

```
def isPowerOfTwo(n: int) -> bool:
    # Edge case: 0 is not a power of two
    if n == 0:
        return False

    # Check if the rightmost bit is the only bit set
    return (n & (n - 1)) == 0
```

Explanation:

1. The edge case is when n is 0, which is not a power of two, so we return False in this case.
2. The key insight here is that a number is a power of two if and only if it has a single bit set in its binary representation. For example:
 - 1 (2^0) has the rightmost bit set: 0001
 - 2 (2^1) has the second rightmost bit set: 0010
 - 4 (2^2) has the third rightmost bit set: 0100
 - 8 (2^3) has the fourth rightmost bit set: 1000
3. To check if a number has a single bit set, we can use the bitwise AND operation & between the number and the number minus 1. If the result is 0, it means the number has a single bit set, and it is a power of two.
 - For example, let's take 16 (2^4):
 - 16 in binary is 10000
 - $16 - 1 = 15$, which in binary is 01111
 - $16 \& 15$ is 0, which means 16 is a power of two.
 - For 3, which is not a power of two:
 - 3 in binary is 11
 - $3 - 1 = 2$, which in binary is 10
 - $3 \& 2$ is 2, which is not 0, so 3 is not a power of two.

The time complexity of this solution is $O(1)$, as it only performs a constant number of bitwise operations, regardless of the input size. The space complexity is $O(1)$ as well, as it only uses a constant amount of extra space.