Problem 2169: Count Operations to Obtain Zero

Problem Information

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

Problem Description

You are given two

non-negative

integers

num1

and

num2

.

In one

operation

, if

num1 >= num2

, you must subtract

num2

from

num1

, otherwise subtract

num1

from

num2

.

For example, if

num1 = 5

and

num2 = 4

, subtract

num2

from

num1

, thus obtaining

num1 = 1

and

num2 = 4

. However, if

num1 = 4

and

num2 = 5

, after one operation,

num1 = 4

and

num2 = 1

.

Return

the

number of operations

required to make either

num1 = 0

or

num2 = 0

.

Example 1:

Input:

num1 = 2, num2 = 3

Output:

3

Explanation:

- Operation 1: num1 = 2, num2 = 3. Since num1 < num2, we subtract num1 from num2 and get num1 = 2, num2 = 3 - 2 = 1. - Operation 2: num1 = 2, num2 = 1. Since num1 > num2, we subtract num2 from num1. - Operation 3: num1 = 1, num2 = 1. Since num1 == num2, we subtract num2 from num1. Now num1 = 0 and num2 = 1. Since num1 == 0, we do not need to perform any further operations. So the total number of operations required is 3.

Example 2:

Input:

num1 = 10, num2 = 10

Output:

1

Explanation:

- Operation 1: num1 = 10, num2 = 10. Since num1 == num2, we subtract num2 from num1 and get num1 = 10 - 10 = 0. Now num1 = 0 and num2 = 10. Since num1 == 0, we are done. So the total number of operations required is 1.

Constraints:

0 <= num1, num2 <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int countOperations(int num1, int num2) {

}
};
```

Java:

```
class Solution {
public int countOperations(int num1, int num2) {

}
}
```

Python3:

```
class Solution:
def countOperations(self, num1: int, num2: int) -> int:
```

Python:

```
class Solution(object):
def countOperations(self, num1, num2):
"""
:type num1: int
:type num2: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countOperations(num1: number, num2: number): number {

};
```

C#:

```
public class Solution {
public int CountOperations(int num1, int num2) {

}
}
```

C:

```
int countOperations(int num1, int num2) {

}
```

Go:

```
func countOperations(num1 int, num2 int) int {

}
```

Kotlin:

```
class Solution {
fun countOperations(num1: Int, num2: Int): Int {

}
}
```

Swift:

```
class Solution {
func countOperations(_ num1: Int, _ num2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_operations(num1: i32, num2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num1
# @param {Integer} num2
# @return {Integer}
def count_operations(num1, num2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num1
* @param Integer $num2
* @return Integer
*/
function countOperations($num1, $num2) {

}
}
```

Dart:

```
class Solution {
int countOperations(int num1, int num2) {

}
}
```

Scala:

```
object Solution {
def countOperations(num1: Int, num2: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec count_operations(num1 :: integer, num2 :: integer) :: integer
def count_operations(num1, num2) do

end
end
```

Erlang:

```
-spec count_operations(Num1 :: integer(), Num2 :: integer()) -> integer().
count_operations(Num1, Num2) ->
.
```

Racket:

```
(define/contract (count-operations num1 num2)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Operations to Obtain Zero
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int countOperations(int num1, int num2) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Count Operations to Obtain Zero
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countOperations(int num1, int num2) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Operations to Obtain Zero
Difficulty: Easy
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countOperations(self, num1: int, num2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countOperations(self, num1, num2):
"""
:type num1: int
:type num2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Operations to Obtain Zero
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} num1
* @param {number} num2
* @return {number}
*/
var countOperations = function(num1, num2) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Operations to Obtain Zero
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function countOperations(num1: number, num2: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Count Operations to Obtain Zero
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int CountOperations(int num1, int num2) {

}
}
```

C Solution:

```
/*
* Problem: Count Operations to Obtain Zero
* Difficulty: Easy
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int countOperations(int num1, int num2) {

}
```

Go Solution:

```
// Problem: Count Operations to Obtain Zero
// Difficulty: Easy
```

```
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func countOperations(num1 int, num2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countOperations(num1: Int, num2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countOperations(_ num1: Int, _ num2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Operations to Obtain Zero
// Difficulty: Easy
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_operations(num1: i32, num2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num1
# @param {Integer} num2
# @return {Integer}
def count_operations(num1, num2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num1
* @param Integer $num2
* @return Integer
*/
function countOperations($num1, $num2) {

}
}
```

Dart Solution:

```
class Solution {
int countOperations(int num1, int num2) {

}
}
```

Scala Solution:

```
object Solution {
def countOperations(num1: Int, num2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_operations(num1 :: integer, num2 :: integer) :: integer
def count_operations(num1, num2) do

end
end
```

Erlang Solution:

```
-spec count_operations(Num1 :: integer(), Num2 :: integer()) -> integer().
count_operations(Num1, Num2) ->
.
```

Racket Solution:

```
(define/contract (count-operations num1 num2)
(-> exact-integer? exact-integer? exact-integer?)
)
```