Pseudocode:

```
def IsSolution(A, k, S)
        return k == Length(A)     # every queen was placed

def ConstructCandidates(A, k, S):
If previous rows (row 1 to k – 1) have queens attacking position of new queen vertically
return false
If previous rows diagonally (in the \ direction) have previous queens attacking
return false
If previous rows diagonally (in the / direction) have previous queens attacking
return false
Return true if current position of new queen has no previously placed queens attacking

def Process(A, k, S)
        Count the number of solutions in a global variable count.
        Print(A)

def IsFinished()
        return false # Assuming we want to find any ALL (total) solutions

def Backtrack(A, k, S)
        if IsSolution(A, k, S):
                Process(A, k, S)
        else:
        L = ConstructCandidates(A, k, S)
        If position L is a candidate:
                Place queen
                Backtrack(A, k + 1, S) # recursively find more solutions with the current placement of
                                       queen
```

```
Remove queen (backtrack to find more solutions)
if Finished():
    Return
```

Analysis:

The algorithm starts from the first row and places a Queen in each square of the first row and recursively checks the remaining rows to see if this leads to a solution or not. If the current configuration does not result in a solution, the algorithm will backtrack and find a configuration of queens that works. Squares are ignored if two queens can attack each other. The time complexity of this algorithm is exponential, O(n^n). It checks every placement of queens that is possible in every single square.

Implementation:

See .py file

Test cases:

See .py file (Ran algorithm with N=8, N=10, N=12)

Findings:

N	NUMBER OF SOLUTIONS	NUMBER OF RECURSIVE CALLS
8	92	2057
10	724	35539
12	14200	856189

Based on the data collected from the table, it appears that increasing the board size slightly raises the number of solutions and recursive calls exponentially. This aligns with the analysis of this algorithm, and that it is exponential. As the input size (board size) increases, the number of solutions also increases exponentially as there are more combinations available. Since the run time of the algorithm is checking every single position that is possible for queens against every combination of other positions, it makes sense that as the board size grows the number of solutions grows in a similar fashion (in this case, exponentially).

As is with the number of solutions, the number of recursive calls will grow exponentially as well as the board size increases. The reasoning is the same, as the board size grows the number of combinations of positions of queens grows exponentially as well since more combinations are possible with a larger board. All these positions and combinations will be checked recursively, and results in an exponential growth in recursive calls.