Dear reader, welcome to the next problem in the Recursion & Backtracking section named ‘[**Queens Combinations - 2d as 2d - Queen Chooses**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/queens-combinations-2das2d-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 - Box Chooses**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/queens-permutations-2das2d-box-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***.

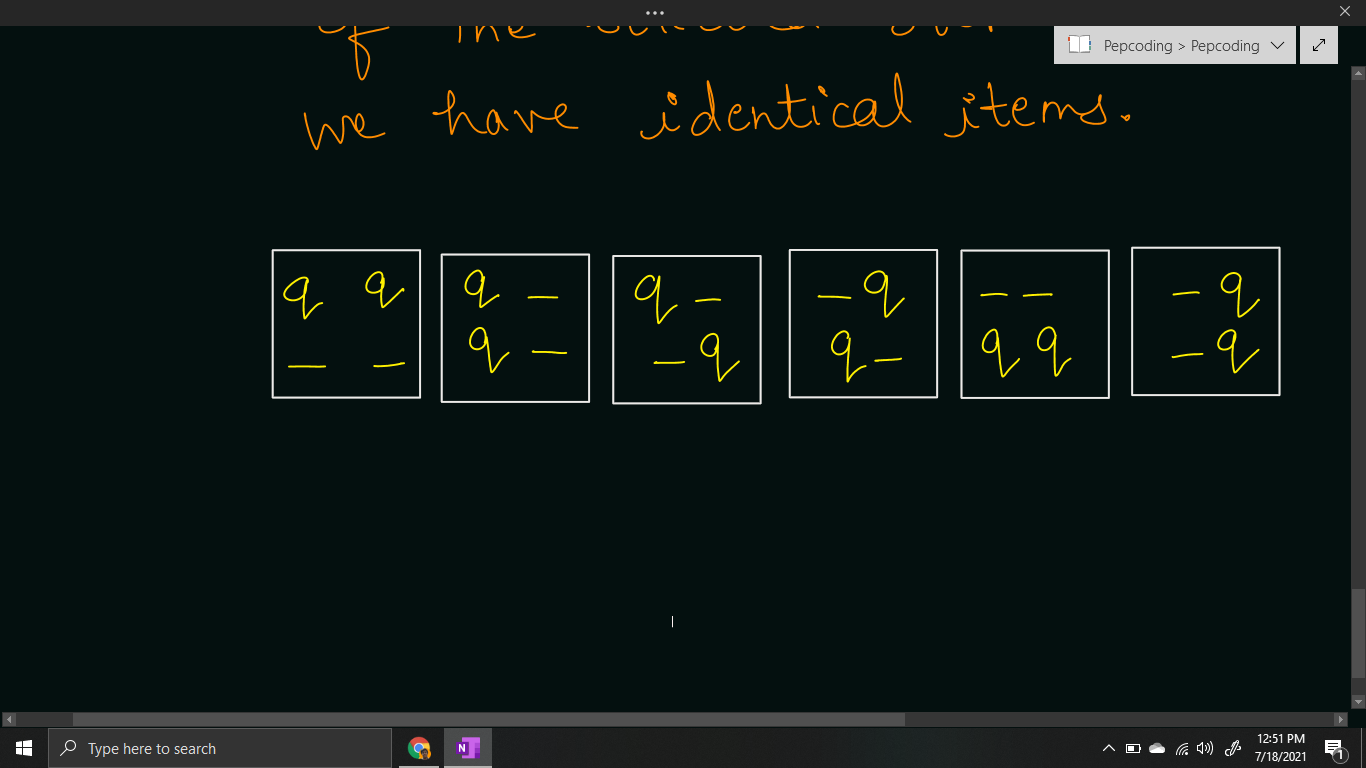
***Problem Statement:***

* You are given a number n, representing the size of a n \* n chess board.
* You are required to calculate and print the **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 already learnt how to generate [**combinations**](https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/recursion-and-backtracking/combinations-2-official/ojquestion) of identical items in a 1d array by taking levels as items (in increasing order only) and choices as selecting an empty box.

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 **queens (in increasing order only) as the levels** in the recursion tree, and the **choice/edge will be selecting an empty cell after the last queen’s cell**.

Isn’t it simple? Just replace the traversal over 1d array for selecting an empty box to traversal over 2d array for selecting an empty cell. But, please be careful that we need to traverse through the cells only after the last queen’s cell (to avoid permutations).

Hence, we will skip all the rows less than the last queen’s row, and also skip all the cells in the left of the last queen’s row.

Thus, we will traverse through the cells in the ith row to the right of last queen’s cell and all the cells in the bottom rows.

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

for(int col = (row == i? j + 1: 0);

col < chess.length; col++){

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

chess[row][col] = true;

queensCombinations(qpsf + 1, tq, chess, row, col);

chess[row][col] = false;

}

}

}

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 writing the ‘q’ for filled cells, 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]? "q\t": "-\t");

}

System.out.println();

}

System.out.println();

return;

}

**Java Code**

import java.io.\*;

import java.util.\*;

public class Main {

public static void queensCombinations(int qpsf, int tq, boolean[][] chess, int i, int j){

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]? "q\t": "-\t");

}

System.out.println();

}

System.out.println();

return;

}

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

for(int col = (row == i? j + 1: 0);

col < chess.length; col++){

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

chess[row][col] = true;

queensCombinations(qpsf + 1, tq, chess, row, col);

chess[row][col] = false;

}

}

}

}

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

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

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

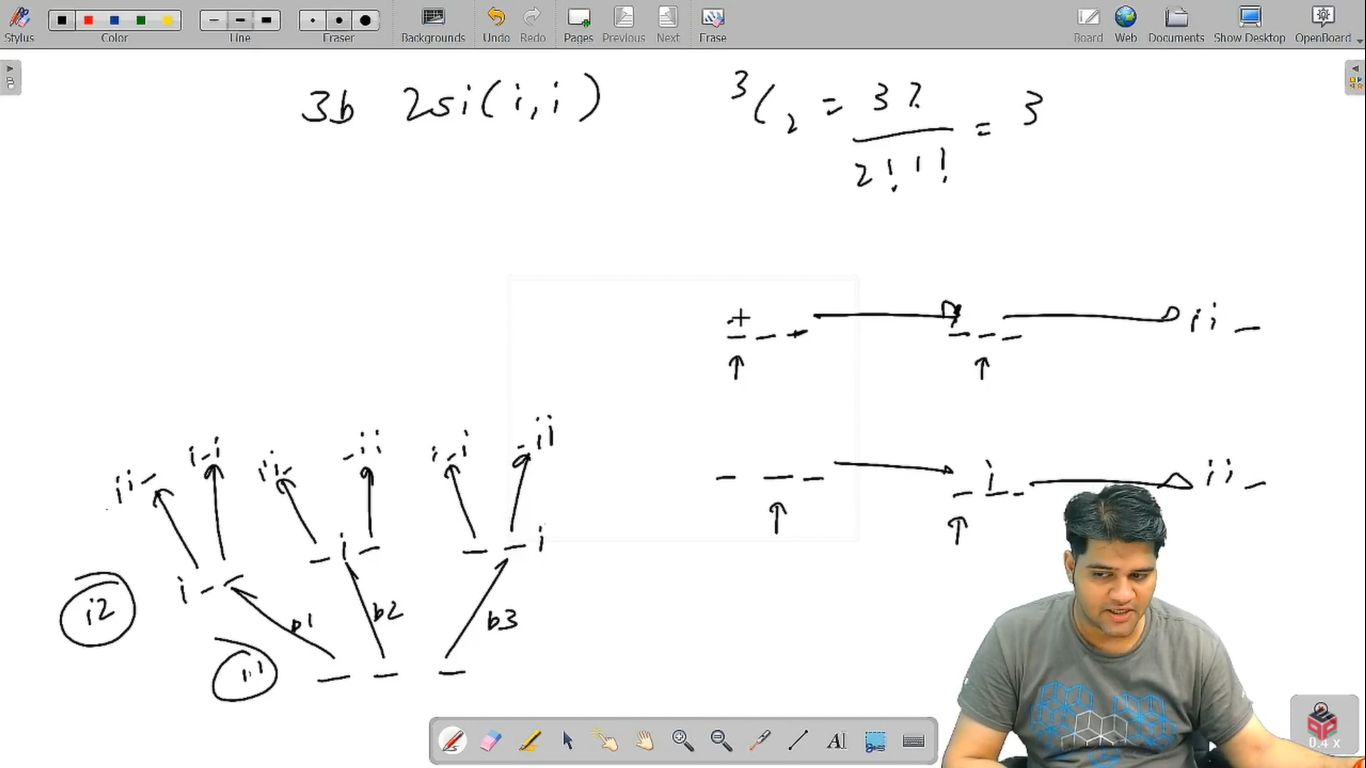
boolean[][] chess = new boolean[n][n];

queensCombinations(0, n, chess, 0, -1);

}

}

Java Code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=zOmrEPnrrJQ&list=TLGGhyfYkSFJq1IxODA3MjAyMQ). 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 an empty box. Since levels are n (queens) and the total cells (choices) 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/)