Minimum Bracket Swap to make string-balanced

LeetCode

Given a string containing square brackets **[]** that may be unbalanced, implement a function to calculate the minimum number of swaps needed to balance the string. A swap is defined as changing a pair of adjacent brackets from **][** to **[]**.

Example

Consider the following unbalanced string:

String: "]]][[["

Output

The minimum number of swaps needed to balance the string: 2

Approach 1: Minimum bracket reversal to make string balanced by using stack

In this approach, a stack is used to keep track of the brackets. For each character in the string:

- If an opening square bracket **[** is encountered, it is pushed onto the stack.
- If a closing square bracket **]** is encountered and it matches with the top of the stack (which indicates a pair of balanced brackets), the opening bracket is popped from the stack.
- If the closing bracket does not match with the top of the stack, it indicates an unbalanced pair. We need to swap the current bracket with an adjacent bracket to create a balanced pair.

Count the number of such unbalanced pairs and calculate the minimum number of swaps required to balance the string.

Steps:

1. Initialize an empty stack and a variable to count swaps (**swap**).
2. For each character in the string:
    - If the character is an opening square bracket **[**, push it onto the stack.
    - If the character is a closing square bracket **]**:
        - If the stack is not empty and the top element is an opening bracket **[**, pop the opening bracket.
        - Otherwise, push the closing bracket onto the stack and increment the **swap** count.

3. Calculate the minimum number of swaps required by counting the number of unbalanced pairs.

Time Complexity:

- The approach involves iterating through each character of the string once.
- The stack operations (push, pop) are constant time.
- The count calculation at the end is also linear.
- **Time Complexity**: **O(n), where n is the length of the string.**

Space Complexity:

- The space complexity is determined by the stack's maximum size, which depends on the number of unbalanced brackets.
- In the worst case, the stack could hold all unbalanced brackets.
- **Space Complexity: O(n), where n is the length of the string.**