Minimum Bracket Reversal to make String Balanced CodeStudio

Given a string containing curly braces **{}** that may be unbalanced, implement a function to calculate the minimum number of bracket reversals needed to balance the string. A reversal is defined as changing a pair of adjacent brackets from **{}** to **{}**.

Example

Consider the following unbalanced string:

String: "}{{}}{{{"

Output

The minimum number of bracket reversals required to balance the string: 3

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 curly brace **{** is encountered, it is pushed onto the stack.
- If a closing curly brace **}** is encountered, and it matches with the top of the stack (which indicates a pair of balanced brackets), the opening brace is popped from the stack.
- If the closing brace does not match with the top of the stack, it is pushed onto the stack.

After processing the entire string, the stack may contain some unbalanced brackets. Count the number of opening and closing brackets in the stack and calculate the minimum number of reversals needed to balance the string.

Steps:

1. Initialize an empty stack.
2. For each character in the string:
    - If the character is an opening brace **{**, push it onto the stack.
    - If the character is a closing brace **}**:
        - If the stack is not empty and the top element is an opening brace **{**, pop the opening brace.
        - Otherwise, push the closing brace onto the stack.
3. After processing the string, count the remaining unbalanced brackets in the stack.

4. Calculate the minimum number of reversals required based on the counts.

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 opening brackets.
- In the worst case, the stack could hold all opening brackets.
- **Space Complexity**: **O(n), where n is the length of the string**