Intuition of this Problem:

The implementation uses two pointers, i and j, to traverse the character array. The variable i is used to iterate over the array, while j is used to keep track of the position to which the compressed character needs to be written. The variable prev is used to store the previous character encountered, and count is used to keep track of the number of consecutive occurrences of the current character.

Approach for this Problem:

1. Initialize two pointers, i and j, to 0. Pointer i will be used to traverse the input array, while pointer j will be used to modify the array in place.
2. While i is less than the length of the input array, do the following:
    - a. Initialize a variable count to 1. This variable will keep track of the number of consecutive repeating characters in the input array.
    - b. While i is less than the length of the input array minus 1, and the character at index i is equal to the character at index i+1, increment count and i.
    - c. Write the character at index i to the output array at index j, and increment both i and j.
    - d. If count is greater than 1, convert it to a string and write each character of the string to the output array at index j, incrementing j for each character.
3. Return j, which is the length of the compressed array.

Time Complexity and Space Complexity:

- **Time complexity**: **O(n)**, where n is the length of the input character array. This is because we traverse the array only once, and the operations within the loop take constant time.

- **Space complexity**: **O(1)**, i.e. constant extra space. This is because we are modifying the input array in place, without using any additional data structures. The only extra space used is the variables i, j, prev, and count, which all take constant space.
    - Note that the conversion of the count of consecutive characters to a string may take up to O(log(count)) space, but this is still considered constant space as the maximum length of the string is bounded by a constant (i.e., 10). Therefore, the overall space complexity of the algorithm is O(1).