Problem 2485: Find the Pivot Integer

Problem Information

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

Problem Description

Given a positive integer

n

, find the

pivot integer

x

such that:

The sum of all elements between

1

and

x

inclusively equals the sum of all elements between

x

and

n

inclusively.

Return

the pivot integer

x

. If no such integer exists, return

-1

. It is guaranteed that there will be at most one pivot index for the given input.

Example 1:

Input:

n = 8

Output:

6

Explanation:

6 is the pivot integer since: 1 + 2 + 3 + 4 + 5 + 6 = 6 + 7 + 8 = 21.

Example 2:

Input:

n = 1

Output:

1

Explanation:

1 is the pivot integer since: 1 = 1.

Example 3:

Input:

n = 4

Output:

-1

Explanation:

It can be proved that no such integer exist.

Constraints:

1 <= n <= 1000

Code Snippets

C++:

```
class Solution {
public:
int pivotInteger(int n) {

}
};
```

Java:

```
class Solution {
public int pivotInteger(int n) {

}
}
```

Python3:

```
class Solution:
def pivotInteger(self, n: int) -> int:
```

Python:

```
class Solution(object):
def pivotInteger(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var pivotInteger = function(n) {

};
```

TypeScript:

```
function pivotInteger(n: number): number {

};
```

C#:

```
public class Solution {
public int PivotInteger(int n) {

}
}
```

C:

```
int pivotInteger(int n) {

}
```

Go:

```
func pivotInteger(n int) int {

}
```

Kotlin:

```
class Solution {
fun pivotInteger(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func pivotInteger(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn pivot_integer(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def pivot_integer(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function pivotInteger($n) {

}
}
```

Dart:

```
class Solution {
int pivotInteger(int n) {

}
}
```

Scala:

```
object Solution {
def pivotInteger(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec pivot_integer(n :: integer) :: integer
def pivot_integer(n) do

end
end
```

Erlang:

```
-spec pivot_integer(N :: integer()) -> integer().
pivot_integer(N) ->
.
```

Racket:

```
(define/contract (pivot-integer n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Pivot Integer
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int pivotInteger(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Pivot Integer
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int pivotInteger(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Pivot Integer
Difficulty: Easy
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def pivotInteger(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def pivotInteger(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Pivot Integer
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* @param {number} n
* @return {number}
*/
var pivotInteger = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Pivot Integer
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function pivotInteger(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Pivot Integer
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int PivotInteger(int n) {

}
}
```

C Solution:

```
/*
* Problem: Find the Pivot Integer
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int pivotInteger(int n) {

}
```

Go Solution:

```
// Problem: Find the Pivot Integer
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func pivotInteger(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun pivotInteger(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func pivotInteger(_ n: Int) -> Int {
}
}
```

Rust Solution:

```
// Problem: Find the Pivot Integer
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn pivot_integer(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def pivot_integer(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function pivotInteger($n) {

}
}
```

Dart Solution:

```
class Solution {
int pivotInteger(int n) {

}
}
```

Scala Solution:

```
object Solution {
def pivotInteger(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec pivot_integer(n :: integer) :: integer
def pivot_integer(n) do

end
end
```

Erlang Solution:

```
-spec pivot_integer(N :: integer()) -> integer().
pivot_integer(N) ->
.
```

Racket Solution:

```
(define/contract (pivot-integer n)
(-> exact-integer? exact-integer?)
)
```