ANALYSIS OF ALGORITHMS

Goal: Have some way of evaluating algorithms formally.

First idea: Why can't we just measure execution time?

* Dependent on how fast your computer is, how busy it is, ...
* Also dependent on programming language used
* Clunky to describe how performance scales with changing input.

Something better: Time complexity

* Fully abstract: uses math, not compute execution
* Captures scale by definition: complexity is relative to the input

How: Express the #. of primitive operations (arithmetic ops, memory/variable accesses, ...) used by the algorithm as a function of the input size, i.e.

$$f(n) = \left\{ \begin{array}{l} \text{\#. of prim. ops needed} \\ \text{for an input of size } n \end{array} \right\} \quad \text{e.g. } 5n+3,\ 2n^2+3n+4$$

Often we specialise to 3 types of time complexity

1. Worst case: #. ops required when the algorithm has to do the most work
2. Best case: opposite of worst case
3. Average case: #. ops required for the average input.

[Remark: average TC ≠ average of best and worst in general!]

It's usually worst case TC we care about.

Observation: We don't really need to be so specific about TC,

$$f_1(n) = 2n^2 + 5n + 3 \quad , \quad f_2(n) = 3n^2 + 6n + 4$$

will give similar performance for big inputs. What matters is the fact that both are of the form

constant $\cdot\, n^2$ + lower order terms

Big-O notation: A way of being able to regard similar complexities as the same thing in terms of limiting behaviour.

This is a formal math thing, but for 2521 we just need to know

1. $f(n)$ is $O(g(n))$ means that eventually, f will grow no faster than a constant multiple of g (i.e. $|f(n)| \leq C|g(n)|$, $n \geq n_0$)

2. Big-O ignores lower order terms and constant multiples:

   $2n^2 + 5n + 3$ is $O(n^2)$ because in the long run, the $2n^2$ bit is dominant, but also similar to just n^2

3. Always give the tightest possible Big-O class:

   $2n^2 + 5n + 3 = O(n^3)$ ✗ could be tighter
   $O(n^2)$ ✓ most useful answer

4. Hierarchy of common complexities:

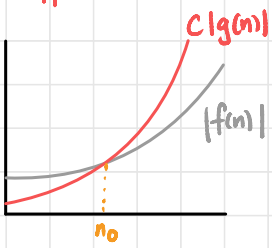
$$1 \xrightarrow{\text{is O}} \log n \xrightarrow{\text{is O}} n^\alpha, \alpha > 0 \xrightarrow{\text{is O}} \beta^n \text{ for } \beta > 1 \xrightarrow{\text{is O}} n! \xrightarrow{\text{is O}} \dots$$

constants | logarithms | polynomials | exponentials | factorials, other special things

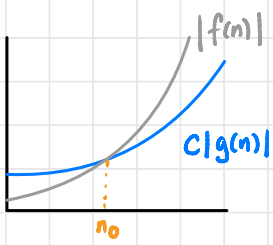
Related notions: Can similarly define

$O + \Omega$
$=$
Θ

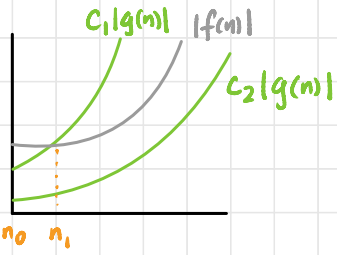
$O(g(n))$
Upper bound



$\Omega(g(n))$
Lower bound



$\Theta(g(n))$
Exact bound



Big-Ω, Θ not hugely important in 2521 though