Problem 415: Add Strings

Problem Information

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

Problem Description

Given two non-negative integers,

num1

and

num2

represented as string, return

the sum of

num1

and

num2

as a string

.

You must solve the problem without using any built-in library for handling large integers (such as

BigInteger

). You must also not convert the inputs to integers directly.

Example 1:

Input:

num1 = "11", num2 = "123"

Output:

"134"

Example 2:

Input:

num1 = "456", num2 = "77"

Output:

"533"

Example 3:

Input:

num1 = "0", num2 = "0"

Output:

"0"

Constraints:

1 <= num1.length, num2.length <= 10

num1

and

num2

consist of only digits.

num1

and

num2

don't have any leading zeros except for the zero itself.

Code Snippets

C++:

```
class Solution {
public:
string addStrings(string num1, string num2) {

}
};
```

Java:

```
class Solution {
public String addStrings(String num1, String num2) {

}
}
```

Python3:

```
class Solution:
def addStrings(self, num1: str, num2: str) -> str:
```

Python:

```
class Solution(object):
def addStrings(self, num1, num2):
"""
:type num1: str
:type num2: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} num1
* @param {string} num2
* @return {string}
*/
var addStrings = function(num1, num2) {

};
```

TypeScript:

```
function addStrings(num1: string, num2: string): string {

};
```

C#:

```
public class Solution {
public string AddStrings(string num1, string num2) {

}
}
```

C:

```
char* addStrings(char* num1, char* num2) {

}
```

Go:

```
func addStrings(num1 string, num2 string) string {

}
```

Kotlin:

```
class Solution {
fun addStrings(num1: String, num2: String): String {

}
}
```

Swift:

```
class Solution {
func addStrings(_ num1: String, _ num2: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn add_strings(num1: String, num2: String) -> String {

}
}
```

Ruby:

```
# @param {String} num1
# @param {String} num2
# @return {String}
def add_strings(num1, num2)

end
```

PHP:

```
class Solution {

/**
* @param String $num1
* @param String $num2
* @return String
*/
function addStrings($num1, $num2) {

}
}
```

Dart:

```
class Solution {
String addStrings(String num1, String num2) {

}
}
```

Scala:

```
object Solution {
def addStrings(num1: String, num2: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec add_strings(num1 :: String.t, num2 :: String.t) :: String.t
def add_strings(num1, num2) do

end
end
```

Erlang:

```
-spec add_strings(Num1 :: unicode:unicode_binary(), Num2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
add_strings(Num1, Num2) ->
.
```

Racket:

```
(define/contract (add-strings num1 num2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Add Strings
* 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 addStrings(string num1, string num2) {

}
};
```

Java Solution:

```
/**
* Problem: Add Strings
* 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 addStrings(String num1, String num2) {

}
}
```

Python3 Solution:

```
"""
Problem: Add Strings
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 addStrings(self, num1: str, num2: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def addStrings(self, num1, num2):
"""
:type num1: str
:type num2: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Add Strings
* 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} num1
* @param {string} num2
* @return {string}
*/
var addStrings = function(num1, num2) {

};
```

TypeScript Solution:

```
/**
* Problem: Add Strings
* 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 addStrings(num1: string, num2: string): string {

};
```

C# Solution:

```
/*
* Problem: Add Strings
* 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 AddStrings(string num1, string num2) {

}
}
```

C Solution:

```
/*
* Problem: Add Strings
* 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* addStrings(char* num1, char* num2) {

}
```

Go Solution:

```
// Problem: Add Strings
// 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 addStrings(num1 string, num2 string) string {

}
```

Kotlin Solution:

```
class Solution {
fun addStrings(num1: String, num2: String): String {

}
}
```

Swift Solution:

```
class Solution {
func addStrings(_ num1: String, _ num2: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Add Strings
// 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_strings(num1: String, num2: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} num1
# @param {String} num2
# @return {String}
def add_strings(num1, num2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num1
* @param String $num2
* @return String
*/
function addStrings($num1, $num2) {


}
}
```

Dart Solution:

```
class Solution {
String addStrings(String num1, String num2) {

}
}
```

Scala Solution:

```
object Solution {
def addStrings(num1: String, num2: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec add_strings(num1 :: String.t, num2 :: String.t) :: String.t
def add_strings(num1, num2) do

end
end
```

Erlang Solution:

```
-spec add_strings(Num1 :: unicode:unicode_binary(), Num2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
add_strings(Num1, Num2) ->
.
```

Racket Solution:

```
(define/contract (add-strings num1 num2)
(-> string? string? string?)
)
```