COT 6405
ANLYSIS OF ALGORITHMS

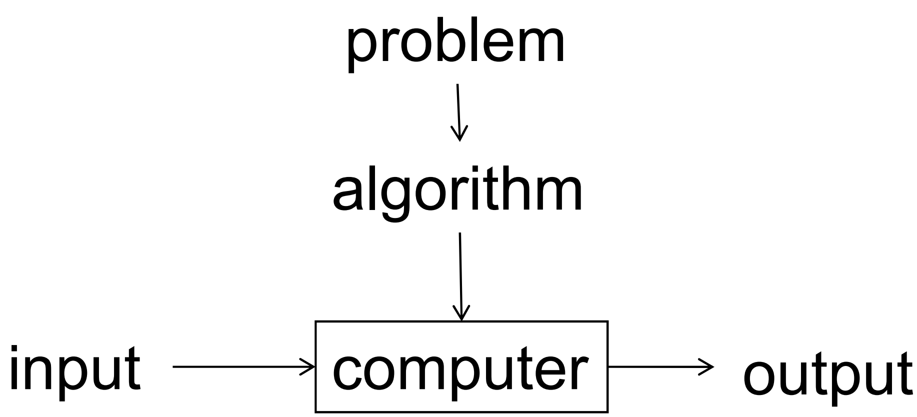
Growth of Functions and Recurrences

Computer & Electrical Engineering and Computer Science Dept.
Florida Atlantic University

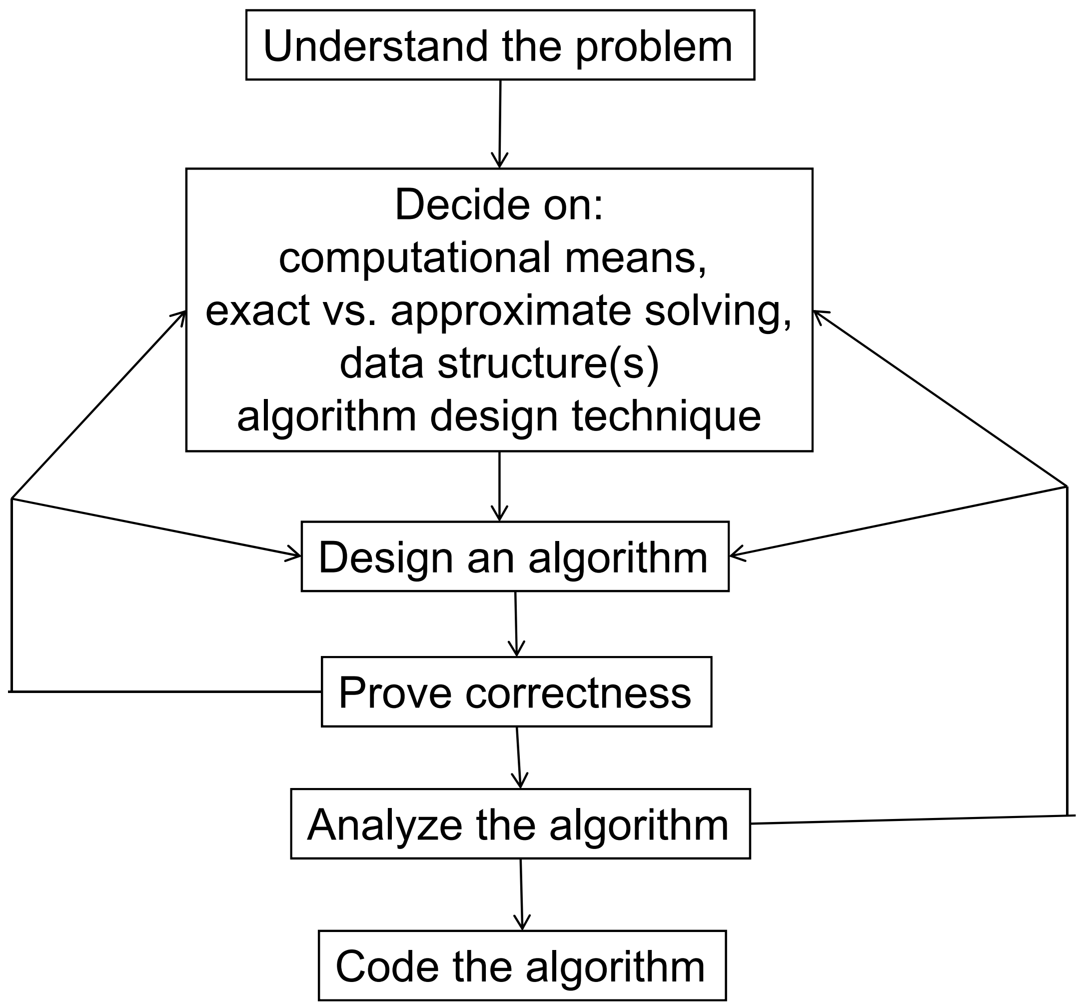
Spring 2017

What is an Algorithm?

- Well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output



Fundamentals of Algorithmic Problem Solving



Analyzing Algorithms

- Use pseudocode to describe algorithms
- Analyzing algorithms
  - Want to predict resources that the algorithm requires
    - <u>Running time (or computational time)</u>
    - Memory
    - Bandwidth
    - Hardware components
    - so on.

Random Access Machine (RAM) Model

Assumptions:

- Instructions executed one after another (e.g. sequential algorithms)
- Primitive instructions take a constant amount of time
  - *Arithmetic*: add, subtract, multiply, divide, remainder, floor, ceiling, shift left, shift right
  - *Data movement*: load, store, copy
  - *Control*: conditional/unconditional branch, subroutine call and return
- Uses integer and floating-point types

reference: CLRS pg 23
CLRS – book by Cormen, Leiserson, Rivest, and Stein

Computing the running time

- Express running time using *asymptotic notations* as a function of the *input size*
- Input size depends on the problem being studied
- Running time on a particular input is the number of primitive operations (steps) executed

  - Examples

When is an algorithm considered "efficient"?

- Platform-independent, instance-independent, and of predictive value with respect to increasing input sizes
- Think about the worst-case RT
- An algorithm is *efficient* if:
  - When implemented, it runs quickly on real input instances
  - Achieves qualitatively better worst-case performance, at analytical level, than brute force
- An algorithm is *efficient* if the worst-case running time is polynomial

Tractable vs intractable problems

- Problems that have worst-case polynomial-time algorithms are called **feasible** or **tractable**
- A problem that does not have a worst case polynomial time algorithm is called **intractable**
- A problem for which there is no algorithm is said to be **unsolvable**
  - Halting problem: given a Turing machine M and an input, will M eventually halt?
- **NP-complete problems**
  - solvable problems that have an undermined status: they are thought to be intractable, but none of them has been proved to be intractable
  - if one NP-complete problem has a polynomial-time algorithm, *all* of them will have polynomial-time algorithms
  - no polynomial-time algorithm discovered so far $\Rightarrow$ believed they are intractable

Asymptotic Notations

O - notation

Ω- notation

Θ- notation

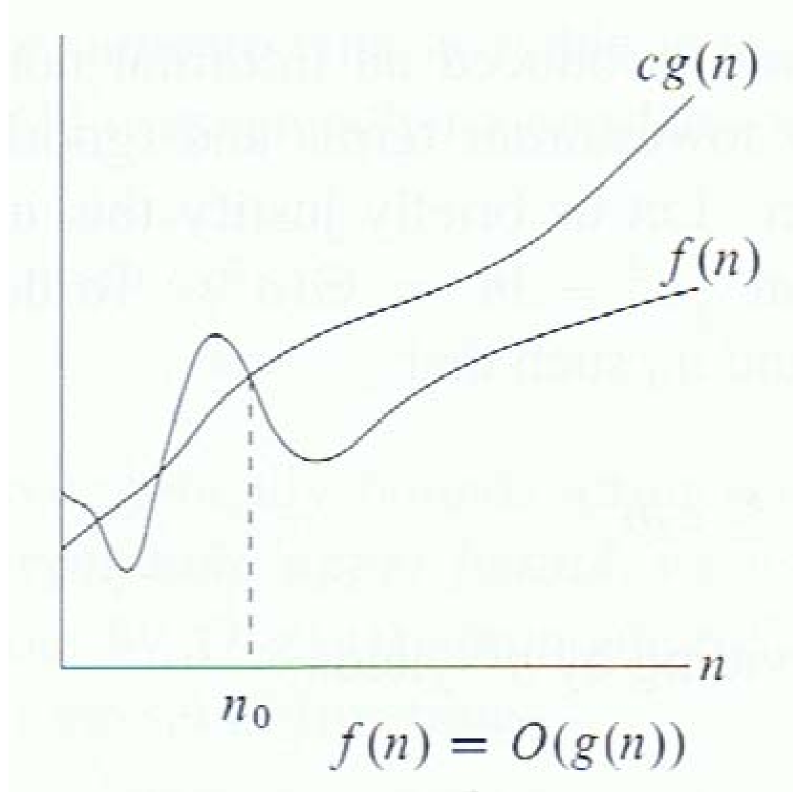
o - notation

ω - notation

Reference: CLRS, chapter 3

O-notation

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 s.t. $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ $\}$



- $g(n)$ is an asymptotic upper bound
- we usually write $f(n) = O(g(n))$

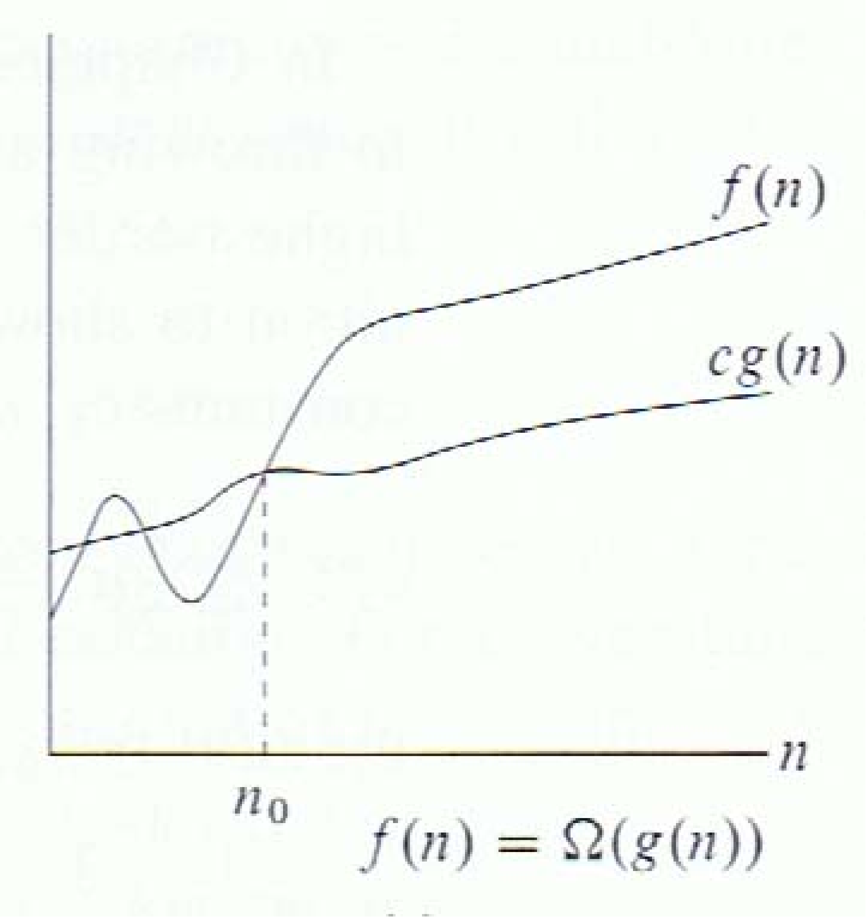
Examples:

$3n^2 + 5n - 100 = O(n^2)$

$3n^2 + 5n - 100 = O(n^4)$

Ω-notation

$\Omega(g(n))$ = {f(n): there exist positive constants c and n_0 s.t. $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$ }



- g(n) is an asymptotic lower bound
- we usually write $f(n) = \Omega(g(n))$

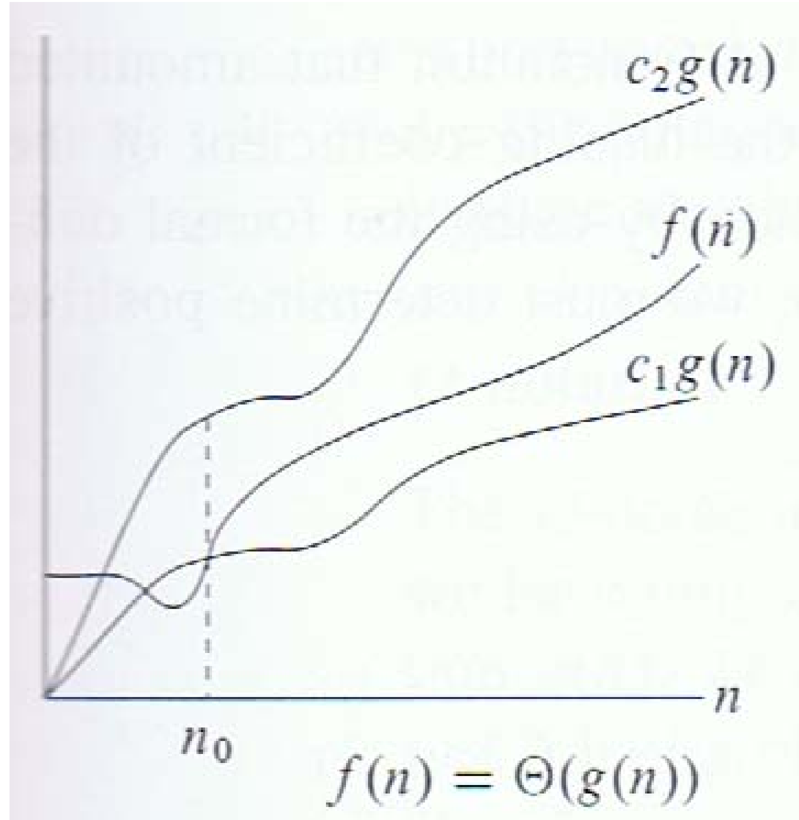
Examples:

$3n^2 + 5n - 100 = \Omega(n^2)$

$3n^2 + 5n - 100 = \Omega(n)$

Θ-notation

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 s.t.
$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$
for all $n \ge n_0$ $\}$



- $g(n)$ is an asymptotic tight bound
- we usually write $f(n) = \Theta(g(n))$

Theorem:

$f(n) = \Theta(g(n)$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Example: $3n^2 + 5n - 100 = \Theta(n^2)$

o-notation

- used to indicate an upper bound that is not asymptotically tight

$o(g(n)) = \{f(n)$: for any positive const $c > 0$, there exists a positive constant n_0 s.t. $0 \leq f(n) < cg(n)$ for all $n \geq n_0\}$

"quick" definition:

$f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Example: $3n^2 - 100 = o(n^3)$

ω-notation

- used to indicate a lower bound that is not asymptotically tight

$\omega(g(n))$ = {f(n): for any positive const c > 0, there exists a positive constant n_0 s.t. $0 \leq cg(n) < f(n)$ for all $n \geq n_0$}

"quick" definition:

$f(n) = \omega(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Example: $3n^2 - 100 = \omega(n)$

Analogy between asymptotic notations and comparison of two real numbers

$f(n) = O(g(n))$	is like	$a \le b$
$f(n) = \Omega(g(n))$	is like	$a \ge b$
$f(n) = \Theta(g(n))$	is like	$a = b$
$f(n) = o(g(n))$	is like	$a < b$
$f(n) = \omega(g(n))$	is like	$a > b$

Example:
asymptotic notations for $3n^2 + 10n - 500$

$3n^2+10n-500 = O(n^3)$

$3n^2+10n-500 = O(n^2)$

$\cancel{3n^2+10n-500 = O(n)}$

$\cancel{3n^2+10n-500 = \Omega(n^3)}$

$3n^2+10n-500 = \Omega(n^2)$

$3n^2+10n-500 = \Omega(n)$

$\cancel{3n^2+10n-500 = \Theta(n^3)}$

$3n^2+10n-500 = \Theta(n^2)$

$\cancel{3n^2+10n-500 = \Theta(n)}$

$3n^2+10n-500 = o(n^3)$

$\cancel{3n^2+10n-500 = o(n^2)}$

$\cancel{3n^2+10n-500 = o(n)}$

$\cancel{3n^2+10n-500 = \omega(n^3)}$

$\cancel{3n^2+10n-500 = \omega(n^2)}$

$3n^2+10n-500 = \omega(n)$

Rates of growth between polylogarithmic, polynomial, and exponential functions

- Any exponential function (base > 1) grows faster than any polynomial function
- Any positive polynomial function grows faster than any polylogarithmic function

Use limits to determine order of growth between functions

Limit value	Asymptotic Notation
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$	f(n) = o(g(n))
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	f(n) = ω(g(n))
$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	f(n) = O(g(n))
$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$	f(n) = Ω(g(n))
$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	f(n) = Θ(g(n))
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = undefined$	cannot use

Common order of growth functions

Theta Form	*Name*
$\Theta(1)$	Constant
$\Theta(\lg \lg n)$	Log log
$\Theta(\lg n)$	Log
$\Theta(n^c),\ 0 < c < 1$	Sublinear
$\Theta(n)$	Linear
$\Theta(n \lg n)$	$n \log n$
$\Theta(n^2)$	Quadratic
$\Theta(n^3)$	Cubic
$\Theta(n^k),\ k \geq 1$	Polynomial
$\Theta(c^n),\ c > 1$	Exponential
$\Theta(n!)$	Factorial

Summations

CLRS Appendix A

- Arithmetic Series

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \theta(n^2)$$

- Sum of Squares

$$\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} = \theta(n^3)$$

- Sum of Cubes

$$\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4} = \theta(n^4)$$

Summations, cont.

- Geometric Series

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + x^3 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

If $|x| < 1$ and $n \to \infty$ then $\boxed{\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}}$ (since $x^{n+1} \to 0$)

Recurrence

- A *recurrence* is an equation or inequality that describes a function in terms of its value on smaller inputs

Examples:

$T(n) = 2T(n/2) + n$ for $n > 1$

$T(1) = \Theta(1)$

$T(n) = T(n-1) + n$ for $n > 0$

$T(0) = 0$

Methods for Solving Recurrence Relations

- No universal method that can be used to solve every recurrence
- Techniques:
  - Method of forward substitutions
  - Method of backward substitutions
  - Master Theorem

Method of Forward Substitutions

- Start from the initial term(s) and use the recurrence equation to generate the first few terms, in the hope of seeing a pattern that can be expressed by a closed-end formula
- If such a formula is found, check its validity:
  - Substitute into the recurrence equation and the initial condition, OR
  - Prove using mathematical induction

Method of Forward Substitutions

Example:

$T(n) = 2T(n-1) + 1 \quad \text{for } n > 1$

$T(1) = 1$

Solution: $T(1) = 1$

$T(2) = 3$

$T(3) = 7$

$T(4) = 15$

Observation: these numbers are one less than consecutive powers of 2

$T(n) = 2^n - 1 \quad \text{for } n \geq 1$

- Check validity

Method of Backward Substitutions

- Using the recurrence relation, express $T(n-1)$ as a function of $T(n-2)$ and substitute into the original equation to get $T(n)$ as a function of $T(n-2)$
- Repeat this step and get $T(n)$ as a function of $T(n-3)$
- So on…. in the hope of seeing a pattern in expressing $T(n)$ as a function of $T(n-i)$, $i = 1, 2, \ldots$
- Selecting i to make $n-i$ reach the initial condition and using one of the standard summation formulas often leads to a closed-end formula

Method of Backward Substitutions

Example:

$$T(n) = T(n-1) + n \quad \text{for } n > 0$$

$$T(0) = 0$$

Solution:

$T(n-1) = T(n-2) + n - 1 \Rightarrow T(n) = T(n-2) + (n-1) + n$

$T(n-2) = T(n-3) + n - 2 \Rightarrow T(n) = T(n-3) + (n-2) + (n-1) + n$

After i substitutions:

$$T(n) = T(n-i) + (n-i+1) + (n-i+2) + \ldots + n$$

Taking i = n, we get:

$$T(n) = T(0) + 1 + 2 + 3 + \ldots + n = n(n+1)/2 \quad \text{(arithmetic series)}$$

Master Theorem (CLRS pg 95)

Let a $\geq$ 1 and b >1 be constants, let f(n) be a function, and let T(n) be defined on nonnegative integers by the recurrence:

T(n) = $aT\left(\frac{n}{b}\right)$ + f(n)

1. If f(n) = $O(n^{\log_b^a - \varepsilon})$ for some const ε > 0, then T(n) = $\Theta(n^{\log_b^a})$
2. If f(n) = $\Theta(n^{\log_b^a})$, then T(n) = $\Theta(n^{\log_b^a} \lg n)$
3. If f(n) = $\Omega(n^{\log_b^a + \varepsilon})$ for some const ε > 0, and if

   af(n/b) $\leq$ cf(n) for some constant c < 1 and all sufficiently large n, then

   T(n) = $\Theta(f(n))$

- Examples