Power Of Two LeetCode

To determine whether an integer is a power of 2. Given an integer n, the task is to check whether it is a power of 2 or not.

For example, 1, 2, 4, 8, 16, etc. are all powers of 2, while 3, 6, 10, etc. are not.

Approach 1: Using For loop till 2^30 to get the result.

The function isPowerOfTwoLoop takes an integer n as input and returns true if n is a power of 2, and false otherwise. Here's how the function works:

Check if n is less than or equal to 0. If it is, return false, because 0 and negative numbers are not powers of 2.

Initialize a variable ans to 1. This variable will be used to store the potential powers of 2 that we will check against n.

Loop from 0 to 30 (inclusive). We only need to loop up to 30, because INT_MAX (the maximum value that an integer can store) is 2^31 - 1, and 2^31 is not a power of 2.

Check if ans is equal to n. If it is, we have found that n is a power of 2, so return true.

Check if ans is less than INT_MAX/2. If it is, multiply ans by 2. This will produce the next power of 2, which we will check against n in the next iteration of the loop.

If we have looped through all 31 possible powers of 2 and haven't found a match for n, return false.

Here's an example of how the function works for n = 8:

n is not less than or equal to 0, so we proceed to step 2.

ans is initialized to 1.

We loop from 0 to 30. In the first iteration, ans is 1, which is not equal to n = 8. In the second iteration, ans is 2, which is also not equal to n. In the third iteration, ans is 4, which is still not equal to n. In the fourth iteration, ans is 8, which is equal to n. We have found a match, so we return true.

Time And Space Complexity:

The time complexity of this function is O(1), because we are looping a fixed number of times (31) and performing constant-time operations within the loop.

The space complexity of this function is also O(1), because we are only using a constant amount of memory to store the n, ans, and i variables.

Approach 2: Using Bitwise operators (Bit Manipulation)

The function isPowerOfTwoBitwise uses a bitwise operation to determine if an integer is a power of 2. Here's how it works:

Check if n is greater than 0. If it is not, return false, because 0 and negative numbers are not powers of 2.

Use the bitwise AND operator (&) to check if n and n-1 have any common set bits (bits that are set to 1 in both numbers). If they don't have any common set bits, then n must be a power of 2. This is because powers of 2 have only one bit set to 1, and subtracting 1 from a power of 2 flips that bit to 0 and sets all lower-order bits to 1.

Return true if the condition in step 2 is true, and false otherwise.

Here's an example of how the function works for n = 8:

n is greater than 0, so we proceed to step 2.

We check if n & (n - 1) == 0. This is equivalent to 8 & 7 == 0. In binary, 8 is 1000 and 7 is 0111. When we perform a bitwise AND operation on these two numbers, we get 0000, which is 0. Because the result is 0, n is a power of 2.

We return true.

Space And Time Complexity:

The time complexity of this function is O(1), because we are only performing a few bitwise operations, which take a constant amount of time.

The space complexity of this function is also O(1), because we are only using a constant amount of memory to store the n variable.