326. Power of Three

Easy Recursion Math

Problem Description

The task is to determine if a given integer `n` is a power of three. In other words, you need to check if there exists an integer `x` such that when you raise 3 to the power of `x` (`3^x`), the result is `n`. This is a binary decision problem where the output is either `true` if `n` is a power of three or `false` if it is not.

Intuition

The intuition behind the solution is based on number theory. First, notice that if `n` is less than or equal to zero, it cannot be a power of three. Hence, we first check that `n` is positive. Next, we observe that if `n` is a power of three, `n` must be a divisor of the largest power of three that fits in a signed 32-bit integer, because the powers of three are spread at constant ratios and every power of three is a multiple of smaller powers of three.

The number `1162261467` is used in the provided solution because it is the largest power of three (`3^19`) that fits in a signed 32-bit integer range. If `n` is truly a power of three, it must divide this number without leaving a remainder. This is based on the properties of exponents in which any lower power of the base (in this case, three) will be a factor of the higher powers of the base.

The condition `1162261467 % n == 0` is checking for this exact scenario. If `n` is a power of three, the modulo operation will result in zero meaning `n` divides `1162261467` exactly, validating that `n` is a power of three.

Solution Approach

The implementation of the solution is straightforward and does not require complex algorithms or data structures. It is a matter of a single comparison and a modulo operation. Here's a step-by-step walkthrough of the code provided:

1. The condition `n > 0` ensures that we disregard any non-positive integers. Numbers less than or equal to zero cannot be a power of three, so this condition quickly handles these cases by returning `false`.
2. For positive values of `n`, the solution uses the modulo operator `%` to check if `n` is a divisor of `1162261467`, the largest power of three (`3^19`) within the 32-bit signed integer range.
3. `1162261467 % n == 0` is the key operation. The modulo operator calculates the remainder of the division of `1162261467` by `n`. If the result is `0`, it implies that `n` divides the number evenly, indicating that `n` is a power of three. If the remainder is not `0`, `n` is not a power of three because it does not divide the largest 32-bit power of three exactly.

The algorithm's efficiency lies in its O(1) time complexity since it requires a constant number of operations irrespective of the size of `n`. There is also O(1) space complexity, as no additional space is needed besides the input and the single boolean output. The pattern here is using a mathematical property of powers to avoid iteration or recursion, yielding an elegant and efficient solution.

Example Walkthrough

Let's illustrate the solution approach with an example by determining if the number `n = 27` is a power of three.

1. First, we verify that `n` is positive. Since 27 is greater than 0, we proceed to the next step.
2. We know that the largest power of three that can be contained in a 32-bit signed integer is 3^19, which is 1162261467.
3. We then perform the modulus operation with `n` and this number: `1162261467 % 27`.
4. Calculating the modulus, we get 1162261467 % 27 = 0. Since there is no remainder, this operation confirms that 27 is a divisor of 1162261467.
5. Given the result of the modulus operation is zero, it indicates that 27 is a power of three, and therefore, our function should return `true`.

In this example, we've applied the provided solution to demonstrate that 27 is indeed a power of three. The steps are concise, avoiding the need for loops or recursion, and taking constant time to complete, which is an efficient way to reach the answer.

Solution Implementation

Python

```python
class Solution:
    def isPowerOfThree(self, number: int) -> bool:
        # The largest power of 3 value that fits in a 32-bit signed integer is 3^19, which is 1162261467.
        # If 'number' is a power of 3, it must divide 1162261467 without any remainder.
        # 'number' must be positive since 0 and negative numbers cannot be powers of 3.

        # We check that the number is positive and that the modulo operation of 1162261467 by 'number' is zero.
        # If the result is zero, 'number' is a divisor of 1162261467, implying it is a power of 3.
        return number > 0 and 1162261467 % number == 0
```

Java

```java
class Solution {
    // Method to check if a number is a power of three
    public boolean isPowerOfThree(int n) {
        // 1162261467 is the maximum integer that is a power of three
        // (it is 3^19, as 3^20 is bigger than int).
        // If 'n' is a power of three, it must divide this number without a remainder.

        // The condition 'n > 0' ensures that 'n' is positive,
        // because negative numbers and zero cannot be powers of three.
        return n > 0 && 1162261467 % n == 0;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to check if a given number is a power of three
    bool isPowerOfThree(int n) {
        // 1162261467 is the largest integer that is a power of three (3^19)
        // If n is a power of three, it must be a divisor of 1162261467.

        // Check if n is positive and if 1162261467 is divisible by n
        return n > 0 && 1162261467 % n == 0;
    }
};
```

TypeScript

```typescript
// Determines if the given number is a power of three.
// Uses the largest power of three that fits in a 32-bit signed integer to check divisibility.
// @param {number} n The number to check.
// @returns {boolean} True if the number is a power of three, otherwise false.
function isPowerOfThree(n: number): boolean {
    // The constant is the highest power of three (3^19) that fits in a 32-bit signed integer.
    const maxPowerOfThree: number = 1162261467;

    // A number is a power of three if it is positive and the maximum power of three (3^19) is divisible by it.
    return n > 0 && maxPowerOfThree % n === 0;
}
```

```python
class Solution:
    def isPowerOfThree(self, number: int) -> bool:
        # The largest power of 3 value that fits in a 32-bit signed integer is 3^19, which is 1162261467.
        # If 'number' is a power of 3, it must divide 1162261467 without any remainder.
        # 'number' must be positive since 0 and negative numbers cannot be powers of 3.

        # We check that the number is positive and that the modulo operation of 1162261467 by 'number' is zero.
        # If the result is zero, 'number' is a divisor of 1162261467, implying it is a power of 3.
        return number > 0 and 1162261467 % number == 0
```

Time and Space Complexity

Time Complexity

The time complexity of the provided code snippet is `O(1)`. This is because the operation involves a single modulo operation, which is executed in constant time regardless of the value of `n`.

Space Complexity

The space complexity of the code is also `O(1)`. The algorithm does not use any additional space that scales with the input size, as it only involves checking a condition on the input number `n` and a fixed integer `1162261467`, which is the largest power of 3 that fits within the 32-bit signed integer range.