sequence of **bytes**, a collection of eight bits. A byte can therefore represent an unsigned number ranging from **0** to **255**. If we take one of the bits and make it the sign bit, we can make the same byte to represent numbers ranging from −**128** to **127**. But a computer cannot represent all possible real numbers. The fault is not with the binary number system; rather having only a **fnite** number of bytes is the problem.

While a gigabyte of memory may seem to be a lot, it takes an infnite number of bits to represent **π**. Since we want to store many numbers in a computer's memory, we are restricted to those that have a **fnite** binary representation. Large integers can be represented by an ordered sequence of bytes. Common lengths, usually expressed in terms of the number of bits, are 16, 32, and 64. Thus, an unsigned 32-bit number can

represent integers ranging between 0 and 232 − 1 (4,294,967,295), a number almost big enough to enumerate every human in the world!6

For both 32-bit and 64-bit integer representations, what are the largest numbers that can be represented if a sign bit must also be included.

**Exercise 5.2.1**

While this system represents integers well, how about numbers having nonzero digits to the right of the decimal point? In other words, how are numbers that have fractional parts represented? For such numbers, the binary representation system is used, but with a little more complexity. The **foating-point** system uses a number of bytes -typically 4 or 8 -to represent the number, but with one byte (sometimes two bytes) reserved to represent the **exponent** e of a power-of-two multiplier for the number -the mantissa m -expressed by the remaining bytes.



The mantissa is usually taken to be a binary fraction having a magnitude in the range



, which means that the binary representation is such that d−1 =1. 2 The number zero is an exception to this rule, and it is the **only** foating point number having a zero fraction. The sign of the mantissa represents the sign of the number and the exponent can be a signed integer.

A computer's representation of integers is either perfect or only approximate, the latter situation occurring when the integer exceeds the range of numbers that a limited set of bytes can represent. Floating point representations have similar representation problems: **if** the number ***x*** can be multiplied/divided by enough powers of two to yield a fraction lying between 1/2 and 1 that has a **fnite** binary- fraction representation, the number is represented exactly in foating point. Otherwise, we can only represent the number approximately, not catastrophically in error as with integers. For example, the number 2.5 equals **0.625 × 22**, the fractional part of which has an exact binary representation. 3 However, the number **2.6** does **not** have an exact binary representation, and only be represented approximately in foating point.

1. In some computers, this normalization is taken to an extreme: the leading binary digit is not explicitly expressed, providing an extra bit to represent the mantissa a little more accurately. This convention is known as the hidden-ones notation.
2. See if you can nd this representation.