Problem 3216: Lexicographically Smallest String After a Swap

Problem Information

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

Problem Description

Given a string

s

containing only digits, return the

lexicographically smallest string

that can be obtained after swapping

adjacent

digits in

s

with the same

parity

at most

once

.

Digits have the same parity if both are odd or both are even. For example, 5 and 9, as well as 2 and 4, have the same parity, while 6 and 9 do not.

Example 1:

Input:

s = "45320"

Output:

"43520"

Explanation:

s[1] == '5'

and

s[2] == '3'

both have the same parity, and swapping them results in the lexicographically smallest string.

Example 2:

Input:

s = "001"

Output:

"001"

Explanation:

There is no need to perform a swap because

s

is already the lexicographically smallest.

Constraints:

2 <= s.length <= 100

s

consists only of digits.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: str
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* getSmallestString(char* s) {

}
```

Go:

```
func getSmallestString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (get-smallest-string s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest String After a Swap
* Difficulty: Easy
* Tags: string, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest String After a Swap
* Difficulty: Easy
* Tags: string, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String getSmallestString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest String After a Swap
Difficulty: Easy
Tags: string, graph, greedy
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest String After a Swap
* Difficulty: Easy
* Tags: string, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest String After a Swap
* Difficulty: Easy
* Tags: string, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function getSmallestString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest String After a Swap
* Difficulty: Easy
* Tags: string, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest String After a Swap
* Difficulty: Easy
* Tags: string, graph, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

char* getSmallestString(char* s) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest String After a Swap
// Difficulty: Easy
// Tags: string, graph, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func getSmallestString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest String After a Swap
// Difficulty: Easy
// Tags: string, graph, greedy
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-smallest-string s)
(-> string? string?)
)
```