COMP 250

Lecture 37

Big Theta

best and worst cases

limits (revisited)

Dec. 4, 2018

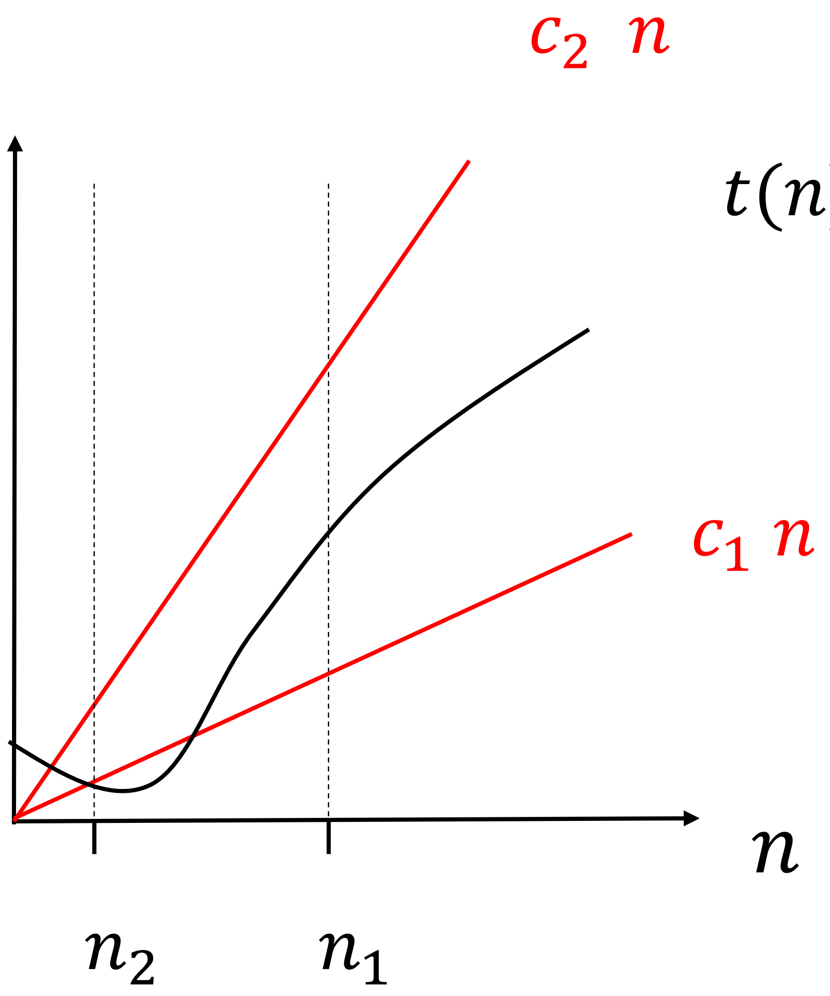
Definition of Big Theta (Θ)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

We say $t(n)$ is $\Theta(g(n))$

if $t(n)$ is *both* $\mathrm{O}(g(n))$ and $\Omega(g(n))$.

Example: $t(n)$ is $\Theta(\, n \,)$



For every $t(n)$, does there exist a “simple” $g(n)$ such that $t(n)$ is $\Theta(\ )$?

For every $t(n)$, does there exist a "simple" $g(n)$ such that $t(n)$ is $\Theta(\ )$?

No, as this contrived example shows:

$$t(n) = \begin{cases} 5, & n \text{ is odd} \\ n^2, & n \text{ is even.} \end{cases}$$

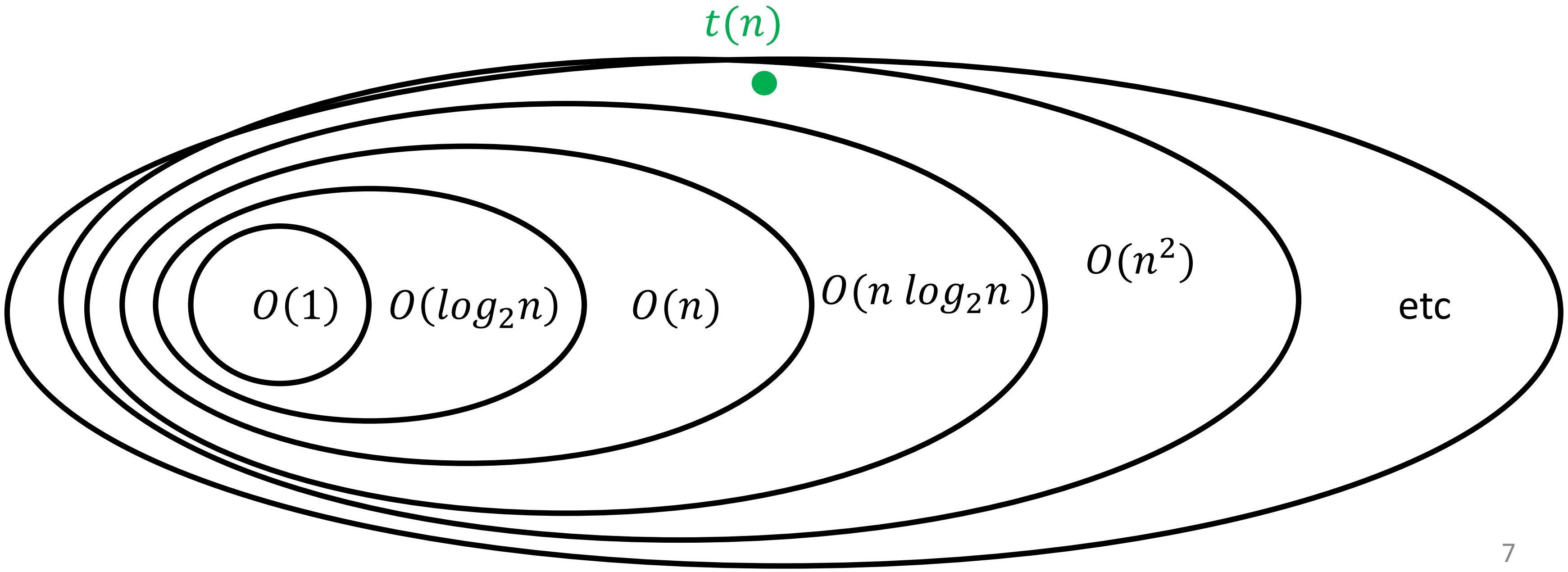
For every $t(n)$, does there exist a "simple" $g(n)$ such that $t(n)$ is $\Theta(\;)$?

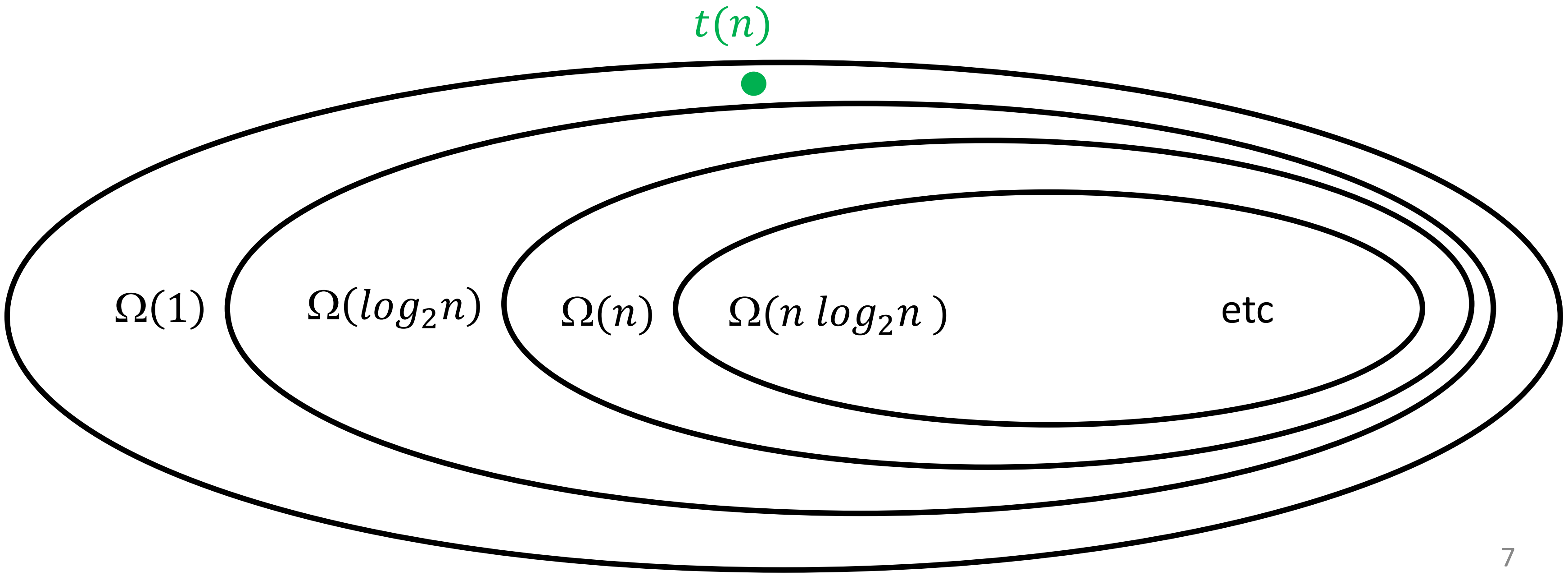
No, as this contrived example shows:

Let $t(n)$ = $\begin{cases} 5, & n \text{ is odd} \\ n^2, & n \text{ is even.} \end{cases}$

$t(n)$ is $O(n^2)$, and this is the tight upper bound.

$t(n)$ is $\Omega(1)$, and this is the tight lower bound.

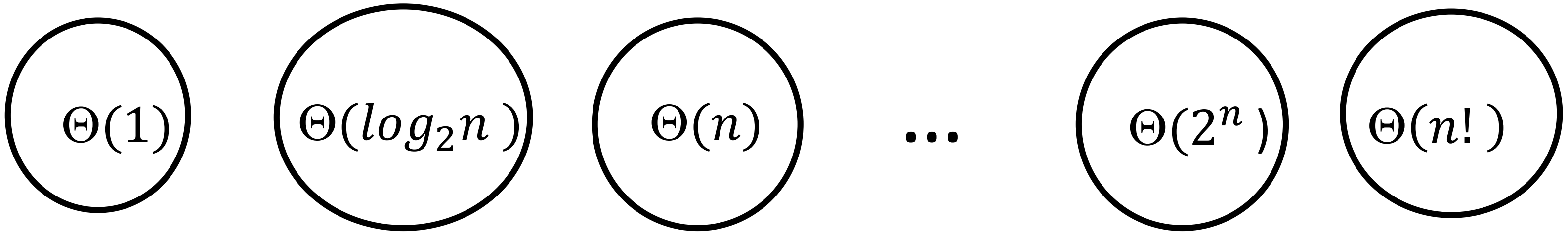

$t(n)$
$O(1)$
$O(log_2n)$
$O(n)$
$O(n\ log_2n\ )$
$O(n^2)$
etc


$t(n)$
$\Omega(1)$
$\Omega(log_2n)$
$\Omega(n)$
$\Omega(n\ log_2n\ )$
etc

Sets of $\Theta(\ )$ functions

If $t(n)$ is $\Theta(\ g(n)\ )$, we often write $t(n) \in \Theta(\ g(n)\ )$,

That is, $t(n)$ is a member of the set of functions that are $\Theta(\ g(n)\ )$.



For most of the semester,   we’ve talked about big O.
But what we usually meant was big Theta.

More on this next!

COMP 250

Lecture 37

Big Theta

best and worst cases

limits (revisited)

Dec. 4, 2018

The time it takes for an algorithm to run depends on:

- constant factors (often implementation dependent)
- the size n of the input
- ?

The time it takes for an algorithm to run depends on:

- constant factors (often implementation dependent)
- the size n of the input
- the values of the input

What are the best and worst cases?

For any algorithm,

let $t_{best}(n)$ be the runtime on the best case input(s).

let $t_{worst}(n)$ be the runtime for the worse case input(s).

e.g. Consider removing an element from an arbitrary position in an arraylist :

```
     arraylist.remove (i)
```

In the best case, ….

In the worst case, ….

e.g. Consider removing an element from an arbitrary position in an arraylist :

```
      arraylist.remove (i)
```

In the best case, the element is removed from the end of and so it can be done in constant time. So,

$$t_{best}(n) \text{ is } \Theta(1).$$

In the worst case, ...

e.g. Consider removing an element from an arbitrary position in an arraylist :

```
      arraylist.remove (i)
```

In the best case, the element is removed from the end of and so it can be done in constant time. So,

$$t_{best}(n) \text{ is } \Theta(1).$$

In the worst case, the element is removed from the start of the arraylist, so all elements must be shifted. So,

$$t_{worst}(n) \text{ is } \Theta(n).$$

Operations/Algorithms for Lists		$t_{best}(n)$	$t_{worst}(n)$
add, remove, find an element (array list)		$\Theta(1)$	$\Theta(n)$
add, remove, find an element (doubly linked list)		$\Theta(1)$	$\Theta(n)$
insertion sort		$\Theta(n)$	$\Theta(n^2)$
selection sort	best = worst	$\Theta(n^2)$	$\Theta(n^2)$
binary search (sorted array list)		$\Theta(1)$	$\Theta(\log n)$
mergesort		$\Theta(n \log n)$	$\Theta(n \log n)$
quick sort		$\Theta(n \log n)$	$\Theta(n^2)$

Heads up #1

When people want to express that an algorithm has different asymptotic behavior in best versus worst cases, they sometimes say things like:

"Quicksort best case is $\Omega(n \, log_2 n)$ and worst case is $O(n^2)$."   e.g. http://bigocheatsheet.com/

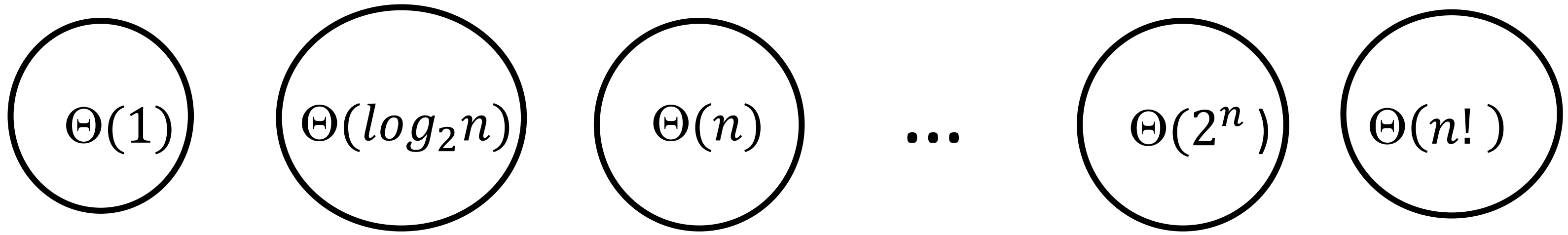
However, this is not correct, since it equates $\Omega(\,)$ with best case and it equates $O(\,)$ with worst case. But there is nothing in the formal definitions that requires this.

Rather, for any algorithm,

$t_{best}(n)$ is the best case runtime.

$t_{worst}(n)$ is the worse case runtime.

$t_{best}(n)$ and $t_{worst}(n)$ each *typically* belong to some $\Theta(\ )$ set, which are often different as in the table from a few slides ago.



So…

For most of the course, we've used the term O( ) loosely. We had in mind *worst case*.
But now we know better:

- O( ) is an upper bound
- It is *not necessarily* a tight upper bound.
- The formal definition of O( ) itself says nothing about algorithms, and hence nothing about best and worst cases of an algorithm.

COMP 250

Lecture 18

best and worst cases

limits (revisited)

Oct. 20, 2017

Q: Can we use limits to prove asymptotic bounds ?

A: Yes, if we apply certain rules.

Limit Rules: Case 1

If

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0$$

then

$t(n)$ is O($g(n)$)

From Calculus 1,

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0$$

means:

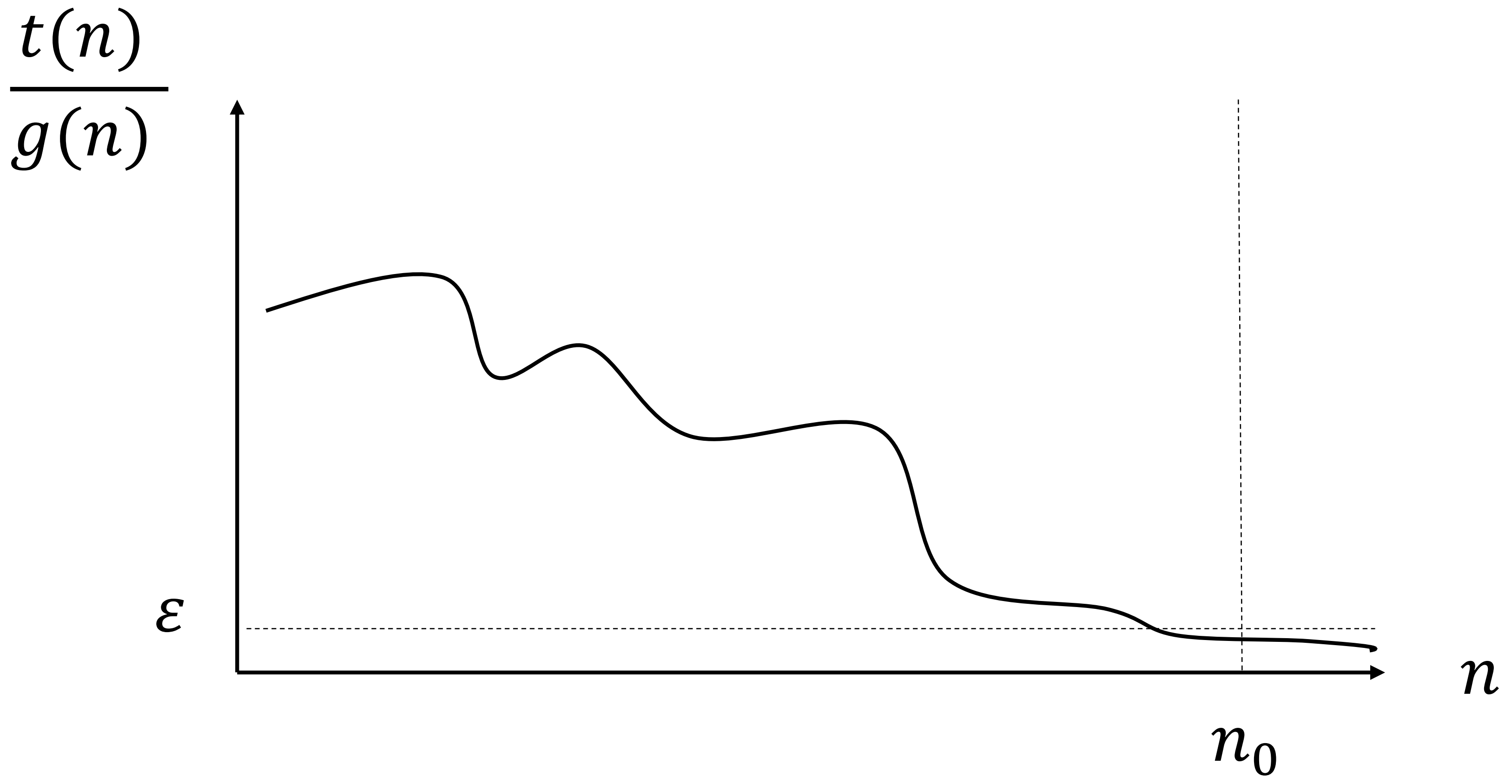
for any $\varepsilon > 0$, there exists an n_0 such that,
for any $n \geq n_0$,

$$\left| \frac{t(n)}{g(n)} \right| < \varepsilon.$$

and in particular,

$$t(n) < \varepsilon \; g(n)$$

Assume we are dealing with positive functions here.



Assume we are dealing with positive functions here.

From Calculus 1,

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0$$

means:

$\boxed{\text{for any } \varepsilon > 0,}$ there exists an n_0 such that,

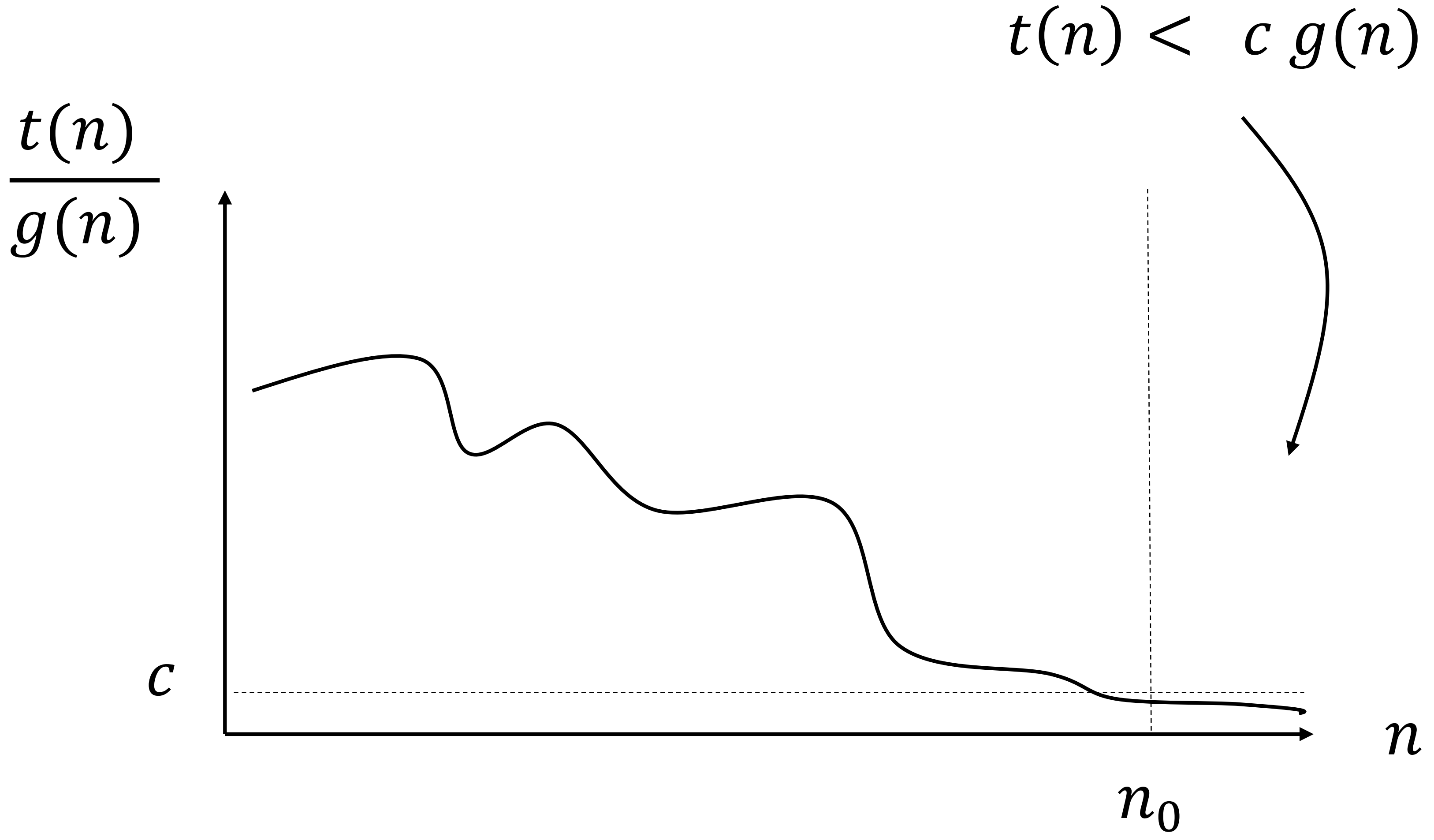
for any $n \geq n_0$,

$$\left| \frac{t(n)}{g(n)} \right| < \varepsilon.$$

Take such a constant. Call it c.

$$t(n) < c \; g(n)$$

Thus, $t(n)$ is O($g(n)$)



Assume we are dealing with positive functions here.

What about the opposite statement?

If $t(n)$ is $\mathrm{O}(\, g(n\,))$

then can we conclude: $\lim\limits_{\boldsymbol{n} \to \infty} \dfrac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \mathbf{0}$?

What about the opposite statement?

If $t(n)$ is $\mathrm{O}(\, g(n\,))$

then can we conclude: $\displaystyle \lim_{\boldsymbol{n} \to \infty} \frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \mathbf{0}$?

Absolutely not! Take $t(n) = g(n\,)$.

Then $t(n)$ is $\mathrm{O}(\, g(n\,))$, but $\frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = 1$ for all n.

Definition of Big Omega

Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

We say $t(n)$ is $\Omega(\, g(n)\,)$, if there exist two **positive** constants n_0 and c such that, for all $n \geq n_0$,

$$t(\,n\,) \geq \ c\ g(\,n\,)$$

Definition of Big Omega

Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

We say $t(n)$ is $\Omega(g(n))$, if there exist two **positive** constants n_0 and c such that, for all $n \geq n_0$,

$$t(n) \geq c \; g(n)$$

or equivalently

$$\frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} \geq \boldsymbol{c} \; .$$

Limit Rules: Case 1 (cont.)

If

$$\lim_{\boldsymbol{n} \to \infty} \frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \mathbf{0}$$

then: $t(n)$ is not $\Omega(\, g(n\,))$.

Proof is on the next slide.

$t(n)$ is $\Omega(\, g(n)\,)$ means (by definition)

there exist two constants n_0 and $c > 0$ such that,

for all $n \geq n_0$, $\dfrac{t(n)}{g(n)} \geq c$.

But this directly contradicts:

$$\lim_{\boldsymbol{n} \to \infty} \frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \mathbf{0}$$

since this limit requires that the opposite inequality

holds for large n.

Limit Rules: Case 1

If
$$\lim_{\boldsymbol{n} \to \infty} \frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \mathbf{0}$$

then: $t(n)$ is O$(\ g(n\ ))$ (1)

$t(n)$ is not $\Omega(\ g(n\ ))$ (2)

Thus, $t(n)$ is not $\Theta\ (\ g(n\ ))$.

Limit Rules: Case 2

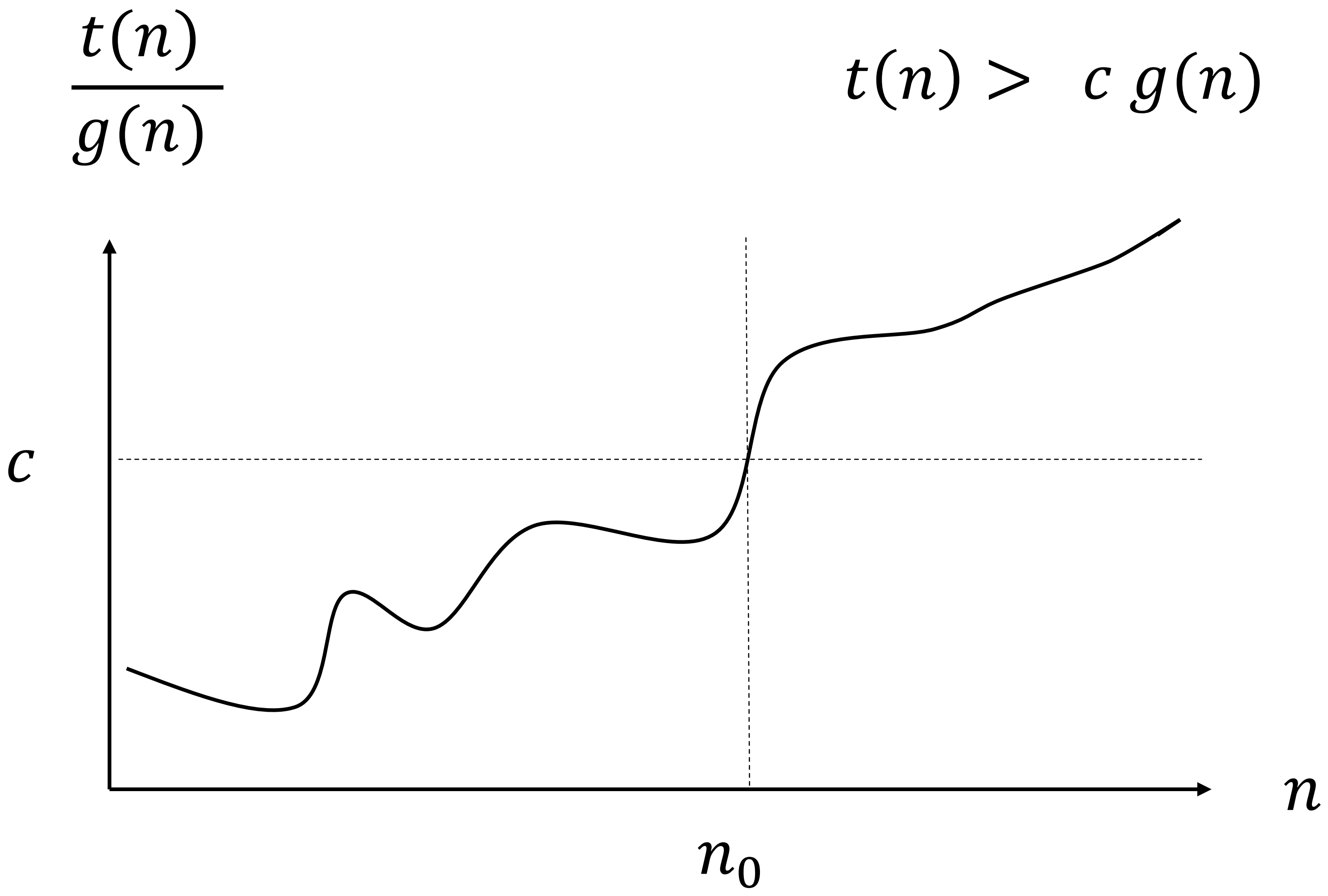
If $$\lim_{\boldsymbol{n} \to \infty} \frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \infty$$

then:

$t(n)$ is … ?

$t(n)$ is not … ?

$t(n) > \ c\, g(n)$



Limit Rules: Case 2

If $$\lim_{\boldsymbol{n} \to \infty} \frac{\boldsymbol{t(n)}}{\boldsymbol{g(n)}} = \infty$$

then:

$t(n)$ is $\Omega(\, g(n\,))$.

$t(n)$ is not $\mathrm{O}(\, g(n\,))$

Thus,

$t(n)$ is not $\Theta\,(\, g(n\,))$.

Proof has same idea. See updated lecture notes for details.

Limit Rules: Case 3

If

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = a\,, \quad 0 < a < \infty$$

then

$$t(n) \text{ is } \ldots ?$$

Limit Rules: Case 3

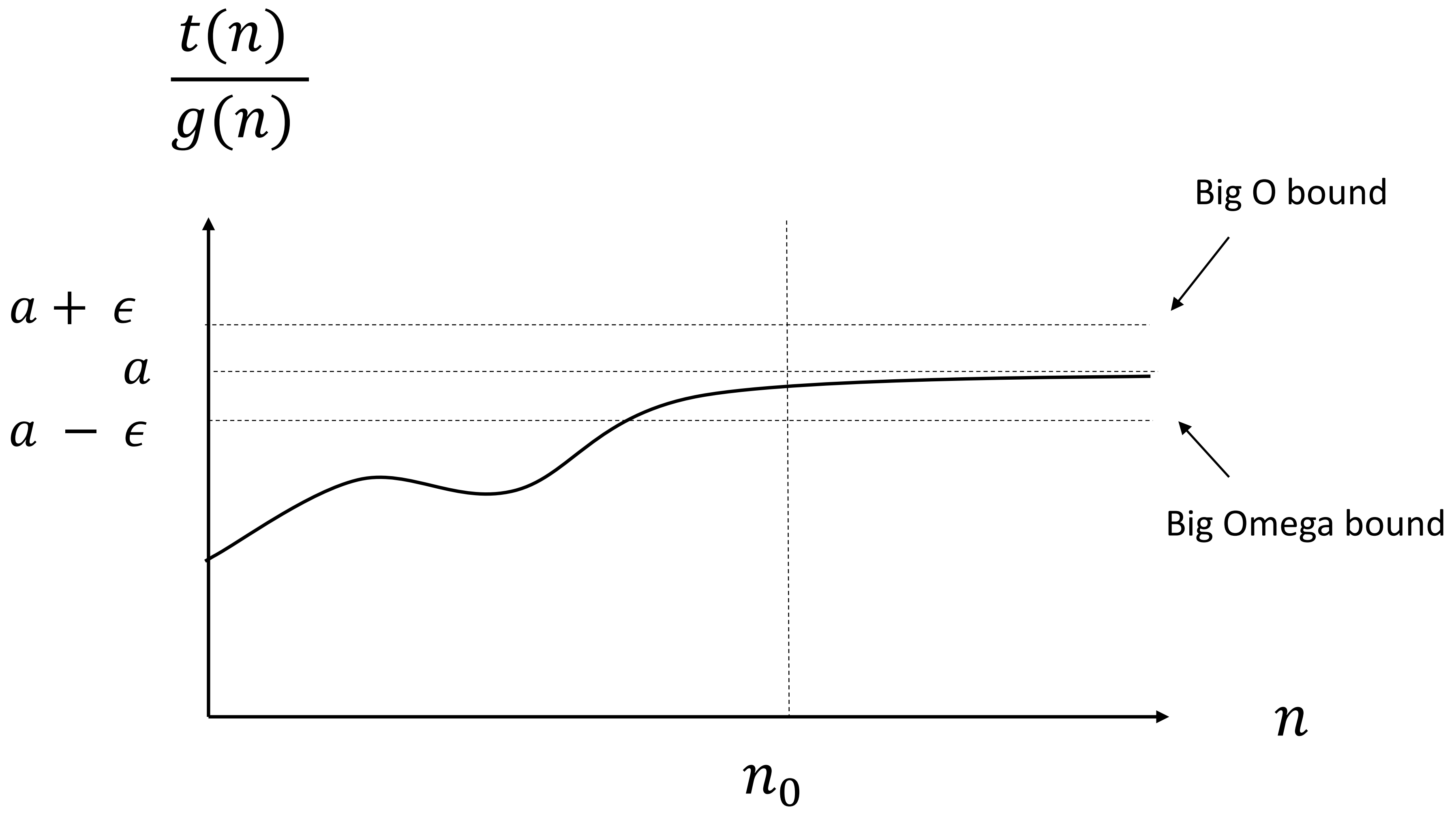
If

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = a, \quad 0 < a < \infty$$

Then

$$t(n) \text{ is } \Theta(g(n)).$$

Proof sketched on next slide (see lecture notes for details).


$\frac{t(n)}{g(n)}$
$a + \epsilon$
a
$a - \epsilon$
Big O bound
Big Omega bound
n
n_0

Limit Rules

The three rules just discussed say that if the limit exists:

$$\lim_{n \to \infty} \frac{t(n)}{g(n)}$$

then we can say something about the O, Ω, Θ bounds.

So, this gives us another way that we can quickly identify these bounds.

Final exam

- Closed book. No crib sheet. No calculators.
- 50 Questions. Multiple choice: 4 choices per question.
- Select one or leave blank.
  - Correct: 1/1
  - Leave blank: 0/1
  - Incorrect: $-\frac{1}{5}$ (*small* penalty to counter lucky guesses)

See grading policy on the Course Outline.

Final Exam

- About half of the questions require calculations applying an algorithm.   So you need to know all the algorithms.
- About half the questions are about definitions or properties of data structures, or time complexity i.e. no calculation.
- There are no “none of the above” choices or trick questions.   One correct answer!

How to prepare for the Final Exam?

- Review the slides (or 2017 or updated notes)
- Do the Exercises.
- When done with the above, get some rest!
  (Top athletes do not train directly before a big
  competition.)

After COMP 250 ?

If you are thinking of taking COMP 251 and you have not yet taken MATH 240,   make sure that you can follow the logical arguments of the last few lectures.

If you can’t, then taken MATH 240 in Winter and save COMP 251 for Fall 2019.

BTW,   a second section of COMP 251 has opened up in Winter.   Both sections will be co-taught by Luc Devroye and Erin McLeish.

See you in office hours (to be announced),
or at the final exam.

Good luck!