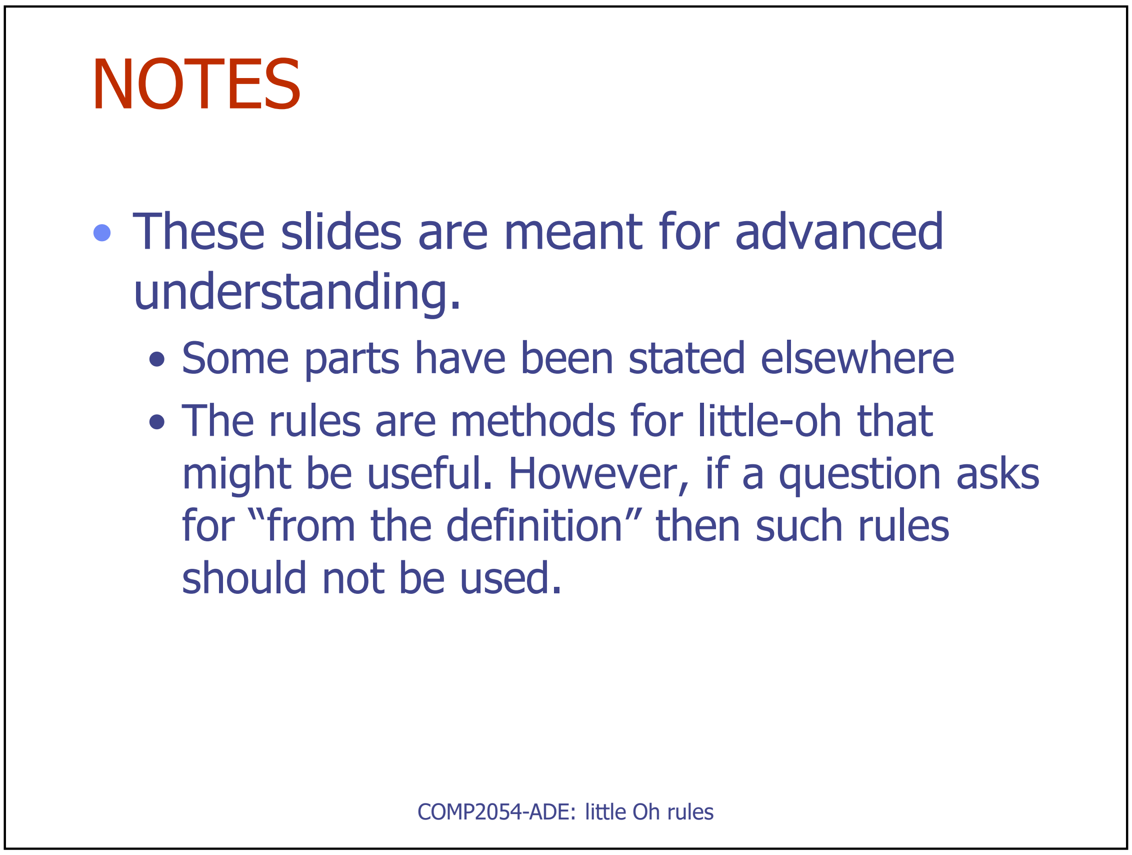
COMP2054-ADE:

ADE Lec05

Rules for little-oh proofs and other advanced usages

Lecturer: Andrew Parkes
andrew.parkes@Nottingham.ac.uk

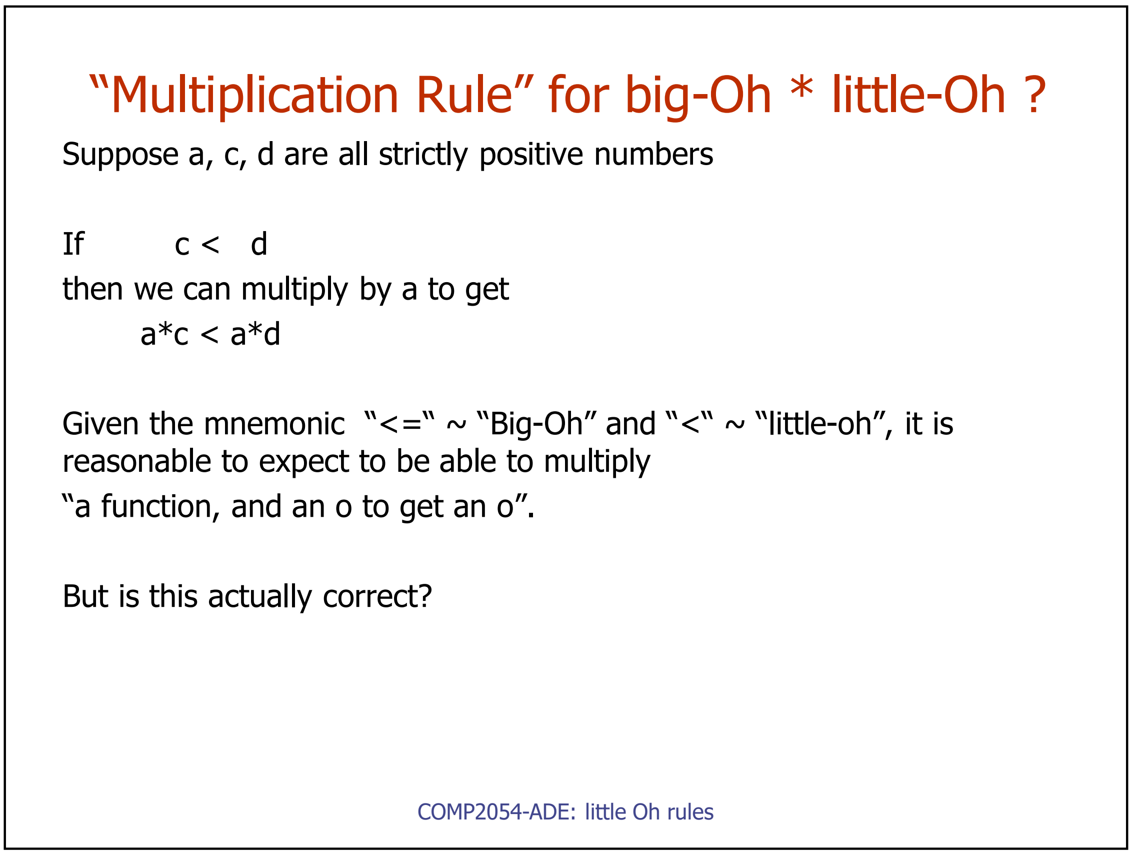
from http://www.cs.nott.ac.uk/~pszajp/



NOTES

- These slides are meant for advanced understanding.
  - Some parts have been stated elsewhere
  - The rules are methods for little-oh that might be useful. However, if a question asks for “from the definition” then such rules should not be used.

As it says, these are just some extra notes, and that are about “rules” to help with doing faster “in your head” computations.



"Multiplication Rule" for big-Oh * little-Oh ?

Suppose a, c, d are all strictly positive numbers

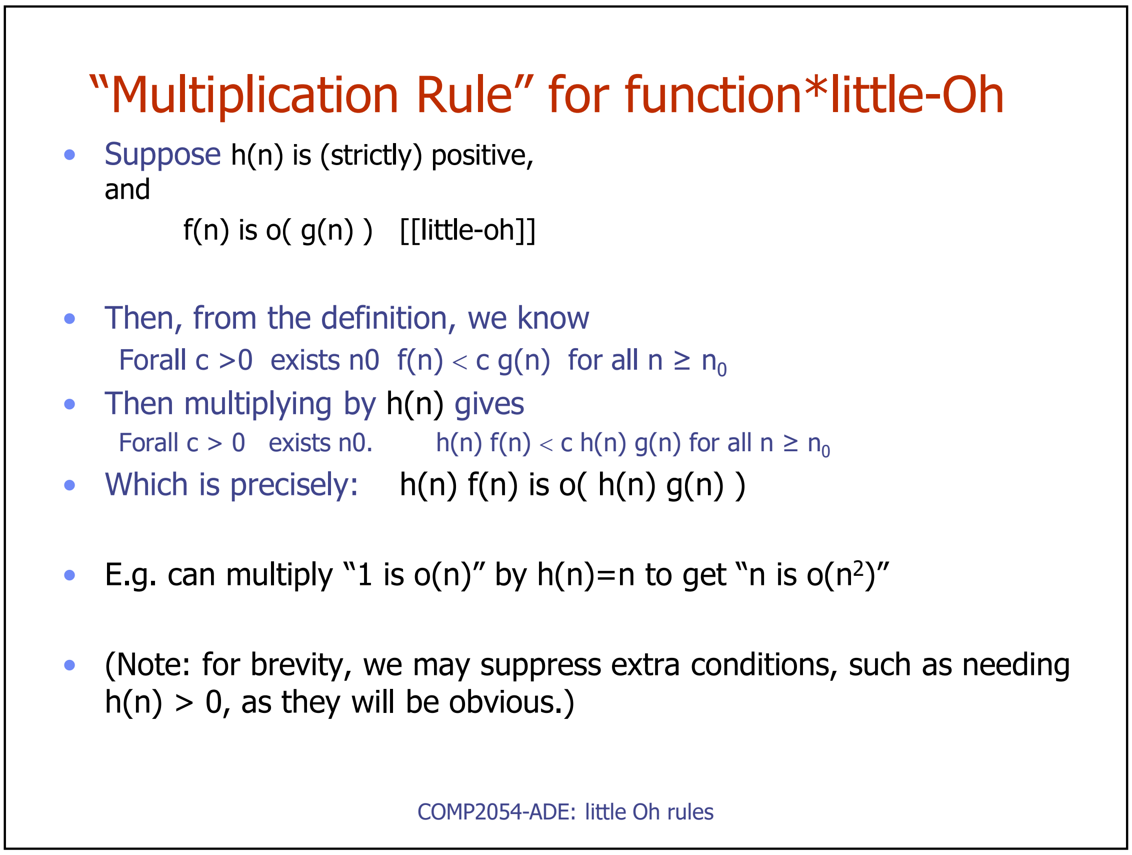
If $c < d$
then we can multiply by a to get
$a*c < a*d$

Given the mnemonic "<=" ~ "Big-Oh" and "<" ~ "little-oh", it is reasonable to expect to be able to multiply
"a function, and an o to get an o".

But is this actually correct?

If we use the analogy of little-oh with "strictly less than" and Big-Oh with "less than or equal", then some rules become very natural.

However, the inequalities are just analogies and so we need to check.



"Multiplication Rule" for function*little-Oh

- Suppose h(n) is (strictly) positive,
  and

  f(n) is o(g(n))   [[little-oh]]

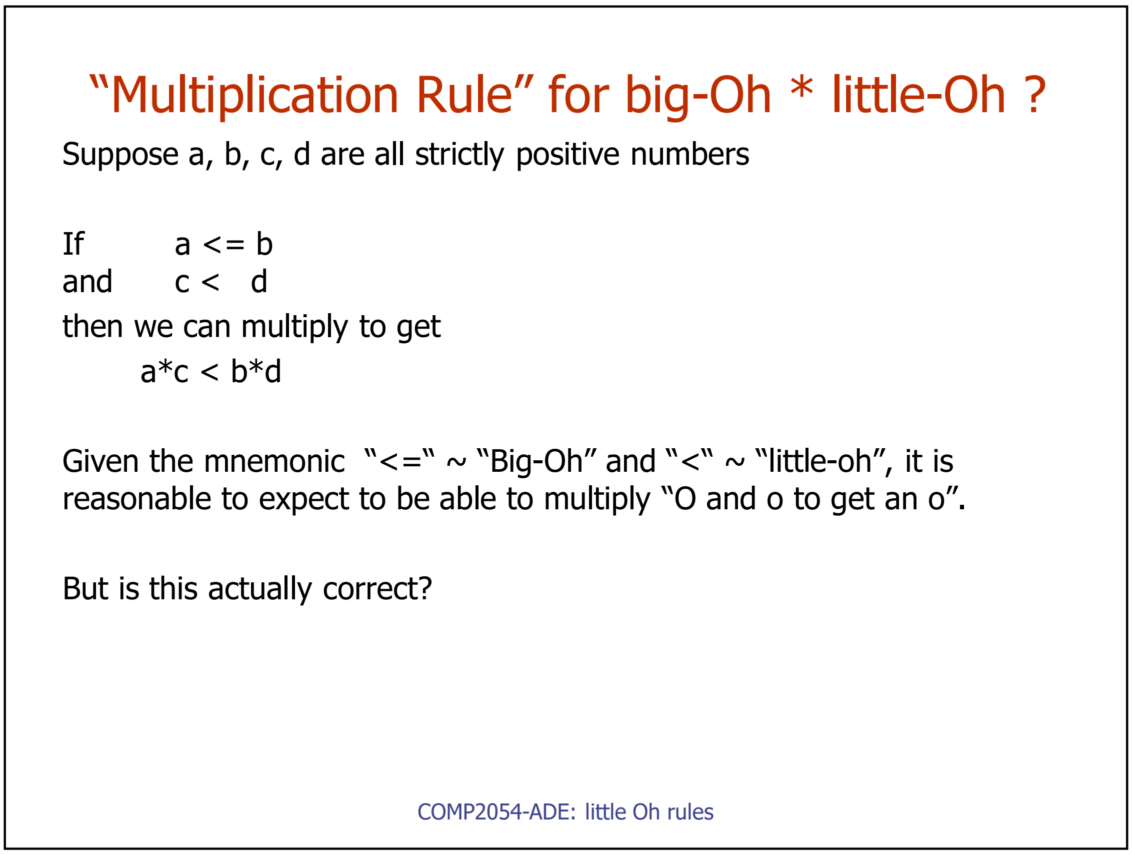
- Then, from the definition, we know

  Forall c >0 exists n0 f(n) < c g(n) for all n ≥ n_0
- Then multiplying by h(n) gives

  Forall c > 0   exists n0.   h(n) f(n) < c h(n) g(n) for all n ≥ n_0
- Which is precisely: h(n) f(n) is o(h(n) g(n))
- E.g. can multiply "1 is o(n)" by h(n)=n to get "n is o(n^2)"
- (Note: for brevity, we may suppress extra conditions, such as needing h(n) > 0, as they will be obvious.)

COMP2054-ADE: little Oh rules

A starting case is just whether we can multiply "both sides by a function".
The intuition here is simply that we can multiply both sides of an inequality by a function and it will still hold as long as it is not zero or negative.
More formally, it is just a matter of writing out from the definition.
Although "obvious" it is still a good idea to write it out fully at least once.



"Multiplication Rule" for big-Oh * little-Oh ?

Suppose a, b, c, d are all strictly positive numbers

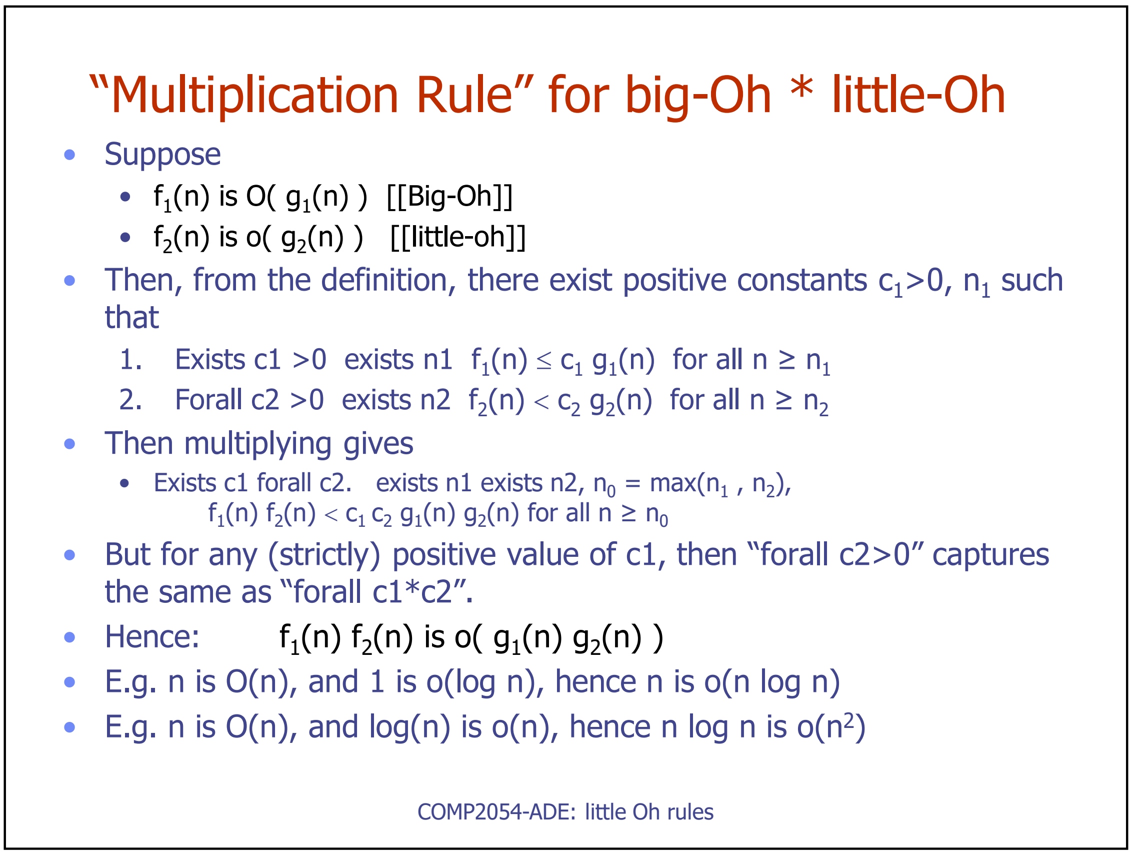
If a <= b
and c < d
then we can multiply to get
a*c < b*d

Given the mnemonic "<=" ~ "Big-Oh" and "<" ~ "little-oh", it is reasonable to expect to be able to multiply "O and o to get an o".

But is this actually correct?

COMP2054-ADE: little Oh rules

In fact, we can get a lot of inspiration from just the analogy with standard inequalities.
A "less than" and a "less than or equal" combine to give a "less than".



"Multiplication Rule" for big-Oh * little-Oh

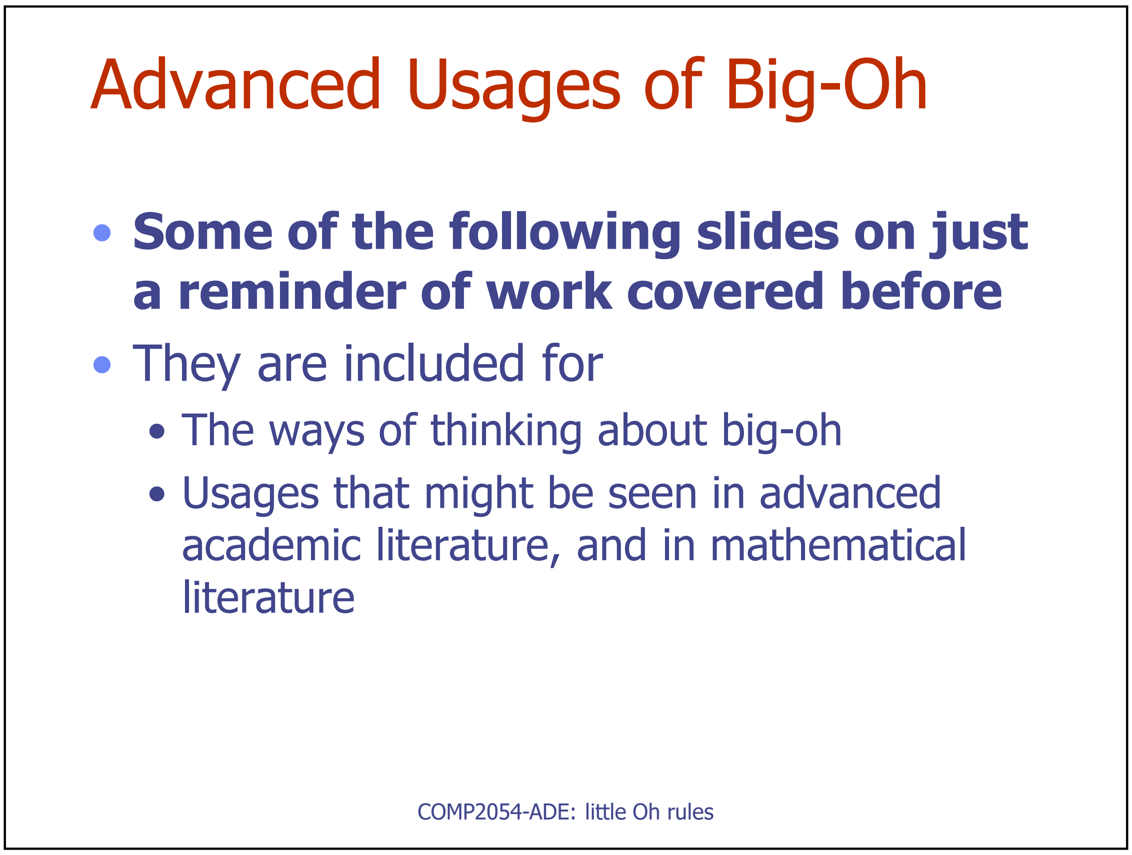
- Suppose
  - $f_1(n)$ is O($g_1(n)$) [[Big-Oh]]
  - $f_2(n)$ is o($g_2(n)$) [[little-oh]]
- Then, from the definition, there exist positive constants $c_1>0$, n_1 such that
  1. Exists c1 >0 exists n1 $f_1(n) \le c_1\, g_1(n)$ for all $n \ge n_1$
  2. Forall c2 >0 exists n2 $f_2(n) < c_2\, g_2(n)$ for all $n \ge n_2$
- Then multiplying gives
  - Exists c1 forall c2. exists n1 exists n2, $n_0 = \max(n_1, n_2)$,
    $f_1(n)\, f_2(n) < c_1\, c_2\, g_1(n)\, g_2(n)$ for all $n \ge n_0$
- But for any (strictly) positive value of c1, then "forall c2>0" captures the same as "forall c1*c2".
- Hence: $f_1(n)\, f_2(n)$ is o($g_1(n)\, g_2(n)$)
- E.g. n is O(n), and 1 is o(log n), hence n is o(n log n)
- E.g. n is O(n), and log(n) is o(n), hence n log n is o(n^2)

COMP2054-ADE: little Oh rules

So we write out the inputs, and then use the definitions to convert them to statements.
These can then be multiplied.

Note that we do the usual trick to merge two values of n0 by taking the larger of the two.
The quantifiers combine because "forall x > 0" is the same as "forall y=2x > 0".
As long as we only multiply by something non-zero.
So the proof follows what we might expect, and we just get that we can combine as we might expect.

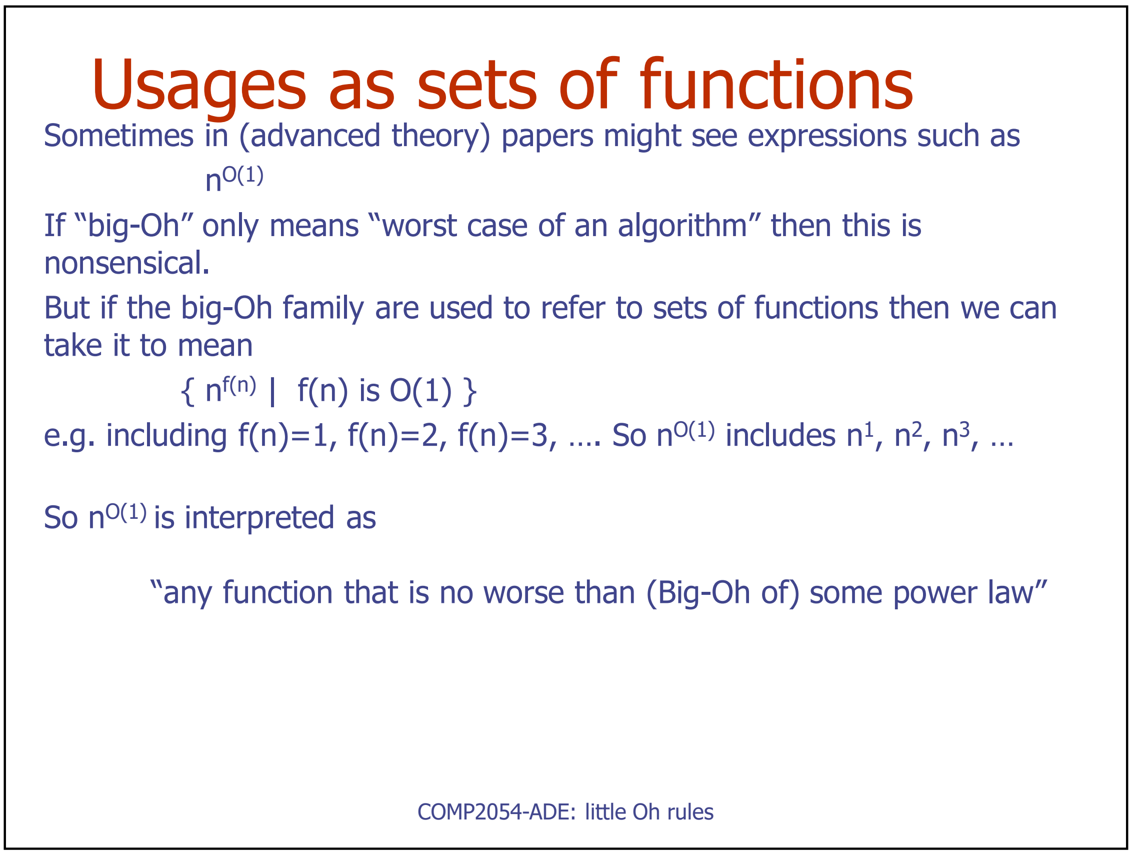
The examples are ones that should be remembered, along with the mnemonic that little-oh is "definitely better than"



Advanced Usages of Big-Oh

- **Some of the following slides on just a reminder of work covered before**
- They are included for
  - The ways of thinking about big-oh
  - Usages that might be seen in advanced academic literature, and in mathematical literature

So a few last comments.



Usages as sets of functions

Sometimes in (advanced theory) papers might see expressions such as

$n^{O(1)}$

If "big-Oh" only means "worst case of an algorithm" then this is nonsensical.

But if the big-Oh family are used to refer to sets of functions then we can take it to mean

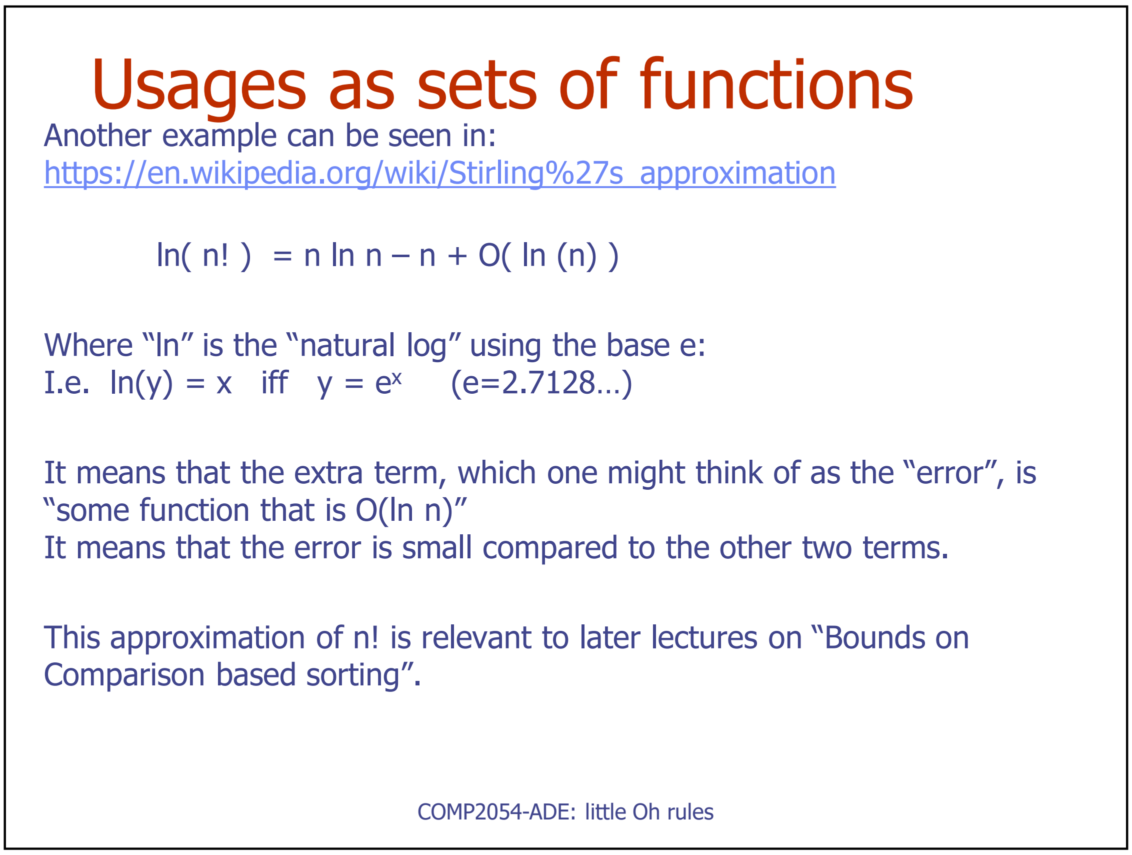
$\{ n^{f(n)} \mid f(n) \text{ is } O(1) \}$

e.g. including f(n)=1, f(n)=2, f(n)=3, …. So $n^{O(1)}$ includes n^1, n^2, n^3, …

So $n^{O(1)}$ is interpreted as

"any function that is no worse than (Big-Oh of) some power law"

We already say that O(.) can be used to represent "any function from a set of functions", and is a neat way to describe "any power law".



Usages as sets of functions

Another example can be seen in:
https://en.wikipedia.org/wiki/Stirling%27s_approximation

ln(n!) = n ln n – n + O(ln (n))

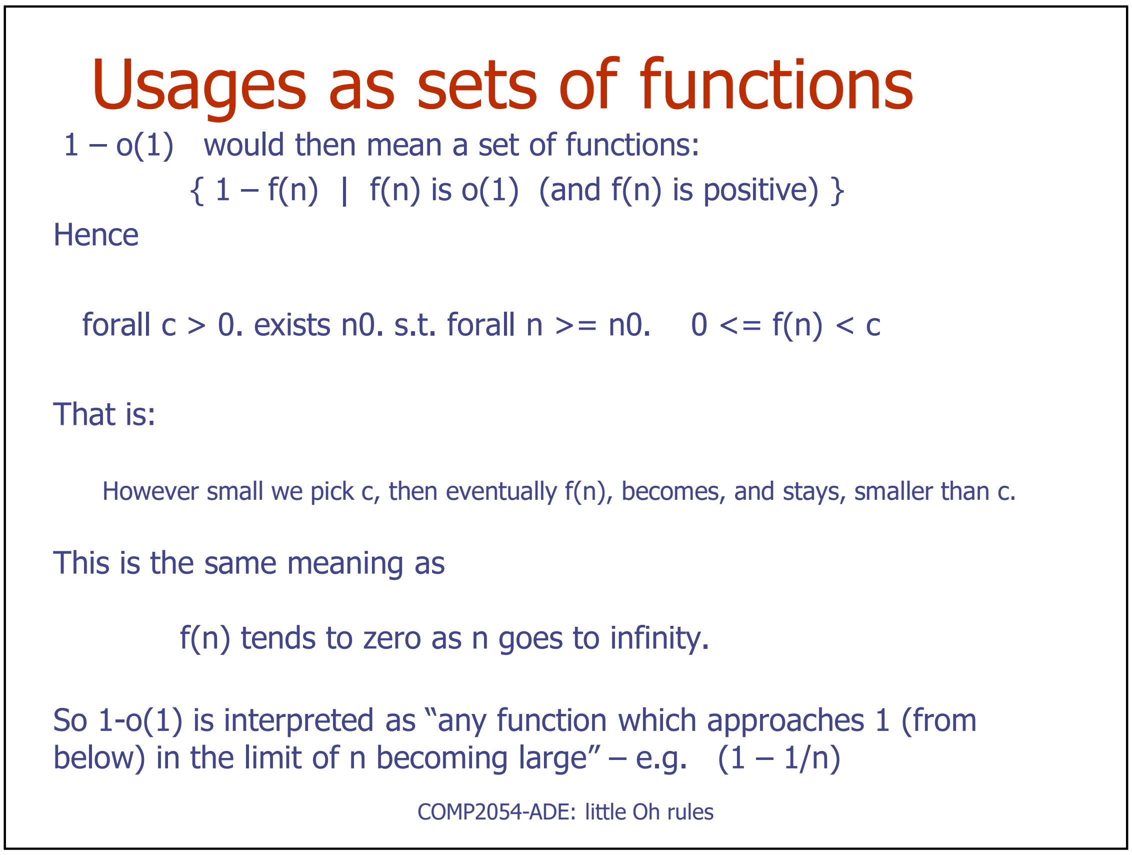
Where "ln" is the "natural log" using the base e:
I.e. ln(y) = x iff y = e^x (e=2.7128…)

It means that the extra term, which one might think of as the "error", is "some function that is O(ln n)"
It means that the error is small compared to the other two terms.

This approximation of n! is relevant to later lectures on "Bounds on Comparison based sorting".

COMP2054-ADE: little Oh rules

It is also used in mathematics.
For example, there is something called "Stirling's approximation" and that gives a way to get values close to n factorial, and is useful for theory, and for when n is very large.
It just says that the natural log is given by the first two terms, and the bit left over, is a function which is Big-Oh of the log.
Hence, the error grows strictly less quickly than the n ln n.
The full approximation is actually remarkably close even for regular size values of n – see the graph on the wiki page.
The "n log n" that you see will be used later. It is the same "n log n" as you have probably heard about for complexity of mergesort and quicksort.



Usages as sets of functions

1 – o(1) would then mean a set of functions:

{ 1 – f(n) | f(n) is o(1) (and f(n) is positive) }

Hence

forall c > 0. exists n0. s.t. forall n >= n0. 0 <= f(n) < c

That is:

However small we pick c, then eventually f(n), becomes, and stays, smaller than c.

This is the same meaning as

f(n) tends to zero as n goes to infinity.

So 1-o(1) is interpreted as "any function which approaches 1 (from below) in the limit of n becoming large" – e.g. (1 – 1/n)

Finally, another potential usage of the little-oh is to represent the limit of a function.
The interpretation of o(1) is a function that becomes ever smaller, and so has 0 as a limit.
If we say "one minus little-oh of n" then it is interpreted as just approaching 1 from below.

Hence, we see that the Big-oh-family has lots of general uses, and so is worth understanding well.