Problem 1768: Merge Strings Alternately

Problem Information

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

Problem Description

You are given two strings

word1

and

word2

. Merge the strings by adding letters in alternating order, starting with

word1

. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return

the merged string.

Example 1:

Input:

word1 = "abc", word2 = "pqr"

Output:

"apbqcr"

Explanation:

The merged string will be merged as so: word1: a b c word2: p q r merged: a p b q c r

Example 2:

Input:

word1 = "ab", word2 = "pqrs"

Output:

"apbqrs"

Explanation:

Notice that as word2 is longer, "rs" is appended to the end. word1: a b word2: p q r s merged: a p b q r s

Example 3:

Input:

word1 = "abcd", word2 = "pq"

Output:

"apbqcd"

Explanation:

Notice that as word1 is longer, "cd" is appended to the end. word1: a b c d word2: p q merged: a p b q c d

Constraints:

1 <= word1.length, word2.length <= 100

word1

and

word2

consist of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public String mergeAlternately(String word1, String word2) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

```
function mergeAlternately(word1: string, word2: string): string {

};
```

C#:

```
public class Solution {
public string MergeAlternately(string word1, string word2) {

}
}
```

C:

```


char * mergeAlternately(char * word1, char * word2){

}
```

Go:

```
func mergeAlternately(word1 string, word2 string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}

}
```

Scala:

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

}
}
```

Racket:

```
(define/contract (merge-alternately word1 word2)
(-> string? string? string?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Strings Alternately
* 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 mergeAlternately(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Strings Alternately
* 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 mergeAlternately(String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge Strings Alternately
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 mergeAlternately(self, word1: str, word2: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Merge Strings Alternately
* 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} word1
* @param {string} word2
* @return {string}
*/
var mergeAlternately = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Strings Alternately
* 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 mergeAlternately(word1: string, word2: string): string {

};
```

C# Solution:

```
/*
* Problem: Merge Strings Alternately
* 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 MergeAlternately(string word1, string word2) {

}
}
```

C Solution:

```
/*
* Problem: Merge Strings Alternately
* 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 * mergeAlternately(char * word1, char * word2){

}
```

Go Solution:

```
// Problem: Merge Strings Alternately
// 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 mergeAlternately(word1 string, word2 string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Merge Strings Alternately
// 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 merge_alternately(word1: String, word2: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {String}
```

```
def merge_alternately(word1, word2)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```

Racket Solution:

```
(define/contract (merge-alternately word1 word2)
(-> string? string? string?)

)
```