Problem 1859: Sorting the Sentence

Problem Information

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

Problem Description

A

sentence

is a list of words that are separated by a single space with no leading or trailing spaces. Each word consists of lowercase and uppercase English letters.

A sentence can be

shuffled

by appending the

1-indexed word position

to each word then rearranging the words in the sentence.

For example, the sentence

"This is a sentence"

can be shuffled as

"sentence4 a3 is2 This1"

or

"is2 sentence4 This1 a3"

.

Given a

shuffled sentence

s

containing no more than

9

words, reconstruct and return

the original sentence

.

Example 1:

Input:

s = "is2 sentence4 This1 a3"

Output:

"This is a sentence"

Explanation:

Sort the words in s to their original positions "This1 is2 a3 sentence4", then remove the numbers.

Example 2:

Input:

s = "Myself2 Me1 I4 and3"

Output:

"Me Myself and I"

Explanation:

Sort the words in s to their original positions "Me1 Myself2 and3 I4", then remove the numbers.

Constraints:

2 <= s.length <= 200

s

consists of lowercase and uppercase English letters, spaces, and digits from

1

to

9

.

The number of words in

s

is between

1

and

9

.

The words in

s

are separated by a single space.

s

contains no leading or trailing spaces.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def sortSentence(self, s):
```

```
"""
:type s: str
:rtype: str
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char * sortSentence(char * s){

}
```

Go:

```
func sortSentence(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Racket:

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

)
```

Solutions

C++ Solution:

```
/*
* Problem: Sorting the Sentence
* Difficulty: Easy
* 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 sortSentence(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Sorting the Sentence
* Difficulty: Easy
* 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 sortSentence(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Sorting the Sentence
Difficulty: Easy
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 sortSentence(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sorting the Sentence
* Difficulty: Easy
* 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} s
* @return {string}
*/
var sortSentence = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Sorting the Sentence
* Difficulty: Easy
* 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 sortSentence(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Sorting the Sentence
* Difficulty: Easy
* 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 SortSentence(string s) {

}
}
```

C Solution:

```
/*
* Problem: Sorting the Sentence
* Difficulty: Easy
* 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 * sortSentence(char * s){

}
```

Go Solution:

```
// Problem: Sorting the Sentence
// Difficulty: Easy
// 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 sortSentence(s string) string {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sorting the Sentence
// Difficulty: Easy
// 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 sort_sentence(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```

Racket Solution:

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

)
```