```
Alg getLongestBalancedSubstringRecursive(String S, int startIndex)

    maxLength <- 0
    if startIndex >= s.length()
        return 0

    count1 <- 0
    count2 <- 0

    a <- S[startIndex]
    b = ‘ ‘

    for I <- startIndex to s.length()
        char c = s[i]
        if c == a
            count1 = count1 + 1
            if count1 == count2 and maxLength < count1 * 2
                maxLength <- count1 * 2
        else
            If b == ‘ ‘
                b <- c
                count2 = count2 + 1
                if count1 == count2 and maxLength < count1 * 2
                    maxLength <- count1 * 2
            else if c == b
                count2++
                if count1 == count2 and maxLength < count1 * 2
                    maxLength = count1 * 2
            else
                break
    recursiveMaxLength <- getLongestBalancedSubstringRecursive(S, startIndex + 1)
    if maxLength < recursiveMaxLength
        maxLength <- recursiveMaxLength
    return maxLength
```

algorithm analysis:
the first loop iterates over all the element of the string starting from the startIndex to the end, in worst case it will loop over all the characters in the string,
the assignment and comparasons inside the loop are constant time O(1)
so time complexity of the for loop is O(n)

the recursive call takes the starIndex+1 and same string so its time complexity is T(n-1)

$T(n) = T(n-1) + O(n)$

using iteration method

$T(n) = T(n-1) + O(n)$
$T(n-1) = T(n-2) + O(n-1)$
$T(n-2) = T(n-3) + O(n-2)$

base case: $T(1) = O(1)$

$T(n) = O(n) + O(n-1) + O(n-2) + ... + O(1) + T(1)$

this form is similar to the arithmatic sequence so:

$O((n*(n+1))/2)$
$T(n) = T(1) + O((n*(n+1))/2)$
$T(n) = T(1) + O((n^2+n)/2$

so time complexity of this algorithm is $O(n^2)$