Documentation: Longest Valid Parentheses

Problem Statement:

Given a string `s` containing only '(' and ')' characters, the task is to find the length of the longest valid (well-formed) parentheses substring.

Function Signature:

def longestValidParentheses(self, s: str) -> int:

Parameters:

`s`: A string consisting of '(' and ')' characters.

Return Value:

-An integer representing the length of the longest valid parentheses substring.

Approach:

1. **Stack Implementation:**
    a. We utilize a stack to keep track of indices of the opening parentheses.
    b. Additionally, we maintain a variable `max_length` to store the length of the longest valid substring encountered so far.
    c. Another variable `start_index` is used to keep track of the starting index of the current substring being evaluated.

2. **Iteration:**
    a. Iterate through each character in the input string `s`.
    b. If the current character is '(', push its index onto the stack.
    c. If the current character is ')':
    d. If the stack is not empty, pop the topmost index from the stack.
    e. If the stack is still not empty after popping, update `max_length` by calculating the length of the valid substring from the current index to the index at the top of the stack.
    f. If the stack becomes empty after popping, update `max_length` by calculating the length of the valid substring from the current index to `start_index`.
    g. Update `start_index` to the current index if the stack becomes empty.

3. **Return:**
    a. After iterating through the entire string, return the final value of `max_length`.

Examples:

Example 1:

- Input: `s = "(()"`

- Output: `2`

- Explanation: The longest valid parentheses substring is "()".

Example 2:

- Input: `s = ")()())"`

- Output: `4`

- Explanation: The longest valid parentheses substring is "()()".

Example 3:

- Input: `s = ""`

- Output: `0`

Explanation: Since the input string is empty, there are no valid parentheses substrings, so the output is `0`.

Constraints:

- `0 <= s.length <= 3 * 10^4`

- Each character in `s` is either '(' or ')'.

Complexity Analysis:

1. **Time Complexity:** O(n), where n is the length of the input string `s`. The algorithm involves a single pass through the string.
2. **Space Complexity:** O(n), where n is the length of the input string `s`. The stack can potentially store all indices of opening parentheses.