

CSE 250

Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 06: Asymptotic Analysis

Announcements and Feedback

- PA1 out now
 - Testing due Sunday (no late submissions, no grace days)
 - Implementation due next Sunday

Analysis Checklist

1. Don't think in terms of wall-time, think in terms of "number of steps"
2. To give a useful solution, we should take "scale" into account
 - How does the runtime change as we change the size of the input?
3. Focus on "large" inputs
 - Rank functions based on how they behave at large scales
4. Decouple algorithm from infrastructure/implementation

Attempt #1: Wall-clock time?

- What is fast?
 - 10s? 100ms? 10ns?
 - ...it depends on the task
- Algorithm vs Implementation
 - Compare Grace Hopper's implementation to yours
- What machine are you running on?
 - Your old laptop? A lab machine? The newest, shiniest processor?
- What bottlenecks exist? CPU vs IO vs Memory vs Network...

Wall-clock time is not terribly useful... 4

Attempt #2: Growth Functions

Not a function in code...but a mathematical function:

$$T(n)$$

n: The “size” of the input

ie: number of users, rows, pixels, etc

$T(n)$: The number of “steps” taken for input of size n

ie: 20 steps per user, where $n = |\text{Users}|$, is $20 \times n$

Some Basic Assumptions:

Problem sizes are non-negative integers

$$n \in \{0, 1, 2, 3, \dots\} = \{0\} \cup \mathbb{Z}^+$$

We can't reverse time...(obviously)

$$T(n) > 0$$

Smaller problems aren't harder than bigger problems

$$n_1 < n_2 \Rightarrow T(n_1) \leq T(n_2)$$

Some Basic Assumptions:

Problem sizes are non-negative integers

$$n \in \{0, 1, 2, 3, \dots\} = \{0\} \cup \mathbb{Z}^+$$

We can't reverse time...(obviously)

$$T(n) > 0$$

Smaller problems aren't harder than bigger problems

$$n_1 < n_2 \Rightarrow T(n_1) \leq T(n_2)$$

$$T: \{0\} \cup \mathbb{Z}^+ \rightarrow \mathbb{R}^+$$

T is non-decreasing

First Problem...

We are still implementation dependent...

$$T_1(n) = 19n$$

$$T_2(n) = 20n$$

First Problem...

We are still implementation dependent...

$$T_1(n) = 19n$$

$$T_2(n) = 20n$$

Does 1 extra step per element really matter...?

Is this just an implementation detail?

First Problem...

We are still implementation dependent...

$$T_1(n) = 19n$$

$$T_2(n) = 20n$$

$$T_3(n) = 2n^2$$

T_1 and T_2 are much more “similar” to each other than they are to T_3

First Problem...

We are still implementation dependent...

$$T_1(n) = 19n$$

$$T_2(n) = 20n$$

$$T_3(n) = 2n^2$$

T_1 and T_2 are much more “similar” to each other than they are to T_3

How do we capture this idea formally?

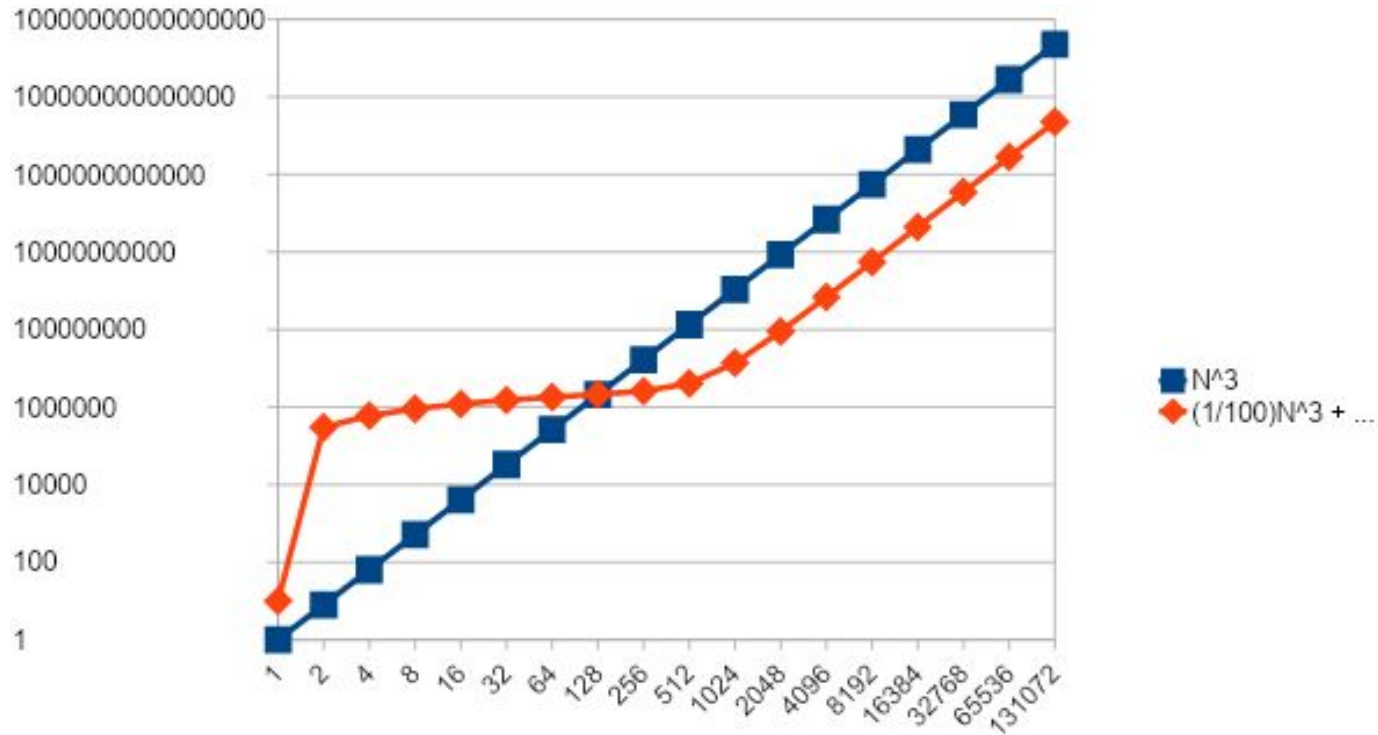
How Do We Capture Behavior at Scale?

Consider the following two functions:

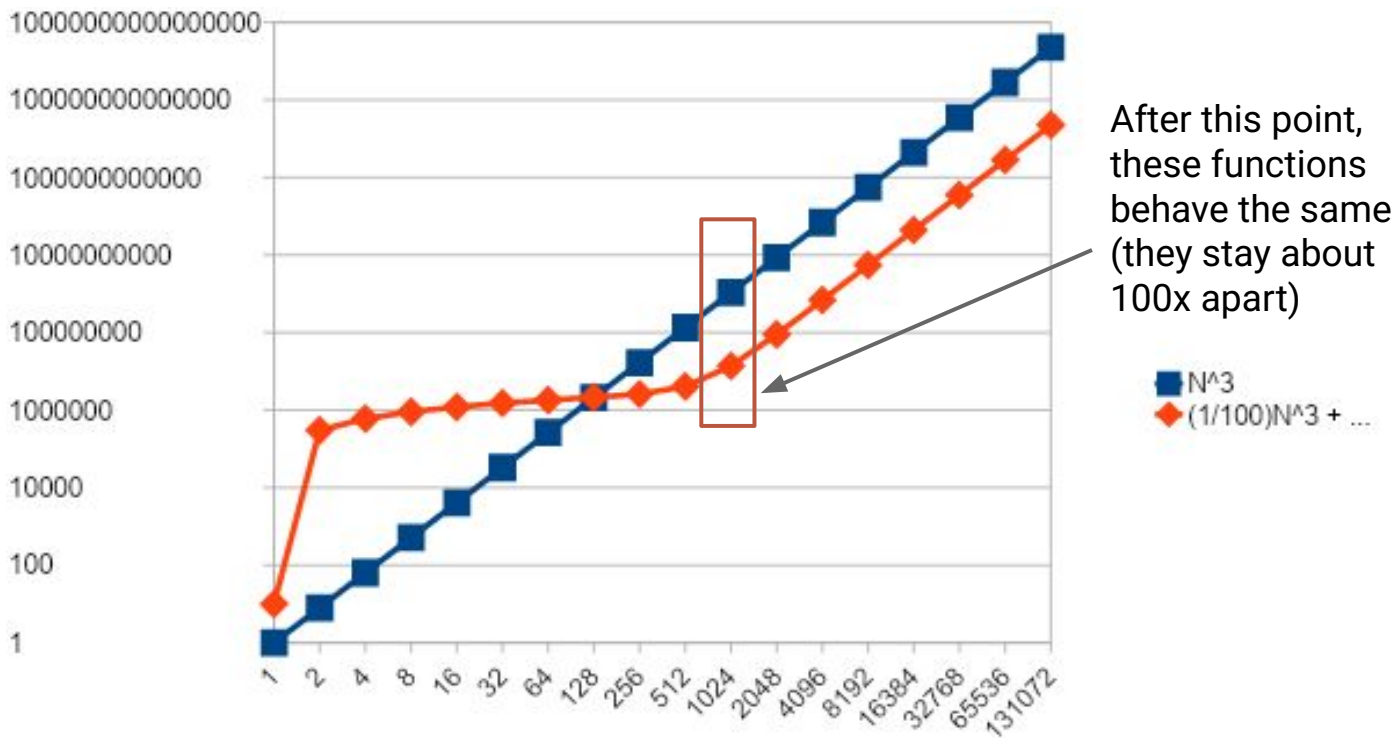
$$\frac{1}{100}n^3 + 10n + 10000000 \log(n)$$

$$n^3$$

How Do We Capture Behavior at Scale?



How Do We Capture Behavior at Scale?



Attempt #3: Asymptotic Analysis

We want to organize runtimes (growth functions) into different ***Complexity Classes***

Within the same complexity class, runtimes “behave the same”/“have the same shape” (at scale)

Getting More Formal

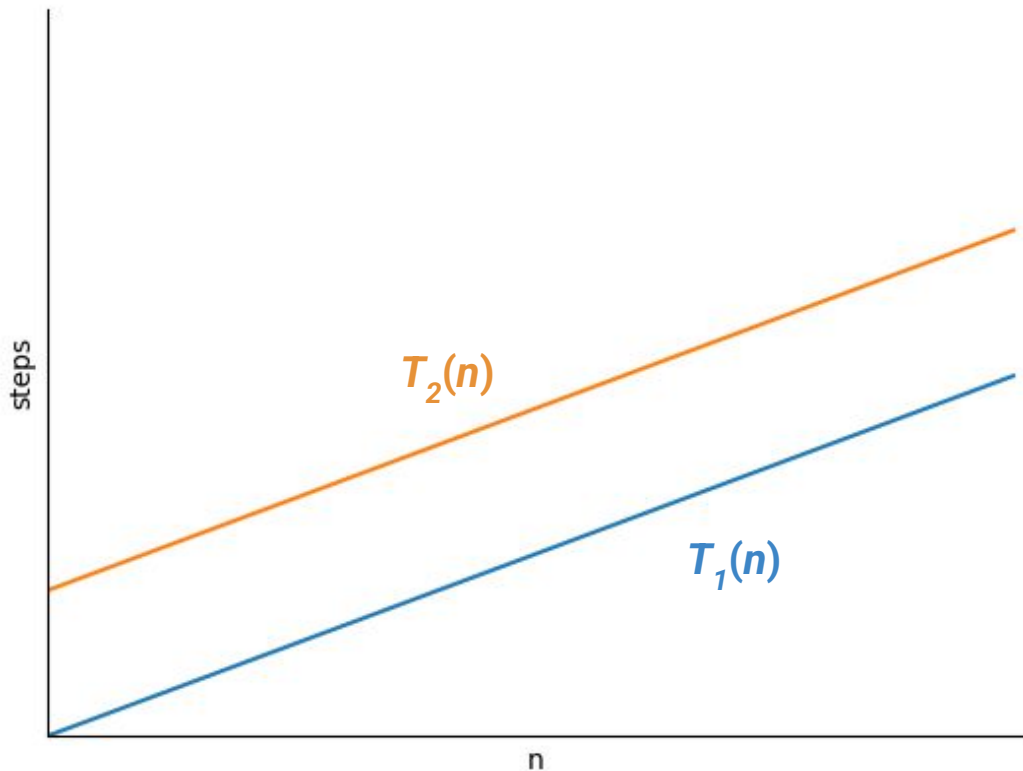
When do we consider two functions to have the same shape?

Additive Factors

Consider two growth functions:

$$T_1(n) = 3n$$

$$T_2(n) = 3n + 3$$

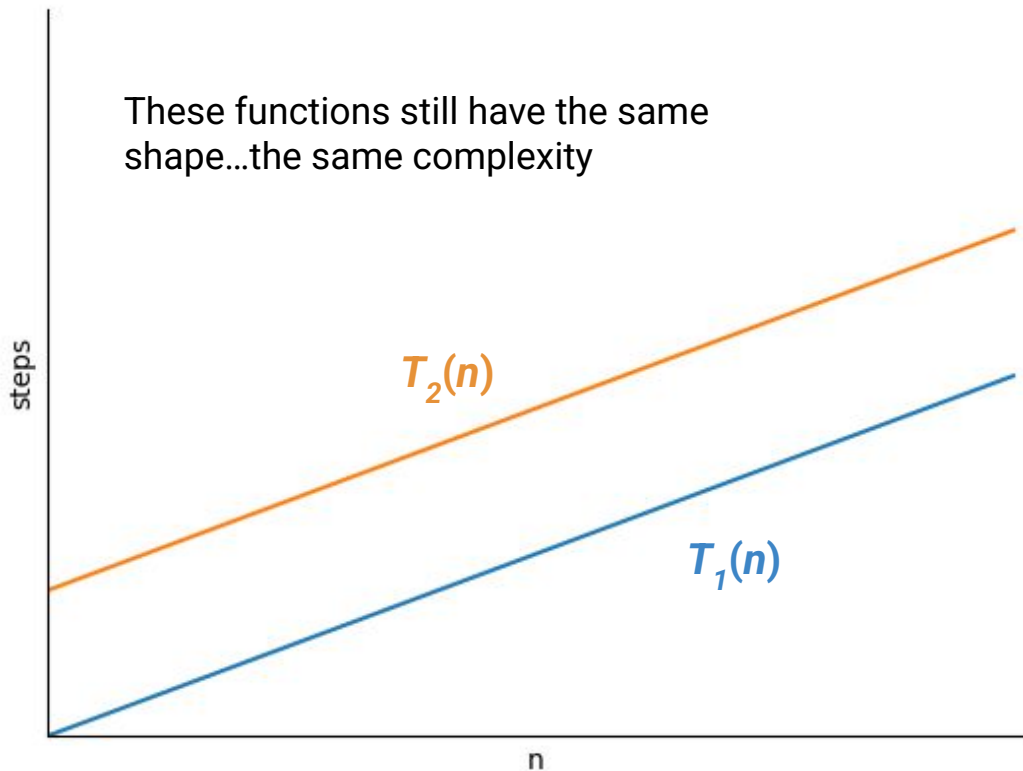


Additive Factors

Consider two growth functions:

$$T_1(n) = 3n$$

$$T_2(n) = 3n + 3$$

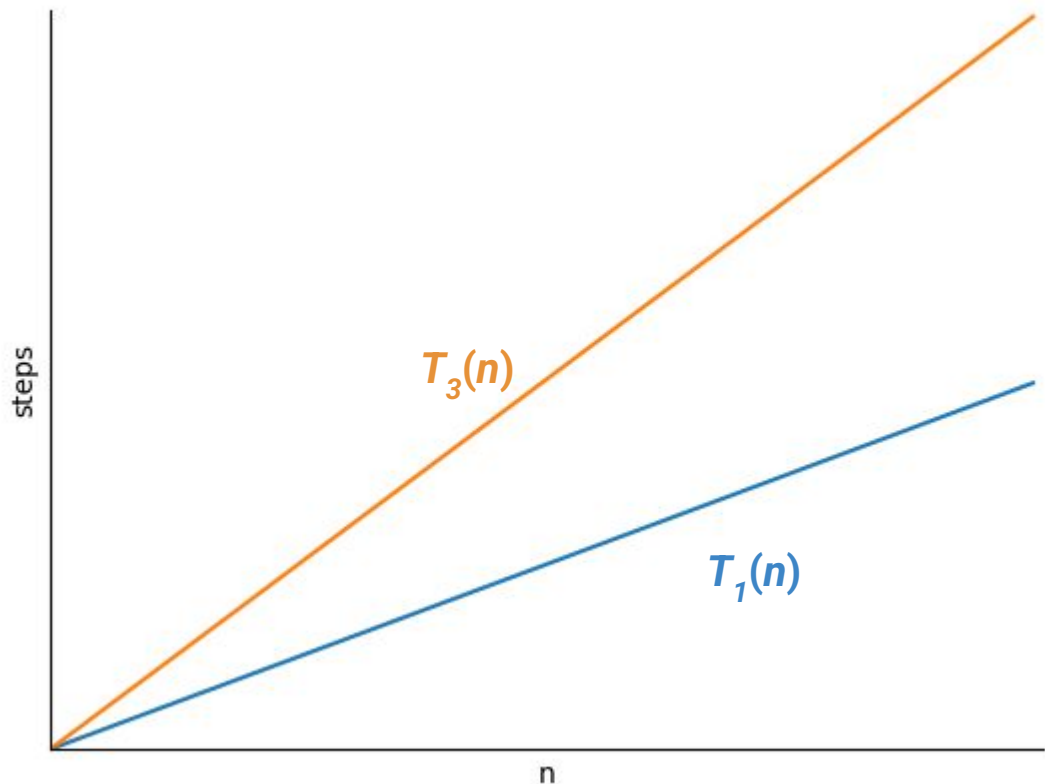


Multiplicative Factors

Consider two growth functions:

$$T_1(n) = 3n$$

$$T_3(n) = 6n$$

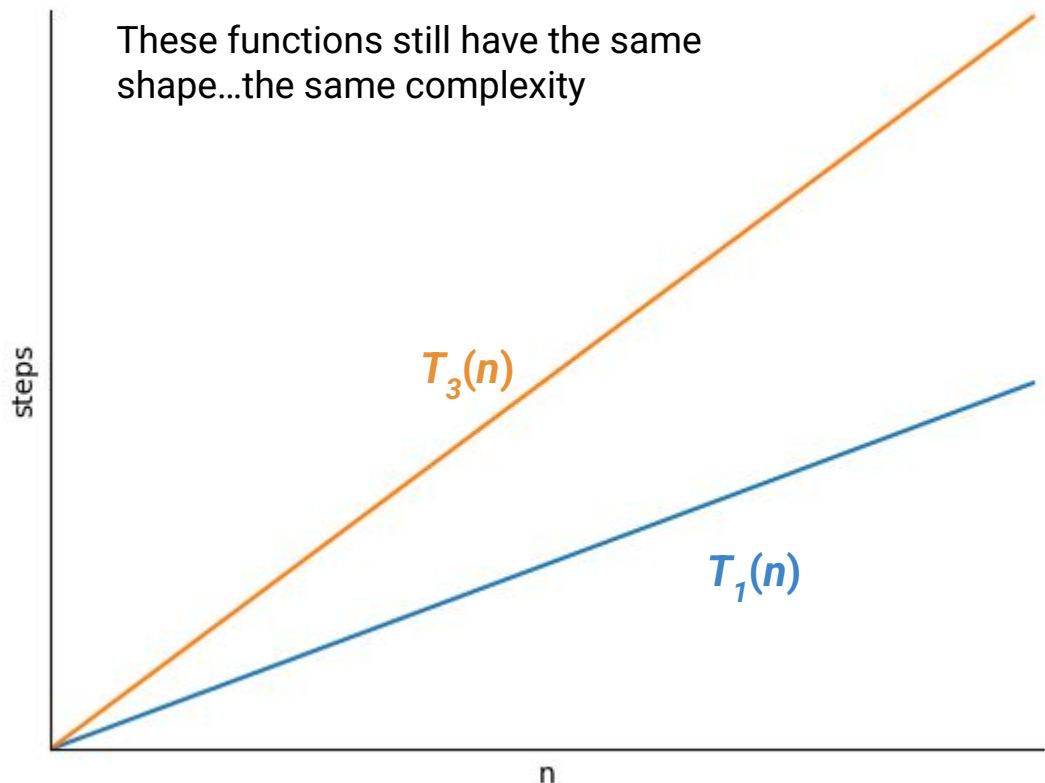


Multiplicative Factors

Consider two growth functions:

$$T_1(n) = 3n$$

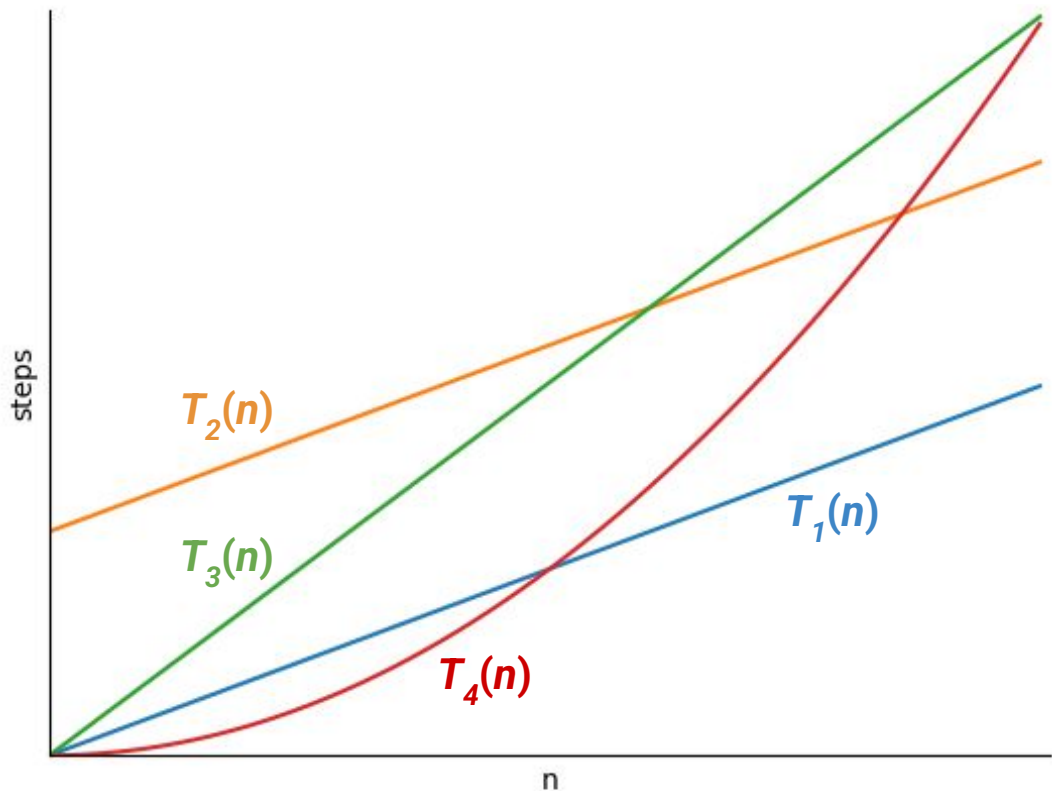
$$T_3(n) = 6n$$



A Counter Example

Now consider:

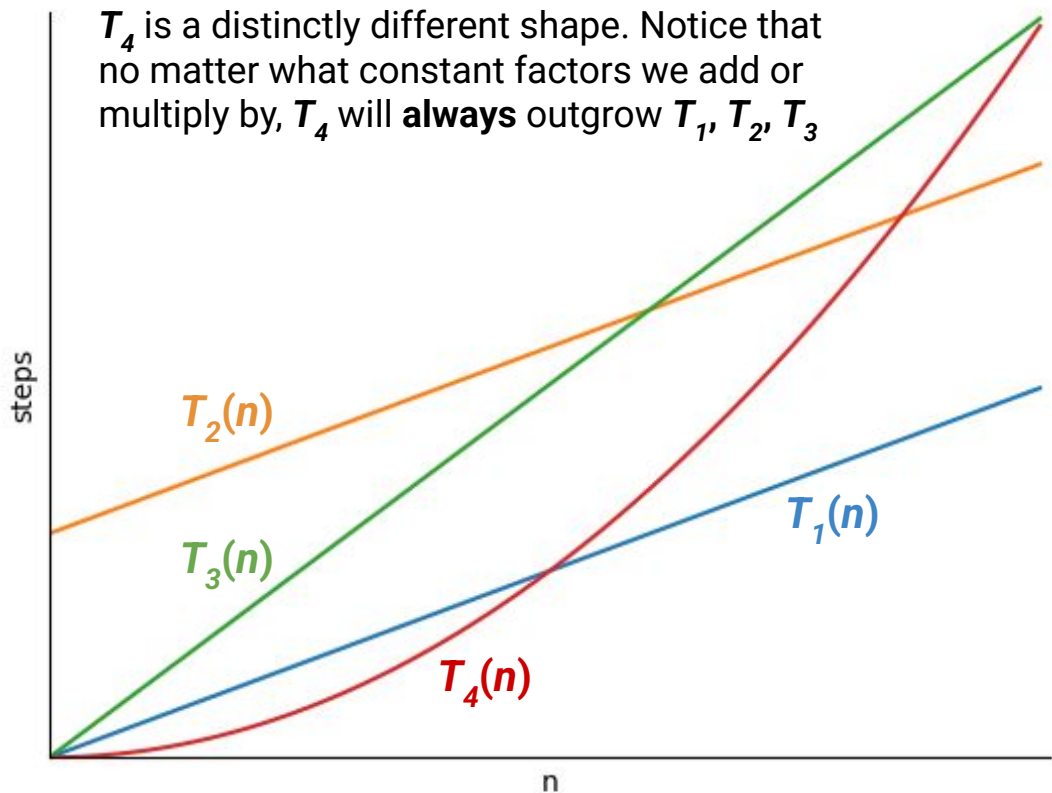
$$T_4(n) = n^2$$



A Counter Example

Now consider:

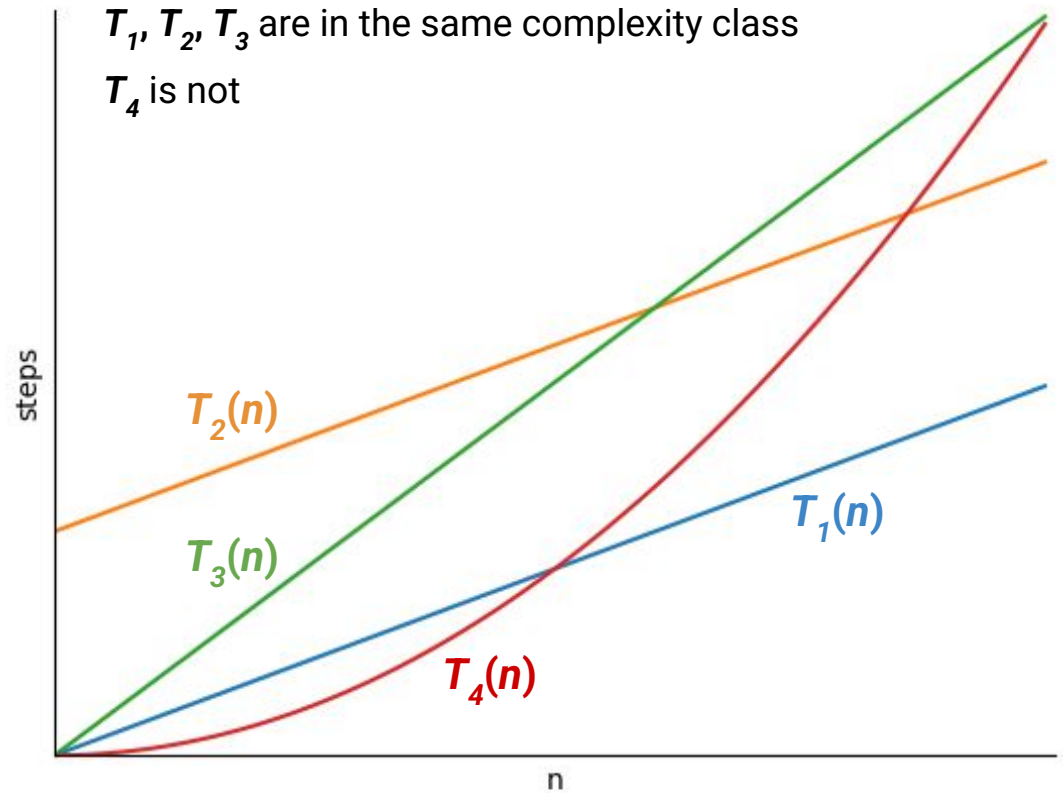
$$T_4(n) = n^2$$



A Counter Example

Now consider:

$$T_4(n) = n^2$$



Complexity (so far...)

If there are constants c_1 and c_2 such that:

$$T_1(n) = c_1 + c_2 T_2(n)$$

then we say T_1 and T_2 are in the same complexity class*

** not a complete definition...but we are getting there*

Back To Growth Functions

So what exactly counts as a step?

Back To Growth Functions

So what exactly counts as a step?

- An arithmetic operation
- Accessing a variable
- Printing to the screen
- etc

but...

Counting Steps

How many steps in each of these snippets?

1	x = 10;
---	---------

1	x = 10;
---	---------

2	y = 20;
---	---------

Counting Steps

How many steps in each of these snippets?

1	x = 10;
---	---------

$T_1(n) = 1$

1	x = 10;
---	---------

2	y = 20;
---	---------

Counting Steps

How many steps in each of these snippets?

1	x = 10;
---	---------

$$T_1(n) = 1$$

1	x = 10;
---	---------

2	y = 20;
---	---------

$$T_2(n) = 2$$

Counting Steps

How many steps in each of these snippets?

1	x = 10;
---	---------

$$T_1(n) = 1$$

1	x = 10;
---	---------

2	y = 20;
---	---------

$$T_2(n) = 2$$

$$T_2(n) = T_1(n) + 1$$

They are in the same complexity class...in 250 we treat them as the same 30

Counting Steps

A ***step*** therefore is ***any code*** that always has the same runtime

Notation - Big Theta

$\Theta(f(n))$ is the **set** of all functions in the same complexity class as f

Notation - Big Theta

$\Theta(f(n))$ is the **set** of all functions in the same complexity class as f

Example: $\Theta(3n + 4) = \{$
 $n,$
 $n - 6,$
 $15n,$
 \dots
 $\}$

Notation - Big Theta

$\Theta(f(n))$ is the **set** of all functions in the same complexity class as f

Example: $\Theta(3n + 4) = \{$

$n,$

$n - 6,$

$15n,$

...

$\}$

$g(n) \in \Theta(f(n))$ means g and f are in the same complexity class

Common Shorthand

$g(n) = \Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

Common Shorthand

$g(n) = \Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

$g(n)$ is in $\Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

Common Shorthand

$g(n) = \Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

$g(n)$ is in $\Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

Algorithm Foo is in $\Theta(f(n))$ is common shorthand for $T(n) \in \Theta(f(n))$ where $T(n)$ is the growth function describing the runtime of Foo

Common Shorthand

$g(n) = \Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

$g(n)$ is in $\Theta(f(n))$ is common shorthand for $g(n) \in \Theta(f(n))$

Algorithm Foo is in $\Theta(f(n))$ is common shorthand for $T(n) \in \Theta(f(n))$ where $T(n)$ is the growth function describing the runtime of Foo

Moving forward: $f(n)$, $g(n)$, $f_1(n)$, etc will be used to name any mathematical function that's a growth function

$T(n)$, $T_1(n)$, etc will be used for growth functions for specific algorithms

Complexity Class Names

$\Theta(1)$: Constant

$\Theta(\log(n))$: Logarithmic

$\Theta(n)$: Linear

$\Theta(n \log(n))$: Log-Linear

$\Theta(n^2)$: Quadratic

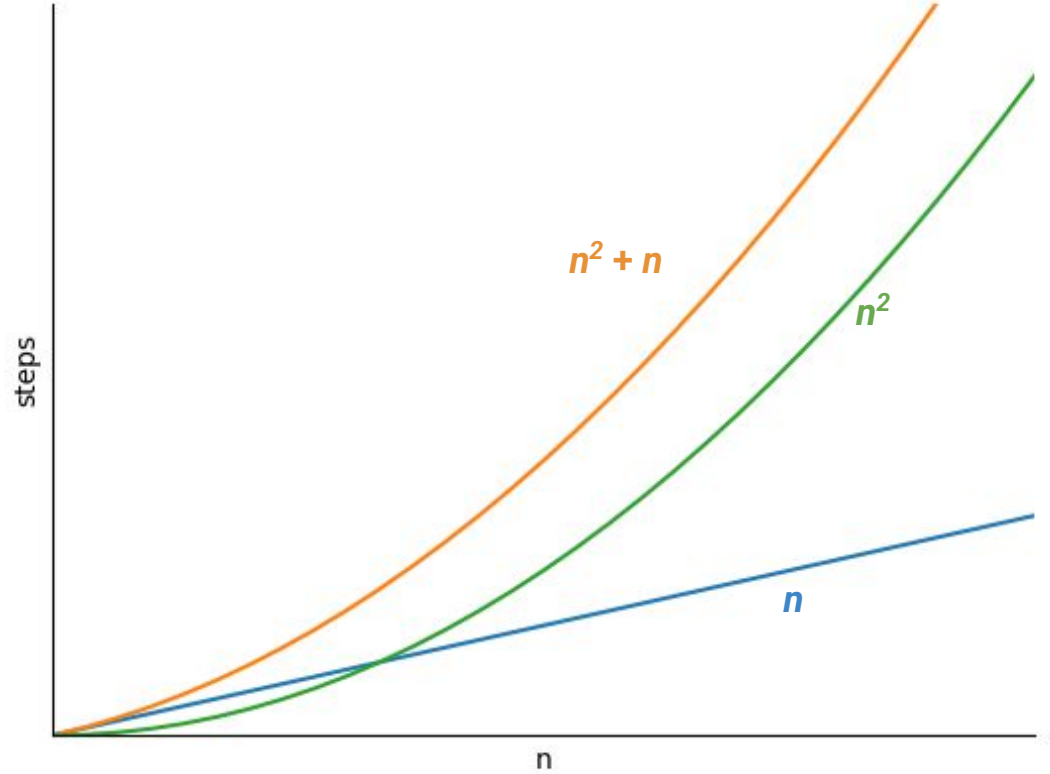
$\Theta(n^k)$: Polynomial

$\Theta(2^n)$: Exponential

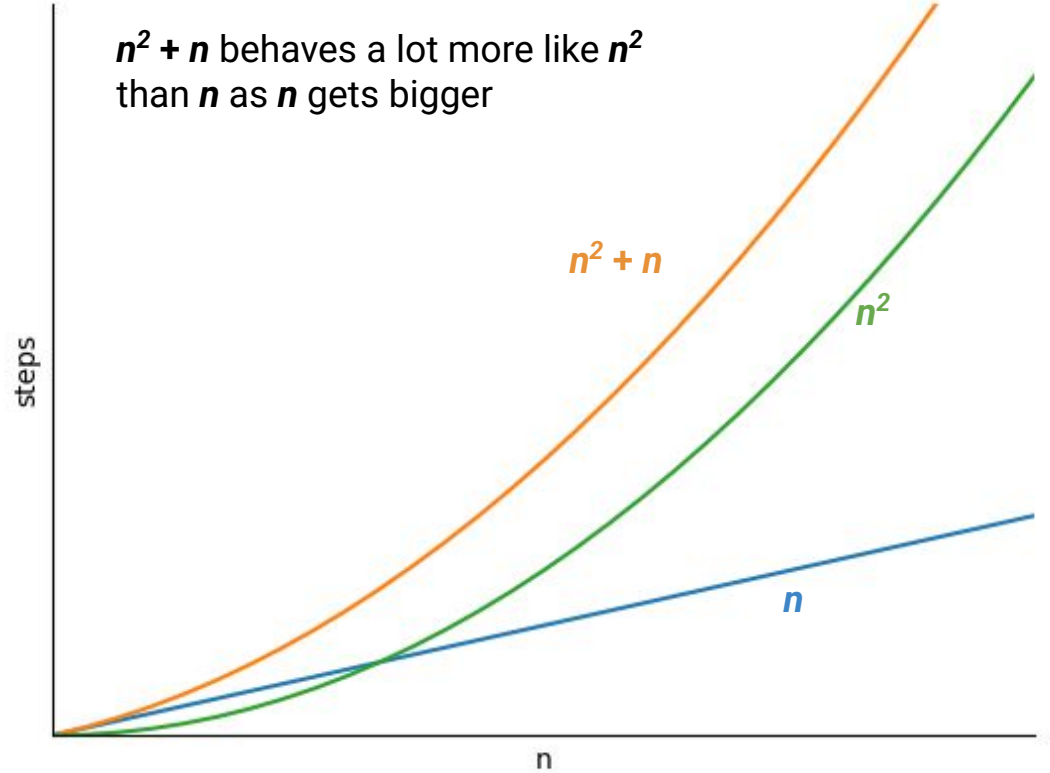
Combining Classes

What complexity class is $g(n) = n + n^2$ in?

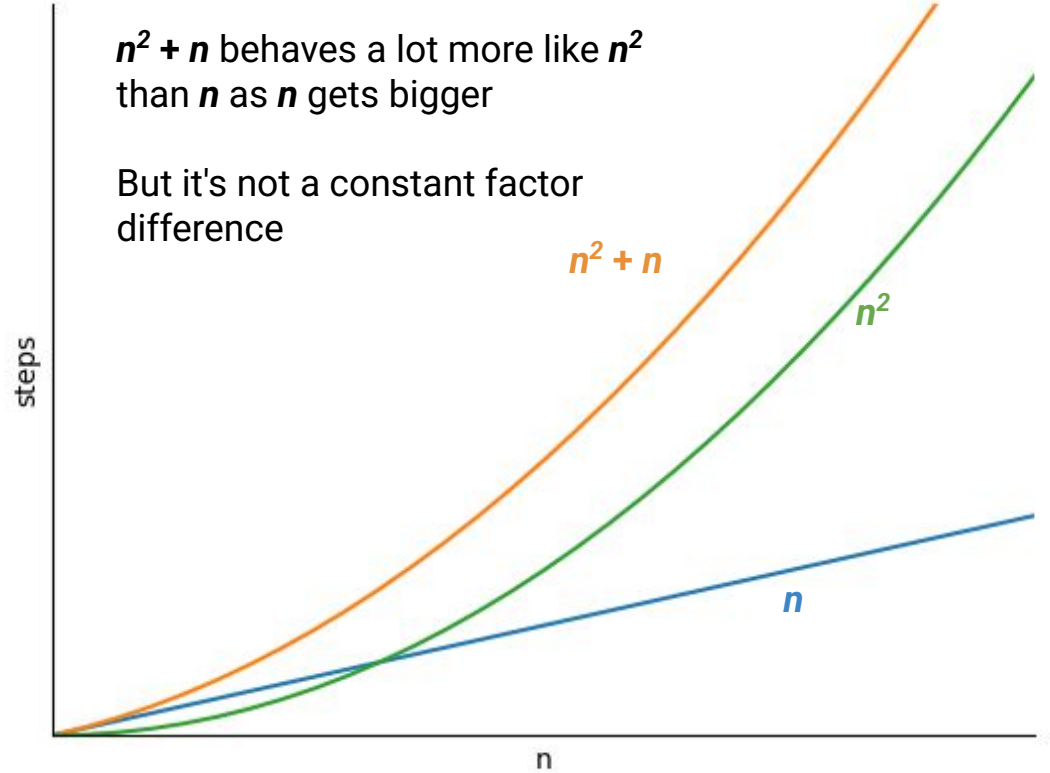
Combining Classes



Combining Classes



Combining Classes



Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

How does $n^2 + n$ relate to these two functions?

Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

$$1 \leq n$$

remember, we only care about problems
with non-negative input sizes

Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

$$1 \leq n$$

$$n \leq n^2$$

multiply both sides by n

Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

$$1 \leq n$$

$$n \leq n^2$$

$$n + n^2 \leq 2n^2$$

add n^2 to both sides

Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

$$0 \leq n \quad \text{obviously true}$$

Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

$$0 \leq n$$

$$n^2 \leq n + n^2 \quad \text{add } n^2 \text{ to both sides}$$

Combining Classes

Consider the fact that n^2 and $2n^2$ are in the same complexity class...

$$n^2 \leq n + n^2 \leq 2n^2$$

So $n^2 + n$ should probably be in $\Theta(n^2)$ too...

Complexity: A More Complete Definition

f and g are in the same complexity class iff:

g is bounded from above by something f -shaped

and

g is bounded from below by something f -shaped

Complexity: A More Complete Definition

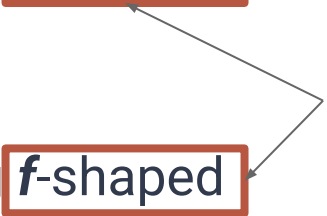
f and g are in the same complexity class iff:

g is bounded from above by something **f -shaped**

and

g is bounded from below by something **f -shaped**

f shifted or stretched by
a constant factor



Complexity: A More Complete Definition

f and g are in the same complexity class iff:

g is bounded from above by something f -shaped

and

g is bounded from below by something f -shaped

What do we mean by bounded from above/below?

Bounding from Above: Big O

$g(n)$ is bounded from above by $f(n)$ if:

There exists a constant $n_0 \geq 0$ and a constant $c > 0$ such that:

$$\text{For all } n \geq n_0, g(n) \leq c \cdot f(n)$$

Bounding from Above: Big O

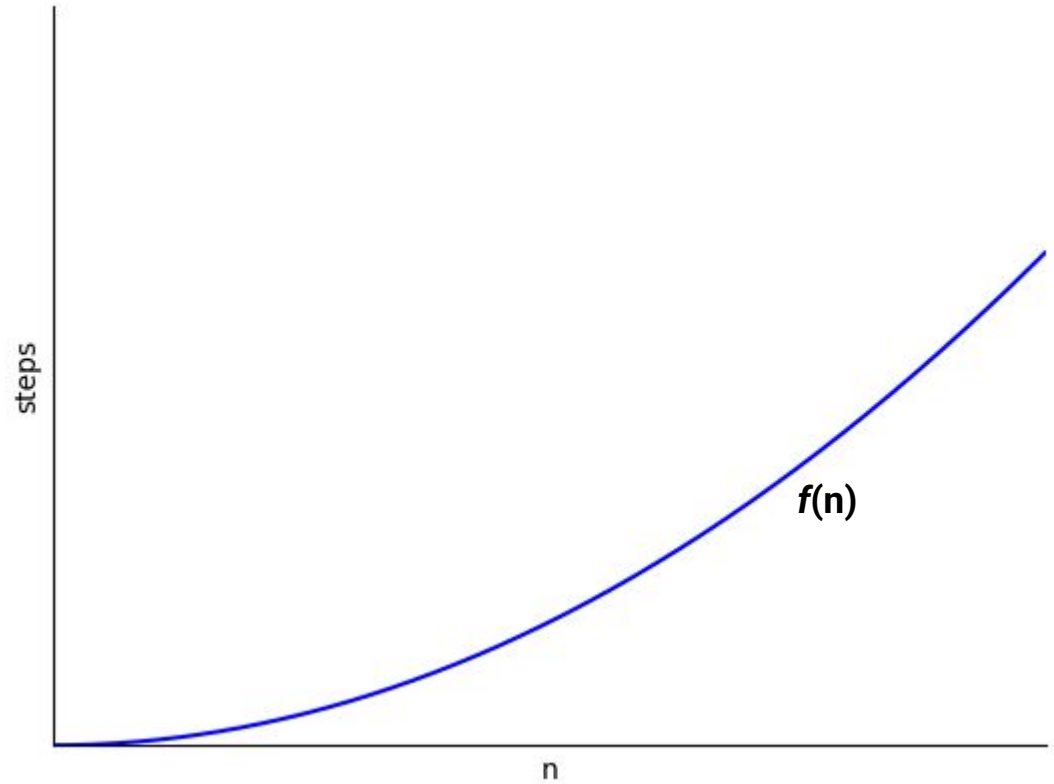
$g(n)$ is bounded from above by $f(n)$ if:

There exists a constant $n_0 \geq 0$ and a constant $c > 0$ such that:

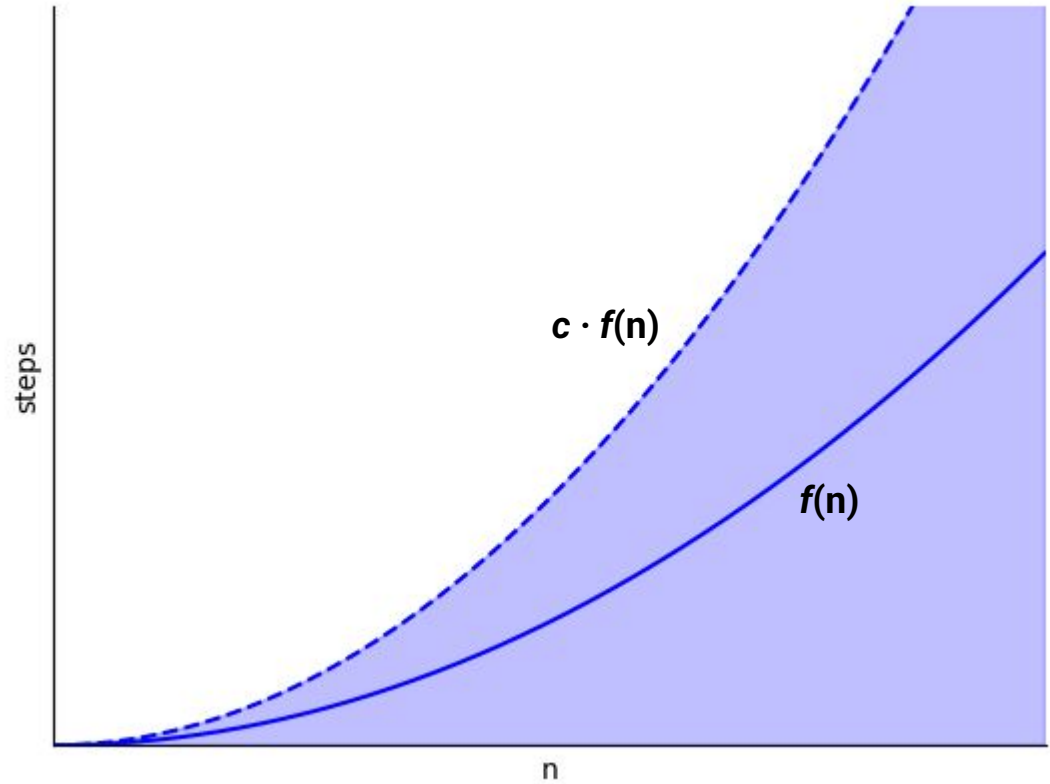
$$\text{For all } n \geq n_0, g(n) \leq c \cdot f(n)$$

In this case, we say that $g(n) \in O(f(n))$

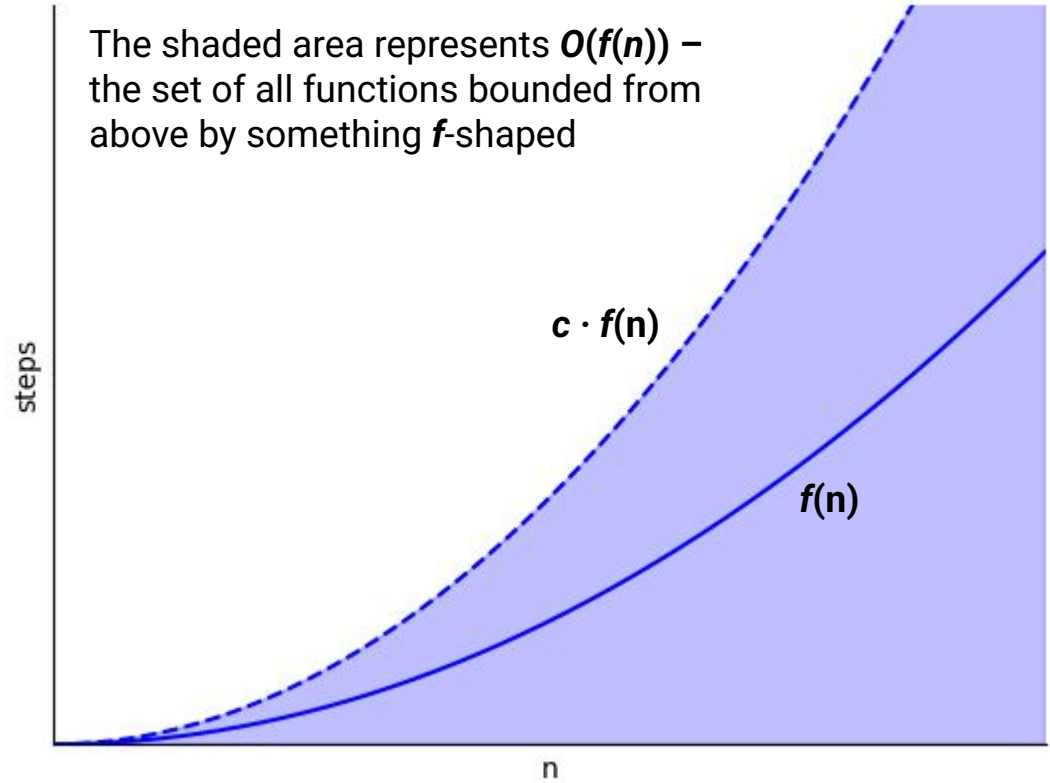
Bounded from Above: Big O



Bounded from Above: Big O



Bounded from Above: Big O



Bounding from Below: Big Omega

$g(n)$ is bounded from below by $f(n)$ if:

There exists a constant $n_0 \geq 0$ and a constant $c > 0$ such that:

$$\text{For all } n \geq n_0, g(n) \geq c \cdot f(n)$$

Bounding from Below: Big Omega

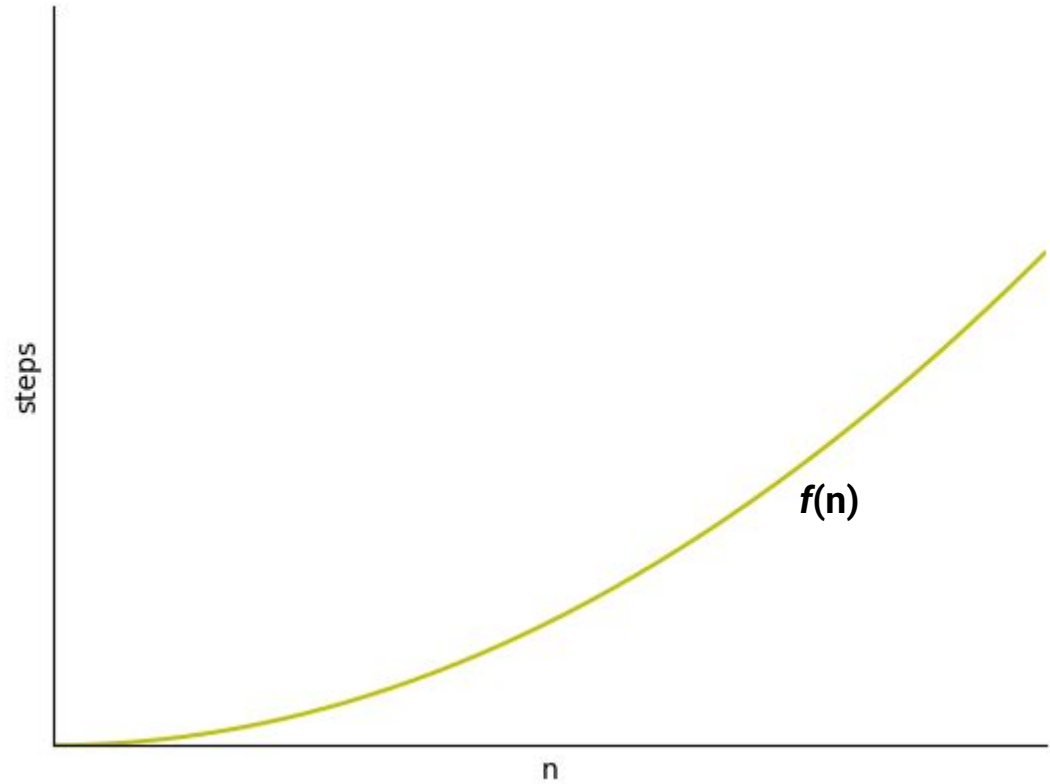
$g(n)$ is bounded from below by $f(n)$ if:

There exists a constant $n_0 \geq 0$ and a constant $c > 0$ such that:

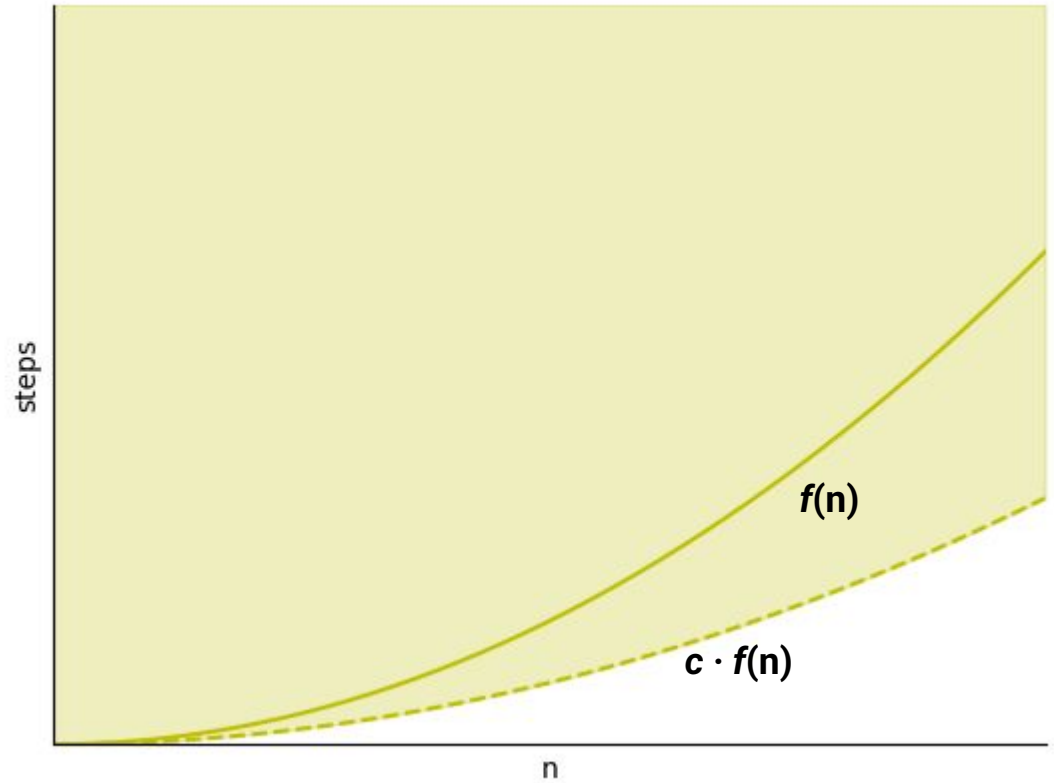
$$\text{For all } n \geq n_0, g(n) \geq c \cdot f(n)$$

In this case, we say that $g(n) \in \Omega(f(n))$

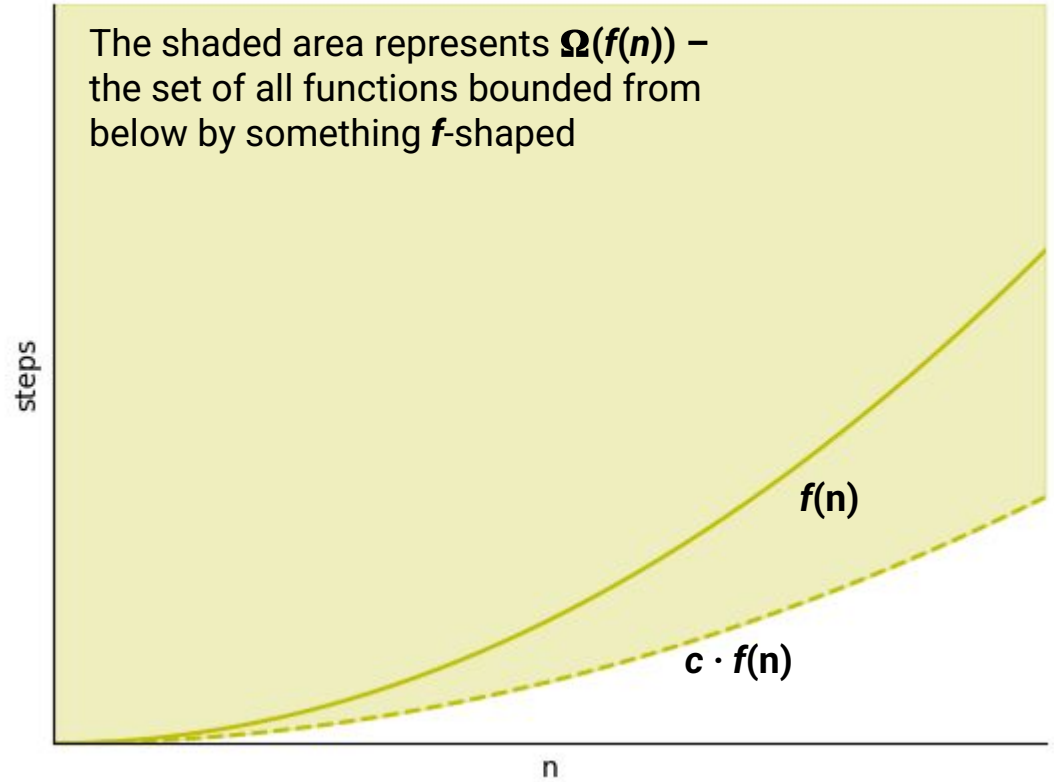
Bounded from Below: Big Ω



Bounded from Below: Big Ω



Bounded from Below: Big Ω



Complexity: A More Complete Definition

f and g are in the same complexity class iff:

g is bounded from above by something f -shaped

and

g is bounded from below by something f -shaped

Complexity: A More Complete Definition

$g(n) \in \Theta(f(n))$ iff:

g is bounded from above by something f -shaped

and

g is bounded from below by something f -shaped

Complexity: A More Complete Definition

$g(n) \in \Theta(f(n))$ iff:

$g(n) \in O(f(n))$

and

g is bounded from below by something f -shaped

Complexity: A More Complete Definition

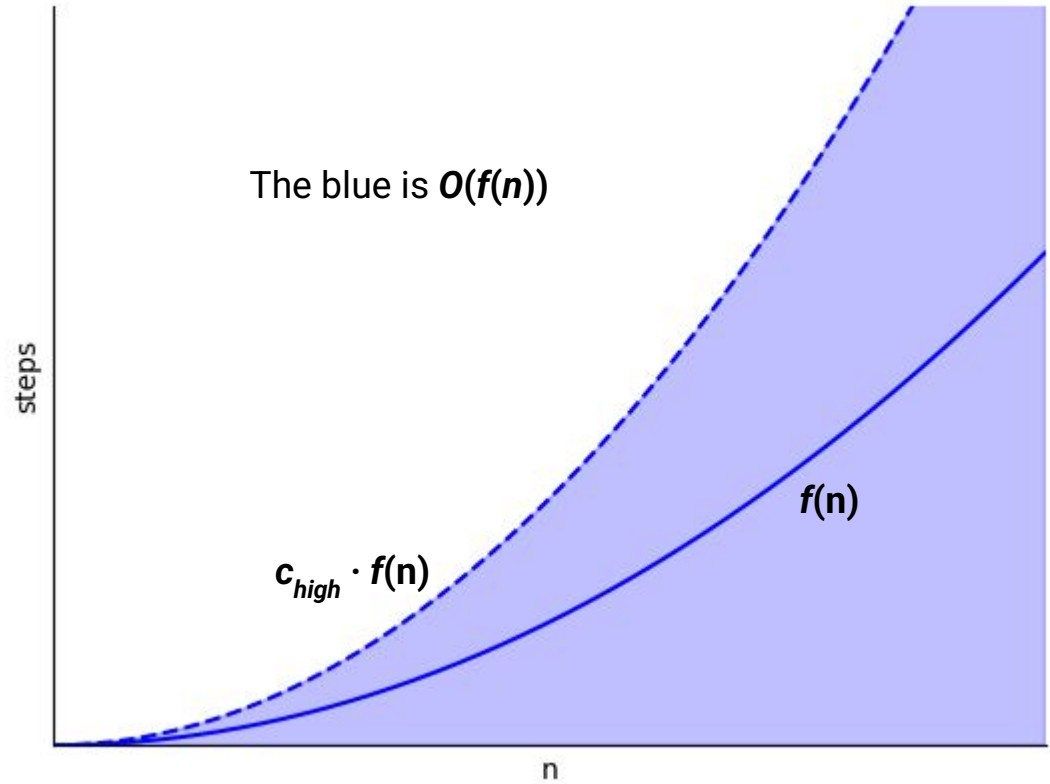
$g(n) \in \Theta(f(n))$ iff:

$$g(n) \in O(f(n))$$

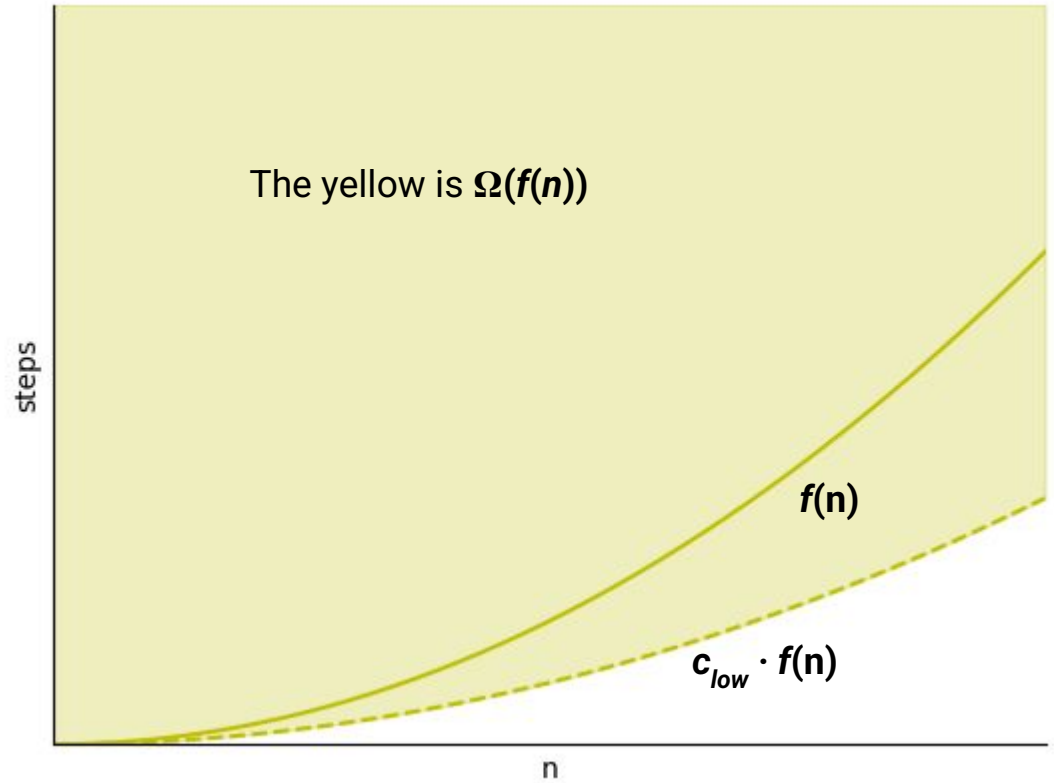
and

$$g(n) \in \Omega(f(n))$$

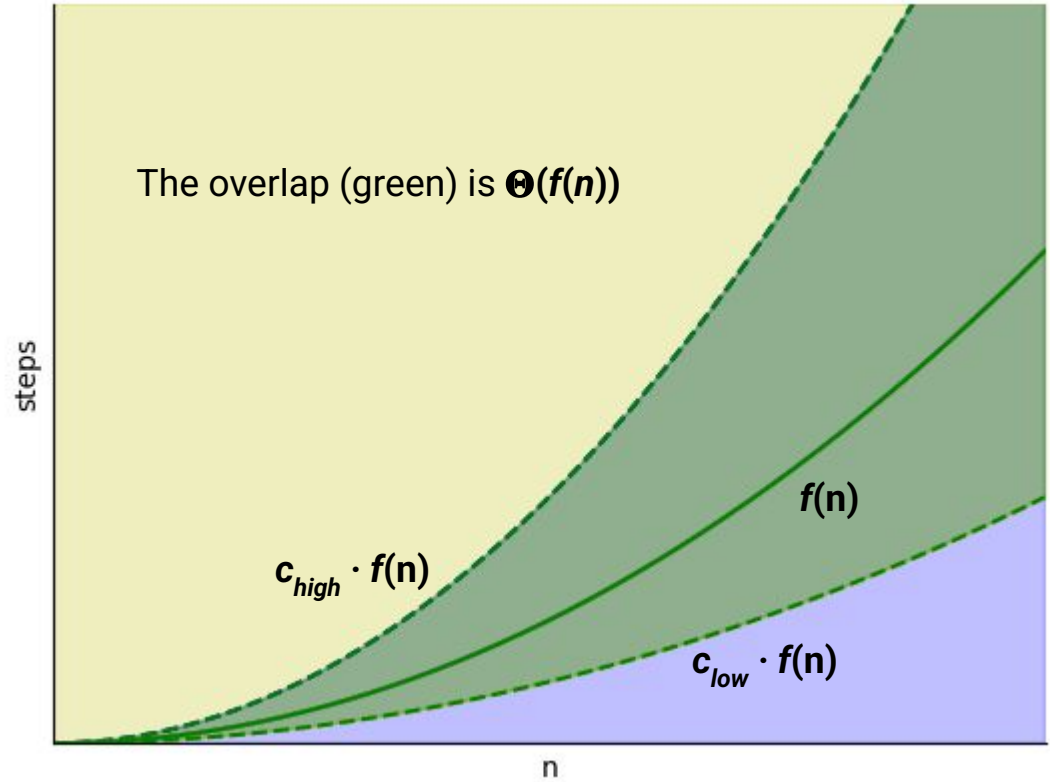
Complexity Class: Big Θ



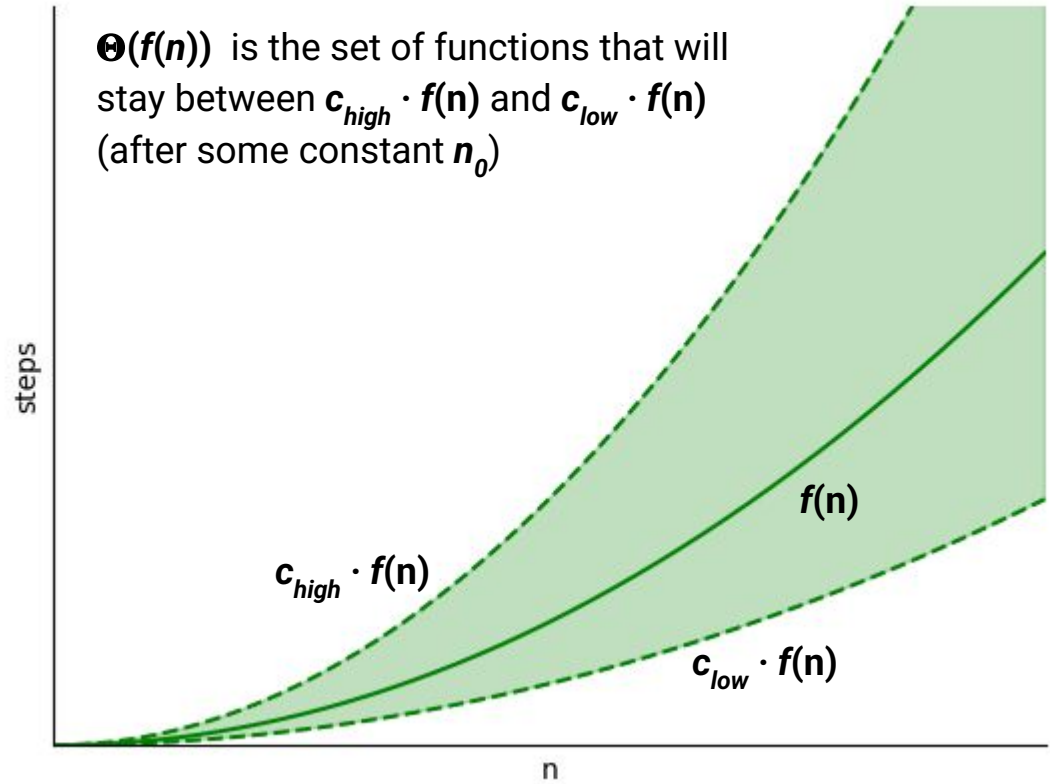
Complexity Class: Big Θ



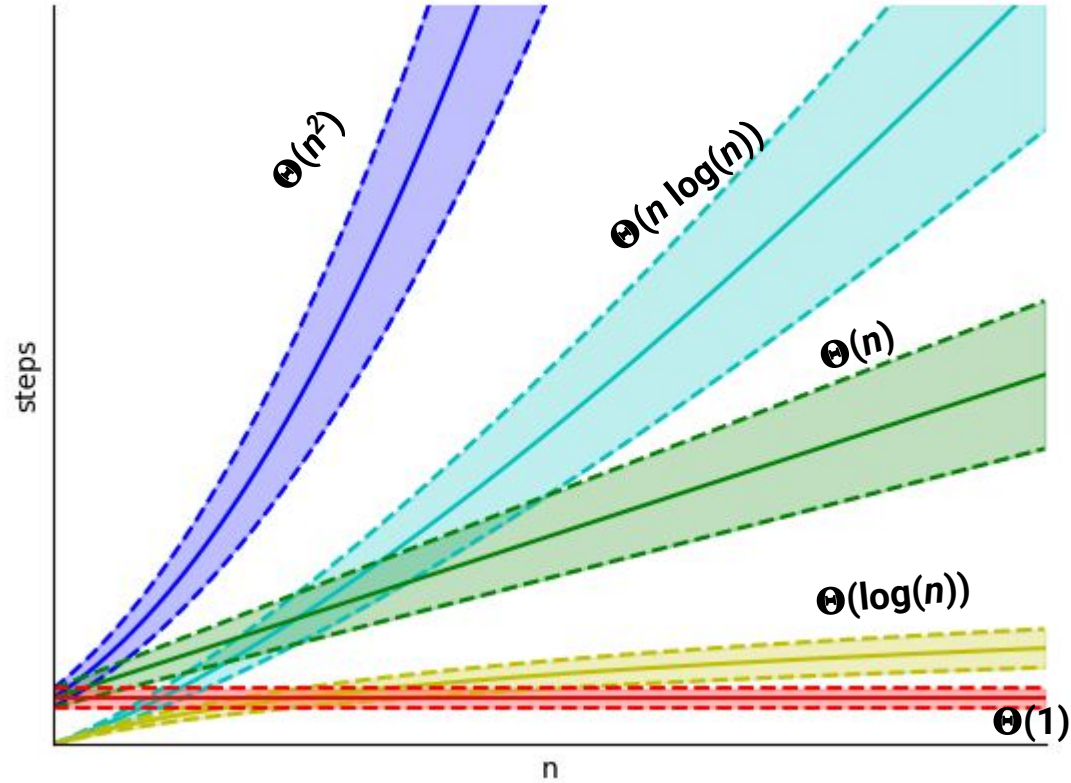
Complexity Class: Big Θ



Complexity Class: Big Θ



Complexity Class Ranking



$$\Theta(1) < \Theta(\log(n)) < \Theta(n) < \Theta(n \log(n)) < \Theta(n^2) < \Theta(n^3) < \Theta(2^n)$$

Rules of Thumb

$$\Theta(1) < \Theta(\log(n)) < \Theta(n) < \Theta(n \log(n)) < \Theta(n^2) < \Theta(n^3) < \Theta(2^n)$$

Rules of Thumb

$$O(1) \subset O(\log(n)) \subset O(n) \subset O(n \log(n)) \subset O(n^2) \subset O(n^3) \subset O(2^n)$$

$$\Omega(2^n) \subset \Omega(n^3) \subset \Omega(n^2) \subset \Omega(n \log(n)) \subset \Omega(n) \subset \Omega(\log(n)) \subset \Omega(1)$$

Rules of Thumb

If something is bounded from above by $\log(n)$, it's also bounded from above by n


$$O(1) \subset O(\log(n)) \subset O(n) \subset O(n \log(n)) \subset O(n^2) \subset O(n^3) \subset O(2^n)$$

$$\Omega(2^n) \subset \Omega(n^3) \subset \Omega(n^2) \subset \Omega(n \log(n)) \subset \Omega(n) \subset \Omega(\log(n)) \subset \Omega(1)$$

Rules of Thumb

$$O(1) \subset O(\log(n)) \subset O(n) \subset O(n \log(n)) \subset O(n^2) \subset O(n^3) \subset O(2^n)$$

$$\Omega(2^n) \subset \Omega(n^3) \subset \Omega(n^2) \subset \Omega(n \log(n)) \subset \Omega(n) \subset \Omega(\log(n)) \subset \Omega(1)$$



If something is bounded from below by n^2 , it's also bounded from below by n

Rules of Thumb

$O(f(n))$ (**Big-O**): The complexity class of $f(n)$ and every lesser class

$\Theta(f(n))$ (**Big- Θ**): The complexity class of $f(n)$

$\Omega(f(n))$ (**Big- Ω**): The complexity class of $f(n)$ and every greater class

