Problem 1842: Next Palindrome Using Same Digits

Problem Information

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

Problem Description

You are given a numeric string

num

, representing a very large

palindrome

.

Return

the

smallest palindrome larger than

num

that can be created by rearranging its digits. If no such palindrome exists, return an empty string

""

.

A

palindrome

is a number that reads the same backward as forward.

Example 1:

Input:

num = "1221"

Output:

"2112"

Explanation:

The next palindrome larger than "1221" is "2112".

Example 2:

Input:

num = "32123"

Output:

""

Explanation:

No palindromes larger than "32123" can be made by rearranging the digits.

Example 3:

Input:

num = "45544554"

Output:

"54455445"

Explanation:

The next palindrome larger than "45544554" is "54455445".

Constraints:

1 <= num.length <= 10

5

num

is a

palindrome

.

Code Snippets

C++:

```
class Solution {
public:
string nextPalindrome(string num) {

}
};
```

Java:

```
class Solution {
public String nextPalindrome(String num) {

}
```

```
}
```

Python3:

```
class Solution:
def nextPalindrome(self, num: str) -> str:
```

Python:

```
class Solution(object):
def nextPalindrome(self, num):
"""
:type num: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} num
* @return {string}
*/
var nextPalindrome = function(num) {

};
```

TypeScript:

```
function nextPalindrome(num: string): string {

};
```

C#:

```
public class Solution {
public string NextPalindrome(string num) {

}
}
```

C:

```
char* nextPalindrome(char* num) {

}
```

Go:

```
func nextPalindrome(num string) string {

}
```

Kotlin:

```
class Solution {
fun nextPalindrome(num: String): String {

}
}
```

Swift:

```
class Solution {
func nextPalindrome(_ num: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn next_palindrome(num: String) -> String {

}
}
```

Ruby:

```
# @param {String} num
# @return {String}
def next_palindrome(num)

end
```

PHP:

```
class Solution {

/**
* @param String $num
* @return String
*/
function nextPalindrome($num) {

}
}
```

Dart:

```
class Solution {
String nextPalindrome(String num) {

}
}
```

Scala:

```
object Solution {
def nextPalindrome(num: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec next_palindrome(num :: String.t) :: String.t
def next_palindrome(num) do

end
end
```

Erlang:

```
-spec next_palindrome(Num :: unicode:unicode_binary()) ->
unicode:unicode_binary().
next_palindrome(Num) ->
.
```

Racket:

```
(define/contract (next-palindrome num)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Next Palindrome Using Same Digits
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string nextPalindrome(string num) {

}
};
```

Java Solution:

```
/**
* Problem: Next Palindrome Using Same Digits
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String nextPalindrome(String num) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Next Palindrome Using Same Digits
Difficulty: Hard
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

```
class Solution(object):
def nextPalindrome(self, num):
"""
:type num: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Next Palindrome Using Same Digits
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {string} num
* @return {string}
*/
var nextPalindrome = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Next Palindrome Using Same Digits
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function nextPalindrome(num: string): string {

};
```

C# Solution:

```
/*
* Problem: Next Palindrome Using Same Digits
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string NextPalindrome(string num) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Next Palindrome Using Same Digits
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* nextPalindrome(char* num) {

}
```

Go Solution:

```
// Problem: Next Palindrome Using Same Digits
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func nextPalindrome(num string) string {

}
```

Kotlin Solution:

```
class Solution {
fun nextPalindrome(num: String): String {

}
}
```

Swift Solution:

```
class Solution {
func nextPalindrome(_ num: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Next Palindrome Using Same Digits
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn next_palindrome(num: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} num
# @return {String}
def next_palindrome(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num
* @return String
*/
function nextPalindrome($num) {

}
}
```

Dart Solution:

```
class Solution {
String nextPalindrome(String num) {

}
}
```

Scala Solution:

```
object Solution {
def nextPalindrome(num: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec next_palindrome(num :: String.t) :: String.t
def next_palindrome(num) do

end
end
```

Erlang Solution:

```
-spec next_palindrome(Num :: unicode:unicode_binary()) ->
unicode:unicode_binary().
next_palindrome(Num) ->
.
```

Racket Solution:

```
(define/contract (next-palindrome num)
(-> string? string?)
)
```