Problem 1370: Increasing Decreasing String

Problem Information

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

Problem Description

You are given a string

s

. Reorder the string using the following algorithm:

Remove the

smallest

character from

s

and

append

it to the result.

Remove the

smallest

character from

s

that is greater than the last appended character, and

append

it to the result.

Repeat step 2 until no more characters can be removed.

Remove the

largest

character from

s

and

append

it to the result.

Remove the

largest

character from

s

that is smaller than the last appended character, and

append

it to the result.

Repeat step 5 until no more characters can be removed.

Repeat steps 1 through 6 until all characters from

s

have been removed.

If the smallest or largest character appears more than once, you may choose any occurrence to append to the result.

Return the resulting string after reordering

s

using this algorithm.

Example 1:

Input:

s = "aaaabbbbcccc"

Output:

"abccbaabccba"

Explanation:

After steps 1, 2 and 3 of the first iteration, result = "abc" After steps 4, 5 and 6 of the first iteration, result = "abccba" First iteration is done. Now s = "aabbcc" and we go back to step 1 After steps 1, 2 and 3 of the second iteration, result = "abccbaabc" After steps 4, 5 and 6 of the second iteration, result = "abccbaabccba"

Example 2:

Input:

s = "rat"

Output:

"art"

Explanation:

The word "rat" becomes "art" after re-ordering it with the mentioned algorithm.

Constraints:

1 <= s.length <= 500

s

consists of only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* sortString(char* s) {

}
```

Go:

```
func sortString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}

}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Increasing Decreasing String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Increasing Decreasing String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String sortString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Increasing Decreasing String
Difficulty: Easy
Tags: string, hash
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Increasing Decreasing String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Increasing Decreasing String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function sortString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Increasing Decreasing String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

C Solution:

```
/*
* Problem: Increasing Decreasing String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

char* sortString(char* s) {

}
```

Go Solution:

```
// Problem: Increasing Decreasing String
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func sortString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Increasing Decreasing String
// Difficulty: Easy
// Tags: string, hash
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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