COMP2054-ADE:

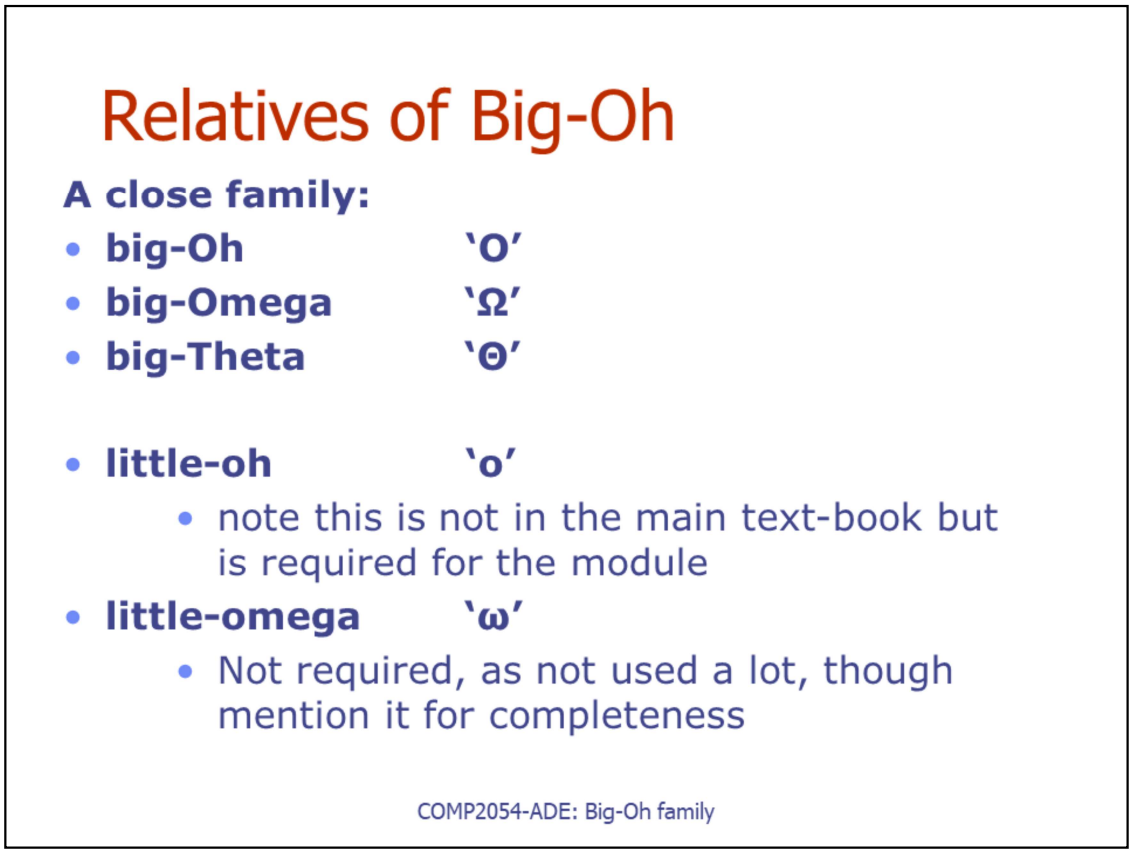
ADE Lec04a:
The Big-Oh family

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.



As well as Big-Oh we also have Big-Omega and Big-Theta, which are quite close to Big-Oh. Think of them as siblings.

A little further out in the family – think of them as cousins – are little-oh and little-omega.

Note that the notation of "Big-" and "little-" is not official, but I include it because it can be hard to tell the difference from just the symbols.

Especially between the Big and little oh. And of course in ASCII or quick discussions, finding how to type the proper Greek symbol may take too much time.

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

So let's start with "bIg-Omega"

This time we start straight off with the definition.

Most of the definition structure is very similar to Big-Oh.
But what are the differences.

Note: changed "c" to "c > 0" to emphasise that c=0 is NOT allowed.
Strictly speaking this is not needed as the term "positive" does mean c>0, and so already excludes zero, see

http://en.wikipedia.org/wiki/List_of_types_of_numbers#Signed_numbers

"Because zero itself has no sign, neither the positive numbers nor the negative numbers include zero."

If c=0 were to be allowed, then the definition would just require that f(n) >= 0 forall n >= n0, and so would be entirely useless in the context of scaling of algorithms where the functions will always be non-negative anyway.

What else is different?

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 $n \geq n_0$

- Spot the difference from big-Oh?
- "greater than" rather than "less than"
- Note that need c > 0, and we not allowed c=0
- Note that c must be **constant (cannot depend on n)**

COMP2054-ADE: Big-Oh family

The other difference is that the "<=" in "Big-Oh" changed to ">=" in Big-Omega.

As before, we need that c is a constant.

This follows from the logical structure:
exists c … forall n … c …

Means that at the time of picking c, then there is no n (it is like a local variable of a subroutine).

So, for example, writing c =1/n would not be allowed as n would "not be in scope".

Exercise (online):

- Show n is Ω(n)

Need c > 0, n0 such that

n >= c n forall n >= n0

Try c = 1

n >= n forall n >= n0

Pick n0=1.

DONE.

COMP2054-ADE: Big-Oh family

So we do some examples.
Here we just follow the standard pattern, exactly as for Big-Oh.
Take the given information and substitute into the definition to end up with something we want to make true if possible.
As for Big-Oh there is no unique choice of c and n0, so just pick ones that work, and are reasonably simple.
In this case c=1 is a natural choice.
Then we can also pick n0=1 as a natural choice.
One might say that n0=0 would also be permitted, but personally I find that n0=1 is a more natural choice, as is positive.
As usual larger values of n0 are also allowed:

If something is true forall n0 >= 1, then it is also true forall n>= 2.

Exercise (online):

- Show n^2 is $\Omega(n)$

Need c > 0, n0 such that

$n^2 >= c\,n$ forall n >= n0

Try c = 1

n >= 1 forall n >= n0

Pick n0=1

Also works if try c=2

n >= 2 forall n >= n0

n0 = 2

DONE.

COMP2054-ADE: Big-Oh family

Same pattern of working.
Simplification is to “cancel the n”
The case c=2 is given just to show that other choices for (c,n0) are allowed.
It is just more reasonable to pick ones that are easy and “natural”.
In this example, there would be no need to pick c=2, and so it would look “odd”, or “poor style”.

Exercise (online):

- Show $n^3 - n$ is $\Omega(n^3)$

Need c > 0, n0 such that

$n^3 - n >= c\, n^3$ forall $n >= n0$

Try c = 1

$n^3 - n >= n^3$ forall $n >= n0$

$- n >= 0$ forall $n >= n0$

FAILS

Try c = 1 / 2

$2 n^3 - 2 n >= n^3$ forall $n >= n0$

$n^2 >= 2$ forall $n >= n0$

n0 = 2 DONE.

COMP2054-ADE: Big-Oh family

Again plug into the definitions.

Then can just try some values of c=1.

Let's try c=1 – it worked before, so reasonable to try again.

But then simplification is "cancel the n^3" and we get –n >= 0, or n <= 0, and so this fails.

THIS DOES NOT MEAN THERE IS NO PROOF: IT ONLY MEANS ONE CHOICE FAILED.
Other choices might succeed.
In Big-Oh, when this happened it corresponded "try a larger value of c".
But in Big-Omega, increasing c will not help, because it makes it harder to satisfy ">= c g(n)".

Instead need to decrease c.

Note the definition said that need c >0, but did not say that c had to be an integer.
So lets try c = 1 / 2.
Then simplification can work by
- multiply by 2, to get rid of the half
- subtract n^3 from each side
- divide by n. giving n^2 – 2 >= 0

- Move -2 to other side.
Then n0=2 works as n^2 > 2 forall n >= 2.
Hence we are done and the results is proved.

Exercise (online):

- Is it true that: 1 is Ω(n) ?

Need c > 0, n0 such that

1 >= c n forall n >= n0

Note: No need for c to be integer

Try c = … FAILS

Instead use c > 0 to get

(1/c) >= n forall n >= n0

FAILS as eventually n > (1/c)

FAILS.

1 is NOT Ω(n)

c = 1/n is NOT allowed – need c constant

COMP2054-ADE: Big-Oh family

Now another simple case.
Again just plug in the definition.
We need c > 0, hance can divide by c, and we find we need
n <= 1/c forall n >= n0
But there is clearly no pair (c,n0) that can satisfy this.
If write d=1/c then

"Exists d>0,n0. n <= d forall n >= n0" is not a theorem, because we can always take
n=max(d+1,n0).
We saw the same issue for "n is not O(1)"
If the definition allowed c=0 then it would only mean f(n) >= 0 forall n>=n0 and so would be a useless definition.

If the definition allowed c to depend on n, then could (usually) take c=f(n)/g(n) and again the definition would be useless.

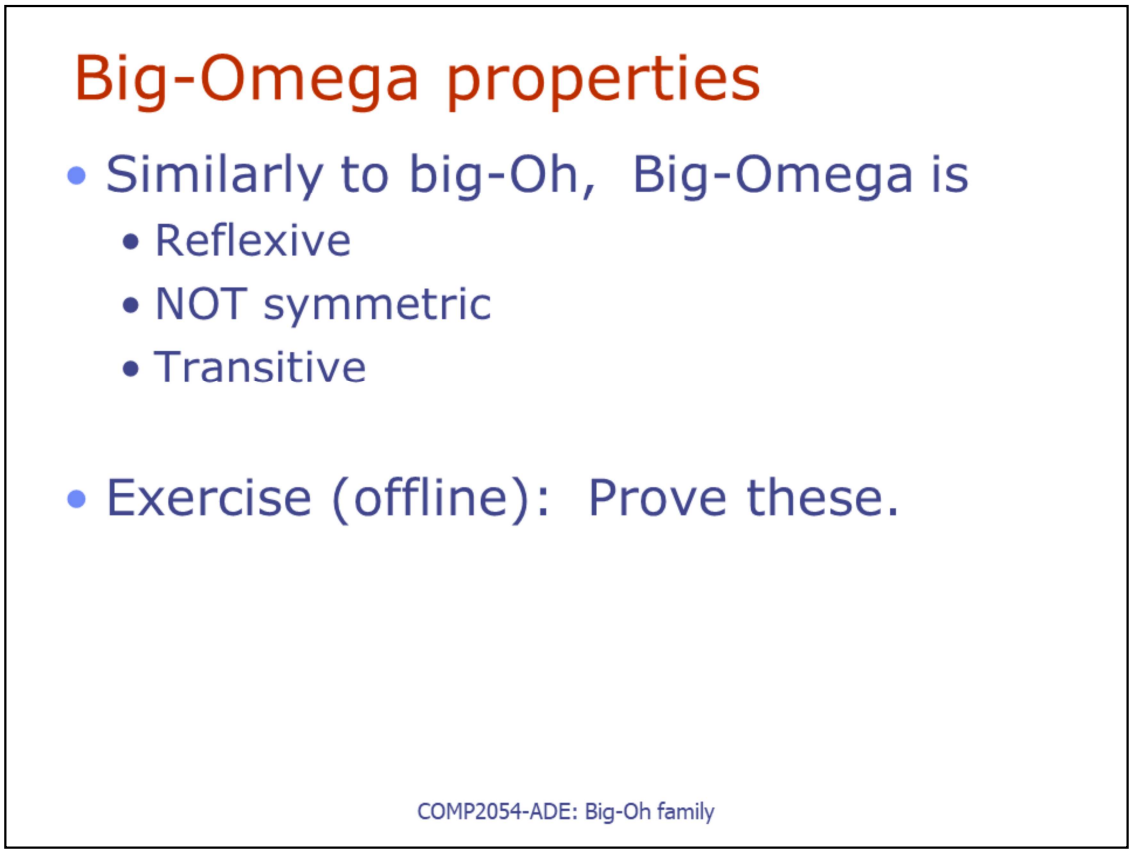
Definitions are engineered to meet the needs of a community, and it is important/useful to understand this engineering process.

Big-Omega Examples

- We have
  - n is $\Omega(1)$
  - n is $\Omega(n)$
  - n is not $\Omega(n^2)$
- This is "the opposite of Big-Oh"
- **f(n) is Ω(g(n)) says that "f(n) grows at least as fast as g(n) at large n"**
- Compared to
  - f(n) is O(g(n)) says that "f(n) grows no faster than g(n) at large n"

COMP2054-ADE: Big-Oh family

So a quick review of some immediate simple results, and it clear that Big-Omega captures the idea of "grows at least as fast as".
Compare this to Big-Oh which captured the idea of "grows no faster than".
They are like a complementary pair.



For Big-Oh we considered it as a relation and asked about it having properties associated with relations.
(Revise the Big-Oh slides as needed).
Can again show that it is

Reflexive: h(n) is automatically Omega(h(n))

Transitive: f is Omega(g), g is Omega(h) forces f is Omega(h)

Linking big-Oh and Big-Omega

- Suppose that we are give "f is O(g)"
- What can we say about big-Omega?

COMP2054-ADE: Big-Oh family

As we saw, Big-Omega is like the opposite of Big-Oh.
Can this be made more explicit?

Linking big-Oh and Big-Omega

- Suppose that we are give "f is O(g)"
  - We know there exist c n_0 such that
    $f(n) \leq c\, g(n) \quad \forall\, n \geq n_0$
  - We can assume $c > 0$ (Why?). Hence
    - $g(n) \geq \left(\frac{1}{c}\right) f(n) \quad \forall\, n \geq n_0$
  - Hence, $g\ is\ \Omega(f)$.
- That is: $f \in O(g) \rightarrow g \in \Omega(f)$
- Similarly: $f \in \Omega(g) \rightarrow g \in O(f)$.
- Note: similar to: x <= y → y >= x

COMP2054-ADE: Big-Oh family

As usual, to proceed we just plug into the definitions.

We can assume c is not zero, as c=0 would force f(n)=0 and this is not a case we care about.

To be more accurate we should exclude that f(n) == 0.
So then turn it around and we get a formula that matches what we need for Big-Omega.
This is very similar to what we have with "<=" and so reinforces that big-Oh behave in a similar way to "less than or equal".
But it is a "comparison of growth rates" not a "comparison of actual values".

Usage of big-Omega

- Once familiar with big-Oh then big-Omega is very similar but "upside down". Same rules apply:
  - Multiplication rule still applies
  - Can still drop smaller terms
- E.g. $n^3 - n$ is $\Omega(n^3)$
- 'Ω ' expresses "grows as least as fast as"
- 'O' expresses "grows at most as fast as"

COMP2054-ADE: Big-Oh family

AS for Big-Oh we want rules to avoid having to go back to the definitions every single time.
It is easy to check that the multiplication rule still works.
We can also use the rule of dropping smaller terms.

'Paradox' of big-Omega

- "Can still drop smaller terms"
- E.g. $n^3 - n$ is $\Omega(n^3)$
- Note that this might seem counter-intuitive
  - thinking "big Omega does `at least' and so need the smallest term" is incorrect!
  - instead think of the large n behaviour being dominated by the n^3, and that this grows at least as fast as n^3.

COMP2054-ADE: Big-Oh family

A very common error is to think that because Big-Oh and Big-Omega are opposites, then the "drop smaller terms" should become to "drop larger terms". But this is wrong.
They both are about the behaviour at very large values of n, and so for both of them it makes sense to drop smaller terms as these do not drive the growth rate.
A proof would use the idea that "n^3 + n" becomes close to n^3 at large enough n.
Then only a small change to c would be needed, as long as n0 is large enough.
E.g. see the proof we did before.
We could have taken c=0.99 and then we would just need
$100\ n^3 - 100\ n >= n^3$ forall $n >= n0$
$99\ n^2 - 100 >= 0$ forall $n >= n0$
$99\ n^2 >= 100$ forall $n >= n0$
and n0=2 is okay.

Omega Usage

- A standard usage might be to capture a limitation on the best one can hope for,
  e.g. "*The best case for algorithm X is $\Omega(n^3)$ and so it will not scale well*"
- However, if the worst case behaviour of an algorithm is a 'bizarre' or 'partially unknown' function of n, then one might say,
  e.g. "*The worst case for algorithm X is not precisely known but we know it is $\Omega(n^3)$ and $O(n^4)$*"
- Gives a lot more flexibility than doing ratios.

COMP2054-ADE: Big-Oh family

Big-Omega is used reasonably often.
Especially as it gives a limit of the best one can do, whereas.

It also gives a way to sandwich together the upper and lower bounds, and so be able to make statements such as no worse than … but no better than …

Together these give a lot of flexibility.

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 $n \geq n_0$

- *What does this say about the growth rate of f(n) ?*
- *How does it connect to O and Omega?*

COMP2054-ADE: Big-Oh family

Now move to Big-Theta and again will start from the definition.

Again suggest to play “spot the difference”.

Big-Theta

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 $n \geq n_0$

- *f(n)* is *Θ(g(n))* if and only if
  f(n) is *O(g(n))* and also *f(n)* is *Ω(g(n))*
- Example: *2n+1* is *Θ(n)*
- *Θ expresses "grows 'exactly' as fast as"*

COMP2054-ADE: Big-Oh family

Fairly clearly it just joins together Big Oh and Big-Omega.

Note that the Oh and Omega can use different values of c.
It does NOT say that need c'=c" as then would have f(n) = c g(n) and this is too strong to be a useful definition.

It expresses that f and g have the same growth rates, but not that they are forced to be proportional.

EXERCISE

- Consider the function:

  f(n) = n if n is even

  2n if n is odd

What is its big-Omega behaviour?

Clearly is Ω(n).

Just take c=1 n0=1 then

f(n) >= 1 n forall n >= 1

COMP2054-ADE: Big-Oh family

To see the utility, consider this slightly less trivial function.

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

Firstly, what can we say about its Big-Oh behaviour?
It is trivial to show it is Omega(n), as both the even and odd cases work with c=1,n0=1.

EXERCISE

- Consider the function:

  $f(n) = n$ if n is even

  $2n$ if n is odd

What is its big-Theta behaviour?

Clearly is $\Theta(n)$. As is both $O(n)$ and $\Omega(n)$.

But note that there is not a single value for ratio $f(n)/n$.

$\Theta(n)$ expresses the growth rate is linear but does not tightly sandwich the specific values.

- This is very useful in CS !
- Sometimes people can really mean Big-Theta when they say Big-Oh

COMP2054-ADE: Big-Oh family

Easy to show it is O(n).
Hence it is Theta(n).
In particular, a function that is not a simple monotonically increasing function, and could occur in real CS.

Theta allows us to specify a growth rate, without over-specifying, and forcing the functions to be proportional.

This is very useful in CS.
Also, note that Big-Theta might be closer to what you intuitively though Big-Oh was going to be ☺.

Big-Theta is reflexive and transitive

- Trivial as both O and Ω are.

COMP2054-ADE: Big-Oh family

As before it is a relation – between two functions, and so can ask the usual questions.

Is Big-Theta symmetric?

- If $f \in \Theta(g)$ then it is (automatically and always) true that $g \in \Theta(f)$
- Exercise (offline): (Follows quickly from previous results).

COMP2054-ADE: Big-Oh family

What about symmetry?
These did not hold for either Big-Oh or Big-Omega on their own.
But when we pair together we find that it does hold.

Θ is an equivalence relation

- Any relation that is
  - Reflexive & Symmetric & Transitive
- is an "equivalence relation"
  - Roughly speaking: it behaves like a "equality":
  - $\Theta(g(n))$ is "the equivalence class of all functions whose large n behaviour is bounded above and below by constants times g(n)"
- It is reasonable to write "$f = \Theta(g)$"
  - (pedantically, arguably, could be "$\{f\} = \Theta(g)$" but dropping "{}" on singleton sets is a common abuse of notation. But also it is somewhat of an abuse as it is not a set inequality, rather it is an "equivalence relation", i.e. closer to "$\{f\} == \Theta(g)$")
  - "Reasonable" is not same as "recommended" ☺

COMP2054-ADE: Big-Oh family

But now it means Theta is behaving a lot more like equality.
One can write it as an equality and this is quite commonly done.

Personally I still prefer to think of it as
".. is Theta(.)" or ".. is a member in the set Theta(.)".
As the equality is an equivalence relation.
Might regard Theta(n) as the infinite set of functions
{n,n+1,n+2,…2n, 2n+1, …3n, 3n+1, …}
which is not the same as {n} as a set equality,
Even though people may write n = Theta(n).

BREAK

- Next lecture "little-oh"