Dear reader, welcome to the next problem in the Recursion & Backtracking section named **‘**[**Queens Permutations - 2d As 2d - Queen Chooses**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/queens-permutations-2das2d-queen-chooses-official/ojquestion)**’**.

If somehow you have landed on this problem directly, then I must tell you that the train is on an intermediate station. The station of origin of our train was ‘[**Queens Combinations - 2d As 2d - Box Chooses**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/queens-combinations-2das2d-box-chooses-official/ojquestion)’.

Also, there is a strong prerequisite for this set of problems on permutations & combinations in 2D grid, which is ***permutations & combinations in 1D***.

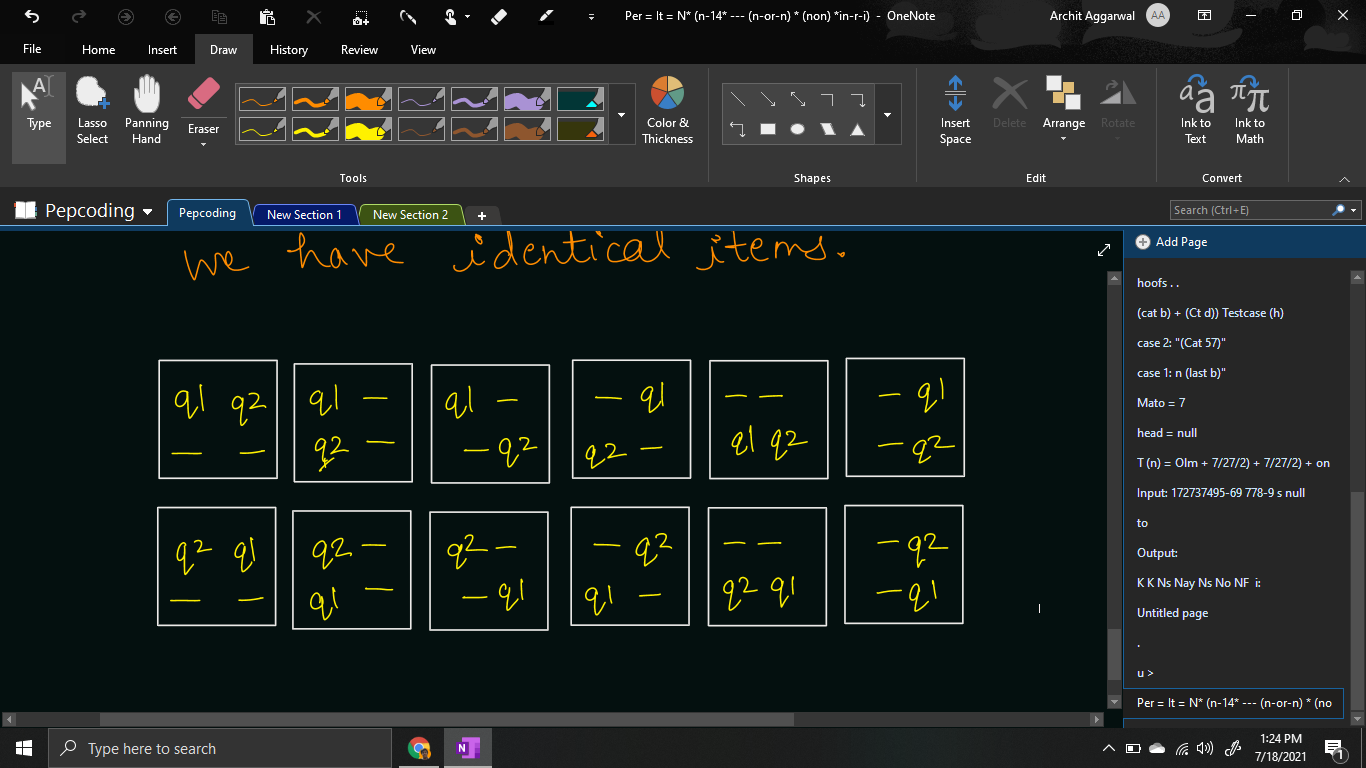
***Problem Statement:***

* You are given a number n, representing the size of a n \* n chess board.
* You are required to calculate and print the **permutations** in which n queens can be placed on the n \* n chess-board.
* Note, in this problem, you can have any queen in any of the cells, i.e. you are **not** taking into consideration whether the queens can kill each other or not.
* Note: Use the code snippet and follow the input/output format. The judge can't force you but the intention is to teach a concept. Play in the spirit of the question.

***Example:***

*Input*: Number of queens (n) = 2

*Output*:



***Solution***

We have already learnt how to generate [**permutations**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/permutation-i-official/ojquestion) of non-identical items in a 1d array by taking levels as items and choices/edges as one of the empty boxes.

In this problem, we are given the **queens as non-identical items**, and there is a slight variation that instead of 1d array of boxes, we are given a 2d array/grid of the chessboard.

So, we will take the **queens as the levels** in the recursion tree, and the **choice/edge will be choosing an empty cell of the grid**.

So, isn’t it simple? Instead of traversing through a 1d array, we need to traverse through the 2d array to find all the empty boxes and generate the permutations by placing the current queen in them.

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

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

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

chess[row][col] = qpsf + 1;

queensPermutations(qpsf + 1, tq, chess);

chess[row][col] = 0;

}

}

}

Please note what should be the **base case** of this problem?

Base case can be considered when we have made a decision for all of the queens, i.e. the queens placed so far (qpsf) is equal to the total number of queens available (n or tq). When we hit the base case, we will print the grid, by appending the ‘q’ letter in front of the queen’s item number, else print ‘-’ followed by tab space for the empty cell.

if(qpsf == tq){

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

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

System.out.print(chess[row][col] != 0

? "q"+ chess[row][col] + "\t" : "-\t");

}

System.out.println();

}

System.out.println();

return;

}

**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void queensPermutations(int qpsf, int tq, int[][] chess){

if(qpsf == tq){

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

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

System.out.print(chess[row][col] != 0

? "q"+ chess[row][col] + "\t" : "-\t");

}

System.out.println();

}

System.out.println();

return;

}

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

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

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

chess[row][col] = qpsf + 1;

queensPermutations(qpsf + 1, tq, chess);

chess[row][col] = 0;

}

}

}

}

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

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

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

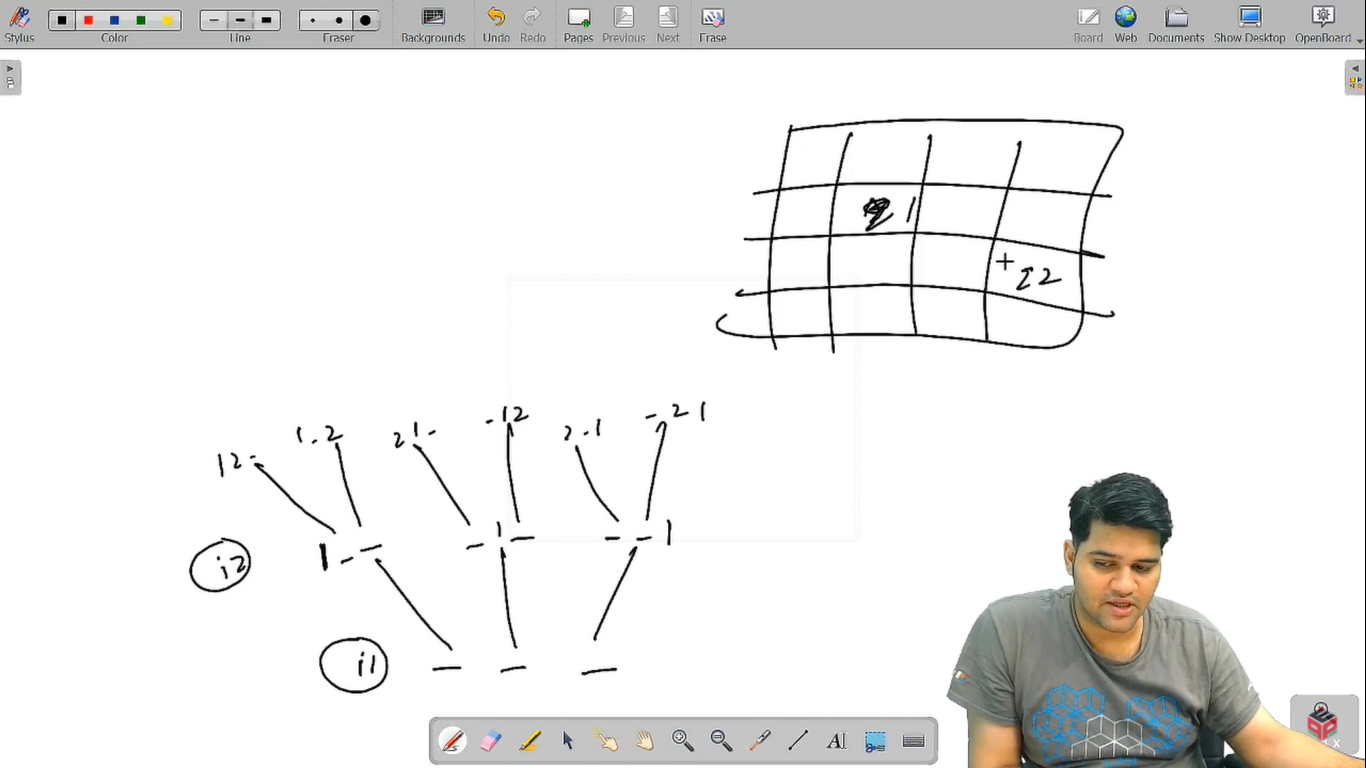
int[][] chess = new int[n][n];

queensPermutations(0, n, chess);

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=mkl6KOwtdbk&list=TLGGB5PBMgORvoExODA3MjAyMQ). Please refer to it for a better understanding of the algorithm and the implementation.



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

In the recursion tree, we are having queens as levels, and the choices as selecting a cell in the grid. Since queens are n and the total cells are n^2, hence the total time complexity will be O(n^2 \* n^2 \* …. n times) = **O(n2n)**.

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

Since, the maximum depth of recursion is equal to the number of queens = n, hence the space complexity will be **O(n)**, as recursion takes function call stack space.

Hope that you liked the article. Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms. 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](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)