Algorithm Analysis: More Analyses & Amortization

CSE 332 Summer 2021

Instructor: Kristofer Wong

Teaching Assistants:

Alena Dickmann	Arya GJ	Finn Johnson
Joon Chong	Kimi Locke	Peyton Rapo
Rahul Misal	Winston Jodjana	

Announcements

- Projects!
  - Expected behavior in javadocs
  - You will fill out the partner matching form for each project – allows you to switch partners if you wish
  - Checkpoints
- Exercise 3 still released tomorrow, two weeks instead of 1

Communication Exercises

Partnership 1	Partnership 2
A1	A2
B1	B2

- 2 parts:
  - A1 & A2 work together on part 1
  - B1 & B2 work together on part 2
- "Work together" can mean:
  - Literally solving the problem together
  - Solving separately, asking each other questions
  - Solving separately, comparing answers
- Canvas Groups with discussion assignment
  - Post your video by Wednesday, 11:59
  - Reply to all the videos with feedback by Friday
- Use your peers' videos to solve the part you did not solve, turn in exercise to gradescope with a reflection
- Check your email often

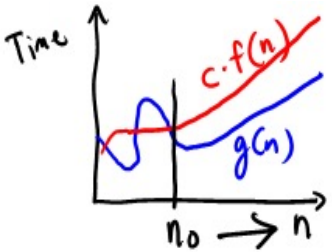
Lecture Outline

- Asymptotics Wrapup
  - **Review: Big-O, Formally**
  - Big-Omega and Big-Theta
- Amortization
- Closing Thoughts on Algorithm Analysis

Big-Oh relates functions

- We use *O* on a function f(*n*) (for example n^2) to mean *the set of functions with asymptotic behavior* ***less than or equal to*** f(*n*)

Big-Oh Review



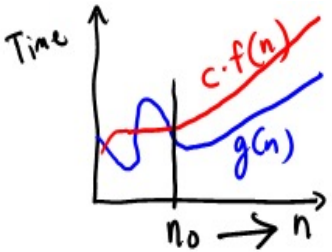
Definition: $g(n)$ is in $O(f(n))$ iff there exist positive constants c and n_0 such that

$g(n) \leq c\, f(n)$ for all $n \geq n_0$

Note: $n_0 \geq 1$ (and a natural number) and $c > 0$

To show $g(n)$ is in $O(f(n))$, pick a c large enough to "cover the constant factors" and n_0 large enough to "cover the lower-order terms"

Big-Oh Example



Definition: **g(*n*)** is in O(**f(*n*)**) iff there exist positive constants ***c*** and n_0 such that

$g(n) \leq c\, f(n)$ for all $n \geq n_0$

Note: $n_0 \geq 1$ (and a natural number) and $c > 0$

Example: Let **g(*n*)** = $3n + 4$ and **f(*n*)** = n

$c = 4$ and $n_0 = 5$ is one possibility

The definition says "less than or equal", so:

$3n+4$ is also in $O(n^5)$, $O(2^n)$, etc.

What's with the c?

- To capture this notion of "similar asymptotic behavior", we allow a constant multiplier called **c**. Consider:

  g(n) = 3n+4

  f(n) = n

- These have the same asymptotic behavior (linear), even though **g(n)** is always larger
  - ie, there is **no** positive n_0 such that **g(n)** ≤ **f(n)** for all n ≥ n_0

- The '**c**' allows us to show their asymptotic relationship:

  $g(n) \leq c\, f(n)$ for all $n \geq n_0$

- To show **g(n)** is in O(**f(n)**), let **c** = 12, n_0 = 1

Example

To show **g(*n*)** is in O(**f(*n*)**), pick a ***c*** large enough to "cover the constant factors" and n_0 large enough to "cover the lower-order terms"

Let **g(*n*)** = $4n^2 + 3n + 4$, **f(*n*)** = n^3.

Prove that **g(*n*)** is in O(**f(*n*)**).

gradescope

gradescope.com/courses/275833

Question 1 (2 minutes)

- Explain why solving for **c** and n_0 within a proof is considered backward reasoning.

Reviewing the Big-O Rules

- Eliminate coefficients
  - $3n^2$ versus $5n^2$ both have similar behavior as n goes to infinity
- Eliminate low-order terms because they have vanishingly small impact as *n* grows
- Do NOT ignore constants that are not multipliers
  - n^3 is not $O(n^2)$
  - 3^n is not $O(2^n)$

(These all follow from the formal definition)

Common Complexity Classes

$O(1)$ **(O(k) for any k)*	Constant
$O(\log \log n)$	
$O(\log n)$	Logarithmic
$O(\log^k n)$ **(for any k>1)*	
$O(n)$	Linear
$O(n \log n)$	Loglinear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(n^k)$ **(for any k>1)*	Polynomial
$O(k^n)$ **(for any k>1)*	Exponential

Lecture Outline

- Asymptotics Wrapup
  - Review: Big-O, Formally
  - **Big-Omega and Big-Theta**
- Amortization
- Closing Thoughts on Algorithm Analysis

More functions: Omega, Theta

- **Upper bound**: $O(\ \mathbf{f(n)}\ )$ is the set of all functions asymptotically *less than or equal* to **f(n)**
  - g(n) is in $O(\ \mathbf{f(n)}\ )$ if there exist constants $\boldsymbol{c}$ and $\boldsymbol{n_0}$ such that

    $g(n) \leq \boldsymbol{c}\ \mathbf{f(n)}$ for all $n \geq \boldsymbol{n_0}$

- **Lower bound**: $\Omega(\ \mathbf{f(n)}\ )$ is the set of all functions asymptotically *greater than or equal* to **f(n)**
  - g(n) is in $\Omega(\ \mathbf{f(n)}\ )$ if there exist constants $\boldsymbol{c}$ and $\boldsymbol{n_0}$ such that

    $g(n) \geq \boldsymbol{c}\ \mathbf{f(n)}$ for all $n \geq \boldsymbol{n_0}$

- **Tight bound**: $\theta(\ \mathbf{f(n)}\ )$ is the set of all functions asymptotically *equal* to **f(n)**
  - Intersection of $O(\ \mathbf{f(n)}\ )$ and $\Omega(\ \mathbf{f(n)}\ )$ (can use *different* $\boldsymbol{c}$ and $\boldsymbol{n_0}$ values)

Big-O: Intuition

- Big-O can be thought of as something like "less-than or equals"

Function	Big-O	Also Big-O
$N^3 + 3N^4$	$O(N^4)$	$O(N^5)$
$(1 / N) + N^3$	$O(N^3)$	$O(N^{423421531542})$
$Ne^N + N$	$O(Ne^N)$	$O(N*3^N)$
$40 \sin(N) + 4N^2$	$O(N^2)$	$O(N^{2.1})$

g(*n*) is in O(**f(*n*)**) iff there exist positive constants ***c*** and n_0 such that

g(*n*) $\leq$ ***c*** **f(*n*)** for all $n \geq n_0$

Big-Omega: Intuition

- Big-Omega can be thought of as something like "greater-than or equals"

Function	Big-O	Big-Omega	Also Big-Omega
$N^3 + 3N^4$	$O(N^4)$	$\Omega(N^4)$	$\Omega(N^2)$
$(1 / N) + N^3$	$O(N^3)$	$\Omega(N^3)$	$\Omega(1)$
$Ne^N + N$	$O(Ne^N)$	$\Omega(Ne^N)$	$\Omega(N)$
$40 \sin(N) + 4N^2$	$O(N^2)$	$\Omega(N^2)$	$\Omega(N)$

$g(n)$ is in $\Omega(\ f(n)\ )$ iff there exist positive constants c and n_0 such that

$g(n) \geq c\ f(n)$ for all $n \geq n_0$

Big-Theta: Intuition

- Big-Theta more closely resembles "equals"

Function	Big-O	Big-Omega	Big-Theta
$N^3 + 3N^4$	$O(N^4)$	$\Omega(N^4)$	$\Theta(N^4)$
$(1 / N) + N^3$	$O(N^3)$	$\Omega(N^3)$	$\Theta(N^3)$
$Ne^N + N$	$O(Ne^N)$	$\Omega(Ne^N)$	$\Theta(Ne^N)$
$40 \sin(N) + 4N^2$	$O(N^2)$	$\Omega(N^2)$	$\Theta(N^2)$

$g(n)$ is in $\Theta(f(n))$ iff there exist positive constants c and n_0 such that

$c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n \geq n_0$

Big-O, Big-Theta, Big-Omega Relationship

- If a function f is in Big-Theta, what does it mean for its membership in Big-O and Big-Omega? Vice versa?

Function	Big-O	Big-Theta	Big-Omega
$N^3 + 3N^4$	$O(N^4)$	$\Theta(N^4)$	$\Omega(N^4)$
$(1 / N) + N^3$		$\Theta(N^3)$	
$Ne^N + N$		$\Theta(Ne^N)$	
$40 \sin(N) + 4N^2$		$\Theta(N^2)$	

A Warning about Terminology

- A common error is to say $O(\ f(n)\ )$ when you mean $\theta(\ f(n)\ )$
  - People often say O() to mean a tight bound
    - Say we have f(n)=n; we could say f(n) is in O(n), which is true, but only conveys the upper-bound
    - Since f(n)=n is *also* $O(n^5)$, it's tempting to say "this algorithm is *exactly* $O(n)$"
    - It's better to say it is $\theta(n)$
      - That means that it is not, for example $O(\log n)$

- Less common notation:
  - "little-oh": like "big-Oh" but strictly less than
    - Example: f(n) is $o(n^2)$ but not $o(n)$
  - "little-omega": like "big-Omega" but strictly greater than
    - Example: f(n) is $\omega(\log n)$ but not $\omega(n)$

What We are Analyzing

- The most common thing to do is give an *O* or θ **bound** to the **worst-case** running **time** of an **algorithm**

- Reminder that Case Analysis != Asymptotic Analysis
  - Cases describe *a specific path through your algorithm*
  - Big-O/Big-Omega/Big-Theta bounds describe *curve shapes for large values*

- When comparing two algorithms, you must pick all of these:
  - A case (eg, best, worst, amortized, etc)
  - A metric (eg, time, space)
  - A bound type (eg, big-O, big-Theta, little-omega, etc)

Example of analysis

- True statements about binary-search ***algorithm***:
  - Common: $\theta(\log n)$ running-time in the worst-case
  - Less common: $\theta(1)$ in the best-case
    - item is in the middle
  - Less common: $\Omega(\log \log n)$ in the worst-case
    - it is not really, really, really fast asymptotically
- The find-in-sorted-array ***problem*** is $\Omega(\log n)$ in the worst-case
  - *No* algorithm can do better (without parallelism)
  - A ***problem*** cannot be $O(f(n))$ since you can always find a slower algorithm, but can mean ***there exists*** an algorithm

gradescope

gradescope.com/courses/275833

Questions 2 & 3 (4 minutes)

- Answer True or False for some asymptotic relationships
- Give an example where there exists a big o, omega, but no theta

Lecture Outline

- Asymptotics Wrapup
  - Review: Big-O, Formally
  - Big-Omega and Big-Theta
- **Amortization**
- Closing Thoughts on Algorithm Analysis

Linear Search: Best vs Worst Case

- Find an integer in a *sorted* array

2	3	5	16	37	50	73	75	126

```
// Requires arr to be sorted
// Returns whether k is in array
boolean findSorted(int[] arr, int k) {
  for(int i=0; i < arr.length; ++i) {
    if(arr[i] == k)
      return true;
    else if(arr[i] > k)
      return false;
  }
  return false;
}
```

Best k:

Worst k:

Complexity Cases

- We started with two cases:
  - **Worst-case complexity**: *maximum* number of steps algorithm takes on "most challenging" input of size N
  - **Best-case complexity**: *minimum* number of steps algorithm takes on "easiest" input of size N

- We punted on one case: **Average-case complexity**
  - Sometimes: relies on distribution of inputs
    - Eg, binary heap's O(1) insert
    - See CSE312 and STAT391
  - Sometimes: uses randomization in the algorithm
    - Will see an example with sorting; also see CSE312

- We've mentioned, but not defined, one *category* of cases:
  - **Amortized-case complexity**

Amortized Analyses = Multiple Executions

Single Execution	Multiple Executions
Worst Case	Amortized Worst Case
Best Case	Amortized Best Case
Average Case	*Amortized Average Case*

Amortized Analysis: Adding to array-backed structures

Amortized Analysis Wrapup

- So, the runtime for adding to our array-backed structure is:

Single Execution	Multiple Executions
Worst Case: **Θ(N)**	Amortized Worst: **Θ(1)**
Best Case: **Θ(1)**	Amortized Best: **Θ(1)**

- See Weiss, ch 11 for more!

Lecture Outline

- Asymptotics Wrapup
  - Review: Big-O, Formally
  - Big-Omega and Big-Theta
- Amortization
- **Closing Thoughts on Algorithm Analysis**

Closing Thoughts: Multivariable

- big-Oh can also use more than one variable
  - Example: can sum all elements of an *n*-by-*m* matrix in $O(nm)$

Closing Thoughts: When NOT to Use Big-Oh

- Asymptotic complexity (Big-Oh) describes behavior for **large *n*** and is independent of any computer / coding trick

- Asymptotic complexity for ***small n*** can be misleading
  - Example: $n^{1/10}$ vs. `log` n
    - Asymptotically, $n^{1/10}$ grows more quickly
    - But the "cross-over" point (n_0) is around $5*10^{17} \approx 2^{58}$; you might prefer $n^{1/10}$
  - Example: QuickSort vs InsertionSort
    - *Expected runtimes:* Quicksort is O(n log n) vs InsertionSort $O(n^2)$
    - In reality, InsertionSort is faster for small n's
    - (we'll learn about these sorts later)

- Asymptotic complexity for ***specific implementations*** can also be misleading …

Closing Thoughts: Timing vs. Big-Oh?

- *Evaluating an algorithm?* Use asymptotic analysis
- *Evaluating an implementation?* Timing can be useful
  - Either a hardware or a software implementation

- At the core of CS is a backbone of theory & mathematics
  - We've spent 2 lectures talking about how to analyze the algorithm itself, mathematically, not the implementation
  - Reason about performance as a function of n

- Yet, timing has its place
  - In the real world, we do want to know whether implementation A runs faster than implementation B on data set C
  - Ex: Benchmarking graphics cards

Algorithm Analysis Summary (1 of 2)

- **What are we analyzing**: Problem or the algorithm
- **Metric**: Time or space
  - Or power, or dollars, or …
- **Complexity Bounds**:
  - Describing curve shapes "at infinity"
    - 'c' allows us to ignore effect of multiplicative constants on curve shape
    - 'n_0' allows us to ignore effect of low-order terms on curve shape
  - *Upper bound*: big-O or little-o
  - *Lower bound*: big-Ω or little-ω
  - *Tight bound*: Θ

Algorithm Analysis Summary (2 of 2)

- **Complexity Cases**: two different dimensions:
  - The specific path through an algorithm for input of size N
    - ***Worst-case***: max # steps on "most challenging" input
    - ***Best-case***: min # steps on "easiest" input
    - ***Average-case****: varying definitions, typically not used in 332*
  - Number of executions considered
    - ***Single-execution***
    - ***Multiple-execution***: *amortized case* is only one of several techniques for combining executions

- Usually:
  - We analyze the algorithm's time complexity to understand its upper or tight bound for a single-execution's worst-case