Time Complexity of Algorithms

Additional reading

- How can the speed of the algorithm be measured in this case?
- The speed is dependent on the architecture of the computer
- The speed is dependent on the compiler that is being used
- The speed is dependent on the memory hierarchy
    - If the entire computation fits into the cache it will run faster
    - If it does not fit and starts doing lookups into RAM, things will be a lot slower
    - If you run out of memory and start writing up into the hard drive it gets even slower
- Different algorithms will have other runtimes
- The same algorithm will not work on everyone's computer at the same speed

The main aim is not to be able to make an estimate of runtime in seconds but to tell how runtime scales with the input size.

Note: All of these issues can multiply runtimes by a large constant

Asymptotic runtime

Growing rate: $\log n < \sqrt{n} < n < n \log n < n^2 < 2^n$

Big-O Notation

Definition: $f(n) = O(g(n))$ (f is a Big-O of g) or $f \leq g$ if there exist constants N and c so that for all $n \geq N$, $f(n) \leq c * g(n)$

- f is bounded by some constant multiple of g

Example: $3n^2 + 5n + 2 = O(n^2)$
Advantages:

- Clarifies growth rate
- Cleans up notation (see example)
- Can ignore complicated details (i.e. how fast is the cpu?)

Disadvantages

- It loses information like constant multiples

Common Rules

- Multiplicative constants can be omitted
  $7n^3 = O(n^3)$
- $n^a \leq n^b \; for \; 0 < a < b$
  $n = O(n^2)$, $\sqrt{n} = O(n)$

- $n^a \leq b^n \ (a > 0,\ b > 1)$ - exponential grows faster than polynomial
  $n^5 = O(\sqrt{2}^n),\ n^{100} = O(1.1^n)$
- $(\log n)^a \leq n^b \ (a,\ b > 0)$
  $(\log n)^3 = O(\sqrt{n}),\ n \log n = O(n^2)$
- Smaller terms can be omitted
  $n^2 + n = O(n^2),\ 2^n + n^9 = O(2^n)$