



Algorithm Analysis

Wellesley College CS230
Lecture 20
Tuesday, April 17
Handout #32

PS5 due 11:59pm Wednesday, April 18
Final Project Phase 2 (Program Outline) due 1:30pm Tuesday, April 24

20 - 1



Overview of Today's Lecture

- Motivation: determining the efficiency of algorithms
- Review of functions
- Best-case, average-case, and worst-case efficiency
- Asymptotic notation: a mathematical tool for describing algorithm efficiency
- Examples: sorting algorithms, set implementations

Next time = another mathematical tool for analyzing algorithms: **recurrence equations**

This is just an appetizer. For the full meal, take **CS231 (Fundamental Algorithms)**

20 - 2



Determining the Efficiency of Algorithms

- What is an **efficient** algorithm?
 - **Time**: Faster is better.
 - How do you measure time? Wall clock? Computer clock? Abstract operations?
 - **Space**: Less is better
 - Which space? Input data? Intermediate data? Output data?
 - Where space is can affect time: registers, cache, main memory, remote machine
- **Algorithm analysis** provides tools for determining the efficiency of an algorithm and comparing algorithms.
- Algorithm analysis should be **independent of**
 - specific implementations and coding tricks (PL, control constructs)
 - specific computers (HW chip, OS, clock speed)
 - particular set of data
- **But size of data** should matter
- **Order of data** can also matter

20 - 3



How do Your Functions Grow?

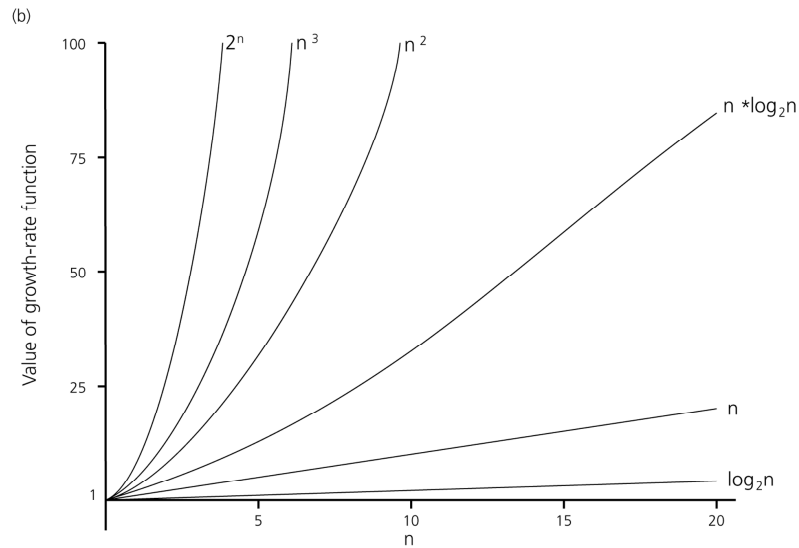
function f(n) =	name	n=10	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶
1	constant	1	1	1	1	1	1
log ₂ n	logarithmic	3.3	6.64	9.97	13.3	16.6	19.9
n	linear	1	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶
n · log ₂ n		33	664	10 ⁴	1.3x 10 ⁵	1.7x 10 ⁶	2x 10 ⁸
n ²	quadratic	10 ²	10 ⁴	10 ⁶	10 ⁸	10 ¹⁰	10 ¹²
n ³	cubic	10 ³	10 ⁶	10 ⁹	10 ¹²	10 ¹⁵	10 ¹⁸
2 ⁿ	exponential	10 ³	10 ³⁰	10 ³⁰¹	10 ³⁰¹⁰	10 ³⁰¹⁰³	10 ³⁰¹⁰³⁰

For many problems, **n · log₂ n** and less is considered **good** and **n³** and greater is considered **bad**, but really depends on the problem. E.g., linear is bad for sorted array search and exponential is the best time for Towers of Hanoi.

20 - 4



Graphical Comparison of Functions



20 - 5



Some Canonical Examples

Constant time: adding elt to end of vector, prepending elt to beginning of list, push/pop on efficient stack, enq/deq on efficient queue.

Logarithmic time: binary search on sorted array/vector, search/insertion/deletion on balanced binary search tree.

Linear time: adding elt to beginning of vector, putting elt at end of immutable list, copying array/vector/list, linear search on array/vector/list, insertion-sort on nearly-sorted array/vector/list

$n \log n$ time: merge-sort on array/vector/list, heap-sort on array/vector/list, quick-sort, tree-sort on unsorted array/vector/list, creating balanced BST with n elements

Quadratic time: insertion-sort on unsorted array/vector/list, selection-sort on any array/vector/list, quick-sort and tree-sort on sorted array/vector/list.

Cubic time: search on unsorted $n \times n$ array

Exponential time: Towers of Hanoi, Traveling Salesman problem, most optimization problems

20 - 6



Caution: What is n ?

n is typically the number of elements in a collection
(array, vector, list, tree, matrix, graph, stack, queue,
set, bag, table, etc.)

Changing the definition of n changes the analysis:

- Search on an unsorted square matrix is linear in the number of elements but quadratic in the side length.
- Search on a balanced BST is logarithmic in its number of nodes but linear its height.
- Search on a unsorted full tree is linear in its number of nodes but exponential in its height.

20 - 7



Asymptotics: Motivation

Fine-grained bean-counting exposes too much detail for comparing functions.

Want a course-grained way to compare functions that ignores constant factors and focuses.

Example:

function	$n=1$	$n=10^3$	$n=10^6$
$p(n) = 100n + 1000$			
$q(n) = 3n^2 + 2n + 1$			
$r(n) = 0.1n^2$			

20 - 8



Asymptotics: Summary

Notation	Pronunciation	Loosely
$f \in \omega(g)$	f is way bigger than g	$f > g$
$f \in \Omega(g)$	f is at least as big as g	$f \geq g$
$f \in \Theta(g)$	f is about the same as g	$f = g$
$f \in O(g)$	f is at most as big as g	$f \leq g$
$f \in o(g)$	f is way smaller than g	$f < g$

- Each of $\omega(g)$, $\Omega(g)$, ... denotes a **set of functions**. E.g., $\omega(g)$ is the set of all functions way bigger than g, etc.
- The notation $f = \omega(g)$ is really shorthand for $f \in \omega(g)$, and similarly for the other notations.
- The phrases “is at least $O(g)$ ” and “is at most $\Omega(g)$ ” are non-sensical.

20 - 9



Asymptotic Analysis

Order of growth of some common functions

- $O(1) \subset O(\log_2 n) \subset O(n) \subset O(n * \log_2 n) \subset O(n^2) \subset O(n^3) \subset O(2^n)$
- Intuitively,
 $\Theta(1) \prec \Theta(\log_2 n) \prec \Theta(n) \prec \Theta(n * \log_2 n) \prec \Theta(n^2) \prec \Theta(n^3) \prec \Theta(2^n)$

Properties of growth-rate functions

- You can ignore low-order terms
- You can ignore a multiplicative constant in the high-order term
- $\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n))$, and similarly for other notations.

20 - 10