[Click on the logo to find **Problem Statement**]

Intuition

This code implements a backtracking algorithm to generate all valid combinations of parentheses given a number, `n`, representing the pairs of parentheses. The `backtrack` function explores possible combinations by adding open and close parentheses to a stack and recursively branching into two scenarios: adding an open parenthesis if the count of open parentheses is less than `n`, and adding a close parenthesis if the count of close parentheses is less than the count of open parentheses. The base case is reached when the counts of open and close parentheses both equal `n`, at which point the current combination is added to the result list (`res`). The algorithm systematically explores different combinations, and the final result is a list of all valid parenthetical expressions. The key idea is to use backtracking to efficiently explore the solution space, considering different choices and undoing them as needed, to generate all possible valid combinations of parentheses.

Approach

1. **Base Case:**
    - The base case checks if both the counts of open and close parentheses are equal to `n`. If so, it means a valid combination has been formed, and it is added to the result list (`res`).
2. **Recursive Exploration:**
    - The algorithm explores two recursive scenarios:
        - If the count of open parentheses is less than `n`, an open parenthesis is added to the stack, and the function is called recursively with an incremented count of open parentheses.
        - If the count of close parentheses is less than the count of open parentheses, a close parenthesis is added to the stack, and the function is called recursively with an incremented count of close parentheses.
3. **Backtracking:**
    - After exploring each recursive scenario, the last parenthesis added to the stack is removed to revert the state, allowing the algorithm to explore other possibilities.
4. **Initialization:**

- The process starts with an initial call to `backtrack(0, 0)` with both counts set to zero.

5. **Result:**
   - The final result is the list of all valid combinations of parentheses stored in the `res` list.

Code

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        stack = []
        res = []
        def backtrack(openN, closeN):
            if openN ==  closeN == n:
                res.append("".join(stack))
                return

            if openN < n:
                stack.append("(")
                backtrack(openN + 1, closeN)
                stack.pop()

            if closeN < openN:
                stack.append(")")
                backtrack(openN, closeN + 1)
                stack.pop()

        backtrack(0, 0)
        return res
```

If you want to see more solutions to coding problems, you can visit: