Computational Complexity (Algorithm Analysis).

*Need method to evaluate the runtime of an algorithm (i.e how fast is the algorithm).

+ often care about worst-case analysis (upper bound on performance)

+ want to understand how an algorithm will perform as the amount of input data increases

* sometimes it might be useful to also examine the average case execution, if an "average case" can be well defined.

big-O Notation:

If $g(n)$ is an upper bound of $f(n)$, then for some constant c, it is possible to find a value of n, n_0, for which any value of $n \geq n_0$ will result in $f(n) \leq cg(n)$

n is typically a variable used to represent the size of data.

+ For big-O notation, we want to capture the growth rate, so we do not have to be extremely precise.

Rules:

$O(c) = O(1)$ ≡ constant runtime not dependent on n

$O(cT) = cO(T) = O(T)$ ≡ executing tasks with a constant multiplicative number have the same growth rate

$O(T_1) + O(T_2) = O(T_1 + T_2) = \max\big(O(T_1), O(T_2)\big)$ ≡ if one algorithm has a dominant growth rate, we only need to capture this growth rate

$O(T_1)\,O(T_2) = O(T_1 T_2)$

Example: $T(n) = 3n^2 + 10n + 10$

$$O(T(n)) = O(3n^2 + 10n + 10) = O(3n^2) = \underline{\underline{O(n^2)}}$$

Common Algorithm Complexities:

$O(1)$

$O(\lg n)$

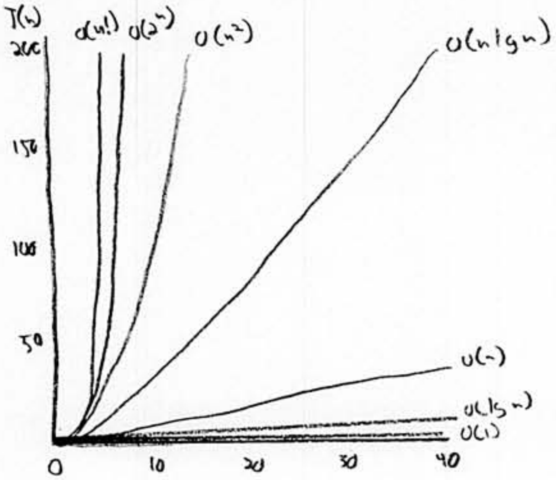
$O(n)$

$O(n \lg n)$

$O(n^2)$

$O(2^n)$

$O(n!)$


T(n)
O(n!)
O(2^n)
O(n^2)
O(n lg n)
O(n)
O(lg n)
O(1)