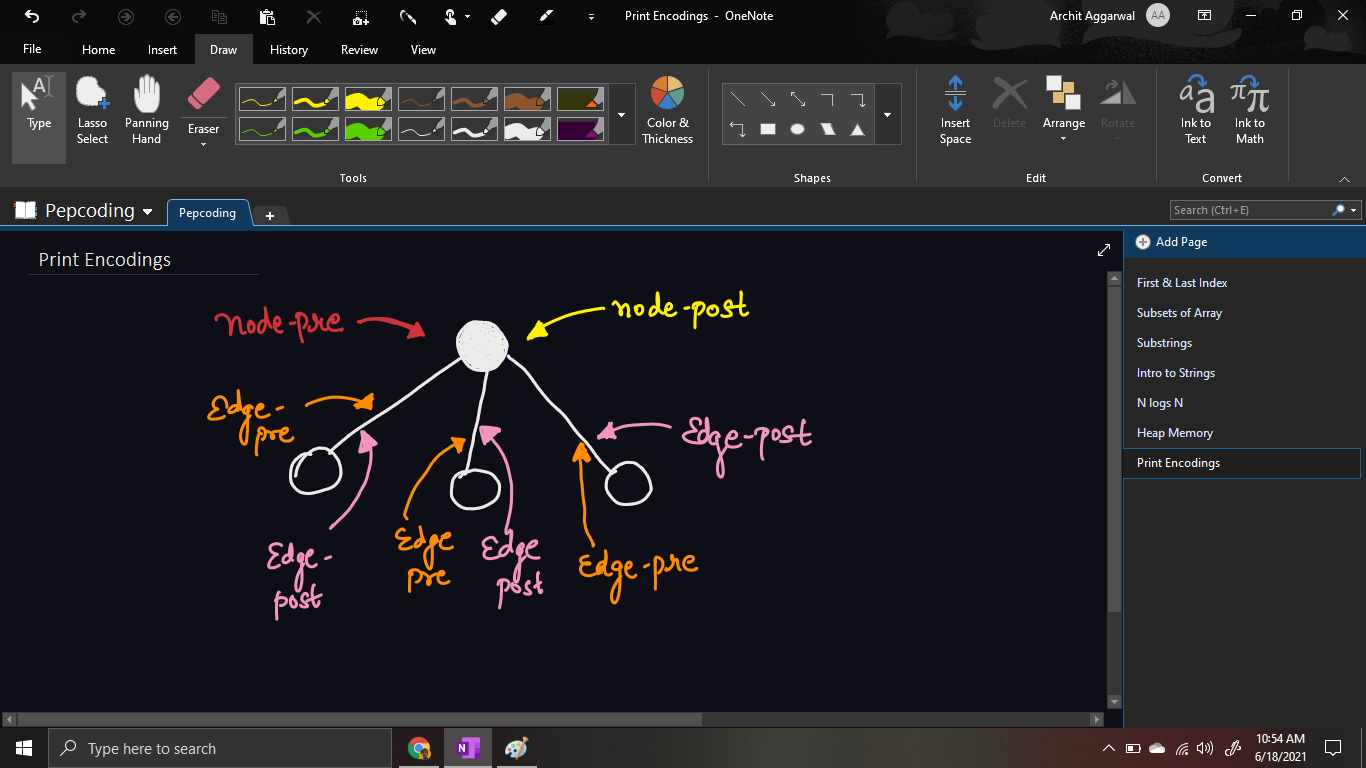
// Edge - Post Statement

}

// Node - Post Statement

}

In the recursion tree, we will find the correct spots of node-pre, node-post, edge-pre and edge-post.



* Node-pre will be the statement in the starting of the for loop (for taking decisions/edges), node-post will be the statement after the end of the for loop.
* Edge-pre is the statement before calling for the next level/depth, and similarly edge-post is the statement after returning from the next level/depth.

We need to try for all columns in a given row, but will all the columns give valid configuration?

No, until now, we have fulfilled the first condition for valid configuration, i.e. no 2 queens in the same row. But to check the remaining three conditions, we need to keep track of all the queens which we have placed so far, and apply some logic to check whether we can place them in the current column for the given row or not.

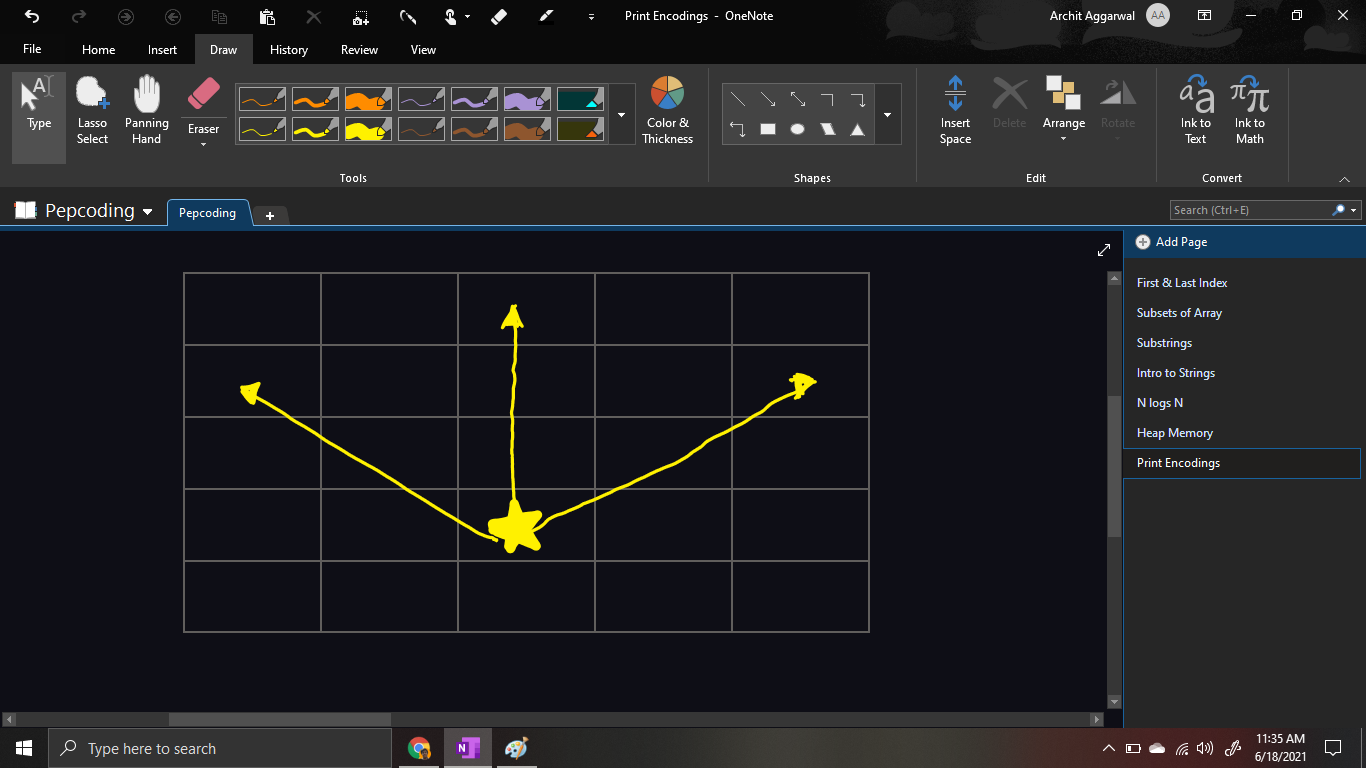
Hence, we need to check whether the column in which we are trying to place the queen is occupied by any of the queens in the previous rows or not. And similarly for the left diagonal and the right diagonal.

We will mark the cell in which the queen is placed by 1 before going to the next level, i.e. in the ***edge-pre statement***. But after returning from the next level, we must remember to remove the queen from that cell back, i.e. make the cell empty (= 0) in the edge-post ***statement***.

* Edge - Pre statement is ***chess[row][col] = 1;***
* Edge - Post statement is ***chess[row][col] = 0;***

We will make a recursive call only if the current column represents valid configuration (satisfy conditions 2,3 and 4). Similar to rows, since there are n columns and no 2 queens can be placed in the same column, hence there must be exactly 1 queen in each column, and same goes for left diagonal and right diagonal as well.

Let us write the code to check if the current cell holds conditions 2, 3 and 4 as well or not.



We will check for all cells along the following 3 directions to check if there is any queen already placed or not (chess[i][j] = 1) or not. We will run three loops, one for each path/direction and return false if any of the cell is occupied else return true.

Now, the only condition left is the ***base case***?

It will occur when we are able to place queens in all the rows, i.e. the current row becomes greater than the number of rows of the table. Hence we will print the current configuration and return.

***Implementation***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[][] chess = new int[n][n];

printNQueens(chess, "", 0);

}

public static void printNQueens(int[][] chess, String qsf, int row){

if(row == chess.length){

System.out.println(qsf + ".");

return;

}

for(int col = 0; col < chess.length; col++){

if(chess[row][col] == 0

&& isQueenSafe(chess, row, col) == true){

chess[row][col] = 1;

printNQueens(chess,

qsf + row + "-" + col + ", ", row + 1);

chess[row][col] = 0;

}

}

}

public static boolean isQueenSafe(int[][] chess,

int row, int col){

for(int i = row - 1, j = col - 1;

i >= 0 && j >= 0; i--, j--){

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

return false;

}

}

for(int i = row - 1, j = col; i >= 0; i--){

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

return false;

}

}

for(int i = row - 1, j = col + 1; i >= 0

&& j < chess.length; i--, j++){

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

return false;

}

}

for(int i = row, j = col - 1; j >= 0; j--){

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

return false;

}

}

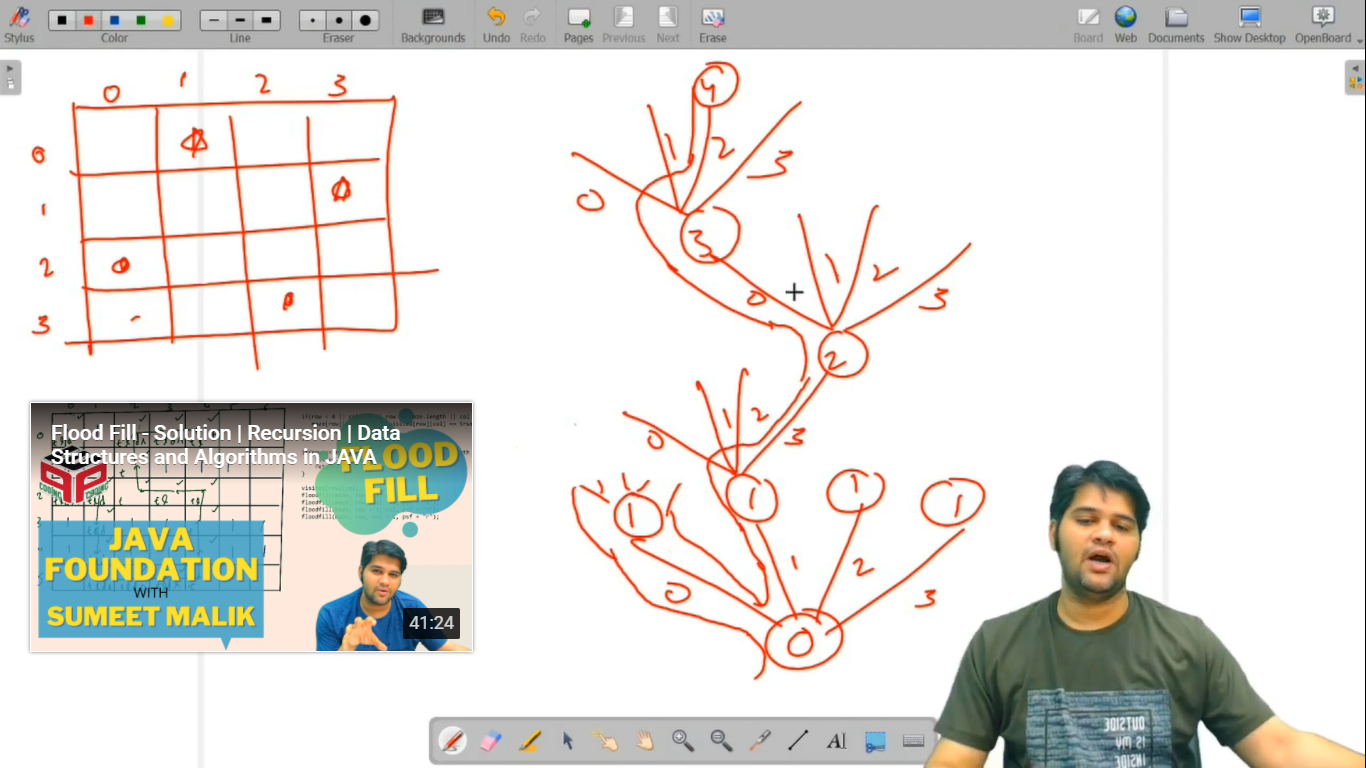
return true;

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=05y82cP3bJo) from timestamp [*4:10 , 23:00]*.

I suggest you draw a recursion tree running your code line by line and see how backtracking works. It is explained using an example in the same video from *[24:21, 30:00]*.



* What is the ***time complexity*** of the above code?

Since we are checking for all the possible cells, time complexity will be ***exponential {O(2^n)}*** in nature.

It is better than brute force/naive approach which has *O(n^n)* time complexity.

* What is the ***space complexity*** of the above code?

The maximum number of recursive calls can be equal to the number of rows. Hence the recursion stack call will take atmax O(n) space.

However, we are not using any extra data structure, hence O(1) auxiliary space.

***Follow Up***: Solution can also be solved using Branch & Bound technique. Also, this approach can be optimized using Bit Manipulation. We will look at these approaches later in the upcoming sections.

***Asked in Companies***: *Accolite, Amazon, Amdocs, D-E-Shaw, MAQ Software, Twitter, Visa and Microsoft*

Hope that you liked the article on N Queen using Backtracking.

Subscribe to Pepcoding’s youtube channel for more such amazing content on Data Structures & Algorithms and follow the resources available for all students in the resources section of Pepcoding’s website.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by:

Archit Aggarwal