

# Algorithm Analysis 2

CSE 332 Summer 2020

**Instructor:** Richard Jiang

## Teaching Assistants:

Hamsa Shankar Kristin Li Winston Jodjana

Maggie Jiang Hans Zhang Michael Duan

Jeffery Tian Annie Mao

*Lecture Q&A: [pollev.com/332summer](https://pollev.com/332summer)*

*Lecture clarifications: [tinyurl.com/332-06-26A](https://tinyurl.com/332-06-26A)*

# Announcements

- ❖ Quiz 1 due tonight, 11:59pm
- ❖ Project 1's Checkpoint open Tuesday!
  - Checkpoint is a Gradescope-administered survey
  - You should have contacted your P1 partner by now!
- ❖ Three exercises released today; all \*due on Monday
  - Ex 2: code modeling
  - Ex 3: big-O proofs

*[pollev.com/332summer](https://pollev.com/332summer) :: [tinyurl.com/332-06-24A](https://tinyurl.com/332-06-24A)*

# Lecture Outline

- ❖ Algorithm Analysis
  - **Review: Big-O, Formally**
  - Big-Omega and Big-Theta
  - Proof example
  - Amortized bounds
  - Where We've Come

*[pollev.com/332summer](https://pollev.com/332summer) :: [tinyurl.com/332-06-24A](https://tinyurl.com/332-06-24A)*

# Big-Oh relates functions

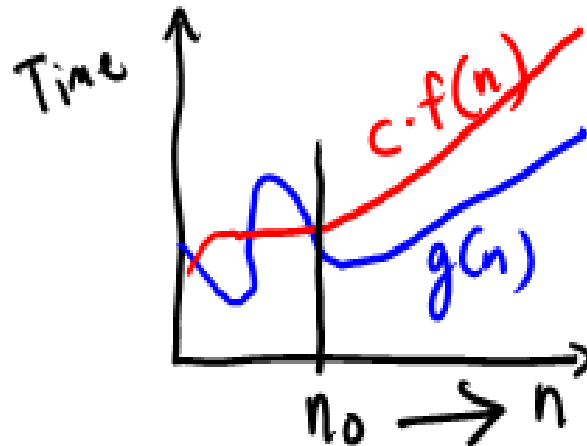
- ❖ We use  $O$  on a function  $f(n)$  (for example  $n^2$ ) to mean *the set of functions with asymptotic behavior less than or equal to  $f(n)$*
- ❖ So  $(3n^2+17)$  **is in**  $O(n^2)$ 
  - $3n^2+17$  and  $n^2$  have the same **asymptotic behavior**
- ❖ Confusingly, we also say/write:
  - $(3n^2+17)$  **is**  $O(n^2)$
  - $(3n^2+17)$  **∈**  $O(n^2)$
  - $(3n^2+17)$  **=**  $O(n^2)$  ← *least ideal*
- ❖ But we would never say  $O(n^2) = (3n^2+17)$

# Big-Oh, Formally (1 of 3)

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

$$g(n) \leq c f(n) \quad \text{for all } n \geq n_0$$

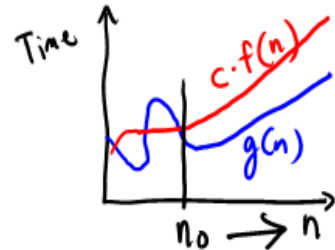
Note:  $n_0 \geq 1$  (and a natural number) and  $c > 0$



## Big-Oh, Formally (2 of 3)

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

$$g(n) \leq c f(n) \quad \text{for all } n \geq n_0$$



**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”

# 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), so  $g(n)$  is in  $O(f(n))$  even though  $g(n)$  is always larger
- ❖ There is no positive  $n_0$  such that  $g(n) \leq f(n)$  for all  $n \geq n_0$ 
  - The ‘ $c$ ’ in the definition allows for that:
$$g(n) \leq c f(n) \quad \text{for all } n \geq n_0$$
  - To show  $g(n)$  is in  $O(f(n))$ , let  $c = 12$ ,  $n_0 = 1$

# An 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”

❖ Example: Let  $g(n) = 4n^2 + 3n + 4$  and  $f(n) = n^3$

# Your Turn!

## ❖ True or false?

- $4+3n$  is  $O(n)$
- $n+2\log n$  is  $O(\log n)$
- $\log n+2$  is  $O(1)$
- $n^{50}$  is  $O(1.1^n)$

## ❖ Notes:

- Do NOT ignore constants that are not multipliers:
  - $n^3$  is  $O(n^2)$  : **FALSE**
  - $3^n$  is  $O(2^n)$  : **FALSE**
- When in doubt, refer to the definition

# Reviewing the Big-O Rules

- ❖ Eliminate coefficients because we don't have units anyway
  - $3n^2$  versus  $5n^2$  doesn't mean anything when we cannot count operations very accurately
- ❖ 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

$O(1)$ <i>*(<math>O(k)</math> for any <math>k</math>)</i>	Constant
$O(\log \log n)$	
$O(\log n)$	Logarithmic
$O(\log^k n)$ <i>*(for any <math>k &gt; 1</math>)</i>	
$O(n)$	Linear
$O(n \log n)$	Loglinear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(n^k)$ <i>*(for any <math>k &gt; 1</math>)</i>	Polynomial
$O(k^n)$ <i>*(for any <math>k &gt; 1</math>)</i>	Exponential

Note: “exponential” does not mean “grows really fast”; it means “grows at rate proportional to  $k^n$  for some  $k > 1$ ”

# Lecture Outline

- ❖ Algorithm Analysis
  - Review: Big-O, Formally
  - **Big-Omega and Big-Theta**
  - Proof example
  - Amortized bounds
  - Where We've Come

*[pollev.com/332summer](https://pollev.com/332summer) :: [tinyurl.com/332-06-24A](https://tinyurl.com/332-06-24A)*

# Big-O: Intuition

- ❖ Big-O can be thought of as something like “less-than or equals”

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

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

$$g(n) \leq c f(n) \quad \text{for all } n \geq n_0$$

# 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

$$g(n) \geq c f(n) \quad \text{for all } n \geq n_0$$

# 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

$$c_1 f(n) \leq g(n) \leq c_2 f(n) \quad \text{for all } n \geq n_0$$

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

## In Other Words ...

- ❖ **Upper bound:**  $O( f(n) )$  is the set of all functions asymptotically *less than or equal to*  $f(n)$ 
  - $g(n)$  is in  $O( f(n) )$  if there exist constants  $c$  and  $n_0$  such that
$$g(n) \leq c f(n) \text{ for all } n \geq n_0$$
- ❖ **Lower bound:**  $\Omega( f(n) )$  is the set of all functions asymptotically *greater than or equal to*  $f(n)$ 
  - $g(n)$  is in  $\Omega( f(n) )$  if there exist constants  $c$  and  $n_0$  such that
$$g(n) \geq c f(n) \text{ for all } n \geq n_0$$
- ❖ **Tight bound:**  $\theta( f(n) )$  is the set of all functions asymptotically *equal to*  $f(n)$ 
  - Intersection of  $O( f(n) )$  and  $\Omega( f(n) )$  (can use *different*  $c$  values)

# 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  $\theta$  **bound** to the **worst-case** running **time** of an **algorithm**
- ❖ Reminder that Case Analysis  $\neq$  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)

# What We are Analyzing: Examples

- ❖ 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
  - Less common (but very good to know): 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

# Lecture Outline

- ❖ Algorithm Analysis
  - Review: Big-O, Formally
  - Big-Omega and Big-Theta
  - **Proof example**
  - Amortized bounds
  - Where We've Come

*[pollev.com/332summer](https://pollev.com/332summer) :: [tinyurl.com/332-06-24A](https://tinyurl.com/332-06-24A)*

# The Proof is in the Practice

- ❖ Let  $g(n) = 4n^2 + 3n + 4$ , and  $f(n) = n^2$ , show  $g(n)$  is in  $O(f(n))$

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

$$g(n) \leq c f(n) \quad \text{for all } n \geq n_0$$

# No Backwards Reasoning Please ☹️

- ❖ Let  $g(n) = 4n^2 + 3n + 4$ , and  $f(n) = n^2$ , show  $g(n)$  is in  $O(f(n))$

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

$$g(n) \leq c f(n) \quad \text{for all } n \geq n_0$$

# Lecture Outline

- ❖ Algorithm Analysis
  - Review: Big-O, Formally
  - Big-Omega and Big-Theta
  - Proof example
  - **Amortized bounds**
  - Where We've Come

*[pollev.com/332summer](https://pollev.com/332summer) :: [tinyurl.com/332-06-24A](https://tinyurl.com/332-06-24A)*

# Linear Search: Best vs Worst Case

Find an integer in a *sorted* array

2	3	5	16	37	50	73	75	126
---	---	---	----	----	----	----	----	-----

```
// requires array is sorted
// returns whether k is in array
boolean find(int[] arr, int k) {
    for (int i=0; i < arr.length; ++i)
        if(arr[i] == k)
            return true;
    return false;
}
```

Best case: **find(2)**

Worst case: **find(126)**

# 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 are punting on one case: **Average-case complexity**
  - Sometimes: relies on distribution of inputs
    - Eg, binary heap’s  $O(1)$  insert (we will get to this)
    - 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**

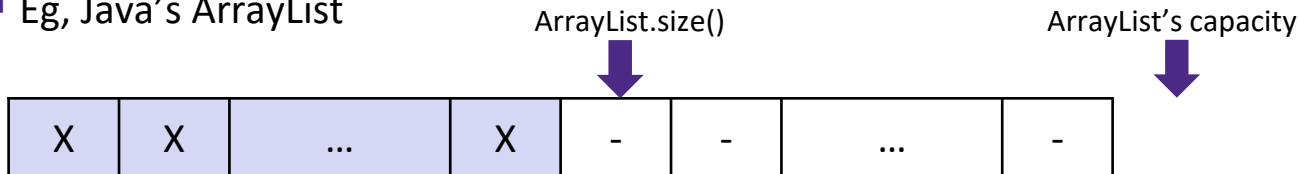
# Amortized Analyses = Multiple Executions

Single Execution	Multiple Executions
Worst Case	Amortized Worst Case
Best Case	Amortized Best Case
<i>Average Case</i>	<i>Amortized Average Case</i>

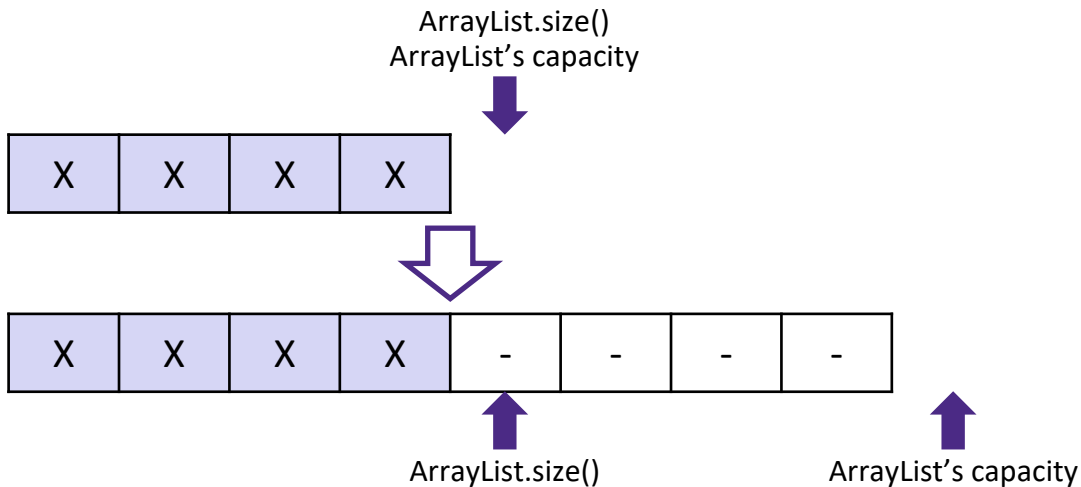
# Amortized Analysis: `ArrayList.add()`

- ❖ Consider adding an element to an array-backed structure

- Eg, Java's `ArrayList`



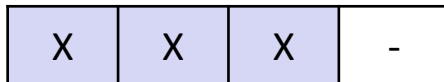
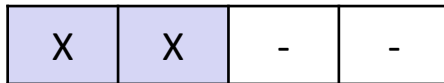
- ❖ When the underlying array fills, we allocate and copy contents



# ArrayList.add() Runtime (1 of 2)

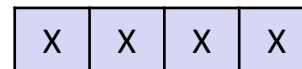
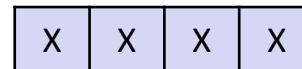
- ❖ We know that copying a single element and allocating arrays are both constant-time operations
  - Let's call their runtimes 'c' and 'd', respectively

*Most of the time*

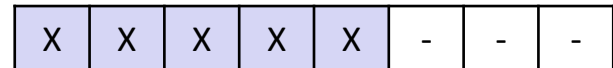
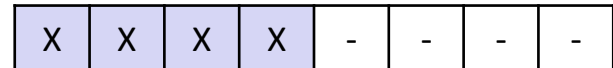
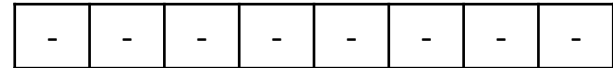


Runtime:

*Worst case*



Runtime:



# ArrayList.add() Runtime (2 of 2)

Single Execution	Multiple Executions
Worst Case:	Amortized Worst Case:
Best Case:	Amortized Best Case:

- ❖ Some applications *cannot tolerate* the “occasional  $O(n)$  behavior”
- ❖ Other applications *can tolerate* “occasional  $O(n)$  behavior” if we can show that it’s “not too bad” / “not too common”
- ❖ E.g. Self driving cars v.s. Adding songs to a playlist

## ArrayList.add(): Best-Case Aggregate Runtime



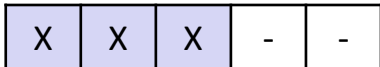
add(X)



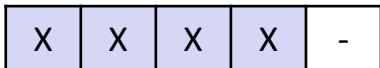
add(X)



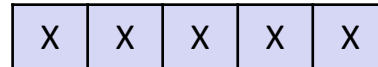
add(X)



add(X)

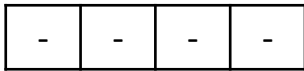


add(X)

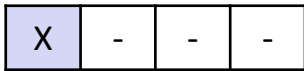


Best-case Aggregate Runtime:

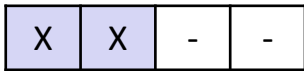
# ArrayList.add(): Worst-Case Aggregate Runtime



add(X)



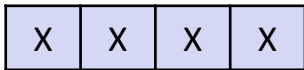
add(X)



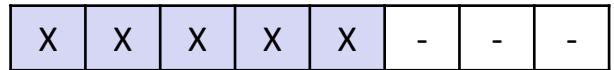
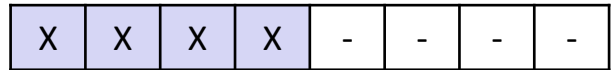
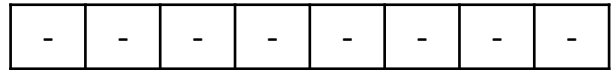
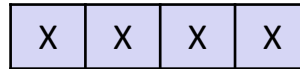
add(X)



add(X)



add(X)



Worst-case Aggregate Runtime:

# Amortized Analysis Intuition

- ❖ See Weiss, ch 11, for formal methods
- ❖ But the intuition is: if our client is willing to tolerate it, we will “smooth” the *aggregate cost of  $n$  operations* over  $n$  itself

Single Execution	Multiple Executions
Worst Case:	Amortized Worst Case:
Best Case:	Amortized Best Case:

- ❖ Note: we increased our array size by a factor of  $n$  (eg,  $2n$ ,  $3n$ , etc). What if we increased it by a constant factor (eg, 1, 100, 1000) instead?

# Lecture Outline

- ❖ Algorithm Analysis
  - Review: Big-O, Formally
  - Big-Omega and Big-Theta
  - Proof example
  - Amortized bounds
  - **Where We've Come**

*[pollev.com/332summer](https://pollev.com/332summer) :: [tinyurl.com/332-06-24A](https://tinyurl.com/332-06-24A)*

# Closing Thoughts

- ❖ Asymptotic analysis gives us a common “frame of reference” with which to compare algorithms
  - Most common comparisons are Big-O, Big-Omega, and Big-Theta
  - But also little-o and little-omega
- ❖ Case Analysis != Asymptotic Analysis
  - We combine asymptotic analysis and case analysis to compare the behavior of data structures and algorithms
- ❖ 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)

# Closing Thoughts

- ❖ Big-Oh can also use more than one variable
  - Example: can sum all elements of an  $n$ -by- $m$  matrix in  $O(nm)$
  - We will use this when we get to graphs!
- ❖ 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  $5 \cdot 10^{17} \approx 2^{58}$ ; 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)

# Closing Thoughts

- ❖ Asymptotic complexity for *specific implementations* can also be misleading ...
  - *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 on how to analyze an algorithm, mathematically, not the implementation
  - But timing has it's 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