# N-Queens Problem - Explanation and Output

## 1. Explanation of the Code

The N-Queens problem is a backtracking algorithm used to place N queens on an N×N chessboard that no two queens attack each other. The algorithm follows these steps:  
1. Start from the first row and try to place a queen in each column.  
2. Before placing a queen, check if it is safe using the `is\_safe` function.  
3. If it is safe, place the queen and move to the next row.  
4. If all queens are placed successfully, store the solution.  
5. If a placement leads to a dead-end, backtrack and try the next possibility.

## 2. Code:

def is\_safe(board, row, col, n):  
 for i in range(row):  
 if board[i][col] == 'q':  
 return False  
  
 for i, j in zip(range(row - 1, -1, -1), range(col - 1, -1, -1)):  
 if board[i][j] == 'q':  
 return False  
  
 for i, j in zip(range(row - 1, -1, -1), range(col + 1, n)):  
 if board[i][j] == 'q':  
 return False  
  
 return True  
  
def solve\_n\_queens(board, row, n, solutions):  
 if row == n:  
 solutions.append([''.join(row) for row in board])  
 return  
  
 for col in range(n):  
 if is\_safe(board, row, col, n):  
 board[row][col] = 'q'  
 solve\_n\_queens(board, row + 1, n, solutions)  
 board[row][col] = '.'   
  
def print\_board(board):  
 for row in board:  
 print('|' + '|'.join(row) + '|')  
 print()  
  
def main():  
 n = int(input("Enter the size of the chessboard: "))  
 board = [['.' for \_ in range(n)] for \_ in range(n)]  
 solutions = []  
 solve\_n\_queens(board, 0, n, solutions)  
 print\_board(solutions[0])  
   
main()

## 3. Screenshot

