What’s an Algorithm?

• Instructions for doing a task
  • meal recipes
  • how-to’s
  • IKEA furniture assembly
  • programs

• Repeatable process

• Reproducible results
Algorithm Analysis

- Theoretical study of how algorithms perform
- Does how long it takes change when the input grows?
- What happens if we change underlying data structure?
- High-level view, unconcerned with mundane details
  - “Hardware? That costs money.”
  - “Operating System? Did you call IT?”
  - “Java? C++? Pseudocode is more than sufficient.”
What it is not

• Performance tuning/optimization concerning:
  • hardware
    • CPU vs GPU, Intel vs AMD, Nvidia vs AMD
    • HDD vs SDD
  • OS
    • Linux vs Windows vs OS X
  • language
    • C vs Java vs Fortran, etc.
  • compilers and settings
    • gcc vs MSVC
    • -O2 vs –O3
Nor is it

• Numerical analysis
  • Floating-point specific
  • Specific course, take if interested in scientific computing

• Empirical analysis/Benchmarking
  • Directly measuring performance
    • BASH `time`
    • `time` module
  • Have to worry about consistent runtime conditions
    • Same OS, hardware, no other programs running, etc.
  • (still very important activity, though!)
Bald-faced Lies

• “All simple operations take the same amount of time”
  • arithmetic: =, +, -, *, /, **
  • comparisons: <, >, <=, >=, !=, ==
  • element access: [ ]
  • function calls
  • function returns

• “We have infinite RAM.”

• “Cache? Never heard of it.”
Theoretical Running Time

The sum total of all the simple operations, taking into account any looping or recursion.

• False in practice
  • Floating point division is slower than addition
  • Memory reads affected by cache coherency
  • finite RAM

• Our claim is that the practical gets overwhelmed by the theoretical as input grows, so we can ignore the practical (for now).
Worst Case Scenario

• Best case scenarios are often trivial
• Average case can be hard to verify
  • statistics and probability required
• Worst case relative often relatively easy to derive
  • by default we’ll be talking about worst case most of the time, with some notable exceptions later
Growth Functions

• $f : n \to t$
  • function from $n$ to $t$
  • $n$ is the size of the input
    • number of users
    • number of file entries
  • $t$ is the time the algorithm takes to run on $n$
    • often in unspecified units
Growth Function Categories 1

• Constant, c
  • same t for every n

• Logarithmic, log n
  • can cut down n repeatedly
  • assume logarithm base 2 unless told otherwise

• Linear
  • as n grows, t grows

• Superlinear \((n \times \log(n))\)
  • can cut down n, but has to do it for each n
Growth Functions Categories 2

• Quadratic, \(n^{**2}\)
  • often due to nested loops

• Cubic, other Polynomial, \(n^{**3}, n^{**m}\)
  • often due to more nesting of loops, 3D grids

• Exponential, \(c^{**n}\)
  • compounding growth

• Factorial
  • \(n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1\)
  • if have to work with every permutation of \(n\) items
What We Want

• Polynomial or better
  • $n^3 >> n^2 >> n \log n >> n >> \log n >> c$
  • $>>$ means “much greater than” or “dominates”
    • $n^3$ dominates $n$, etc.

• World peace
What We Don’t Want

• Anything else
  • exponential, factorial
    • $c^n \ll n!$
  • even worse things we didn’t mention like Ackermann’s

• Caution:
  • c can be anything finite
  • $10^{18}$ is c, but HUGE, in a case like that, being constant doesn’t help

• Rabies
Growth Function Graphs

• (see link in web notes)
• note: the textbook likes to use log-log graphs
  • the slope of the line indicates growth
Asymptotic Analysis

Let $f, g$ map $\mathbb{Z}^+$ to $\mathbb{R}^+$

- $f(n)$ is $O(g(n))$ if $\exists c \in \mathbb{R}$ and $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
- $cg(n)$ is an upper bound for $f(n)$
- “$f$ is big oh of $g$”

(fig 3.5 from textbook on white board)

(If you’re unfamiliar with this math notation, it’s in closer to plain English in the notes and textbook.)
Asymptotic Analysis, 2

Sometimes we find a lower bound. Same setup as before but change it so that if \( f(n) \geq cg(n) \) for all \( n \geq n_0 \) then \( f(n) \) is \( \Omega(n) \).

• “f is big omega of g”

If f is both big O of g and big \( \Omega \) of g, then f is big \( \Theta \) of g

• “f is big theta of g”

• big \( \Theta \) is the strongest statement
Comparative Analysis

• We’re only concerned with the unmodified dominant term

• \( f(n) = 10n^3 + 5n \)
  • “\( f \) has cubic growth” or “\( f \) is big \( O \) \( n \) cubed”

• as \( n \) grows, the sub-dominant terms and the constant factor become insignificant

• Chart of comparative runtimes
Caveats

• In theory, ignore coefficients, in practice can be significant
  • A is linear, B is superlinear “Oh, A has better asymptotic growth, we should use A.”
  • A is $1000000000n$, B is $n \log n$ “Oh, wait, B is better until ~62K and we never break 20K…”

• Programmer time is part of the real cost!
  • Runs in 5 minutes but takes a 40 hours to program vs runs in 10 minutes but only takes 1 hour to program
  • [xkcd: Is It Worth the Time?]