Problem 451: Sort Characters By Frequency

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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.

Code Snippets

C++:

```
class Solution {
public:
string frequencySort(string s) {

}
};
```

Java:

```
class Solution {
public String frequencySort(String s) {

}
}
```

Python3:

```
class Solution:
def frequencySort(self, s: str) -> str:
```

Python:

```
class Solution(object):
def frequencySort(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var frequencySort = function(s) {

};
```

TypeScript:

```
function frequencySort(s: string): string {

};
```

C#:

```
public class Solution {
public string FrequencySort(string s) {

}
}
```

C:

```
char* frequencySort(char* s) {

}
```

Go:

```
func frequencySort(s string) string {

}
```

Kotlin:

```
class Solution {
fun frequencySort(s: String): String {

}
}
```

Swift:

```
class Solution {
func frequencySort(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn frequency_sort(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def frequency_sort(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
```

```
*/
function frequencySort($s) {

}
}
```

Dart:

```
class Solution {
String frequencySort(String s) {

}
}
```

Scala:

```
object Solution {
def frequencySort(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec frequency_sort(s :: String.t) :: String.t
def frequency_sort(s) do

end
end
```

Erlang:

```
-spec frequency_sort(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
frequency_sort(S) ->
.
```

Racket:

```
(define/contract (frequency-sort s)
(-> string? string?)
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Characters By Frequency
* Difficulty: Medium
* Tags: string, tree, hash, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
string frequencySort(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Characters By Frequency
* Difficulty: Medium
* Tags: string, tree, hash, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String frequencySort(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Characters By Frequency
Difficulty: Medium
Tags: string, tree, hash, sort, queue, heap

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def frequencySort(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def frequencySort(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Sort Characters By Frequency
* Difficulty: Medium
* Tags: string, tree, hash, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} s
* @return {string}
*/
```

```
var frequencySort = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Characters By Frequency
* Difficulty: Medium
* Tags: string, tree, hash, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function frequencySort(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Sort Characters By Frequency
* Difficulty: Medium
* Tags: string, tree, hash, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public string FrequencySort(string s) {

}
}
```

C Solution:

```
/*
* Problem: Sort Characters By Frequency
* Difficulty: Medium
* Tags: string, tree, hash, sort, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* frequencySort(char* s) {

}
```

Go Solution:

```
// Problem: Sort Characters By Frequency
// Difficulty: Medium
// Tags: string, tree, hash, sort, queue, heap
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func frequencySort(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun frequencySort(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func frequencySort(_ s: String) -> String {

}
```

```
}
```

Rust Solution:

```
// Problem: Sort Characters By Frequency
// Difficulty: Medium
// Tags: string, tree, hash, sort, queue, heap
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn frequency_sort(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def frequency_sort(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function frequencySort($s) {

}
}
```

Dart Solution:

```
class Solution {
String frequencySort(String s) {

}
}
```

Scala Solution:

```
object Solution {
def frequencySort(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec frequency_sort(s :: String.t) :: String.t
def frequency_sort(s) do

end
end
```

Erlang Solution:

```
-spec frequency_sort(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
frequency_sort(S) ->
.
```

Racket Solution:

```
(define/contract (frequency-sort s)
(-> string? string?)
)
```