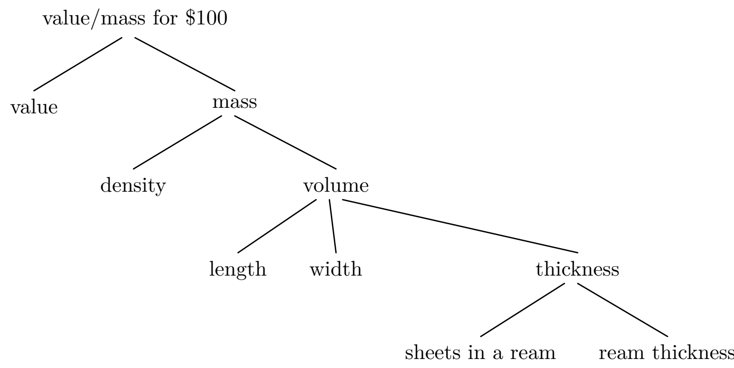
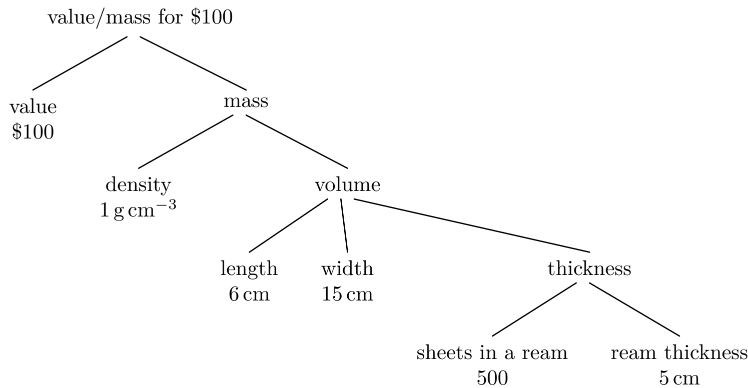


The ream (500 sheets) is roughly 5 cm thick. The only missing leaf value is the density of a bill. To find the density, use what you know: Money is paper. Paper is wood or fabric, except for many complex processing stages whose analysis is far beyond the scope of this book. When a process, here papermaking, looks formidable, forget about it and hope that you'll be okay anyway. More important is to get an estimate; correct the egregiously inaccurate assumptions later (if ever). How dense is wood? Wood barely floats, so its density is roughly that of water, which is $\rho \sim 1\,\mathrm{g\,cm^{-3}}$. So the density of a $100 bill is roughly $1\,\mathrm{g\,cm^{-3}}$.

Here is a tree including all the leaf values:



Now propagate the leaf values upward. The thickness of a bill is roughly 10^{-2} cm, so the volume of a bill is roughly

$$V \sim 6\,\mathrm{cm} \times 15\,\mathrm{cm} \times 10^{-2}\,\mathrm{cm} \sim 1\,\mathrm{cm}^3.$$

So the mass is

$$m \sim 1\,\mathrm{cm}^3 \times 1\,\mathrm{g\,cm^{-3}} \sim 1\,\mathrm{g}.$$

How simple! Therefore the value per mass of a $100 bill is $100/g. To choose between the bills and gold, compare that value to the value per mass of gold. Unfortunately our figure for gold is in dollars per ounce rather than per gram. Fortunately one ounce is roughly 27 g so $800/oz is roughly $30/g. Moral: Take the $100 bills but leave the $20 bills.

2.5 Random walks

The estimates in **Section 2.1** and **Section 2.3** are surprisingly accurate. The true pit spacing in a CDROM varies from 1 μm to 3 μm, according to the so-called *Red Book* where Philips and Sony give the specification of the CDROM; our estimate of 1 μm is not too bad. The true value for the oil imports is only 10% different from our estimate.

Equally important, the estimates are more accurate after doing divide-and-conquer reasoning. My 95% probability interval for oil imports, if I had to guess a value without subdividing the problem, is say from 10^6 b/yr to 10^{12} b/yr. In other words, if someone had claimed that the value is 10 million barrels per year, it would have seemed low, but I wouldn't have bet too much against it. After doing the divide-and-conquer estimate, I'd have been surprised if the true answer were more than a factor of 10 smaller or larger than the estimate.

This section presents a model for guessing in order to explain how divide-and-conquer reasoning can make estimates more accurate. The idea is that when we guess a value far outside our intuitive experience – for example, micron-sized distances or gigabarrels – the error in the exponent will be proportional to the exponent. For example, when guessing a quantity like 10^9 in one gulp, I really mean: 'It could be, say, 10^6 on the low side or, say, 10^{12} on the high side.' And when guessing a quantity like 10^{30} (the mass of the sun in kilograms), I would like to hedge my bets with a region like 10^{20} to 10^{40}. So, in this model any quantity 10^β is really shorthand for

$$10^\beta \rightarrow 10^{\beta-\beta/3} \ldots 10^{\beta+\beta/3}.$$

Now further simplify the model: Replace the range of values by its endpoints. So, if we try to guess a quantity whose true value is 10^β, we are equally likely to guess $10^{2\beta/3}$ or $10^{4\beta/3}$. A more realistic model would include 10^β as a likely possibility, but the simplest model is easy to simulate and to reason with (that justification is a fancy way to say that I am lazy).

To see the consequences of the model, I'll compare subdividing and not subdividing by using a numerical example. Suppose that we want to guess a quantity whose true value is 10^{12}. Without subdividing, we might guess 10^8 or 10^{16} (adding or subtracting one-third of the exponent), a wide range.

Compare that range to the range when we subdivide the estimate into 16 equal factors. Each factor is $10^{12/16} = 10^{3/4}$. When guessing each factor, the model says that we would guess $10^{1/2}$ or 10^1 each with $p = 0.5$. Here is an example of choosing 16 such factors randomly from $10^{1/2}$ and 10^1 and multiplying them:

$$10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{0.5} = 10^{10.5}$$

Here are three other randomly generated examples:

$$10^{1} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} = 10^{13.0}$$

$$10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} = 10^{11.5}$$

$$10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{1} \cdot 10^{1} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{0.5} \cdot 10^{1} \cdot 10^{0.5} = 10^{10.5}$$

These estimates are mostly within one factor of 10 from the true answer of 10^{12}, whereas the one-shot estimate might be off by four factors of 10. What has happened is that the

errors in the individual pieces are unlikely to point in the same direction. Some pieces will be underestimates, some will be overestimates, and the product of all the pieces is likely to be close to the true value.

This numerical example is our first experience with the random walk. Their crucial feature is that the expected wanderings are significantly smaller than if one walks in a straight line without switching back and forth. How much smaller is a question that we will answer in **Chapter 8** when we introduce special-cases reasoning.

2.6 The Unix philosophy

Organizing complexity by breaking it into manageable parts is not limited to numerical estimation; it is a general design principle. It pervades the Unix and its offspring operating systems such as GNU/Linux and FreeBSD. This section discusses a few examples.

2.6.1 Building blocks and pipelines

Here are a few of Unix's building-blocks programs:

- `head`: prints the first *n* lines from the input; for example, `head -15` prints the first 15 lines.
- `tail`: prints the last *n* lines from the input; for example, `tail -15` prints the last 15 lines.

How can you use these building blocks to print the 23rd line of a file? Divide and conquer! One solution is to break the problem into two parts: printing the first 23 lines and, from those lines, printing the last line. The first subproblem is solved with `head -23`. The second subproblem is solved with `tail -1`.

To combine solutions, Unix provides the pipe operator. Denoted by the vertical bar |, it connects the output of one program to the input of another command. In the numerical estimation problems, we combined the solutions to the subproblems by using multiplication. The pipe operator is analogous to multiplication. Both multiplication in numerical estimation, and pipes in programming, are examples of composition operators, which are essential to a divide-and-conquer solution.

To print the 23rd line, use this combination:

```
head -23 | tail -1
```

To tell the system where to get the input, there are alternatives:

1. Use the preceding combination as is. Then the input comes from the keyboard, and the combination will read 23 typed lines, print out the final line from those 23 lines, and then will exit.
2. Tell `head` to get its input from a file. An example file is the dictionary. On my GNU/Linux laptop it is the file `/usr/share/dict/words`, with one word per line. To print the 23rd line (i.e. the 23rd word):