COMP2054-ADE

ADE Lec03
Rules for quick big-Oh proofs

Lecturer: Andrew Parkes

Email: andrew.parkes@Nottingham.ac.uk

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

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

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

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

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 $O(\ g_1(n)\ g_2(n)\ )$

"Multiplication Rule" for big-Oh

- "Sanity checks":
  - On doing something new it is usually good to consider some "trivial" cases and verify it works correctly - the "unit tests" of maths ☺
- E.g. What is the big-Oh of f(n) = 5 n ? Does it "get rid of the constants"?

Use 5 is O(1)

n is O(n)

Hence 5 * n is O (1 * n) which is O (n), as expected.

Big-Oh Rules: Drop smaller terms

- If $f(n) = (1 + h(n))$
  with $h(n) \to 0$ as $n \to \infty$
- Then f(n) is O(1)
- (The utility will be to combine with the multiplication rule).

Proof (sketch):

- $h(n) \to 0$ as $n \to \infty$ means that for large enough n then h(n) will become arbitrarily close to zero
- Hence, in particular, there exists n_0 such that
  $h(n) \le 1$ for all $n \ge n_0$
- So, $f(n) \le 2$ for all $n \ge n_0$
- Hence is f(n) 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, (with positive largest term) 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 and so will be $O(n^4)$
Proof (sketch):
rewrite the polynomial as $k * n^d * (1 + \ldots)$

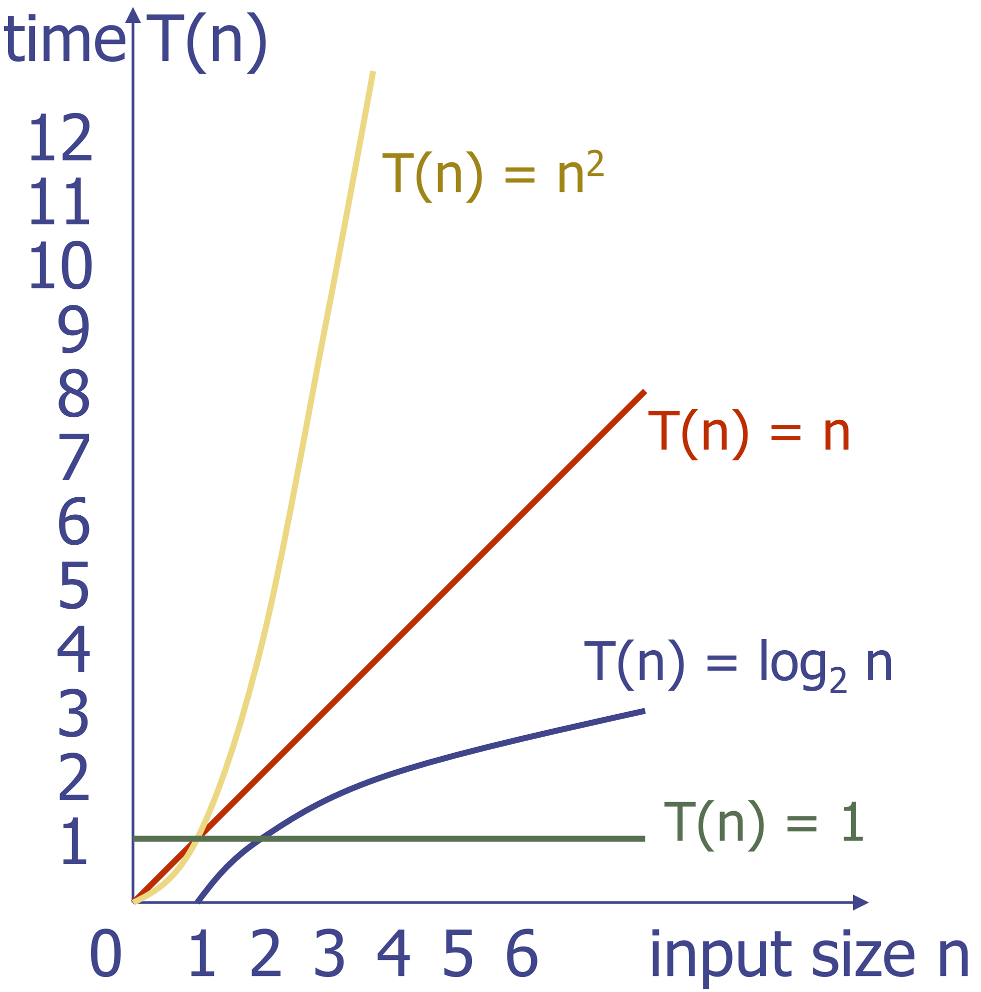
Seven Important Functions

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

- Constant ≈ $\mathbf{1}$
- Logarithmic ≈ log $\boldsymbol{n}$
- Linear ≈ $\boldsymbol{n}$
- N-Log-N ≈ $\boldsymbol{n}$ log $\boldsymbol{n}$
- Quadratic ≈ $\boldsymbol{n}^2$
- Cubic ≈ $\boldsymbol{n}^3$
- Exponential ≈ 2^n

YOU NEED TO KNOW THESE WELL!!

Plot some graphs yourself



Recall: "Drop smaller terms"

If $f(n) = (1 + h(n))$
with $h(n) \to 0$ as $n \to \infty$

Then f(n) is O(1)

- To use this we will need to have some rules for limits of various functions
  - E.g. what do we do with (1 + (log(n) / n))
  - Does h(n) = log(n) / n go to 0 as n -> infinity?
- If have a messy function can try plotting a graph.
- But there are some useful rules:

Useful limits: exponents vs. powers

- Exponentials grow faster (as $n \to \infty$) than any power:
- for any fixed $k>0$, and $b > 1$

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

  and

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

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

E.g. $n^{2000} / 2^{n/100} \to 0$

Exercise

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

Reminder on Terminology:

We say "n^2" is a "power law" but we do not call it an "exponential".
An "exponential function of n" is one with n in the exponent, such as 2^n not just having a constant exponent (such as the 2 in n^2).

The difference between n^2 and 2^n is enormous (try computing them at n=1000).

Exercise

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

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

but $n^2/2^n \to 0$ 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.

Roughly:

"**Exponentials dominate powers which dominate logs**"

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

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

Exercises (offline):

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, e.g. in labs/tutorials.

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 $O(3n)$”

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, and done so as to convey the maximum (practical) information
- Warning: This is a convention and not directly part of the definition – and so causes a lot of confusion.
- **It will be expected on assessments that you use the conventions when appropriate**

“Algorithm” or “Problem”

- We will see that the big-Oh for a solution to a particular problem depend on the algorithm used.
- Point: picking an appropriate algorithm is needed in order to get good behaviour, and the big-Oh helps this

- 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

Exercise

- What is the big Oh of $f(n) = 2^{n/100} + n^{200}$?
- ANS:

$$f(n) = 2^{n/100} * (1 + n^{200} / 2^{n/100})$$

but $n^{200}/2^{n/100} \to 0$ so can drop it.

Hence $f(n)$ is $O(2^{n/100})$.

But in practice, the large power law will dominate until n is very large.

"Big-Oh" is not a panacea – need to know when it might be misleading

Exercise

Consider an (extreme) example

- Algorithm A has runtime $f1(n) = n^{20}$
- Algorithm B has runtime $f2(n) = 2^{n/10}$

Which is the best?

- If only look at Big-Oh, then A wins
- But at n=100 we have
  - $f1(100) = 100^{20} = 10^{40}$
    far more than the 10^{80} atoms in the universe
  - $f2(100) = 2^{10} = 1024 \sim 10^3$
- Now B clearly wins.
  - Less extreme versions of this happen e.g. "simplex algorithm" (used in optimisation) is exponential in worst case, but often better in practice.
- "Big-Oh" is not a panacea!

Reminder: 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"

Although many sources do it, personally, 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)

The (subset) inclusion is strict
e.g. consider n^2 which is in $O(n^2)$ but not in $O(n)$

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

- 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

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

But have the freedom to say:
For algorithm X,

- the worst case (over all possible inputs) is $O(n^3)$
- the average case, given inputs generated uniformly at random, is $O(n^2)$
- the best case (over all possible inputs) is $O(n)$.

E.g. will see:
worst case of quicksort is $O(n^2)$ but average case is $O(n \log n)$

Big-Oh: Usage for Algorithms

Consider:

"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."

Which follows the definition, but the convention might also imply it is the best, and so we might add that the O is a "tight bound" or "cannot be improved" because:

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

Asymptotic Algorithm Analysis in practice

- The asymptotic analysis of an algorithm determines the running time in big-Oh notation
- To perform the asymptotic analysis
  - We find the (worst-case, etc.) 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"
- In practice:
  Since constant factors and lower-order terms are eventually dropped anyhow, if we only want the big-Oh behaviour, then we could decide to disregard them when counting primitive operations – but be careful not to drop the important terms!

Summary & Expectations

Included, (but not limited to)

- 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
- Know the "conventions of usage" and also the advanced usages e.g. "best case is O(.)" "$n^{O(1)}$"

Next Lecture

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?