Summary

The time complexity of the solution is equal to the number of balanced parentheses combinations possible, which is equivalent to the nth Catalan Number.
Hence the complexity is O((1 / (n + 1)) * (2n C n)) which is approximated to **O((4 ^ n) / sqrt(n))**.
Read more about Catalan Numbers and its applications in combinatorial problems at https://cp-algorithms.com/combinatorics/catalan-numbers.html
The space complexity is the same as the time complexity.

Problem Pattern

1. Given the number of pairs of parentheses, we have to generate all possible combinations of well-formed parentheses.
2. For eg, if n = 2, out output should be
   a. ( ) ( )
   b. (( ))
3. We can create a function which generates all the 2 ^ (2 * n) possibilities, and checks whether each possibility is correct or not.

- The complexity of such a solution will be O(2 ^ (2 * n)) for generation, and then checking each possibility shall take an extra O(n), hence the total complexity will be O((2 ^ (2 * n)) * n).
- Can we do better than this?

Problem Approach

1. We can create a backtracking algorithm which generates all of the correct possibilities, and never backtracks for unbalanced possibilities.
2. Let's say the function is backtrack(S, left, right), in which S is the currently generated string, left is the number of opening parenthesis we have already added, and right is the same for the closing parenthesis.

- Now we shall only iterate for cases where the bracketing will always be balanced.
- The way we do this is by checking the following conditions:

a. if the current number of open parenthesis is less than n, then we can recurse for S + '('.
b. if the current number of closing parenthesis is less than the number of opening parenthesis, then we can recurse for S + ')'.

Problem Pseudocode

```
generateParenthesis(N):
    ans = []
    backtrack('', 0, 0, N, ans)
    return ans


backtrack(S, left, right, N, ans):
    if len(S) == 2 * N:
        ans.append(S)
        return
    if left < N:
        backtrack(S+'(', left+1, right, N, ans)
    if right < left:
        backtrack(S+')', left, right+1, N, ans)
```