Relations and Functions 3

Today:

- ☐ The size of infinite sets
- ☐ Practical and impractical algorithms
- ☐ Asymptotics: O, Ω, and Θ

1 The Size of Sets

A beautiful definition describes when we would like to say that $|A| = |B|$ or $|A| \le |B|$ even for infinite A, B.

Definition. We say that $|A| = |B|$ (the sets are *equinumerous*) if there exists a bijection $\pi\colon A \to B$. We say that $|A| \le |B|$ if there exists an injective function $f\colon A \to B$. We say that $|A| < |B|$ if $|A| \le |B|$ but $|A| \ne |B|$ (meaning $\neg(|A| = |B|)$).

We earlier spoke of a set being countably infinite: a set A is *countably infinite* if there exists a permutation $\pi\colon \mathbb{N} \to A$. In other words, A is countably infinite if $|\mathbb{N}| = |A|$. A set is *countable* if it is finite or countably infinite. A set is *uncountable* if it is not countable.

Note that we haven't actually given a meaning to $|A|$, we only gave a meaning to predicates $\text{eqCard}(A, B)$ and $\text{leCard}(A, B)$ and $\text{ltCard}(A, B)$, which went by notational idioms $|A| = |B|$ and $|A| \le |B|$ and $|A| < |B|$.

Proposition. eqCard is an equivalence relation, while leCard and ltCard are reflexive and transitive.

Well, we had better not ask what these are relations *on*, as that sounds problematic.

Propositions. $\mathbb{Z} \times \mathbb{Z}$ is countable. So is $\mathbb{Q}$. So is Σ^*. So is the set of all valid programs over some specified programming language.

Theorem. [Cantor-Schröder-Bernstein, circa 1896.] If $|A| \le |B|$ and $|B| \le |A|$ then $|A| = |B|$.

We won't prove this, but I encourage you to read a proof of it. It's a very cool theorem!

The theorem above is that makes the $|\cdot|$ "work" and seem familiar.

Uncountable sets. A fascinating theorem establishes that some infinites are bigger than others.

Theorem. There are uncountable sets. In particular, the set of all languages over the alphabet $\Sigma = \{0, 1\}$ is uncountable.

Proof: Suppose for contradiction that the languages over Σ were countable. Let $L_1, L_2, \ldots$ be an enumeration of them. Let $x_1, x_2, \ldots,$ be an enumeration of the strings in Σ^*. Define a new language $D \subseteq \{\mathtt{0}, \mathtt{1}\}^*$ by asserting that

$$D = \{x_i \colon x_i \notin L_i\}.$$

I claim that $D \notin \{L_1, L_2, \ldots\}$. After all, if $D = L_k$ for some particular k, then is x_k in D or is it not? By definition, $x_k \in D$ if and only if $x_k \notin L_k = D$, a contradiction. $\diamondsuit$

There are several other sets equinumerous to (0) the set of all languages over $\{0, 1\}$. These would include: (1) the set of all functions from $\mathbb{N}$ to $\{0, 1\}$; (2) the set of all functions from $\mathbb{N}$ to $\mathbb{N}$; (3) the set of all real numbers; (4) reals on the interval $[0, 1]$; and (5) $\mathcal{P}(\mathbb{N})$. One can prove all of the uncountable as with our former proof, and you can also find bijections between them.

How about proving that two of these sets are equinumerous? Let's say (0) and (5), for example, as an easy pair.

One sometimes refers to the cardinality (size) of the sets above as c, "the cardinality of the continuum." This contrasts with the cardinality of the natural numbers, which is denoted $\aleph_0$.

Example (5) is an interesting example because it points to a way of making sets with ever bigger cardinality. Just go: $\mathbb{N}$; $\mathcal{P}(\mathbb{N})$; $\mathcal{P}(\mathcal{P}(\mathbb{N}))$; and so on. We won't prove it, but we will always have that $|X| < |\mathcal{P}(X)|$.

There is a fascinating immediate consequence to the uncountability of (0) coupled with programs being countable:

Corollary. Some languages can't be decided by any computer program. In fact, *most* languages can't be decided by any computer program.

The continuum hypothesis. A natural question is whether or not there exists a set A with cardinality that is strictly between that of the naturals and that of the reals: a set A such that $|\mathbb{N}| < |A| < |\mathbb{R}|$. There is no loss of generality to assume $A \subseteq \mathbb{R}$.

The *the continuum hypothesis* (CH) is the presumed *in*existence of such a set. The conjecture was advanced by Georg Cantor in 1878. But he could neither prove nor disprove it. In 1963, building on earlier work of Kurt Gödel, Paul Cohen proved that there is, in some sense, no answer to this question. He showed that the CH is *independent* of the axioms of ZFC.

To *pluralists*, like Paul Cohen, showing CH to be independent of ZFC settled the question: it shows that it has no answer, and so you are *done*. You could make CH an axiom; you could make $\neg$CH an axiom; and you could, if you wanted, develop further either system of axioms. But to *non-pluralists*, like Kurt Gödel, the result meant that ZFC wasn't strong enough of a foundation for set theory: we needed better foundations.

2 Practical and Impractical Algorithms

We have sometimes sketched how to do things—given algorithms—like how to figure out if a boolean formula is satisfiable, or is a tautology. Some of our suggested mechanisms have not been practical. Where does practicality end and impracticality begin?

We could take a *concrete* approach. Nowadays a digital computer can do a handful of simple computations each nanosecond, which is about 10^{-9} seconds. So a single computer can do a little more than 2^{30} basic operations a second. So if a task you want to perform is going to take less than $\sim 2^{30}$ operations, it's probably easy to perform.

If your task takes more like 2^{40} basic operations, it should still be practical, but you might need to be a bit more careful or patient. Or you might need to throw quite a few computers at the problem.

Around 2^{50}–2^{60} basic operations and you're looking at a major effort. You might need to develop specialized hardware. You might need millions of dollars. But you still have a good chance to get the task solved.

By 2^{80} basic operations the task is going to get infeasible. It's not going to matter how many people or dollars you throw at the problem, you're not going to be able to get there. Our time is limited, and the earth's resources are limited.

By 2^{100} basic operations the task is totally infeasible. Now and for decades to come.

Of course if you can change the computation in some fundamental way, then all bets are off. If you can find a way that turns your 2^{100}-time computation into a 2^{50}-time operation, all has changed.

Cryptographic shared keys are usually 128–256 bits, so that "brute force" attack will take 2^{128}–2^{256} time. The latter, and perhaps the former, is not only impractical now, but will remain so forever. The universe is less than 2^{60} seconds old, and the earth has fewer than 2^{170} atoms.

While it's good to have these sort of waypoints in mind, it's more useful to develop a theory that that doesn't care about technology or constant. The traditional approach is to look at growth rates. One assumes there to be a parameter n that measure how big problems are.

Often there is a natural parameter n in the problem. If you're talking about a boolean formula, for example, you might speak of the number of variables, assuming grows in size roughly with this. Or you might speak of the length of the formula—the total number of characters to write it down.

Quite broadly, it has become tradition to differentiate polynomial and exponential growth rates; or polynomial and super-polynomial.

Poly: The set of all polynomials p with natural (say) coefficients.

Exp: In one way of defining this, the set of all functions of the form $c^{p(n)}$ where $c > 0$ is a constant and $p \in$ Poly.

One then equates growth rates that are bounded above by an element in Poly with things that are practical to do. And one equates growth rates that are bounded below by an element of Exp with things that are impractical to do.

Imperfect, but often useful, for a great deal seems to fall in one camp or the other. And the differences in behavior can be marked.

n	$n \lg n$	n^2	n^3	2^n
10	30 ns	100 ns	1 μs	1 μs
100	700 ns	10 μs	1 ms	10^{13} years
1000	10 μs	1 ms	1 sec	10^{284} years
10000	100 μs	0.1 sec	17 mins	10^{3000} years
10^5	2 ms	10 sec	1 day	—
10^6	20 ms	17 min	32 years	—
10^9	30 sec	31 years	10^{10}years	—

Figure 1: How long things take when 1 computational step takes 1 ns (=10^{-9} sec)

3 Asymptotics: O, Ω, and Θ

Motivate the notion, then introduce the following definition, starting with O (the most common), then Θ, then Ω (the least common).

$$\begin{aligned}
\Omega(g) &= \{f\colon \mathbb{N} \to \mathbb{R}\colon \exists c, N_0 \text{ s.t. } c \cdot g(n) \le f(n) \text{ for all } n \ge N_0\} \\
O(g) &= \{f\colon \mathbb{N} \to \mathbb{R}\colon \exists C, N_0 \text{ s.t. } f(n) \le C \cdot g(n) \text{ for all } n \ge N_0\} \\
\Theta(g) &= \{f\colon \mathbb{N} \to \mathbb{R}\colon \exists c, C, N_0 \text{ s.t. } c \cdot g(n) \le f(n) \le C \cdot g(n) \text{ for all } n \ge N_0\}
\end{aligned}$$

It's fine to replace OK to replace $\mathbb{N}$ by some arbitrary infinite subset of $\mathbb{R}^+$.

People often use "is" or "=" for is an element of." I myself don't like this.

Do a bunch of examples.

Simple propositions/manipulations:

- If $f \in O(n^2)$ and $g \in O(n^2)$ then $f + g \in O(n^2)$.
- If $f \in O(n^2)$ and $g \in O(n^3)$ then $f + g \in O(n^3)$.
- If $f \in O(n \lg n)$ and $g \in O(n)$ then $fg \in O(n^2 \lg n)$.

True/False: If $f \in \Theta(n^2)$ then $f \in O(n^2)$. True.

True/False: $n! \in O(n^n)$. True.

etc..

Arguments for:

- Simplicity — makes arithmetic simpler, makes analysis easier
- When applied to running times, routinely good enough, in practice, to get a feel for efficiency and to compare candidate solutions.
- When applied to running times, facilitates greater model-independence

Arguments for:

- Hidden constants **can** matter
- Might make you fail to see things you *should* see
- Not everything has an "n" value to grow

List of common/typical growth rates:

$\Theta(n!)$

$\Theta(2^n)$

$\Theta(n^3)$

$\Theta(n^2 \log n)$

$\Theta(n^2)$

$\Theta(n \log n \log \log n)$

$\Theta(nlgn)$

$\Theta(n)$

$\Theta(\sqrt{n})$.

$\Theta(\log n)$

$\Theta(1)$

The highest degree term in a polynomial is the term that determines the asymptotic growth rate of that polynomial.

General rule: characterize functions in simplest and tightest terms that you can.

In general we should use the big-O notation to characterize a function as closely as possible. For example, while it is true that $f(n) = 4n^3 + 3n^2$ is $O(n^5)$ or $O(n^4)$, it is "better" to say that $f(n)$ is $O(n^3)$.

It is correct yet inappropriate to include constant factors and lower order terms in the big-O notation. For example, it is poor usage to say that the function $2n^3$ is $O(4n^3+8n \log n)$, although it is technically correct. The "4" has no place in the expression, and the $8n \log n$ term doesn't belong there, either.

"Rules" of using big-O:

- If $f(n)$ is a polynomial of degree d, then $f(n)$ is $O(n^d)$. We can drop the lower order terms and constant factors.
- Use the smallest/closest possible class of functions, for example, "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$."
- Use the simplest expression of the class, for example, "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$".

Intertwine examples with the analysis of the resulting recurrence relation

1. How long will the following fragment of code take [nested loops, second loop a nontrivial function of the first] – something $O(n^2)$
2. How long will a computer program take, in the worst case, to run binary search, in the worst case? $T(n) = T(n/2) + 1$
3. $T(n) = 3T(n/2) + 1$. Solution (repeated substitution) $n^{\lg 3} = n^{1.5849...}$. What about $T(n) = 3T(n/2) + n$? Or $T(n) = 3T(n/2) + n^2$? [recursion tree]
4. How many gates do you need to multiply two n-bit numbers using grade-school multiplication?
5. How many comparisons to "selection sort" a list of n elements? $T(n) = 1+T(n-1)$.
6. How many comparisons to "merge sort" a list of n elements? $T(n) = T(n/2) + n$.
7. What's the running time of deciding SAT using the obvious algorithm? Careful.

Warning: Don't think that asymptotic notation is only for talking about the running time or work of algorithms; it is a convenient way of dealing with functions in many settings.