2.Medium\07.Sum_of_beauty_of_all_substrings.cpp

```cpp
/*Question:

The beauty of a string is the difference in frequencies between the most frequent and least frequent characters.

Given a string `s`, you need to calculate the sum of beauty for all of its substrings. The beauty of a substring is defined as the difference between the highest and lowest frequency of any character in the substring.

Write a function `beautySum` that takes a string `s` as input and returns the sum of beauty for all substrings.

Example:

Input: s = "aabcb"
Output: 5
Explanation: The substrings with non-zero beauty are ["aab","aabc","aabcb","abcb","bcb"], each with beauty equal to 1.

Input: s = "aabcbaa"
Output: 17

Approach:

1. Initialize a variable `ans` to store the total beauty sum.
2. Iterate over the string `s` with the first loop, starting from index `i`.
   - Initialize a frequency array `freq` of size 26, initialized with zeros.
   - Iterate over the string `s` with the second loop, starting from index `j` equal to `i`.
      - Increment the frequency of the character `s[j]` in the `freq` array.
      - Calculate the difference between the highest and lowest frequencies in the `freq` array and add it to `ans`.
3. Return the value of `ans` as the final result.

CODE:-
*/
int get_maxmin(vector<int>& freq){
    int maxi = INT_MIN, mini = INT_MAX;
    for(auto it:freq){
        maxi = max(maxi,it);
        if(it!=0)
            mini = min(mini,it);
    }
    return (mini==INT_MAX)?0:maxi-mini;
}

int beautySum(string s) {
    int ans = 0;
    // 2 loops to generate all substrings
    for(int i=0; i<s.size(); i++){
        vector<int>freq(26,0);
        for(int j=i; j<s.size(); j++){
            freq[s[j]-'a']++;
            int maxmin = get_maxmin(freq);
            ans += maxmin;
        }
    }
    return ans;
}

/*
Time complexity :- for generating all substrings is O(n^2), where n is the length of the string `s`. For each substring, we calculate the difference between the highest and lowest frequencies, which takes O(26) or O(1) time since there are 26 lowercase alphabets. Therefore, the overall time complexity is O(n^2).
Space complexity :- O(26) or O(1) since we use a constant-sized frequency array to store the counts of characters.
*/

```