

Then we can immediately write down the following equations:

$$\text{mults}(n) = 0 \quad \text{if } n = 0 \quad (2.5a)$$

$$\text{mults}(n) = 1 + \text{mults}(n - 1) \quad \text{if } n > 1 \quad (2.5b)$$

This is a pair of *recurrence equations*. We shall skip the mathematics and simply state the solution:

$$\text{mults}(n) = n \quad (2.6)$$

The recursive simple power algorithm performs exactly the same number of multiplications as the non-recursive simple power algorithm.

Example 2.5 suggests guidelines for ensuring that a recursive algorithm terminates:

- The problem must have one or more ‘easy’ cases and one or more ‘hard’ cases.
- In an ‘easy’ case, the algorithm must give a direct answer without calling itself.
- In a ‘hard’ case, the algorithm may call itself, but only to deal with an ‘easier’ case of the same problem.

In Example 2.5, the problem had one easy case, $n = 0$, and one hard case, $n > 0$. In the hard case, the algorithm called itself to deal with an easier case, $n-1$, and used that to compute its answer.

To compute b^n (where n is a nonnegative integer):

1. If $n = 0$:
 - 1.1. Terminate with answer 1.
2. If $n > 0$:
 - 2.1. Terminate with answer $b \times b^{n-1}$.

Algorithm 2.12 Recursive simple power algorithm.

```

static int power (int b, int n) {
  // Return the value of b raised to the n'th power
  // (where n is a nonnegative integer).
  if (n == 0)
    return 1;
  else
    return b * power(b, n-1);
}

```

Program 2.13 Java implementation of the recursive simple power algorithm.