Problem 557: Reverse Words in a String III

Problem Information

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

Problem Description

Given a string

s

, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Example 1:

Input:

s = "Let's take LeetCode contest"

Output:

"s'teL ekat edoCteeL tsetnoc"

Example 2:

Input:

s = "Mr Ding"

Output:

"rM gniD"

Constraints:

1 <= s.length <= 5 * 10

4

s

contains printable

ASCII

characters.

s

does not contain any leading or trailing spaces.

There is

at least one

word in

s

.

All the words in

s

are separated by a single space.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

```
char* reverseWords(char* s) {

}
```

Go:

```
func reverseWords(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reverse-words s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Words in a String III
* Difficulty: Easy
* 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 reverseWords(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Words in a String III
* Difficulty: Easy
* 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 reverseWords(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Words in a String III
Difficulty: Easy
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 reverseWords(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reverse Words in a String III
* Difficulty: Easy
* 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} s
* @return {string}
*/
var reverseWords = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Words in a String III
* Difficulty: Easy
* 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 reverseWords(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Reverse Words in a String III
* Difficulty: Easy
* 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 ReverseWords(string s) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Words in a String III
* Difficulty: Easy
* 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* reverseWords(char* s) {

}
```

Go Solution:

```
// Problem: Reverse Words in a String III
// Difficulty: Easy
// 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 reverseWords(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reverse Words in a String III
// Difficulty: Easy
// 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 reverse_words(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reverse-words s)
(-> string? string?)
)
```