COMP2054-ACE: Introduction to big-Oh

ADE-Lec02a

Lecturer: Andrew Parkes

Email: andrew.parkes@Nottingham.ac.uk

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

Recap: Removing details

- (For an example program) According to precisely how we count steps we might get one of
  - $5 \log_2(n) + 2$
  - $9 \log_2(n) + 5$, etc.
- Also this counts "steps"
  - the translation to runtime depends on the compiler, hardware, etc.
- **Need a way to suppress such 'implementation-dependent details'**

Aim: **Classification of Functions**

- CS needs a way to group together **functions** by their scaling behaviour, and the classification should
  - Remove unnecessary details
  - Be (relatively) quick and easy
  - Handle ‘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”:

  O (Big-Oh), Ω (Big-Omega), Θ (Big-Theta),
  o (little-oh) and ω (little-omega)

“Surprising”/”Advanced” uses of Big-Oh family

- Stirling’s approximation (which we use later)
  https://en.wikipedia.org/wiki/Stirling%27s_approximation

$$\ln(n!) = n \ln n - n + O(\ln n)$$

  - where “ln” is the natural logarithm “log base e”
  - (Note there is no reference to an algorithm!)
- Polynomial functions are $n^{O(1)}$
- The “best case of algorithm X is O(n log n)”
- Does there exist an algorithm for TSP with runtime $2^{o(n)}$?

Need a good/proper understanding of Big-Oh family to interpret the above

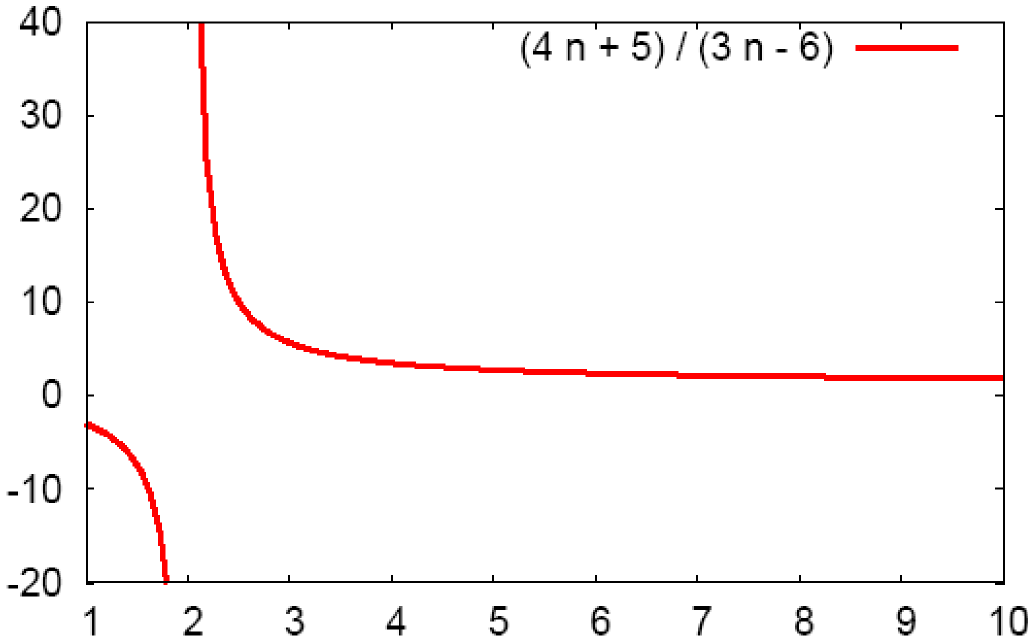
Big-Oh Notation: Motivations

- Suppose we have two functions
  1. $4n + 5$
  2. $3n - 6$
- We want to express that the behaviour is driven by fact that both are linear in n, and to suppress the details of the exact function
- One way (not the only way!) to motivate definitions is to look at their ratio:

Important (Subtle?) Comment

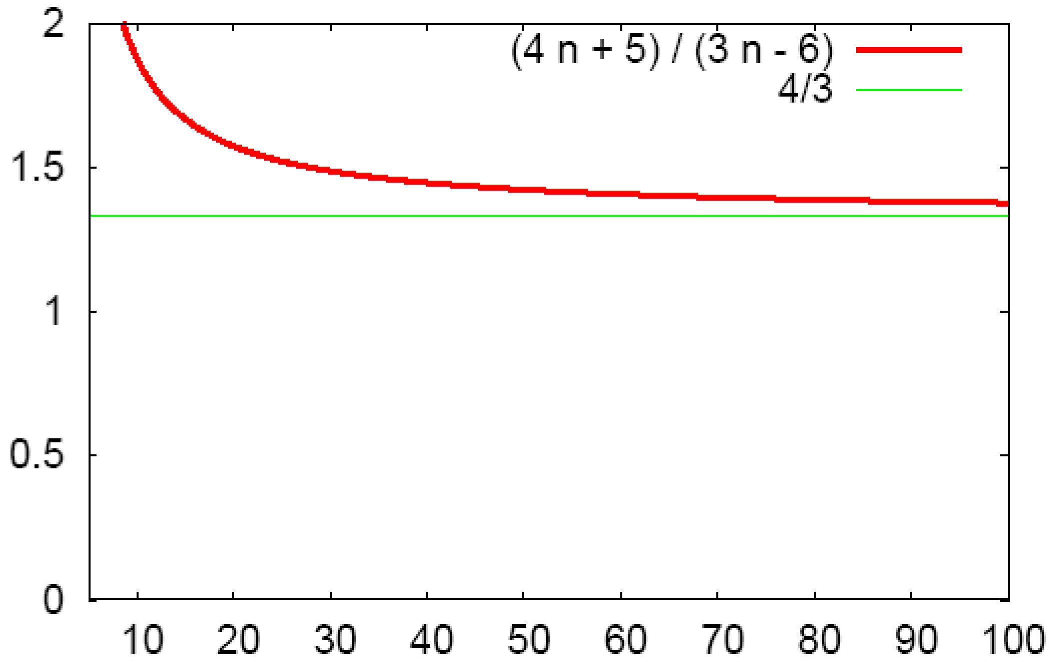
- Note that at this point we are just talking about functions, "f(n)", of a single parameter n
- Big-Oh is often applied to runtimes, but this is not essential
  - The big-Oh definitions are just in terms of functions
  - It is not only for "worst case runtime"
  - But the big-Oh definition should be "designed" to be suitable for discussion of "runtime functions"
  - How do we motivate a good design of the definitions?

Ratio of two linear functions



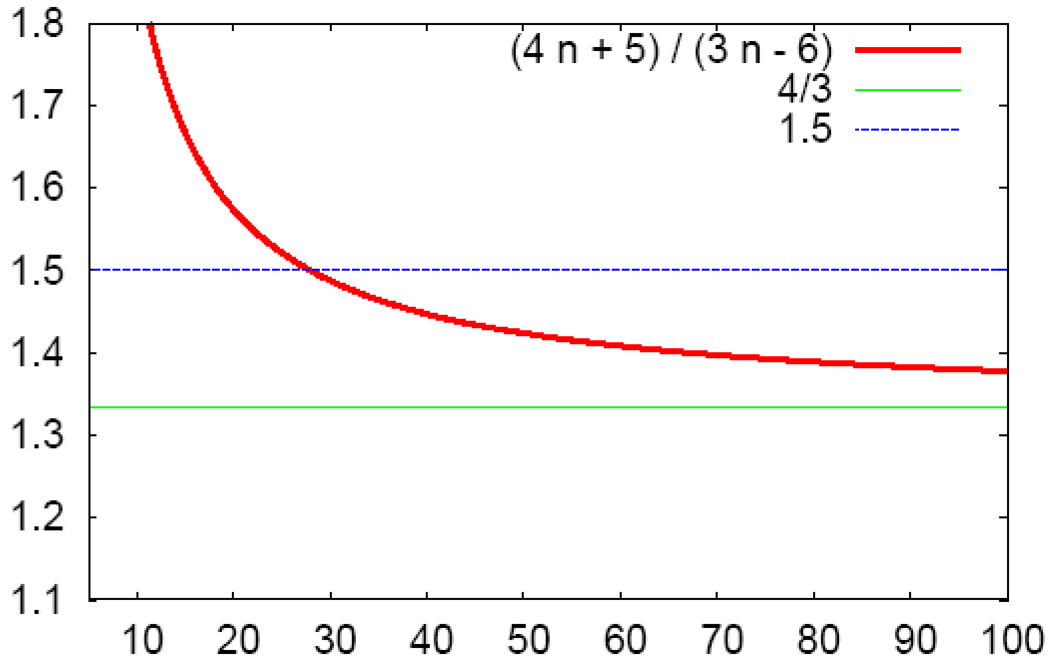
- The behaviour at small ***n*** is messy

Ratio of two linear functions



- At larger values of ***n*** the ratio starts to behave more predictably
  - Suggests definitions should use "forall n >= ..."

Ratio of two linear functions



- It looks like:
  $(4n+5) \le 1.5 * (3n-6)$ for all $n \ge 30$?

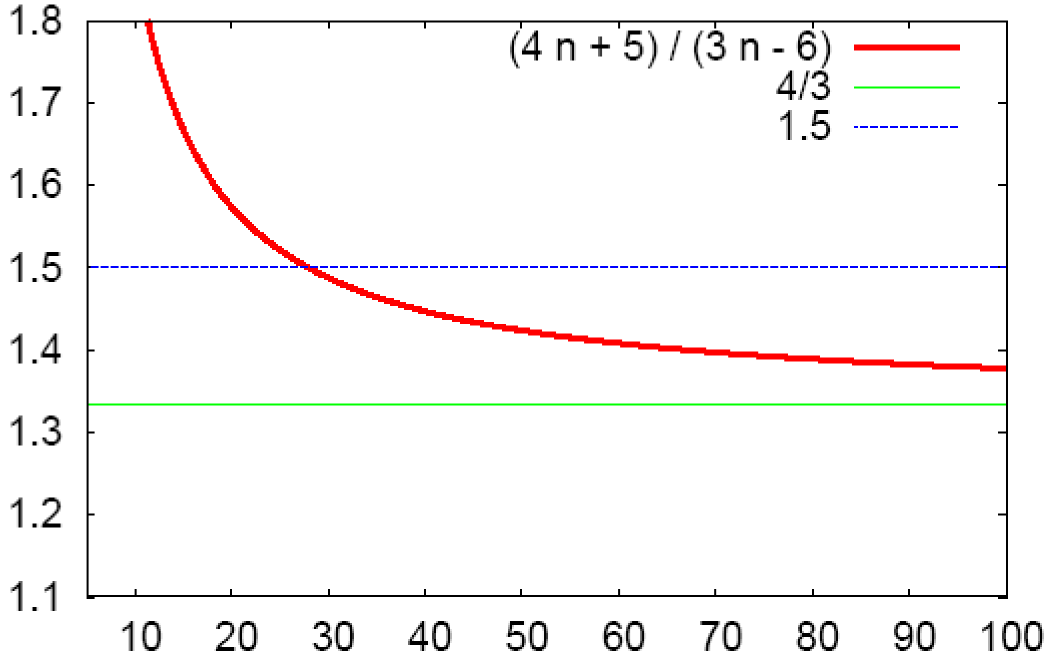
Exercise: computation

- For precisely what values of n is the following true:

  $$(4n+5) \le 1.5 * (3n-6)$$

- Answer:
  - $8n + 10 \le 9n - 18$ (by x2)
  - $28 \le n$ (add 18-8n to both sides)

Ratio of two linear functions



- Hence we have:

  $(4n+5) \leq 1.5 * (3n-6)$ for all $n \geq 28$
- Suggests definitions that place upper bounds on the function relative to another function:

Big-Oh Notation: Definition

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

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

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

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

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

Which functions are used?

- Big Oh is intended to functions of **positive integers** (size), and that are **positive real values** because they (often) represent runtimes:
  - f $: \mathbb{N}^+ \rightarrow \mathbb{R}^+$ (and similarly for g)
  - Where $\mathbb{N}^+ = \{ 1, 2{,}3, \ \ldots \}$ and
  - $\mathbb{R}^+ = \{ x \in \mathbb{R} \,|\, x \geq 0 \}$
- That is, assume, $f(n) \geq 0 \ \ \forall n \geq 1$
- Sometimes, for convenience, might relax to
  - $f(n) \geq 0 \ \ \forall n \geq N$ for some constant N

Big-Oh Notation: Definition***

Carefully note the structure and order of the quantifiers:

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** $n \geq n_0$

i.e. "exists-exists-forall" structure:

$\exists\ c>0.\ \exists\ n_0.$ such that $\forall\ n \geq n_0.\ f(n) \leq c\ g(n)$

A common mistake is to get the nesting and placement of the quantifiers in the wrong order

Big-Oh Notation: Definition

Carefully note that the quantifier structure is "pick c, before handling forall n" and so c is not allowed to be a function of n:

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** $n \geq n_0$

- We cannot pick that c depends on n.
- Otherwise we can (usually) just take c = f/g and the definition becomes useless as it would not fail, so
  - it would not allow us to say anything useful about the relative growth rates of f and g.

Big-Oh Notation: Definition

Carefully note "forall n …"

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** $n \geq n_0$

Showing something works at $n = n_0$ is pointless as says nothing about the growth rates at large n.

How do formally prove statements of "forall n"?

In worst case would need induction;
but in general we will just use "obvious properties" ("standard lemmas")

Big-Oh Notation: Definition***

Note the words that do **not** appear in

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

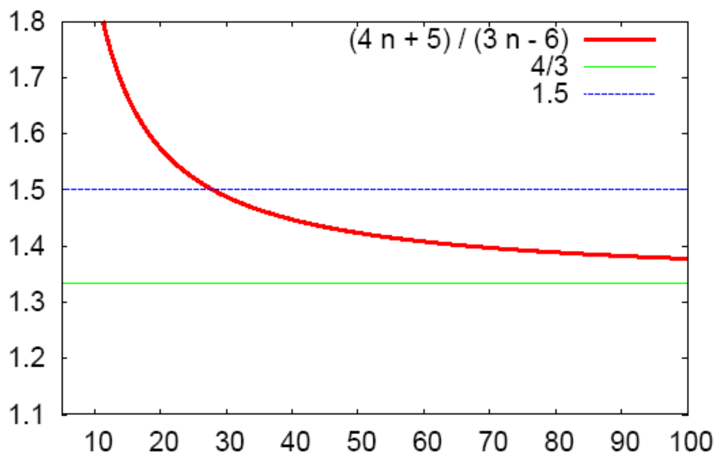
$f(n)$ is $O(\, g(n)\,)$

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

The definition does not mention "worst case of an algorithm runtime". It does not even mention "algorithm".

It only mentions "functions".

Ratio of two linear functions



- Since: $(4n+5) \le 1.5 * (3n-6)$ for all $n \ge 28$
- We now have: $(4n+5)$ is $O((3n-6))$
  - Using $c=1.5$ and $n_0=28$

EXERCISE

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

- Show that the function:

  $$f(n) = 1$$

  is O(1)

  ANS: (pause and try!)

EXERCISE

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

- Show that the function:

  f(n) = 1

  is O(1)

  ANS: pick c,n0 s.t 1 <= c 1 forall n>=n0

  c=1 n0=1 Done.

  c=3499993, n0=392875

  Proof still works. **Definition does not say pick smallest c,n0.**

EXERCISE

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

- Can we show that the function:

  $$f(n) = 2$$

  is O(1)?

ANS: (pause and try!)

EXERCISE

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

- Can we show that the function:

$$f(n) = 2$$

is O(1)?

ANS: need to

pick c,n0 s.t. 2 <= c 1 A n >= n0

c=2 n0=1

DONE

EXERCISE

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

- Show that the function:

  f(n) = k

  is O(1) for any fixed constant k

ANS: pick c,n0. k <= c 1

c = 2*max(k,1) is allowed (not using n)

n0=2

c=k/2 fails – but does not matter as need "exists c" not "forall c"

EXERCISE

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

- Can we show that the function:

  $$f(n) = 1$$

  is O(n)

  ANS: (pause and try!)

EXERCISE

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

- Can we show that the function:

  f(n) = 1

  is O(n)

  ANS: pick c,n0. s.t. 1 <= c n forall n >= n0

  n0=1

  c=1 need 1 <= n forall n >= 1

DONE: i.e. 1 is O(n),

but have 1 is O(1)

EXERCISE

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

- Can we show that the function:

$$f(n) = k$$

is O(n) for any fixed constant $k \geq 0$

ANS:

EXERCISE

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

- Show that the function:

$$f(n) = k$$

is O(n) for any fixed constant $k \geq 0$

ANS: want

$$k \leq c\, n \qquad \text{for } n \geq n_0$$

Note that c=k, n_0=1 will suffice

Hence, k is O(n), as well as O(1).

EXERCISE

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

- Show that the function:

  f(n) = k n

  is O(n) for any fixed constant k

ANS: (pause and try!)

EXERCISE

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

- Show that the function:

  f(n) = n + k

  is O(n) for any fixed constant k

ANS: pick c,n0 s.t
  n+k <= c n forall n >= n0

Pick c=1   n+k <= n   fails (for k>0)

c=2   k <= n   pick n0=k DONE

EXERCISE

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

- Show that the function:

  f(n) = n

  is **not** O(1).

  ANS: (pause and try)

EXERCISE

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

- Show that the function:

  f(n) = n

  is **not** O(1).

  ANS: can we pick? c,n0 s.t.

  n <= c forall n >= n0

Pick c = n ? NO!!!!!!

CANNOT pick c. c=infty is not allowed

Hence not O(1).

It is also O(n^2), but O(n) is "best".

EXERCISE

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

- Show that the function:

  $f(n) = n$

  is not $O(1)$.

ANS: We would need to provide **constants** c and n_0 (c cannot depend on n) and prove

$$n \le c\,.\,1 \qquad \forall\, n \ge n_0$$

But this is clearly impossible.

E.g. take $n = \max(c+1, n_0)$.

BREAK until next lecture lec02b