Introduction to complexity and Big-Oh notation

William Hendrix

Lecture 4

Today

- Review
- Introduction to complexity
- RAM model of computation
- Big-Oh Notation
- Logarithms review

Basic algorithmic strategies

- **Strategy #0:** exhaustive search
  - Try everything
  - Always finds the solution
  - Often prohibitively slow

- **Strategy #1:** greedy search
  - Pick the "best" option at every decision point
  - Applicable to optimization problems (find largest/smallest/etc.)
  - Generally very efficient
  - Might not be correct
  - Need to figure out how to assess "best"

Time complexity

- What are the factors that contribute to the running time of an algorithm?
  - Processor speed
  - Number of instructions executed
  - Cache coherency
  - Resource conflicts (network, hard disk, etc.)

- Which of these are important when comparing algorithms?
  - Processor speed affects fast and slow algorithms equally
    - Not an important factor

- What can we *most reliably* control when designing an algorithm?
  - Number of instructions executed

RAM model of computation

- Set of assumptions that make analysis more reasonable

<u>Assumptions</u>

1. All "basic" operations (assignment, arithmetic, branching, etc.) take 1 operation
   - Loops and functions do not qualify
2. Memory access is instantaneous
   - All variables are in registers
3. We have "infinite" memory

Cons

- Different operations take different number of clock cycles
- Cache locality has significant impact on performance
- Virtual memory can slow performance

Pros

- Can actually analyze algorithms

RAM model example

```
data:  an array of integers to find the min
n:     the number of values in data
```

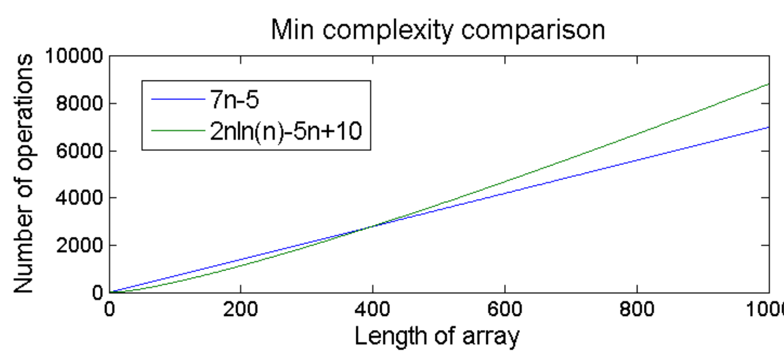
Min algorithm:

	Ops per line	Times executed
`min = 1`	1	1
`for i = 2 to n`	2	n-1
`  if data[i] < data[min]`	3	n-1
`    min = i`	1	??? (≤ n-1)
`  end`	0	n/a
`end`	1	n-1
`return min`	1	1

Total ops: $\leq 7(n-1) + 2 = 7n - 5$

Question: is this better or worse than an algorithm that takes at most $2n \ln n - 5n + 10$ ops?

Better unless n < 396



Big-Oh notation

- Technique for *abstracting away details* of complexity
  - Can be used for time complexity, space complexity, etc.

- **Main idea:** most important aspect of complexity is *how fast it grows* relative to input size
  - Focus on asymptotic (eventual) growth rate
  - "Fast" functions will eventually pass "slow" functions for large *n*
  - Coefficients only matter if growth rate is similar
  - Predicting behavior for small *n* is difficult and often pointless

- Big-Oh notation
  - Organizes growth rates into classes
  - Three main symbols: $O(f(n)), \Omega(f(n)), \Theta(f(n))$
    - Analogous to "at least", "at most", and "similar to" *f(n)*

Big-Oh

- Upper bound *("at most")*

> $f(n) = O(g(n))$ if and only if there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

- We say "*g(n)* dominates *f(n)*" when *f(n)* = *O(g(n))*
- Notation weirdness:
  - *O*, *Ω*, and *Θ* are classes (sets) of functions
  - BUT: we use = to assign class, not ∈
- **Example**
  - Prove that $7n^2 + 19n - 4444 = O(n^2)$.

Proof. If $n \geq 19$,

$$\begin{aligned} 7n^2 + 19n - 4444 &\leq 7n^2 + 19n \\ &\leq 7n^2 + n^2 \\ &= 8n^2 \end{aligned}$$

Therefore, there exist positive constants $c = 8$ and $n_0 = 19$ such that $7n^2 + 19n - 4444 \leq cn^2$ for all $n \geq n_0$. $\square$

Big-Omega

- Lower bound *("at least")*

$f(n) = \Omega(g(n))$ if and only if there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

- **Example**
  - Prove that $7n^2 + 19n - 4444 = \Omega(n)$.

Proof. If $n \geq 4444$,

$$\begin{aligned} 7n^2 + 19n - 4444 &\geq 19n - 4444 \\ &\geq 19n - n \\ &= 18n \end{aligned}$$

Therefore, there exist positive constants $c = 19$ and $n_0 = 4444$ such that $7n^2 + 19n - 4444 \geq cn^2$ for all $n \geq n_0$. $\square$

Big-Theta

- Upper *and* lower bound *("same rate as")*

> $f(n) = \Theta(g(n))$ if and only if there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

- **Example**
  - Prove that $7n^2 + 19n - 4444 = \Theta(n^2)$.

Proof. If $n \geq 4444$,

$$\begin{aligned}
7n^2 + 19n - 4444 &\geq 7n^2 + 19n - n \\
&= 7n^2 + 18n \\
&\geq 7n^2 \\
7n^2 + 19n - 4444 &\leq 7n^2 + 19n \\
&\leq 7n^2 + n^2 \\
&= 8n^2
\end{aligned}$$

Therefore, there exist positive constants $c_1 = 7$, $c_2 = 8$, and $n_0 = 4444$ such that $c_1 n^2 \leq 7n^2 + 19n - 4444 \leq c_2 n^2$ for all $n \geq n_0$. $\square$

Connection to calculus

- You can also determine *O*, *Ω*, and *Θ* by limits:

g grows faster ⟶ $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \quad \to f(n) = O(g(n))$$

Same growth rate ⟶ $$\lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0, \infty) \to f(n) = \Theta(g(n))$$

g grows slower ⟶ $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \quad \to f(n) = \Omega(g(n))$$

- Standard rules for taking limits apply
  - Including L'Hôpital's Rule

Observations on Big-Oh

- Big-Oh can be larger than needed
  - $n^3 = O(n^3), O(n^4), O(n^5) \ldots$
- Big-Omega can be smaller than needed
- *Analogy*: Big-Oh "acts like" ≤, Big-Omega ≥, and Big-Theta =
- We will generally look for tight upper bounds ($O(f(n))$) in this class
- Most algorithms we discuss will belong to the following classes:

$$O(1) \ll O(\lg n) \ll O(n) \ll O(n \lg n) \ll O(n^2) \ll O(n^3) \ll O(2^n) \ll O(n!)$$

  - Constant, logarithmic, linear, n log n (or "linearithmic"), quadratic, cubic, exponential, or factorial

Proofs

- Use formal definitions!!!
- Finding smallest c or n_o isn't necessary
  - Choosing well can make your life easier, though

Big-Oh exercises

- Use the *formal definitions* of Big-Oh, Big-Omega, and Big-Theta to prove the following:

1. $$\frac{n(n+1)}{2} = O(n^2)$$

2. If $f(n) = O(g(n)$, $g(n) = \Omega(f(n))$.

3. If $f(n) = \Omega(g(n)$ and $g(n) = \Omega(h(n))$, then $f(n) = \Omega(h(n))$.

Big-Oh exercises

- Use the *formal definitions* of Big-Oh, Big-Omega, and Big-Theta to prove the following:

1. *Proof.* If $n \geq 1$, $n \leq n^2$, so

$$\begin{aligned} \frac{n(n+1)}{2} &\leq n(n+1) \\ &= n^2 + n \\ &\leq n^2 + n^2 \\ &= 2n^2 \end{aligned}$$

Hence, there exist constants $c = 2$ and $n_0 = 1$ such that $\frac{n(n+1)}{2} \leq cn^2$ for all $n \geq n_0$, so $\frac{n(n+1)}{2} = O(n^2)$. $\square$

Big-Oh exercises

- Use the *formal definitions* of Big-Oh, Big-Omega, and Big-Theta to prove the following:

2. *Proof.* If $f(n) = O(g(n))$, there exist positive constants c_1 and n_0 such that $f(n) \leq c_1 g(n)$ for all $n \geq n_0$. Since $c_1 > 0$, we can multiply both sides of this expression by $\frac{1}{c}$, yielding $\left(\frac{1}{c}\right) f(n) \leq g(n)$ for all $n \geq n_0$. Thus, there exist positive constants $c_2 = \frac{1}{c}$ and $n_2 = n_0$ such that $g(n) \geq c_2 f(n)$ for all $n \geq n_2$, so $g(n) = \Omega(f(n))$. $\square$

3. *Proof.* If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, there exist positive constants c_1, c_2, n_0, and n_1 such that $f(n) \leq c_1 g(n)$ for all $n \geq n_0$ and $g(n) \leq c_2 h(n)$ for all $n \geq n_1$. In particular, if we let $n_2 = \max\{n_0, n_1\}$, $f(n) \leq c_1 g(n)$ and $g(n) \leq c_2 h(n)$ for all $n \geq n_2$, so $f(n) \leq c_1(c_2 h(n))$. Thus, there exist constants $c_3 = c_1 c_2$ and $n_2 = \max\{n_0, n_1\}$ such that $f(n) \leq c_3 h(n)$ for all $n \geq n_2$, so $f(n) = O(h(n))$. $\square$

Properties of Big-Oh notation

- Transitivity

$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \rightarrow f(n) = \Omega(h(n))$$
$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \rightarrow f(n) = \Theta(h(n))$$

- Equivalence rules

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$
$$f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)) \Leftrightarrow f(n) = \Theta(f(n))$$

- Reflexivity and symmetry

$$f(n) = O(f(n)),\ f(n) = \Omega(f(n)), \text{ and } f(n) = \Theta(f(n))$$
$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

- All three ignore constant coefficients

$$\forall x > 0, xf(n) = O(f(n)),\ xf(n) = \Omega(f(n)), \text{ and}$$
$$xf(n) = \Theta(f(n))$$

- Only the largest term matters

$$f(n) = O(g(n)) \rightarrow O(f(n) + g(n)) = O(g(n))$$
$$f(n) = O(g(n)) \rightarrow \Omega(f(n) + g(n)) = \Omega(g(n))$$
$$f(n) = O(g(n)) \rightarrow \Theta(f(n) + g(n)) = \Theta(g(n))$$

Coming up

- Big-Oh practice

- **Homework 2** is due tonight
- **Homework 3** is due Tuesday
- **Homework 4** is due Thursday

- **Recommended readings:** Section 2.5
- **Practice problems:** attempt 1-2 problems from "Interview Problems" (p. 63)