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/

Today we continue with more about big-Oh.
The focus is on how it used in practice.

Big-Oh Notation: Definition***

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 $n \geq n_0$

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

COMP2054-ADE: big Oh rules

In the last lectures we say the definition of big-Oh – which I just repeat as it is so important to really "absorb" 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

We also did many proofs by using the definition from scratch.
However, this is going to be slow, and is not really what people usually do in practice.

Indeed part of the point of the way that big-Oh is constructed is that also has enough freedom that allows easier computations.

Hence we want “rules” or “ways of working” that allow to quickly decide 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 $O(g_1(n) g_2(n))$

COMP2054-ADE: big Oh rules

Given two functions, and their known big Oh behaviour.
Then what can we say we about the big-Oh of their product?

As usual, the first step is to just use the information that we are given.

As we have seen before, the first issue is that we have different values of the "n0" but we can merge these by just taking the maximum.

We can then just multiple – this being allowed as everything is non-negative.
Note, an example when number can be negative.

-2 <= -1

-1 <= -1

but multiplying to get +2 <= +1 would be wrong!

However, we assume that everything is positive (at least for n >= n0)

Then we get the limit on the product. Taking c=c1*c2 we then just get that the big-Oh of a product is the product of the big-Oh's.

"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.

COMP2054-ADE: big Oh rules

After doing something we generally should do some "trivial" examples so as to check that everything works correctly.

Best to do some simple checks before relying on something.

Here we do a trivial case.

Though it does rely on the proof in an earlier lecture that k is O(1) for any fixed k > 0.

Big-Oh Rules: Drop smaller terms

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

Proof (sketch):

- h(n) → 0 as n → ∞ 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) ≤ 1 for all n ≥ n_0
- So, f(n) ≤ 2 for all n ≥ n_0
- Hence is f(n) is O(1) (by using c=2 in the definition)

COMP2054-ADE: big Oh rules

The second very useful rule is one that you might already expect.

We have seen, for example that n+3 is O(n) and this is basically

because at large values of n, the "+3" is only a small extra effect compared to n.

Hence we want a rule to capture:

If we have that f is one plus an extra piece and that extra piece becomes small at large n, then the results if O(1).

The proof is just working through the meaning of something tending to zero as n tends to infinity.

In particular, we can pick a limit, in this case 1, and then we are guaranteed that there is some n0 such that the

function h will not exceed the limit once above n0.

Then the rest follows immediately because it puts a constant upper limit on f itself – for n >= n0.

We could argue that it also shows f(n) is O(2), but as discussed later one would not say this without good reason, and

There is no good reason here.

Exercise

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

So let's do an example that uses both rules.

We could easily do this "from the definition" (and you should do so as an exercise).
Again thinking of this as a way to check, and also help understand the rule.

Intuitively, at large values of n, the square will dominate, and it is reasonable to guess that it will be O(n^2).
But suppose we want to use the rules.
The catch is that it does not immediately match the "drop smaller terms" rule.
So what can we do? How can we rearrange so that we can use the rules?
Can we write as a product?

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

The key is to rewrite as a product in which one of the factors

can be reduced by using the "drop smaller terms" and then can put back the product.

Hence, since the largest term, and the expected big-Oh is the quadratic term, n^2, we want it as one factor.

So we rewrite f as "n^2 multiplied by something."
Then in the second term, as n goes to infinity, 1/n goes to zero as needed.
Hence it is O(1) using the rules.

So we then easily find the big-Oh of each of the factors.
Then put them back together again using the multiplication rule.

Of course, we could do this case direct from the definition, but we are using it to practice using the rules.

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 + …)

COMP2054-ADE: big Oh rules

With a bit of practice it should become "obvious" that this can be applied to any polynomial.
The highest power beats the other terms with lower powers, and we can also ignore the overall coefficient (assuming it is positive) and we just pick out the highest power.

Eventually this becomes natural.

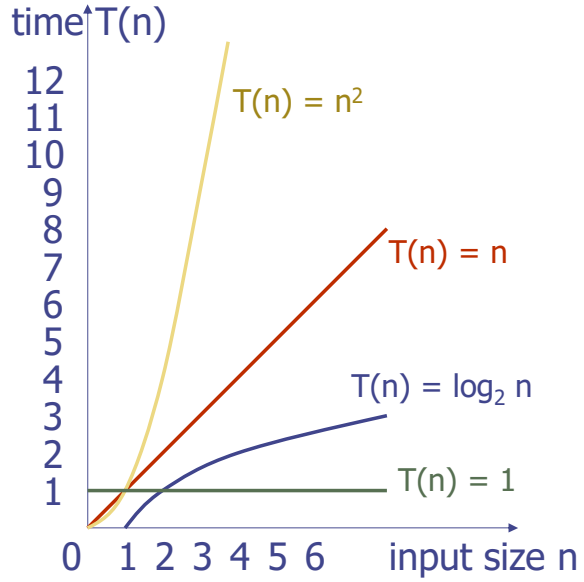
Seven Important Functions

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



COMP2054-ADE: big Oh rules

Well what kind of functions do we usually meet?
These are some of the most commons ones.

We will often see log(n) algorithms to access data in good data structures – it is common in data bases.

For sorting we get “n log n”
An example of cubic would be naïve matrix multiplication.
Exponentials happen in combinatoric search problems such as scheduling and timetabling.
(See the AIM module next semester, or maybe AI in first year)

You should have a good intuition about the shapes of the graphs and the relative growth rates.
Being able to use a calculator to compute specific value is not the same as having some intuitive familiarity with them.

Recall: "Drop smaller terms"

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

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:

COMP2054-ADE: big Oh rules

Given the rule for "drop smaller terms", in order to use it we will need to be able to use cases in which the limit of the function is zero.
For example, what if the function h has a log?

A good way to start is to use some system for doing graphs and so at least get an initial idea.

But we now do some rules that will be useful.

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

The first main rule is that an exponentials beat powers – they grow faster than any power.

The case we need is that a power law divided by an exponential will go to zero at large n

(with the restrictions on the parameters).

So a reasonable example is that a square over 2^n will go to zero.

Strongly recommend to plot some graphs yourself. E.g. Consider

n^20 / 1.01^n

n^8 + 2^n

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).

Let's use this to do a big-Oh using the rules.
Firstly, just a reminder on terminology/jargon.

If were referred to them both as exponentials then it makes it much harder to talk about "exponentials beat powers"!

Note that 1000^2 is only a million.
In contrast 2^1000 ~ 10^301 whereas the number of atoms (protons) in the (visible) universe is roughly estimated as only around 10^80.

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 $O(2^n)$

For this we gain follow the system of spotting the largest term as being the exponential and factor it out.

Then the second term is of the required form and the ratio tends to zero and can be dropped.

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:

$(\log n)^k / n^{k'} \to 0$ 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**"

If we think of "take of the log" of "exponents beat powers" then we get to "powers beat logs".

In this a similar kind of rule applies.

A power (with positive exponent) grows faster than a power of a log. Though in extreme cases might not be obvious.

Overall exponentials beat powers beat logs.

Exercise

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

Again let's do a simple example.

The previous results should make it natural the the quadratic term will be the dominating factor.

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 $O(n^2)$

As before we convert the dominating term to be an overall factor, and then get an expression were we have a terms with limit of zero, and that can be dropped.

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.

COMP2054-ADE: big Oh rules

As exercises offline, you should do these examples.

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 $O(3n)$"

COMP2054-ADE: big Oh rules

Now we move onto some more awkward issues of how big-Oh is used in everyday discussion.

These are not part of the definition, but are "community standards" of what is generally expected.

Mostly the idea is to use the big-Oh to convey useful information but remove unneeded details.

Strictly, from the definitions, we do try to give the slowest growing function inside the big-Oh, even if others are technically also correct.

If something is O(n) then would say that.

It would be considered unreasonable to say that 2n is O(n^2) without a good reason, even though it is technically correct.

This kind of "say the most useful thing" is a natural expectation in human-human communication.

Similarly part of the point of big-Oh is to allow to remove extraneous constants, so it would be considered as bad practice to keep them (unless there were very good reason).

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**

Similarly, if someone gives a function f and asks what is the big-Oh behaviour, then technically, if only thinking of the definitions one can just use reflexivity,

However, this conveys no extra information and would again be considered as bad practice.
The point of big-Oh is to convey information and understanding, hence it is expected that these reasonable community standards will be used.

"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

COMP2054-ADE: big Oh rules

A different question is whether we are analysing a problem, or a specific algorithm,

Almost invariably the analysis is of an algorithm.

It would be nice to be able to give the best possible big-Oh that is achievable using any algorithm.
But this is rarely achievable – there are few non-trivial cases where it is possible. (Though we will see that 'sorting' is one such case).
Indeed the whole P vs. P issue, which will be described briefly next semester in LAC, is about

whether a class of problems have any algorithm that is big-Oh of a power law, even though all known algorithms are all exponential.

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 $n^{200}/2^{n/100} \rightarrow 0$ 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

Besides the conventions, Big-Oh is not something that is necessarily the final useful description.

Consider again a sum of power and exponential,.

The big-Oh is driven by the exponential, but possibly the power law will dominate at all practical values of n.

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 ~ 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!

COMP2054-ADE: big Oh rules

Think of this in terms of deciding between algoriths,
One with a "bad power law" – bad because the exponent is large.

Versus a "good exponential" – good because the value of n is divided by constant.
If purely think in terms of big-Oh then the bad power law is the winner.

However, at practical values of n, then maybe the good exponential is actually faster,

These kinds of things can happen in practice – there are cases where an algorithm has exponential scaling in the worst case, but is almost always very good in practice.

The impact here is that one cannot just say "use the best big-Oh" but other considerations might also be needed.

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, ... log n, 2 log n, ..., n,2n,3n,...,n+1,n+2, ...}

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

COMP2054-ADE: big Oh rules

As we said before, it can also be helpful to think of big-Oh as being a set of functions and the "f is O()" to be like membership.

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 $O(n)$

COMP2054-ADE: big Oh rules

However, some literature uses "n = O(n)" for "n is O(n)".

You might need to recognise this as you might see it.

Personally I recommend (strongly) against using it, because we are used to "'=" being transitive, which then gives nonsense such as given,

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.

COMP2054-ADE: big Oh rules

Also, from the beginning it was stressed that big-Oh is a way to describe and classify functions.

This then gives a lot more freedom of usage than being tied to "big-Oh is worst case of an algorithm"
For example, one can compute the best and average runtimes of an algorithm,

These will also be functions and so it makes sense to be able to also classify them using big-Oh.

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)

COMP2054-ADE: big Oh rules

As an example, with some hypothetical algorithm X then we might have different big-Oh behaviour for the worst, average and best runtimes.

Indeed it is quite common that the average runtime is better than the worst case.

An example we will see later is quicksort.

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."

COMP2054-ADE: big Oh rules

Hence see that big-Oh is really just a very compressed form of communication. The full version of a simple statement can be quite long.

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!

COMP2054-ADE: big Oh rules

This is again a summary of what was said before.

But also, when doing a practical analysis of an algorithm it is likely to be known that some parts will not be contributing to the overall big-Oh, but will be faster, and so might not be analysed carefully.
Btu with the caveat it can take experience to be sure that one is not missing something; and so such shortcuts should not be done in the C/W!

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(.)" "$n^{O(1)}$"

Naturally the expectations are to understand this material, but the highlights are as given.

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?

COMP2054-ADE: big Oh rules

And just to finish, here is a look into a topic in the next lecture.

Big-Oh is a member of a family of similar “descriptions of scaling”, and the family is also useful.