Leetcode 1358: Number of Substrings Containing All Three Characters

Given a string s consisting only of characters 'a', 'b' and 'c'. Return the number of substrings containing at least one occurrence of all these characters.

```
Example 1:
Input: s = "abcabc"
Output: 10

Example 2:
Input: s = "aaacb"
Output: 3

Example 3:
Input: s = "abc"
Output: 1
```

Brute Force Approach (O(n^3) or O(n^2)): Logic: 1. Generate all substrings of s. 2. For each substring, check if it contains 'a', 'b', and 'c'. 3. Count such substrings. Time Complexity: O(n^3) or O(n^2) with optimization.

```
C++ Code:
#include <bits/stdc++.h>
using namespace std;
int numberOfSubstrings(string s) {
    int n = s.size(), count = 0;
    for (int i = 0; i < n; i++) {
        vector<int> freq(3, 0);
        for (int j = i; j < n; j++) {
            freq[s[j] - 'a']++;
            if (freq[0] > 0 && freq[1] > 0 && freq[2] > 0)
                count++;
        }
    }
    return count;
}
int main() { cout << numberOfSubstrings("abcabc"); }

Java Code:
public class Main {
    public static int numberOfSubstrings(String s) {
        int n = s.length(), count = 0;
        for (int i = 0; i < n; i++) {
            int[] freq = new int[3];
            for (int j = i; j < n; j++) {
                freq[s.charAt(j) - 'a']++;
                if (freq[0] > 0 && freq[1] > 0 && freq[2] > 0)
                    count++;
            }
        }
        return count;
    }
    public static void main(String[] args) {
        System.out.println(numberOfSubstrings("abcabc"));
    }
}

Python Code:
def numberOfSubstrings(s):
    n = len(s)
    count = 0
    for i in range(n):
        freq = {"a": 0, "b": 0, "c": 0}
        for j in range(i, n):
            freq[s[j]] += 1
            if all(freq[c] > 0 for c in "abc"):
                count += 1
    return count
print(numberOfSubstrings("abcabc"))
```

Optimal Approach (O(n)) - Sliding Window / Two Pointer: Logic: 1. Track last seen indices of 'a', 'b', and 'c'. 2. When all three have been seen, the number of valid substrings ending at index i = min(lastA, lastB, lastC) + 1. 3. Sum these for all indices. Time Complexity: O(n) Space Complexity: O(1)

```
C++ Code:
#include <bits/stdc++.h>
using namespace std;
int numberOfSubstrings(string s) {
    int lastA = -1, lastB = -1, lastC = -1;
    int count = 0;
    for (int i = 0; i < s.size(); i++) {
        if (s[i] == 'a') lastA = i;
        if (s[i] == 'b') lastB = i;
        if (s[i] == 'c') lastC = i;
        int minIndex = min({lastA, lastB, lastC});
        if (minIndex != -1) count += minIndex + 1;
    }
    return count;
}
int main() { cout << numberOfSubstrings("abcabc"); }

Java Code:
public class Main {
    public static int numberOfSubstrings(String s) {
        int lastA = -1, lastB = -1, lastC = -1, count = 0;
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (ch == 'a') lastA = i;
            if (ch == 'b') lastB = i;
            if (ch == 'c') lastC = i;
            int minIndex = Math.min(lastA, Math.min(lastB, lastC));
            if (minIndex != -1)
                count += minIndex + 1;
        }
        return count;
    }
    public static void main(String[] args) {
        System.out.println(numberOfSubstrings("abcabc"));
    }
}

Python Code:
def numberOfSubstrings(s):
    last = {"a": -1, "b": -1, "c": -1}
    count = 0
    for i, ch in enumerate(s):
        last[ch] = i
        min_index = min(last.values())
        if min_index != -1:
            count += min_index + 1
    return count
print(numberOfSubstrings("abcabc"))

Summary:

| Approach | Technique | Time Complexity | Space Complexity |
|----------|-----------|-----------------|------------------|
| Brute Force | Generate all substrings | O(n²)–O(n³) | O(1) |
| Optimal | Sliding Window / Last Index | O(n) | O(1) |
```