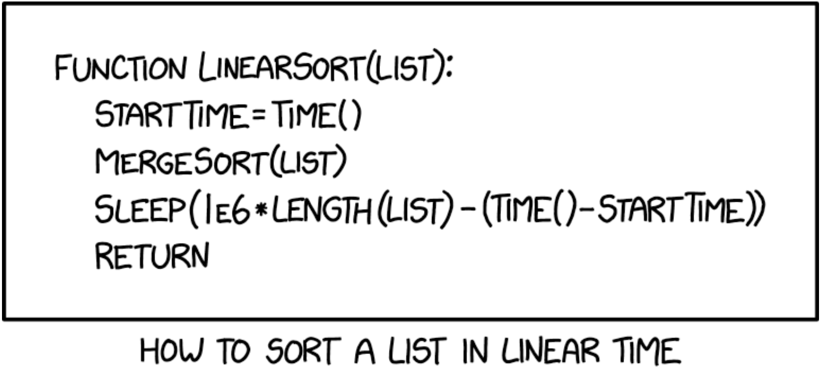
Intro to complexity and asymptotic analysis

Textbook: 7.1



Running time

- A deterministic **halting** TM has a max number of steps it will ever take
- This is usually dependent on the input, most often the length
- We customarily let n be the length of the input

Def: Let M be a deterministic TM that always halts. The **running time** or **time complexity** of M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximal number of steps M will take on input w.

- It's often easier to only look at very large inputs, where we can neglect lower-order terms. This is called **asymptotic analysis**

Big "O"

- If we wanted to say $f(n)$ grows at a rate *no faster than* $g(n)$, we could say $f(n)$ **is** $O(g(n))$
  - In some texts, $f(n) = O(g(n))$ or $f(n) \in O(g(n))$
  - Colloquially, $f(n)$ is order $g(n)$

Def: Big-O notation. Formally, if there exist positive real numbers c and n_0 such that for all $n > n_0$

$$f(n) \leq cg(n)$$

then we say $g(n)$ **is an asymptotic upper bound on** $f(n)$ and $f(n)$ **is** $O(g(n))$.

- Means f is less than or equal to g if we disregard differences up to a constant factor

Big "O"

- Weirdly, we don't need to specify a base when logarithms are involved!
  - Since $log_c(n)$ is always proportional to $log_d(n)$
  - We usually use $log(n)$, $ln(n)$, or $lg(n)$
- $f(n)$ being $O(1)$ means that $f(n)$ is bounded by some constant
- Big-O is the most common type of analysis, since it analyzes the "worst-case" scenario
  - It was popularized by Donald Knuth in the 70's

Little "o"

- Big-O gave us the option to say $f(n)$ is asymptotically *no more than* $g(n)$: What if we want to say $f(n)$ is *less than* $g(n)$? We use **little-o** notation!
- Little-o can be thought of as saying something grows insignificantly when compared to something else

Def: For functions f, g, we say f is $o(g(n))$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

In other words, as $n \to \infty$, $g(n)$ grows infinitely larger and faster than $f(n)$.

$$f(n) \texttt{ is } O(g(n)) \iff \exists c \exists n_0 \forall n [n_0 < n \to f(n) \leq cg(n)]$$
$$f(n) \texttt{ is } o(g(n)) \iff \forall c \exists n_0 \forall n [n_0 \leq n \to f(n) < cg(n)]$$

- The difference is subtle, but far-reaching!

Polynomiality and nonpolynomiality

- Many algorithms run in time EG $O(n)$, $O(n^2)$, $O(n^{100})$
- In real life, anything as bad as or worse than $O(n^2)$ is too costly: We don't even bother with EG $O(2^n)$
- However, $O(2^n)$ is very theoretically interesting!

Def: An algorithm with time complexity of the form $O(n^c)$ for constant c is said to be **polynomial.** If the complexity is not of this form, it is said to be **nonpolynomial.** This is called the **polynomiality/nonpolynomiality** of the algorithm.

Complexity classes and the TIME operator

- There are many problems which can be decided by a deterministic TM in $f(n)$ time
- We introduce the TIME operator to find the set of all languages decidable in some time limit

Def: For some function f, the **time complexity class** $\texttt{TIME}(t(n))$ is the set of all **languages** decidable in $O(t(n))$ time by a **deterministic** Turing Machine.

Note: Computationally equivalent models solving the same problem **may have different time complexities!** Namely, dTMs and nTMs are computationally equivalent but have vastly different complexity classes.

Nondeterministic Running Time

- Running time makes sense on dTMs, but what about nTMs?
- Some branches of the nondeterminism may live longer than others
- We'll just use **the longest running time of any branch**

Def: Let N be a nondeterministic decider Turing machine on input of length n. The **running time** $f(n)$ of N is the maximum number of steps that N uses on any branch of its computation.

- We previously gave an algorithm for simulating an nTM with a dTM
- That algorithm ran in $O(2^n)$ deterministic timesteps for n nondeterministic timesteps
- Therefore, any algorithm running in $f(n)$ nondeterministic time runs in **at worst** $O(2^{f(n)})$.

The `NTIME` operator

- It is also interesting to study the set of all languages decidable in $f(n)$ **nondeterministic** time: We use the operator `NTIME`

Def: For some function f, the **time complexity class** $\texttt{NTIME}(f(n))$ is the set of all languages decidable by a **nondeterministic** TM in $O(f(n))$ time.

Note: All "reasonable" (textbook's word) deterministic models of computation are polynomial-time convertible to one another

Next up: P and NP