Big-Oh Notation

Review Session 1/12

In this class we will use…

- **Big-Oh notation!**
- Gives us a meaningful way to talk about the running time of an algorithm independent of programming language, computing platform, etc., without having to count all the operations.

Main idea:

Focus on how the runtime **scales** with n (the input size).

Some examples…

(Only pay attention to the largest function of n that appears.)

Number of operations	Asymptotic Running Time
$\frac{1}{10}e^n + 10n^2$	$O(e^n)$
$n^3 + 2n^2 + 7$	$O(n^3)$
$0.1\sqrt{n} - 10^9 n^{0.05}$	$O(\sqrt{n})$
$11\log(n) + 1$	$O(\log(n))$

We say this algorithm is "asymptotically faster" than the others.

Example Runtime

$$T(n) = 25n^2 + 5n + 7 \text{ ms}$$

The constant factor of 25 depends on the computing platform..

As n gets large, the lower-order terms don't really matter

$$= O(n^2)$$

pronounced "big-oh of …" or sometimes "oh of …"

Informal definition for O(…)

- Let $T(n)$, $g(n)$ be functions of positive integers.
  - Think of $T(n)$ as a runtime: positive and increasing in n.
- We say "$T(n)$ is $O\big(g(n)\big)$" if:

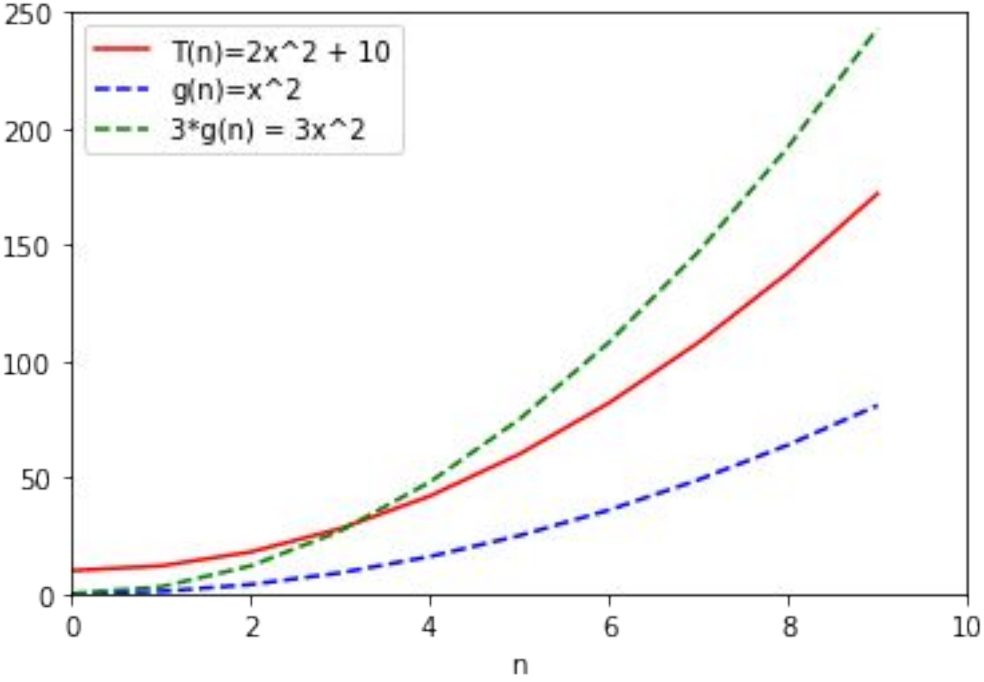
for large enough n,

$T(n)$ is at most some constant multiple of $g(n)$.

Here, "constant" means "some number that doesn't depend on n."

Example
$2n^2 + 10 = O(n^2)$

for large enough n,
$T(n)$ is at most some constant multiple of $g(n)$.



$3g(n) = 3n^2$

$T(n) = 2n^2 + 10$

$g(n) = n^2$

Formal definition of O(…)

- Let $T(n)$, $g(n)$ be functions of positive integers.
  - Think of $T(n)$ as a runtime: positive and increasing in n.

- Formally,

$$T(n) = O\big(g(n)\big)$$

"If and only if"

$$\Longleftrightarrow$$

"For all"

$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$

$$T(n) \leq c \cdot g(n)$$

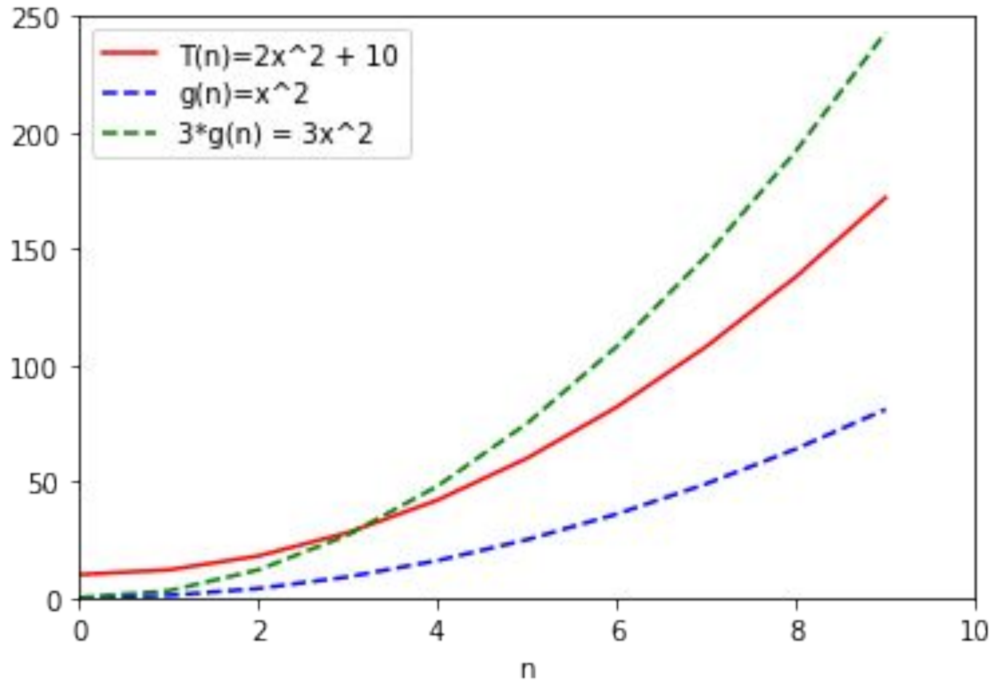
"There exists"

"such that"

Example

$$2n^2 + 10 = O(n^2)$$

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$T(n) \leq c \cdot g(n)$$



$3g(n) = 3n^2$

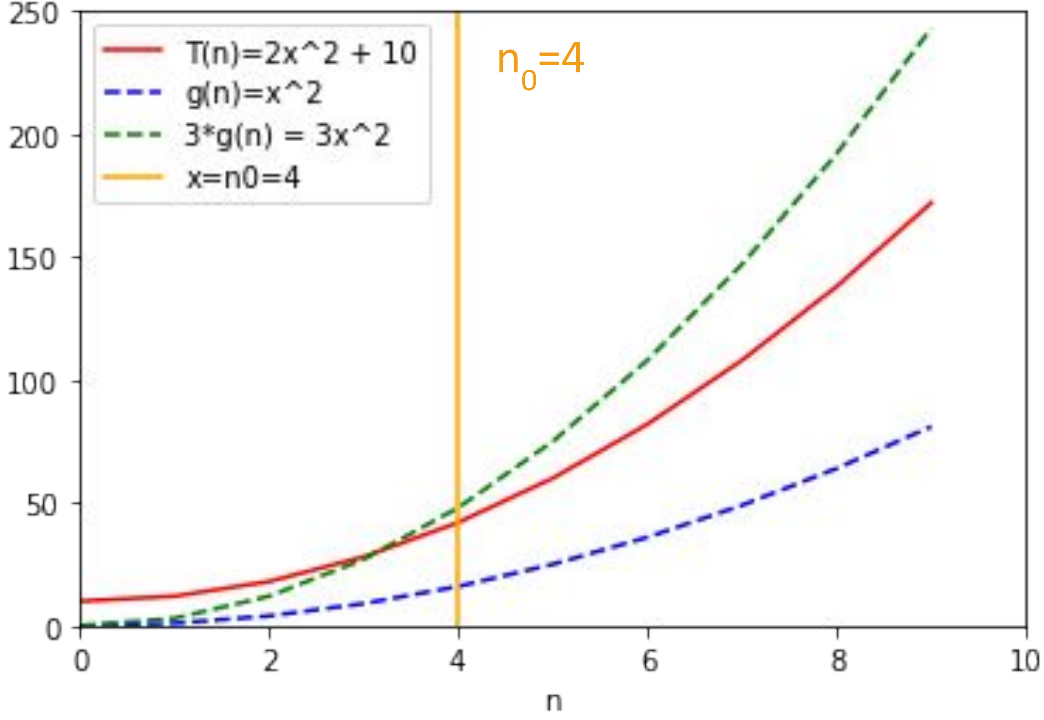
$T(n) = 2n^2 + 10$

$g(n) = n^2$

Example

$2n^2 + 10 = O(n^2)$

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \ \ s.t. \ \ \forall n \geq n_0,$$
$$T(n) \leq c \cdot g(n)$$



$3g(n) = 3n^2$
$(c = 3)$

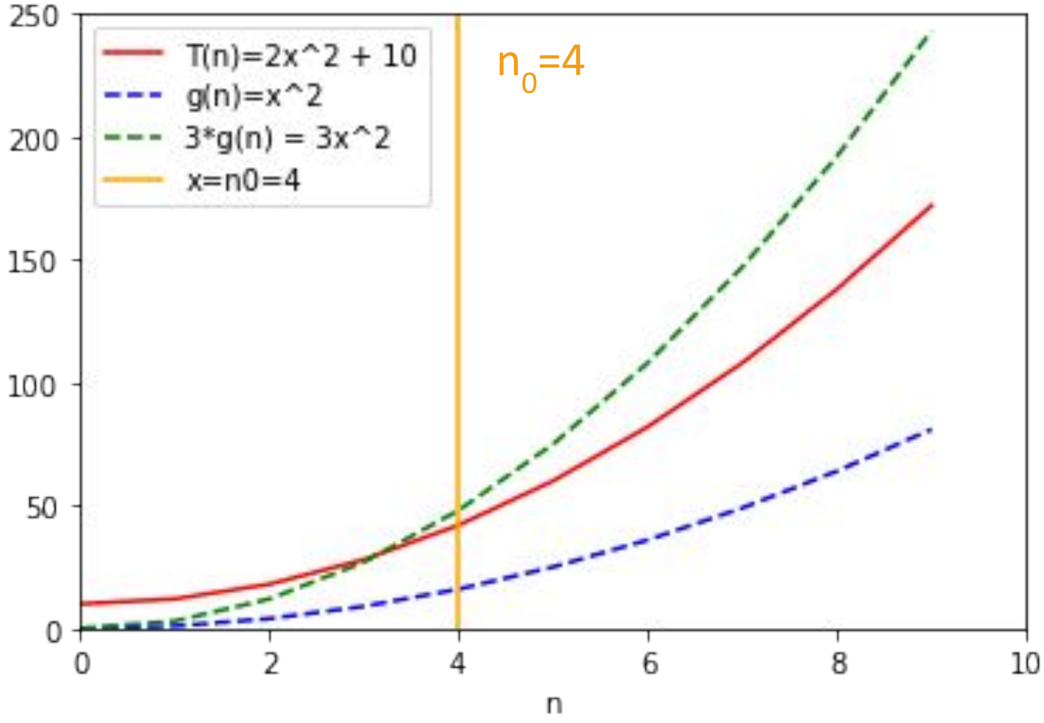
$T(n) = 2n^2 + 10$

$g(n) = n^2$

Example

$$2n^2 + 10 = O(n^2)$$

$$T(n) = O(g(n)) \iff \exists c, n_0 > 0 \ \ s.t. \ \ \forall n \geq n_0, \ T(n) \leq c \cdot g(n)$$



$3g(n) = 3n^2$
($c = 3$)

$T(n) = 2n^2 + 10$

$g(n) = n^2$

Formally:

- Choose c = 3
- Choose n_0 = 4
- Then:

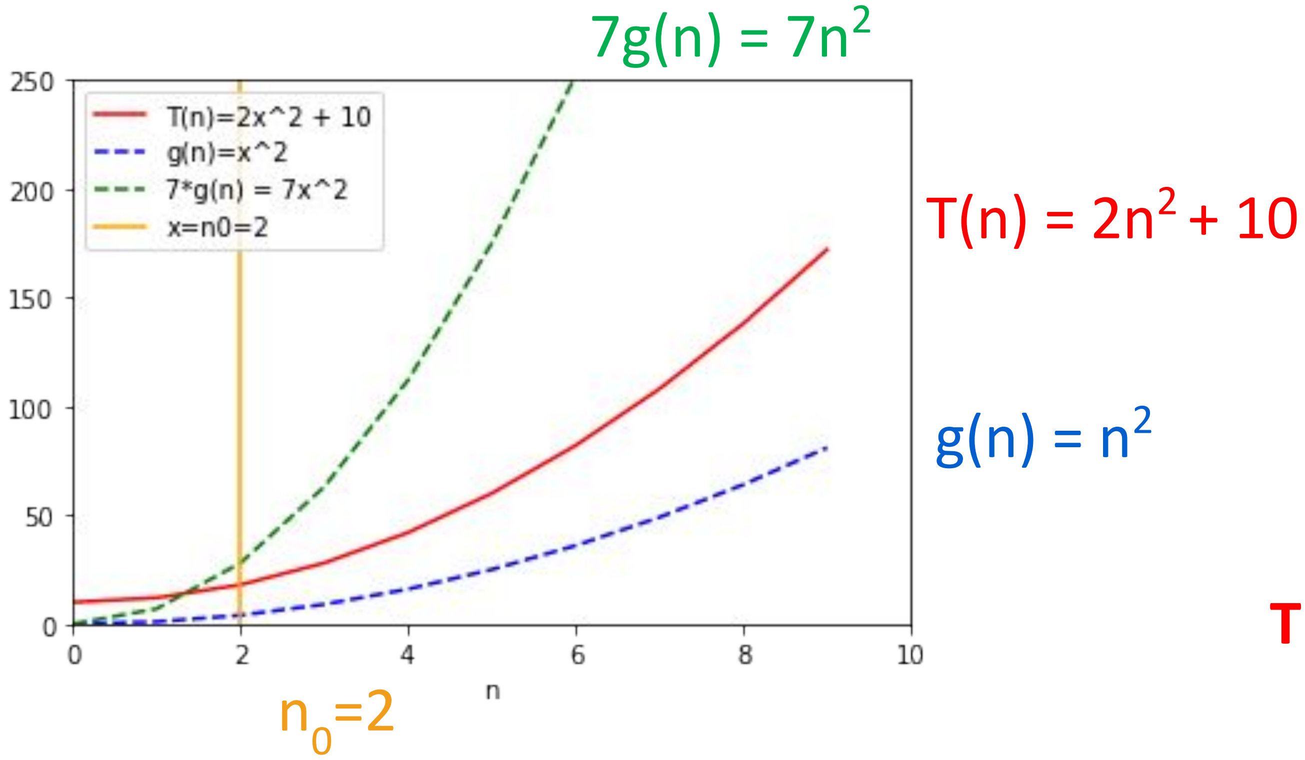
$$\forall n \geq 4,$$

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

Same example

$2n^2 + 10 = O(n^2)$

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$T(n) \leq c \cdot g(n)$$



Formally:

- Choose c = 7
- Choose n_0 = 2
- Then:

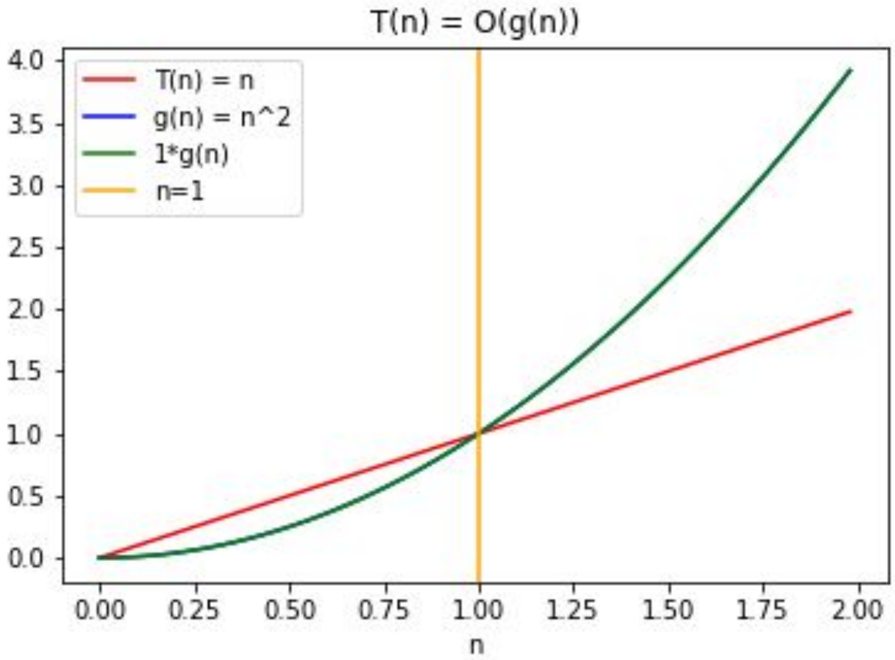
$$\forall n \geq 2,$$
$$2n^2 + 10 \leq 7 \cdot n^2$$

There is no "correct" choice of c and n_0

O(...) is an upper bound:

$$n = O(n^2)$$

$$T(n) = O(g(n))$$
$$\Longleftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$T(n) \leq c \cdot g(n)$$



$g(n) = n^2$

$T(n) = n$

- Choose c = 1
- Choose n_0 = 1
- Then

$$\forall n \geq 1,$$
$$n \leq n^2$$

pronounced "big-omega of …"

$\Omega(\ldots)$ means a lower bound

- We say "$T(n)$ is $\Omega\big(g(n)\big)$" if, for large enough n, $T(n)$ is at least as big as a constant multiple of $g(n)$.

- Formally,

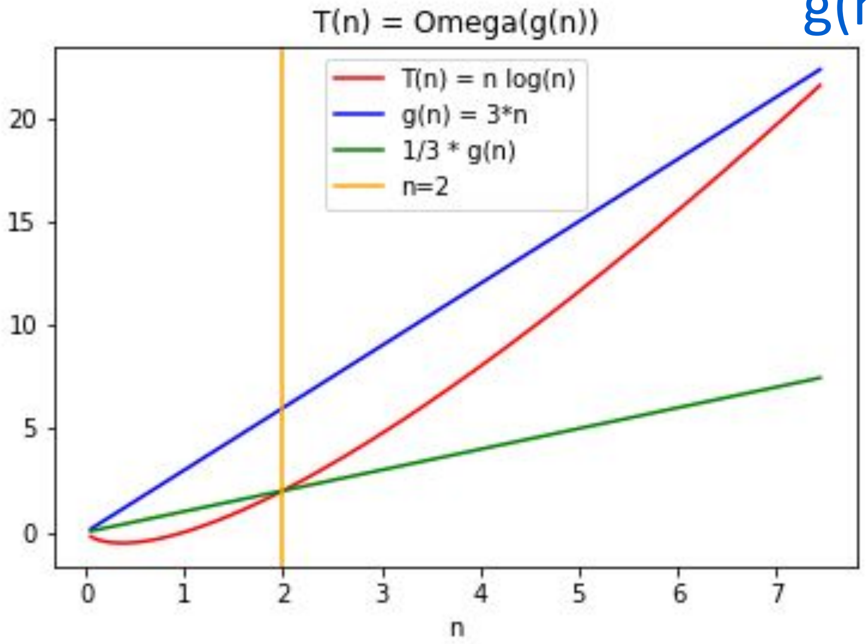
$$T(n) = \Omega\big(g(n)\big)$$
$$\Longleftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$c \cdot g(n) \leq T(n)$$

Switched these!!

Example

$$n \log_2(n) = \Omega(3n)$$

$$T(n) = \Omega(g(n)) \iff \exists c, n_0 > 0 \ s.t. \ \forall n \geq n_0, \ c \cdot g(n) \leq T(n)$$



g(n) = 3n

T(n) = nlog(n)

g(n)/3 = n

- Choose c = 1/3
- Choose n_0 = 2
- Then

$$\forall n \geq 2,$$

$$\frac{3n}{3} \leq n \log_2(n)$$

pronounced "big-theta of …" or sometimes "theta of"

Θ(…) means both!

- We say "$T(n)$ is $\Theta(g(n))$" iff both:

$$T(n) = O(g(n))$$

and

$$T(n) = \Omega(g(n))$$

$$T(n) = O(g(n)) \iff \exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0, \;\; T(n) \leq c \cdot g(n)$$

Non-Example: n^2 is not $O(n)$

- Proof by contradiction:
- Suppose that $n^2 = O(n)$.
- Then there is some positive c and n_0 so that:

$$\forall n \geq n_0, \qquad n^2 \leq c \cdot n$$

- Divide both sides by n:

$$\forall n \geq n_0, \qquad n \leq c$$

- That's not true!!! What about, say, $n_0 + c + 1$?
  - Then $n \geq n_0$, but , $n > c$
- Contradiction!

Take-away from examples

- To prove $T(n) = O(g(n))$, you have to come up with c and n_0 so that the definition is satisfied.
- To prove T(n) is NOT O(g(n)), one way is **proof by contradiction**:
  - Suppose (to get a contradiction) that someone gives you a c and an n_0 so that the definition ***is*** satisfied.
  - Show that this someone must by lying to you by deriving a contradiction.

Practice

- $f(n) = n$ and $g(n) = n^2 - n$

  $f(n) =$ ___ $g(n)$

- $f(n) = 2^n$ and $g(n) = n^2$

  $f(n) =$ ___ $g(n)$

- $f(n) = 8n$ and $g(n) = n \log n$

  $f(n) =$ ___ $g(n)$

Practice

- $f(n) = n$ and $g(n) = n^2 - n$

  $f(n) = O(g(n))$ n grows slower than n^2

- $f(n) = 2^n$ and $g(n) = n^2$

  $f(n) = \Omega(g(n))$ polynomial functions are slower than exponential functions

Practice

- *f(n) = 8n* and *g(n) = n log n*

  $f(n) = O(g(n))$

  $c > 0$, $n^c = O(n^c \log n)$

  with $c = 1$, $f(n) = O(g(n))$

  $\lim_{n \to \infty} 8n / n \log n$

  $= \lim_{n \to \infty} 8 / \log n$

  $= 0$

State order of growth in Θ notation

- $f(n) = 50$

- $f(n) = n + \ldots + 3 + 2 + 1$

- $f(n) = (g(n))^2$ where $g(n) = \sqrt{n} + 5$

State order of growth in Θ notation

- $f(n) = 50$

  $f(n) = \Theta(1)$
- $f(n) = n + \ldots + 3 + 2 + 1$

  $f(n) = n(n+1)/2 = (n^2 + n)/2 = \Theta(n^2)$
- $f(n) = (g(n))^2$ where $g(n) = \sqrt{n} + 5$

  $f(n) = (\sqrt{n} + 5)^2 = n + 10\sqrt{n} + 25 = \Theta(n)$

Summary of Definitions

$f(n) = O(g(n))$ if there exists a $c > 0$ where after large enough n, $f(n) \leq c * g(n)$

Asymptotically f grows as most as much as g

$f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$

Asymptotically, f grows at least as much as g

$f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $g(n) = O(f(n))$

Asymptotically, f and g grow the same

Important Takeaways

- If $d > c$, $n^c = O(n^d)$ but $n^c \neq \Omega(n^d)$
- Asymptotic notation only cares about the highest growing terms: e.g. $n^2 + n = \Omega(n^2)$
- Asymptotic notation does not care about leading constants: e.g. $50n = \Theta(n)$
- Any exponential with base > 1 grows more than any polynomial
- The base of the exponential matters: e.g. $3^n = O(4^n)$ but $3^n \neq \Omega(4^n)$

Any questions?