CSE 373 Analysis of Algorithms
Fall 2016
Instructor: Prof. Sael Lee

LEC01
ASYMPTOTIC NOTATIONS (PP. 31-41)

Lecture slide courtesy of Prof. Steven Skiena

EXERCISE 1-6

The *set cover problem* is as follows: given a set of subsets S_1, ..., S_m of the universal set $U = \{1, ..., n\}$, find the smallest subset of subsets $T \subset S$ such that $\cup_{ti \in T} ti = U$. For example, there are the following subsets, $S_1 = \{1, 3, 5\}$, $S_2 = \{2, 4\}$, $S_3 = \{1, 4\}$, and $S_4 = \{2, 5\}$. The set cover would then be S_1 and S_2.

Find a counterexample for the following algorithm: Select the largest subset for the cover, and then delete all its elements from the universal set. Repeat by adding the subset containing the largest number of uncovered elements until all are covered.

APPLICATIONS OF SET COVER?

- Efficiently acquire items that have been packaged in a fixed set of lots. Seeking a collection of at least one of each distinct type of item.

- Boolean logic minimization of k variables which describes whether the desired output is 0 or 1 for each of the 2^k possible input vectors. Given a set of feasible "and" terms, each of which covers a subset of the vectors we need, we seek to "or" together the smallest number of terms that realized the function.

- For more on Set Cover, read pg 621-624

THE RAM MODEL OF COMPUTATION

Algorithms are an important and durable part of computer science because they can be studied in a machine/language independent way.

This is because we use the **RAM model of computation** for all our analysis.

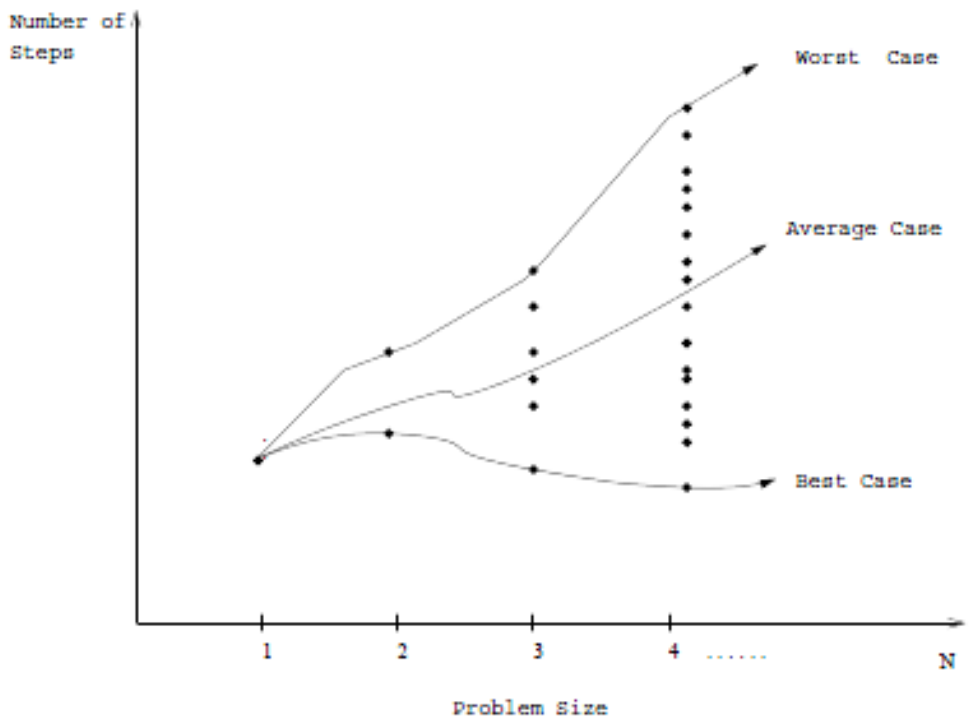
- Each "*simple*" operation (+,*, -, =, if, call) takes 1 step.
- Loops and subroutines are *not* simple operations. They depend upon the size of the data and the contents of a subroutine.
  - ex> "Sort" is not a single step operation.

- Each memory access takes exactly one time step (in cache or on disk) Further, we have as much memory as we need.

- We measure the run time of an algorithm by <u>counting the number of steps</u>, where:
  - This model is useful and accurate in the same sense as the flat-earth model (which *is* useful)!

WORST-CASE TIME COMPLEXITY

The *worst case (time) complexity* of an algorithm is the function defined by the maximum number of steps taken on **any** instance of size n.



BEST-CASE AND AVERAGE-CASE COMPLEXITY

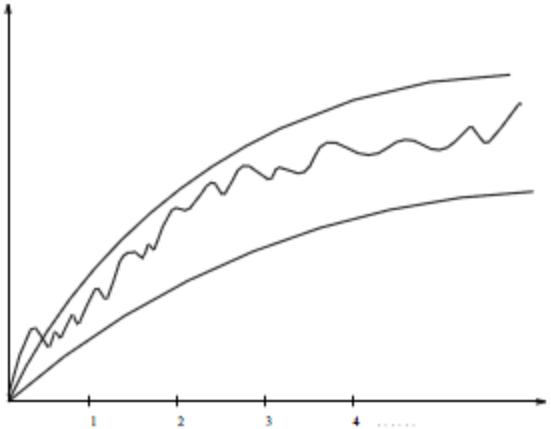
The *best case (time) complexity* of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.

The *average-case (time) complexity* of the algorithm is the function defined by an average number of steps taken on any instance of size *n*.

Each of these complexities defines a numerical function: time vs. size!

EXACT ANALYSIS IS HARD!

Best, worst, and average are difficult to deal with precise ly because the details are very complicated:



It easier to talk about *upper and lower bounds* of the fun ction. **Asymptotic notation (O, Θ, Ω)** are as well as we can practically deal with complexity functions.

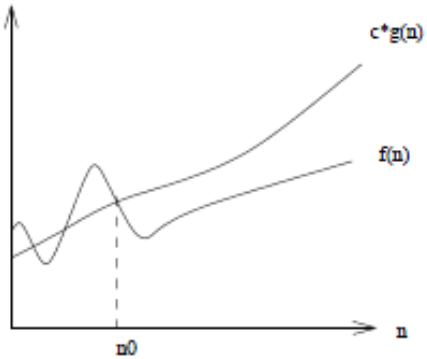
NAMES OF BOUNDING FUNCTIONS

- Big-Oh: $g(n) = \mathbf{O}(f(n))$ means $C*f(n)$ is an *upper bound* on $g(n)$.
- Big-Omega: $g(n) = \mathbf{\Omega}(f(n))$ means $C*f(n)$ is a *lower bound* on $g(n)$.
- Big-Theta: $g(n) = \mathbf{\Theta}(f(n))$ means $C_1*f(n)$ is an upper bound on $g(n)$ and $C_2*f(n)$ is a lower bound on $g(n)$. (a.k.a. *tight bound*)

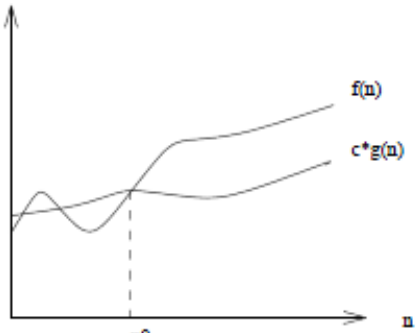
C, C_1, and C_2 are all constants independent of n.

FORMAL DEFINITIONS

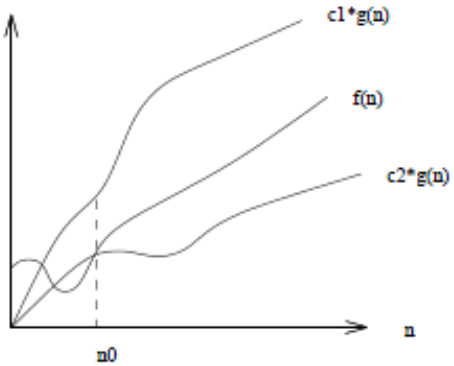
- $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of f(n) always lies on or below c*g(n).



- $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of f(n) always lies on or above c*g(n).



- $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that to the right of n_0, the value of f(n) always lies between c_1*g(n) and c_2*g(n) inclusive.



The definitions imply a constant n_0 *beyond which* they are satisfied. We do not care about small values of n.

BIG OH EXAMPLES

$3n^2 - 100n + 6 = O(n^2)$, because I choose $c = 3$ and $3n^2 > 3n^2 - 100n + 6$;

$3n^2 - 100n + 6 = O(n^3)$, because I choose $c = 1$ and $n^3 > 3n^2 - 100n + 6$ when $n > 3$;

$3n^2 - 100n + 6 \neq O(n)$, because for any c I choose $c \times n < 3n^2$ when $n > c$;

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

Stop and Think: Hip to the Squares?

Problem: Is $(x + y)^2 = O(x^2 + y^2)$.

BIG OMEGA EXAMPLES

$3n^2 - 100n + 6 = \Omega(n^2)$, because I choose $c = 2$ and $2n^2 < 3n^2 - 100n + 6$ when $n > 100$;

$3n^2 - 100n + 6 \neq \Omega(n^3)$, because I choose $c = 3$ and $3n^2 - 100n + 6 < n^3$ when $n > 3$;

$3n^2 - 100n + 6 = \Omega(n)$, because for any c I choose $cn < 3n^2 - 100n + 6$ when $n > 100c$;

BIG THETA EXAMPLES

$3n^2 - 100n + 6 = \Theta(n^2)$, because both O and Ω apply;

$3n^2 - 100n + 6 \neq \Theta(n^3)$, because only O applies;

$3n^2 - 100n + 6 \neq \Theta(n)$, because only Ω applies.

Stop and Think: Back to the Definition

Problem: Is $2^{n+1} = \Theta(2^n)$?

GROWTH RATES

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Growth rates of common functions measured in nanoseconds

DOMINANCE RELATIONS

- Faster-growing function ***dominates*** a slower-growing one
- Common functions that appear in algorithms analysis order of increasing dominance:

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$$

  - *Constant functions,* $f(n) = 1$
  - *Logarithmic functions,* $f(n) = \log n$
  - *Linear functions,* $f(n) = n$
  - *Superlinear functions,* $f(n) = n \lg n$
  - *Quadratic functions,* $f(n) = n^2$
  - *Cubic functions,* $f(n) = n^3$
  - *Exponential functions,* $f(n) = c^n$ *for a given constant* $c > 1$
  - *Factorial functions,* $f(n) = n!$

IMPLICATIONS OF DOMINANCE

- Exponential algorithms gets hopeless fast.
- Quadratic algorithms get hopeless at or before 1,000,000.
- O(n log n) is possible to about one billion.
- O(log n) never sweats.

TESTING DOMINANCE

- f(n) dominates g(n) if

$$\lim_{n\to\infty} g(n)/f(n) = 0,$$

  which is the same as saying g(n) = o(f(n)).
- Note: little-oh means “grows strictly slower than”.

IMPLICATIONS OF DOMINANCE

- n^a dominates n^b if a > b since

$$\lim_{n \to \infty} n^b / n^a = n^{b-a} \to 0$$

- $n^a + o(n^a)$ doesn't dominate n^a since

$$\lim_{n \to \infty} n^a / (n^a + o(n^a)) \to 1$$

ADVANCED DOMINANCE RANKINGS

- Additional functions arise in more sophisticated analysis than we will do in this course

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg \log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1$$

ASYMPTOTIC NOTATIONS: ADDITION/SUBTRACTION

Suppose $f(n) = O(n^2)$ and $g(n) = O(n^2)$.

- What do we know about g'(n) = f(n)+g(n)?
  - Adding the bounding constants shows $g'(n) = O(n^2)$.
- What do we know about $g''(n) = f(n) - |g(n)|$?
  - Since the bounding constants don't necessary cancel, $g''(n) = O(n^2)$
- We know nothing about the lower bounds on g' and g'' because we know nothing about lower bounds on f and g.
- <u>Since the Big Oh gives an upper bound, we can drop any negative term without invalidating the upper bound.</u>

The sum of two functions is governed by the dominant one,

$$O(f(n)) + O(g(n)) \rightarrow O(\max(f(n), g(n)))$$

$$\Omega(f(n)) + \Omega(g(n)) \rightarrow \Omega(\max(f(n), g(n)))$$

$$\Theta(f(n)) + \Theta(g(n)) \rightarrow \Theta(\max(f(n), g(n)))$$

ASYMPTOTIC NOTATIONS: MULTIPLICATION

- Multiplication by a constant does not change the asymptotic notations:

$$O(c \cdot f(n)) \rightarrow O(f(n))$$

$$\Omega(c \cdot f(n)) \rightarrow \Omega(f(n))$$

$$\Theta(c \cdot f(n)) \rightarrow \Theta(f(n))$$

- when two functions in a product are increasing, both are important.

$$O(f(n)) * O(g(n)) \rightarrow O(f(n) * g(n))$$

$$\Omega(f(n)) * \Omega(g(n)) \rightarrow \Omega(f(n) * g(n))$$

$$\Theta(f(n)) * \Theta(g(n)) \rightarrow \Theta(f(n) * g(n))$$

ASYMPTOTIC NOTATIONS: MULTIPLICATION BY FUNCTION

- But when both functions in a product are increasing, both are important:

$$O(f(n)) * O(g(n)) \rightarrow O(f(n) * g(n))$$

$$\Omega(f(n)) * \Omega(g(n)) \rightarrow \Omega(f(n) * g(n))$$

$$\Theta(f(n)) * \Theta(g(n)) \rightarrow \Theta(f(n) * g(n))$$