G52ADS 2014-15: Introduction to big-Oh. Lec. 2

Lecturer: Andrew Parkes

Email: ajp ‘at’ cs.nott.ac.uk

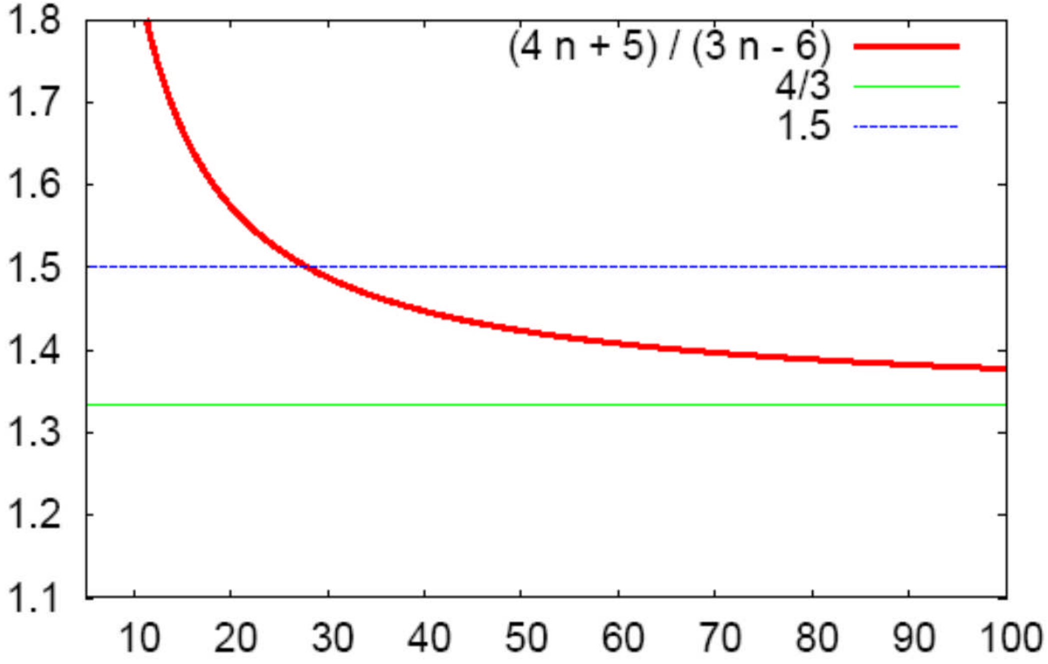
http://www.cs.nott.ac.uk/~ajp/

Recap: Counting Primitive Operations

- Worst case number of primitive operations executed as a function of the input size

Algorithm *arrayMax*(A, n)	# operations	
$currentMax \leftarrow A[0]$	2	
for $i \leftarrow 1$ **to** $n - 1$ **do**	1	
if $A[i] > currentMax$ **then**	$2(n-1)$	
$currentMax \leftarrow A[i]$	$2(n-1)$	(worst case)
{ increment counter: i++ }	$2(n-1)$	("hidden")
{ test counter: $i \leq (n-1)$ }	$2(n-1)$	("hidden")
return $currentMax$	1	
Total	$8n - 4$	

Recap:
Ratio of two linear functions



- Observe: (4n+5) ≤ 1.5 * (3n-6) for all n ≥ 30
- We say: (4n+5) is "big-Oh" of (3n-6)

Recap: Big-Oh Notation: Definition***

Definition: Given 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

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

Meta-Comment

- There are often questions about whether the definition is appropriate or makes sense
- "Never question authority" is not right; instead
  - You should 'intelligently question' 'authority'
  - Definitions are not 'fixed in stone' but done so as to be useful – hence you should try to understand why things are defined the way they are. Considering why a definition is created a particular way is usually a good way to understand it. E.g. For each part of a definition ask yourself what were the alternatives.

Objections?

Possible questions:

1. Why not just take a ratio?
2. It seems that n is $O(n^2)$ so it does not really make sense?

- This lecture will start to answer these

Big-Oh Notation: Definition

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

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

iff there exist positive constants c and n_0 such that

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

- 'iff' is a common abbreviation for 'if and only if'
- "for all $n \ge n_0$" is so that we focus on large n and can ignore messy behaviour at small values

Big-Oh Notation: Definition

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

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

iff there exist positive constants c and n_0 such that

$f(n) \le c\ g(n)$ for all $n \ge n_0$

- the <u>constant</u> c is so that we do not need to exactly capture the ratio – it is allowed to be "loose"
  - A proof does not need the "smallest constant"
- But, "c **must** be chosen before n", the order matters!
- it is "a constant independent of n" : not "a constant once n has been chosen"
  - (do not skip over such 'nuances', they are often vital)

Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \le cg(n)$ for all $n \ge n_0$

- Method:
  - write down $f(n) \le cg(n)$ and try to simplify it to extract a restriction on n as a function of c
  - Try to pick c such that n satisfies the restriction for large enough values, greater than n_0

Big-Oh Notation

- (With new concepts start with the very simplest examples, and work up towards harder examples)
- Example:
  Is it true or not that the function 3 is O(1)?
- Yes. Proof:
  - Have $f(n)=3$ and $g(n)=1$
  - Need $3 <= c$
  - Can pick $c=5$ and $n_0 = 10$
  - Many other choices, but only need one choice

Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ iff there exist positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for all $n \geq n_0$

- Example: $2n$ is $O(n)$??
  - Need: $2n \leq cn$
  - $c \geq 2$
  - E.g. Pick $c = 3$ and can pick any n_0
  - Hence, $2n$ is indeed $O(n)$

Exercises (from last lecture)

Defn: Given 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 $f(n) \le c\, g(n)$ for all $n \ge n_0$

- Show that
  - (3n-6) is O((4n+5))
  - (3n-6) is O(n)
  - (4n+5) is O(n)
- Note the last two 'suppress details' as desired

Exercises (from last lecture)

Show that (3n-6) is O((4n+5))

ANS: we need c, n_0 such that

$(3n-6) \le c\,(4n+5)$ for all $n \ge n_0$

Hence … (done in class)

Exercises (from last lecture)

Show that (3n-6) is O(n)

ANS: we need c, n_0 such that

$(3n-6) \le c\, n$ for all $n \ge n_0$

Hence … (done in class)

Exercises (from last lecture)

Show that $(4n+5)$ is $O(n)$

ANS: we need c, n_0 such that

$(4n+5) \leq c\,n$ for all $n \geq n_0$

Hence … (done in class)

EXERCISE

- Show that the function:

  $f(n) = k\ n$

  is O(n) for any fixed constant k

ANS: <in class>

EXERCISE

- Show that the function:

  $$f(n) = n + k$$

  is $O(n)$ for any fixed constant k

ANS: <in class>

EXERCISE

- Show that the function:

  $f(n) = k$

  is O(n) for any fixed constant k

ANS: <in class>

EXERCISE

- Show that the function:

  $f(n) = k$

  is $O(n)$ for any fixed constant k

ANS: want

$k \le c\, n \quad \text{for } n \ge n_0$

Note that $c=k$, $n_0=1$ will suffice

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

EXERCISE

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is its big-Oh behaviour?

EXERCISE

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is its big-Oh behaviour?

ANS: It is $O(n)$.

Proof: if $n \geq 1$ then $f(n) \leq n$

hence take $c=1$ $n_0=1$

EXERCISE

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is the limit of the ratio $f(n)/n$ as n tends to ∞ (infinity, i.e. becomes very large)?

EXERCISE (ans)

- Consider the function:

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

  $\quad\quad\;\; 1$ if n is odd

What is the limit of the ratio $f(n)/n$?

ANS: It does not have a limiting ratio

$f(n)/n = 1$ if n is even, limit is 1

$\quad\quad\quad\; 1/n$ if n is odd, limit is 0

Ratios vs. big-Oh

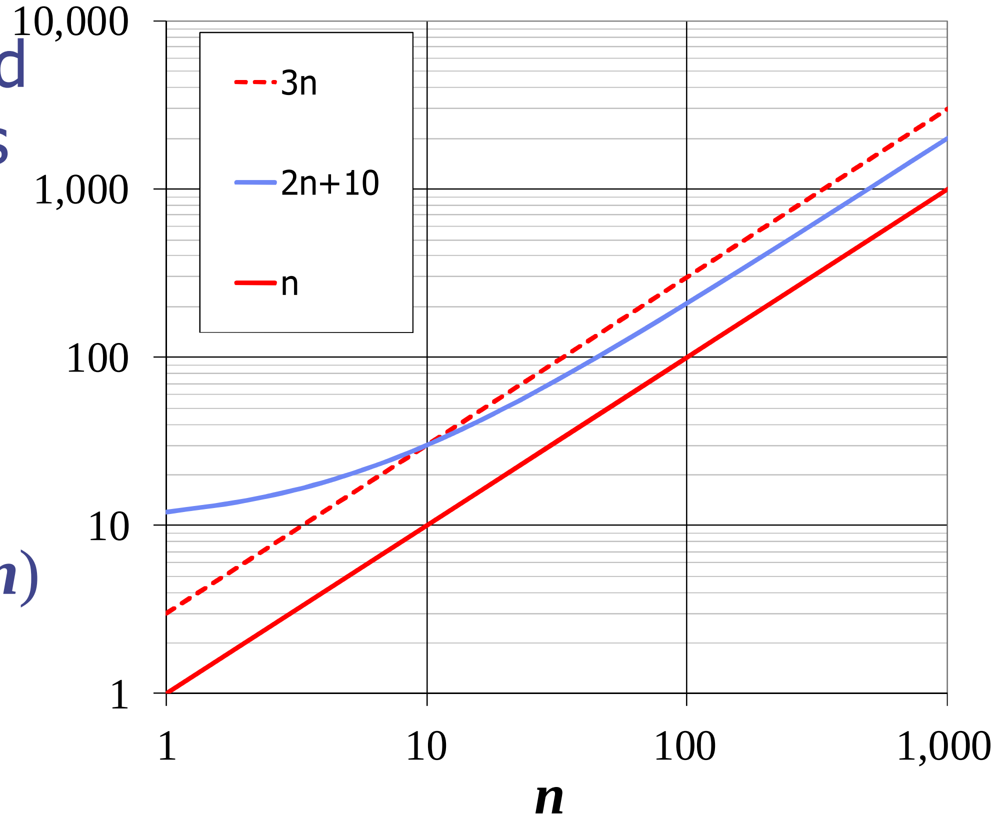
- f(n) can be O(g(n)) even if the ratio f(n)/g(n) does not exist
- Hence, big-Oh can be used in situations that ratios cannot
- The possibility of ‘weird functions’ means that big-Oh is more suitable than ratios for doing analysis of efficiency of programs

Big-Oh Notation: Graphically

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

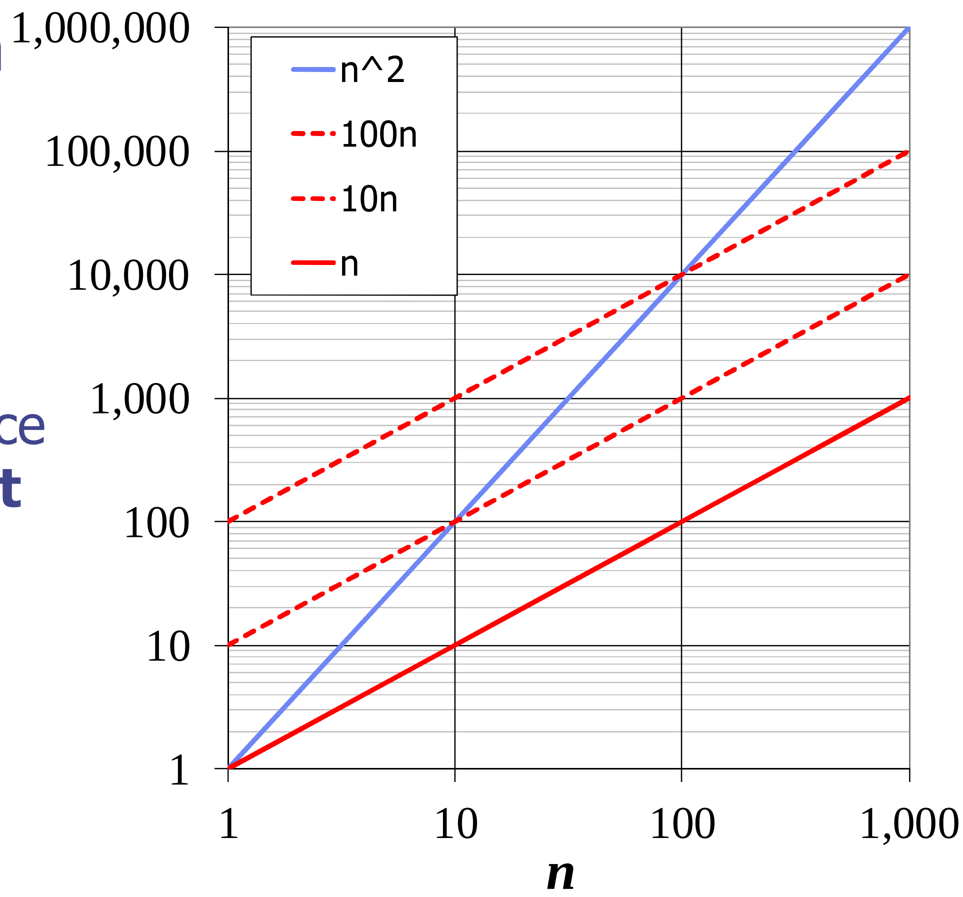
  $f(n) \le cg(n)$ for $n \ge n_0$

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \le cn$
  - $(c - 2)\, n \ge 10$
  - $n \ge 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$



Big-Oh Example

- Example: the function n^2 is not $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - The above inequality cannot be satisfied since c **must be a constant**
- Can also see this graphically:



More Big-Oh Examples

- $7n-2$

  $7n-2$ is $O(n)$

  need $c > 0$ and $n_0 \geq 1$ such that $7n-2 \leq c \bullet n$ for $n \geq n_0$

  this is true for example for $c = 7$ and $n_0 = 1$

- $3n^3 + 20n^2 + 5$

  $3n^3 + 20n^2 + 5$ is $O(n^3)$

  need $c > 0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ for $n \geq n_0$

  this is true for $c = 4$ and $n_0 = 21$

- $3 \log n + 5$

  $3 \log n + 5$ is $O(\log n)$

  need $c > 0$ and $n_0 \geq 1$ such that $3 \log n + 5 \leq c \bullet \log n$ for $n \geq n_0$

  this is true for $c = 8$ and $n_0 = 2$

Big-Oh and Growth Rate

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

	$f(n)$ is $O(g(n))$	$g(n)$ is $O(f(n))$
$g(n)$ grows more	Yes	No
$f(n)$ grows more	No	Yes
Same growth	Yes	Yes

Rules for Finding big-Oh

- Reverting to the definition each time is time-consuming and error prone
- Better to develop a set of rules that allow us to very quickly find big-Oh

"Multiplication Rule" for big-Oh

- Suppose
  - $f_1(n)$ is $O(g_1(n))$
  - $f_2(n)$ is $O(g_2(n))$
- Then, from the definition, there exist positive constants c_1 c_2 n_1 n_2 such that
  - $f_1(n) \leq c_1 g_1(n)$ for all $n \geq n_1$
  - $f_2(n) \leq c_2 g_2(n)$ for all $n \geq n_2$
- Let $n_0 = \max(n_1 , n_2)$, then multiplying gives
- $f_1(n) f_2(n) \leq c_1 c_2 g_1(n) g_2(n)$ for all $n \geq n_0$
- So $f_1(n) f_2(n)$ is $O(g_1(n) g_2(n))$

Big-Oh Rules: Drop smaller terms

- If $\boldsymbol{f(n)}$ = $(1 + h(n))$
  with $h(n) \rightarrow 0$ as $n \rightarrow \infty$
- Then $f(n)$ is $O(1)$

Proof (sketch):

- $h(n) \rightarrow 0$ as $n \rightarrow \infty$ means that for large enough n then it will become arbitrarily close to zero
- Hence, in particular, there exists n_0 such that
  $h(n) \leq 1$ for all $n \geq n_0$
- $f(n) \leq 2$ for all $n \geq n_0$
- Hence is $O(1)$ (by using $c=2$ in the definition)

Exercise

- What is the big-Oh of $f(n) = n^2 + n$?

Exercise

- What is the big-Oh of $f(n) = n^2 + n$?
- ANS:

  $f(n) = n^2 * (1 + 1/n)$

  $1 + 1/n$ is $O(1)$ by ‘drop small terms’

  n^2 is trivially $O(n^2)$

  then use multiplication rule,

  $f(n)$ is $O(n^2 * 1) = O(n^2)$

Big-Oh Rules

- After some thought it should become clear that:

- If $f(n)$ a polynomial of degree d, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant terms

Note: degree of a polynomial is the highest power
e.g. $5\ n^4 + 3\ n^2$ is degree 4

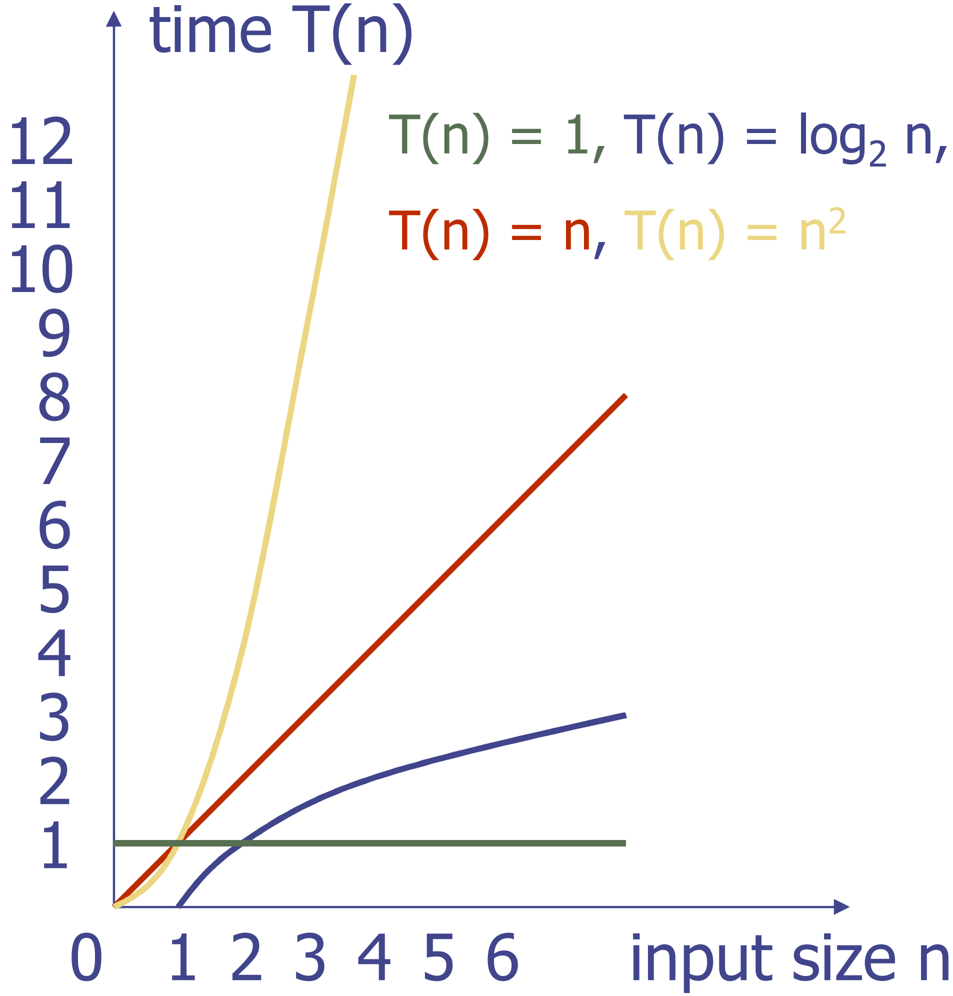
Seven Important Functions

Seven functions that **often** appear in algorithm analysis:

- Constant ≈ ***1***
- Logarithmic ≈ log ***n***
- Linear ≈ ***n***
- N-Log-N ≈ ***n*** log ***n***
- Quadratic ≈ n^2
- Cubic ≈ n^3
- Exponential ≈ 2^n

YOU NEED TO KNOW THESE WELL!!

Plot some graphs yourself



Useful limits: exponents vs. powers

- Exponentials grow faster (as $n \to \infty$) than any power:
- for any fixed k, and $b > 1$

$$b^n / n^k \to \infty$$

$$n^k / b^n \to 0$$

E.g. $n^2 / 2^n \to 0$

Exercise

- What is the big Oh of $f(n) = 2^n + n^2$?

Exercise

- What is the big Oh of $f(n) = 2^n + n^2$?
- ANS:

$$f(n) = 2^n * (1 + n^2 / 2^n)$$

but $n^2/2^n \rightarrow 0$ so can drop it.

Hence $f(n)$ is $O(2^n)$

Useful limits: powers vs. logs

- Powers grow faster (as $n \to \infty$) than any power of a log (assume positive powers)

$$n / (\log n) \to \infty$$
$$(\log n) / n \to 0$$

More generally:

$$(\log n)^k / n^{k'} \to 0 \quad \text{for any fixed } k, k' > 0$$

E.g. $(\log n)^{100} / n^{0.1} \to 0$

Though it might not be obvious until large n

Exercise

- What is the big Oh of $f(n) = (n \log n) + n^2$?

Exercise

- What is the big Oh of $f(n) = (n \log n) + n^2$?
- ANS:

  $f(n) = n^2 * ((\log n)/n + 1)$

  But $(\log n)/n \rightarrow 0$ so can drop it

  Hence $f(n)$ is $O(n^2)$

Exercises:

Give the big-Oh of the following

1. $3\,n^3 + 10000\,n$
2. $n \log(n) + 2\,n$
3. $2^n + n$

If need help, then ask in lab sessions.

Usage of 'O' in practice

- From the definition is true that for any f(n) that f(n) is O(f(n))
- But such an answer is inappropriate because conventions say that we want a 'useful' answer, not a trivial one.
- If asked for the 'big-Oh' then want the 'tightest nice function' - this is defined by community standards.
- Warning: This is a convention and not directly part of the definition – and so causes a lot of confusion.
- **It will be expected on exam questions that you use the conventions**

Big-Oh Conventions

Conventions:

- Use the smallest (slowest growing) ‘reasonable’ possible class of functions
  - Say “$2n$ is $O(n)$” instead of “$2n$ is $O(n^2)$”
- Use the simplest expression of the class
  - Say “$3n + 5$ is $O(n)$” instead of “$3n + 5$ is $O(3n)$”

Big-Oh as a "Set"

One can think of O(n) as

"the set of all functions whose growth is no worse than linear for sufficiently large n"

Hence, it can be thought of as the (infinite) set $\{1,2, \ldots \log n, 2 \log n, \ldots, n, 2n, 3n, \ldots, n+1, n+2, \ldots\}$

Then "2n-3 is O(n)" is just the statement that the function 2n-3 is in this set, i.e. $2n-3 \in O(n)$

Big-Oh as a "Set"

I recommend against writing $n = O(n)$, because

- $n = O(n^2)$
- $n = O(n)$

should lead to $O(n) = O(n^2)$ which is wrong!

Though, note that $O(n) \subset O(n^2)$.

That is, if $f(n) \in O(n)$ then $f(n) \in O(n^2)$

(though not the converse)

Big-Oh: Usage for Algorithms

Big-Oh definitions themselves, in pure sense, are just ways of classifying functions and not algorithms

Their usage for runtimes of algorithms has further choices. One can use big-Oh to describe any of:

- Worst case runtime, w(n), at each value of n
- Best case runtime, b(n), at each value of n
- Average case runtime, b(n), at each value of n
- etc

If simply say "algorithm X is O(.)" then the usual convention is that it will refer to the worst case.

Big-Oh: Usage for Algorithms

"Merge-sort is O(n log n)"

expands into

"If running merge-sort on n integers, then the worst case run-time over all possible inputs, is a member of the set of functions that, is no worse than some fixed constant times n log(n) for all values of n that are at least some fixed value."

"Also, there will be some inputs for which the runtime is as bad as this."

Summary So Far: 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
- Example:
  - We determine that algorithm ***arrayMax*** executes at most $8n - 3$ primitive operations
  - We say that algorithm ***arrayMax*** "runs in $O(n)$ time"
- Since constant factors and lower-order terms are eventually dropped anyhow, if we only want the big-Oh behaviour, then we can disregard them when counting primitive operations – but be careful not to drop the important terms!

“Algorithm” or “Problem”

- We will see that the big-Oh for a solution to a particular problem can depend on the algorithm used.
- Point: picking an appropriate algorithm is needed in order to get good behaviour
- It would often be nice to know, for a particular problem, the “best possible big-Oh”, but
  - such lower-bounds are very rarely known
  - it is very hard to do such analyses
  - there are many problems where the entire CS community has entirely failed to make progress, see Millennium prize on P vs. NP

Summary & Expectations

- Know the definition of big-Oh well!
- Be able to apply it, and prove results on big-Oh of simple functions
- Know how to manipulate
  - Sums
  - Products

  of functions, and be able to prove if needed