Big Oh notation

Let *f(n)* and *g(n)* be non-negative, non-decreasing functions of n. We say f(n) = O(g(n)) iff

∃ +ve constants c and n_o such that

$$f(n) \le c \cdot g(n) \text{ for all } n \ge n_o$$

- I.e. *f(n)* grows no faster than *g(n)* (ignoring constant factors)
- Since we're ignoring constant factors, typically choose *g(n)* to be "simple" looking:

Big Oh notation

$T_A(n) = c_1 . n^2 + c_2 . n$ — $O(c_1 . n^2 + c_2 . n)$

$O(c_1 . n^2)$

$O(n^2)$

$T_B(n) = c_3 . n$ — $O(n)$

$T_C(n) = 4. n^2 + 10 . n$ — $O(n^2)$

$T_D(n) = 6 . n^2$ — $O(n^2)$

Big Oh notation

- We're typically interested in the "tightest" bound we can obtain on an algorithm's runtime complexity (the Theta bound)

- E.g., $T_B(n) = c_3 . n$

  ==> $T_b(n) = O(n^2)$ as well, but we're more interested in the bound $T_b(n) = O(n)$

Common asymptotic functions

Function	Name
1	constant function
log n	logarithmic
n	linear
n log n	---
n^2	quadratic
n^3	cubic
2^n	exponential
n!	factorial

n^k for any constant k: polynomial function

f(n) is O(one of these functions) ==> f(n) is O(every lower[↓] function as well)

Some results

Result: If $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$ then

- $T_1(n) + T_2(n) = O(f(n) + g(n))$
- $T_1(n) * T_2(n) = O(f(n) * g(n))$

- $T_1(n) - T_2(n)$? $O(f(n) - g(n))$
- $T_1(n) / T_2(n)$? $O(f(n) / g(n))$

Some results

Result : If $a_m > 0$ then

$$\left(\sum_{i=0}^{m} a_i \, x^i \right) = O(x^m)$$

Result : f(n) = O(g(n)) iff

$$\lim_{n \to \infty} f(n)/g(n) \le c$$

for some finite constant c