**Report on N-Queens Problem Implementation**

**Introduction**

The N-Queens problem is a classic combinatorial problem in which the goal is to place N queens on an N×N chessboard in such a way that no two queens attack each other. This means that no two queens can be in the same row, column, or diagonal. This report explains the implementation details of the provided NQueens.py script.

**Implementation Overview**

The script uses a backtracking approach to find all valid solutions to the N-Queens problem for a given board size NN. It ensures that the constraints of the problem are satisfied at each step before proceeding further.

**Explanation of Functions**

**1. print\_solution(board, N)**

This function prints the chessboard configuration where queens are placed. The board is represented as a 2D list, with 1 indicating a queen's presence and 0 indicating an empty space. The function prints Q for queens and . for empty spaces.

**2. is\_safe(board, row, col, left\_row, upper\_diag, lower\_diag, N)**

This function checks whether a queen can be placed in a given row and column. It ensures that:

* No other queen exists in the same row.
* No other queen exists in the upper diagonal (i.e., diagonals sloping from left to right).
* No other queen exists in the lower diagonal (i.e., diagonals sloping from right to left). It returns True if it is safe to place a queen, otherwise False.

**3. solve\_n\_queens(board, col, left\_row, upper\_diag, lower\_diag, N, solutions)**

This recursive function attempts to place queens column by column. If a valid position is found in a column, it marks the placement and proceeds to the next column. If all columns are filled successfully, the solution is stored in the solutions list. Otherwise, it backtracks by removing the last placed queen and trying other positions.

**4. n\_queens(N)**

This function initializes the chessboard and the helper arrays that track occupied rows and diagonals:

* left\_row: Tracks occupied rows.
* upper\_diag: Tracks occupied upper diagonals.
* lower\_diag: Tracks occupied lower diagonals. It then calls solve\_n\_queens to find all valid solutions and prints them along with the total count of solutions.

**Execution Flow**

1. The script initializes an N×NN \times N board with zeros.
2. It starts placing queens column by column using backtracking.
3. If a solution is found, it is stored and displayed.
4. Once all solutions are found, the total number of valid solutions is printed.

**Example Output (for N=8)**

Q . . . . . . .

. . . . Q . . .

. . . . . . . Q

. . . . . Q . .

. . Q . . . . .

. . . . . . Q .

. Q . . . . . .

. . . Q . . . .

Total solutions: 92

**Conclusion**

This script efficiently finds all valid solutions to the N-Queens problem using backtracking. The approach ensures that each solution follows the problem constraints while optimizing performance using boolean arrays for fast lookups.

**Output:**

