Problem 2749: Minimum Operations to Make the Integer Zero

Problem Information

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

Problem Description

You are given two integers

num1

and

num2

.

In one operation, you can choose integer

i

in the range

[0, 60]

and subtract

2

i

+ num2

from

num1

.

Return

the integer denoting the

minimum

number of operations needed to make

num1

equal to

0

.

If it is impossible to make

num1

equal to

0

, return

-1

.

Example 1:

Input:

num1 = 3, num2 = -2

Output:

3

Explanation:

We can make 3 equal to 0 with the following operations: - We choose i = 2 and subtract 2

2

+ (-2) from 3, 3 - (4 + (-2)) = 1. - We choose i = 2 and subtract 2

2

+ (-2) from 1, 1 - (4 + (-2)) = -1. - We choose i = 0 and subtract 2

0

+ (-2) from -1, (-1) - (1 + (-2)) = 0. It can be proven, that 3 is the minimum number of operations that we need to perform.

Example 2:

Input:

num1 = 5, num2 = 7

Output:

-1

Explanation:

It can be proven, that it is impossible to make 5 equal to 0 with the given operation.

Constraints:

1 <= num1 <= 10

9

-10

9

<= num2 <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer $num1
* @param Integer $num2
```

```
* @return Integer
*/
function makeTheIntegerZero($num1, $num2) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (make-the-integer-zero num1 num2)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make the Integer Zero
* Difficulty: Medium
* Tags: general
*
* 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 makeTheIntegerZero(int num1, int num2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make the Integer Zero
* Difficulty: Medium
* Tags: general
*
* 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 makeTheIntegerZero(int num1, int num2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make the Integer Zero
Difficulty: Medium
Tags: general

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 makeTheIntegerZero(self, num1: int, num2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make the Integer Zero
* Difficulty: Medium
* Tags: general
*
* 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 makeTheIntegerZero = function(num1, num2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make the Integer Zero
* Difficulty: Medium
* Tags: general
*
* 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 makeTheIntegerZero(num1: number, num2: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make the Integer Zero
* Difficulty: Medium
* Tags: general
*
* 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 MakeTheIntegerZero(int num1, int num2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make the Integer Zero
* Difficulty: Medium
* Tags: general
*
* 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 makeTheIntegerZero(int num1, int num2) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make the Integer Zero
// Difficulty: Medium
// Tags: general
//
// 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 makeTheIntegerZero(num1 int, num2 int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make the Integer Zero
// Difficulty: Medium
// Tags: general
//
// 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 make_the_integer_zero(num1: i32, num2: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (make-the-integer-zero num1 num2)
(-> exact-integer? exact-integer? exact-integer?)
)
```