LeetCode 231 — Power of Two

1. Problem Title & Link

- **Title:** LeetCode 231 — Power of Two
- **Link:** https://leetcode.com/problems/power-of-two/

2. Problem Statement (Short Summary)

Given an integer n, return **true** if it is a **power of two**, false otherwise.

A number is a power of two if:

n = 1, 2, 4, 8, 16, 32, ...

In binary:

1 -> 0001

2 -> 0010

4 -> 0100

8 -> 1000

...

It has exactly **one 1-bit**.

3. Examples (Input → Output)

Input: n = 1

Output: true

Input: n = 16

Output: true

Input: n = 3

Output: false

Input: n = 0

Output: false

Input: n = -4

Output: false

4. Constraints

- -2^31 <= n <= 2^31 - 1
- Negative numbers **cannot** be powers of two.

- Zero is **not** a power of two.

5. Core Concept (Pattern / Topic)

Bit Manipulation — Single Set Bit Check

A power of two in binary has **exactly one set bit**.

The classic trick:

n & (n - 1) == 0

This expression removes the lowest set bit.

If the number becomes zero → it had only one set bit.

But ensure:

n > 0

6. Thought Process (Step-by-Step Explanation)

Approach 1 (bit trick):

1. Check if n > 0
2. Check if n & (n-1) == 0

If both true → power of two.

Approach 2 (loop divide by 2):

Divide repeatedly until you get 1 — works but slower.

Approach 3 (count bits):

Count ones in binary; should equal 1 — also slower.

Best approach → bit manipulation trick (O(1))

7. Visual / Intuition Diagram

```
Example:
n = 8 → binary = 1000
n-1 = 7 → binary = 0111
  1000
& 0111
------
  0000   → power of two ✔
Example:
n = 10 → 1010
n-1 = 9 → 1001
  1010
& 1001
------
  1000 ≠ 0 → NOT a power of two ❌
```

8. Pseudocode

```
if n <= 0: return false
if (n & (n - 1)) == 0:
    return true
else:
    return false
```

9. Code Implementation

✅ **Python**

```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        return n > 0 and (n & (n - 1)) == 0
```

✅ **Java**

```
class Solution {
    public boolean isPowerOfTwo(int n) {
        return n > 0 && (n & (n - 1)) == 0;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(1)
Space	O(1)

11. Common Mistakes / Edge Cases

❌ Forgetting to check n > 0

❌ Returning true for 0

❌ Allowing negative powers (invalid)

❌ Using floating power or modulo methods (bad accuracy)

Edge cases:

- n = 1 → true
- n = 0 → false
- n < 0 → false

12. Detailed Dry Run (Step-by-Step)

```
Let's dry run for: n = 16
n = 16 → binary: 1 0000
n-1 = 15 → binary: 0 1111

n & (n-1):

  10000
& 01111
-------
  00000   → equals 0 → return True
Dry run for n = 12:
n = 12  → 1100
n-1 =11 → 1011

1100
1011
----
1000  → not 0 → return False
```

13. Common Use Cases

- Checking if size is power of two — memory alignment
- Bit-mask operations
- Determining valid binary tree sizes
- Efficient modulo operations in hashmaps

14. Common Traps

- Using while loop division when unnecessary
- Using floating math (Math.log) → precision issues
- Not handling negative numbers properly

15. Builds To (Related Problems)

- **LC 342** — Power of Four
- **LC 326** — Power of Three
- **LC 231 variant** — check for power of two in arrays
- **Bitwise tricks**:
  - count set bits
  - isPowerOfFour → (n & (n-1)) == 0 AND (n & 0x55555555) != 0

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Bit Trick	O(1)	O(1)	✔ Best
Repeated Division	O(log n)	O(1)	Simple but slower
Bit Counting	O(log n)	O(1)	Needs extra logic
Logarithm	O(1)	O(1)	❌ Floating point error risk

17. Why This Solution Works (Short Intuition)

Powers of two always have exactly one set bit in binary.
The trick n & (n-1) removes the lowest set bit.
If result becomes zero, the number had only one set bit.

18. Variations / Follow-Up Questions

- How to check power of four?
- How to check power of three? (use division, no bit trick)
- How to find next power of two greater than a number?
- Use bit hacks: round numbers to nearest power of two.