Dear reader, welcome to the next problem in the Recursion & Backtracking section named ‘[**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)’.

This problem marks the beginning of a new set of problems based on Permutations & Combinations in 2D arrays/matrices. I request you to complete this set in one go, and follow the order of problems.

Also, before moving onto 2D permutations and combinations, there is a strong prerequisite, which is [**Permutations**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/permutation-i-official/ojquestion) **&** [**Combinations**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/combinations-i-official/ojquestion) **in 1D**.

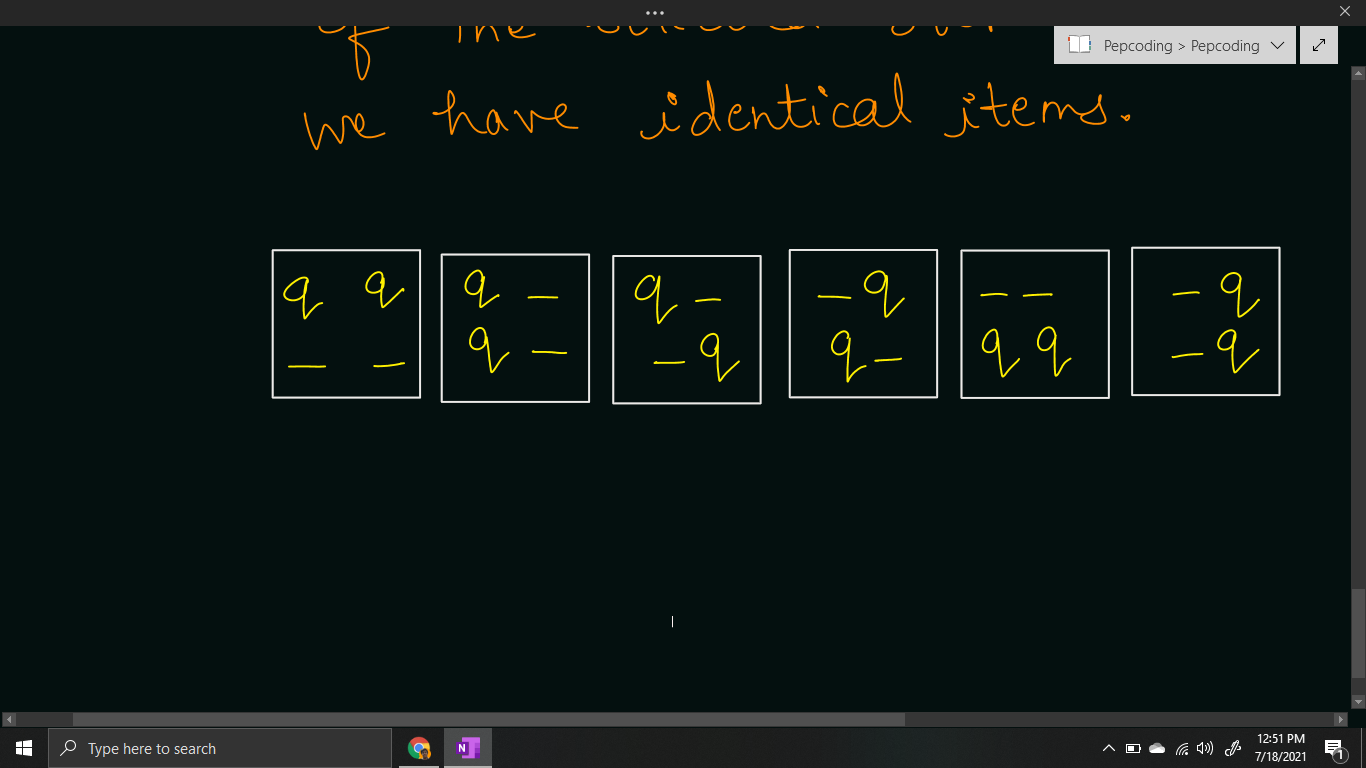
***Problem Statement:***

* You are given a number n, representing the size of a n \* n chess board.
* You are required to calculate and print the **combinations** 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 only learnt how to generate [combinations](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/combinations-i-official/ojquestion) of identical items in a 1d array by taking levels as boxes/cells and choices/edges as placing an item or not.

In this problem, we are given the **queens as 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 whether to place a queen in the cell or let it remain empty**.

So, how to traverse through all the cells of the grid as the levels?

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).

Now, let us define the expectation and faith in our recursive functions, and then derive a recursive relation by meeting expectation with faith.

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 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.

**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void queensCombinations(int qpsf,

int tq, int row, int col, String asf){

if(row == tq){

if(qpsf == tq){

System.out.println(asf);

}

return;

}

queensCombinations(qpsf + 1, tq,

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

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

col == tq - 1? asf + "q\n": asf + "q");

queensCombinations(qpsf + 0, tq,

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

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

col == tq - 1? asf + "-\n": asf + "-");

}

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

BufferedReader br = new

BufferedReader(new InputStreamReader(System.in));

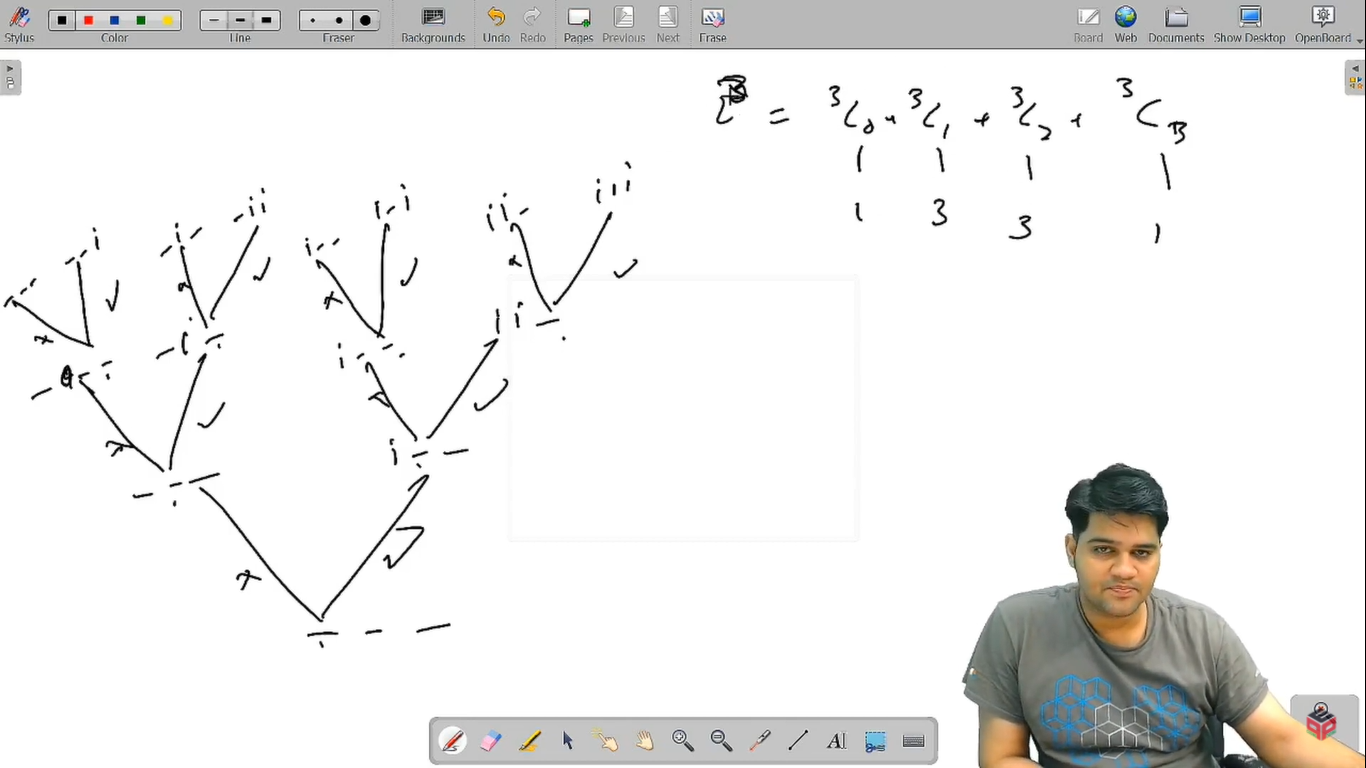
int n = Integer.parseInt(br.readLine());

queensCombinations(0, n, 0, 0, "");

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=Ra_fCLyWtr0&list=TLGG1ixHVb7oB4ExODA3MjAyMQ). Please refer to it for a better understanding of the algorithm and the implementation.



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

Since, the recursion tree will grow for at max n^2 depth (where n = number of queens), and at each level, we are having two choices, the overall time complexity will be O(2 \* 2 \* 2 … n^2 times) = **O(2(n^2))**

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

Since, the maximum depth of recursion is 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/)