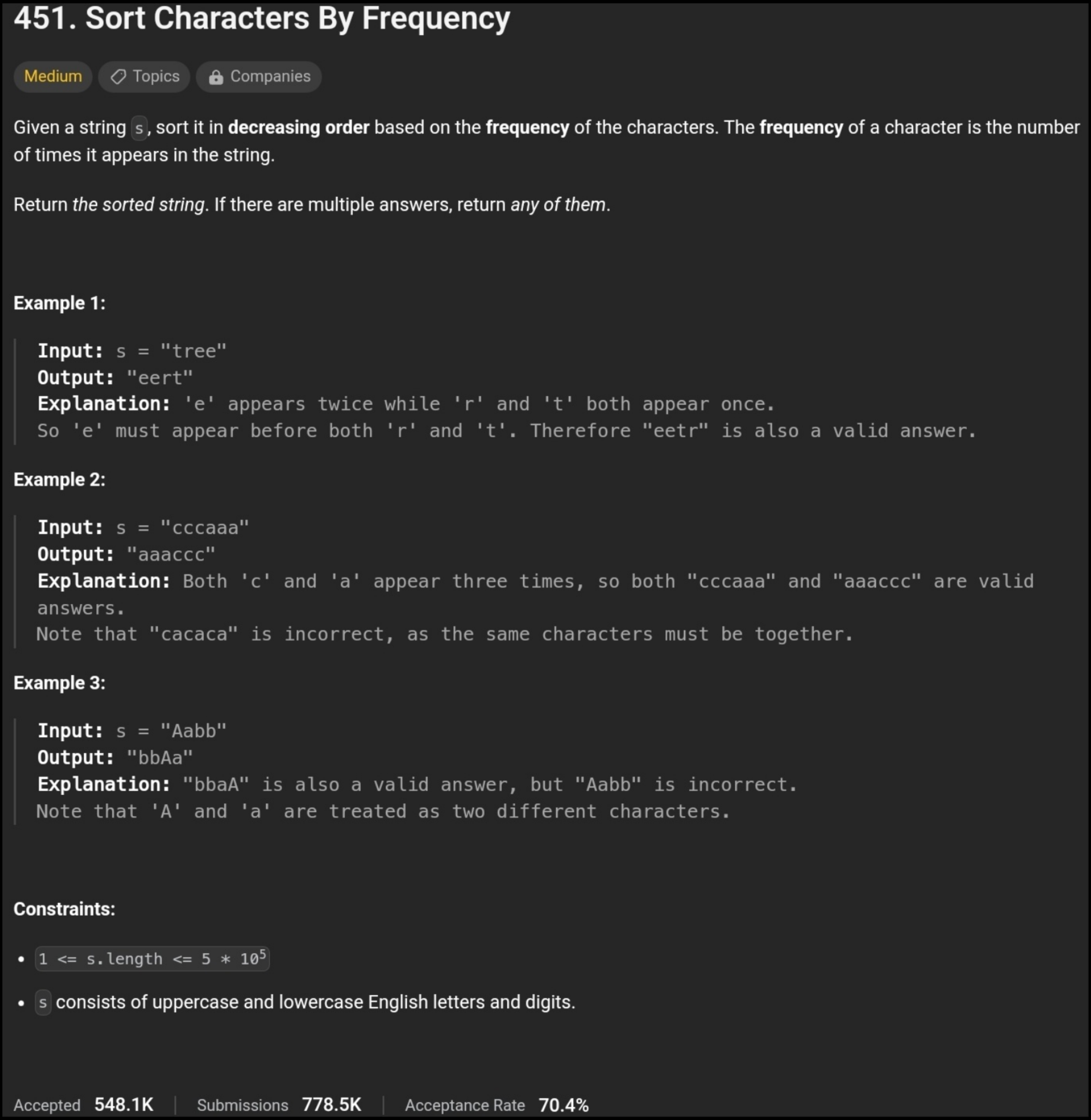


451. Sort Characters By Frequency

Given a string `s`, sort it in **decreasing order** based on the **frequency** of the characters. The **frequency** of a character is the number of times it appears in the string.

Return *the sorted string*. If there are multiple answers, return *any of them*.

Example 1:

```
Input: s = "tree"
Output: "eert"
Explanation: 'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
```

Example 2:

```
Input: s = "cccaaa"
Output: "aaaccc"
Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid
answers.
Note that "cacaca" is incorrect, as the same characters must be together.
```

Example 3:

```
Input: s = "Aabb"
Output: "bbAa"
Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

Constraints:

- `1 <= s.length <= 5 * 10`^{5}
- `s` consists of uppercase and lowercase English letters and digits.

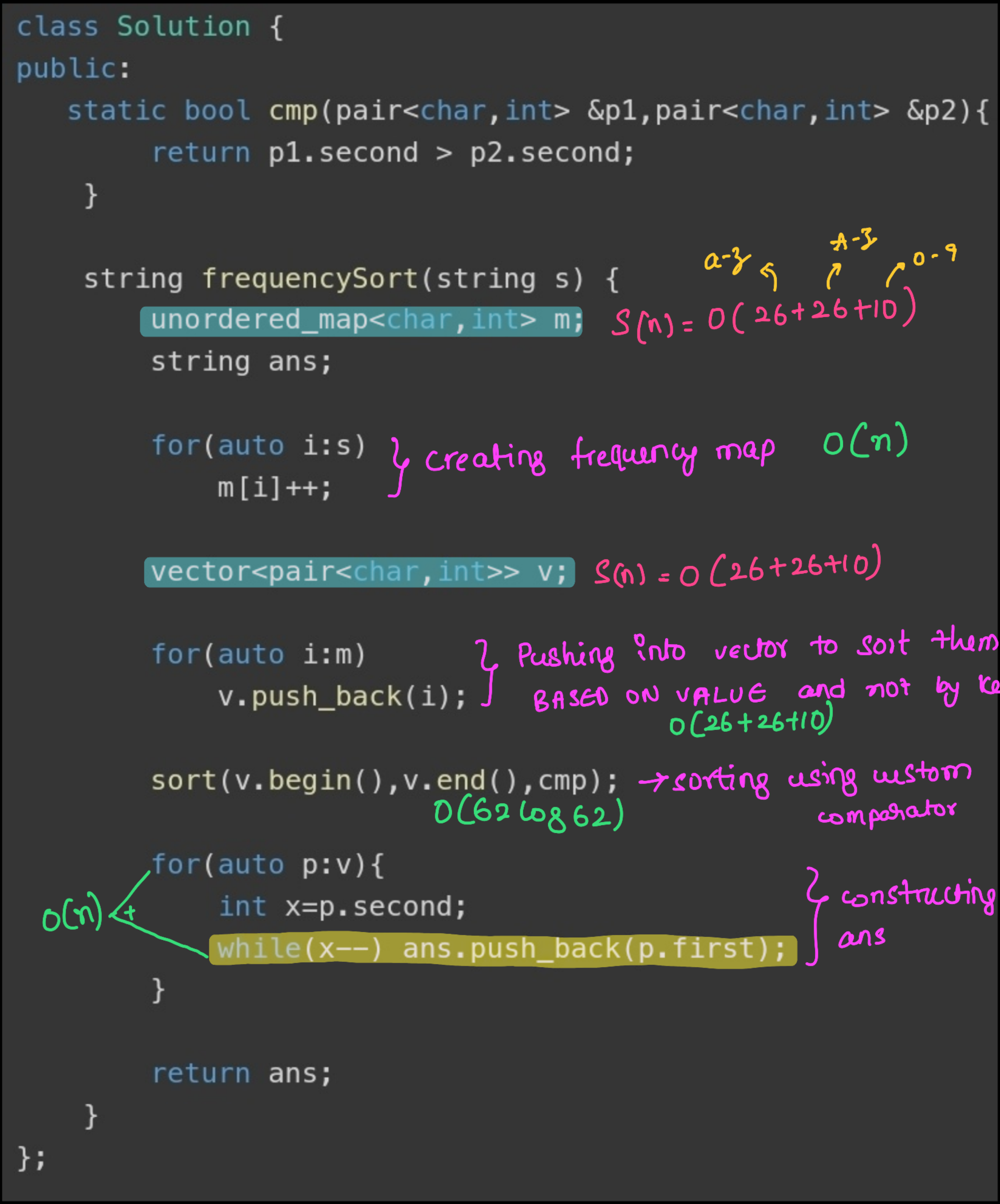
Accepted 548.1K | Submissions 778.5K | Acceptance Rate 70.4%

Approach 1:

→ Create a frequency map of characters of string.

→ now we need to sort them based on frequency in descending order.

→ we can do that by taking help of vector of pairs.



```cpp
class Solution {
public:
    static bool cmp(pair<char,int> &p1,pair<char,int> &p2){
        return p1.second > p2.second;
    }

    string frequencySort(string s) {
        unordered_map<char,int> m;
        string ans;

        for(auto i:s)
            m[i]++;

        vector<pair<char,int>> v;

        for(auto i:m)
            v.push_back(i);

        sort(v.begin(),v.end(),cmp);

        for(auto p:v){
            int x=p.second;
            while(x--) ans.push_back(p.first);
        }

        return ans;
    }
};
```

a-z A-Z 0-9

$S(n) = O(26+26+10)$

} creating frequency map $O(n)$

$S(n) = O(26+26+10)$

} Pushing into vector to sort them BASED ON VALUE and not by key

$O(26+26+10)$

→ sorting using custom comparator

$O(62 \log 62)$

$O(n)$

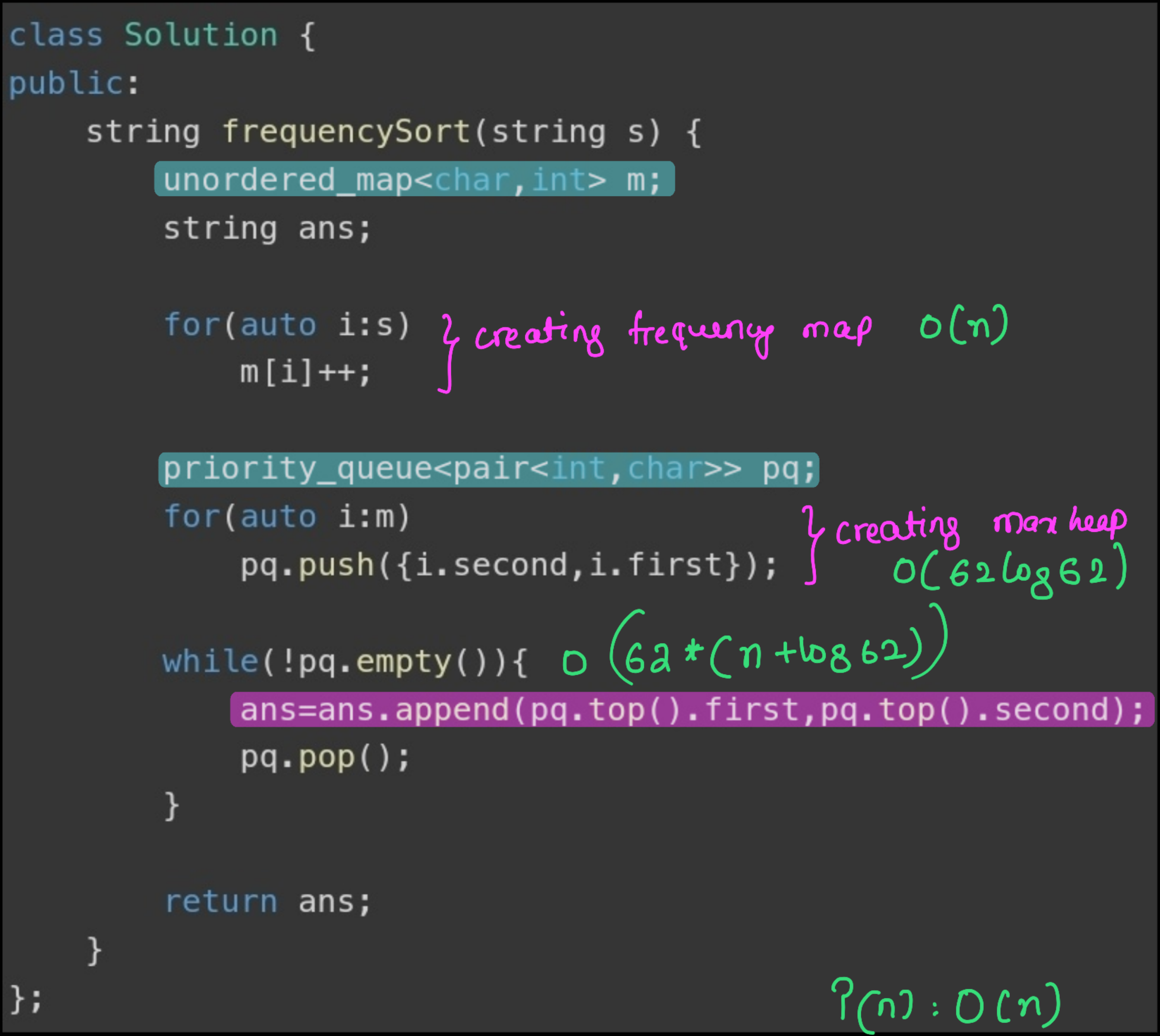
} constructing ans

$T(n) = O(n)$

$S(n) = O(62) + O(62)$

Approach 2: Using priority Queue

Same as above approach but here instead of custom sorting we directly use heaps.



```
class Solution {
public:
    string frequencySort(string s) {
        unordered_map<char,int> m;
        string ans;

        for(auto i:s)
            m[i]++;

        priority_queue<pair<int,char>> pq;
        for(auto i:m)
            pq.push({i.second,i.first});

        while(!pq.empty()){
            ans=ans.append(pq.top().first,pq.top().second);
            pq.pop();
        }

        return ans;
    }
};
```

} creating frequency map O(n)

} creating max heap O(62 log 62)

$O(62 * (n + \log 62))$

$T(n) : O(n)$

$S(n) : O(62) + O(62)$

append ( )

we can construct string with a character and its corresponding frequency

string s

$$s.\text{append}(\overset{\text{frequency}}{3},\ \overset{\text{char}}{'a'})$$

O(n)

s becomes "aaa"

string ( )

[illegible]

we can also use string( )

frequency

string s(3 , 'a') $\rightarrow$ char

cout<<S ;

gives "aaa"

O(n)