Daily Coding Problem #61

Problem

This problem was asked by Google.

Implement integer exponentiation. That is, implement the `pow(x, y)` function, where `x` and `y` are integers and returns `x^y`.

Do this faster than the naive method of repeated multiplication.

For example, `pow(2, 10)` should return 1024.

Solution

Implementing exponention naively is quite straightforward. We can either do it iteratively:

```
def power(x, y):
    if y < 0:
        base = 1 / x
        exponent = -y
    else:
        base = x
        exponent = y
    result = 1
    for _ in range(exponent):
        result *= base
    return result
```

or recursively:

```
def power(x, y):
    if y < 0:
        return power(1 / x, -y)
    elif y == 0:
        return 1
    else:
        return x * power(x, y - 1)
```

Just remember to deal with negative exponents!

However, we need to do faster than just naive multiplication. How can we do this?

Notice that the main bottleneck in performance here is doing multiplications y times. Since the process of multiplication takes about the same amount of time regardless of the actual sizes of the numbers, we should look at trying to move some of the work from the exponent to the base.

We can rewrite x^y as the following.

- If y is even, then `x^y = (x^2) ^ (y/2)`
- If y is odd, then `x^y = x * ((x^2) ^ ((y - 1) / 2))`

Now, by squaring the base, we have half as many multiplications to do! Let's go through an example. Say we want to compute `2^20`. We can then do it like this:

- `2^20 = 4^10 = 16^5 = 16 * (256)^2 = 16 * 256 * 256`

We've reduced the number of multiplications we need to do from 20 to 4. Let's code it up.

Again, we can do this iteratively:

```
def power(x, y):
    base = x
    exponent = y
    if y < 0:
        base = 1 / x
        exponent = -y
    coeff = 1
    while y > 1:
        if y % 2 == 0:
            base *= base
            y = y // 2
        else:
            coeff *= base
            base *= base
            y = (y - 1) // 2
    return coeff * base
```

Or recursively, although it takes up more space on the call stack:

```python
def power(x, y):
    if y < 0:
        return power(1 / x, -y)
    elif y == 0:
        return 1
    elif y == 1:
        return x
    elif y % 2 == 0:
        return power(x * x, y // 2)
    else: # y is odd
        return x * power(x * x, y // 2)
```

Since we're nearly halving the number of multiplications we need to do at each step, this will run in O(log y) time.