Problem 29: Divide Two Integers

Problem Information

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

Problem Description

Given two integers

dividend

and

divisor

, divide two integers

without

using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example,

8.345

would be truncated to

8

, and

-2.7335

would be truncated to

-2

.

Return

the

quotient

after dividing

dividend

by

divisor

.

Note:

Assume we are dealing with an environment that could only store integers within the

32-bit

signed integer range:

[–2

31

, 2

31

– 1]

. For this problem, if the quotient is

strictly greater than

2

31

- 1

, then return

2

31

- 1

, and if the quotient is

strictly less than

-2

31

, then return

-2

31

.

Example 1:

Input:

dividend = 10, divisor = 3

Output:

3

Explanation:

10/3 = 3.33333.. which is truncated to 3.

Example 2:

Input:

dividend = 7, divisor = -3

Output:

-2

Explanation:

7/-3 = -2.33333.. which is truncated to -2.

Constraints:

-2

31

<= dividend, divisor <= 2

31

- 1

divisor != 0

Code Snippets

C++:

```
class Solution {
public:
int divide(int dividend, int divisor) {

}
};
```

Java:

```
class Solution {
public int divide(int dividend, int divisor) {

}
}
```

Python3:

```
class Solution:
def divide(self, dividend: int, divisor: int) -> int:
```

Python:

```
class Solution(object):
def divide(self, dividend, divisor):
"""
:type dividend: int
:type divisor: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} dividend
* @param {number} divisor
* @return {number}
*/
var divide = function(dividend, divisor) {


};
```

TypeScript:

```
function divide(dividend: number, divisor: number): number {

};
```

C#:

```
public class Solution {
public int Divide(int dividend, int divisor) {

}
}
```

C:

```
int divide(int dividend, int divisor) {

}
```

Go:

```
func divide(dividend int, divisor int) int {

}
```

Kotlin:

```
class Solution {
fun divide(dividend: Int, divisor: Int): Int {

}
}
```

Swift:

```
class Solution {
func divide(_ dividend: Int, _ divisor: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn divide(dividend: i32, divisor: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} dividend
# @param {Integer} divisor
# @return {Integer}
def divide(dividend, divisor)

end
```

PHP:

```
class Solution {

/**
* @param Integer $dividend
* @param Integer $divisor
* @return Integer
*/
function divide($dividend, $divisor) {

}
}
```

Dart:

```
class Solution {
int divide(int dividend, int divisor) {

}
}
```

Scala:

```
object Solution {
def divide(dividend: Int, divisor: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec divide(dividend :: integer, divisor :: integer) :: integer
def divide(dividend, divisor) do

end
end
```

Erlang:

```
-spec divide(Dividend :: integer(), Divisor :: integer()) -> integer().
divide(Dividend, Divisor) ->
.
```

Racket:

```
(define/contract (divide dividend divisor)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Two Integers
* Difficulty: Medium
* 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 divide(int dividend, int divisor) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Two Integers
* Difficulty: Medium
* 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 divide(int dividend, int divisor) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide Two Integers
Difficulty: Medium
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 divide(self, dividend: int, divisor: int) -> int:
# TODO: Implement optimized solution
```

```

pass
```

Python Solution:

```
class Solution(object):
def divide(self, dividend, divisor):
"""
:type dividend: int
:type divisor: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Divide Two Integers
* Difficulty: Medium
* 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} dividend
* @param {number} divisor
* @return {number}
*/
var divide = function(dividend, divisor) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide Two Integers
* Difficulty: Medium
* 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 divide(dividend: number, divisor: number): number {

};
```

C# Solution:

```
/*
* Problem: Divide Two Integers
* Difficulty: Medium
* 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 Divide(int dividend, int divisor) {

}
}
```

C Solution:

```
/*
* Problem: Divide Two Integers
* Difficulty: Medium
* 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 divide(int dividend, int divisor) {

}
```

Go Solution:

```
// Problem: Divide Two Integers
// Difficulty: Medium
// 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 divide(dividend int, divisor int) int {

}
```

Kotlin Solution:

```
class Solution {
fun divide(dividend: Int, divisor: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func divide(_ dividend: Int, _ divisor: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Divide Two Integers
// Difficulty: Medium
// 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 divide(dividend: i32, divisor: i32) -> i32 {
}
}
```

Ruby Solution:

```
# @param {Integer} dividend
# @param {Integer} divisor
# @return {Integer}
def divide(dividend, divisor)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $dividend
* @param Integer $divisor
* @return Integer
*/
function divide($dividend, $divisor) {

}
}
```

Dart Solution:

```
class Solution {
int divide(int dividend, int divisor) {

}
}
```

Scala Solution:

```
object Solution {
def divide(dividend: Int, divisor: Int): Int = {

}

}
```

Elixir Solution:

```
defmodule Solution do
@spec divide(dividend :: integer, divisor :: integer) :: integer
def divide(dividend, divisor) do

end
end
```

Erlang Solution:

```
-spec divide(Dividend :: integer(), Divisor :: integer()) -> integer().
divide(Dividend, Divisor) ->
.
```

Racket Solution:

```
(define/contract (divide dividend divisor)
(-> exact-integer? exact-integer? exact-integer?)
)
```