3-Balanced String

	NAME	*ID*
1	يوسف احمد عبدالمرضي امام	20211058
2	يوسف مرزوق يوسف صالح	20211107
3	يوسف محمد عبدالعظيم عبدالغني	*20211099*
4	يوسف علي جمعه رمضان	*20211085*
5	محمد ناصر ابوالسعود محمد	*20210840*
6	محمد معتز محمد شعبان	*20210838*

pseudocode representation of the (non-recursive) algorithm:

```
algorithm is_balanced(substr) \\ for check if string is balanced or not
   count <- [0, 0]
   for i <- 0 to length(substr)-1 do{
      if substr[i] == substr[0] then
         count[0]++
      else
         count[1]++
if (count[0] == count[1]) then return 1
return 0

algorithm longest_balanced_substring(s) \\ for get longest balanced substring

   n <- length(s)
   max_length <- 0
   for i <- 0 to n-2 do
      for j <- i+2 to n do
         substr <- s[i:j]  // Extract the substring }
    if is_balanced(substr) and length(substr) > max_length then
            max_length <- length(substr)  // Update the maximum length
    return max_length
```

Analysis of this algorithm:

The outer loop of the **longest_balanced_substring** algorithm iterates over all possible starting indices **i** in the input string, from 0 to **n-2**. This takes O(n) time.

The inner loop iterates over all possible ending indices **j** in the input string, starting from **i+2** and going up to **n**. This also takes O(n) time.

For each pair of indices **(i, j)**, the algorithm extracts a substring using **strncpy**, which takes O(j-i) time. This is the time required to copy the characters from the input string to the **substr** array. Since there are at most O(n^2) pairs of indices **(i, j)** to consider, the total time taken for all substring extractions is O(n^3).

Finally, the **is_balanced** algorithm is called on each extracted substring, which takes O(k) time, where **k** is the length of the substring. Since the total length of all extracted substrings is O(n^3), the total time taken by **is_balanced** is O(n^3) as well.

Adding up these time complexities, we get a total time complexity of O(n^3) for the entire algorithm.

```c
#include <stdio.h>
#include <string.h>
int is_balanced(char* substr)
{
    int count[2] = {0};
    for(int i = 0; i < strlen(substr); i++)
    {
        if(substr[i] == substr[0])
            count[0]++;
        else
            count[1]++;
    }
    return (count[0] == count[1]);
}
int longest_balanced_substring(char* s)
{
    int n = strlen(s);
    int max_length = 0;
    for(int i = 0; i < n-1; i++)
    {
        for(int j = i+2; j <= n; j++)
        {
            char substr[j-i];
            strncpy(substr, &s[i], j-i);
            substr[j-i] = '\0';
            if(is_balanced(substr) && strlen(substr) > max_length)
                max_length = strlen(substr);
        }
    }
    return max_length;
}
int main()
{
    printf("this code run by non-recursive\n");
    printf("----------------------------------------\n");
    char s1[] = "cabbacc";
    printf("the longest balanced substring for ( %s ) = %d\n",s1,longest_balanced_substring(s1));  // Expected Output: 4

    char s2[] = "abababa";
    printf("the longest balanced substring for ( %s ) = %d\n",s2,longest_balanced_substring(s2));  // Expected Output: 6

    char s3[] = "aaaaaaa";
    printf("the longest balanced substring for ( %s ) = %d\n",s3,longest_balanced_substring(s3));  // Expected Output: 0
```

```
"C:\Users\youssef Ahmed\De:
this code run by non-recursive
----------------------------------------
the longest balanced substring for ( cabbacc ) = 4
the longest balanced substring for ( abababa ) = 6
the longest balanced substring for ( aaaaaaa ) = 0

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

pseudocode representation of the (Recursive) algorithm:

```
algorithm is_balanced(freq)
  count <- 0
   diff_chars <- 0
   FOR i <- 0 to 25   Do
      IF freq[i] > 0 THEN
         diff_chars <- diff_chars + 1
      IF freq[i] <- freq[0] AND freq[i] > 0 THEN
         count <- count + 1
   IF diff_chars == 2 AND count == 2 THEN
      RETURN 1
   ELSE
      RETURN 0
algorithm longest_balanced_substring (s, start, end)
   IF end - start + 1 < 2 THEN
      RETURN 0
   freq [26] <- {0}
   FOR i <- start to end    Do
      freq[s[i] - 'a'] <- freq[s[i] - 'a'] + 1
   IF is_balanced(freq) THEN
      RETURN end - start + 1
   len1 <-longest_balanced_substring (s, start, end - 1)
   len2<- longest_balanced_substring (s, start + 1, end)
   IF len1 > len2 THEN
      RETURN len1
```

```
ELSE
    RETURN len2
algorithm longest_balanced_substring(s)
RETURN longest_balanced_substring (s, 0,length(s) - 1)
```

Analysis of this algorithm:

The time complexity of the **longest_balanced_substring** algorithm is O(n log n) because it uses a divide-and-conquer approach that involves recursively dividing the input string into two halves and then combining the results of these subproblems.

At each level of recursion, the algorithm splits the input string into two substrings of length n/2, where n is the length of the original input string. The algorithm then recursively calls itself on each of these substrings, which results in a binary tree of recursive calls with a height of log n.

At each level of recursion, the algorithm computes the frequency of each character in the substring and checks if it is balanced using the **is_balanced** function, which takes O(n) time in the worst case, where n is the length of the substring. Therefore, the total time complexity of the algorithm is the product of the number of levels in the recursive tree and the time complexity of each level, which is O(log n * n) = O(n log n).

```c
#include <stdio.h>
#include <string.h>
int is_balanced(int* freq) {
    int i;
    int count = 0;
    int diff_chars = 0;
    for (i = 0; i < 26; i++) {
        if (freq[i] > 0) diff_chars++;
        if (freq[i] == freq[0] && freq[i] > 0) count++;
    }
    if (diff_chars == 2 && count == 2) return 1;
    return 0;
}
int longest_balanced_substring_helper(char* s, int start, int end) {
    if (end - start + 1 < 2) {
        return 0; // substring too short to be balanced
    }
    int freq[26] = {0};
    int i;
    for (i = start; i <= end; i++) {
        freq[s[i] - 'a']++;
    }
    if (is_balanced(freq)) {
        return end - start + 1; // this substring is balanced
    }
    int len1 = longest_balanced_substring_helper(s, start, end - 1);
    int len2 = longest_balanced_substring_helper(s, start + 1, end);
    return (len1 > len2) ? len1 : len2; // return the longer of the two balanced substrings
}
int longest_balanced_substring(char* s) {
    return longest_balanced_substring_helper(s, 0, strlen(s) - 1);
}
int main() {
     printf("this code run by Recursive\n");
    printf("---------------------------------------\n");
    char s1[] = "cabbacc";
    printf("the longest balanced substring for ( %s ) = %d\n",s1 ,longest_balanced_substring(s1)); // Expected Output: 4

    char s2[] = "abababa";
    printf("the longest balanced substring for ( %s ) = %d\n",s2 ,longest_balanced_substring(s2)); // Expected Output: 6

    char s3[] = "aaaaaaa";
    printf("the longest balanced substring for ( %s ) = %d\n",s3 ,longest_balanced_substring(s3)); // Expected Output: 0
```

```
"C:\Users\youssef Ahmed\De
this code run by Recursive
---------------------------------------
the longest balanced substring for ( cabbacc ) = 4
the longest balanced substring for ( abababa ) = 6
the longest balanced substring for ( aaaaaaa ) = 0

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

	Recursive algorithm	*Non-Recursive algorithm*
Complexity	O(n log n)	O(n^3)
Understanding	more challenging to follow because of the recursion.	simpler and easier to understand

- The recursive algorithm has a lower time complexity than the non-recursive algorithm because the time complexity of the non-recursive is O(n^3) and the recursive is O(n log n). , so it is more efficient in terms of time complexity.