ECS 20 – Fall 2021 – P. Rogaway   Asymptotic Growth Rates

Comparing growth-rates of functions – Asymptotic notation and view

Motivate the notation. Will do big-O and Theta.
http://en.wikipedia.org/wiki/Big_O_notation

$\Omega(g) = \{ f: \mathbb{N} \to \mathbb{R}: \quad \exists c, N_0 \text{ s.t. } c\, g(n) \le f(n) \text{ for all } n \ge N_0 \}$

$O(g) = \{ f: \mathbb{N} \to \mathbb{R}: \quad \exists C, N_0 \text{ s.t. } f(n) \le C\, g(n) \text{ for all } n \ge N_0 \}$

$\theta(g) = \{ f: \mathbb{N} \to \mathbb{R}: \quad \exists c, C, N_0 \text{ s.t. } cg(n) \le f(n) \le C\, g(n) \text{ for all } n \ge N_0 \}$

OK to replace $\mathbb{N}$ by some arbitrary infinite subset of $\mathbb{R}^+$.

People often use "**is**" or "=" for "is a member of" or "is an anonymous element of"**.** I myself don't like this.

Examples:…

Reasons for asymptotic notation:

1. simplicity – makes arithmetic simple, makes analyses **easier**
2. When applied to running times: Routinely **good enough,** in practice, to get a feel for efficiency and to compare candidate solutions.
3. When applied to running times: Facilitates greater **model-independence**

Reasons against:

1. Hidden constants **can** matter
2. Might make you fail to care about things that one **should** think about
3. Not everything has an "n" value to grow

If $f \in O(n^2)$, $g \in O(n^2)$ the $f+g \in O(n^2)$

If $f \in O(n^2)$ and $g \in O(n^3)$ then $f+g \in O(n^3)$
If $f \in O(n \log n)$ and $g \in O(n)$ then $fg \in O(n^2 \log n)$
etc.
May write $O(f) + O(g)$, and other arithmetic operators

True/False:

If $f \in \Theta(n^2)$ then $f \in O(n^2)$ TRUE
etc…

n	$n \lg n$	n^2	n^3	2^n
10	30 ns	100 ns	1 µs	1 µs
100	700 ns	10 µs	1 ms	10^{13} years
1000	10 µs	1 ms	1 sec	10^{284} years
10000	100 µs	0.1 sec	17 mins	10^{3000} years
10^5	2 ms	10 sec	1 day	---
10^6	20 ms	17 mins	32 years	---
10^9	30 s	31 years	10^{10} years	---

Suppose 1 step = 1 ns (10^{-9} sec)
(about 5 cycles on latest Intel processors;
a generation-12 Core i9 runs at 5.2 GHz)

$5n^3 + 100n^2 + 100 \in O(n^3)$
If $f \in \Theta(n^2)$ then $f \in O(n^2)$ TRUE
$n! \in O(2^n)$ NO
$n! \in O(n^n)$ YES

(Truth: $n! = \Theta((n/e)^n \text{ sqrt}(n))$ --- indeed $n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}.$ (Stirling's formula)

Claim: $H_n = 1/1 + 1/2 + \ldots + 1/n = O(\lg n)$

Upper bound by 1 + integral_1^n $(1/x)dx = 1 + \ln(n) = O(\lg n)$

List common growth rates

θ (n!)
θ (2^n)
θ (n^3)
θ (n^2)
θ (n log n log log n)
θ (n lg n)
θ (n)
θ (sqrt(n)
θ (log n)
θ (1)

Exercise: where is \sqrt(n)

The highest degree term in a polynomial is the term that determines the asymptotic growth rate of that polynomial.

General rule: characterize functions in simplest and tightest terms that you can.

In general we should use the big-Oh notation to characterize a function as closely as possible. For example, while it is true that $f(n) = 4n^3 + 3n^2$ is $O(n^5)$ or $O(n^4)$, it is "better" to say that $f(n)$ is $O(n^3)$.
It is likewise inappropriate to include constant factors and lower order terms in the big-Oh notation. For example, it is poor usage to say that the function $2n^3$ is $O(4n^3 + 8n \log n)$, although it is technically correct. The "4" has no place in the expression, and the $8n \log n$ term doesn't below there, either.

"Rules" of using big-Oh:

- **If $f(n)$ is a polynomial of degree d, then $f(n)$ is $O(n^d)$**. We can drop the lower order terms and constant factors.
- **Use the smallest/closest possible class of functions**, for example, "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- **Use the simplest expression of the class, for example**, "$3n + 5$ is $O(n)$" instead of "$3n+5$ is $O(3n)$"

Example usages and recurrence relations

Intertwine examples with the analysis of the resulting recurrence relation

1. How long will the following **fragment of code** take [nested loops, second loop a nontrivial function of the first] -- something O(n^2)
2. How long will a computer program take, in the worst case, to run **binary search**, in the worst case? T(n) = T(n/2) + 1 -- reminder: have seen recurrence relations before, as with the **Towers of Hanoi** problem. – Then do another recurrence, say T(n) = 3T(n/2) + 1. Solution (repeated substitution) $n^{\log\_2 3}$ = $n^{1.5849...}$ What about T(n) = 3T(n/2) + n ? Or T(n) = 3T(n/2) + n^2 ? **[recursion tree]**
3. How many gates do you need to **multiply** two n-bit numbers using **grade-school** multiplication?
4. How many comparisons to "**selection sort**" a list of n elements? T(n) = 1 + T(n-1)
5. How many comparisons to "**merge sort**" a list of n elements? T(n) = T(n/2) + n
6. What's the running time of deciding SAT using the obvious algorithm? Careful.

Warning: don't think that asymptotic notation is only for talking about the running time or work of algorithms; it is a convenient way of dealing with functions in lots of domains

Table modified from Wikipedia

Notation	**Intuition**	**Formal Definition**
$f(n) \in O(g(n)$	f is bounded above by g (up to constant factor)	$\exists k > 0 \; \exists n_0 \; \forall n > n_0 \; f(n) \leq g(n) \cdot k$
$f(n) \in \Omega(g(n)$	f is bounded below by g	$\exists k > 0 \; \exists n_0 \; \forall n > n_0 \; g(n) \cdot k \leq f(n$
$f(n) \in \Theta(g(n)$	f is bounded above and below by g	$\exists k_1 > 0 \; \exists k_2 > 0 \; \exists n_0 \; \forall n > n_0$ $g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2$

Notation	Name	Example
$O(1)$	constant	Determining if a binary number is even or odd; Calculating $(-1)^n$; Using a constant-size lookup table
$O(\log \log n)$	double logarithmic	Number of comparisons spent finding an item using interpolation search in a sorted array of uniformly distributed values
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap
$O((\log n)^c)$ $c > 1$	polylogarithmic	Matrix chain ordering can be solved in polylogarithmic time on a parallel random-access machine.
$O(n^c)$ $0 < c < 1$	fractional power	Searching in a k-d tree
$O(n)$	linear	Finding an item in an unsorted list or in an unsorted array; adding two *n*-bit integers by ripple carry
$O(n \log^* n)$	*n* log-star *n*	Performing triangulation of a simple polygon using Seidel's algorithm, or the union–find algorithm. Note that $\log^*(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ 1 + \log^*(\log n), & \text{if } n > 1 \end{cases}$
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, quasilinear, or "*n* log *n*"	Performing a fast Fourier transform; Fastest possible comparison sort; heapsort and merge sort
$O(n^2)$	quadratic	Multiplying two *n*-digit numbers by a simple algorithm; simple sorting algorithms, such as bubble sort, selection sort and insertion sort; (worst case) bound on some usually faster sorting algorithms such as quicksort, Shellsort, and tree sort
$O(n^c)$	polynomial or algebraic	Tree-adjoining grammar parsing; maximum matching for bipartite graphs; finding the determinant with LU decomposition
$L_n[\alpha, c] = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$ $0 < \alpha < 1$	L-notation or sub-exponential	Factoring a number using the quadratic sieve or number field sieve
$O(c^n)$ $c > 1$	exponential	Finding the (exact) solution to the travelling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search
$O(n!)$	factorial	Solving the travelling salesman problem via brute-force search; generating all unrestricted permutations of a poset; finding the determinant with Laplace expansion; enumerating all partitions of a set

n	$n \lg n$	n^2	n^3	2^n
10	30 ns	100 ns	1 µs	1 µs
100	700 ns	10 µs	1 ms	10^{13} years
1000	10 µs	1 ms	1 sec	10^{284} years
10000	100 µs	0.1 sec	17 mins	10^{3000} years
10^5	**2 ms**	**10 sec**	**1 day**	---
10^6	**20 ms**	**17 mins**	**32 years**	---
10^9	**30 s**	**31 years**	**10^{10} years**	---

Suppose 1 step = 1 ns (10^{-9} sec)
(about 5 cycles on latest Intel processors;
a generation-12 Core i9 runs at 5.2 GHz)