Ch 16.3 Algorithm analysis

Estimate the efficiency of an algorithm. Usually measuring time and space.

There's a BIG difference between measuring performance for algorithms vs. programs.
- Algorithms - theoretical calculation, estimates, performance in the limit, average vs. worst case analysis, no accounting for implementation factors
- Programs - measuring actual runtimes, language specific, implementation specific, includes many "reality" factors

Definitions
Basic step/operation - an algorithm step that is simple and executes in constant time

Complexity analysis - estimate of the basic steps in an algorithm to process a problem of size N; common to have two kinds of complexity analysis: worst-case and average-case

Asymptotic complexity - complexity analysis as problem size (N) gets large; this represents the general case

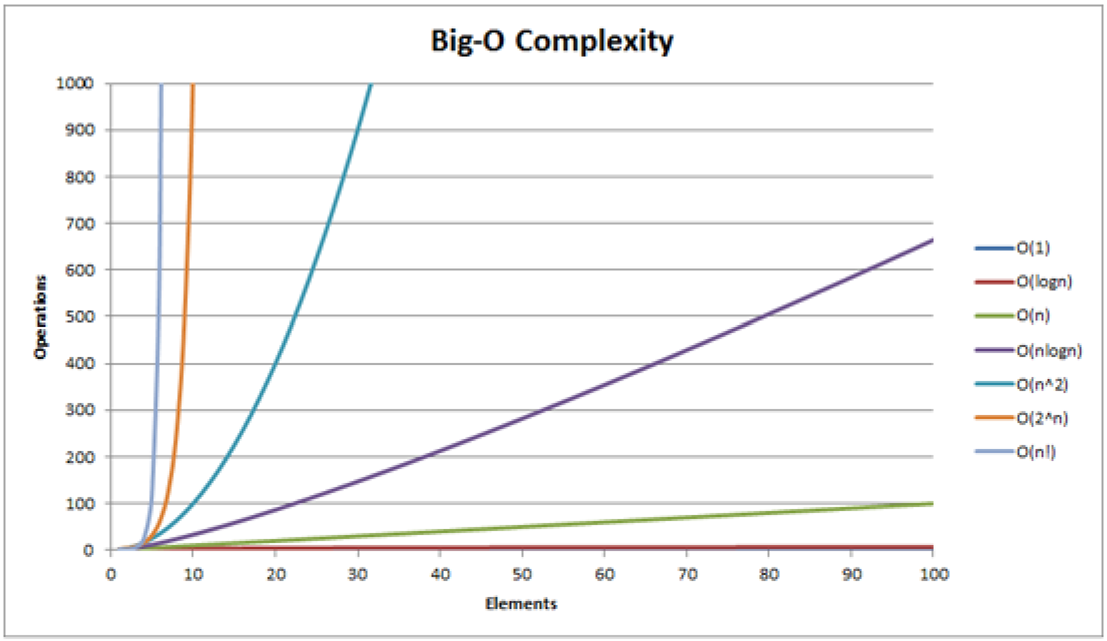
Big O notation
- Describes algorithm complexity at the limit as N approaches infinity
- Strip out constants and other terms overwhelmed as N gets large; example: $5n^2 + 10n + 7$ is $O(n^2)$
- Binary search is O(log n); this is pronounced "Big Oh of log N"
- "a computational problem is said to be in O(g(n)) if there exists an algorithm for the problem whose worst case complexity function is in O(g(n))"... page 995

This table is an expansion on the list on page 995 in our text:

Name	Big-O	Description/Example
constant time	O(1)	Run time is essentially independent of problem size Example: hash table lookup
logarithmic time	O(log n)	Run time increases slowly as problem size increases Example: Binary search
linear time	O(n)	Run time increases directly along with problem size Example: Sequential search
"n log n" time	O(n*log n)	Run time increases slightly faster Example: Quicksort
quadratic time	$O(n^2)$	Run time increases as a square of the problem size. Example: Bubble sort, nested loops

polynomial time	$O(n^c)$ where c>=1	Union of all algorithms that perform at $O(n)$, $O(n^2)$, $O(n^3)$, …
exponential time	$O(c^n)$ where c>	Run time increases very rapidly as problem size increases. Example: Travelling salesman problem
factorial time	$O(n!)$	Run time explodes. $20! = 10^{18}$. Example: Brute force travelling salesman

Big-O Complexity Classes
Source: www.daveperrett.com/articles/2010/12/07/comp-sci-101-big-o-notation/



Big-O	Operations for 10 “things”	Operations for 100 “things”
O(1)	1	1
O(log n)	3	7
O(n)	10	100
O(n log n)	30	700
O(n^2)	100	10000
O(2^n)	1024	2^100
O(n!)	3628800	100!

Even 100 “things” is tiny. What if your algorithm has to process 100K “things”?