Daily Coding Problem #310

Problem

This problem was asked by Pivotal.

Write an algorithm that finds the total number of set bits in all integers between `1` and `N`.

Solution

A naive solution would be to create a function that counts the set bits for a given integer, and then apply this to each number between `1` and `N` in order.

There are a few ways we could implement this function. One method is as follows: we initially set our count to zero. Then, we repeatedly look at the last bit of our number, increment our count if it is one, and right shift. Once we have bit-shifted the number down to zero, we will have the correct count.

```
def count_set_bits(num):
    count = 0

    while num > 0:
        if num & 1 == 1:
            count += 1
        num >>= 1
```

```
    return count
```

A perhaps more Pythonic implementation is presented below:

```
def count_set_bits(num):
    return bin(num).count('1')
```

In any case, the time complexity of this operation will be `O(log N)`, since we must examine each bit of the number. If we then apply this to each successive integer, adding the number of set bits to an accumulated total, we will have an `O(N log N)` solution.

```
def total_set_bits(n):
    total = 0

    for i in range(1, n + 1):
        total += count_set_bits(i)

    return total
```

To see if we can do better, let us examine the binary representations of the first few integers.

```
N  | binary
------------
0  | 000
1  | 001
2  | 010
3  | 011
4  | 100
5  | 101
```

If we divide these numbers into even and odd sets, we end up with two groups, as follows:

```
Even | Odd
------------
000  | 001
010  | 011
100  | 101
```

Note that the two columns only differ in their last column, with even numbers having zeroes and odd numbers having ones. If there are an equal amount of even and odd numbers, which will happen if `N` is odd, this column will contribute a number of set bits equivalent to `(N + 1) // 2`. For example, here `N == 5`, so there are three numbers in each group, and each of the odd integers has a one in the last column.

Furthermore, if we cut off this last column, the prefixes are the same in each group, and represent exactly the numbers between `0` and `N // 2`.

Therefore, for an odd number `N`, we can form a recursive relationship for the number of set bits up to `N`:

```
f(N) = (N + 1) / 2 + 2 * f(N / 2)
```

Finally, in the case of an even number, we can simply count the set bits for that number, add it to our total, and solve for `N - 1`.

```
def total_set_bits(n):
    if n == 0:
        return 0
    elif n % 2 == 1:
        return (n + 1) // 2 + 2 * total_set_bits(n // 2)
    else:
        return count_set_bits(n) + total_set_bits(n - 1)
```

In the worst case, each time we divide by two, we may end up with an odd number, so we will have to perform `count_set_bits` again. Since each of these operation is `O(log N)`, we arrive at an overall time complexity of $O((\log N)^2)$.

Terms of Service
Press