```
def is_safe(board, row, col, N):
    # Check if there is a queen in the same column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check if there is a queen in the upper left diagonal
    i = row - 1
    j = col - 1
    while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return False
        i -= 1
        j -= 1

    # Check if there is a queen in the upper right diagonal
    i = row - 1
    j = col + 1
    while i >= 0 and j < N:
        if board[i][j] == 1:
            return False
        i -= 1
        j += 1

    return True

def solve_n_queens_util(board, row, N):
    if row == N:
        # All queens have been successfully placed
        # Print the board or store the solution
        print_board(board, N)
        return True

    # Try placing a queen in each column of the current row
    for col in range(N):
        if is_safe(board, row, col, N):
```

```
            board[row][col] = 1  # Place the queen

            # Recursively check for the next row
            if solve_n_queens_util(board, row + 1, N):
                return True

            board[row][col] = 0  # Backtrack if the solution is not found

    return False

def solve_n_queens(N):
    # Create an empty board
    board = [[0] * N for _ in range(N)]

    if not solve_n_queens_util(board, 0, N):
        print(f"No solution exists for N = {N}")

def print_board(board, N):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print("Q ", end="")
            else:
                print(". ", end="")
        print()
    print()

# Example usage
N = 4
solve_n_queens(N)
```