```python
result = []
n = int(input("Enter size of board: "))
N = n



def isValid(board, row, col):
    # Check if there's a queen in the same column
    for i in range(row - 1, -1, -1):
        if board[i][col] == "Q":
            return False

    # Check upper-left diagonal
    i, j = row - 1, col - 1
    while i >= 0 and j >= 0:
        if board[i][j] == "Q":
            return False
        i -= 1
        j -= 1

    # Check upper-right diagonal
    i, j = row - 1, col + 1
    while i >= 0 and j < N:
        if board[i][j] == "Q":
            return False
        i -= 1
        j += 1

    return True

# *************************************************************************************** #

def solve(board, row):
    # Base case: If all rows are filled, append the current board configuration to the result
    if row >= N:
        result.append([''.join(row) for row in board])  # Convert each row to a string and append to result
        return

    # Recursive case: Try placing a queen in each column of the current row
    for col in range(N):
        # Check if placing a queen at the current position is valid
        if isValid(board, row, col):
            # If valid, place the queen and move to the next row
            board[row][col] = "Q"
            solve(board, row + 1)
            # Backtrack: Remove the queen from the current position (For finding next possible solution)
            board[row][col] = "."


# *************************************************************************************** #

# Initialize the board(2-D) with '.' for empty cells
board = [["." for _ in range(N)] for _ in range(N)]

# Start solving the N-Queens problem
solve(board, 0)

# *************************************************************************************** #


# Print all solutions
print("Number of solutions:", len(result))
for idx, solution in enumerate(result):
    print(f"Solution {idx + 1}:")
    print('\n'.join(solution))
    print()
```