# N-Queens Problem using Backtracking

**Explanation of the N-Queens Solver using Backtracking**  
  
This Python code solves the N-Queens problem using backtracking . The N-Queens problem requires placing `N` queens on an `N × N` chessboard so that no two queens threaten each other.  
  
  
 **1. is\_safe(board, row, col)` - Checking if a Queen can be Placed**   
This function determines whether placing a queen at `(row, col)` is safe.  
  
 **A. Checking the Left Side of the Current Row**   
Since we place one queen per column , we only need to check leftward in the same row .   
If there's already a queen, the function returns `False`, meaning the position is unsafe.  
  
 **B. Checking the Upper-Left Diagonal**   
A queen attacks diagonally. We must check the upper-left diagonal   
The function moves diagonally up and left , checking if a queen exists. If so, it returns `False`.  
  
 **C. Checking the Lower-Left Diagonal**   
Similarly, we check the lower-left diagonal.   
The function moves down and left , ensuring no queens exist in this diagonal.  
  
 **D. If No Conflicts, Placement is Safe**   
If none of the above checks return `False` , the position `(row, col)` is safe, and the function returns `True`.  
  
  
  
 **2. `solve\_n\_queens\_util(board, col)` - Placing Queens using Recursion**   
This function tries to place a queen in each column and calls itself recursively.  
  
 **A. Stopping Condition**   
If all columns are filled, a solution is found, and the function returns `True`.  
  
 **B. Trying to Place a Queen in Each Row**   
For each row, the function checks if placing a queen in that row at the current column is safe.   
If safe, the queen is placed.  
  
 **C. Recurring to the Next Column**   
The function then recursively tries to place a queen in the next column .   
If successful, it returns `True`, indicating the solution has been found.  
  
 **D. Backtracking (Undo the Last Move)**   
If placing a queen leads to an unsolvable state, the function removes the last placed queen and tries the next possible row.   
This process is called backtracking and helps explore other possible placements.  
  
  
  
 **3. `solve\_n\_queens(n)` - Driver Function**  
This function initializes the board and starts solving the problem.  
  
 **A. Creating the Chessboard**   
An `N × N` empty chessboard is created, where `0` represents an empty space.  
  
 **B. Calling the Recursive Solver**   
The function starts solving from column 0 .   
If no solution is found, it prints `"Solution does not exist"`.  
  
 **C. Printing the Solution**   
Once a valid placement of `N` queens is found, the function prints the chessboard, replacing `1`s with `"Q"` and `0`s with `"."`.  
  
  
 **4. Running the Program**

