Problem 67: Add Binary

Problem Information

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

Problem Description

Given two binary strings

a

and

b

, return

their sum as a binary string

.

Example 1:

Input:

a = "11", b = "1"

Output:

"100"

Example 2:

Input:

a = "1010", b = "1011"

Output:

"10101"

Constraints:

1 <= a.length, b.length <= 10

4

a

and

b

consist only of

'0'

or

'1'

characters.

Each string does not contain leading zeros except for the zero itself.

Code Snippets

C++:

```
class Solution {
public:
```

```
string addBinary(string a, string b) {

}
};
```

Java:

```
class Solution {
public String addBinary(String a, String b) {

}
}
```

Python3:

```
class Solution:
def addBinary(self, a: str, b: str) -> str:
```

Python:

```
class Solution(object):
def addBinary(self, a, b):
"""
:type a: str
:type b: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} a
* @param {string} b
* @return {string}
*/
var addBinary = function(a, b) {

};
```

TypeScript:

```
function addBinary(a: string, b: string): string {

};
```

C#:

```
public class Solution {
public string AddBinary(string a, string b) {

}
}
```

C:

```
char* addBinary(char* a, char* b) {

}
```

Go:

```
func addBinary(a string, b string) string {

}
```

Kotlin:

```
class Solution {
fun addBinary(a: String, b: String): String {

}
}
```

Swift:

```
class Solution {
func addBinary(_ a: String, _ b: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn add_binary(a: String, b: String) -> String {

}
}
```

Ruby:

```
# @param {String} a
# @param {String} b
# @return {String}
def add_binary(a, b)

end
```

PHP:

```
class Solution {

/**
* @param String $a
* @param String $b
* @return String
*/
function addBinary($a, $b) {

}
}
```

Dart:

```
class Solution {
String addBinary(String a, String b) {

}
}
```

Scala:

```
object Solution {
def addBinary(a: String, b: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec add_binary(a :: String.t, b :: String.t) :: String.t
def add_binary(a, b) do

end
end
```

Erlang:

```
-spec add_binary(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary()) -> unicode:unicode_binary().
add_binary(A, B) ->
.
```

Racket:

```
(define/contract (add-binary a b)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Add Binary
* Difficulty: Easy
* Tags: string, math
*
* 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 addBinary(string a, string b) {

}
};
```

Java Solution:

```
/**
* Problem: Add Binary
* Difficulty: Easy
* Tags: string, math
*
* 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 addBinary(String a, String b) {

}
}
```

Python3 Solution:

```
"""
Problem: Add Binary
Difficulty: Easy
Tags: string, math

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 addBinary(self, a: str, b: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def addBinary(self, a, b):
"""
:type a: str
:type b: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Add Binary
* Difficulty: Easy
* Tags: string, math
*
* 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} a
* @param {string} b
* @return {string}
*/
var addBinary = function(a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Add Binary
* Difficulty: Easy
* Tags: string, math
*
* 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 addBinary(a: string, b: string): string {
};
```

C# Solution:

```
/*
* Problem: Add Binary
* Difficulty: Easy
* Tags: string, math
*
* 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 AddBinary(string a, string b) {

}
}
```

C Solution:

```
/*
* Problem: Add Binary
* Difficulty: Easy
* Tags: string, math
*
* 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* addBinary(char* a, char* b) {

}
```

Go Solution:

```
// Problem: Add Binary
// Difficulty: Easy
// Tags: string, math
//
// 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 addBinary(a string, b string) string {

}
```

Kotlin Solution:

```
class Solution {
fun addBinary(a: String, b: String): String {

}
}
```

Swift Solution:

```
class Solution {
func addBinary(_ a: String, _ b: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Add Binary
// Difficulty: Easy
// Tags: string, math
//
// 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 add_binary(a: String, b: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} a
# @param {String} b
# @return {String}
def add_binary(a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $a
* @param String $b
* @return String
*/
function addBinary($a, $b) {

}
}
```

Dart Solution:

```
class Solution {
String addBinary(String a, String b) {

}
}
```

Scala Solution:

```
object Solution {
def addBinary(a: String, b: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec add_binary(a :: String.t, b :: String.t) :: String.t
def add_binary(a, b) do

end
end
```

Erlang Solution:

```
-spec add_binary(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary()) -> unicode:unicode_binary().
add_binary(A, B) ->
.
```

Racket Solution:

```
(define/contract (add-binary a b)
(-> string? string? string?)
)
```