COMP2054-ADE:

ADE Lec04b:
The Big-Oh family

Lecturer: Andrew Parkes

andrew.parkes 'at' Nottingham.ac.uk

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

Big-Oh is not just a one-off, but is a member of a family of such relations, and that are also used to describe efficiency, or inefficiency of algorithms.

Recall: Relatives of Big-Oh

A close family:

- **big-Oh** **'O'**
- **big-Omega** **'Ω'**
- **big-Theta** **'Θ'**

- **little-oh** **'o'**
  - note this is not in the main text-book but is required for the module
- **little-omega** **'ω'**
  - Not required. We will not cover this; but include here so you know it exists.

COMP2054-ADE: Big-Oh family

A quick recap of a previous slide.
We did Omega and Theta in the previous lecture.
Today is little-oh

Recall: Big-Omega: Definition

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

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

if there are (strictly) positive constants c and n_0 such that

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

COMP2054-ADE: Big-Oh family

Firstly a quick recap of a “Big” definition.
Hopefully will become totally ingrained.

Recall: Big-Theta: Definition

Definition: Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $\Theta(g(n))$

if there are positive constants c' , c'' and n_0 such that

$$f(n) \leq c' g(n)$$

$$f(n) \geq c'' g(n)$$

for all $n \geq n_0$

- *Means both big-Oh and Big-Omega together.*

COMP2054-ADE: Big-Oh family

And

Little-Oh: Definition

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

$f(n)$ is $o(g(n))$

if for all positive (real) constants $c > 0$

there exists n_0 such that

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

- Spot the difference from big-Oh?
  - "**for all c > 0**" rather than "**there exists c > 0**"
  - (The change of "≤" to "<" is much less important – see later.)
- Says (roughly) that the ratio *f(n)/g(n) → 0* as *n → infinity*

COMP2054-ADE: Big-Oh family

So for the main topic today, and launch straight into the definition.
Again play ""spot the difference".

Usually people first spot the "less than" rather than the usual "less than or equal". But often (almost always) the quantifiers are the most important thing to look at first, and they drive the whole structure of a definition.
Then should focus on the quantifier of the c. It is universal not existential.
This radically changes the meaning and the proofs.

It is not unusual for people in an exam to correctly repeat the definition verbatim, and then unconsciously switch back to "exists" on trying to do a proof.

Little-Oh: Definition

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

$f(n)$ is $o(g(n))$

if **for all** positive (real) constants $c > 0$

there exists n_0 such that

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

- Note that n_0 is allowed to depend on c
  - (As it is inside the scope of the "forall c")
  - It is not: "exists n0, forall c>0"
  - As ever, the order of quantifiers is vital
- This is relevant, and vital, because we must consider all positive values of c.
  - E.g. the inequality must be satisfied for each of c=1, 0.1, 0.01, 0.001, ...
  - c does not need to be "nat"

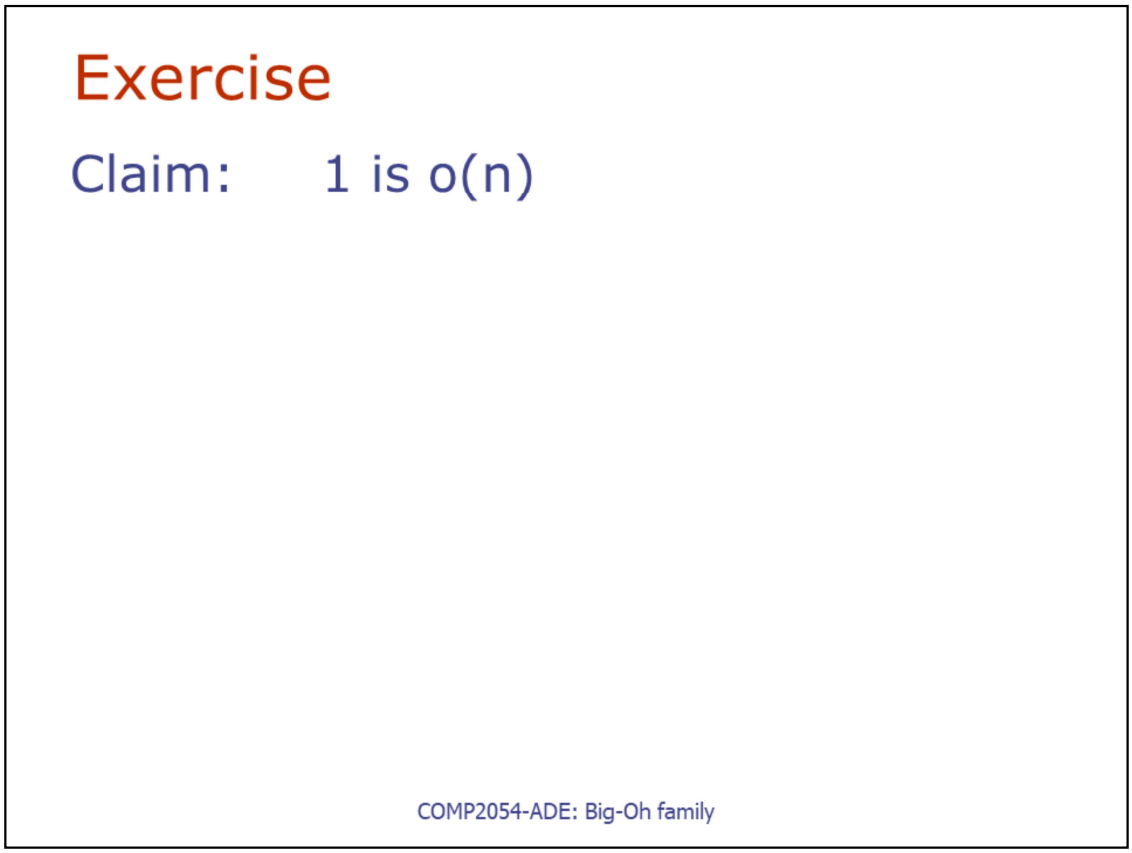
COMP2054-ADE: Big-Oh family

Again, note from the structure it is "for all c, exist n0" and this allows n0 to depend on c.

If it were "exists n0, forall c" we would have to pick some n0 first. This would be VERY different in general.

The catch is that in natural everyday language, the ordering of quantifiers can be lost – or implied by "general knowledge" and "common sense".

But mathematics, and programming, needs to be precise and unambiguous.



So as usual we start with a very simple example.
How do we proceed?
As ever plug the given information, f(n)=1 and g(n)=n into the definition, and then manipulate the algebra:

Exercise

Claim: 1 is o(n)

Claim: **for all** c > 0 there exists some n_0 such that

$1 < c\,n$ for all $n \geq n_0$

E.g. for c = 0.1 we would get

$1 < 0.1\,n$ i.e. $n > 10$,

then, for example, we can take $n_0=20$ (or any n0 > 10)

Hence, suggests we in general we can pick $n_0 = 2/c$

Proof of claim just uses $1 < c\,n$ for all $n \geq 2/c$

Note that c>0 is essential.

COMP2054-ADE: Big-Oh family

So plug it into the definition.
Make sure not to try to skip this step.
Often people try to take short cuts to write a shorter proof- maybe (wrongly) thinking it will look better – but then accidentally change the meaning, or change the ordering of quantifiers and then the rest of the "proof" is just wrong.

Metaphor: "Don't take short cuts when you are in a strange city – wait until you know it really well"

Exercise

Claim: n is $o(n^2)$

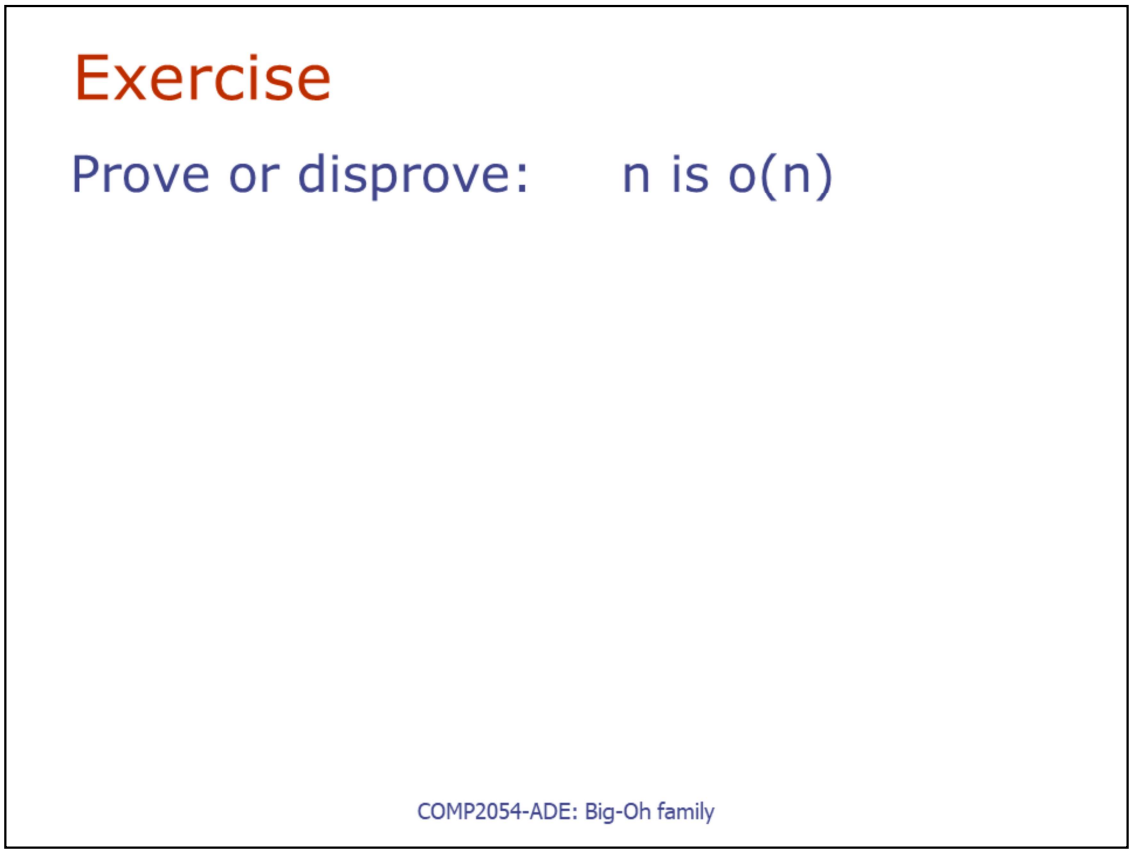
Is it true that **for all** $c > 0$ there exists some n_0 such that

$$n < c\, n^2 \qquad \text{for all } n \geq n_0$$

Exercise: complete the proof of the claim

COMP2054-ADE: Big-Oh family

Next one.
This is the same as before, simply because we can cancel the n.
NOTE: This is why is can be better to not pick n0=0, because then the case n=0 would be included, and we cannot cancel the n.
It makes it a lot simpler if wee just always assume that n>=1.
This is fine, because the point of big-oh-family is to describes the scaling as n becomes large; and the very small n is irrelevant.



One of the cases in big-Oh was that “n is O(n)”. Is it the same for little-oh?

Exercise

Prove or disprove: n is o(n)

Is it true that **for all** c > 0 there exists some n_0 such that

$n < c\,n$ for all $n \geq n_0(c)$

I.e. $1 < c$ for all $n \geq n_0(c)$

This is not true for all c, e.g. it fails for c=0.5

Hence, n is NOT o(n)

Generally: little-oh is NOT reflexive
it is more like "<" than "<="

COMP2054-ADE: Big-Oh family

We start exactly as usual. Plug the known information into the definition.
We can cancel the n.
Then directly we find we need c > 1.
If this were Big-Oh, with "exists c" we would pick c=2 and be fine.
But we need forall c> 1. c > 1 which is clearly false.

When possible it is also good practice to give a specific counter-example, and so we could arbitrarily pick c=0.5.

Little-Oh: Definition

Definition: Given functions $f(n)$ and $g(n)$, we say that
$f(n)$ is $o(g(n))$
if for all positive constants $c > 0$
there exists n_0 such that
$f(n) < c\,g(n)$ for all $n \geq n_0$

- If *f(n) is o(g(n))* then 'obviously' *f(n) is O(g(n))*
- Vital Exercise: study the definitions until this really is 'obvious' !!

COMP2054-ADE: Big-Oh family

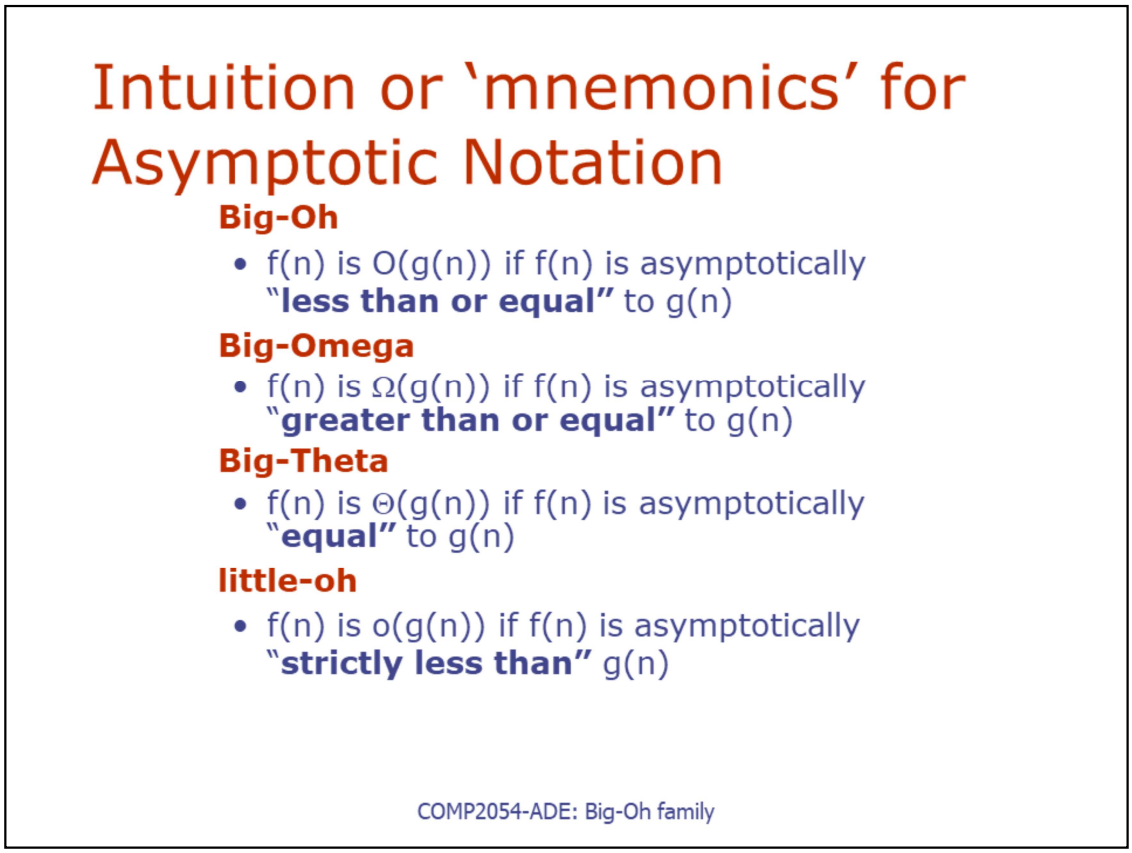
This is straightforward because it is easier to satisfy "exists c" than "forall c".
And no harder to satisfy "<=" than "<".

Little-Oh: Usage of rules

- Can also do multiplication:
  f1 is o(g1), f2 is o(g2) implies
  f1*f2 is o(g1*g2)
- Also, can "drop smaller terms"
- Standard results are
  - "powers of logs" are o("powers")
    (assuming "positive powers)
    e.g. $(\log n)^k$ is o(n), and even o(sqrt(n)), etc.
- "powers" are o("exponentials")
  e.g. n^k is o(2^n)
  e.g. n^{100} is o(1.0001^n)

COMP2054-ADE: Big-Oh family

With Big-Oh we had rules, to help avoid going back to the definition every single time.



So the mnemonic is.
I used to confuse Omega and Theta – but found a helpful mnemonic is that the line/bar across the middle of the symbol looks like part of an equality – which matches its "meaning"

Intuition or 'mnemonics' for Asymptotic Notation

There is also

Little-omega

- f(n) is ω(g(n)) if f(n) is asymptotically "**strictly greater"** g(n)

And is

- Defined in a similar fashion to little-oh
- The "strict" version of Big-Omega
- but it is rarely used in CS
- (not required in this module)

COMP2054-ADE: Big-Oh family

Just so you know of it, if you see something like ω(n).
Not needed in the module, as it is rarely used.
(Can be used in theoretical CS).

Exercises (offline)

- Make up simple questions and answer them
  - Using simple functions
  - E.g. look at the 'big oh' examples of slides, and work out some theta, omega, little-oh results
- Repeat until 'happy' ☺

COMP2054-ADE: Big-Oh family

Part of this is to just use them enough to become familiar.
Doing lots of simple examples is fine.

Exercise (offline)

Multiple choice (pick one answer)

- If f(n) is o(g(n)) then
  1. it can never also be Ω(g(n))
  2. it can sometimes be Ω(g(n))
  3. it is always Ω(g(n))

Important: Even if you cannot do the proofs then do enough examples to be able to have an intelligent guess on such multiple choice questions!

COMP2054-ADE: Big-Oh family

Here the easy/fast way to answer is to use the mnemonic metaphor that o is < and Omega is >=.
If f < g, then clearly it cannot be that f >= g.
Hence the answer is option 1.
However, it is strongly advised to put the definitions side by side and see that they cannot both be true.
In particular for it ever be true for the Omega, there will need to be some value of c, and that same value will not be allowed in the little oh, and so the forall c, will fail.

Example (offline):

The function $90\ n^2 \log(n) + n^3$ is

A. $O(n^3)$ and $o(n^3)$
B. $O(n^2)$ and $\Theta(n \log n)$
C. $O(n \log n)$
D. $\Omega(n^3)$

COMP2054-ADE: Big-Oh family

The leading term is the cubic n^3.

It cannot be $o(n^3)$ and this eliminates A.
Same in the B and C are excluded.

However it does satisfy D.

Asymptotic Algorithm Analysis

- The asymptotic analysis of an algorithm determines the running time in big-Oh notation
- To perform the asymptotic analysis
  - We find the worst-case number of primitive operations executed as a function of the input size
  - We express this function with big-Oh notation
- Since constant factors and lower-order terms are eventually dropped anyhow, we can disregard them when counting primitive operations

COMP2054-ADE: Big-Oh family

This just summarises progress so far.

Caveats & Cautions

- The point of big-Oh-family is that is can hide constants and lower-order terms; but sometimes these are important
- E.g. 100000000 n might be impractical despite being O(n)
- E.g. O(1.02^n) might be practical despite being "exponential"
- Also the worst case might happen too rarely to matter
  - But would you want to ignore it in your flight control software?

COMP2054-ADE: Big-Oh family

Again the slide says it all.

Exercises (offline)

- From the seven functions of a previous slide determine which are O, Ω, Θ and o of n^2
- Work through all the exercises in these slides (and the tutorials!)

COMP2054-ADE: Big-Oh family

This is left as an exercise – but should definitely be done.

Example of Usage

- For a complicated algorithm one could have an analysis such as:
- *"The worst case for algorithm X is known to be $o(n^4)$ and also $\Omega(n^3)$ but the exact behaviour is not known.*
- *The best case is known to be $\Theta(n^2)$.*
- *The average case (over uniformly random inputs) is $O(n^3)$."*
- Note: e.g. the true worst case could be $\Theta(n^{3.5})$ but it could be too hard to compute.

COMP2054-ADE: Big-Oh family

The point here is to note that the language is much richer than just "Oh for worst case, and Omega for best case".
The point it to be able to mix the concepts so as to allow a rich description when needed.

Summary So Far

- Role of Algorithms and Data Structures within computer science
- Random Access Machine (RAM) model
- Developing methods to reason about programs
  - Counting of steps of algorithms in RAM
  - Use of "big-Oh" to omit irrelevant and machine-specific details
  - Maths of big-Oh, Omega, Theta, little-Oh
  - Ability to analyse big-Oh of (simple) programs

COMP2054-ADE: Big-Oh family

The slide says it all.

"Appendix"

- The following is included to
  - Give a deeper insight into the various definitions
  - Give examples of formal/mathematical reasoning and "ways of thinking"

Definitions and ≤ vs. <

- It is tempting to think that
  - $f(n) \leq c\ g(n)$
  - and
  - $f(n) < c\ g(n)$
  - would give very different definitions,
  - and so, for example, would explain the big difference between O an o.
- But how much difference does it really make?
  - Strategy: try different definitions and see when they differ:

COMP2054-ADE: Big-Oh family

If unsure effects of changing a definition, then the idea is just to take both options seriously, and do reasoning to find the actual differences.

Other definitions of big-Oh?

Suppose that we had an alternative definition "$O_<$" that used

$$f \text{ is } O_<(g) \text{ iff } \exists\, c > 0, n_0. \forall n \geq n_0.\ f(n) < c\, g(n)$$

instead of usual O, which here we write as "$O_\leq$"

$$f \text{ is } O_\leq(g) \text{ iff } \exists\, c > 0, n_0. \forall n \geq n_0.\ f(n) \leq c\, g(n)$$

Are there pairs f,g for which this makes a difference?

COMP2054-ADE: Big-Oh family

So create two potential definitions.

Other definitions of big-Oh?

It is trivial that

$f \text{ is } O_{<}(g)$ implies $f \text{ is } O_{\leq}(g)$

For the converse: assume that we are given $f \text{ is } O_{\leq}(g)$ then we try to show f is also the O<(g). That is:

We know

$\exists c > 0. \ldots f(n) \leq c\, g(n)$

but can we deduce

$\exists c' > 0. \ldots f(n) < c'\, g(n)$

Hint: When does $0 \leq x \leq y$ not imply $x < 2y$?

COMP2054-ADE: Big-Oh family

If x=y=0 then although x <= y, we do not get x < 2y.
The zero case can be special.

Other definitions of big-Oh?

So the two potential definitions only differ
When does $0 \leq x \leq y$ not imply $x < 2y$?

If y > 0, then 2y – y = y > 0, so 2y > y.

So if g(n) > 0, and c > 0, then c g(n) > 0,
and so 2 c g(n) > c g(n).

So if $c > 0$ and $g(n) > 0$, and we are given
$f(n) \leq c\, g(n)$
then $f(n) \leq c\, g(n) < (2c)\, g(n)$

Which means, when g>0,
we can prove f(n) is $O_<(g(n))$ using c′ = 2c.

COMP2054-ADE: Big-Oh family

We can link this to the definitions.

Other definitions of big-Oh?

When g>0, we can prove
f(n) is $O_<(g(n))$ iff f(n) is $O_\leq(g(n))$

So unless g(n)=0 (i.e. the vast majority of time) it makes no difference which of "<" of "<=" is used in the definition!

What about when g(n)=0?
Do we prefer, or not, to be able to say that
0 is O(0) "zero is big Oh of zero" ?

COMP2054-ADE: Big-Oh family

Hence when the g(n) > 0, the definitions are equivalent!!
This rules out the "<" vs "<=" being of vital importance.
Instead the issue becomes a "corner case" which we do not directly care about a lot.

Other definitions of big-Oh?

Do we want, or not, to be able to say that
0 is O(0) "zero is big Oh of zero" ?

With big-Oh, we are used to it being reflexive, or to be able to say "f is big-Oh of f".

We don't want to constantly want to add the caveat "unless f is zero".

Hence, we want a definition in which "zero is big Oh of zero"

Hence, we use "$f(n) \leq c\ g(n)$" in the definition.

COMP2054-ADE: Big-Oh family

Instead the goal becomes to avoid having to deal with special cases.
We want to say big-Oh is reflexive, and not have to remove the special case.
So this motivates the "<=" in the definition.
It also make the proof of "f is O(f)" much easier.
This strategy is common in mathematics – definitions are engineered for convenience.

Structure of the argument

- We took two alternative definitions, found that a lot of the time they agreed
  - But then one of the definitions was better on the "corner cases" – the properties were more consistent
  - Hence, we select the definition that is more general.
- This sort of "argument structure" is common, but "behind the scenes"
  - Imagine being on a "standard committee for CS, and had to invent definitions for common usage" – there is no authority to ask the answer!

COMP2054-ADE: Big-Oh family

The definitions for the Big-Oh family are fixed of course, but still the strategy and ideas can be relevant, in case you ever have to come up with definitions for some general usage.

Other definitions of little-oh?

Suppose that we had an alternative definition "$o_{\leq}$" that used

$$f \text{ is } o_{\leq}(g) \text{ iff } \forall\, c > 0, n_0. \forall n \geq n_0.\ f(n) \leq c\, g(n)$$

instead of usual o, which here we write as "$o_{<}$"

$$f \text{ is } o_{<}(g) \text{ iff } \forall\, c > 0, n_0. \forall n \geq n_0.\ f(n) < c\, g(n)$$

Are there pairs f,g for which this makes a difference?

COMP2054-ADE: Big-Oh family

We can play a similar game with little-oh.

Other definitions of little-oh?

Similarly to the big-Oh we can show that if g(n) > 0, then

$f\ is\ o_{\leq}(g)$ iff $f\ is\ o_{<}(g)$

and so we then have a choice of what to do on the zero function $zero(n)$

Do we prefer, or not, to be able to say

$zero(n)$ is $o(zero(n))$

In this case, generally "f is not o of f".
So we do not want this.
So we can use "<" in the little-oh definition to prevent it.
And maybe "<" is more natural ?

COMP2054-ADE: Big-Oh family

And again the difference only has any effect on the special case of zero functions.
But now we do not want reflexive.
It is also more natural with the "general meaning" to use "<".

Rationale of definitions

- We use
  - "<=" in Big-Oh
  - "<" in little-oh
- definitions to give wider consistency.
- But this difference is not the most important part of the difference.

The vital difference between O and o is the difference between

$$\exists\, c > 0 \quad \text{and} \quad \forall\, c > 0$$

COMP2054-ADE: Big-Oh family

But the main lesson is that the "<" vs. "<=" is rather secondary.

The big difference is in the quantifiers.
LESSON: Look at the quantifier structures of definitions very carefully !!