COMP2054-ADE:

ADE Lec04b: The Big-Oh family

Lecturer: Andrew Parkes

andrew.parkes ‘at’ Nottingham.ac.uk

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

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.

Recall: Big-Omega: Definition

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

$$f(n) \text{ is } \Omega(\ g(n)\ )$$

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

$$f(n) \geq c\, g(n) \qquad \text{for all } n \geq n_0$$

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.*

Little-Oh: Definition

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

$$f(n) \text{ is } o(\ g(n)\ )$$

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

there exists n_0 such that

$$f(n) < c\, g(n) \quad \text{for all } n \geq n_0$$

- 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 $f(n)/g(n) \to 0$ as $n \to$ *infinity*

Little-Oh: Definition

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

$$f(n) \text{ is } o(\, g(n)\,)$$

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

there exists n_0 such that

$$f(n) < c\, g(n) \quad \text{for all } n \geq n_0$$

- 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"

Exercise

Claim: 1 is $o(n)$

Exercise

Claim: 1 is $o(n)$

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

$$1 < c\,n \qquad \text{for all } n \geq n_0$$

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.

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

Exercise

Prove or disprove: n is o(n)

Exercise

Prove or disprove: n is $o(n)$

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

$$n < c\,n \qquad \text{for all } n \geq n_0(c)$$

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 "<="

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' !!

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)

Intuition or ‘mnemonics’ for Asymptotic Notation

Big-Oh

- f(n) is O(g(n)) if f(n) is asymptotically “**less than or equal”** to g(n)

Big-Omega

- f(n) is Ω(g(n)) if f(n) is asymptotically “**greater than or equal”** to g(n)

Big-Theta

- f(n) is Θ(g(n)) if f(n) is asymptotically “**equal”** to g(n)

little-oh

- f(n) is o(g(n)) if f(n) is asymptotically “**strictly less than”** g(n)

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)

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' ☺

Exercise (offline)

Multiple choice (pick one answer)

- If $f(n)$ is $o(g(n))$ then

1. it can never also be $\Omega(g(n))$
2. it can sometimes be $\Omega(g(n))$
3. it is always $\Omega(g(n))$

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!

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. $\Omega(n^3)$

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

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?

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!)

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.

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

“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 $\leq$ 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:

Other definitions of big-Oh?

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

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

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

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

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

Other definitions of big-Oh?

It is trivial that

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

For the converse: assume that we are given $f\ is\ O_{\leq}(g)$ 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 $x < 2y$?

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.

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" ?

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.

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!

Other definitions of little-oh?

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

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

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

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

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

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 ?

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

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