# Problem 3297: Count Substrings That Can Be Rearranged to Contain a String I

## Problem Information

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

## Problem Description

You are given two strings

word1

and

word2

.

A string

x

is called

valid

if

x

can be rearranged to have

word2

as a

prefix

.

Return the total number of

valid

substrings

of

word1

.

Example 1:

Input:

word1 = "bcca", word2 = "abc"

Output:

1

Explanation:

The only valid substring is

"bcca"

which can be rearranged to

"abcc"

having

"abc"

as a prefix.

Example 2:

Input:

word1 = "abcabc", word2 = "abc"

Output:

10

Explanation:

All the substrings except substrings of size 1 and size 2 are valid.

Example 3:

Input:

word1 = "abcabc", word2 = "aaabc"

Output:

0

Constraints:

1 <= word1.length <= 10

5

1 <= word2.length <= 10

4

word1

and

word2

consist only of lowercase English letters.

## Code Snippets

**C++:**

```
class Solution {
public:
long long validSubstringCount(string word1, string word2) {

}
};
```

**Java:**

```
class Solution {
public long validSubstringCount(String word1, String word2) {

}
}
```

**Python3:**

```
class Solution:
def validSubstringCount(self, word1: str, word2: str) -> int:
```

**Python:**

```
class Solution(object):
def validSubstringCount(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
```

```
"""
```

**JavaScript:**

```
/**
* @param {string} word1
* @param {string} word2
* @return {number}
*/
var validSubstringCount = function(word1, word2) {

};
```

**TypeScript:**

```
function validSubstringCount(word1: string, word2: string): number {

};
```

**C#:**

```
public class Solution {
public long ValidSubstringCount(string word1, string word2) {

}
}
```

**C:**

```
long long validSubstringCount(char* word1, char* word2) {

}
```

**Go:**

```
func validSubstringCount(word1 string, word2 string) int64 {

}
```

**Kotlin:**

```
class Solution {
fun validSubstringCount(word1: String, word2: String): Long {

}
}
```

**Swift:**

```
class Solution {
func validSubstringCount(_ word1: String, _ word2: String) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn valid_substring_count(word1: String, word2: String) -> i64 {

}
}
```

**Ruby:**

```
# @param {String} word1
# @param {String} word2
# @return {Integer}
def valid_substring_count(word1, word2)

end
```

**PHP:**

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function validSubstringCount($word1, $word2) {

}
}
```

**Dart:**

```
class Solution {
int validSubstringCount(String word1, String word2) {

}
}
```

**Scala:**

```
object Solution {
def validSubstringCount(word1: String, word2: String): Long = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec valid_substring_count(word1 :: String.t, word2 :: String.t) :: integer
def valid_substring_count(word1, word2) do

end
end
```

**Erlang:**

```
-spec valid_substring_count(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
valid_substring_count(Word1, Word2) ->
.
```

**Racket:**

```
(define/contract (valid-substring-count word1 word2)
(-> string? string? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Count Substrings That Can Be Rearranged to Contain a String I
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long validSubstringCount(string word1, string word2) {

}
};
```

### Java Solution:

```
/**
* Problem: Count Substrings That Can Be Rearranged to Contain a String I
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long validSubstringCount(String word1, String word2) {

}
}
```

### Python3 Solution:

```
"""
Problem: Count Substrings That Can Be Rearranged to Contain a String I
```

```
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def validSubstringCount(self, word1: str, word2: str) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def validSubstringCount(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
"""
```

**JavaScript Solution:**

```
/**
* Problem: Count Substrings That Can Be Rearranged to Contain a String I
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} word1
* @param {string} word2
* @return {number}
*/
var validSubstringCount = function(word1, word2) {
};
```

**TypeScript Solution:**

```
/**
* Problem: Count Substrings That Can Be Rearranged to Contain a String I
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function validSubstringCount(word1: string, word2: string): number {

};
```

**C# Solution:**

```
/*
* Problem: Count Substrings That Can Be Rearranged to Contain a String I
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long ValidSubstringCount(string word1, string word2) {

}
}
```

**C Solution:**

```
/*
* Problem: Count Substrings That Can Be Rearranged to Contain a String I
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long validSubstringCount(char* word1, char* word2) {

}
```

**Go Solution:**

```
// Problem: Count Substrings That Can Be Rearranged to Contain a String I
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func validSubstringCount(word1 string, word2 string) int64 {

}
```

**Kotlin Solution:**

```
class Solution {
fun validSubstringCount(word1: String, word2: String): Long {

}
}
```

**Swift Solution:**

```
class Solution {
func validSubstringCount(_ word1: String, _ word2: String) -> Int {

}
}
```

## Rust Solution:

```
// Problem: Count Substrings That Can Be Rearranged to Contain a String I
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn valid_substring_count(word1: String, word2: String) -> i64 {


}
}
```

## Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {Integer}
def valid_substring_count(word1, word2)

end
```

## PHP Solution:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function validSubstringCount($word1, $word2) {

}
}
```

## Dart Solution:

```
class Solution {
int validSubstringCount(String word1, String word2) {

}
}
```

### Scala Solution:

```
object Solution {
def validSubstringCount(word1: String, word2: String): Long = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec valid_substring_count(word1 :: String.t, word2 :: String.t) :: integer
def valid_substring_count(word1, word2) do

end
end
```

### Erlang Solution:

```
-spec valid_substring_count(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
valid_substring_count(Word1, Word2) ->
.
```

### Racket Solution:

```
(define/contract (valid-substring-count word1 word2)
(-> string? string? exact-integer?)
)
```