Dear reader, welcome to the next problem in the Recursion & Backtracking section named **‘**[**Queens Permutations - 2d as 2d - Box Chooses**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/queens-permutations-2das2d-box-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)’ and the previous station was **‘**[**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)**’**. Please join the journey from the beginning to experience the full joy.

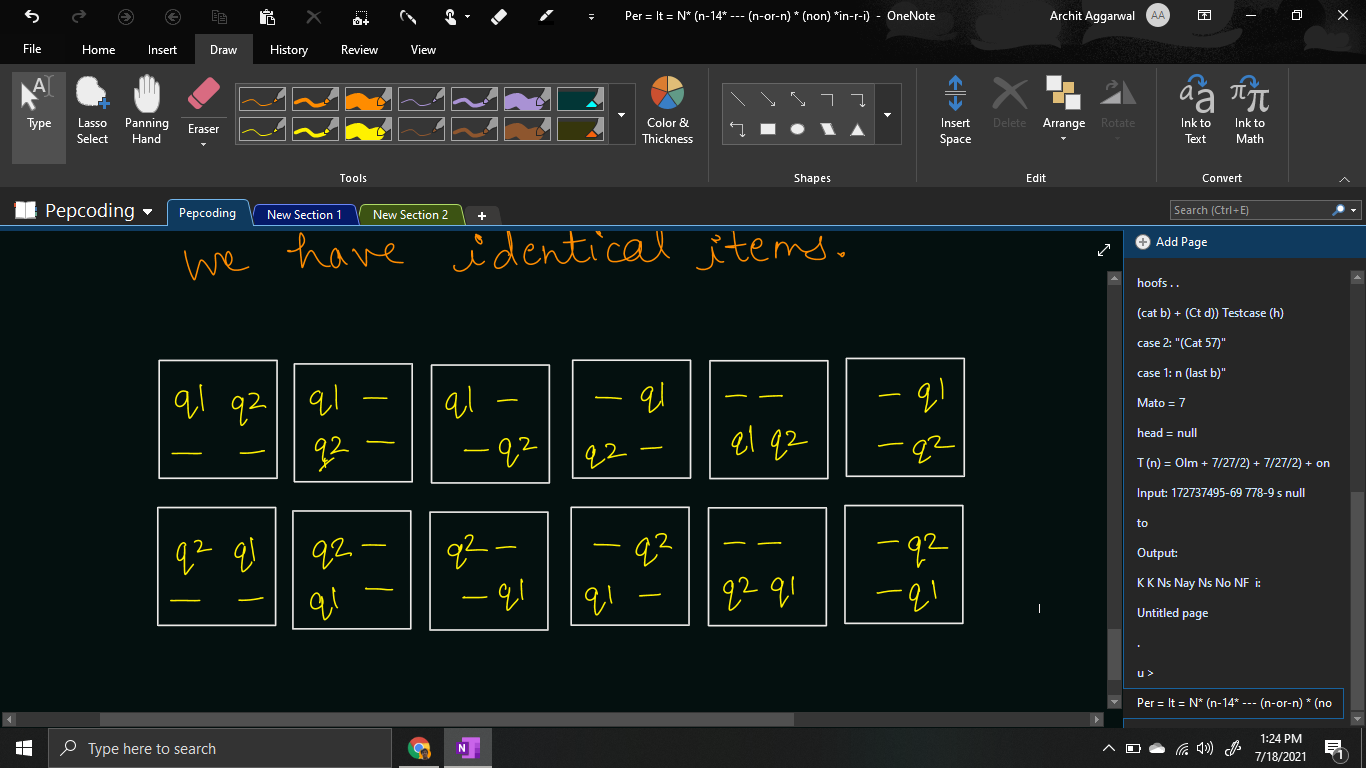
Also, there is a strong prerequisite for this set of problems on permutations & combinations in 2D grid, which is ***permutations & combinations in 1D***.

***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/permutations-ii-official/ojquestion) of non-identical items in a 1d array by taking levels as boxes and choices as selecting a non-identical item out of the items not placed so far, or letting the box remain empty.

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 **cells of the grid as the levels** in the recursion tree, and the **choice/edge will be selecting a queen not placed so far or letting the box remain empty**.

So, the only change will be how to take cells of the grid as the levels instead of indices of 1d array in the simpler version.

We will take the row number and the column number of the cell. There are two possibilities:

1. If the current cell is the last cell in it’s row, then the next cell will be the first cell of the next row, i.e. if the current cell is (r, n-1), then the next cell will be (r + 1, 0).
2. If the current cell is not the last cell in it’s row, then the next cell will be the neighbouring cell in right, i.e. if the current cell is (r, c), then the next cell will be (r, c+1).

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

Base case can be considered when we have taken all the cells of the grid in consideration. Please note that the last cell in the recursion tree will be the bottom-right most cell i.e. (n-1, n-1). Hence, the base case can be considered as when the row number becomes n, because the next cell of (n-1, n-1) will be (n, 0) but it does not exist in the grid.

if (row == tq) {

if (qpsf == tq) {

System.out.println(asf);

System.out.println();

}

return;

}

**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void queensPermutations(int qpsf,

int tq, int row, int col, String asf, boolean[] queens) {

if (row == tq) {

if (qpsf == tq) {

System.out.println(asf);

System.out.println();

}

return;

}

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

if (queens[i] == false) {

queens[i] = true;

queensPermutations(qpsf + 1, tq,

col == tq - 1 ? row + 1 : row,

col == tq - 1 ? 0 : col + 1,

col == tq - 1 ? asf + "q" + (i + 1) + "\n"

: asf + "q" + (i + 1) + "\t",

queens);

queens[i] = false;

}

}

queensPermutations(qpsf + 0, tq,

col == tq - 1 ? row + 1 : row,

col == tq - 1 ? 0 : col + 1,

col == tq - 1 ? asf + "-\n" : asf + "-\t",

queens);

}

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

BufferedReader br = new

BufferedReader(new InputStreamReader(System.in));

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

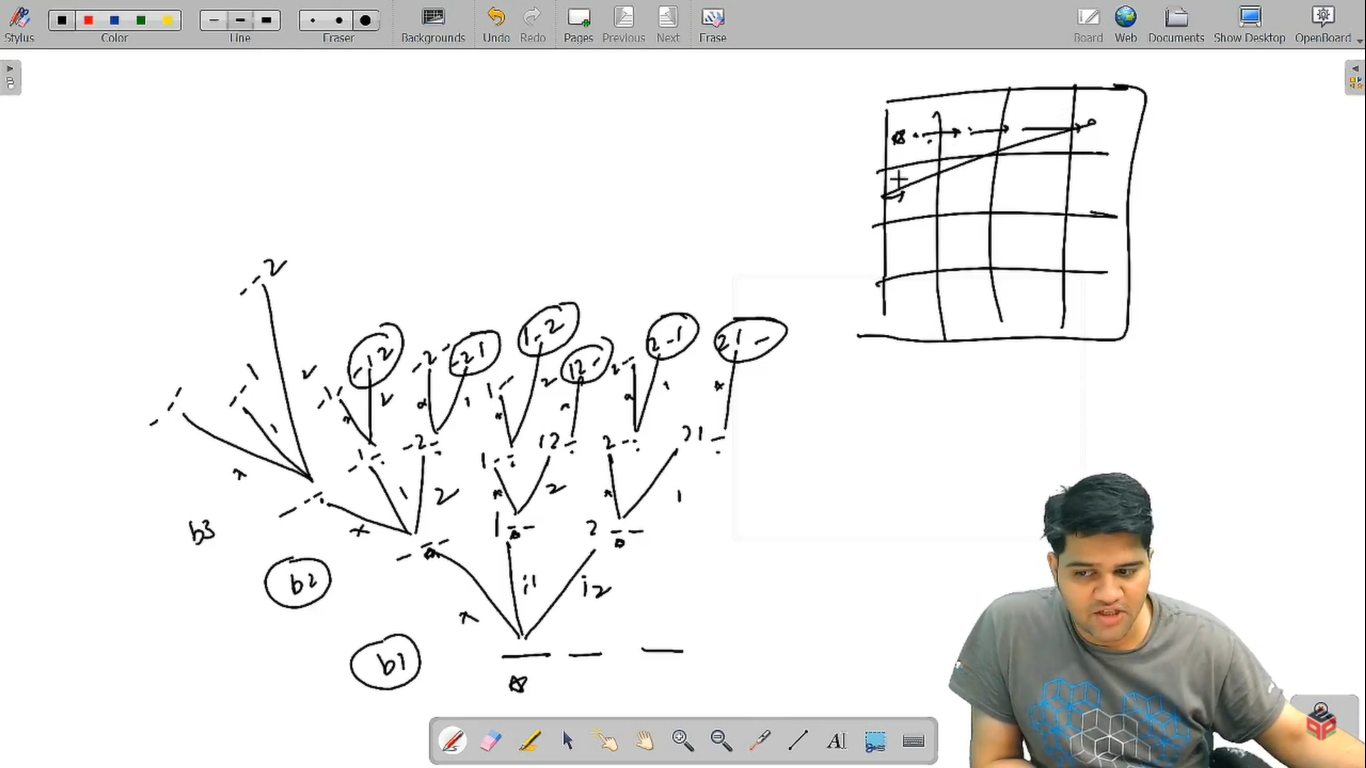
boolean[] queens = new boolean[n];

queensPermutations(0, n, 0, 0, "", queens);

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=5ujm7QQUwhs&list=TLGGn9BmIFVM47oxODA3MjAyMQ). 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 cells as levels, and the choices as selecting a queen or not making it remain empty.

Since choices are (n+1) {+1 for no queen selected} and the total cells are n^2, hence the total time complexity will be O((n + 1) \* (n + 1) \* …. n^2 times) = O((n+1)n^2) = **O(nn^2)**.

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

Since, the maximum depth of recursion is equal to the number of cells = n^2, hence the space complexity will be **O(n^2)**, 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/)