1 Machines & Models

5 October 2023

Sebastian Wild

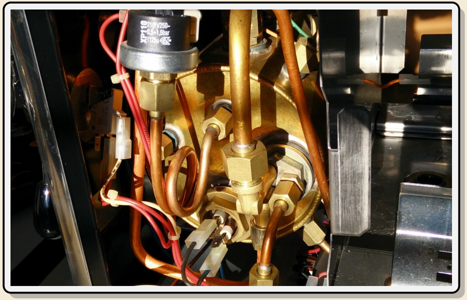
COMP526 (Fall 2023)
University of Liverpool

version `2023-09-30 16:40`

Learning Outcomes

1. Understand the difference between empirical *running time* and algorithm *analysis*.
2. Understand *worst / best / average case* models for input data.
3. Know the *RAM machine* model.
4. Know the definitions of *asymptotic notation* (Big-Oh classes and relatives).
5. Understand the reasons to make *asymptotic approximations*.
6. Be able to *analyze* simple *algorithms*.

Unit 1: ***Machines & Models***



Outline

What is an algorithm?

An algorithm is a sequence of instructions.

think: recipe

More precisely:

e. g. Python script

1. mechanically executable
   ⇝ no "common sense" needed
2. finite description   ≠ finite computation!
3. solves a *problem*, i. e., a class of problem instances

$x + y$, not only $17 + 4$

- input-processing-output abstraction

Typical example: *bubblesort*

⇝ not a specific program
but the underlying idea

What is a data structure?

A data structure is

1. a rule for encoding data (in computer memory), plus
2. algorithms to work with it (queries, updates, etc.)

typical example: binary search tree

1.1 Algorithm analysis

Good algorithms

Our goal: Find good (best?) algorithms and data structures for a task.

Good "usually" means

can be complicated in distributed systems

- fast running *time*
- moderate memory *space* usage

Algorithm analysis is a way to

- compare different algorithms,
- predict their performance in an application

Running time experiments

Why not simply run and time it?

- results only apply to
  - single *test* machine
  - tested inputs
  - tested implementation
  - ...
  - ≠ *universal truths*

- instead: consider and analyze algorithms on an abstract machine

  survives Pentium 4

  ⇝ provable statements for model

  ⇝ testable model hypotheses

⇝ Need precise model of machine (costs), input data and algorithms.

Data Models

Algorithm analysis typically uses one of the following simple data models:

- **worst-case performance:**
  consider the *worst* of all inputs as our cost metric
- **best-case performance:**
  consider the *best* of all inputs as our cost metric
- **average-case performance:**
  consider the average/expectation of a *random* input as our cost metric

Usually, we apply the above for *inputs of same size n*.

⇝ performance is only a **function of** n.

1.2 The RAM Model

Clicker Question

What is the cost of *adding* two d-digit integers?
(For example, for $d = 5$, what is $45\,235 + 91\,342$?)

- **A** constant time
- **B** logarithmic in d
- **C** proportional to d
- **D** quadratic in d
- **E** no idea what you are talking about

→ sli.do/comp526

Clicker Question

What is the cost of *adding* two d-digit integers?
(For example, for $d = 5$, what is $45\,235 + 91\,342$?)

- **A** constant time ✓ if numbers have ≤ 64 bits
- **B** ~~logarithmic in d~~
- **C** proportional to d ✓ if numbers are larger
- **D** ~~quadratic in d~~
- **E** no idea what you are talking about ✓

→ `sli.do/comp526`

Machine models

The machine model decides

- what algorithms are possible
- how they are described (= programming language)
- what an execution *costs*

Goal: Machine model should be
detailed and powerful enough to reflect actual machines,
abstract enough to unify architectures,
simple enough to analyze.

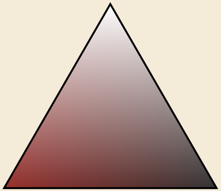
Machine models

The machine model decides

- what algorithms are possible
- how they are described (= programming language)
- what an execution *costs*

Goal: Machine model should be
detailed and powerful enough to reflect actual machines,
abstract enough to unify architectures,
simple enough to analyze.

$\rightsquigarrow$ usually some compromise is needed



Random Access Machines

Random access machine (RAM)

more detail in §2.2 of *Sequential and Parallel Algorithms and Data Structures* by Sanders, Mehlhorn, Dietzfelbinger, Dementiev

- unlimited *memory* $\texttt{MEM}[0], \texttt{MEM}[1], \texttt{MEM}[2], \ldots$
- fixed number of *registers* $R_1, \ldots, R_r$ (say $r = 100$)
- memory cells $\texttt{MEM}[i]$ and registers R_i store w-bit integers, i. e., numbers in $[0..2^w - 1]$
  w is the word width/size; typically $w \propto \lg n$ $\rightsquigarrow$ $2^w \approx n$
- Instructions:
  - load & store: $R_i := \texttt{MEM}[R_j]$ $\texttt{MEM}[R_j] := R_i$
  - operations on registers: $R_k := R_i + R_j$ (arithmetic is *modulo* 2^w!)
    also $R_i - R_j$, $R_i \cdot R_j$, R_i div R_j, R_i mod R_j
    C-style operations (bitwise and/or/xor, left/right shift)
  - conditional and unconditional jumps
- cost: number of executed instructions

we will see further models later

$\rightsquigarrow$ The RAM is the standard model for sequential computation.

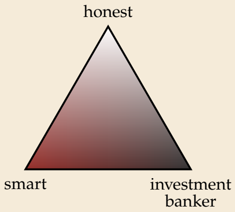
Pseudocode

- Programs for the random-access machine are very low level and detailed

≈ assembly/machine language

Typical simplifications when describing and analyzing algorithms:

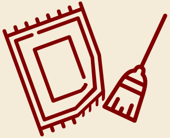
- more abstract *pseudocode* ← code that humans understand (easily)
  - control flow using **if**, **for**, **while**, etc.
  - variable names instead of fixed registers and memory cells
  - memory management (next slide)
- count *dominant operations* (e. g. memory accesses) instead of all RAM instructions



In both cases: We can go to full detail where needed.

Memory management & Pointers

- A random-access machine is a bit like a bare CPU … without any operating system
  ⇝ cumbersome to use
- All high-level programming languages add *memory management* to that:
  - Instruction to *allocate* a contiguous piece of memory of a given size (like `malloc`).
    - used to allocate a `new` array (of a fixed size) or
    - a `new` object/record (with a known list of instance variables)
    - There's a similar instruction to `free` allocated memory again.
  - A *pointer* is a memory address (i. e., the i of $\mathtt{MEM}[i]$).
  - Support for procedures (a.k.a. functions, methods) calls including recursive calls
    - (this internally requires maintaining call stack)



We will mostly ignore *how* all this works in COMP526.

1.3 Asymptotics & Big-Oh

Clicker Question

What is the correct way to complete the equation?

$8n + \frac{1}{2}n^2 + 1024 = \square$

A $O(1)$

B $O(n)$

C $O(n \log(n))$

D $O(n^2)$

E I don't know $O(\cdot)$

→ sli.do/comp526

Clicker Question

What is the correct way to complete the equation?

$8n + \frac{1}{2}n^2 + 1024 = \boxed{}$

- **A** ~~$O(1)$~~
- **B** ~~$O(n)$~~
- **C** ~~$O(n \log(n))$~~
- **D** $O(n^2)$ ✓
- **E** ~~I don't know $O(\cdot)$~~

→ sli.do/comp526

Why asymptotics?

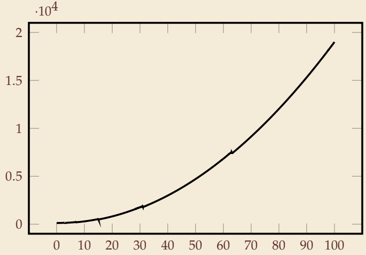
Algorithm analysis focuses on (the limiting behavior for infinitely) **large** inputs.

- abstracts from unnecessary detail
- simplifies analysis
- often necessary for sensible comparison

Asymptotics = approximation around ∞

Example: Consider a function $f(n)$ given by

$2n^2 - 3n\lfloor \log_2(n+1) \rfloor + 7n - 3\lfloor \log_2(n+1) \rfloor + 120$



Why asymptotics?

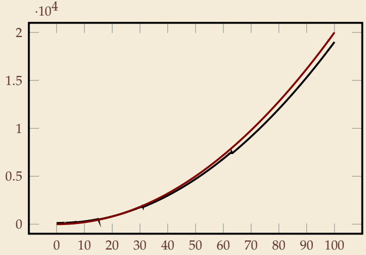
Algorithm analysis focuses on (the limiting behavior for infinitely) **large** inputs.

- abstracts from unnecessary detail
- simplifies analysis
- often necessary for sensible comparison

Asymptotics = approximation around ∞

Example: Consider a function $f(n)$ given by

$$2n^2 - 3n\lfloor \log_2(n+1) \rfloor + 7n - 3\lfloor \log_2(n+1) \rfloor + 120 \quad \sim \quad \mathbf{2n^2}$$



Asymptotic tools – Formal & definitive definition

- **"Tilde Notation":** $f(n) \sim g(n)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ (iff: if, and only if)

  „f and g are *asymptotically equivalent*"

Asymptotic tools – Formal & definitive definition

- **"Tilde Notation":** $f(n) \sim g(n)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ (iff: if, and only if)

  „f and g are *asymptotically equivalent*"

- **"Big-Oh Notation":** $f(n) \in O(g(n))$ iff $\left|\frac{f(n)}{g(n)}\right|$ is bounded for $n \geq n_0$ (also write '=' instead)

  iff $\lim_{n\to\infty} \textbf{sup} \left|\frac{f(n)}{g(n)}\right| < \infty$ (need supremum since limit might not exist!)

Variants:

- $f(n) \in \Omega(g(n))$ iff $g(n) \in O(f(n))$ ("Big-Omega")
- $f(n) \in \Theta(g(n))$ iff $f(n) \in O(g(n))$ **and** $f(n) \in \Omega(g(n))$ ("Big-Theta")

Asymptotic tools – Formal & definitive definition

- **"Tilde Notation":** $f(n) \sim g(n)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ (iff: if, and only if)

  „f and g are *asymptotically equivalent*"

- **"Big-Oh Notation":** $f(n) \in O(g(n))$ (also write '=' instead) iff $\left|\frac{f(n)}{g(n)}\right|$ is bounded for $n \geq n_0$

  iff $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$ (need supremum since limit might not exist!)

  Variants:

  - $f(n) \in \Omega(g(n))$ ("Big-Omega") iff $g(n) \in O(f(n))$
  - $f(n) \in \Theta(g(n))$ ("Big-Theta") iff $f(n) \in O(g(n))$ **and** $f(n) \in \Omega(g(n))$

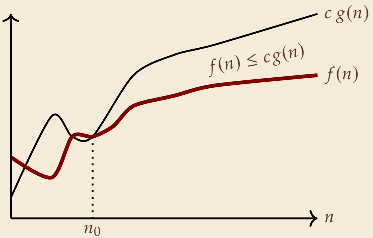
- **"Little-Oh Notation":** $f(n) \in o(g(n))$ iff $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$

  $f(n) \in \omega(g(n))$ if $\lim = \infty$

Asymptotic tools – Intuition

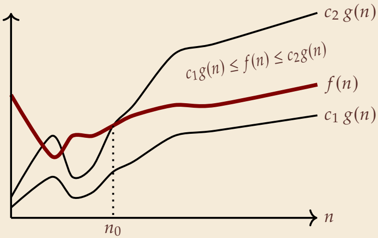
- $f(n) = O(g(n))$: $f(n)$ is **at most** $g(n)$ up to constant factors and for sufficiently large n

  "$f(n) \le g(n)$"



- $f(n) = \Theta(g(n))$: $f(n)$ is **equal to** $g(n)$ up to constant factors and for sufficiently large n

  $f(n) \approx g(n)$



Plots can be misleading!

Example

Clicker Question

Assume $f(n) \in O(g(n))$. What can we say about $g(n)$?

- **A** $g(n) = O(f(n))$
- **B** $g(n) = \Omega(f(n))$
- **C** $g(n) = \Theta(f(n))$
- **D** Nothing (it depends on f and g)

→ `sli.do/comp526`

Clicker Question

"$f(n) \le g(n)$" $\iff$ "$g(n) \ge f(n)$"

Assume $f(n) \in O(g(n))$. What can we say about $g(n)$?

- **A** ~~$g(n) = O(f(n))$~~
- **B** $g(n) = \Omega(f(n))$ ✓
- **C** ~~$g(n) = \Theta(f(n))$~~
- **D** ~~Nothing (it depends on f and g)~~

→ sli.do/comp526

Clicker Question

Assume $f(n) \in O(g(n))$. What can we say about $g(n)$?

- **A** ~~$g(n) = O(f(n))$~~
- **B** $g(n) = \Omega(f(n))$ ✓ (if $f(n) \neq 0$)
- **C** ~~$g(n) = \Theta(f(n))$~~
- **D** Nothing (it depends on f and g) ✓

→ sli.do/comp526

Asymptotics – Example 1

Basic examples:

- $20n^3 + 10n\ln(n) + 5 \sim 20n^3 = \Theta(n^3)$
- $3\lg(n^2) + \lg(\lg(n)) = \Theta(\log n)$
- $10^{100} = O(1)$

$\lg = \log_2$

$f(n) \sim g(n) \quad \frac{f(n)}{g(n)} \xrightarrow[n\to\infty]{} 1$

$$\lim_{n\to\infty} \frac{20n^3 + 10\,n\ln n + 5}{20n^3} = \lim_{n\to\infty} \frac{20n^3}{20n^3} + \lim_{n\to\infty} \frac{10\,n\ln n}{20n^3} + \lim_{n\to\infty} \frac{5}{20n^3}$$

$\frac{10 \ln n}{20 n^2} \le \frac{10}{20n}$, $= 0$, $= 0$

$f(n) = \Theta(n^3)$: $f(n) = O(n^3)$ and $f(n) = \Omega(n^3)$

$\frac{f(n)}{n^3} \to 1 \Rightarrow \frac{f(n)}{n^3}$ is bounded

$\frac{n^3}{f(n)} \to 1 \Rightarrow \frac{n^3}{f(n)}$ is bounded

$$\frac{3 \cdot 2\lg(n) + \lg(\lg(n))}{\log_2 n} \xrightarrow[n\to\infty]{} 6$$

$\Rightarrow \frac{f(n)}{g(n)}$ and $\frac{g(n)}{f(n)}$ bounded

Use *wolfram alpha* to compute/check limits.

Clicker Question

Is $(\sin(n) + 2)n^2 = \Theta(n^2)$?

A Yes

B No

→ sli.do/comp526

Clicker Question

$$\left|\frac{(\sin(n)+2)n^2}{n^2}\right| \le 3$$

$$(\sin(n)+2)n^2 = O(n^2)$$

Is $(\sin(n)+2)n^2 = \Theta(n^2)$?

A Yes ✓

B ~~No~~

$$\frac{n^2}{(\sin(n)+2)n^2} \le \frac{1}{-1+2} = 1$$

→ sli.do/comp526

Asymptotics – Frequently used facts

- Rules:
  - $c \cdot f(n) = \Theta(f(n))$ for constant $c \neq 0$
  - $\Theta(f + g) = \Theta(\max\{f, g\})$ largest summand determines Θ-class

- Frequently used orders of growth:
  - logarithmic $\Theta(\log n)$ Note: $a, b > 0$ constants $\rightsquigarrow$ $\Theta(\log_a(n)) = \Theta(\log_b(n))$
  - linear $\Theta(n)$
  - linearithmic $\Theta(n \log n)$
  - quadratic $\Theta(n^2)$
  - polynomial $O(n^c)$ for constant c
  - exponential $O(c^n)$ for constant c Note: $a > b > 0$ constants $\rightsquigarrow$ $b^n = o(a^n)$

Asymptotics – Example 2

Square-and-multiply algorithm
for computing x^m with $m \in \mathbb{N}$

Inputs:

- m as binary number (array of bits)
- n = #bits in m
- x a floating-point number

```
1 def pow(x, m):
2     # compute binary representation of exponent
3     exponent_bits = bin(m)[2:]
4     result = 1
5     for bit in exponent_bits:
6         result *= result
7         if bit == '1':
8             result *= x
9     return result
```

- Cost: C = # multiplications
- $C = n$ (line ~~4~~ 6) + #one-bits binary representation of m (line ~~5~~ 8)

$\rightsquigarrow$ $n \le C \le 2n$

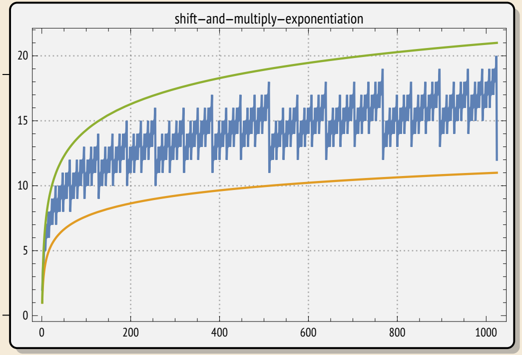
Clicker Question

We showed $n \leq C(n) \leq 2n$; what is the most precise asymptotic approximation for $C(n)$ that we can make?

Write e. g. `O(n^2)` for $O(n^2)$ or `Theta(sqrt(n))` for $\Theta(\sqrt{n})$.

→ *sli.do/comp526*

Asymptotics – Example 2



Square-and-multiply algorithm
for computing x^m with $m \in \mathbb{N}$

Inputs:

- m as binary number (array of bits)
- n = #bits in m
- x a floating-point number

$$\frac{C(n)}{n} \le \frac{2n}{n} = 2$$

- Cost: C = # multiplications
- $C = n$ (line 4) + #one-bits binary representation of m (line 5)

$$\frac{n}{C(n)} \le 1$$

$\rightsquigarrow$ $n \le C \le 2n$

$\rightsquigarrow$ $C = \Theta(n) = \Theta(\log m)$

Note: Often, you can pretend Θ is "like $\sim$ with an unknown constant"
but in this case, no such constant exists!