

## Problem Statement

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

An integer  $n$  is a power of two if there exists an integer  $x$  such that  $n == 2^x$ .

## ## Examples

- **Example 1:**

- Input:  $n = 1$
- Output: true
- Explanation:  $2^0 = 1$

- **Example 2:**

- Input:  $n = 16$
- Output: true
- Explanation:  $2^4 = 16$

- **Example 3:**

- Input:  $n = 3$
- Output: false

## ## Constraints

-  $-2^{31} \leq n \leq 2^{31} - 1$

## ## Solution

### ### Method 1: Mathematical Approach

An integer  $n$  is a power of two if it can be expressed as  $2^x$  for some integer  $x$ .

```
```python
import math

def is_power_of_two(n: int) -> bool:
    if n <= 0:
        return False
    log_value = math.log2(n)
    return log_value.is_integer()

# Example usage
print(is_power_of_two(1)) # True
print(is_power_of_two(16)) # True
print(is_power_of_two(3)) # False
```
```

## Method 2: Bit Manipulation

A number is a power of two if there is only one bit set in its binary representation.

```
```python
def is_power_of_two(n: int) -> bool:
    return n > 0 and (n & (n - 1)) == 0
```

# Example usage

```
print(is_power_of_two(1)) # True
print(is_power_of_two(16)) # True
print(is_power_of_two(3)) # False
```
```

### Explanation of the Bit Manipulation Method

1. **\*\*Condition  $n > 0$ \*\***: Ensures that negative numbers and zero are not considered powers of two.
2. **\*\*Condition  $(n \& (n - 1)) == 0$ \*\***: Checks that there is only one bit set in  $n$ .

You can choose either method based on your preference!