2413. Smallest Even Multiple

Easy Math Number Theory

Problem Description

The given problem requires determining the smallest positive integer that can be divided evenly (without leaving a remainder) by both `2` and a given positive integer `n`. This integer is essentially the least common multiple (LCM) of 2 and `n`. Since the question only involves `2` as one of the numbers, it simplifies the conditions for the LCM. If `n` is already an even number, then it itself is the least multiple of both `2` and `n` (as any even number is divisible by `2`). However, if `n` is odd, then the smallest even multiple is simply twice the value of `n`, as multiplying an odd number by `2` yields the smallest even number which is a multiple of both `2` and the odd number.

Intuition

To solve this problem, we check if `n` is already an even number. An even number is characterized by having no remainder when divided by `2`. We can check this by performing `n % 2` and see if the result is `0`. If `n` is even, then `n` is our answer because `n` would be the smallest number that is a multiple of both itself and `2`.

If `n` is not even (implying it's odd), then the smallest even multiple that is also a multiple of `n` would be twice `n`. This is because the only way to make an odd number even (and therefore a multiple of `2`) without changing its divisibility by `n` is to multiply it by `2`.

Thus, the solution can be succinctly implemented with a single line of code that returns `n` if it's even, or `n * 2` if it's odd.

Solution Approach

The provided solution is straightforward and does not require complicated algorithms or data structures. It leverages a straightforward mathematical fact that every multiple of an even number is even, and the smallest even multiple of an odd number is the number itself multiplied by `2`.

Here is a step-by-step breakdown of the implementation process:

1. Check if the given integer `n` is even by using the modulo operator `%`. In Python, `n % 2 == 0` is `True` if `n` is divisible by `2` without any remainder. It is a common way to determine if a number is even.
2. If `n` is even, the function returns `n` itself since `n` is the smallest even number that satisfies the condition of being a multiple of both `n` and `2`.
3. If `n` is odd (the check resulted in `False`), multiply `n` by `2` to find the smallest even multiple of `n`. Odd numbers can only yield even numbers when multiplied by an even number, and multiplying by `2` ensures the smallest possible outcome.

Please note that there is no explicit loop or conditional structures needed; the solution uses a ternary-like expression in Python that is a concise way to write an if-else statement on a single line.

The equation: `return n if n % 2 == 0 else n * 2` covers the solution approach completely.

In this specific problem, the solution's time complexity is constant, `O(1)`, because the calculation requires a maximum of two operations regardless of the size of `n`. The space complexity is also constant, `O(1)`, as it does not require any additional space that depends on the input size.

Example Walkthrough

Let's consider an example to illustrate the solution approach. Suppose we are given `n = 7`, which is an odd number.

1. First, we check if `n` is even by performing `n % 2`. For our example, `7 % 2` equals `1`, which is not zero. So, `n` is not even.
2. Since `n` is odd, our solution should return `n * 2`. Hence, we multiply `7` by `2`, which gives us `14`.
3. The number `14` is the smallest positive integer that is even and can be divided evenly by both `2` and `7` (since the other divisor is `2`, and `14` is already an even number).

Therefore, the function will return `14` as the answer in this case.

Now let's take an even number for `n`, for instance, `n = 6`.

1. We perform the modulo operation again: `6 % 2`. This time it equals `0`, which means `n` is even.
2. Since `n` is even, the smallest even multiple of `2` that can also be divided by `n` is `n` itself. There is no need to perform any multiplication.
3. The function will return `6` as the least common multiple that meets the problem's conditions.

The key takeaway here is the simple check for evenness, which dictates whether we can return `n` directly or need to return `n * 2`. This example walkthrough demonstrates the directness and efficiency of the solution approach for both odd and even input scenarios.

Solution Implementation

Python

```python
class Solution:
    def smallestEvenMultiple(self, n: int) -> int:
        # Check if n is even by using the modulo operator.
        # If n is even, it is already the smallest even multiple, so we return it.
        if n % 2 == 0:
            return n
        else:
            # If n is odd, the smallest even multiple is n multiplied by 2.
            return n * 2
```

Java

```java
class Solution {
    // Method to calculate the smallest even multiple of a given number n
    public int smallestEvenMultiple(int n) {
        // Check if n is already even by checking remainder when divided by 2
        if (n % 2 == 0) {
            // If n is even, then it is the smallest even multiple of itself
            return n;
        } else {
            // If n is odd, then the smallest even multiple is n times 2
            return n * 2;
        }
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to find the smallest even multiple of a given number 'n'
    int smallestEvenMultiple(int n) {
        // If 'n' is already even, it is its own smallest even multiple
        if (n % 2 == 0) {
            return n;
        }
        // If 'n' is odd, double 'n' to get the smallest even multiple
        return n * 2;
    }
};
```

TypeScript

```typescript
// This function finds the smallest even multiple of a given number.
// If 'n' is already even, it returns 'n'; else it returns 'n' multiplied by 2.
function smallestEvenMultiple(n: number): number {
    // Check if 'n' is even using modulo operator
    if (n % 2 === 0) {
        // 'n' is even, so return 'n' itself
        return n;
    } else {
        // 'n' is odd, so return 'n' doubled to make it even
        return n * 2;
    }
}
```

```python
class Solution:
    def smallestEvenMultiple(self, n: int) -> int:
        # Check if n is even by using the modulo operator.
        # If n is even, it is already the smallest even multiple, so we return it.
        if n % 2 == 0:
            return n
        else:
            # If n is odd, the smallest even multiple is n multiplied by 2.
            return n * 2
```

Time and Space Complexity

The given code consists of a single function `smallestEvenMultiple` which takes an integer `n` and returns the smallest even multiple of `n`.

Time Complexity

The time complexity of this function is `O(1)` which means that it runs in constant time. This is because no matter the size of the input `n`, the function performs a maximum of one comparison (`n % 2 == 0`) and possibly one multiplication (`n * 2`). Both operations are basic arithmetic operations that take constant time to complete.

Space Complexity

The space complexity of this function is also `O(1)`. The function uses a fixed amount of space; it allocates space for just one integer (the return value), and the amount of memory used does not scale with the input size `n`.

The overall performance of the code is thus highly efficient, with both time and space complexities being constant.