Lecture 3: Algorithm Analysis 2

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Announcements

- EX00
  - Due **Monday**
- EX01
  - Out later today
  - Due **Monday, July 7**

Today

- Asymptotic Analysis Review
- Big-Oh Summary
- Cases vs Asymptotics
- Amortization

Recap Summary

```
// 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;
}
```

1. Count Code
   1. How to **count code constructs**
   2. Get a count in terms of n, usually in the **worst-case**

Worst Case: when integer not in sorted array

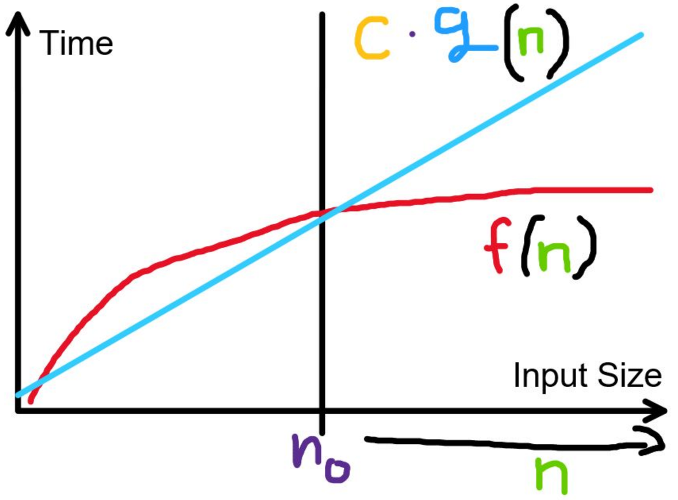
Count: $1 \text{ ops} + n \text{ iterations} \times 3\ ops = 1 + 3n$

1. Big-Oh - Group into set (**family**) of functions
   1. Asymptotic Behavior (what happens as $n \to$ big?)
   2. Informally: "Drop" coefficients, lower-order terms
   3. Formally: Find c and n_0

$1 + 3n \in \mathcal{O}(n)$

$1 + 3n$ is in $\mathcal{O}(n)$

Recap: Formally Big-Oh



Formal Definition:

Suppose $f: \mathbb{N} \to \mathbb{R}$, $g: \mathbb{N} \to \mathbb{R}$ are two functions, then,

$$f(n) \in \mathcal{O}(g(n)) \equiv \exists_{c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}} \forall_{n \in \mathbb{N} \geq n_0} f(n) \leq c \cdot g(n)$$

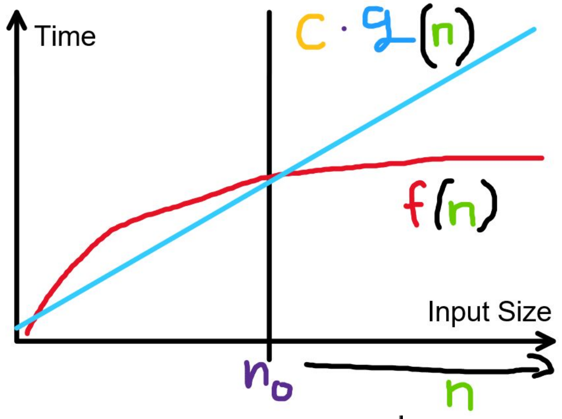
In English:

$f(n)$ is in $\mathcal{O}(g(n))$ iff there exists constants c and n_0 such that:

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

Note: c is a positive real number, n_0 and n are natural numbers

Recap: Why n_0? Why c?



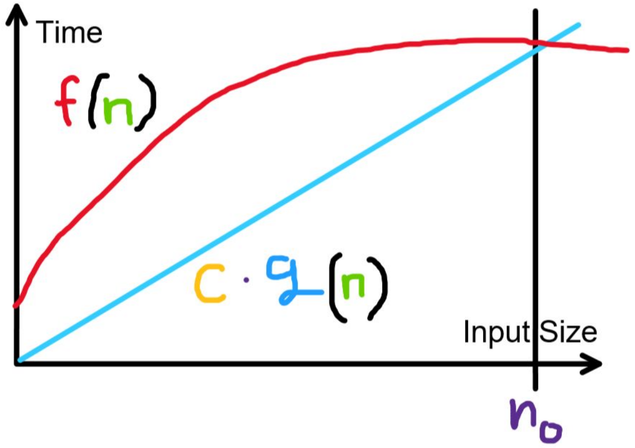
In English:

$f(n)$ is in $\mathcal{O}(g(n))$ iff there exists constants c and n_0 such that:

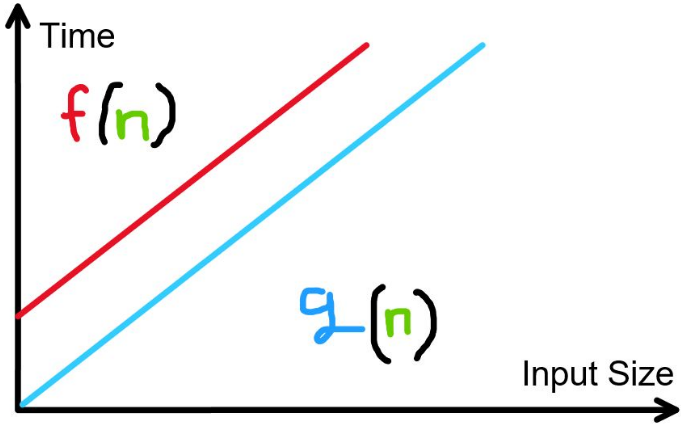
$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

Note: c is a positive real number, n_0 and n are natural numbers

Why n_0?



Why c?



Big-Oh: Example 1

In English:
$f(n)$ is in $\mathcal{O}(g(n))$ iff there exists constants c and n_0 such that:
$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
Note: c is a positive real number, n_0 and n are natural numbers

- How to show $f(n)$ is in $\mathcal{O}(g(n))$?
  - Pick a c large enough to "cover the constant factors"
  - Pick a n_0 large enough to "cover the lower-order terms"

Example: Let $f(n) = 3n + 4$ and $g(n) = n$, show $f(n) \in \mathcal{O}(g(n))$

Big-Oh: Example 1

- How to show $f(n)$ is in $\mathcal{O}(g(n))$?
  - Pick a c large enough to "cover the constant factors"
  - Pick a n_0 large enough to "cover the lower-order terms"

Example: Let $f(n) = 3n + 4$ and $g(n) = n$, show $f(n) \in \mathcal{O}(g(n))$

Observe the following:

$$3n \leq 3n, \quad \forall\ n \geq 4.$$
$$4 \leq n, \quad \forall\ n \geq 4.$$

Adding these inequalities, we see that,

$$3n + 4 \leq 4n, \ \forall\ n \geq 4.$$

Therefore, $f(n) \in O(g(n))$, with $c = 4$ and $n_0 = 4$.■

Big-Oh: Example 2

Let $f(n) = 4n^2 + 3n + 4$ and $g(n) = n^3$, show $f(n) \in \mathcal{O}(g(n))$.

Big-Oh: Example 2 (Solution)

Let $f(n) = 4n^2 + 3n + 4$ and $g(n) = n^3$, show $f(n) \in \mathcal{O}(g(n))$.

Observe the following:

$$4n^2 \leq 4n^3, \ \forall \ n \geq 1.$$
$$3n \leq 3n^3, \ \forall \ n \geq 1.$$
$$4 \leq 4n^3, \ \forall \ n \geq 1.$$

Adding these inequalities, we get:

$$4n^2 + 3n + 4 \leq 11n^3, \ \forall \ n \geq 1.$$

Therefore, $f(n) \in O(g(n))$, with $c = 11$ and $n_0 = 1$. ■

Big-Oh: Example 3

Let $f(n) = n$ and $g(n) = n - 1$. Show that $f(n) \in O(g(n))$.

Big-Oh: Example 3 (scratch work)

Let $f(n) = n$ and $g(n) = n - 1$. Show that $f(n) \in O(g(n))$.

Want to find c, n_0 so that $n \le c * (n - 1)$

Playing with algebra, we can rearrange this as:

$$n \le c * n - c$$

$$c \le (c - 1) * n$$

If we try plugging in $c = 2$, we get:

$$2 \le n$$

Which is true for $n \ge 2$. (So we should pick $c, n_0 = 2$)

Big-Oh: Example 3 (Proof)

Observe that

$$2 \leq n, \ \forall \ n \geq 2$$

We can rewrite this inequality as

$$2 \leq 2n - n, \ \forall \ n \geq 2$$

Rearranging, this is equivalent to

$$n \leq 2 * (n - 1), \ \forall \ n \geq 2$$

Therefore, $f(n) \in O(g(n))$, with $c = 2$ and $n_0 = 2$.

Big-Oh: Example Exercise

In English:

$f(n)$ is in $\mathcal{O}(g(n))$ iff

there exists constants c and n_0 such that:

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

Note: c is a positive real number,

n_0 and n are natural numbers

True or False?

1. $4 + 3n \in \mathcal{O}(n)$
2. $n + 2\log n \in \mathcal{O}(\log n)$
3. $\log n + 2 \in \mathcal{O}(1)$
4. $n^{50} \in \mathcal{O}(1.1^n)$

Note:

- Do NOT ignore constants that are not multipliers:
  - $n^3 \in \mathcal{O}(n^2)$ is $O(n^2)$: FALSE
  - $3^n \in \mathcal{O}(2^n)$: FALSE

Big-Oh: Example Exercise (Solution)

True or False?

1. $4 + 3n \in \mathcal{O}(n)$: TRUE
2. $n + 2\log n \in \mathcal{O}(\log n)$: FALSE
3. $\log n + 2 \in \mathcal{O}(1)$: FALSE
4. $n^{50} \in \mathcal{O}(1.1^n)$: TRUE

Note:

- Do NOT ignore constants that are not multipliers:
  - $n^3 \in \mathcal{O}(n^2)$ is $O(n^2)$: FALSE
  - $3^n \in \mathcal{O}(2^n)$: FALSE

Today

- Asymptotic Analysis Review
- Big-Oh Summary
- Cases vs Asymptotics
- Amortization

What can you drop?

- Coefficients
  - e.g., $3n^2$ vs $4n^2$ is same
- Low-order terms
  - e.g., $n^2 + n$ vs n^2 is same
- NOT constants that are not multipliers
  - e.g., n^2 vs n^3 is NOT same
  - e.g., 2^n vs 3^n is NOT same

(Intuitive way to understand the definition)

Big-Oh: Common Functions

• $\mathcal{O}(1)$	Constant	Fastest
• $\mathcal{O}(\log n)$	Logarithmic	
• $\mathcal{O}(n)$	Linear	
• $\mathcal{O}(n \log n)$	"$n \log n$" or Loglinear	
• $\mathcal{O}(n^2)$	Quadratic	
• $\mathcal{O}(n^3)$	Cubic	
• $\mathcal{O}(n^k)$	Polynomial	
• $\mathcal{O}(k^n)$	Exponential	Slowest

Usage note: "exponential" does not mean "grows really fast", it means "grows at rate proportional to k^n for some $k > 1$"

Beyond Big-Oh: More Asymptotic Notations

- Upper bound (Big-Oh):
  - Suppose $f: \mathbb{N} \to \mathbb{R}$, $g: \mathbb{N} \to \mathbb{R}$ are two functions, then,
  - $f(n) \in \mathcal{O}(g(n)) \equiv \exists_{c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}} \forall_{n \in \mathbb{N} \geq n_0} f(n) \leq c \cdot g(n)$
- Lower bound (Big-Omega):
  - Suppose $f: \mathbb{N} \to \mathbb{R}$, $g: \mathbb{N} \to \mathbb{R}$ are two functions, then,
  - $f(n) \in \Omega(g(n)) \equiv \exists_{c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}} \forall_{n \in \mathbb{N} \geq n_0} f(n) \geq c \cdot g(n)$
- Big-Theta bound:
  - Suppose $f: \mathbb{N} \to \mathbb{R}$, $g: \mathbb{N} \to \mathbb{R}$ are two functions, then,
  - $f(n) \in \Theta(g(n)) \equiv f(n) \in \mathcal{O}(g(n)) \wedge f(n) \in \Omega(g(n))$
    - (Can use different c)

Formally Big-Oh

In English:

$f(n)$ is in $\mathcal{O}(g(n))$ iff there exists constants c and n_0 such that:

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

Note: c is a positive real number, n_0 and n are natural numbers

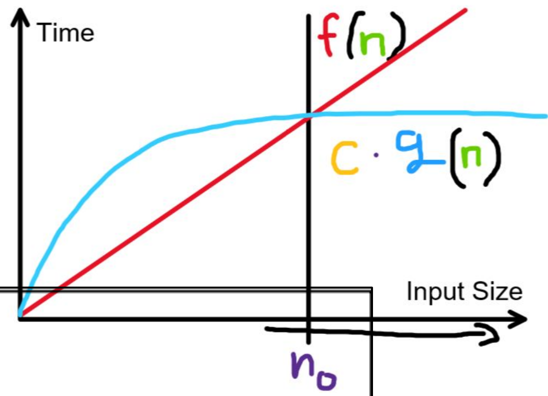
- How to show $f(n)$ is in $\mathcal{O}(g(n))$?
  - Pick a c large enough to "cover the constant factors"
  - Pick a n_0 large enough to "cover the lower-order terms"

Example: Let $f(n) = 3n + 4$ and $g(n) = n$, show $f(n) \in \mathcal{O}(g(n))$.

- Let $c = 4$ and $n_0 = 5$. Then, $3n + 4 \leq 4 \cdot n$ for all $n \geq 5$.

Notice: If $g(n)$ becomes bigger (e.g., $g(n) = n^5$ or $g(n) = 2^n$), same result

Now $c \cdot g(n)$ is at the bottom!



Formally Big-Omega (Ω)

In English:

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

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

Note: c is a positive real number, n_0 and n are natural numbers

Example: Let $f(n) = 3n + 4$ and $g(n) = n$, show $f(n) \in \Omega(g(n))$.

- Let $c = 1$ and $n_0 = 1$. Then, $3n + 4 \geq 1 \cdot n$ for all $n \geq 1$.

Notice: If $g(n)$ becomes smaller (e.g., $g(n) = \log n$ or $g(n) = 1$), same result

Formally Big-Theta (Θ)

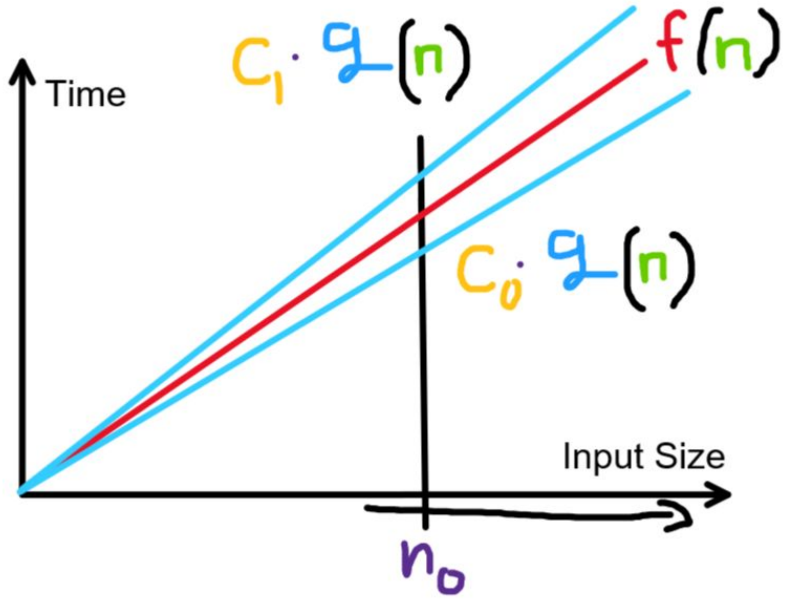
In English:

$f(n)$ is in $\Theta(g(n))$ such that:

$f(n)$ is in $\mathcal{O}(g(n))$ AND $f(n)$ is in $\Omega(g(n))$

i.e., $c_0 g(n) \leq f(n) \leq c_1 g(n)$

Note: c_0, c_1 is a positive real number, n_0 and n are natural numbers

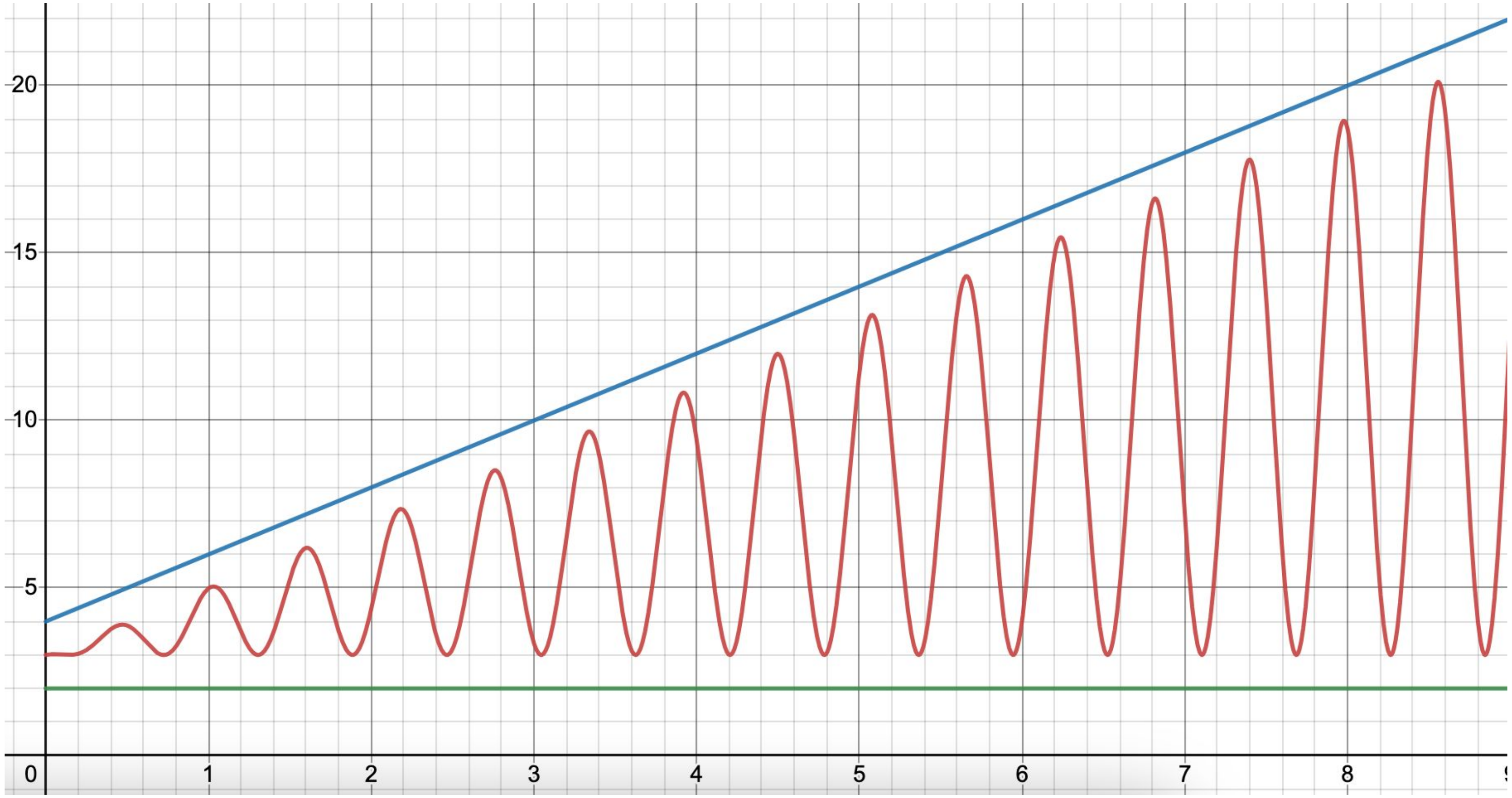


Note on "tight bound"

- It's technically true that $n + 2$ is $O(2^n)$
- But that's not a very good bound
- A better bound would be O(n)
  - It's the upper bound that's closest to the actual function
- We call the closest (i.e. lowest) asymptotic upper bound the "tight big-oh bound"
- Similarly the highest asymptotic lower bound is the "tight big-omega bound"
- On Exercises/Exams, we'll ask for "simplified tight bound"

When Big-Oh and Big-Omega differs

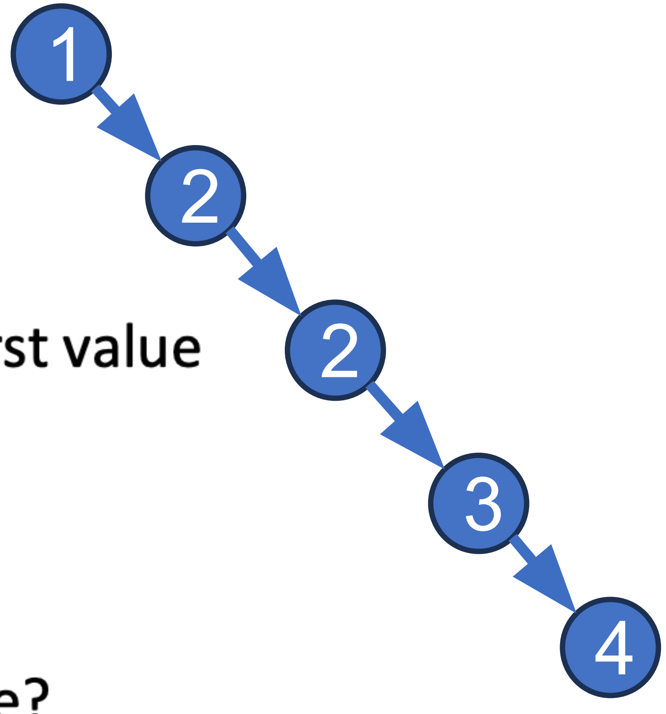
- Almost never :) https://www.desmos.com/calculator/zo6kikpgay



Today

- Asymptotic Analysis Review
- Big-Oh Summary
- Cases vs Asymptotic
- Amortization

Worst/Best-Case vs Asymptotic Analysis



- Completely Different! How?
- Worst/Best: all about case (*scenario*)
  - Linear Search Best Case: value we are looking for is the first value
  - BST Worst Case: skewed "linked list"-like structure
- Asymptotic Analysis:
  - Assuming that case (*scenario*), what happens as $n \rightarrow$ big?
  - Big-Oh: what's the worst growth this algorithm could have?
  - Big-Omega: what's the best growth this algorithm could have?

Analysis: Cases AND Asymptotic Analysis

Asymptotic Analysis

Cases		Lower	Upper
	Best		
	Worst		

Today

- Asymptotic Analysis Review
- Big-Oh Summary
- Cases vs Asymptotics
- Amortization

Amortization

Motivation: Worst-case too pessimistic (e.g., array resizing)

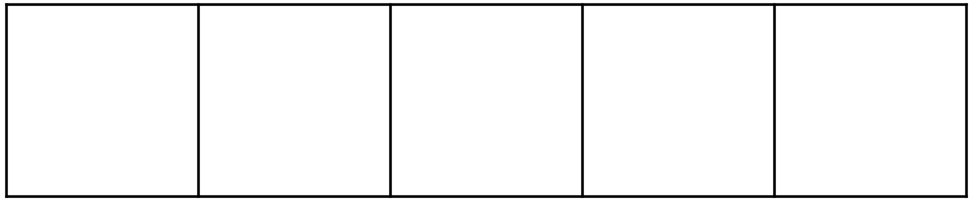
- max total # steps algorithm takes on M "most challenging" consecutive inputs of size n, divided by M (i.e., divide the max total # by M)
- **averages** the running times of operations in a **sequence** over that sequence

Sounds like the average case but is NOT average

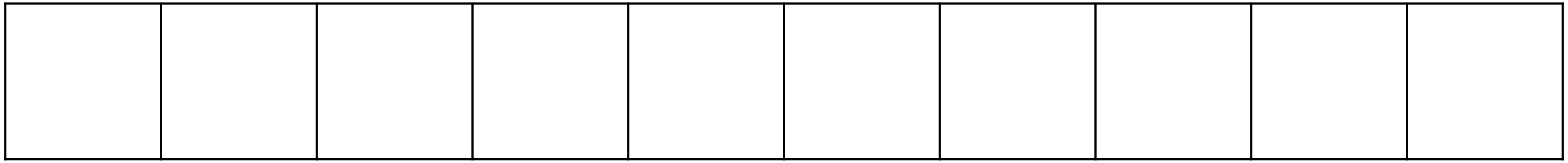
FYI: Mathematically incorrect but easy to illustrate

Amortization: Example

- insert() in ArrayList of capacity (not size) 5



1. 5x insert() - $\mathcal{O}(1)$ each
2. insert() - $\mathcal{O}(n)$ (because resize)



Total runtime? $\frac{(n-1)\mathcal{O}(1)+\mathcal{O}(n)}{n} = \mathcal{O}(1)$

Amortization: Why double size?

The most common strategy for increasing array size is **doubling**.
Why not just increase the size by 10 each time we fill up?

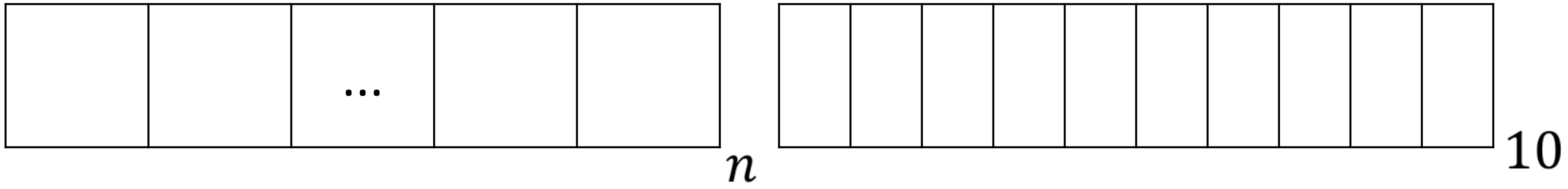
- Let's do amortized analysis
- Total cost of n insertions:

n [these are normal inserts] + 10 + 20 + 30 + … + n

$= O(n^2)$

- Divide by n

Amortized runtime = $O(n)$



Summary (wow that was a lot!)

1. Performance: Time vs Space (usually time)
   - Counting Code
2. Best, Worst, Amortized Case (usually worst or amortized)
3. Asymptotic Analysis (usually tight/Big-Theta)
   - Confusingly called Big-Oh

Timeline

- Asymptotic Analysis
  - Big-Oh (and Big-) Definition
- Big-Oh Summary
- Cases vs Asymptotics
- Amortization
- Priority Queue ADT
- Tree Stuff
- Binary Min-Heap Data Structure
  - Basics, Properties, Operations
  - Array Representation