Problem 165: Compare Version Numbers

Problem Information

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

Problem Description

Given two

version strings

,

version1

and

version2

, compare them. A version string consists of

revisions

separated by dots

'.'

. The

value of the revision

is its

integer conversion

ignoring leading zeros.

To compare version strings, compare their revision values in

left-to-right order

. If one of the version strings has fewer revisions, treat the missing revision values as

0

.

Return the following:

If

version1 < version2

, return -1.

If

version1 > version2

, return 1.

Otherwise, return 0.

Example 1:

Input:

version1 = "1.2", version2 = "1.10"

Output:

-1

Explanation:

version1's second revision is "2" and version2's second revision is "10": 2 < 10, so version1 < version2.

Example 2:

Input:

version1 = "1.01", version2 = "1.001"

Output:

0

Explanation:

Ignoring leading zeroes, both "01" and "001" represent the same integer "1".

Example 3:

Input:

version1 = "1.0", version2 = "1.0.0.0"

Output:

0

Explanation:

version1 has less revisions, which means every missing revision are treated as "0".

Constraints:

1 <= version1.length, version2.length <= 500

version1

and

version2

only contain digits and

'.'

.

version1

and

version2

are valid version numbers

.

All the given revisions in

version1

and

version2

can be stored in a

32-bit integer

.

Code Snippets

C++:

```
class Solution {
public:
int compareVersion(string version1, string version2) {

}
};
```

Java:

```
class Solution {
public int compareVersion(String version1, String version2) {

}
}
```

Python3:

```
class Solution:
def compareVersion(self, version1: str, version2: str) -> int:
```

Python:

```
class Solution(object):
def compareVersion(self, version1, version2):
"""
:type version1: str
:type version2: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} version1
* @param {string} version2
* @return {number}
*/
var compareVersion = function(version1, version2) {

};
```

TypeScript:

```
function compareVersion(version1: string, version2: string): number {

};
```

C#:

```
public class Solution {
public int CompareVersion(string version1, string version2) {

}
}
```

C:

```
int compareVersion(char* version1, char* version2) {

}
```

Go:

```
func compareVersion(version1 string, version2 string) int {

}
```

Kotlin:

```
class Solution {
fun compareVersion(version1: String, version2: String): Int {

}
}
```

Swift:

```
class Solution {
func compareVersion(_ version1: String, _ version2: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn compare_version(version1: String, version2: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} version1
# @param {String} version2
# @return {Integer}
def compare_version(version1, version2)

end
```

PHP:

```
class Solution {

/**
* @param String $version1
* @param String $version2
* @return Integer
*/
function compareVersion($version1, $version2) {

}
}
```

Dart:

```
class Solution {
int compareVersion(String version1, String version2) {

}
}
```

Scala:

```
object Solution {
def compareVersion(version1: String, version2: String): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec compare_version(version1 :: String.t, version2 :: String.t) :: integer
def compare_version(version1, version2) do

end
end
```

Erlang:

```
-spec compare_version(Version1 :: unicode:unicode_binary(), Version2 ::
unicode:unicode_binary()) -> integer().
compare_version(Version1, Version2) ->
.
```

Racket:

```
(define/contract (compare-version version1 version2)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Compare Version Numbers
* Difficulty: Medium
* 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:
```

```
int compareVersion(string version1, string version2) {

}
};
```

Java Solution:

```
/**
* Problem: Compare Version Numbers
* Difficulty: Medium
* 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 int compareVersion(String version1, String version2) {

}
}
```

Python3 Solution:

```
"""
Problem: Compare Version Numbers
Difficulty: Medium
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 compareVersion(self, version1: str, version2: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def compareVersion(self, version1, version2):
"""
:type version1: str
:type version2: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Compare Version Numbers
* Difficulty: Medium
* 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} version1
* @param {string} version2
* @return {number}
*/
var compareVersion = function(version1, version2) {

};
```

TypeScript Solution:

```
/**
* Problem: Compare Version Numbers
* Difficulty: Medium
* 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 compareVersion(version1: string, version2: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Compare Version Numbers
* Difficulty: Medium
* 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 int CompareVersion(string version1, string version2) {

}
}
```

C Solution:

```
/*
* Problem: Compare Version Numbers
* Difficulty: Medium
* 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
*/

int compareVersion(char* version1, char* version2) {

}
```

Go Solution:

```
// Problem: Compare Version Numbers
// Difficulty: Medium
```

```
// 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 compareVersion(version1 string, version2 string) int {

}
```

Kotlin Solution:

```
class Solution {
fun compareVersion(version1: String, version2: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func compareVersion(_ version1: String, _ version2: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Compare Version Numbers
// Difficulty: Medium
// 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 compare_version(version1: String, version2: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} version1
# @param {String} version2
# @return {Integer}
def compare_version(version1, version2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $version1
* @param String $version2
* @return Integer
*/
function compareVersion($version1, $version2) {

}
}
```

Dart Solution:

```
class Solution {
int compareVersion(String version1, String version2) {

}
}
```

Scala Solution:

```
object Solution {
def compareVersion(version1: String, version2: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec compare_version(version1 :: String.t, version2 :: String.t) :: integer
def compare_version(version1, version2) do

end
end
```

Erlang Solution:

```
-spec compare_version(Version1 :: unicode:unicode_binary(), Version2 ::
unicode:unicode_binary()) -> integer().
compare_version(Version1, Version2) ->
.
```

Racket Solution:

```
(define/contract (compare-version version1 version2)
(-> string? string? exact-integer?)
)
```