Problem 166: Fraction to Recurring Decimal

Problem Information

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

Problem Description

Given two integers representing the

numerator

and

denominator

of a fraction, return

the fraction in string format

.

If the fractional part is repeating, enclose the repeating part in parentheses

If multiple answers are possible, return

any of them

.

It is

guaranteed

that the length of the answer string is less than

10

4

for all the given inputs.

Note

that if the fraction can be represented as a

finite length string

, you

must

return it.

Example 1:

Input:

numerator = 1, denominator = 2

Output:

"0.5"

Example 2:

Input:

numerator = 2, denominator = 1

Output:

"2"

Example 3:

Input:

numerator = 4, denominator = 333

Output:

"0.(012)"

Constraints:

-2

31

<= numerator, denominator <= 2

31

- 1

denominator != 0

Code Snippets

C++:

```
class Solution {
public:
string fractionToDecimal(int numerator, int denominator) {

}
};
```

Java:

```
class Solution {
public String fractionToDecimal(int numerator, int denominator) {
```

```
}
}
```

Python3:

```
class Solution:
def fractionToDecimal(self, numerator: int, denominator: int) -> str:
```

Python:

```
class Solution(object):
def fractionToDecimal(self, numerator, denominator):
"""
:type numerator: int
:type denominator: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number} numerator
* @param {number} denominator
* @return {string}
*/
var fractionToDecimal = function(numerator, denominator) {

};
```

TypeScript:

```
function fractionToDecimal(numerator: number, denominator: number): string {

};
```

C#:

```
public class Solution {
public string FractionToDecimal(int numerator, int denominator) {

}
```

```
}
```

C:

```
char* fractionToDecimal(int numerator, int denominator) {

}
```

Go:

```
func fractionToDecimal(numerator int, denominator int) string {

}
```

Kotlin:

```
class Solution {
fun fractionToDecimal(numerator: Int, denominator: Int): String {

}
}
```

Swift:

```
class Solution {
func fractionToDecimal(_ numerator: Int, _ denominator: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn fraction_to_decimal(numerator: i32, denominator: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer} numerator
# @param {Integer} denominator
```

```
# @return {String}
def fraction_to_decimal(numerator, denominator)

end
```

PHP:

```
class Solution {

/**
* @param Integer $numerator
* @param Integer $denominator
* @return String
*/
function fractionToDecimal($numerator, $denominator) {

}
}
```

Dart:

```
class Solution {
String fractionToDecimal(int numerator, int denominator) {

}
}
```

Scala:

```
object Solution {
def fractionToDecimal(numerator: Int, denominator: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec fraction_to_decimal(numerator :: integer, denominator :: integer) ::
String.t
def fraction_to_decimal(numerator, denominator) do
```

```
end
end
```

Erlang:

```
-spec fraction_to_decimal(Numerator :: integer(), Denominator :: integer())
-> unicode:unicode_binary().
fraction_to_decimal(Numerator, Denominator) ->
.
```

Racket:

```
(define/contract (fraction-to-decimal numerator denominator)
(-> exact-integer? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Fraction to Recurring Decimal
* Difficulty: Medium
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string fractionToDecimal(int numerator, int denominator) {

}
};
```

Java Solution:

```
/**
* Problem: Fraction to Recurring Decimal
* Difficulty: Medium
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String fractionToDecimal(int numerator, int denominator) {

}
}
```

Python3 Solution:

```
"""
Problem: Fraction to Recurring Decimal
Difficulty: Medium
Tags: string, math, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def fractionToDecimal(self, numerator: int, denominator: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def fractionToDecimal(self, numerator, denominator):
"""
:type numerator: int
:type denominator: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Fraction to Recurring Decimal
* Difficulty: Medium
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} numerator
* @param {number} denominator
* @return {string}
*/
var fractionToDecimal = function(numerator, denominator) {

};
```

TypeScript Solution:

```
/**
* Problem: Fraction to Recurring Decimal
* Difficulty: Medium
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function fractionToDecimal(numerator: number, denominator: number): string {

};
```

C# Solution:

```
/*
* Problem: Fraction to Recurring Decimal
* Difficulty: Medium
```

```
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string FractionToDecimal(int numerator, int denominator) {

}
}
```

C Solution:

```
/*
* Problem: Fraction to Recurring Decimal
* Difficulty: Medium
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* fractionToDecimal(int numerator, int denominator) {

}
```

Go Solution:

```
// Problem: Fraction to Recurring Decimal
// Difficulty: Medium
// Tags: string, math, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func fractionToDecimal(numerator int, denominator int) string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun fractionToDecimal(numerator: Int, denominator: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func fractionToDecimal(_ numerator: Int, _ denominator: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Fraction to Recurring Decimal
// Difficulty: Medium
// Tags: string, math, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn fraction_to_decimal(numerator: i32, denominator: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer} numerator
# @param {Integer} denominator
# @return {String}
def fraction_to_decimal(numerator, denominator)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $numerator
* @param Integer $denominator
* @return String
*/
function fractionToDecimal($numerator, $denominator) {

}
}
```

Dart Solution:

```
class Solution {
String fractionToDecimal(int numerator, int denominator) {

}
}
```

Scala Solution:

```
object Solution {
def fractionToDecimal(numerator: Int, denominator: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec fraction_to_decimal(numerator :: integer, denominator :: integer) ::
String.t
def fraction_to_decimal(numerator, denominator) do

end
end
```

Erlang Solution:

```
-spec fraction_to_decimal(Numerator :: integer(), Denominator :: integer())
-> unicode:unicode_binary().
fraction_to_decimal(Numerator, Denominator) ->
.
```

Racket Solution:

```
(define/contract (fraction-to-decimal numerator denominator)
(-> exact-integer? exact-integer? string?)
)
```