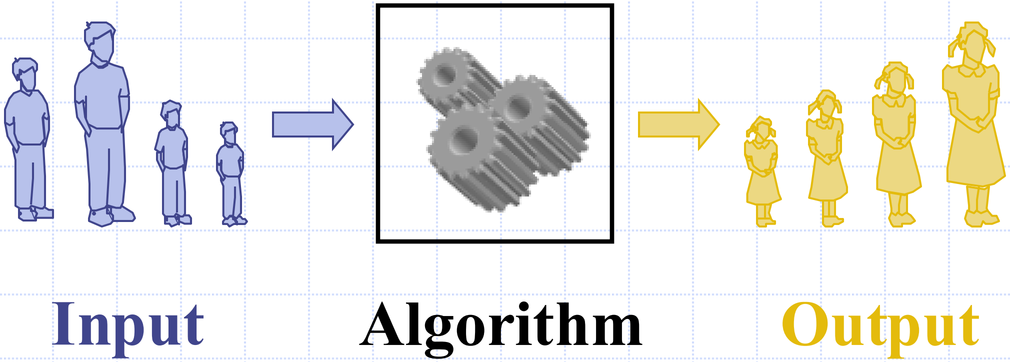
Analysis of Algorithms



An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.

What We Want

- A method for measuring, roughly, the cost (or speed) or executing an algorithm
- Several potential methods
  - Obvious one is measuring an implementation
  - Another is counting instructions

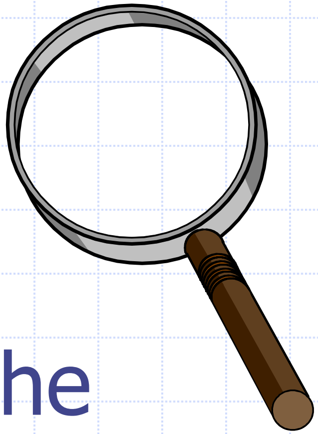
Implementation

- Who implements it
  - And how do they do this?
- With what compiler? Using what settings?
- On whose hardware?
- Using whose clock?
- On what OS
  - Are other programs running? If not is this realistic?
- On what data sets?
  - Small number of data sets may not give good representation
  - Lots of data sets may not be practical

Counting Instructions

- On what machine?
  - RISC? CISC? Special Instructions (such as Intel MMX)?
  - With what compiler?
    - Is it optimized for applications
  - On what data?
    - Again large data sets versus few data sets.

Theoretical Analysis



- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, n.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Primitive Operations

- Roughly, a primitive operation is one that can be performed in a constant number of steps that ***does not depend on the size of the input***
  - Ex. Assume CPU can access arbitrary memory location in a single primitive operation

Primitive Operations

- Assignment of variable
  - If assigning to array, two primitive ops (index into array, then write value)
- Comparing two numbers
- Basic algebraic operations
  - Addition, subtraction, multiplication, division
- This list is not exhaustive

Big-Oh Notation (§ 1.2)

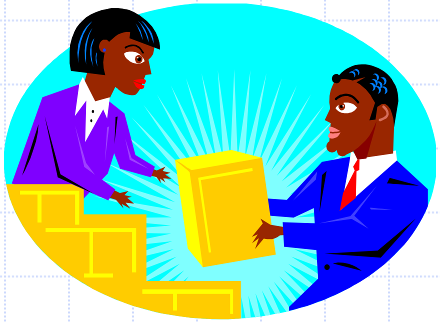
- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for $n \geq n_0$
- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$

Big-Oh Example

- Example: the function n^2 is not $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - The above inequality cannot be satisfied since c must be a constant

More Big-Oh Examples



- $7n-2$

  $7n-2$ is $O(n)$

  need $c > 0$ and $n_0 \geq 1$ such that $7n-2 \leq c \bullet n$ for $n \geq n_0$

  this is true for $c = 7$ and $n_0 = 1$

- $3n^3 + 20n^2 + 5$

  $3n^3 + 20n^2 + 5$ is $O(n^3)$

  need $c > 0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ for $n \geq n_0$

  this is true for $c = 4$ and $n_0 = 21$

- $3 \log n + \log \log n$

  $3 \log n + \log \log n$ is $O(\log n)$

  need $c > 0$ and $n_0 \geq 1$ such that $3 \log n + \log \log n \leq c \bullet \log n$ for $n \geq n_0$

  this is true for $c = 4$ and $n_0 = 2$

Big-Oh Rules

- If is $f(n)$ a polynomial of degree d, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant factors
- Use the smallest possible class of functions
  - Say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- Use the simplest expression of the class
  - Say "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$"

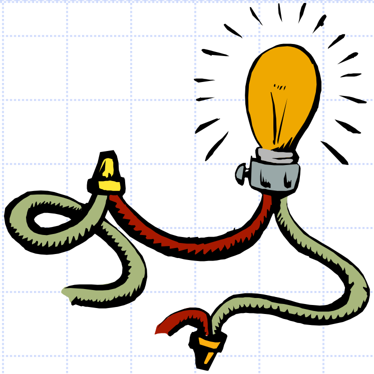
Asymptotic Algorithm Analysis

- The asymptotic analysis of an algorithm determines the running time in big-Oh notation
- To perform the asymptotic analysis
  - We find the ***worst-case*** number of primitive operations executed as a function of the input size
    - Note we sometimes discuss average time
  - We express this function with big-Oh notation
- Since constant factors and lower-order terms are eventually dropped anyhow, we can disregard them when counting primitive operations

Relatives of Big-Oh

- **big-Omega**
  - $f(n)$ is $\Omega(g(n))$ if there is a constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$
- **big-Theta**
  - $f(n)$ is $\Theta(g(n))$ if there are constants $c' > 0$ and $c'' > 0$ and an integer constant $n_0 \geq 1$ such that $c' \bullet g(n) \leq f(n) \leq c'' \bullet g(n)$ for $n \geq n_0$
- **little-oh**
  - $f(n)$ is $o(g(n))$ if, for any constant $c > 0$, there is an integer constant $n_0 \geq 0$ such that $f(n) \leq c \bullet g(n)$ for $n \geq n_0$
- **little-omega**
  - $f(n)$ is $\omega(g(n))$ if, for any constant $c > 0$, there is an integer constant $n_0 \geq 0$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$

Intuition for Asymptotic Notation



Big-Oh

- f(n) is O(g(n)) if f(n) is asymptotically **less than or equal** to g(n)

big-Omega

- f(n) is Ω(g(n)) if f(n) is asymptotically **greater than or equal** to g(n)

big-Theta

- f(n) is Θ(g(n)) if f(n) is asymptotically **equal** to g(n)

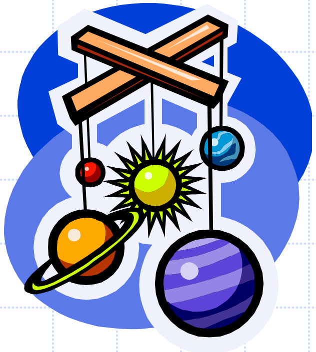
little-oh

- f(n) is o(g(n)) if f(n) is asymptotically **strictly less** than g(n)

little-omega

- f(n) is ω(g(n)) if is asymptotically **strictly greater** than g(n)

Example Uses of the Relatives of Big-Oh



- **$5n^2$ is $\Omega(n^2)$**

  f(n) is $\Omega(g(n))$ if there is a constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$

  let $c = 5$ and $n_0 = 1$

- **$5n^2$ is $\Omega(n)$**

  f(n) is $\Omega(g(n))$ if there is a constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$

  let $c = 1$ and $n_0 = 1$

- **$5n^2$ is $\omega(n)$**

  f(n) is $\omega(g(n))$ if, for any constant $c > 0$, there is an integer constant $n_0 \geq 0$ such that $f(n) \geq c \bullet g(n)$ for $n \geq n_0$

  need $5n_0^2 \geq c \bullet n_0 \rightarrow$ given c, the n_0 that satifies this is $n_0 \geq c/5 \geq 0$

More Rules

- Know which functions get butts kicked by which functions (and why!)
  - I.e, look at Theorem 1.7 on p. 15 in text
- Be sure to look at Section 1.2.3 (carefully)
  - In particular, you should realize why the constants in the definition of Big-O notations are not really a factor!