CSE 373: Data Structures and Algorithms

Lecture 5: Finishing up Asymptotic Analysis

Big-O, Big-Ω, Big-θ, little-o, little-ω & Amortized Analysis

Instructor: Lilian de Greef
Quarter: Summer 2017

Today:

- Announcements
- Big-O and Cousins
    - Big-Omega
    - Big-Theta
    - little-o
    - little-omega
- Average running time: Amortized Analysis

News about Sections

Updated times:

- **10:50** – 11:50am
- 12:00 – **1:00**pm

Bigger room!

- 10:50am section now in **THO 101**

Which section to attend:

- Last week, section sizes were unbalanced (~40 vs ~10 people)
- If you can, I encourage you to choose the 12:00 section to rebalance sizes
  - Helps the 12:00 TA's feel less lonely
  - More importantly: improves TA:student ratio in sections (better for tailoring section to your needs)

Homework 1

- Due today at 5:00pm!

- A note about grading methods:
  - Before we grade, we'll run a script on your code to replace your name with `### anonymized ###` so we won't know who you are as we grade it (to address unconscious bias).

  - It's still good practice to have your name and contact info in the comments!

Homework 2

- Written homework about asymptotic analysis (no Java this time)
- Will be out this evening
- Due Thursday, July 6th at 5:00pm
  - Because July 4th is a holiday

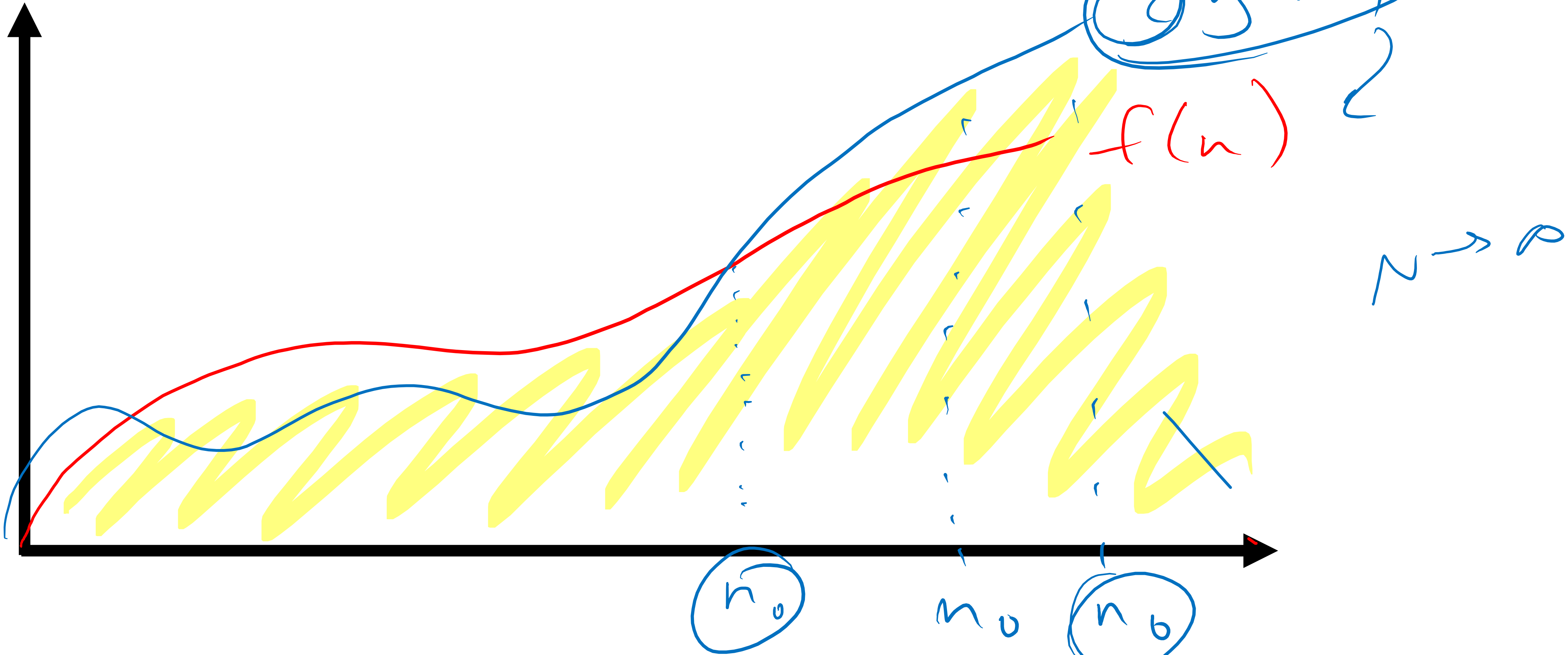
- A note for help on homework:
  - Note that holidays means fewer office hours
  - Remember: although you cannot share solutions, you can talk to classmates about concepts or work through non-homework examples (e.g. from section) together.
  - Give these classmates credit, write their names at the top of your homework.

Big-O: Formal Definition

(Finishing up from last time)

Formal Definition of Big-O

Definition: f(n) is in **O(g(n))** if there exist constants c and n_0 such that f(n) ≤ c g(n) for all $n \geq n_0$



More Practice with the Definition of Big-O

Let $a(n) = 10n+3n^2$ and $b(n) = n^2$

What are some values of c and n_0 we can use to show $a(n) \in O(b(n))$?

$c = 50$

$10n + 3n^2 \leq 50n^2$

$10 \leq 47n \to \frac{10}{47} \leq n$

$n_0 = 1$

$c = 50 \quad n_0 = 10$

$a(10) = 10(10) + 3(10)^2 = 400$

$c\, b(10) = 50(10)^2 = 5000$

$n_0 > 0$

wts: $10n + 3n^2 \leq 50n^2$

$10n \leq 47n^2$

$10 \leq 47n$

$n > 10$

$10 \leq 47 \ (n > 10)$

Definition: $f(n)$ is in $O(g(n))$ if there exist constants c and n_0 such that $f(n) \leq c\, g(n)$ for all $n \geq n_0$

Constants and Lower Order Terms

$O(\underbrace{100000}n^2)$

- The constant multiplier c is what allows functions that differ only in their largest coefficient to have the same asymptotic complexity

  Example: $\underline{2}n^2 \in O(n^2)$ $\qquad \underline{1000000}n^2 + \underbrace{n+3} \in O(n^2)$

- Eliminate lower-order terms because they become negligible as $n \to \infty$

- Eliminate coefficients because we don't have "units of execution"
  - $3n^2$ vs $5n^2$ is meaningless without the cost of constant-time operations
  - Can always re-scale anyways
  - Do not ignore constants that are not multipliers! n^3 is not $O(n^2)$, 3^n is not $O(2^n)$

Constants and Lower Order Terms

- The constant multiplier ***c*** is what allows functions that differ only in their largest coefficient to have the same asymptotic complexity

  e.g. for $g(n) = 3n^2$ and $h(n) = 9999n^2 + 9999n + 2$ and $f(n) = n^2$,

  $g(n)$ and $h(n)$ are both in $O(f(n))$

Analyzing “Worst-Case” Cheat Sheet

Basic operations take “some amount of” constant time

- Arithmetic (fixed-width)
- Assignment
- Access one Java field or array index
- *etc.*

(This is an *approximation* of reality: a very useful “lie”)

Control Flow	Time Required
Consecutive statements	Sum of time of statement
Conditionals	Time of test plus slower branch
Loops	Sum of iterations * time of body
Method calls	Time of call’s body
Recursion	Solve *recurrence relation*

Cousins of Big-O

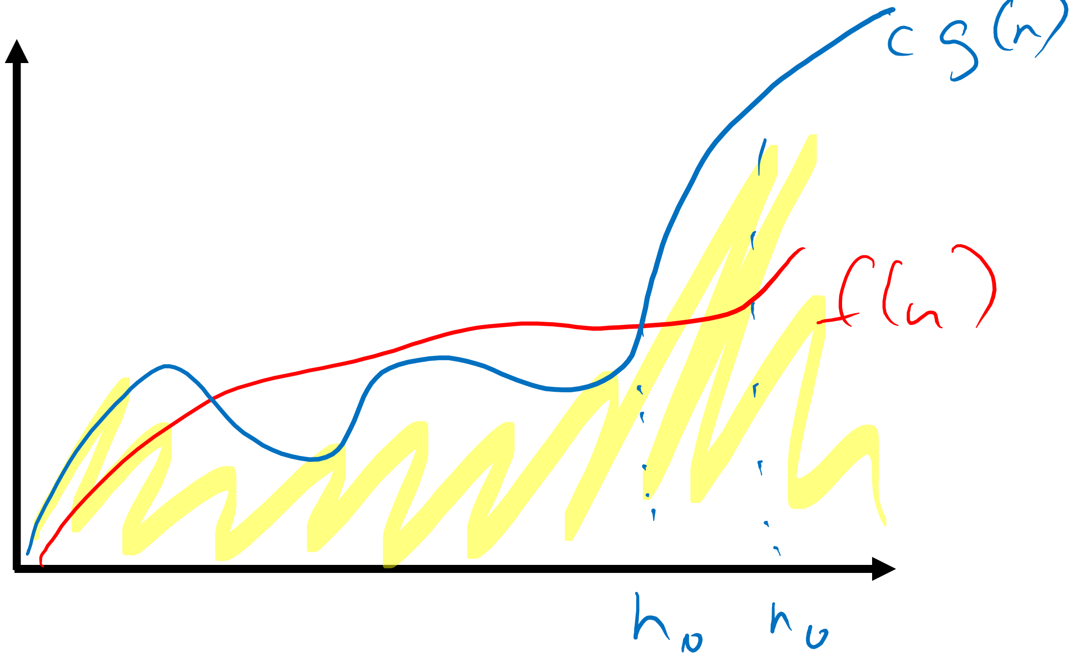
Big-O, Big-Omega, Big-Theta, little-o, little-omega

Big-O & Big-Omega

Big-O: Upper Bound

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

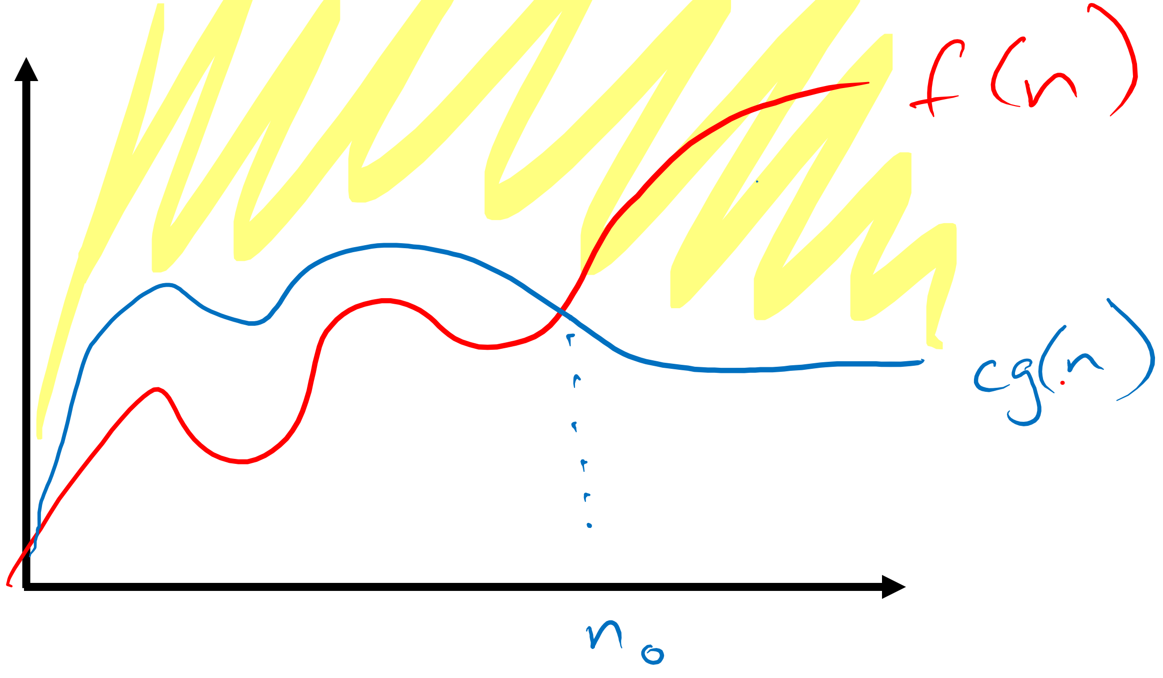
f(n) $\leq$ c g(n) for all $n \geq n_0$



Big-Ω: Lower Bound

f(n) is in **Ω (g(n))** if there exist constants c and n_0 such that

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



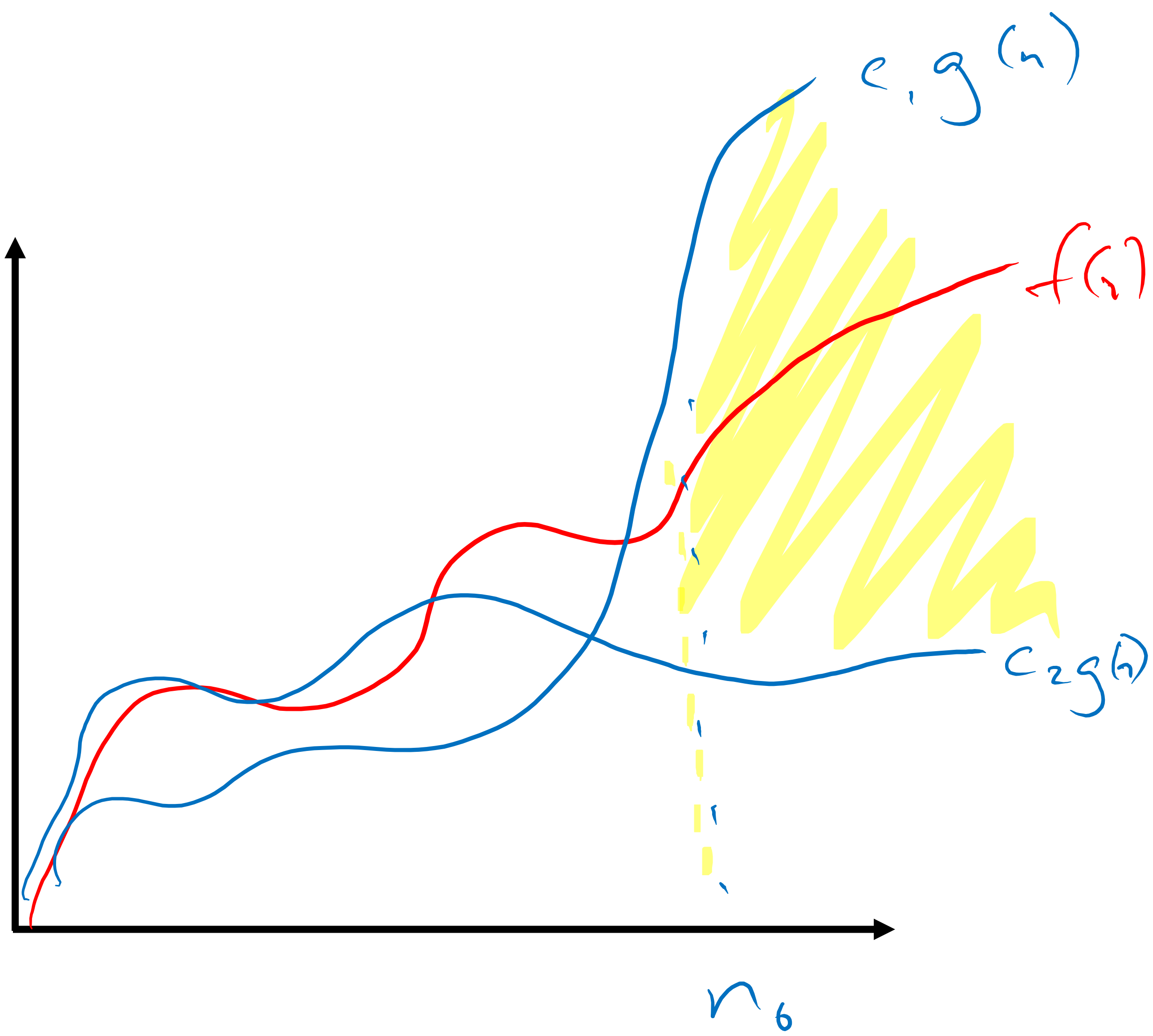
Big-Theta

Big-θ: Tight Bound

f(*n*) is in **θ(** g(*n*) **)** if f(n) is in both O(g(*n*)) *and* Ω (g(*n*))

use two different c's

(c_1 & c_2)



little-o & little-omega

little-o: STRONG upper bound

f(n) is in **o(** g(n) **)** if for all constants c >0 there exists an n_0 s.t. f(n) $<$ c g(n) for all $n \geq n_0$

$<$ ~~$\leq$~~

little-ω: STRONG lower bound

f(n) is in **ω(** g(n) **)** if for all constants c >0 there exists an n_0 s.t. f(n) $>$ c g(n) for all $n \geq n_0$

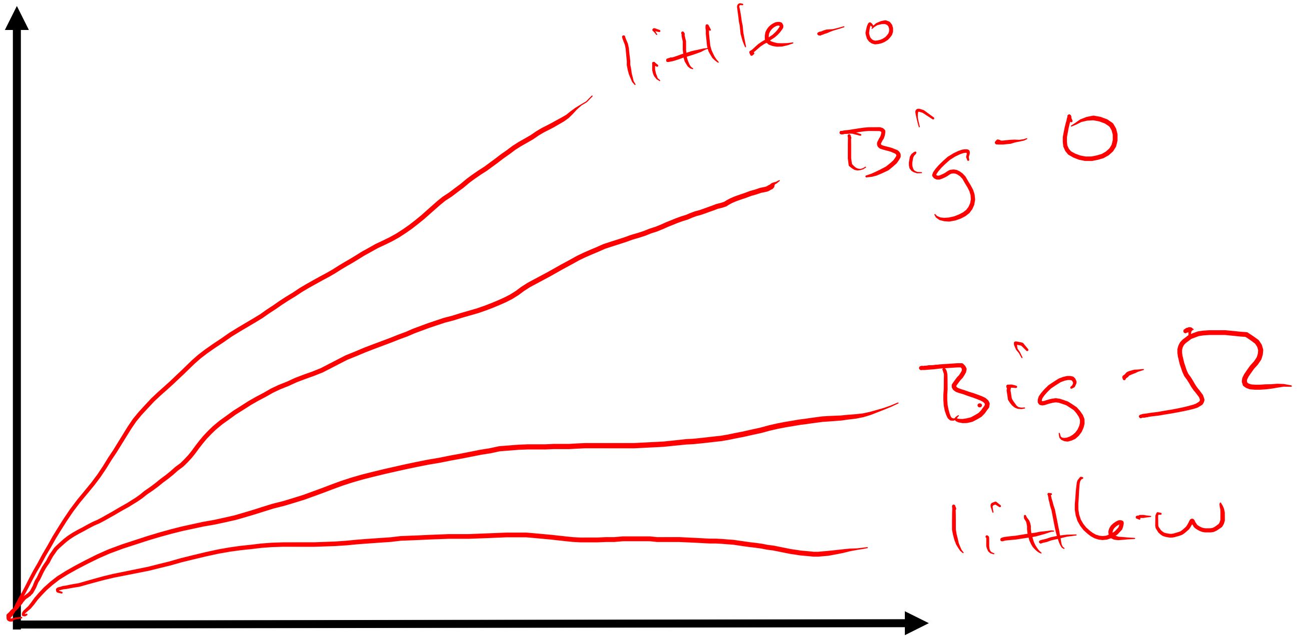
$f(n) = 4n^2$

$O(n^2)$ ✓

$O(n^5)$ ✓

$o(n^2)$ ✗

$o(n^5)$ ✓



Practice Time!

n_0 = integer > 0
c = any rational # > 0

Let $f(n) = 75n^3 + 2$ and $g(n) = n^3 + 6n + 2n^2$
Then $f(n)$ is in… (choose all that apply)

c = 75 $\quad 75n^3 + 2 \leq 75(n^3 + 6n + 2n^2)$ $\quad n > 1$, $n_0 = 1$

A. Big-O(g)
B. Big-Ω(g)
C. θ(g)
D. little-o(g)
E. little-ω(g)

$\exists c \quad f(n) \geq c\, g(n)$ $\quad 75n^3+2 \geq 0.00001(\quad)$

$0 < c < 1$

$\Theta(g)$ = intersection $O(g)$ & $\Omega(g)$

Second Practice Time!

$f(n) = 4n^2$

$O(n^2)$ ✓

$o(n^2)$ ✗

Let $f(n) = 3^n$ and $g(n) = n^3$

Then $f(n)$ is in… (choose all that apply)

no value of c & n_0 for $f(n) \le c\,g(n)$ for $n > n_0$

A. Big-O(g)

B. Big-Ω(g)

C. θ(g)

D. little-o(g)

E. little-ω(g)

If little-ω is true

→ Big Ω is true

If little-o is true

→ Big-O

Big-O, Big-Omega, Big-Theta

$f(n) = 4n^2$

$O(n^2)$

$O(n^3)$

$\boxed{O(2^n)}$

$\Theta(n^2)$

- Which one is more useful to describe asymptotic behavior?

Big-Θ is more specific

- A common error is to say *O*(f(*n*)) when you mean θ(f(*n*))
  - A linear algorithm is in both O(n) and *O*(*n*5) ← $O(n^5)$
  - Better to say it is θ(*n*)
  - That means that it is not, for example *O*(**log** *n*)

Comments on Asymptotic Analysis

- Is choosing the lowest Big-O or Big-Theta the best way to choose the fastest algorithm?

worst-case

No!

sometimes we care about
is average case

- Big-O can use other variables (e.g. can sum all of the elements of an n-by-m matrix in O(nm))

Amortized Analysis

How we calculate the average time!

Case Study: the Array Stack

What's the **worst-case** running time of `push()`?

$\Theta(n)$ $O(n)$

What's the **average** running time of `push()`?

$O(1)$

Calculating the average: not based off of running a *single operation*,
but running *many operations in sequence*.

Technique: **Amortized Analysis**

Amortized Cost

The **amortized cost** of *n* operations is the worst-case total cost of the operations divided by *n*.

if $T(n)$ = worst case / upper bound

for n = # operations

$\rightarrow$ Amortized cost $= \dfrac{T(n)}{n}$

Amortized Cost

The **amortized cost** of *n* operations is the worst-case total cost of the operations divided by *n*.

Practice:

- n operations taking $O(n)$ → amortized cost = $\frac{O(n)}{n} = O(1)$
- n operations taking $O(n^3)$ → amortized cost = $O(n^2)$
- n operations taking $O(n\, f(n))$ → amortized cost = $\frac{O(\cancel{n} f(n))}{\cancel{n}} = O(f(n))$

Example: Array Stack

What's the amortized cost of calling `push()` *n* times if we double the array size when it's full?

n operations

- n pushes @ O(1) each → cost is $\underline{n}$
- cost of resizing (doubling array when full) $= n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \dots$

⇒ upper bound $= \underline{2n}$

total cost $= 3n$

amortized cost $= \frac{3n}{n} = 3$ ← $\boxed{O(1)}$

The **amortized cost** of *n* operations is the worst-case total cost of the operations divided by *n*.

Another Perspective: Paying and Saving "Currency"

A	B	C	D

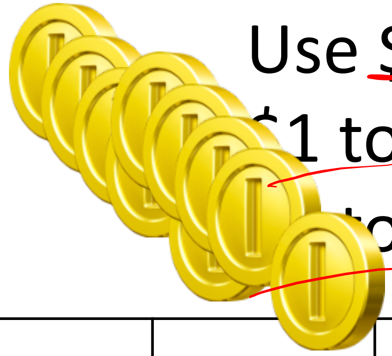
1 operation costs us
1$ to the computer

A	B	C	D	E			

9

Another Perspective: Paying and Saving "Currency"

A	B	C	D



Use $2 for each push:

1 to computer,

to bank

A	B	C	D	E			

Potential Function

4

Spend our savings in the bank to resize. That way it only costs $1 to push(E)!

9