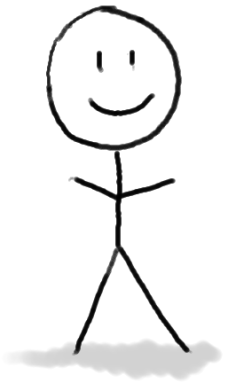
Worst-case analysis

Sorting a sorted list should be fast!!

1	2	3	4	5	6	7	8

- In this class, we will focus on worst-case analysis



Algorithm designer

Here is my algorithm!

```
Algorithm:
  Do the thing
  Do the stuff
  Return the answer
```

- Pros: very strong guarantee
- Cons: very strong guarantee

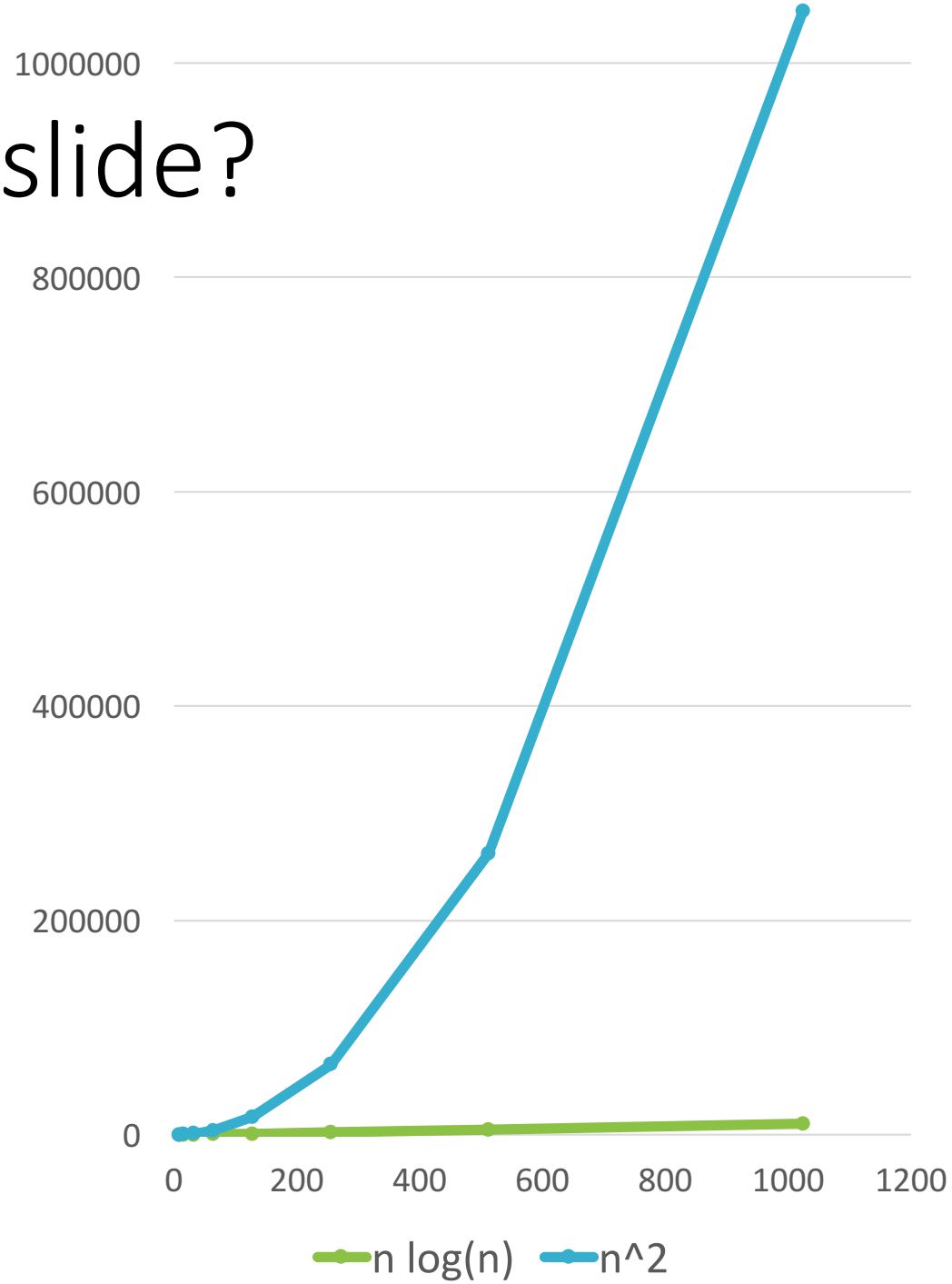
Big-O notation

How long does an operation take? Why are we being so sloppy about that "6"?

- What do we mean when we measure runtime?
  - We probably care about wall time: how long does it take to solve the problem, in seconds or minutes or hours?
- This is heavily dependent on the programming language, architecture, etc.
- These things are very important, but are not the point of this class.
- We want a way to talk about the running time of an algorithm, independent of these considerations.

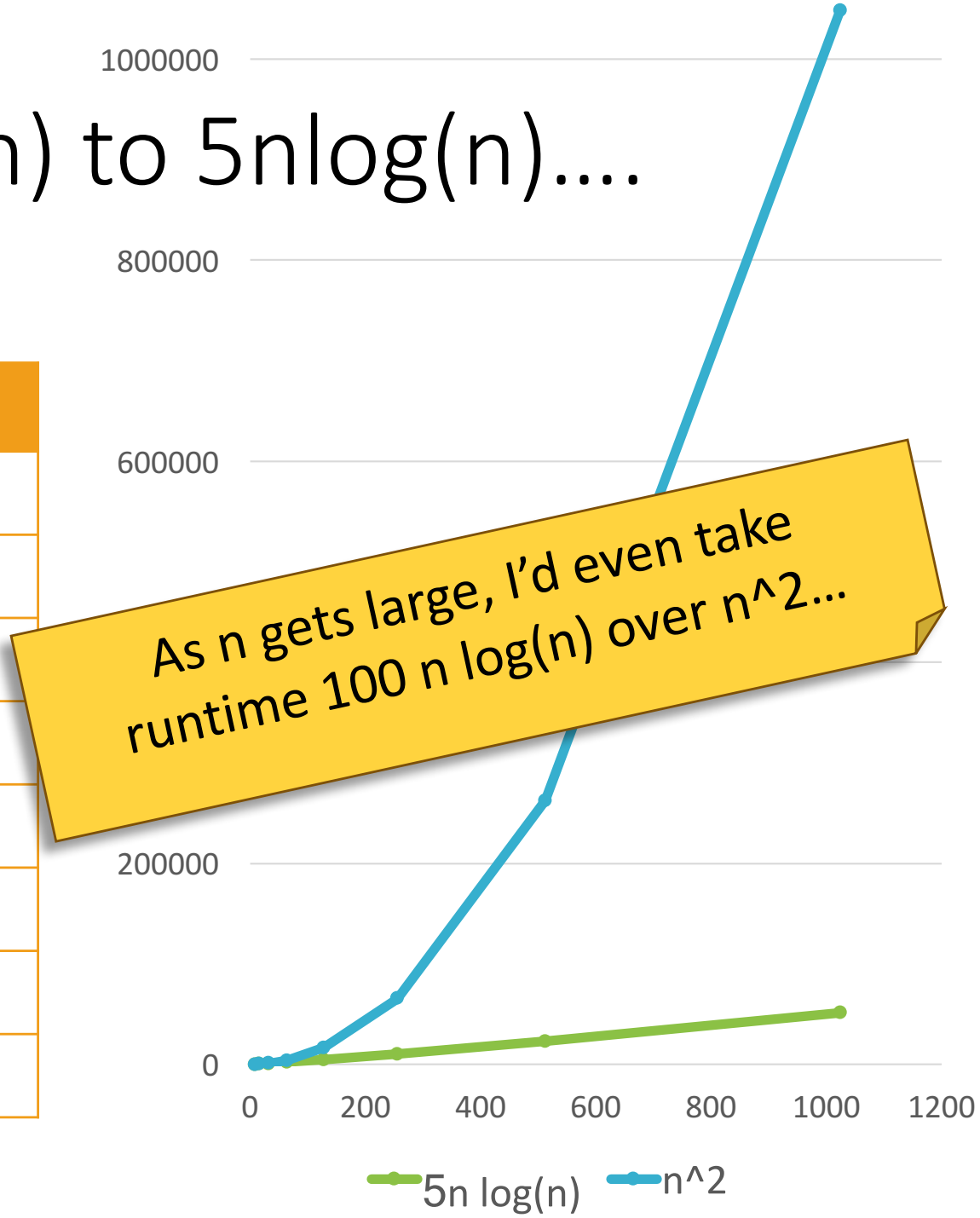
Remember this slide?

n	n log(n)	n^2
8	24	64
16	64	256
32	160	1024
64	384	4096
128	896	16384
256	2048	65536
512	4608	262144
1024	10240	1048576



Change nlog(n) to 5nlog(n)….

n	5n log(n)	n^2
8	120	64
16	320	256
32	800	1024
64	1920	4096
128	4480	16384
256	10240	65536
512	23040	262144
1024	51200	1048576



As n gets large, I'd even take runtime 100 n log(n) over n^2…

Asymptotic Analysis

How does the running time scale as n gets large?

One algorithm is "faster" than another if its runtime grows more "slowly" as n gets large.

This will provide a formal way of saying that n^2 is "worse" than 100 n log(n).

This is especially relevant now, as data get bigger and bigger and bigger…

Pros:

- Abstracts away from hardware- and language-specific issues.
- Makes algorithm analysis much more tractable.

Without making Plucky grumpy!

Cons:

- Only makes sense if n is large (compared to the constant factors).

$2^{100000000000000}\, n$ is "better" than n^2 ?!?!

Now for some definitions…

- Quick reminders:
  - ∃: “There exists”
  - ∀: ”For all”
  - Example: ∀ students in CS161, ∃ an algorithms problem that really excites the student.
  - Much stronger statement: ∃ an algorithms problem so that, ∀ students in CS161, the student is excited by the problem.
- We’re going to formally define an upper bound:
  - “T(n) grows no faster than f(n)”

pronounced “big-oh of …” or sometimes “oh of …”

O(…) means an upper bound

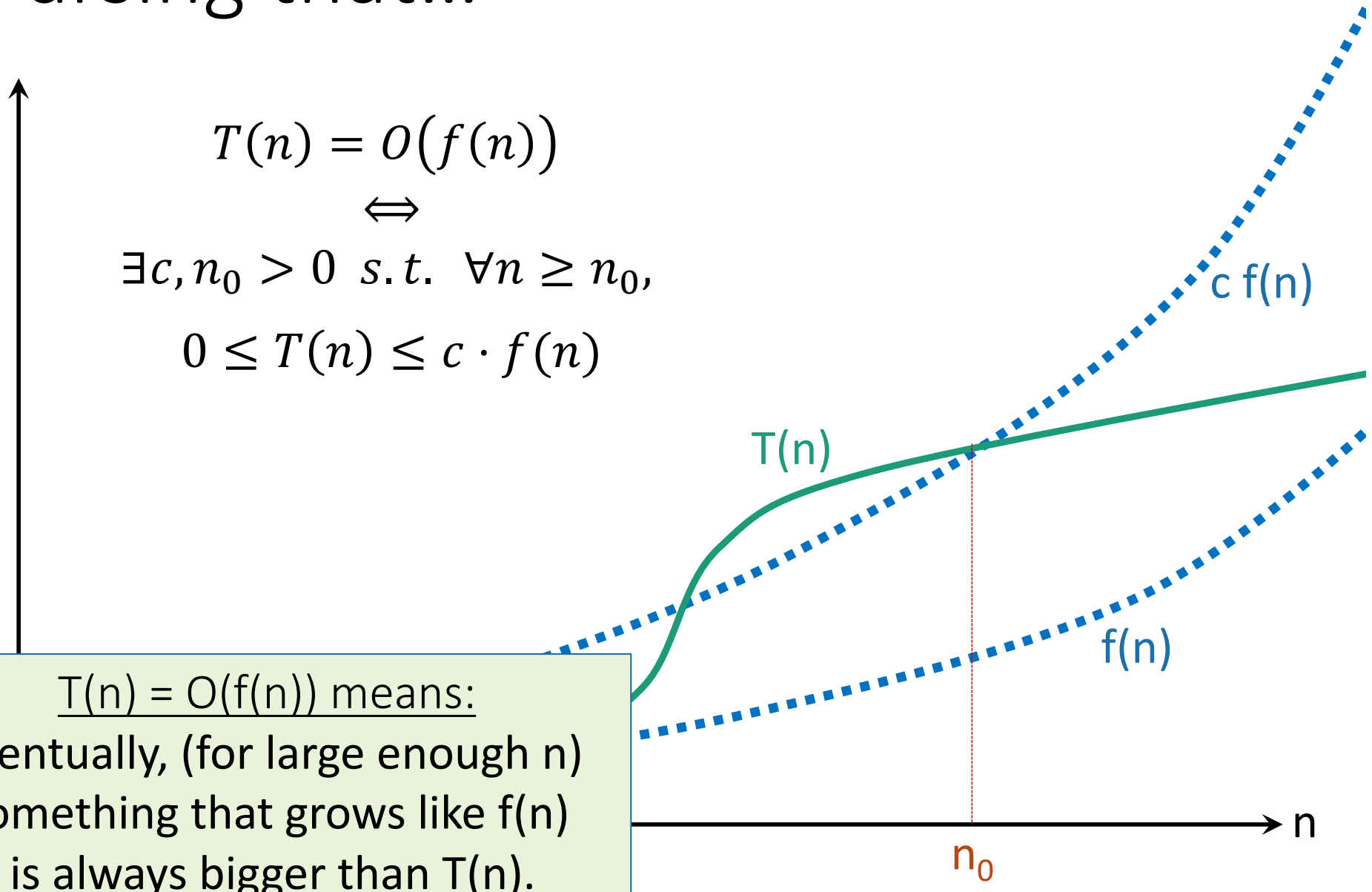
- Let T(n), f(n) be functions of positive integers.
  - Think of T(n) as being a runtime: positive and increasing in n.
- We say “T(n) is O(f(n))” if f(n) grows at least as fast as T(n) as n gets large.

- Formally,

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

Parsing that…

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

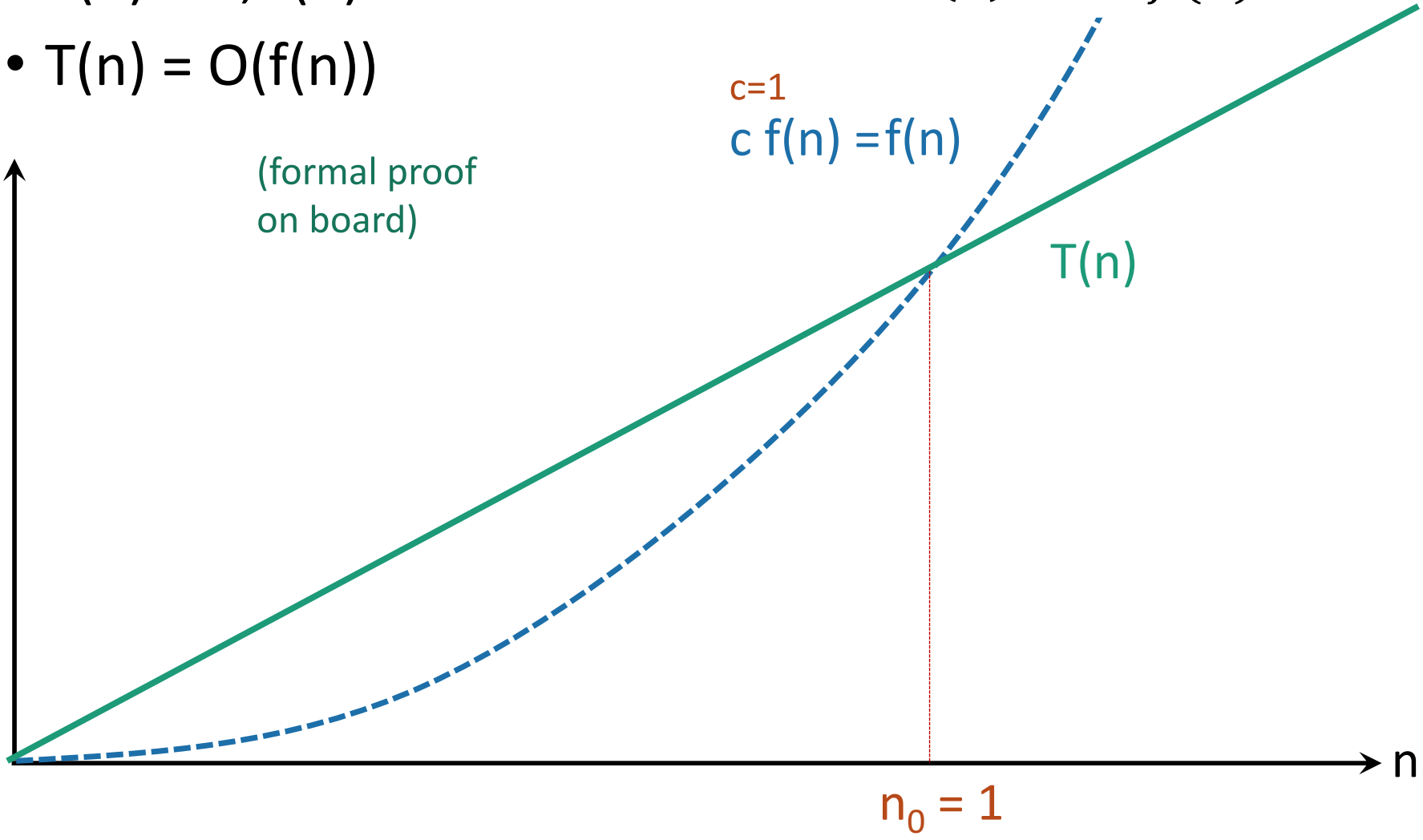


T(n) = O(f(n)) means:
Eventually, (for large enough n) something that grows like f(n) is always bigger than T(n).

Example 1

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

- T(n) = n, f(n) = n^2.
- T(n) = O(f(n))



Examples 2 and 3

- All degree k polynomials with positive leading coefficients are $O(n^k)$.
- For any $k \geq 1$, n^k is not $O(n^{k-1})$.

(On the board)

Take-away from examples

- To prove T(n) = O(f(n)), you have to come up with c and n_0 so that the definition is satisfied.

- To prove T(n) is NOT O(f(n)), one way is by contradiction:
  - Suppose 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.

O(…) means an upper bound, and

Ω(…) means a lower bound

- We say "T(n) is Ω(f(n))" if f(n) grows at most as fast as T(n) as n gets large.

- Formally,

$$T(n) = \Omega\big(f(n)\big)$$

$$\Leftrightarrow$$

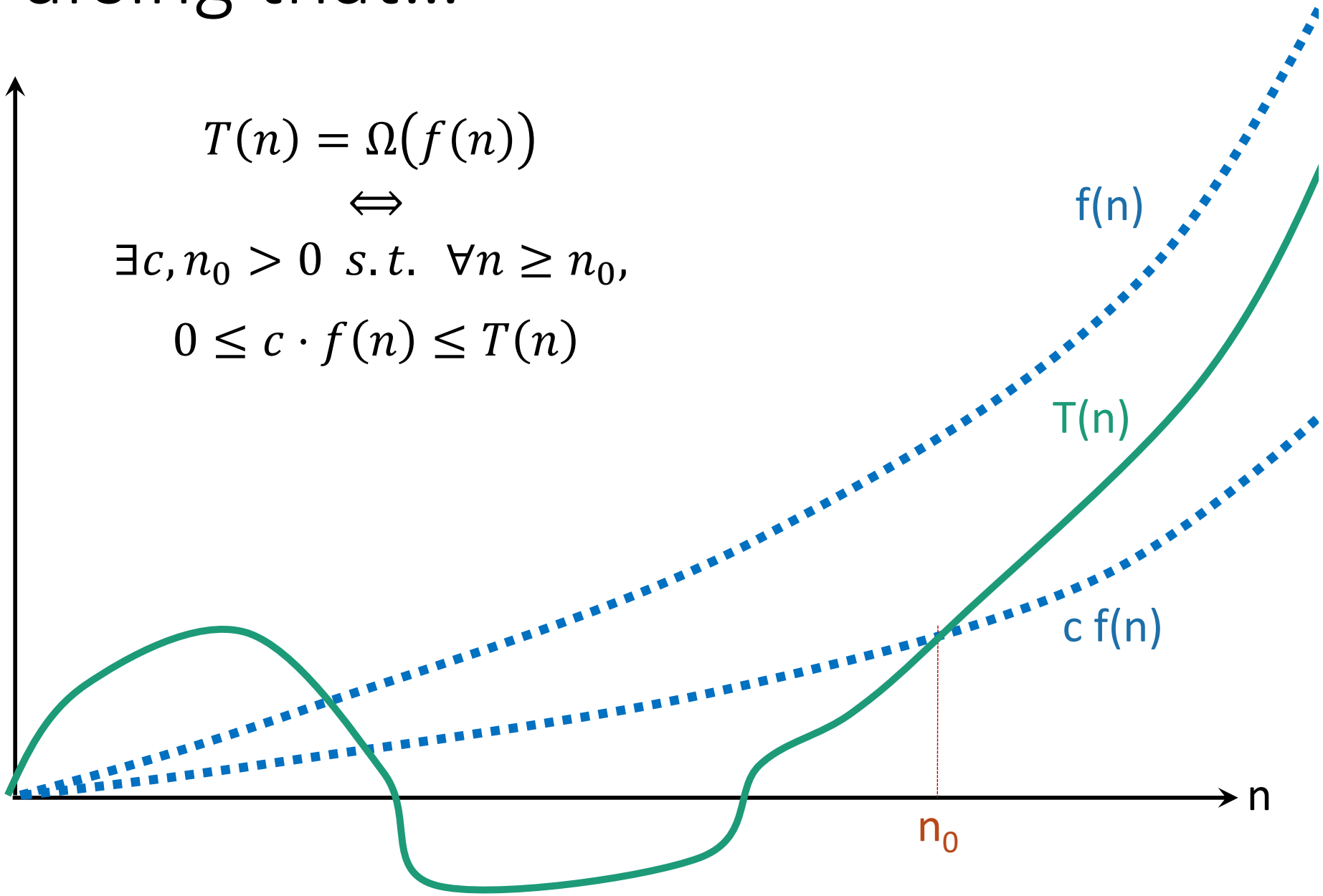
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$

$$0 \leq c \cdot f(n) \leq T(n)$$

Switched these!!

Parsing that…

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



Θ(…) means both!

- We say "T(n) is Θ(f(n))" if:

$$T(n) = O(f(n))$$

-AND-

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

Yet more examples

- $n^3 - n^2 + 3n = O(n^3)$
- $n^3 - n^2 + 3n = \Omega(n^3)$
- $n^3 - n^2 + 3n = \Theta(n^3)$

- 3^n is **not** $O(2^n)$
- $n \log(n) = \Omega(n)$
- $n \log(n)$ is **not** $\Theta(n)$.

Fun exercise:
check all of these
carefully!!

We'll be using lots of asymptotic notation from here on out

- This makes both Plucky and Lucky happy.
  - Plucky the Pedantic Penguin is happy because there is a precise definition.
  - Lucky the Lackadaisical Lemur is happy because we don't have to pay close attention to all those pesky constant factors like "4" or "6".
- But we should always be careful not to abuse it.
- In the course, (almost) every algorithm we see will be actually practical, without needing to take $n \geq n_0 = 2^{10000000}$.

This is my happy face!

Questions about asymptotic notation?