Daily Coding Problem #268

Problem

This problem was asked by Indeed.

Given a 32-bit positive integer N, determine whether it is a power of four in faster than O(log N) time.

Solution

There are a number of ways to solve this.

One method would be to search for increasingly higher powers of four until we find one that is at least as large as our input. Then, we can check if this integer is equal to our input, in which case it will be a power of four. This is equivalent to finding the logarithm in base four, so the time complexity will be O(log N).

```
def power_of_four(num):
    power = 0
    while 4 ** power < num:
        power += 1

    if 4 ** power == num:
        return True
    else:
        return False
```

To make this more efficient, we can use bitwise operations. Note that any power of four (or two) will have only one bit set. We can check that this is the case by performing an `and` operation between the input and itself minus one.

Now let's look at the binary representations of powers of two:

```
2:  10
4:  100
8:  1000
16: 10000
...
```

A pattern we will notice is that powers of four will always have an even number of zeros after the set bit. We can find out this information by bit shifting the number down by one until we have removed all the trailing zeros, and incrementing a counter each time.

```
def power_of_four(num):
    if num & (num - 1):
        return False

    count = 0
    while num > 1:
        num >>= 1
        count += 1

    if count % 2 == 0:
        return True
    else:
        return False
```

Unfortunately this solution may still run in logarithmic time, as there may be `O(log N)` zeros in the binary representation of our integer.

With a bit of math, we can remove the offending portion of the last solution. Instead of counting the number of trailing zeros, note that any power of two will have `2` as a remainder when dividing by three, and any power of four will have `1` instead. Therefore our solution will look like the following:

```python
def power_of_four(num):
    return num & (num - 1) == 0 and num % 3 == 1
```