LeetCode 51 — N Queens

1. Problem Title & Link

51. N-Queens

🔗 https://leetcode.com/problems/n-queens/

2. Problem Summary

Place **n queens** on an **n×n chessboard** such that:

- No two queens attack each other
- Meaning → no same **row**, **column**, or **diagonal**

Return **all distinct board configurations**.

Each solution must be represented as:

```
[".Q..",   # Q in (0,1)
 "...Q",
 "Q...",
 "..Q."]
```

3. Examples

Input:

n = 4

Output:

```
[
 [".Q..",
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",
  "Q...",
  "...Q",
  ".Q.."]
]
```

4. Constraints

- 1 <= n <= 9
- Output count grows super fast (backtracking essential)
- Board must be valid (no two queens attack each other)

5. Thought Process (Step-by-Step)

This is a **backtracking + constraints pruning** problem.

💡 **Brain Model**

Place queens **row by row**:

At each row:

- Try placing queen in each column
- But only if:
  - Column not used
  - Major diagonal not used (r − c)
  - Minor diagonal not used (r + c)

If valid → place queen → move to next row
If all rows placed → store solution
If invalid later → backtrack (remove queen)

⚔ **Attack Conditions**

For position (r, c):

Type	Condition	Representation
Column	same c	use set: `cols`
Main Diagonal	r − c	use set: `diag1`
Anti Diagonal	r + c	use set: `diag2`

These three sets guarantee **O(1)** safety check.

🌿 **Backtracking Structure**

```
row = 0
board = ["." * n] * n

Try col = 0…n-1:
    If safe:
        place Q
        recurse to next row
        remove Q
```

6. Pseudocode

```
Initialize board
Initialize sets: cols, diag1, diag2
res = []

function backtrack(row):
    if row == n:
        add board copy to res
        return
```

```
    for col in 0..n-1:
        if col in cols or (row-col) in diag1 or (row+col) in diag2:
            continue

        place Q at (row,col)
        add col, row-col, row+col to sets
        backtrack(row+1)
        remove Q and remove from sets

backtrack(0)
return res
```

7. Code Implementation (Python — with Clean Comments)

```
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:

        res = []
        board = [["."] * n for _ in range(n)]

        cols = set()      # columns where queens exist
        diag1 = set()     # r - c (main diagonal)
        diag2 = set()     # r + c (anti diagonal)

        def backtrack(row):
            # If all rows placed → valid solution
            if row == n:
                res.append(["".join(r) for r in board])
                return

            for col in range(n):
                # If queen attacks → skip
                if col in cols or (row - col) in diag1 or (row + col) in diag2:
                    continue

                # Place queen
                board[row][col] = "Q"
                cols.add(col)
                diag1.add(row - col)
                diag2.add(row + col)

                backtrack(row + 1)

                # Backtrack
```

```
                board[row][col] = "."
                cols.remove(col)
                diag1.remove(row - col)
                diag2.remove(row + col)

        backtrack(0)
        return res
```

8. Time & Space Complexity

Complexity	Value	Reason
Time	O(n!)	Backtracking tree
Space	O(n)** + result	board + diagonal sets

This is expected for recursive backtracking.

9. Dry Run (n = 4)

Board positions are tried in this order:

Row 0: try col 0 1 2 3

Row 1: valid placements depend on row 0

Row 2: skip attacks

Row 3: final placements → store solution

Valid boards eventually found:

Solution 1

```
. Q . .
. . . Q
Q . . .
. . Q .
```

Solution 2

```
. . Q .
Q . . .
. . . Q
. Q . .
```

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds
Backtracking with Constraints	Placement problems, chess puzzles, Sudoku	Not removing items from sets, not copying board, wrong diag conditions	LC 52 (Count N-Queens), Sudoku Solver, Word Search

11. Common Mistakes

- Incorrect diagonal formulas
- Missing backtracking removal
- Mutating board without making string copy
- Using ["."] * n incorrectly (same row reference)

12. Variations / Follow-Ups

- **LC 52** — Count N-Queens solutions
- Sudoku Solver
- M-Coloring graph problem
- Knight’s tour
- N-Rooks / N-Bishops variants