

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

1. Set  $p$  to 1, set  $q$  to  $b$ , and set  $m$  to  $n$ .
2. While  $m$  is positive, repeat:
  - 2.1. If  $m$  is odd, multiply  $p$  by  $q$ .
  - 2.2. Halve  $m$  (neglecting any remainder), and multiply  $q$  by itself.
3. Terminate with answer  $p$ .

**Algorithm 2.5** Smart 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).
  int p = 1, q = b, m = n;
  while (m > 0) {
    if (m%2 != 0) p *= q;
    m /= 2; q *= q;
  }
  return p;
}

```

**Program 2.6** Java implementation of the smart power algorithm.

operations performed by it. In general, the number of characteristic operations depends on the algorithm's input data.

For some algorithms (like the simple power algorithm in Example 2.2) the analysis is straightforward. For other algorithms (like the smart power algorithm in Example 2.2) the analysis is more complicated. We sometimes have to make simplifying assumptions, and we sometimes have to be content with estimating the maximum (or average) number of characteristic operations rather than the exact number.

The simple power algorithm takes  $n$  multiplications, while the smart power algorithm takes at most  $2 \lfloor \log_2 n \rfloor + 2$  multiplications. If we double  $n$ , the simple power algorithm takes twice as many multiplications, while the smart power algorithm takes at most two extra multiplications. Now this is the heart of the matter. When we compare the efficiencies of alternative algorithms, what is most illuminating is the comparison of *growth rates*. The function  $2 \lfloor \log_2 n \rfloor + 2$  grows much more slowly than  $n$ . The fundamental reason for this is that  $\log_2 n$  grows much more slowly than  $n$ , as shown in Figure 2.8.

Of course we are interested in the actual times taken by alternative algorithms, but we are especially interested in the rates at which their time requirements grow with  $n$ . This interest in growth rates is easily justified. When  $n$  is small, we do not really care which algorithm is fastest, because none of the algorithms will take much time. But when  $n$  is large, we certainly do care, because all of the algorithms will take more time, and some