N Queen Problem [LeetCode](https://leetcode.com/problems/n-queens/description/)

The **n-queens** puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return *all distinct solutions to the****n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.

Example: The Distinct Representation of Chessboard

{ {

. . Q .

Q . . .

. . . Q

. Q . . }, {

. Q . .

. . . Q

Q . . .

. . Q . } }

**Approach 1: function to solve N-Queens problem using the Backtracking approach**

**Function Purpose:**

Solve the N-Queens problem using the backtracking approach.

**Explanation:**

* **isPossible Function:**
  + Checks if placing a Queen at the specified position is feasible, considering the row, column, and diagonals.
* **solve Function:**
  + Recursive backtracking function to explore all possible placements of Queens on the chessboard.
* **solveNQueens Function:**
  + Initializes an empty chessboard and starts solving from the first column using the **solve** function.

**Time Complexity:**

* **Backtracking per Queen Placement: O(N!), where N is the size of the chessboard (number of queens).**

**Space Complexity:**

* **Chessboard Storage: O(N^2), where N is the size of the chessboard.**

**Approach 2: function to solve N-Queens problem using the Optimized Backtracking approach**

**Function Purpose:**

Solve the N-Queens problem using an optimized backtracking approach.

**Explanation:**

* **isPossible Function:**
  + Checks if placing a Queen at the specified position is feasible using hash maps to track occupied rows and diagonals.
* **setMapValues Function:**
  + Sets values in hash maps when placing a Queen.
* **resetMapValues Function:**
  + Resets values in hash maps during backtracking.
* **nQueensHelper Function:**
  + Recursive backtracking function to explore all possible placements of Queens on the chessboard using hash maps.
* **solveNQueensOptimized Function:**
  + Initializes an empty chessboard and starts solving from the first column using the **nQueensHelper** function.

**Time Complexity:**

* **Backtracking per Queen Placement: O(N!), where N is the size of the chessboard (number of queens).**

**Space Complexity:**

* **Chessboard Storage: O(N^2), where N is the size of the chessboard.**
* **Hash Map Storage: O(N).**

**Conclusion:**

* Both approaches solve the N-Queens problem using backtracking.
* The optimized approach reduces redundant checks using hash maps for row and diagonal occupancy.
* The time complexity remains exponential due to the nature of the problem.