2485. Find the Pivot Integer

Easy Math Prefix Sum

Problem Description

The problem presents an arithmetic challenge where we are given a positive integer `n`, and we are tasked to find a special integer `x`, referred to as the "pivot integer". The pivot integer `x` has a unique property: the sum of all the integers from `1` to `x` (inclusive) is equal to the sum of all the integers from `x` to `n` (inclusive). Our goal is to determine the value of `x` that satisfies this condition.

If such a pivot integer exists, then we should return the value of `x`; if not, we are directed to return `-1`. The problem also assures us that there would be at most one such pivot integer for any given input `n`.

Our challenge is to find the most efficient way to calculate this pivot integer without having to iterate through all possible values, which would be computationally expensive especially for large values of `n`.

Intuition

The key to solving this problem lies in understanding the properties of arithmetic sequences and leveraging mathematical formulas to simplify the calculations.

Given the sequential nature of numbers from `1` to `n`, we know that the sum of these numbers forms an arithmetic series. The formula to find the sum of the first `n` natural numbers is `n * (n + 1) / 2`. If we are to find a pivot `x`, the sums on either side of `x` must both equal half of the total sum of numbers from `1` to `n`.

A direct approach to find the sum of numbers from `1` to `x` and the sum of numbers from `x` to `n` would typically involve complex calculations or iterative methods. However, by using the summation formula above, we can simplify the search for the pivot integer.

Since the sum on both sides of the pivot `x` should equal, we essentially need to find a number `x` such that `x * (x + 1) / 2` equals half of the sum of all integers from `1` to `n`. This leads us to the equation `x * (x + 1) = n * (n + 1)`.

The implementation simplifies this by computing the total sum `y` as `n * (n + 1) / 2`, then solving for `x` by taking the square root of `y` (as the equation would be `x * x = y`). If the square of `x` perfectly equals `y` (meaning `x` is a perfect square root of `y`), `x` is the pivot integer. Otherwise, if no such `x` exists (the square root is not a whole number), the function returns `-1`.

Thus, the solution efficiently solves for the pivot integer by reducing the problem to a simple equation that can be solved with basic arithmetic and avoids the need for iterative computation.

Solution Approach

The solution approach for finding the pivot integer efficiently utilizes mathematical reasoning rather than relying on data structures or iterative algorithms.

To understand the implementation, let's first break down the mathematical approach into steps:

1. Calculate the total sum (`y`) of all numbers from `1` to `n` using the formula `n * (n + 1) / 2`. Since the series of numbers from `1` to `n` is an arithmetic progression, this formula gives us the sum swiftly without the need for iteration.
2. Once we have the total sum `y`, we need to find a number `x` such that the sum of numbers from `1` to `x` — which can be given by `x * (x + 1) / 2` — is equal to half of `y`. That would mean that `x` is a pivot since the sum from `x` to `n` would also be equal to that amount (because the total is `y`, and we're seeking a midpoint).
3. The key insight here is recognizing that if such an `x` exists, then `x * (x + 1)` must equal `n * (n + 1)`. In essence, we are looking for `x` which makes `x * (x + 1) / 2` equal to `y`.
4. To find `x`, the solution simply calculates the square root of `y`, because the equation `x * (x + 1) / 2 = y` can be rearranged to `x^2 + x - 2y = 0`, which is a quadratic equation in the form `ax^2 + bx + c = 0`. Since the equation is symmetric around `x`, and we're solving for `x^2`, finding the square root of `y` gives us `x` directly.
5. The solution checks whether `x` is a whole number and whether `x * x` exactly equals `y`. If so, `x` is returned as it is guaranteed to be our pivot. Otherwise, the function returns `-1` indicating no such pivot integer exists. This is checked with the line `x * x == y`.

The elegance of this approach is that it completely bypasses the need for any iterative searching or complex data structures. It only uses basic arithmetic operations and the `sqrt` function from the `math` module to calculate the square root. This provides a highly efficient solution especially for large values of `n`, as the time complexity is constant, O(1), with respect to `n`.

Here's the code snippet from the implementation demonstrating the process:

```
from [math](/problems/math-basics) import sqrt

class Solution:
    def pivotInteger(self, n: int) -> int:
        # Calculate the total sum of numbers from 1 to n
        y = n * (n + 1) // 2
        # Find the potential square root of y (our pivot x)
        x = int(sqrt(y))
        # Check if x is indeed the pivot
        return x if x * x == y else -1
```

The code uses integer division `//` to avoid floating-point results since we are dealing with summation of integers. The `sqrt` function is used to find `x`, and then an integer conversion is necessary since `sqrt` returns a float. The implementation checks if `x` squared equals `y` to confirm that `x` is our pivot integer.

Example Walkthrough

Let's consider `n = 8` as an example to illustrate the solution approach. We are tasked with finding a pivot integer `x` where the sum of integers from `1` to `x` is equal to the sum of integers from `x` to `8`.

1. We first calculate the total sum `y` of all numbers from `1` to `8` using the formula `n * (n + 1) / 2`. Applying the formula gives us:

   `y = 8 * (8 + 1) / 2 y = 72 / 2 y = 36`

2. Now, we need to find a number `x` such that when we take the sum of numbers from `1` to `x`, it is equal to half of `y`. In other words, `x` should satisfy the equation `x * (x + 1) / 2 = 36`.

3. We then compute the square root of `y` since the equation `x * (x + 1) / 2 = y` can be rearranged into `x^2 + x - 2y = 0`, which implies `x^2 = y` when `x` is the pivot.

   Since `y = 36`, the square root of `y` is `x = sqrt(36) = 6`.

4. To verify if 6 is indeed the pivot, we check if `x * x` equals `y`. We have:

   `x * x = 6 * 6 = 36`

   Since `36` is the total sum `y`, this confirms that `6` is the pivot integer. Both sides of the pivot will have the same sum:

   Sum from `1` to `6` = `6 * (6 + 1) / 2` = `21` Sum from `6` to `8` = `6 + 7 + 8` = `21`

As a result, `x = 6` is the pivot integer for `n = 8`, as it satisfies the property that the sum of numbers from `1` to `x` is equal to the sum of numbers from `x` to `n`. The method works efficiently for this example and can be similarly applied to any positive integer `n`.

Solution Implementation

Python

```
from math import sqrt

class Solution:
    def pivotInteger(self, n: int) -> int:
        # Calculate the sum of all numbers from 1 to n using the formula for the sum of an arithmetic series
        sum_of_numbers = n * (n + 1) // 2

        # Check if the sum is a perfect square by taking the square root and checking if the squared root equals the sum
        square_root = int(sqrt(sum_of_numbers))

        # Return the square root if it's a perfect square, otherwise return -1
        return square_root if square_root * square_root == sum_of_numbers else -1
```

Java

```
class Solution {
    /**
     * This method finds a pivot integer 'x' such that the sum of numbers from 1 to 'x' is a perfect square.
     * If such an integer does not exist, it returns -1.
     *
     * @param n The upper limit of the number range to check for a pivot integer.
     * @return The pivot integer 'x' if found, otherwise -1.
     */
    public int pivotInteger(int n) {
        // Calculate the sum of the integers from 1 to n using the formula n*(n+1)/2
        int sum = n * (n + 1) / 2;

        // Find the square root of the sum
        int sqrtOfSum = (int) Math.sqrt(sum);

        // Check if the square of the square root equals the sum,
        // which would mean the sum is a perfect square
        if (sqrtOfSum * sqrtOfSum == sum) {
            // If sum is a perfect square, the square root is our pivot integer x
            return sqrtOfSum;
        } else {
            // If sum is not a perfect square, return -1
            return -1;
        }
    }
}
```

C++

```
#include <cmath> // Include cmath library for sqrt function

class Solution {
public:
    // Method to find the pivot integer for a given n
    int pivotInteger(int n) {
        // Calculate the sum of all integers from 1 to n
        int sum = n * (n + 1) / 2;

        // Find the integer square root of the sum, if it exists
        int squareRoot = static_cast<int>(sqrt(sum));

        // Check if the square of the square root equals the oriainal sum
        // If so, the pivot inteoer exists and is equal to squareRoot
        if (squareRoot * squareRoot == sum) {
            return squareRoot;
        }
        // If no such pivot integer exists, return -1
        return -1;
    }
};
```

TypeScript

```
// The function pivotInteoer calculates the pivot inteoer for a given 'n'.
// A pivot inteaer is the laroest inteaer 'x' such that x^2 is equal to the sum of all
// inteaers from 1 to 'n' inclusive. If no such 'x' exists, the function returns -1.
function pivotInteger(n: number): number {
    // Calculate the sum of all integers from 1 to 'n' using the formula S = n(n + 1)/2.
    const sumOfIntegers = Math.floor((n * (n + 1)) / 2);

    // Find the inteaer part of the square root of the sum.
    const integerRoot = Math.floor(Math.sqrt(sumOfIntegers));

    // Check if the square of the inteaer root is exactlv equal to the sum of integers.
    // If it is, then 'integerRoot' is the pivot inteaer we're lookina for.
    return integerRoot * integerRoot === sumOfIntegers ? integerRoot : -1;
}
```

```
from math import sqrt

class Solution:
    def pivotInteger(self, n: int) -> int:
        # Calculate the sum of all numbers from 1 to n using the formula for the sum of an arithmetic series
        sum_of_numbers = n * (n + 1) // 2

        # Check if the sum is a perfect square by taking the square root and checking if the squared root equals the sum
        square_root = int(sqrt(sum_of_numbers))

        # Return the square root if it's a perfect square, otherwise return -1
        return square_root if square_root * square_root == sum_of_numbers else -1
```

Time and Space Complexity

The time complexity of the provided code is `O(1)` because the computation of the sum of the first `n` integers using the formula `y = n * (n + 1) // 2` is a direct arithmetic calculation which happens in constant time irrespective of the value of `n`. The subsequent operation to check if `y` is a perfect square is also `O(1)` since it involves calculating the square root `x` of `y` and then checking if `x * x == y`, both of which are constant time operations.

The space complexity of the code is also `O(1)`. This is because the amount of memory used does not depend on the input size `n`; it uses a fixed number of variables (`y` and `x`).