Problem 1451: Rearrange Words in a Sentence

Problem Information

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

Problem Description

Given a sentence

text

(A

sentence

is a string of space-separated words) in the following format:

First letter is in upper case.

Each word in

text

are separated by a single space.

Your task is to rearrange the words in text such that all words are rearranged in an increasing order of their lengths. If two words have the same length, arrange them in their original order.

Return the new text following the format shown above.

Example 1:

Input:

text = "Leetcode is cool"

Output:

"Is cool leetcode"

Explanation:

There are 3 words, "Leetcode" of length 8, "is" of length 2 and "cool" of length 4. Output is ordered by length and the new first word starts with capital letter.

Example 2:

Input:

text = "Keep calm and code on"

Output:

"On and keep calm code"

Explanation:

Output is ordered as follows: "On" 2 letters. "and" 3 letters. "keep" 4 letters in case of tie order by position in original text. "calm" 4 letters. "code" 4 letters.

Example 3:

Input:

text = "To be or not to be"

Output:

"To be or to be not"

Constraints:

text

begins with a capital letter and then contains lowercase letters and single space between words.

1 <= text.length <= 10^5

Code Snippets

C++:

```
class Solution {
public:
string arrangeWords(string text) {

}
};
```

Java:

```
class Solution {
public String arrangeWords(String text) {

}
}
```

Python3:

```
class Solution:
def arrangeWords(self, text: str) -> str:
```

Python:

```
class Solution(object):
def arrangeWords(self, text):
"""
:type text: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} text
* @return {string}
*/
var arrangeWords = function(text) {

};
```

TypeScript:

```
function arrangeWords(text: string): string {

};
```

C#:

```
public class Solution {
public string ArrangeWords(string text) {

}
}
```

C:

```
char* arrangeWords(char* text) {

}
```

Go:

```
func arrangeWords(text string) string {

}
```

Kotlin:

```
class Solution {
fun arrangeWords(text: String): String {

}
}
```

Swift:

```
class Solution {
func arrangeWords(_ text: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn arrange_words(text: String) -> String {

}
}
```

Ruby:

```
# @param {String} text
# @return {String}
def arrange_words(text)

end
```

PHP:

```
class Solution {

/**
* @param String $text
* @return String
*/
function arrangeWords($text) {

}
}
```

Dart:

```
class Solution {
String arrangeWords(String text) {

}
}
```

Scala:

```
object Solution {
def arrangeWords(text: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec arrange_words(text :: String.t) :: String.t
def arrange_words(text) do

end
end
```

Erlang:

```
-spec arrange_words(Text :: unicode:unicode_binary()) ->
unicode:unicode_binary().
arrange_words(Text) ->
.
```

Racket:

```
(define/contract (arrange-words text)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rearrange Words in a Sentence
* Difficulty: Medium
* Tags: string, sort
*
* 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 arrangeWords(string text) {

}
};
```

Java Solution:

```
/**
* Problem: Rearrange Words in a Sentence
* Difficulty: Medium
* Tags: string, sort
*
* 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 arrangeWords(String text) {

}
}
```

Python3 Solution:

```
"""
Problem: Rearrange Words in a Sentence
Difficulty: Medium
Tags: string, sort

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 arrangeWords(self, text: str) -> str:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def arrangeWords(self, text):
"""
:type text: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Rearrange Words in a Sentence
* Difficulty: Medium
* Tags: string, sort
*
* 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} text
* @return {string}
*/
var arrangeWords = function(text) {

};
```

TypeScript Solution:

```
/**
* Problem: Rearrange Words in a Sentence
* Difficulty: Medium
* Tags: string, sort
*
* 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 arrangeWords(text: string): string {

};
```

C# Solution:

```
/*
* Problem: Rearrange Words in a Sentence
* Difficulty: Medium
* Tags: string, sort
*
* 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 ArrangeWords(string text) {

}
}
```

C Solution:

```
/*
* Problem: Rearrange Words in a Sentence
* Difficulty: Medium
* Tags: string, sort
*
* 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* arrangeWords(char* text) {

}
```

Go Solution:

```
// Problem: Rearrange Words in a Sentence
// Difficulty: Medium
// Tags: string, sort
//
// 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 arrangeWords(text string) string {

}
```

Kotlin Solution:

```
class Solution {
fun arrangeWords(text: String): String {

}
}
```

Swift Solution:

```
class Solution {
func arrangeWords(_ text: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Rearrange Words in a Sentence
// Difficulty: Medium
// Tags: string, sort
//
// 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 arrange_words(text: String) -> String {
}
}
```

Ruby Solution:

```
# @param {String} text
# @return {String}
def arrange_words(text)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $text
* @return String
*/
function arrangeWords($text) {

}
}
```

Dart Solution:

```
class Solution {
String arrangeWords(String text) {

}
}
```

Scala Solution:

```
object Solution {
def arrangeWords(text: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec arrange_words(text :: String.t) :: String.t
def arrange_words(text) do

end
end
```

Erlang Solution:

```
-spec arrange_words(Text :: unicode:unicode_binary()) ->
unicode:unicode_binary().
arrange_words(Text) ->
.
```

Racket Solution:

```
(define/contract (arrange-words text)
(-> string? string?)
)
```