Algorithms: Complexity (Big Oh)

Analytical Time Complexity Analysis

- We would like to compare efficiencies of different algorithms for the same problem, instead of different programs or implementations. This removes dependency on machines and programming skill.

- It becomes meaningless to measure absolute time since we do not have a particular machine in mind. Instead, we measure the number of steps. We call this the *time complexity* or *running time* and denote it by $T(n)$.

- We would like to estimate how $T(n)$ varies with the input size n.

Guiding Principle #1

"worst – case analysis" : our running time bound holds for every input of length n.
-Particularly appropriate for "general-purpose" routines

As Opposed to
--"average-case" analysis
--benchmarks

REQUIRES DOMAIN KNOWLEDGE

BONUS : worst case usually easier to analyze.

Guiding Principle #2

Won't pay much attention to constant factors, lower-order terms

<u>Justifications</u>

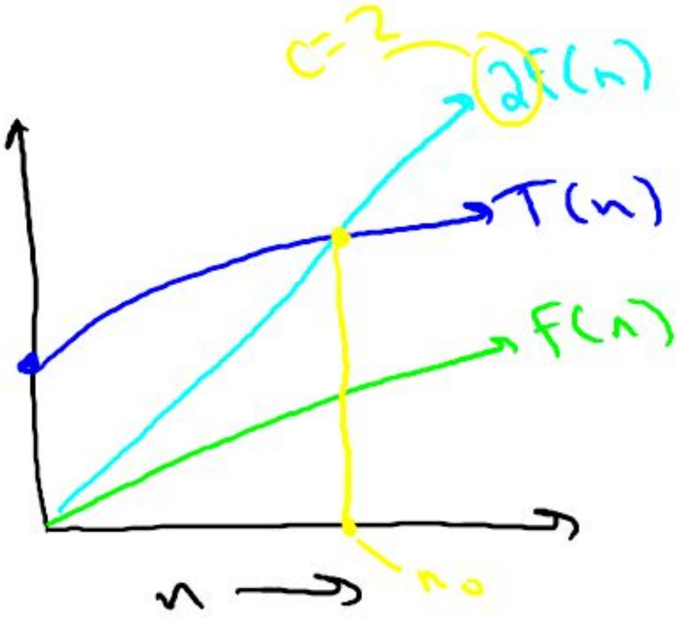
1. Way easier
2. Constants depend on architecture / compiler / programmer anyways
3. Lose very little predictive power (as we'll see)

Big-Oh: English Definition

Let $T(n)$ = function on $n = 1,2,3,\ldots$
[usually, the worst-case running time of an algorithm]

Q : When is $T(n) = O(f(n))$?
A : if eventually (for all sufficiently large n), $T(n)$ is bounded above by a constant multiple of $f(n)$

Big-Oh: Formal Definition



$Picture \;\; T(n) = O(f(n))$

Formal Definition : T(n) = O(f(n)) if and only if there exist constants $c, n_0 > 0$ such that

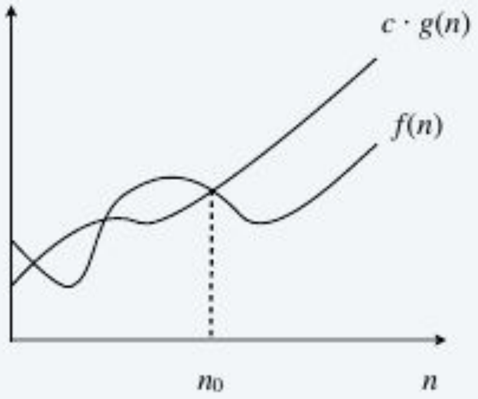
$$T(n) \leq c \cdot f(n)$$

For all $n \geq n_0$

Warning : c, n_0 cannot depend on n

Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



Ex. $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $O(n^2)$. ← choose $c = 50$, $n_0 = 1$
- $f(n)$ is neither $O(n)$ nor $O(n \log n)$.

Typical usage. Insertion sort makes $O(n^2)$ compares to sort n elements.

Big O notational abuses

One-way "equality." $O(g(n))$ is a set of functions, but computer scientists often write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.

Ex. Consider $g_1(n) = 5n^3$ and $g_2(n) = 3n^2$.

- We have $g_1(n) = O(n^3)$ and $g_2(n) = O(n^3)$.
- But, do not conclude $g_1(n) = g_2(n)$.

$g_2(n) = O(n^2)$ also can be written.

Domain and codomain. f and g are real-valued functions.

- The domain is typically the natural numbers: $\mathbb{N} \to \mathbb{R}$.
- Sometimes we extend to the reals: $\mathbb{R}_{\geq 0} \to \mathbb{R}$.
- Or restrict to a subset.

input size, recurrence relations

plotting, limits, calculus

Bottom line. OK to abuse notation in this way; not OK to misuse it.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.
Pf.

- $\exists\, c_1 > 0$ and $n_1 \geq 0$ such that $0 \leq f_1(n) \leq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
- $\exists\, c_2 > 0$ and $n_2 \geq 0$ such that $0 \leq f_2(n) \leq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
- Then, $0 \leq f_1(n) \cdot f_2(n) \leq \underbrace{c_1 \cdot c_2}_{c} \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \underbrace{\max\{ n_1, n_2 \}}_{n_0}$. ▪

Sums. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(\max\{g_1, g_2\})$.

ignore lower-order terms

Transitivity. If f is $O(g)$ and g is $O(h)$, then f is $O(h)$.

Ex. $f(n) = 5n^3 + 3n^2 + n + 1234$ is $O(n^3)$.

Example: $2n^2 = O(n^3)$, with $c = 1$ and $n_0 = 2$.

Examples of functions in $O(n^2)$:

n^2
$n^2 + n$
$n^2 + 1000n$
$1000n^2 + 1000n$
Also,
n
$n/1000$
$n^{1.99999}$
$n^2/\lg\lg\lg n$

pronounced “big-oh of …” or sometimes “oh of …”

O(…) means an upper bound

- Let $T(n)$, $g(n)$ be functions of positive integers.
  - Think of $T(n)$ as a runtime: positive and increasing in n.
- We say “$T(n)$ is $O\left(g(n)\right)$” if $T(n)$ grows no faster than $g(n)$ as n gets large.
- Formally,

$$T(n) = O\left(g(n)\right)$$

$$\Leftrightarrow$$

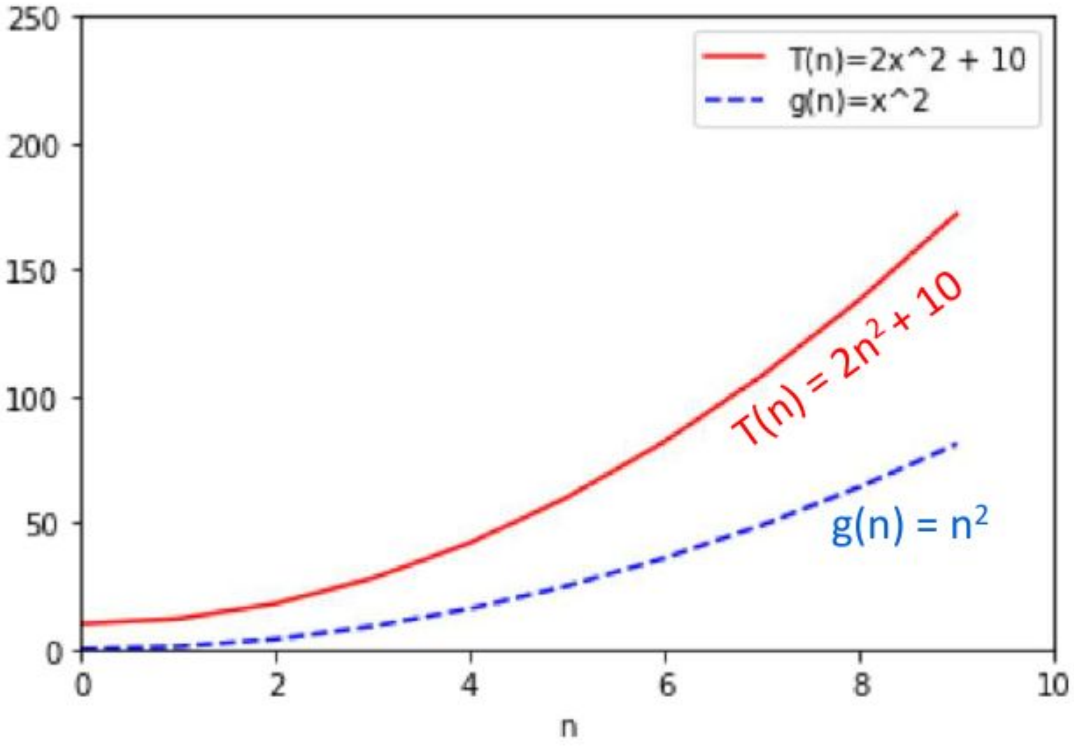
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$

$$0 \leq T(n) \leq c \cdot g(n)$$

Example

$2n^2 + 10 = O(n^2)$

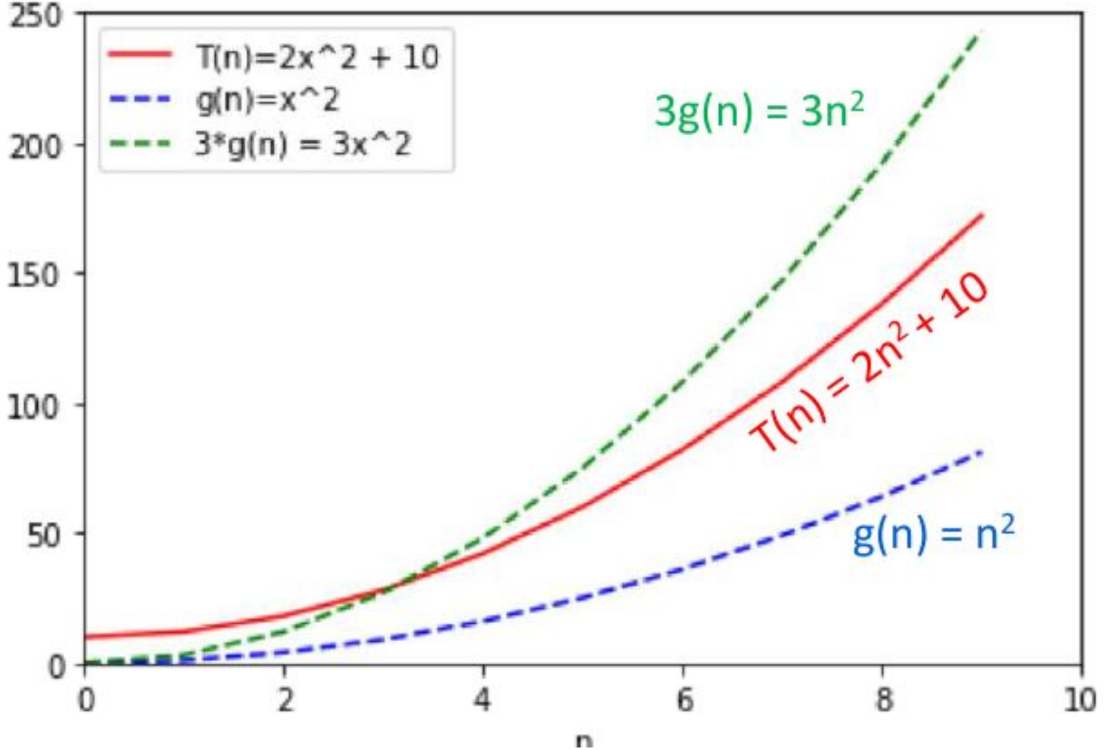
$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



Example

$$2n^2 + 10 = O(n^2)$$

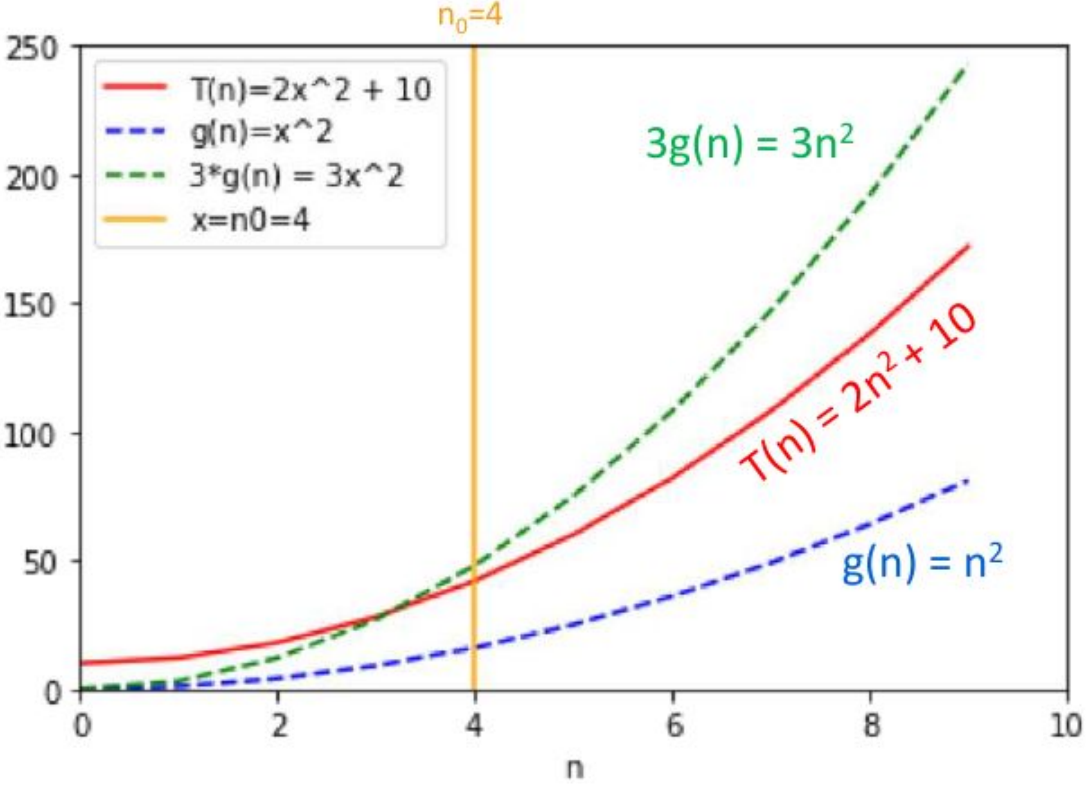
$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



Example

$2n^2 + 10 = O(n^2)$

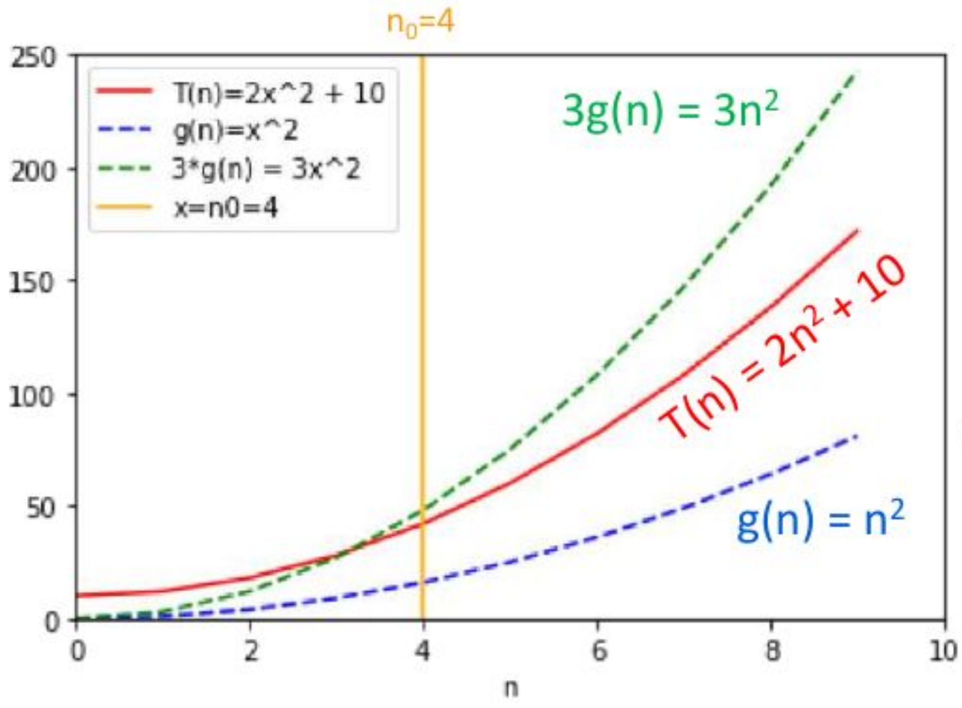
$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \ \ s.t. \ \ \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



Example

$2n^2 + 10 = O(n^2)$

$$T(n) = O\big(g(n)\big)$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



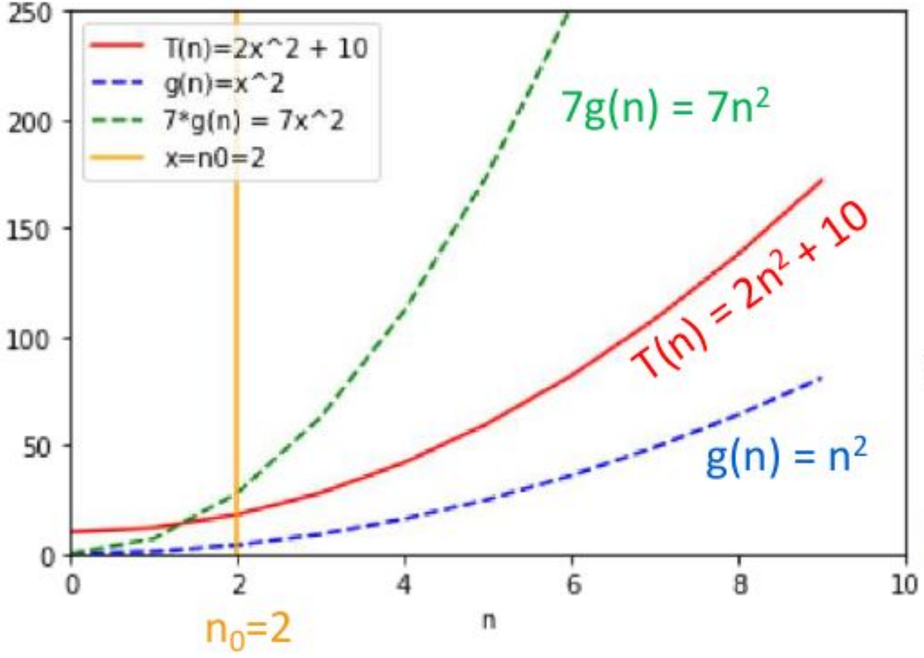
Formally:

- Choose c = 3
- Choose n_0 = 4
- Then:

$$\forall n \geq 4,$$
$$0 \leq 2n^2 + 10 \leq 3 \cdot n^2$$

Same example
$2n^2 + 10 = O(n^2)$

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



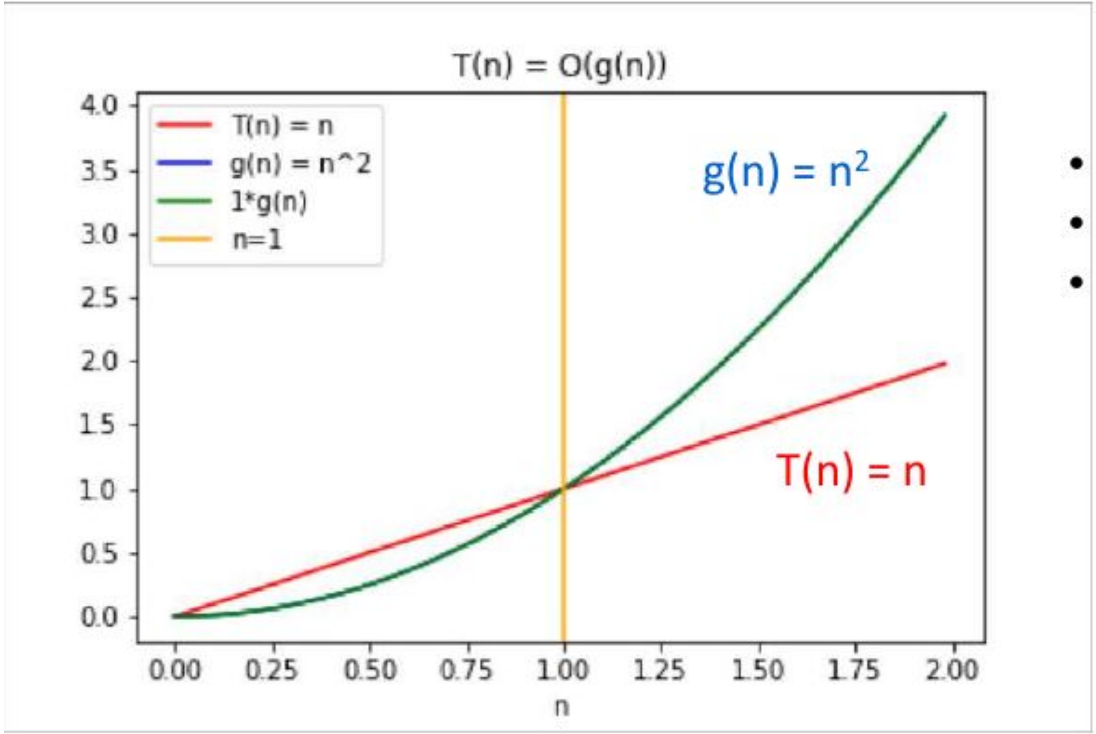
Formally:
- Choose c = 7
- Choose n_0 = 2
- Then:

$$\forall n \geq 2,$$
$$0 \leq 2n^2 + 10 \leq 7 \cdot n^2$$

There is not a "correct" choice of c and n_0

Another example: $n = O(n^2)$

$$T(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists c, n_0 > 0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$0 \leq T(n) \leq c \cdot g(n)$$



- Choose c = 1
- Choose n_0 = 1
- Then

$$\forall n \geq 1,$$
$$0 \leq n \leq n^2$$

Big O notation with multiple variables

Upper bounds. $f(m, n)$ is $O(g(m, n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and $n_0 \geq 0$ such that $0 \leq f(m, n) \leq c \cdot g(m, n)$ for all $n \geq n_0$ and $m \geq m_0$.

Ex. $f(m, n) = 32mn^2 + 17mn + 32n^3$.

- $f(m, n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $f(m, n)$ is $O(n^3)$ if a precondition to the problem implies $m \leq n$.
- $f(m, n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. In the worst case, breadth-first search takes $O(m + n)$ time to find a shortest path from s to t in a digraph with n nodes and m edges.

Big Oh Examples

$$
\begin{aligned}
3n^2 - 100n + 6 &= O(n^2)\ because\ 3n^2 > 3n^2 - 100n + 6 \\
3n^2 - 100n + 6 &= O(n^3)\ because\ .01n^3 > 3n^2 - 100n + 6 \\
3n^2 - 100n + 6 &\neq O(n)\ because\ c \cdot n < 3n^2\ \ when\ \ n > c
\end{aligned}
$$

Think of the equality as meaning *in the set of functions*.

Suggested Reading

- Algorithms (CLRS)
  - Chapter 3.1
    - Section:
- Algorithm illuminated (Part 1) by Tim Roughgarden
  - Chapter 2
    - Section 2.1, 2.2