COMP2054-ACE: Introduction to big-Oh

ADE-Lec02b

Lecturer: Andrew Parkes

Email: andrew.parkes@Nottingham.ac.uk

from http://www.cs.nott.ac.uk/~pszajp/

We continue with the motivations and context for the definition of big-Oh, by giving some examples.

The examples are designed to be very easy in terms of the mechanics/arithmetic in the proofs, but to capture important principles.

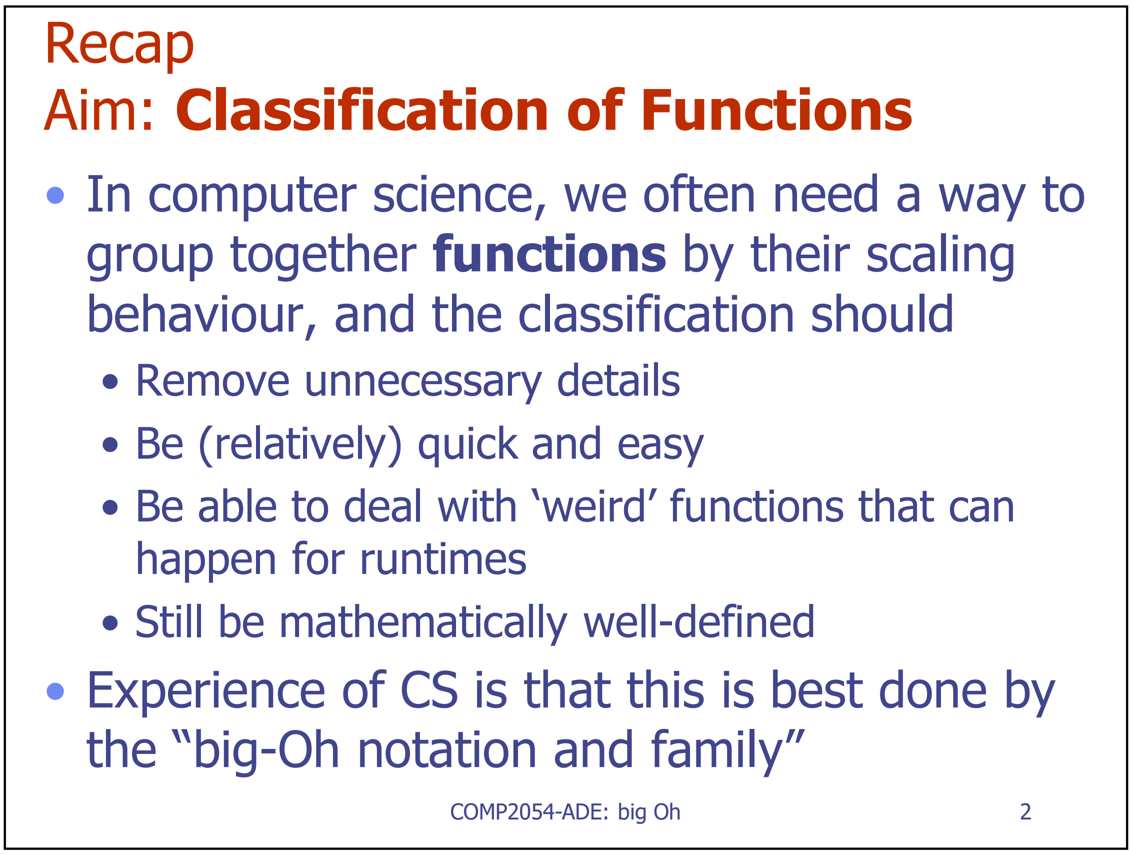
Suggest not to let the easiness of the arithmetic distract from the importance of the principles.

An aim is to counteract many standard misconceptions and errors BEFORE moving to the confusion of non-trivial arithmetic.

Again it is important to emphasise that the main point is that a proper understanding of a definition

should include an understanding of the components of a definition.

Definitions are not things that exist of their own accord, but are designed by people to be useful to people.



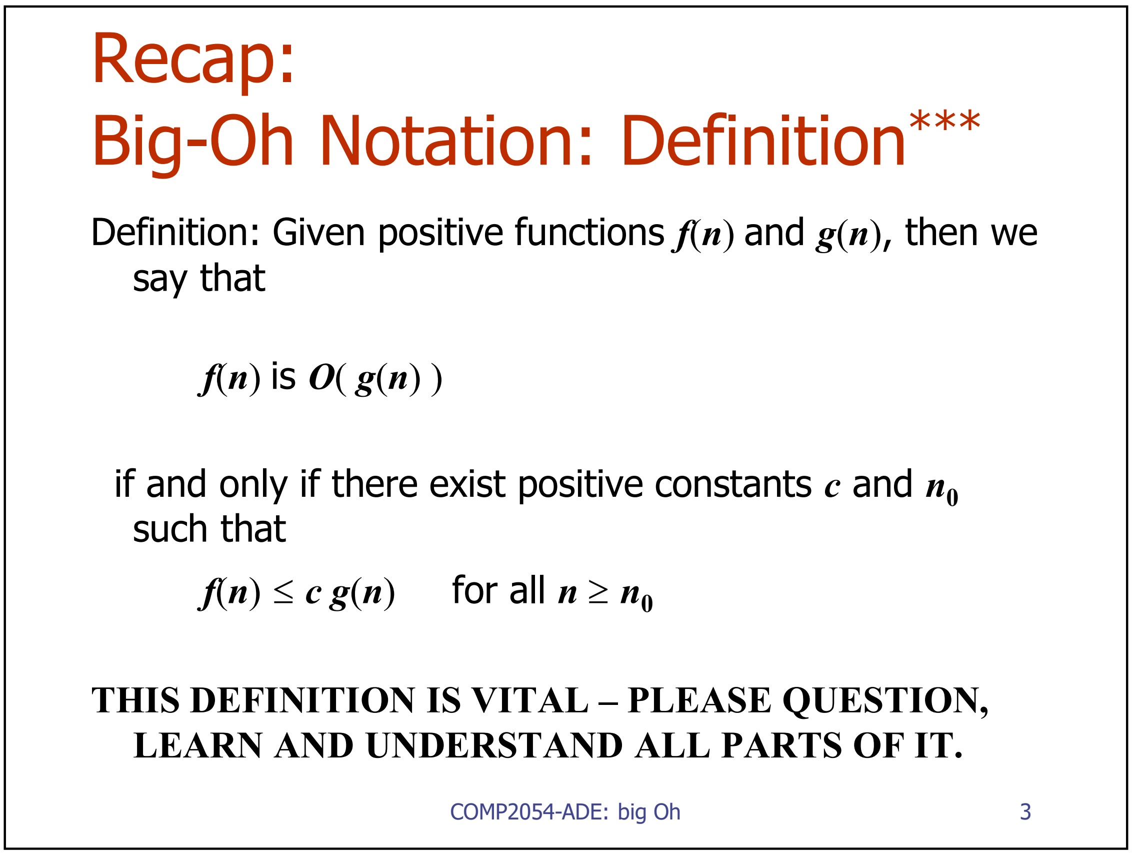
Recap
Aim: **Classification of Functions**

- In computer science, we often need a way to group together **functions** by their scaling behaviour, and the classification should
  - Remove unnecessary details
  - Be (relatively) quick and easy
  - Be able to deal with 'weird' functions that can happen for runtimes
  - Still be mathematically well-defined
- Experience of CS is that this is best done by the "big-Oh notation and family"

First a recap, that the aim is to give a way to classify functions in ways that are most useful for CS.

Firstly, notice here that I deliberately write "**FUNCTIONS**" and not algorithms. The approach is to consider the functions that arise from analysis of runtimes of algorithms,

but then forget about the algorithms and instead focus on grouping together functions in ways that put functions into sensible classes.



Recap: Big-Oh Notation: Definition***

Definition: Given positive functions $f(n)$ and $g(n)$, then we say that

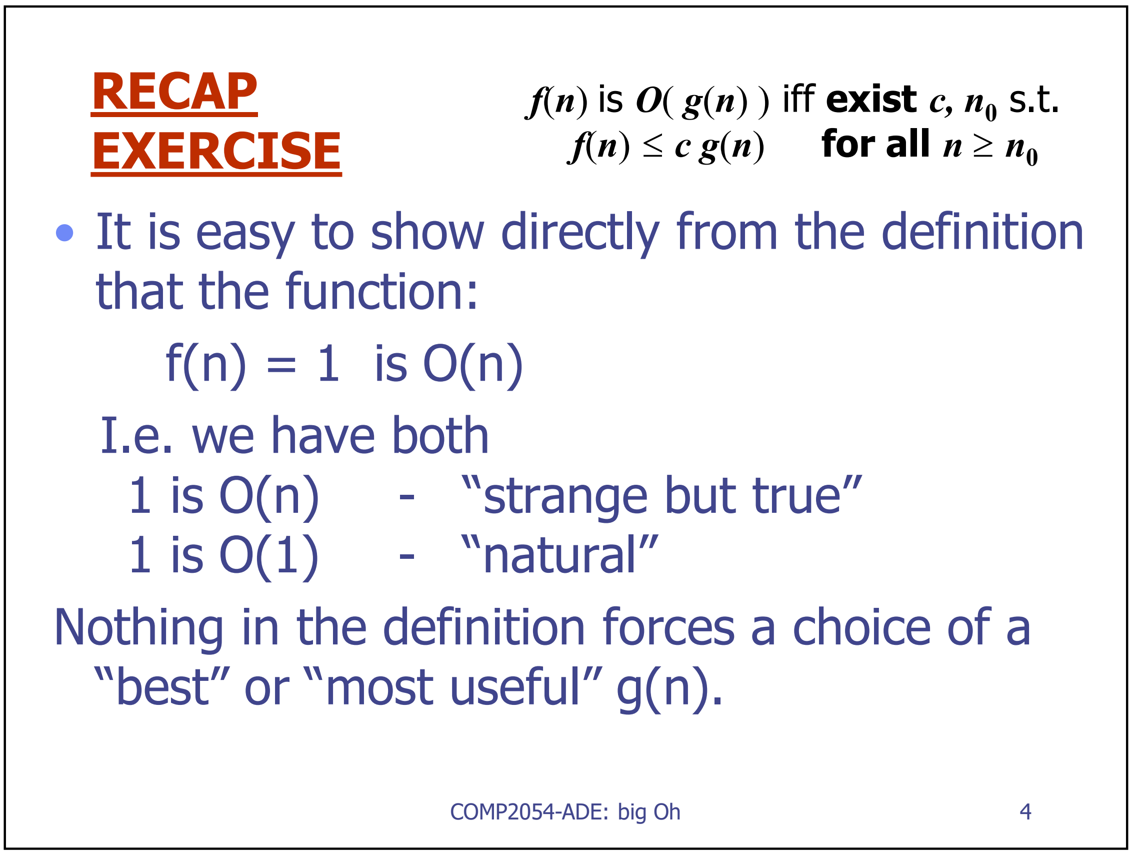
$f(n)$ is $O(\, g(n)\,)$

if and only if there exist positive constants c and n_0 such that

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

THIS DEFINITION IS VITAL – PLEASE QUESTION, LEARN AND UNDERSTAND ALL PARTS OF IT.

Hence, the natural definition that captures a lot of what we need in CS.



RECAP EXERCISE

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \leq c\ g(n)$ **for all** $n \geq n_0$

- It is easy to show directly from the definition that the function:

  f(n) = 1 is O(n)

  I.e. we have both

  1 is O(n) - "strange but true"
  1 is O(1) - "natural"

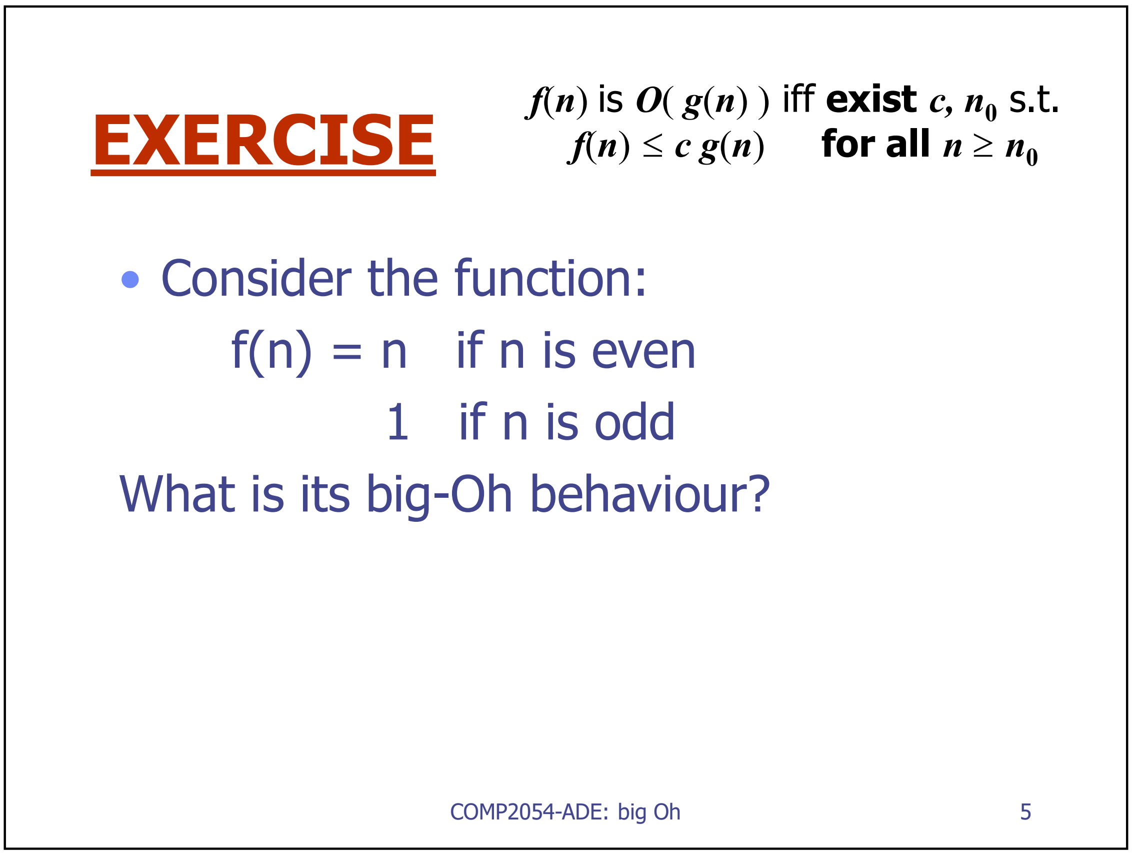
Nothing in the definition forces a choice of a "best" or "most useful" g(n).

COMP2054-ADE: big Oh 4

One of the results was possibly unexpected was that the definition in itself does not force that only the natural big-Oh is provable.

Weaker results are also possible in terms of what follows strictly from the definition.

It is just a convention, or community standard, to try to pick the function inside the big-Oh to be the most useful.



EXERCISE

$f(n)$ is $O(\,g(n)\,)$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- Consider the function:

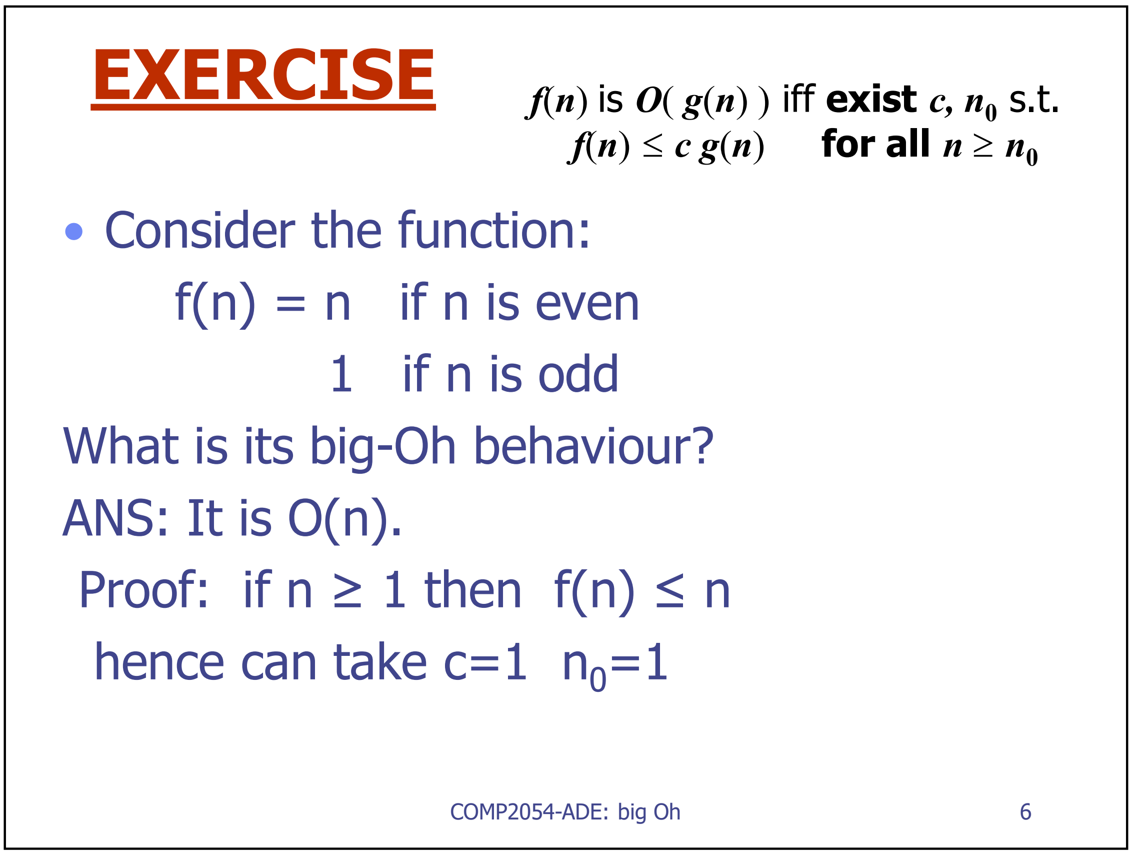
$$f(n) = \begin{array}{ll} n & \text{if n is even} \\ 1 & \text{if n is odd} \end{array}$$

What is its big-Oh behaviour?

But let us continue with a slightly less trivial function.

In particular, a function that is not a simple monotonically increasing function, and could occur in real CS.

What can we say about its big-Oh behaviour.



EXERCISE

$f(n)$ is $O(g(n))$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is its big-Oh behaviour?

ANS: It is O(n).

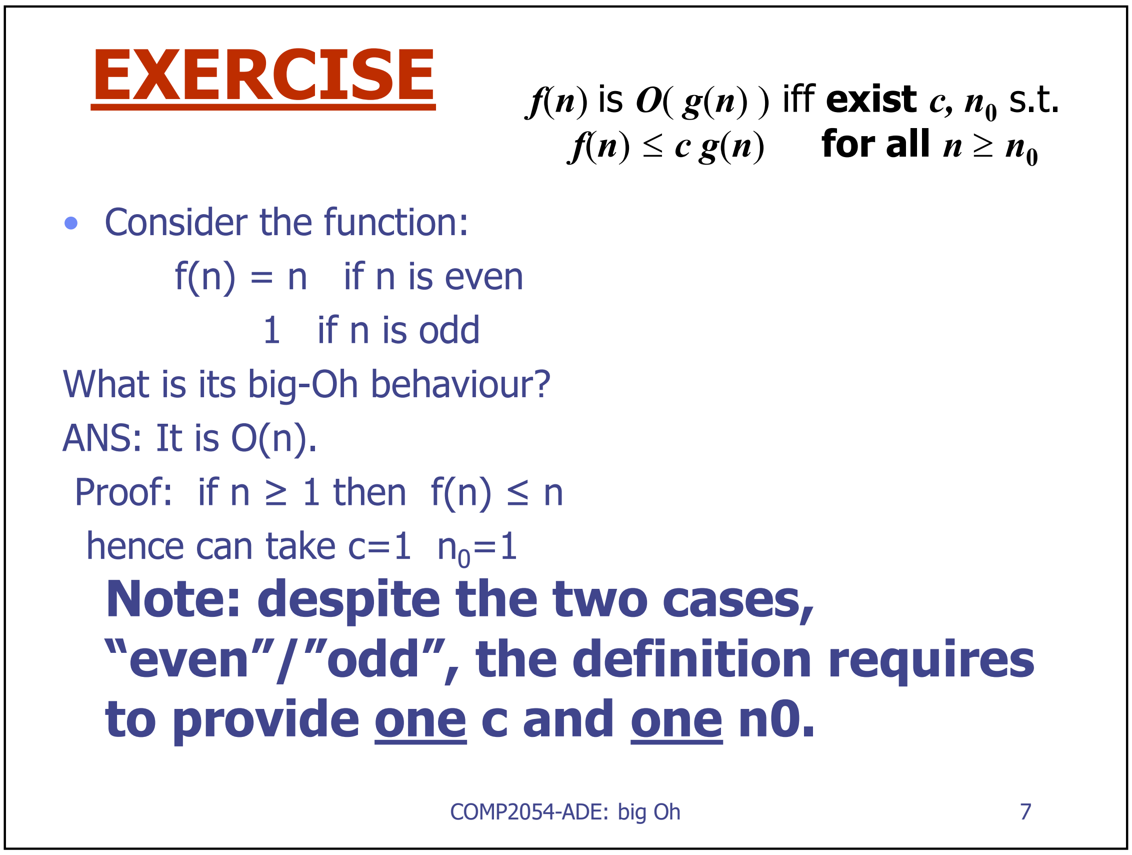
Proof: if $n \ge 1$ then $f(n) \le n$

hence can take $c=1$ $n_0=1$

As standard the first thing is to just go to the definitions for inspiration and so we need to find some g(n).

By just looking at the formula in this case, we can see that we get f(n) <= n.

Hence, it is straightforward to show that it is O(n).



EXERCISE

$f(n)$ is $O(g(n))$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- Consider the function:

  f(n) = n if n is even
  1 if n is odd

What is its big-Oh behaviour?

ANS: It is O(n).

Proof: if n ≥ 1 then f(n) ≤ n

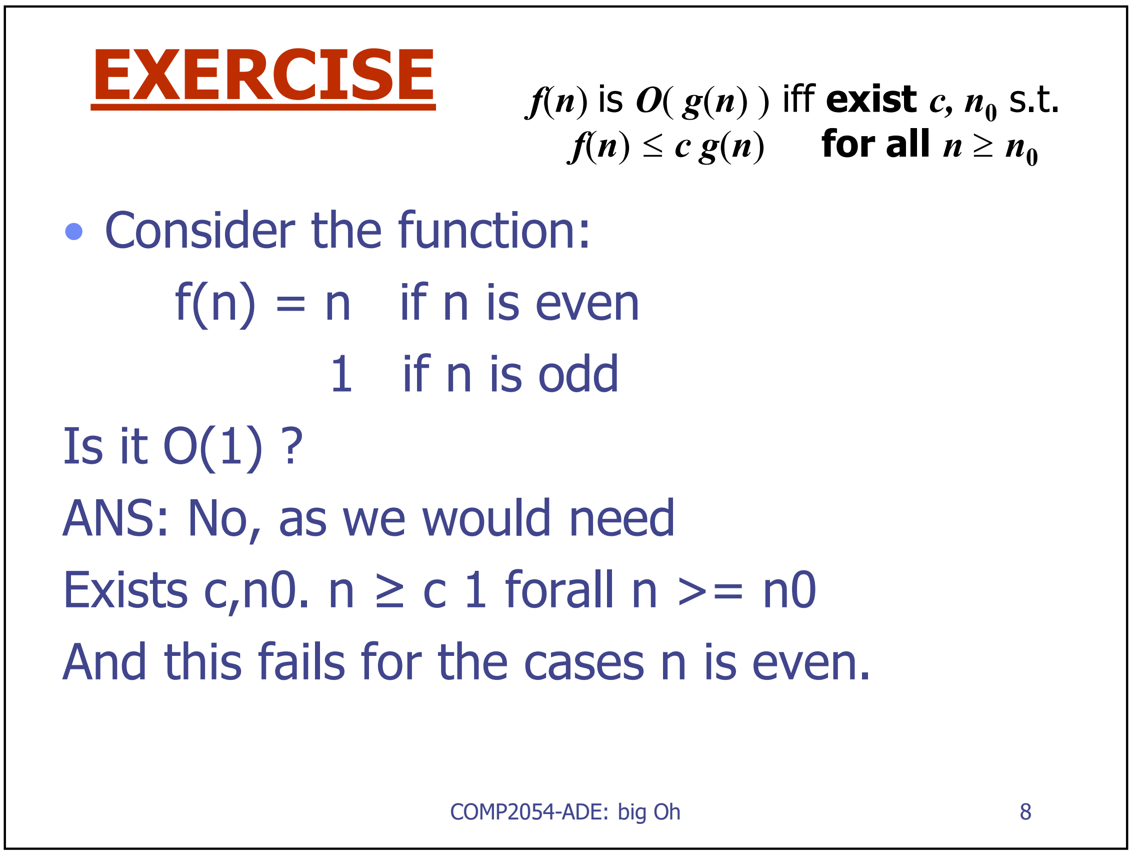
hence can take c=1 n_0=1

Note: despite the two cases, "even"/"odd", the definition requires to provide one c and one n0.

Firstly, a common error in such cases is to discuss the two cases separately, and not combine into one result.

Note that the definition says just "exists c,n0" and does not allow to do case splitting.

When working towards a proof, one might well do case splitting.
But the final proof "from the definition" must use one value of c, and one value of n0,
because these are the only things accepted by the definition.



EXERCISE

$f(n)$ is $O(g(n))$ iff **exist** c, n_0 s.t. $f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

Is it O(1) ?

ANS: No, as we would need

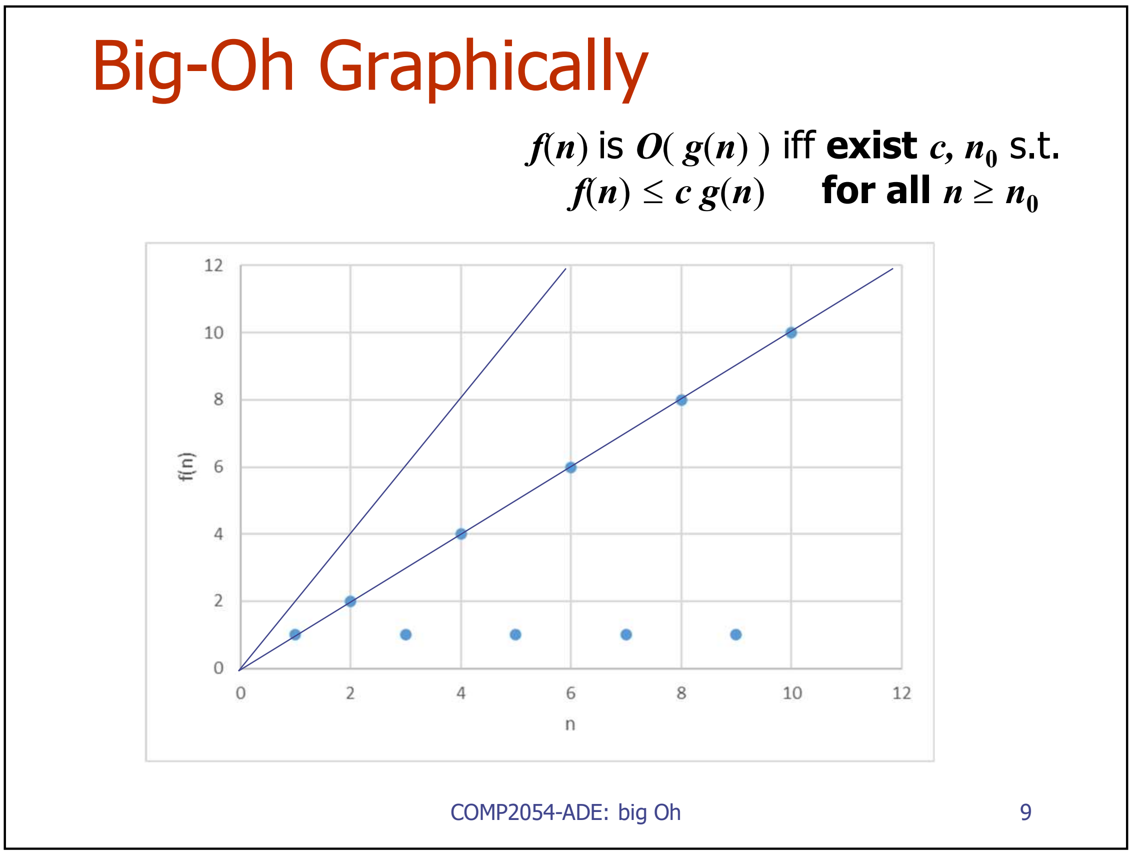
Exists c,n0. n ≥ c 1 forall n >= n0

And this fails for the cases n is even.

As standard the first thing is to just go to the definitions for inspiration and so we need to find some g(n).

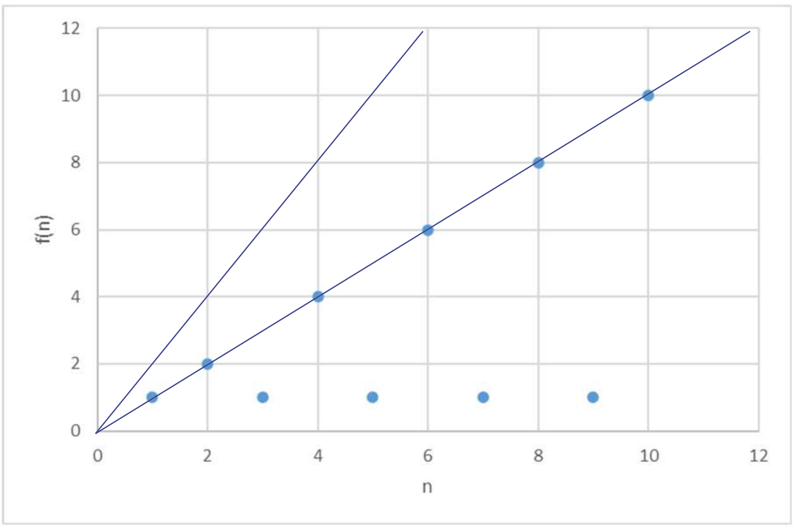
By just looking at the formula in this case, we can see that we get f(n) <= n.

Hence, it is straightforward to show that it is O(n).



Big-Oh Graphically

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \le c\ g(n)$ **for all** $n \ge n_0$



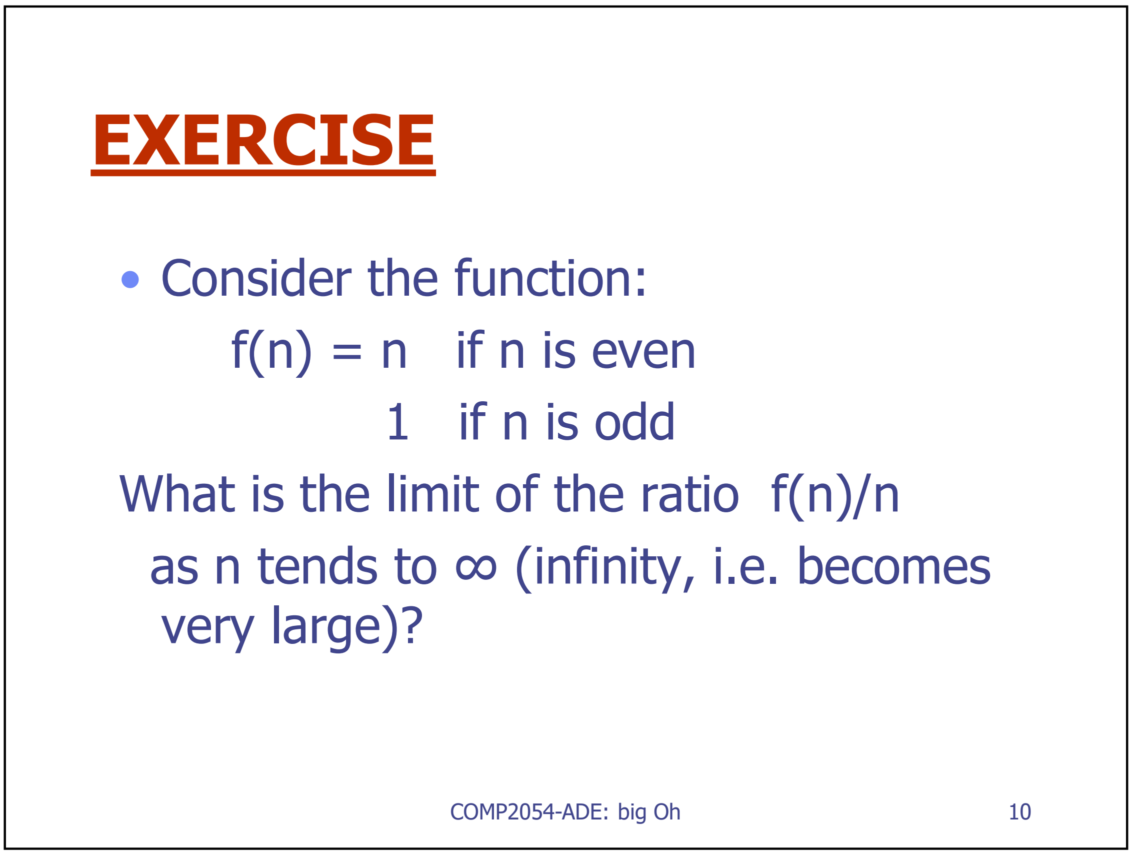
We can also view the O(n) graphically.
The dots are the f(n).

But we can put the line "g(n)=n".

(In PowerPoint: "Insert"->"Shapes"->"Lines".)

Then see that no dots fall above that that line.

This graphically demonstrates how big-Oh is just "put a function above the data".



EXERCISE

- Consider the function:

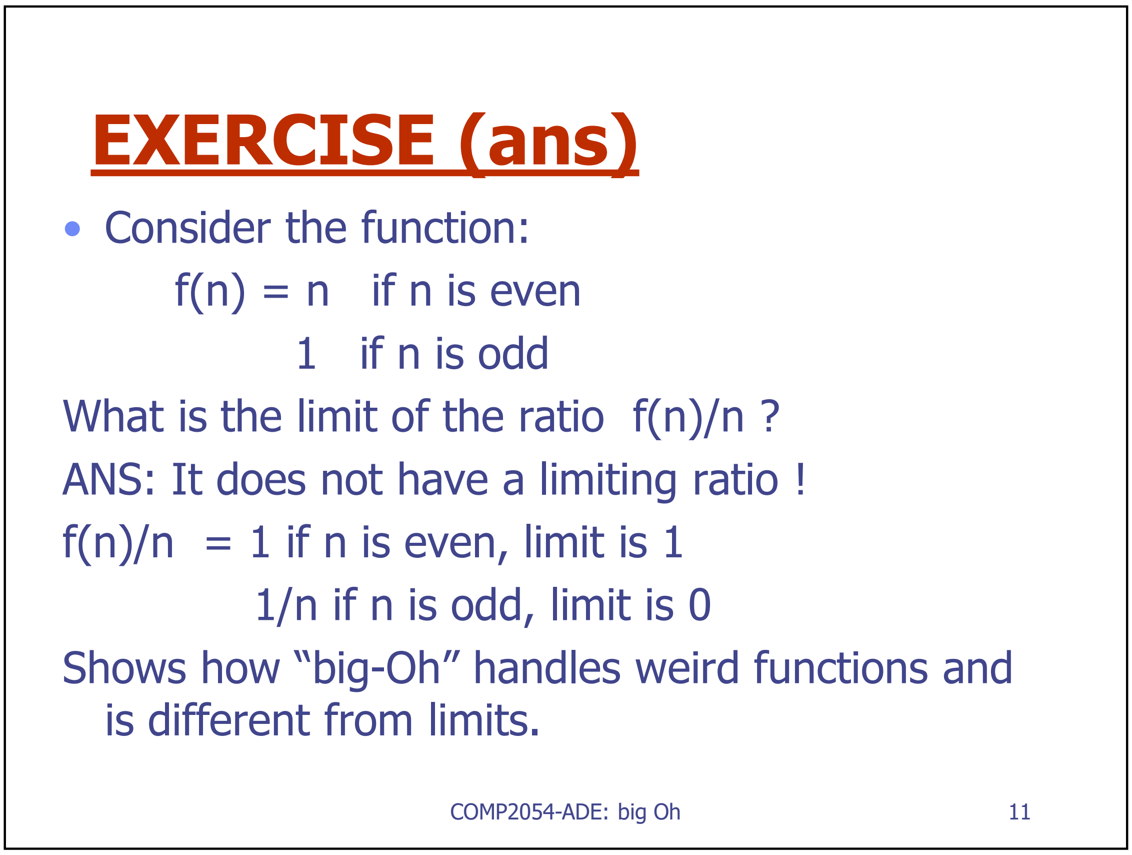
  f(n) = n if n is even

  1 if n is odd

  What is the limit of the ratio f(n)/n as n tends to ∞ (infinity, i.e. becomes very large)?

In the first part of the lectures we motivated the big-Oh using a study of the ratio of two functions, one for f, and one for g.

This corresponds here to looking at the ratio f(n)/g(n), and asking what happens to it when n goes to infinity, i.e. when n is arbitrarily large.



EXERCISE (ans)

- Consider the function:

  f(n) = n   if n is even

  1   if n is odd

What is the limit of the ratio f(n)/n ?

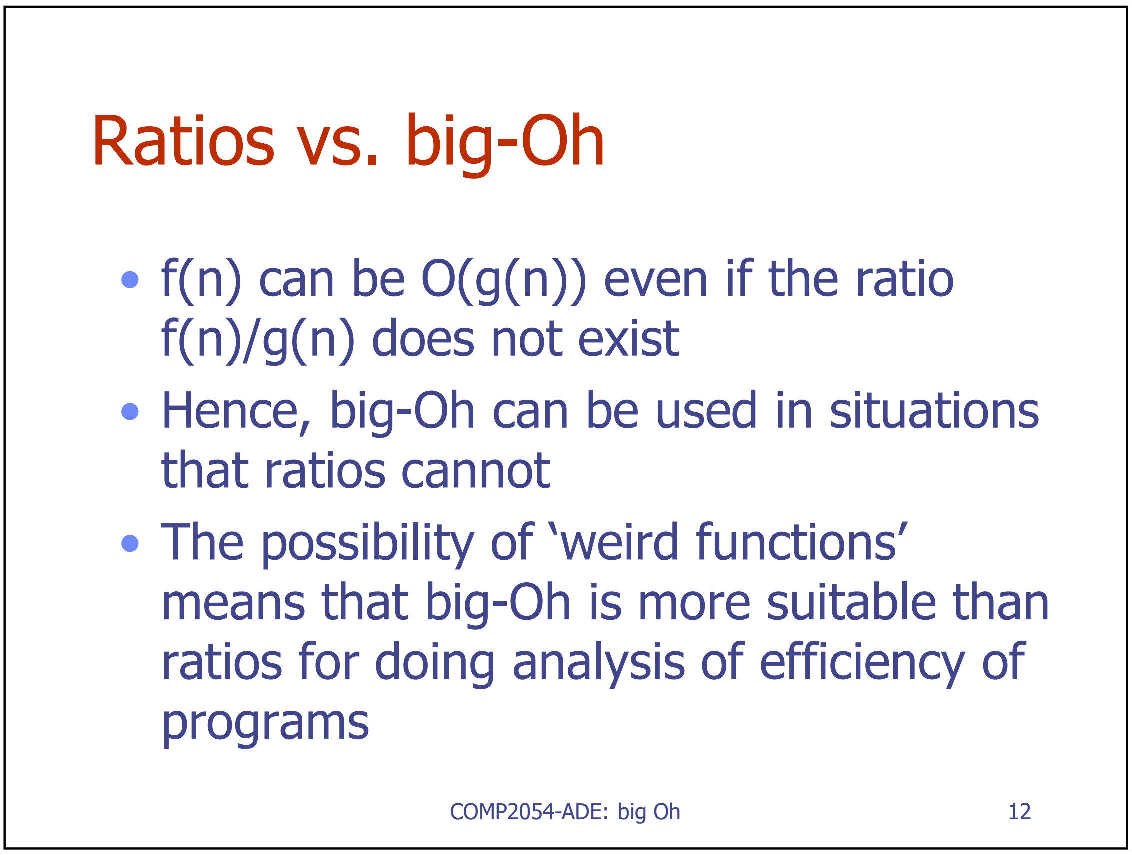
ANS: It does not have a limiting ratio !

f(n)/n = 1 if n is even, limit is 1

1/n if n is odd, limit is 0

Shows how "big-Oh" handles weird functions and is different from limits.

Well it clearly does not have a unique limit.
The even case has limit 1, but the odd case as limit 0.
Only using limits would be a lot more restrictive and would not be as generally appropriate for usage in CS.

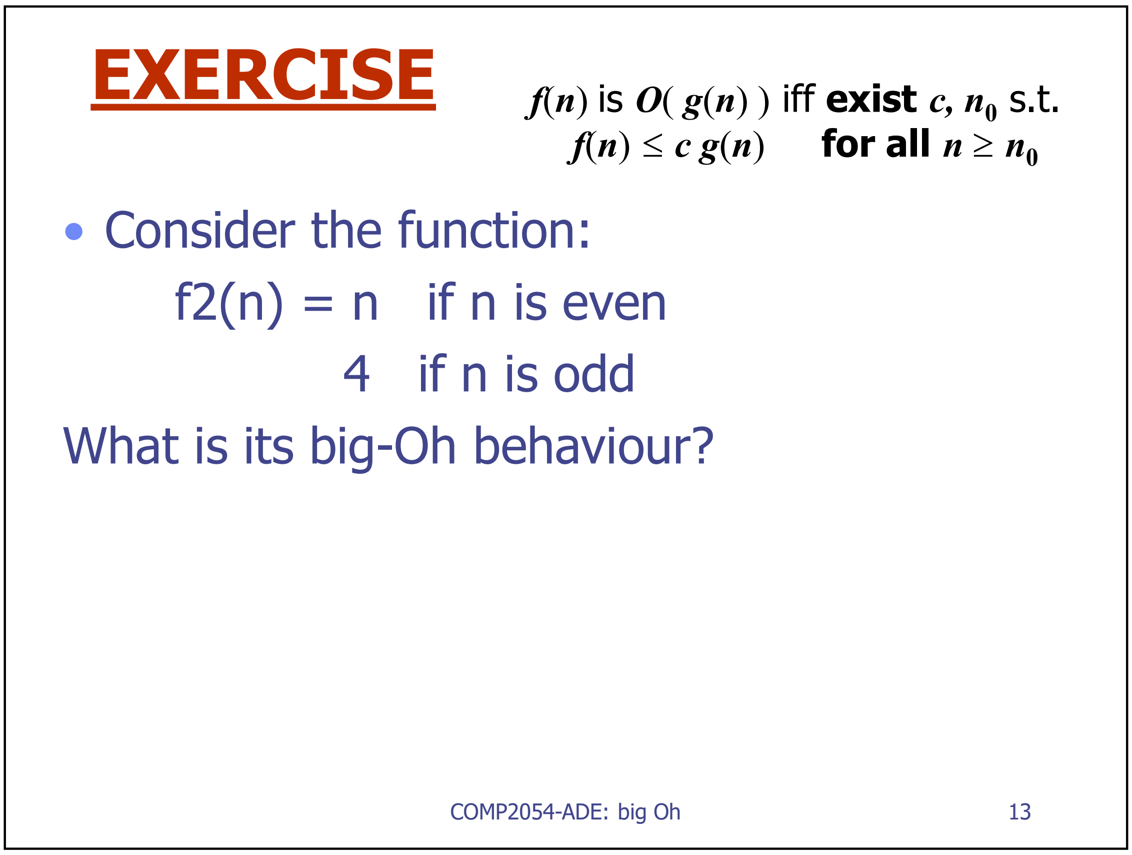


Ratios vs. big-Oh

- f(n) can be O(g(n)) even if the ratio f(n)/g(n) does not exist
- Hence, big-Oh can be used in situations that ratios cannot
- The possibility of ‘weird functions’ means that big-Oh is more suitable than ratios for doing analysis of efficiency of programs

The point is important, and so this slide is just to emphasise it again.

Generally, the point is that Big-Oh family is the best suited for CS



EXERCISE

$f(n)$ is $O(\, g(n)\,)$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

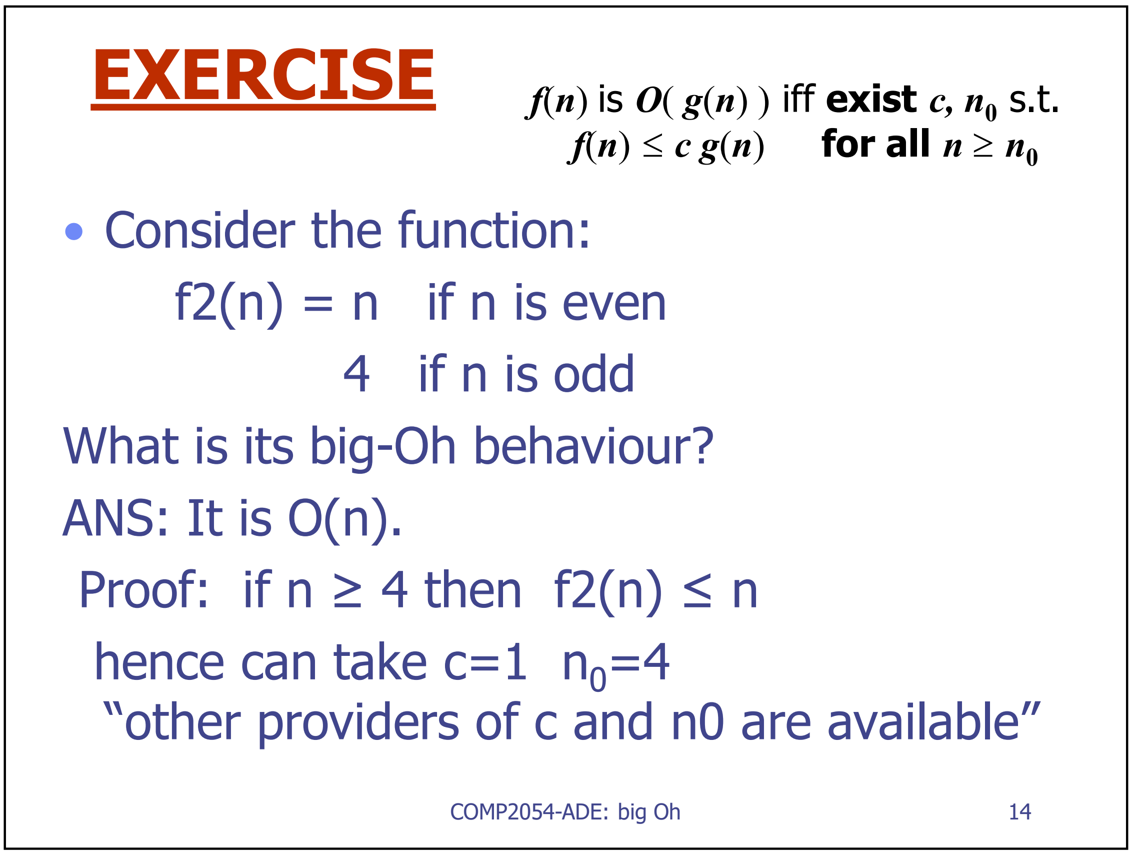
- Consider the function:

  f2(n) = n if n is even
  4 if n is odd

What is its big-Oh behaviour?

COMP2054-ADE: big Oh 13

Consider a slight variant, in which the odd case is now not one, but a larger constant.
What is the big-Oh behaviour.



EXERCISE

$f(n)$ is $O(g(n))$ iff **exist** c, n_0 s.t.
$f(n) \leq c\, g(n)$ **for all** $n \geq n_0$

- Consider the function:

$$f2(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 4 & \text{if } n \text{ is odd} \end{cases}$$

What is its big-Oh behaviour?

ANS: It is O(n).

Proof: if $n \geq 4$ then $f2(n) \leq n$

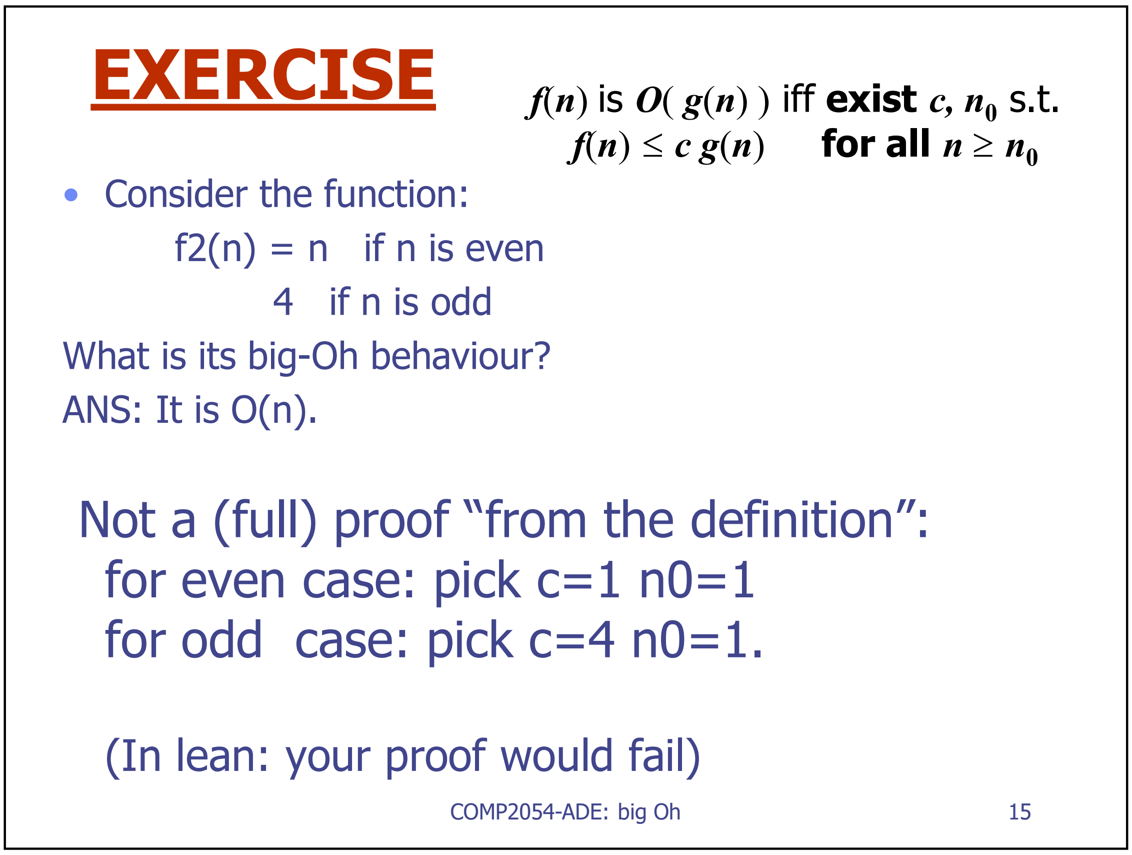
hence can take $c=1$ $n_0=4$

"other providers of c and n0 are available"

We can do a very similar observation to easily show that it is still O(n).

But this case, after the natural choice of c=1, we needed to pick n0 to be at least 4.

Given these choices then the arithmetic of the proof is easy.



EXERCISE

$f(n)$ is $O(\,g(n)\,)$ iff **exist** c, n_0 s.t.
$f(n) \le c\,g(n)$ **for all** $n \ge n_0$

- Consider the function:

  f2(n) = n if n is even

  4 if n is odd

What is its big-Oh behaviour?

ANS: It is O(n).

Not a (full) proof "from the definition":

for even case: pick c=1 n0=1

for odd case: pick c=4 n0=1.

(In lean: your proof would fail)

COMP2054-ADE: big Oh 15

As mentioned before, it does not count as a full proof "from the definition" to do a case split and produce separate values of c,n0 for each case.

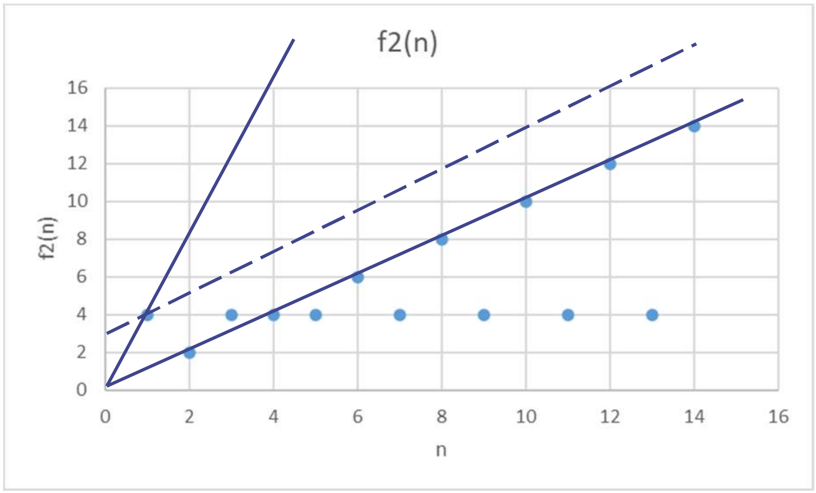
Of course, it is trivial to merge these to give a single c and single n0, but it should be done for completeness.

E.g. can take c=4 for both cases, because increasing the value of c does not break the proof for the even case.

$f(n)$ is $O(\ g(n)\ )$ iff **exist** $c,\ n_0$ s.t.
$f(n) \leq c\ g(n)$ **for all** $n \geq n_0$

EXERCISE

- "Messy at small n" ; fine past n=4



What about graphically analysis again for f2?

(Note: in the version for the recording I had accidentally swapped even/odd, but is fixed above)

In this case, we see the case again of graphs being messy at small n.

Showing that it is O(n) is simply a matter of finding a straight line that can be put above all the data points past a given value of n.
We can do it in two ways

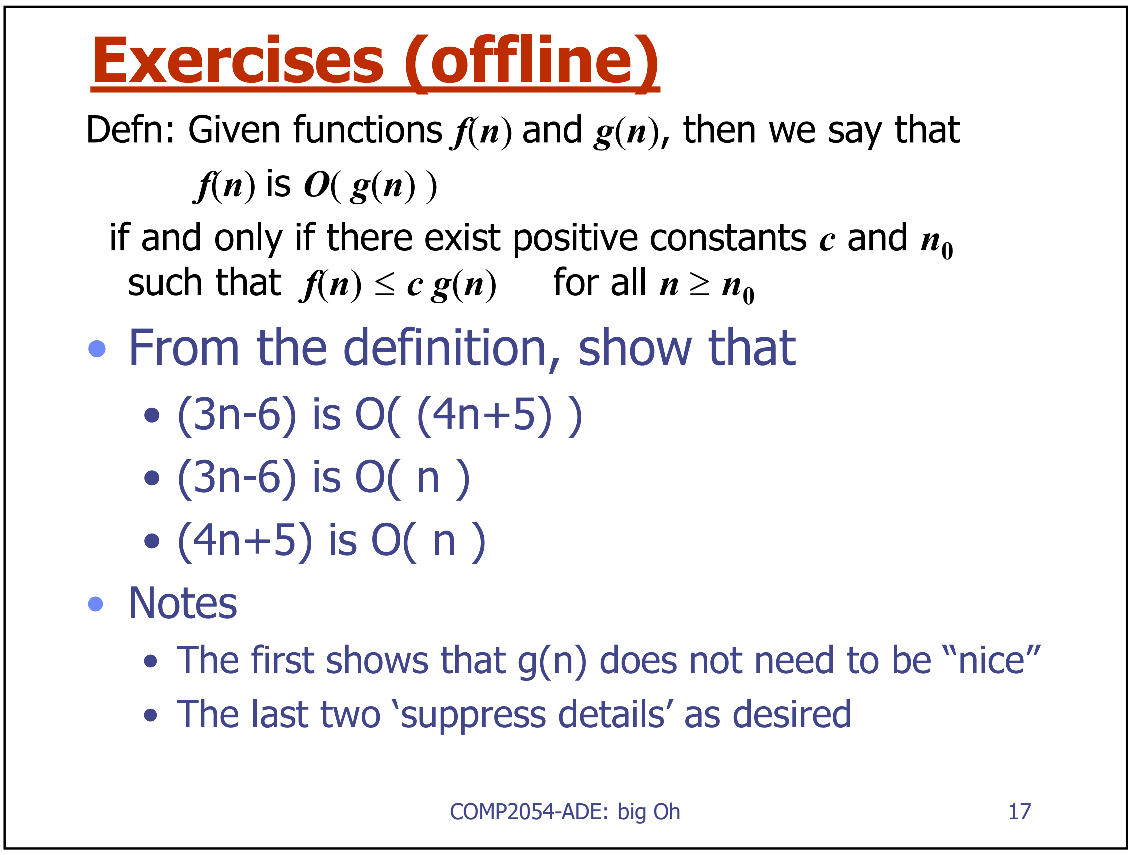
1 * n for n >= 4

4 * n for n>= 1

Corresponding to the two solid lines on the graph.

Strongly recommend: reproduce this yourself in Excel – or any other software to produce graphs.

The dashed line is n+3 and also lies above the data, and so does show the data has linear growth;
but it cannot be directly used for showing the data is O(n).
Can show that the data is O(n+3) and then a separate (easy) step is needed to show that it is hence also O(n) as n+3 is O(n).



Exercises (offline)

Defn: Given functions $f(n)$ and $g(n)$, then we say that

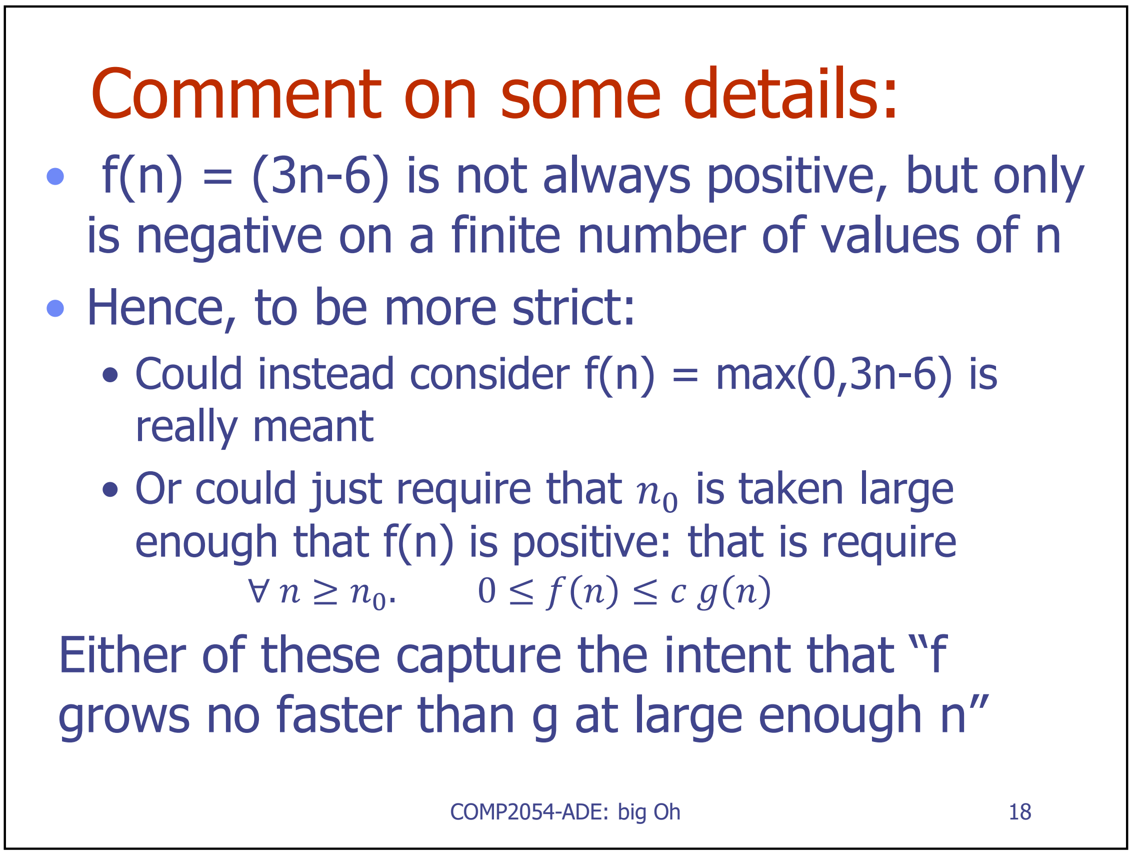
$f(n)$ is $O(\ g(n)\ )$

if and only if there exist positive constants c and n_0 such that $f(n) \le c\, g(n)$ for all $n \ge n_0$

- From the definition, show that
  - (3n-6) is O((4n+5))
  - (3n-6) is O(n)
  - (4n+5) is O(n)
- Notes
  - The first shows that g(n) does not need to be "nice"
  - The last two 'suppress details' as desired

COMP2054-ADE: big Oh 17

So here are a few more exercises, and relating back to the motivating example in the first part of this lecture – lec01a
The point of the first one is that the definition permits to use "messy" formulas for g(n); and it is only a convention to avoid this.



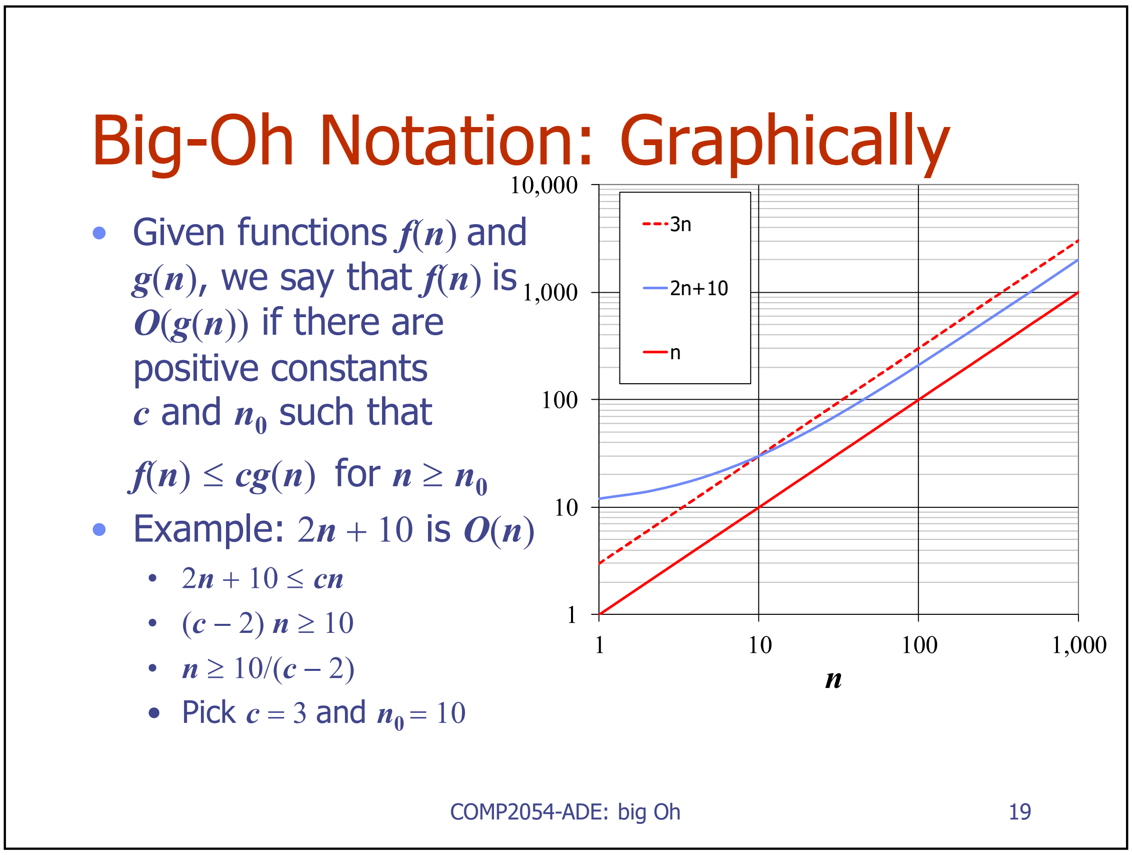
Comment on some details:

- f(n) = (3n-6) is not always positive, but only is negative on a finite number of values of n
- Hence, to be more strict:
  - Could instead consider f(n) = max(0,3n-6) is really meant
  - Or could just require that n_0 is taken large enough that f(n) is positive: that is require

$$\forall\, n \geq n_0. \qquad 0 \leq f(n) \leq c\, g(n)$$

Either of these capture the intent that "f grows no faster than g at large enough n"

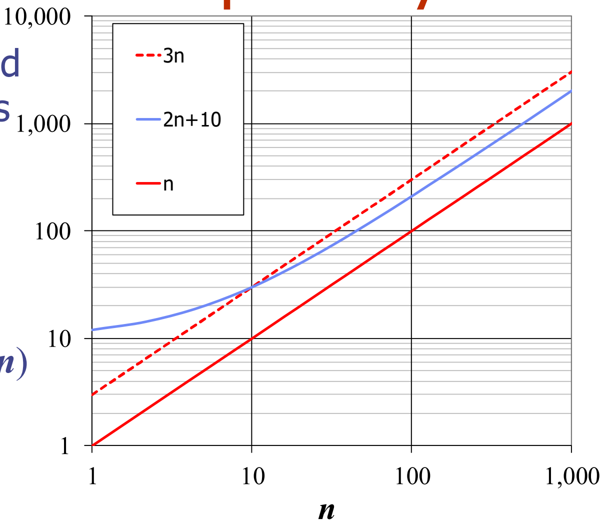
Also, in the definition, to be more exact, instead of "positive functions" we could just require that it is positive after some value of n.



Big-Oh Notation: Graphically

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for $n \geq n_0$
- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$



COMP2054-ADE: big Oh 19

It is also important and helpful to be able to think in terms of graphs of functions in general.

Here we see a log-log plot – the axes are on a log scale.

These are useful because a power law is then a straight line.

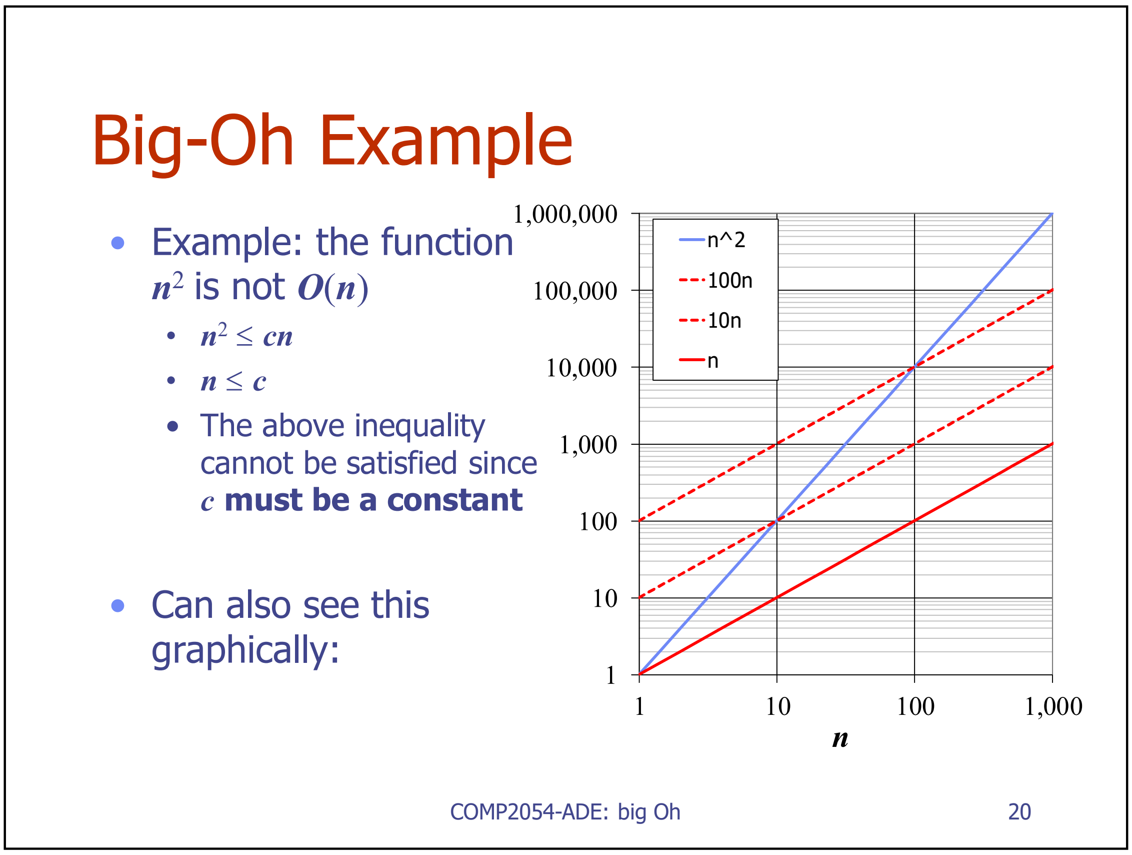
Suppose y = n^k then log y = k * log n

Showing that in terms of log y as a function of log n, we have a linear relation and the slope is k.

It is highly recommended to plot this yourself if at all confused.

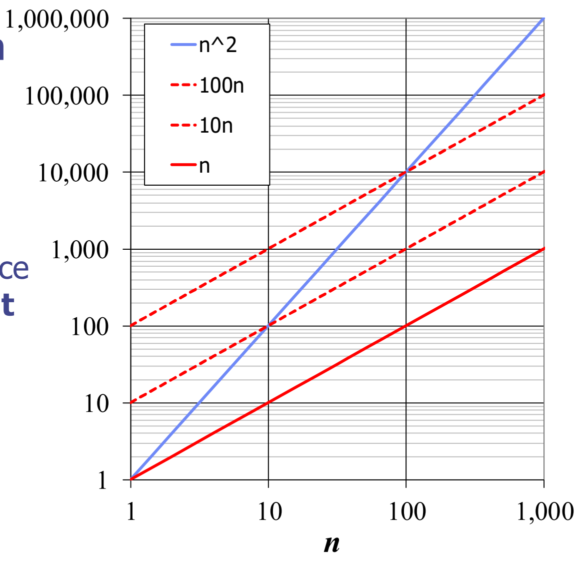
When n is very large then 2n+10 also becomes ever close to a straight line, and one can see that the line 3n always falls above it once n is more than 10.

This is the graphical equivalent of the proof.



Big-Oh Example

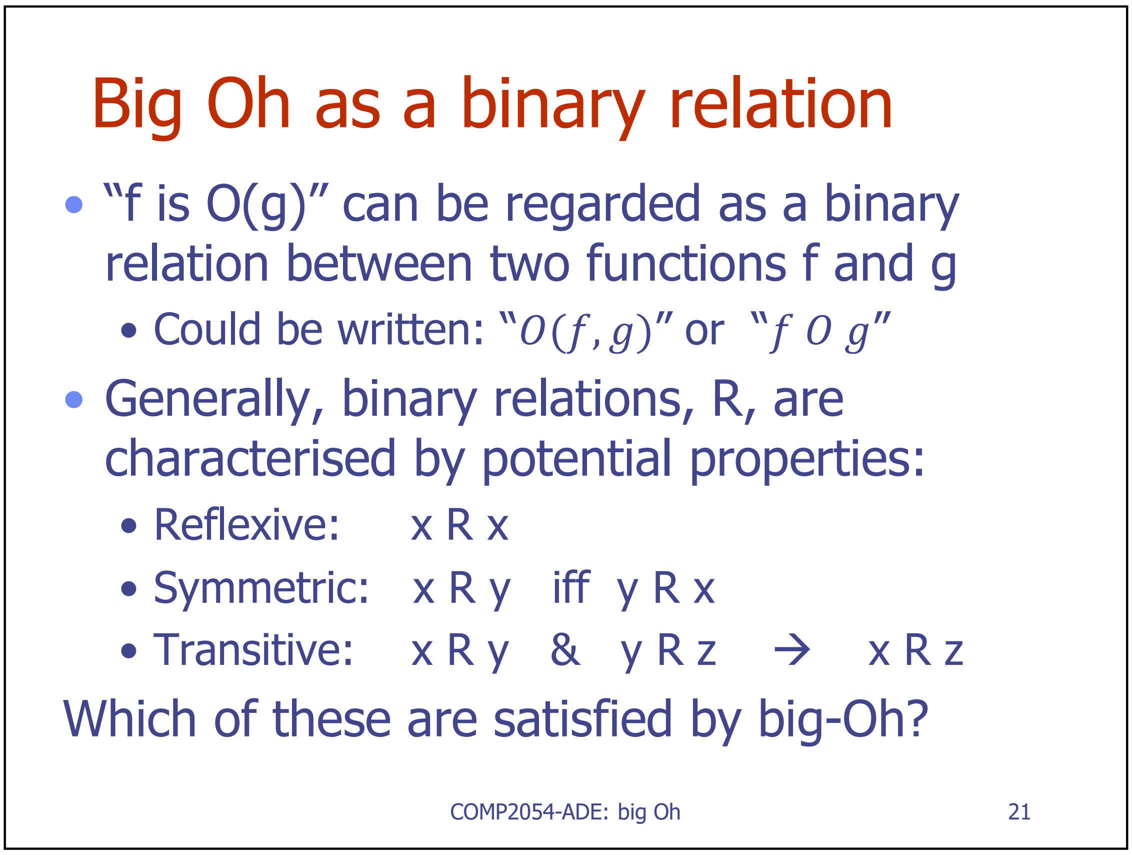
- Example: the function n^2 is not $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - The above inequality cannot be satisfied since c **must be a constant**
- Can also see this graphically:



If we consider f(n) = n^2, then the slope is double that of any linear function.

Since log(cn) = log(c) + log(n) then increasing c moves the line upwards, but does not change the slope.

Hence it is graphically clear that the n^2 will eventually be larger than any cn. Hence the n^2 is not O(n).



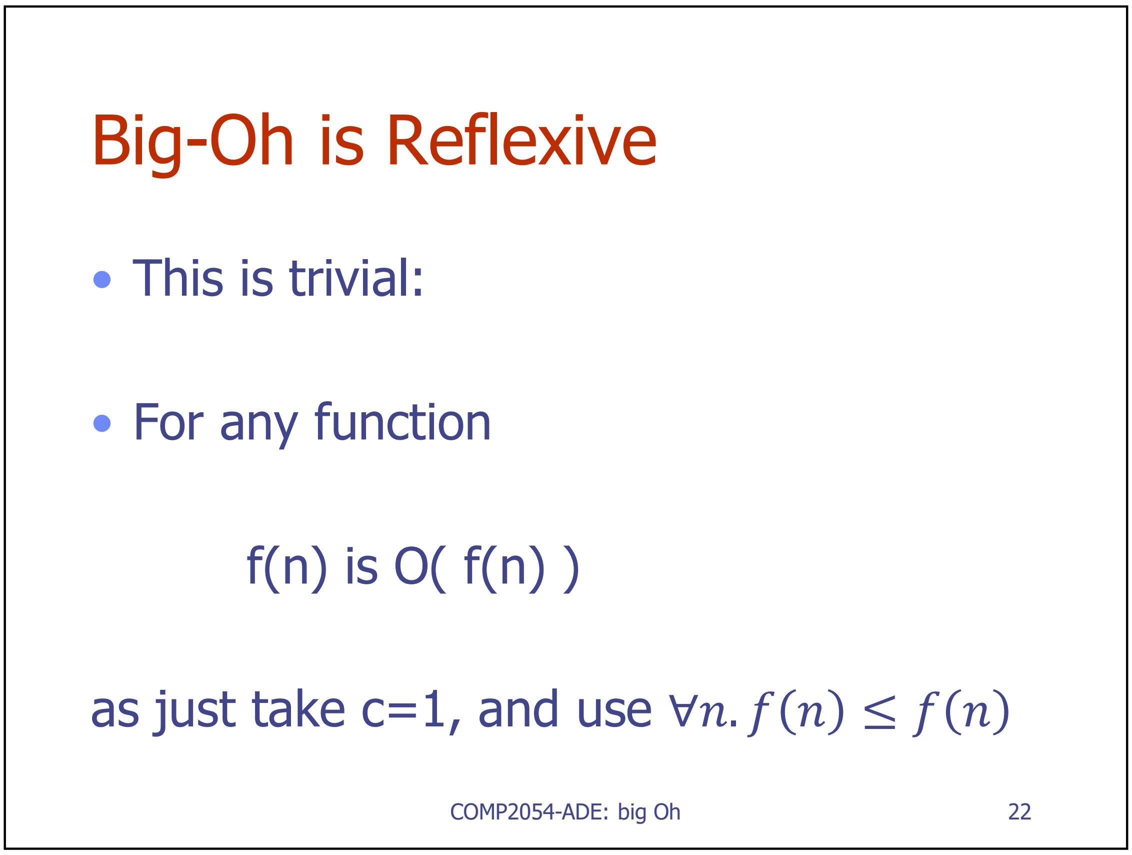
Big Oh as a binary relation

- "f is O(g)" can be regarded as a binary relation between two functions f and g
  - Could be written: "$O(f,g)$" or "$f\ O\ g$"
- Generally, binary relations, R, are characterised by potential properties:
  - Reflexive: x R x
  - Symmetric: x R y iff y R x
  - Transitive: x R y & y R z → x R z

Which of these are satisfied by big-Oh?

Now think of big Oh as just defining a binary relation between two functions with O(f,g) just being the same as "f is O(g)".

We can then ask the standard questions about binary relations.



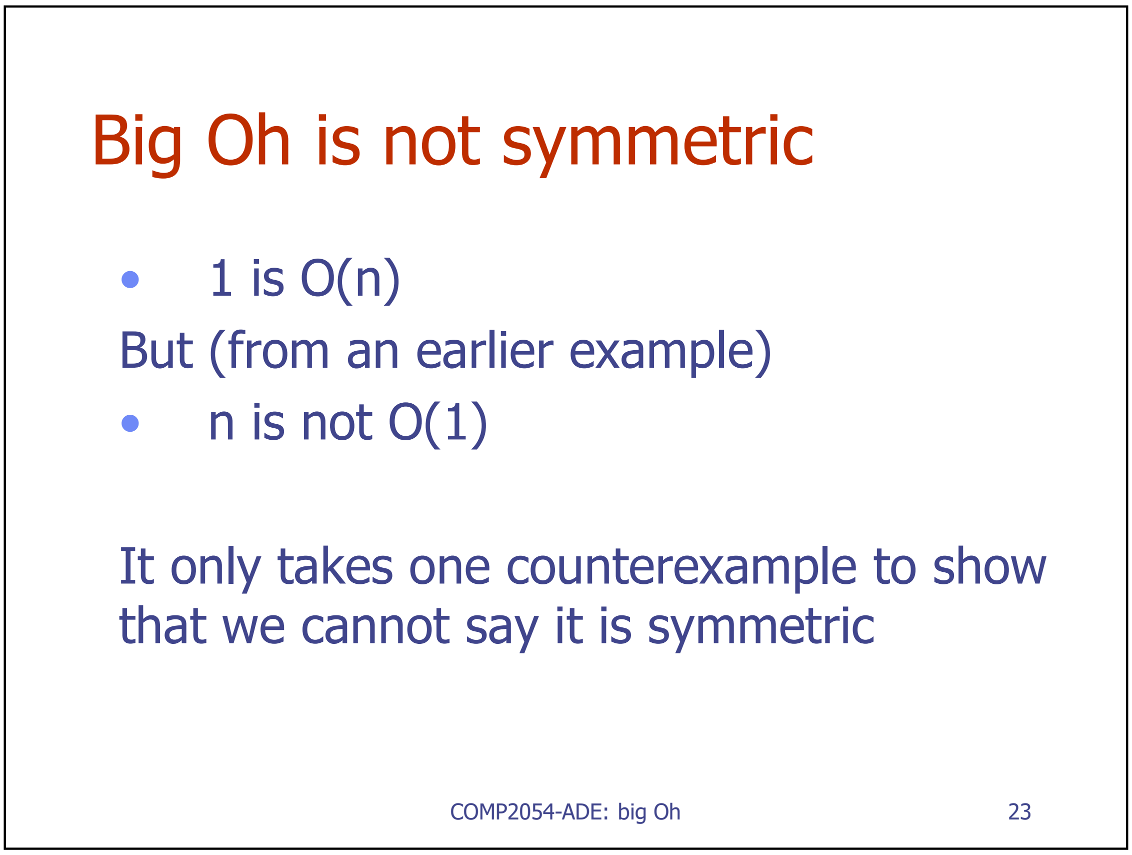
Big-Oh is Reflexive

- This is trivial:
- For any function

f(n) is O(f(n))

as just take c=1, and use $\forall n. f(n) \leq f(n)$

Firstly, it is straightforward that it is reflexive because "O(f,f)" is always true, but simply taking c=1 and any value for n0.



Big Oh is not symmetric

- 1 is O(n)

But (from an earlier example)

- n is not O(1)

It only takes one counterexample to show that we cannot say it is symmetric

COMP2054-ADE: big Oh 23

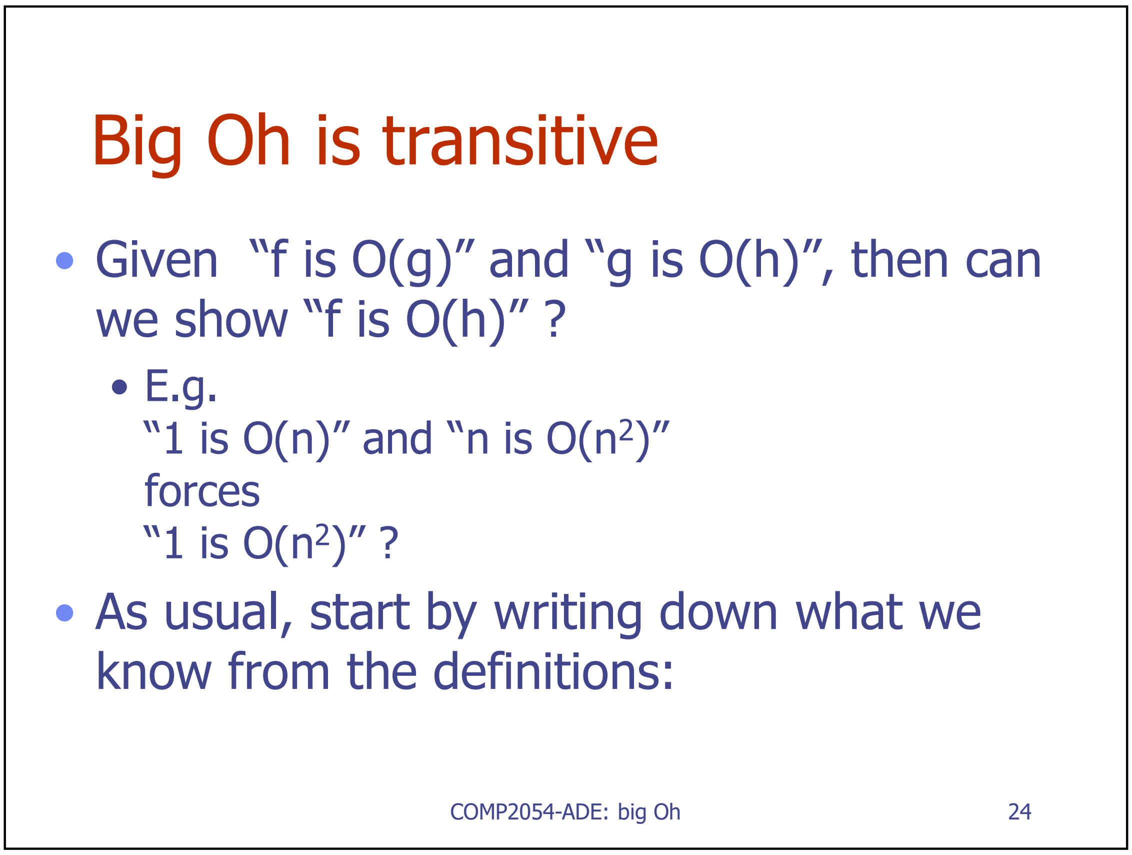
However, it is not symmetric.
Just using the examples from before where we take

f(n) = 1

g(n) = n

Then switching f and g changes the answer.
"f is O(g)" is not always the same as "g is O(f)".
Of course it can be in some circumstances, e.g.

"2n is O(3n)" and "3n is O(2n)" but for it to be symmetric it would need to always be true.

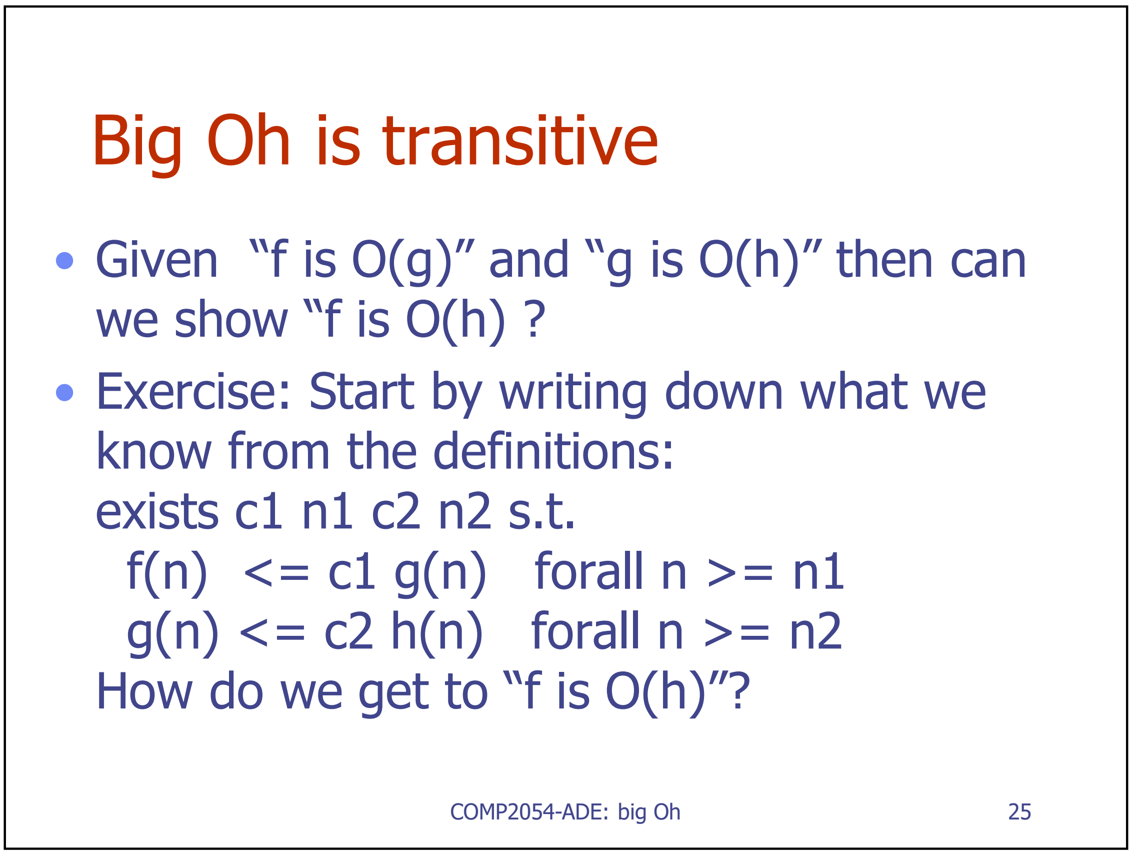


Big Oh is transitive

- Given "f is O(g)" and "g is O(h)", then can we show "f is O(h)" ?
  - E.g.
    "1 is O(n)" and "n is O(n^2)"
    forces
    "1 is O(n^2)" ?
- As usual, start by writing down what we know from the definitions:

Finally a standard question about binary relations is whether they are transitive – can we "chain them together"?
This makes sense because it just says that "f grows no faster than g" and "g grows no faster than h" and so naturally we should get that

"f grows no faster than h"
As usual the first step is to use the definitions to write down the things we know:

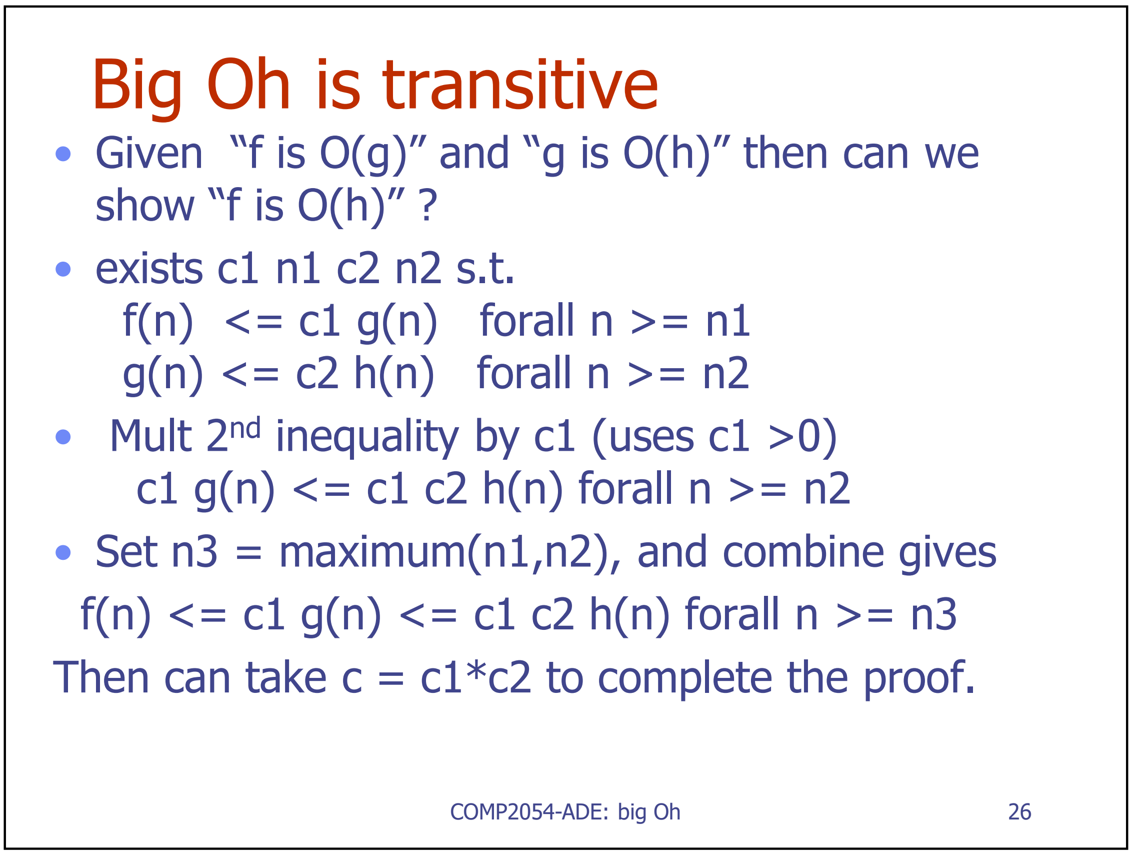


Big Oh is transitive

- Given “f is O(g)” and “g is O(h)” then can we show “f is O(h) ?
- Exercise: Start by writing down what we know from the definitions:
  exists c1 n1 c2 n2 s.t.
    f(n) <= c1 g(n)   forall n >= n1
    g(n) <= c2 h(n)   forall n >= n2
  How do we get to “f is O(h)”?

These are just from plugging into the definitions and accounting that the two cases can use different sets of value for c,n0.

Then how do we get to something corresponding to “f is O(h)”?



Big Oh is transitive

- Given "f is O(g)" and "g is O(h)" then can we show "f is O(h)" ?
- exists c1 n1 c2 n2 s.t.
  f(n) <= c1 g(n) forall n >= n1
  g(n) <= c2 h(n) forall n >= n2
- Mult 2nd inequality by c1 (uses c1 >0)
  c1 g(n) <= c1 c2 h(n) forall n >= n2
- Set n3 = maximum(n1,n2), and combine gives

  f(n) <= c1 g(n) <= c1 c2 h(n) forall n >= n3

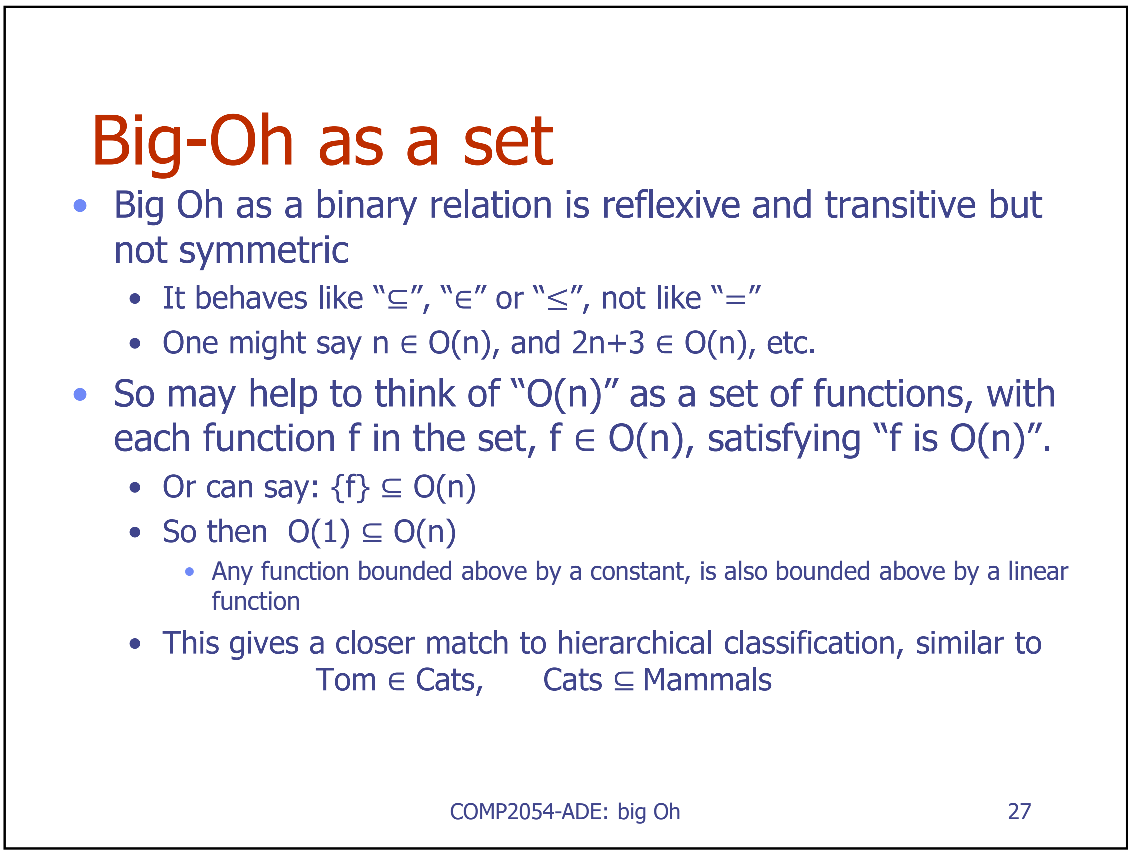
Then can take c = c1*c2 to complete the proof.

We can just multiple the second one by c1 – and using that it will have c1 > 0

(We are assuming known of the functions are the zero function.)

We have to pick a new "n0", called n3, and can pick it to be at least as large as n1 and as large as n2.

Then we just chain together the inequalities, and observe we get the needed proof with c=c1*c2.



Big-Oh as a set

- Big Oh as a binary relation is reflexive and transitive but not symmetric
  - It behaves like "⊆", "∈" or "≤", not like "="
  - One might say n ∈ O(n), and 2n+3 ∈ O(n), etc.
- So may help to think of "O(n)" as a set of functions, with each function f in the set, f ∈ O(n), satisfying "f is O(n)".
  - Or can say: {f} ⊆ O(n)
  - So then O(1) ⊆ O(n)
    - Any function bounded above by a constant, is also bounded above by a linear function
  - This gives a closer match to hierarchical classification, similar to Tom ∈ Cats, Cats ⊆ Mammals

COMP2054-ADE: big Oh 27

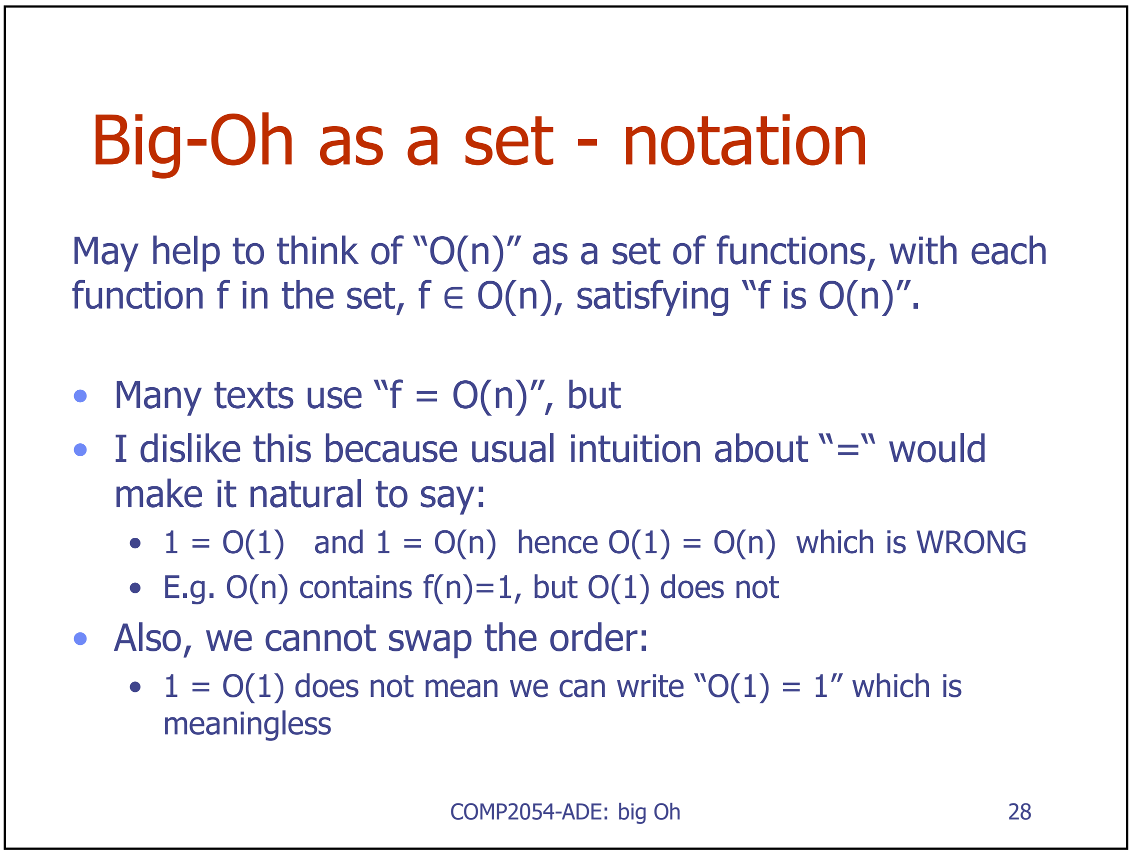
Now think of which binary relations are reflexive and transitive and the natural ones are "subset or equal" or "less than or equal"

Actually, the "subset or equal" will often be strict subsets, but it does not matter here.

So "f is O(g)" is like f is a member of set of functions.
We can effectively regard ""O(g)" as the set of all functions which grow no faster than g at large n.

If f is a member of the set O(g) then it means "f is O(g)".

This means that the big Oh is more like hierarchical classification in biology.
Tom is from "Tom and Jerry" cartoons ☺.



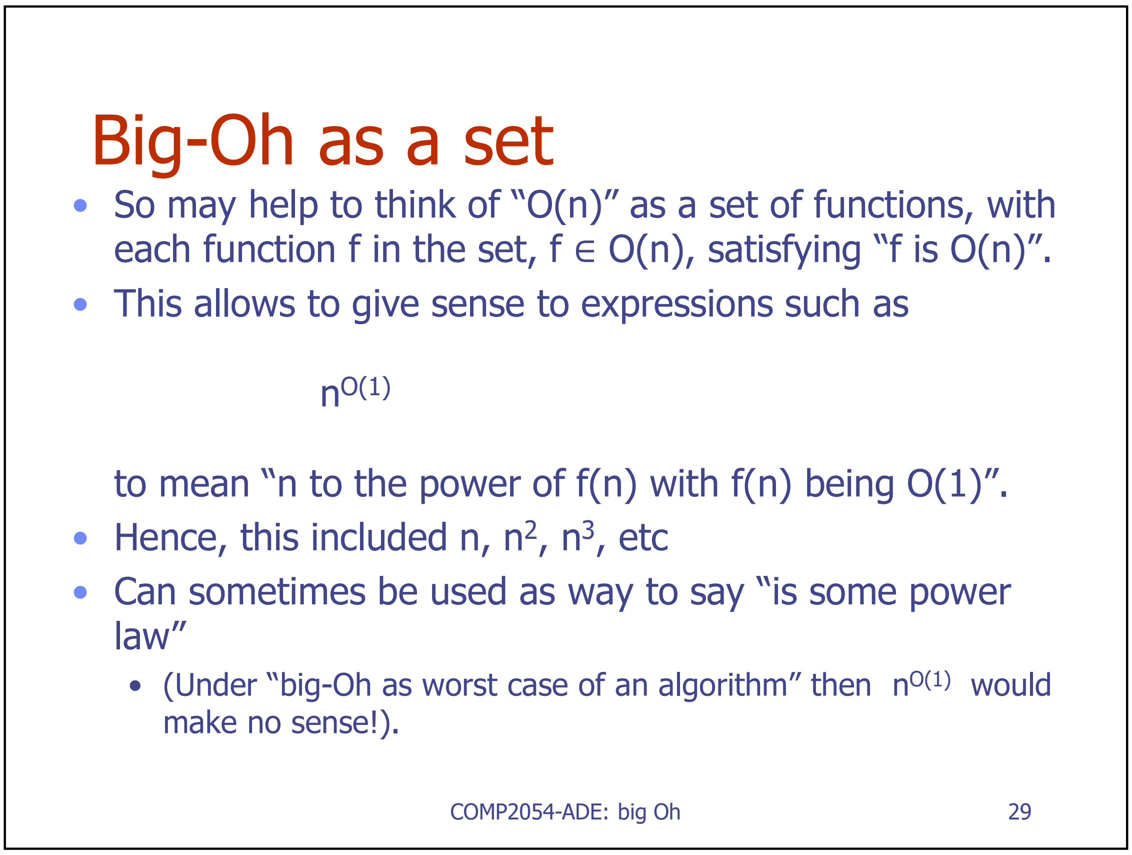
Big-Oh as a set - notation

May help to think of "O(n)" as a set of functions, with each function f in the set, $f \in O(n)$, satisfying "f is O(n)".

- Many texts use "f = O(n)", but
- I dislike this because usual intuition about "=" would make it natural to say:
  - 1 = O(1) and 1 = O(n) hence O(1) = O(n) which is WRONG
  - E.g. O(n) contains f(n)=1, but O(1) does not
- Also, we cannot swap the order:
  - 1 = O(1) does not mean we can write "O(1) = 1" which is meaningless

COMP2054-ADE: big Oh 28

Saying same thing in slightly different words, as the concept is important.



Big-Oh as a set

- So may help to think of "O(n)" as a set of functions, with each function f in the set, $f \in O(n)$, satisfying "f is O(n)".
- This allows to give sense to expressions such as

  $$n^{O(1)}$$

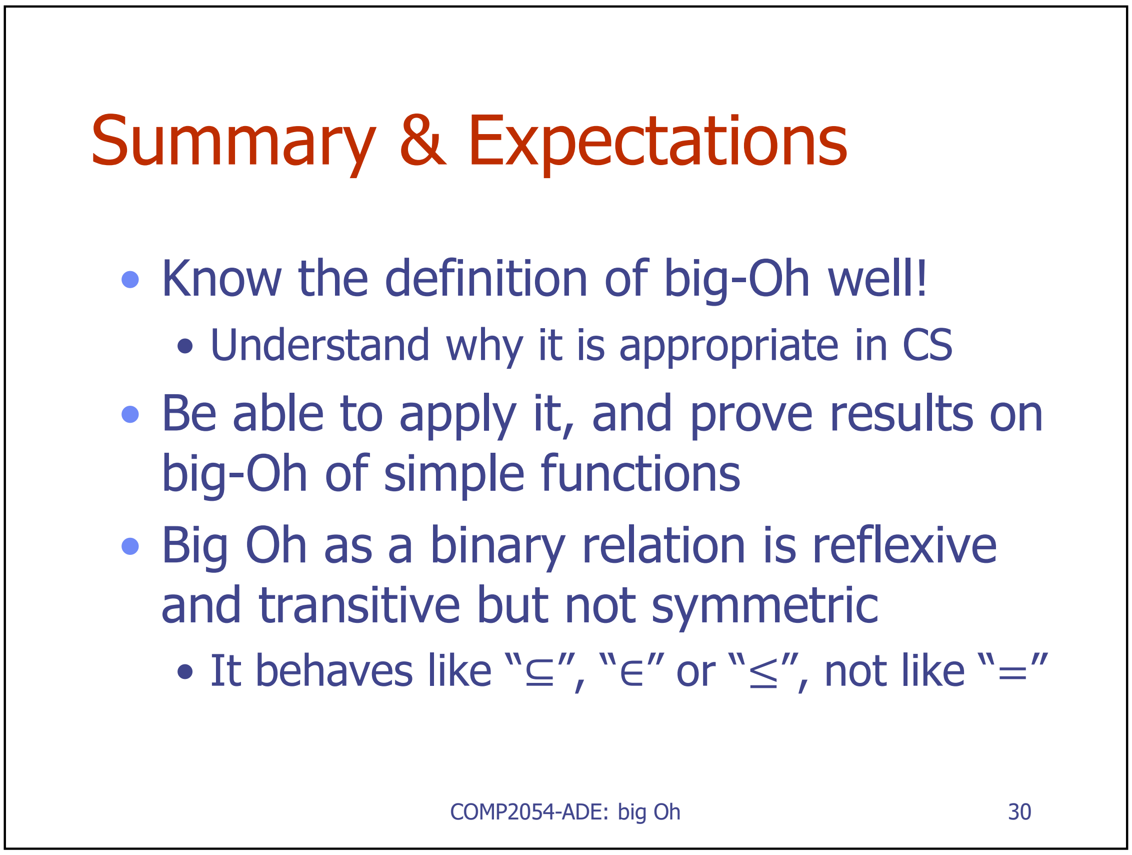
  to mean "n to the power of f(n) with f(n) being O(1)".
- Hence, this included n, n^2, n^3, etc
- Can sometimes be used as way to say "is some power law"
  - (Under "big-Oh as worst case of an algorithm" then $n^{O(1)}$ would make no sense!).

COMP2054-ADE: big Oh 29

This view of taking "O(g)" to be a definition of a set of functions allows more flexibility in usage.

For example, some academic papers refer to power laws as being the expression "n to the O(1)" which is interpreted as given.

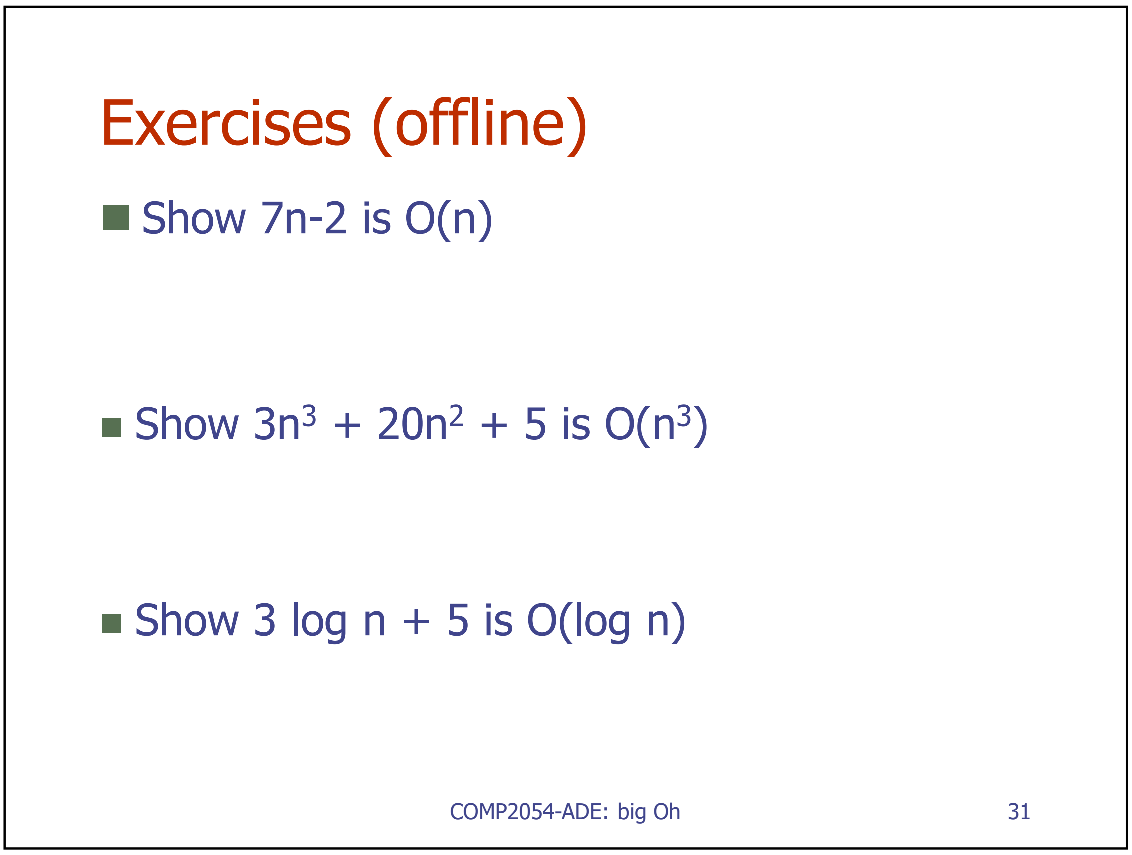
The point here is that limiting oneself to only thinking of big-Oh as being about worst cases of algorithms does not allow the much more powerful usage.



Summary & Expectations

- Know the definition of big-Oh well!
  - Understand why it is appropriate in CS
- Be able to apply it, and prove results on big-Oh of simple functions
- Big Oh as a binary relation is reflexive and transitive but not symmetric
  - It behaves like "$\subseteq$", "$\in$" or "$\leq$", not like "$=$"

This just summarises what have done so far.



Exercises (offline)

- Show 7n-2 is O(n)
- Show $3n^3 + 20n^2 + 5$ is $O(n^3)$
- Show 3 log n + 5 is O(log n)

Some examples for self-study.