# Problem 43: Multiply Strings

## Problem Information

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

## Problem Description

Given two non-negative integers

num1

and

num2

represented as strings, return the product of

num1

and

num2

, also represented as a string.

Note:

You must not use any built-in BigInteger library or convert the inputs to integer directly.

Example 1:

Input:

num1 = "2", num2 = "3"

Output:

"6"

Example 2:

Input:

num1 = "123", num2 = "456"

Output:

"56088"

Constraints:

1 <= num1.length, num2.length <= 200

num1

and

num2

consist of digits only.

Both

num1

and

num2

do not contain any leading zero, except the number

0

itself.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

### Scala:

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

}
}
```

### Elixir:

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

end
end
```

### Erlang:

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

### Racket:

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

## Solutions

### C++ Solution:

```
/*
* Problem: Multiply Strings
* Difficulty: Medium
* 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 multiply(string num1, string num2) {

}
};
```

### Java Solution:

```
/**
* Problem: Multiply Strings
* Difficulty: Medium
* 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 multiply(String num1, String num2) {

}
}
```

### Python3 Solution:

```
"""
Problem: Multiply Strings
Difficulty: Medium
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 multiply(self, num1: str, num2: str) -> str:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Multiply Strings
* Difficulty: Medium
* 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 multiply = function(num1, num2) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Multiply Strings
* Difficulty: Medium
* 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 multiply(num1: string, num2: string): string {

};
```

### C# Solution:

```
/*
* Problem: Multiply Strings
* Difficulty: Medium
* 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 Multiply(string num1, string num2) {

}
}
```

### C Solution:

```
/*
* Problem: Multiply Strings
* Difficulty: Medium
* 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* multiply(char* num1, char* num2) {

}
```

**Go Solution:**

```
// Problem: Multiply Strings
// Difficulty: Medium
// 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 multiply(num1 string, num2 string) string {

}
```

**Kotlin Solution:**

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

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Multiply Strings
// Difficulty: Medium
// 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 multiply(num1: String, num2: String) -> String {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

end
end
```

**Erlang Solution:**

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

**Racket Solution:**

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