

CS 538: Artificial Intelligence

Fall 2007

Tutorial 1: Algorithmic Complexity
and Analysis of Algorithms
09/06/2007

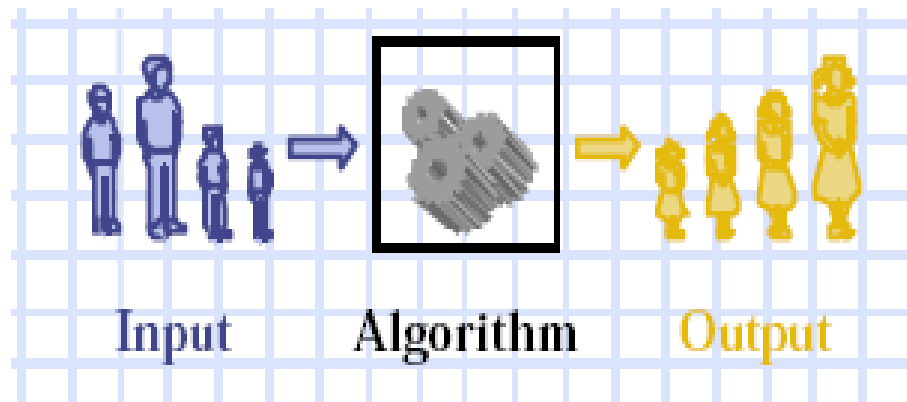
Derek Harter - Texas A&M University - Commerce

Resources

- Wikipedia:
http://en.wikipedia.org/wiki/Analysis_of_algorithms
- Algorithm Design: Foundations, Analysis, and Internet Examples by Michael T. Goodrich and Roberto Tamassia
<http://ww3.algorithmdesign.net/>

Algorithm

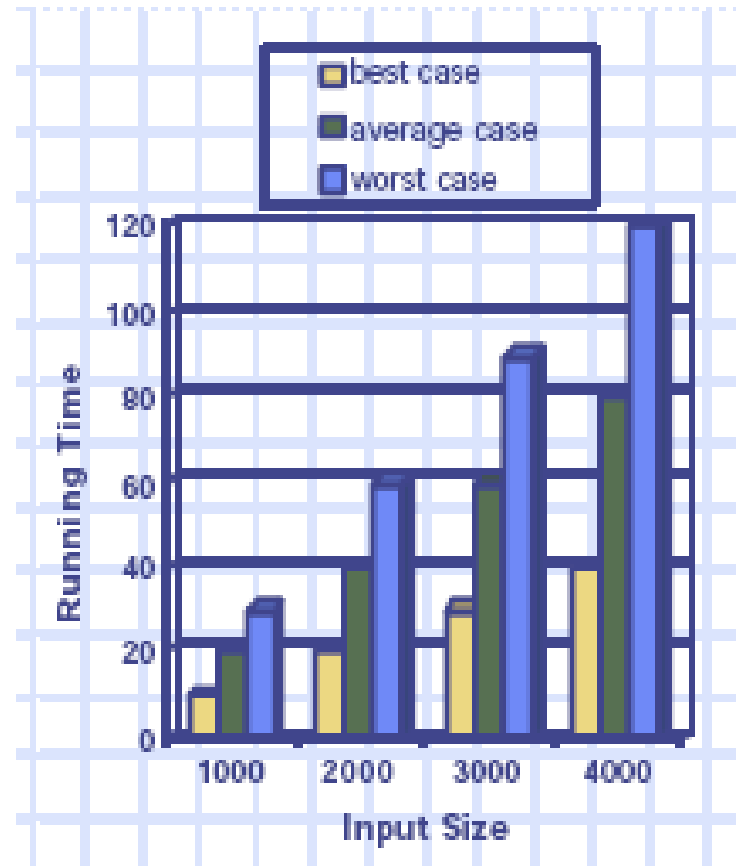
- What the heck is an algorithm anyway?



- An Algorithm is a step-by-step procedure for solving a problem in a finite amount of time.
 - For all practical purposes, you can think of it as a kind of (very formal and overly specific) recipe for a machine.

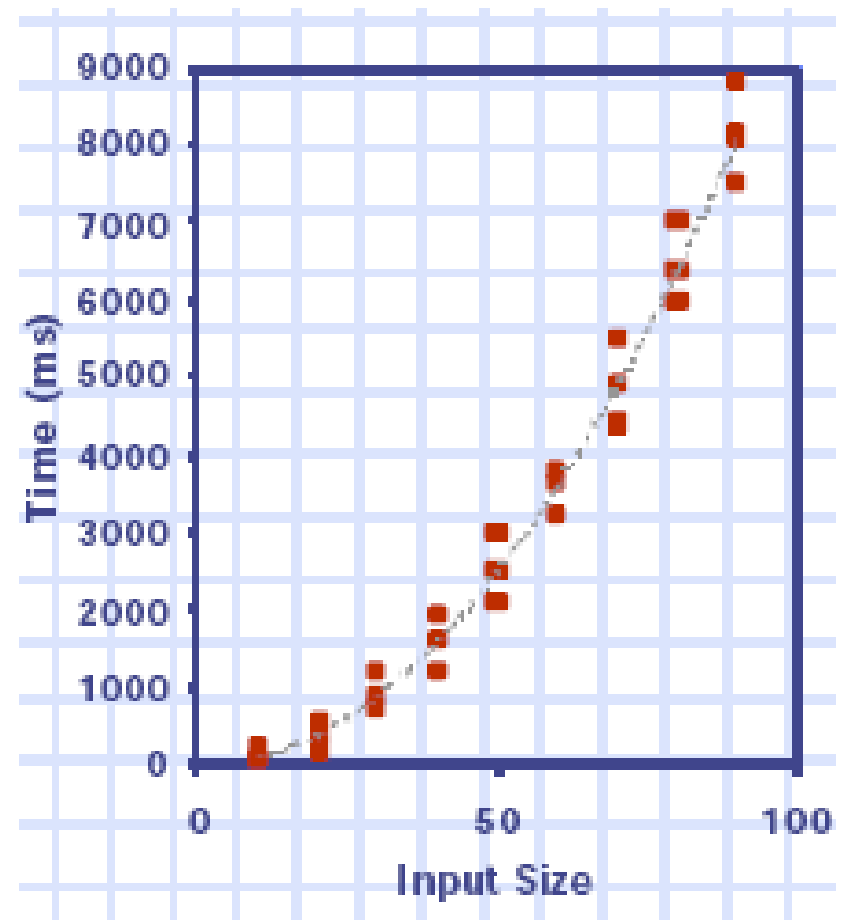
Running Time

- Most Algorithms transform input objects into output objects.
- The running time of an algorithm typically grows with the input size.
- Average case time is often difficult to determine.
- We focus on the worst-case running time.
 - Easier to analyze
 - Crucial to applications such as games, finance and robotics.



Experimental Studies

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a method like `System.currentTimeMillis()` to get an accurate measure of the actual running time
- Plot the results



Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used.

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

Pseudocode

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides program design issues

function SUMMATION(*sequence*) returns a number

Input *sequence* a list of numbers

Output *sum* of numbers in sequence

sum ← 0

for *i* ← 1 **to** LENGTH(*sequence*) **do**

sum ← *sum* + *sequence*[*i*]

return *sum*

Example: Find sum of a list of numbers

Pseudocode Details

- **Control flow**
 - if ... then ... [else ...]
 - while ... do ...
 - repeat ... until ...
 - for ... do ...
 - Indentation replaces braces
- **Function/Method declaration**

function METHOD (arg [, arg...])
 Input ...
 Output ...
- **Function/Method call**
 - FUNCTION (arg [, arg...])
- **Return value**
 - return expression
- **Expressions**
 - ← Assignment
 - = Equality testing
 - n² Superscripts and other mathematical formatting allowed

Primitive Operations

- Basic computations performed by an algorithm
 - Identifiable in pseudocode
 - Largely independent from the programming language
 - Exact definition not important (we will see why later)
 - Assumed to take a constant amount of time in our examples
- Examples:
 - Evaluating an expression
 - Assigning a value to a variable
 - Indexing into an array
 - Calling a method
 - Returning from a method

Counting Primitive Operations

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size

function SUMMATION(<i>sequence</i>) returns a number	# Operations
<i>sum</i> ← 0	1
for <i>i</i> ← 1 to LENGTH(<i>sequence</i>) do	<i>n</i>
<i>sum</i> ← <i>sum</i> + <i>sequence</i> [<i>i</i>]	3 <i>n</i>
return <i>sum</i>	1
	Total: 4 <i>n</i> + 2

Estimating Running Time

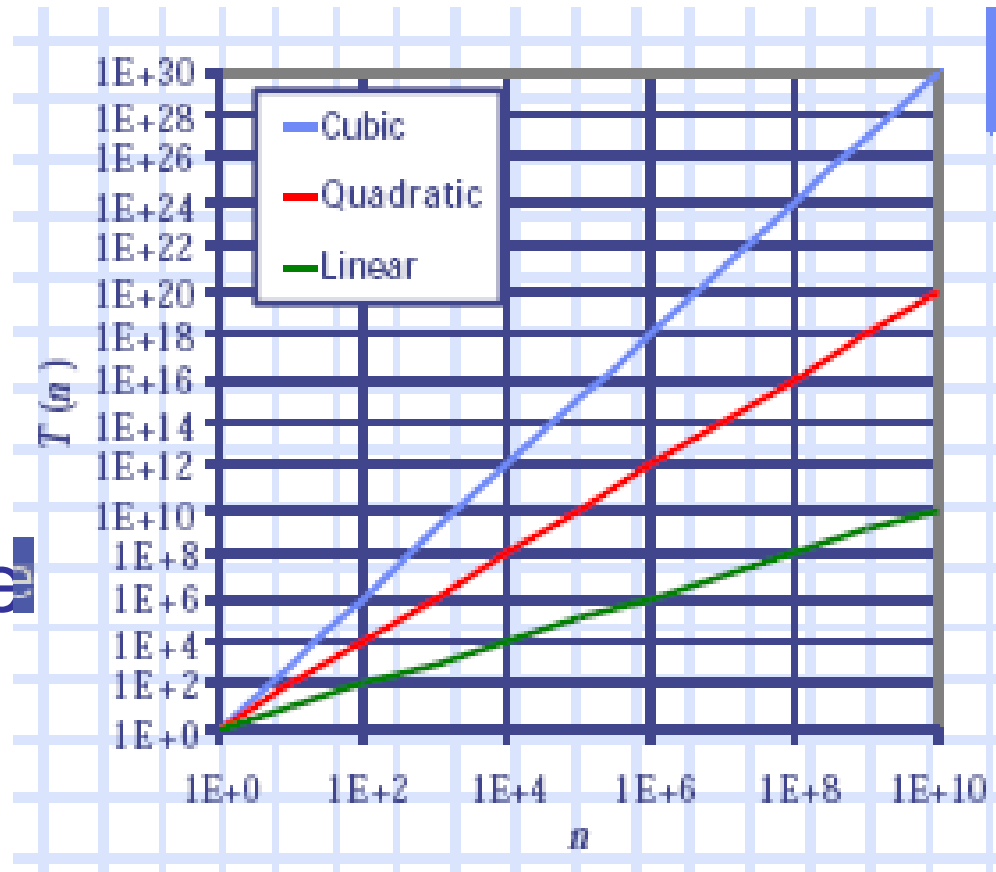
- Algorithm SUMMATION executes $4n + 2$ primitive operations in the worst case.
Define:
 - a = Time taken by the fastest primitive operation
 - b = Time taken by the slowest primitive operation
- Let $T(n)$ be worst-case time of arrayMax.
Then
$$a(4n + 2) < T(n) < b(4n + 2)$$
- Hence, the running time $T(n)$ is bounded by two linear functions

Growth Rate of Running Time

- Changing the hardware/ software environment
 - Affects $T(n)$ by a constant factor, but
 - Does not alter the growth rate of $T(n)$
 - The linear growth rate of the running time $T(n)$ is an intrinsic property of algorithm
- SUMMATION

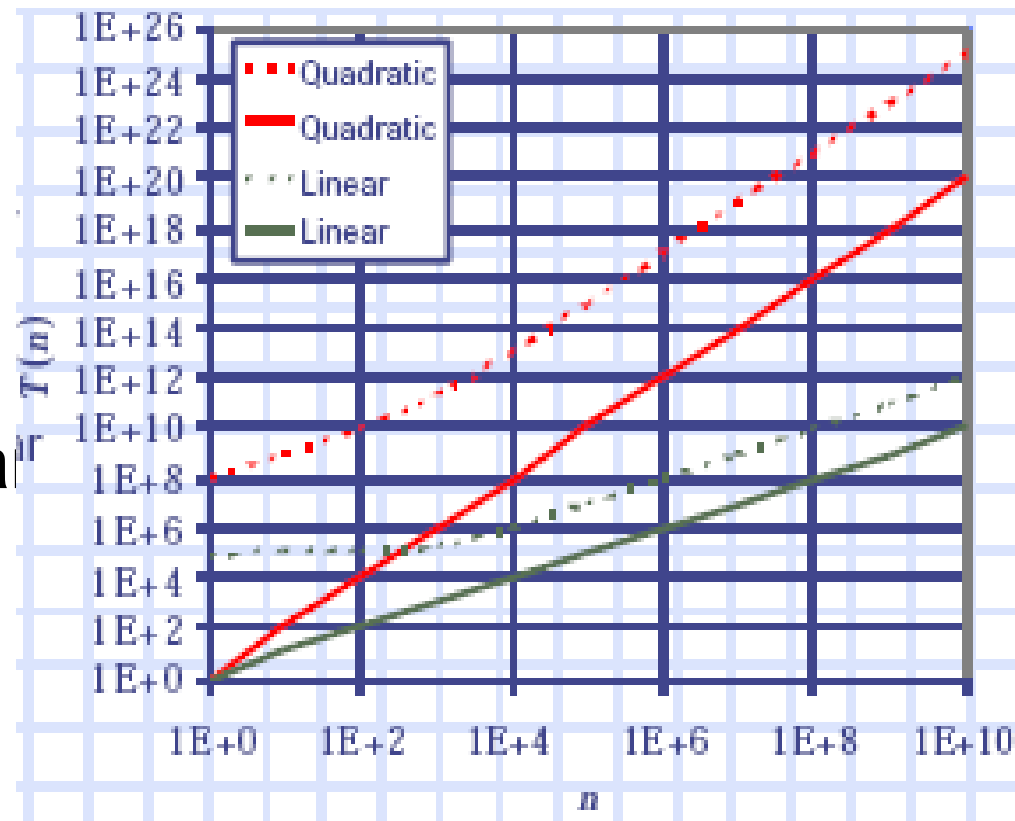
Growth Rates

- Growth rates of functions:
 - Linear = n
 - Quadratic = n^2
 - Cubic = n^3
- In a log-log chart, the slope of the line corresponds to the growth rate of the function



Constant Factors

- The growth rate is not affected by
 - constant factors or
 - lower-order terms
- Examples
 - $10^2n + 10^5$ is a linear function
 - $10^5n^2 + 10^8n$ is a quadratic function

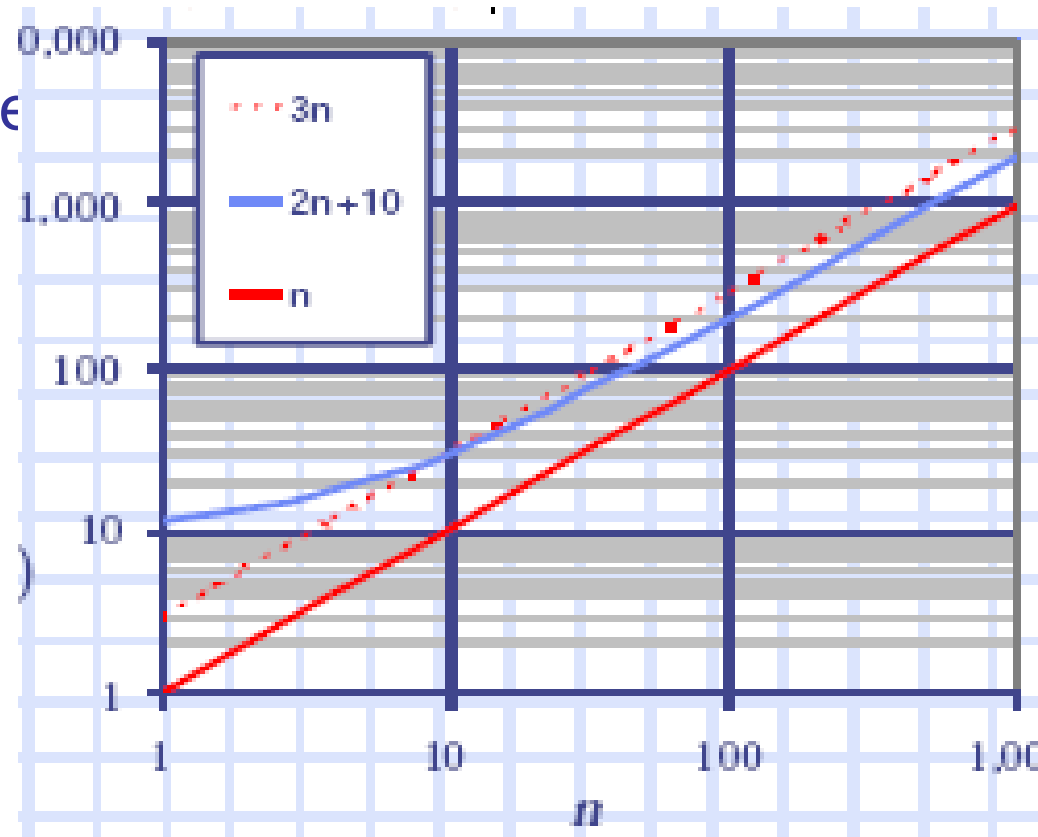


Big-Oh Notation

- 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

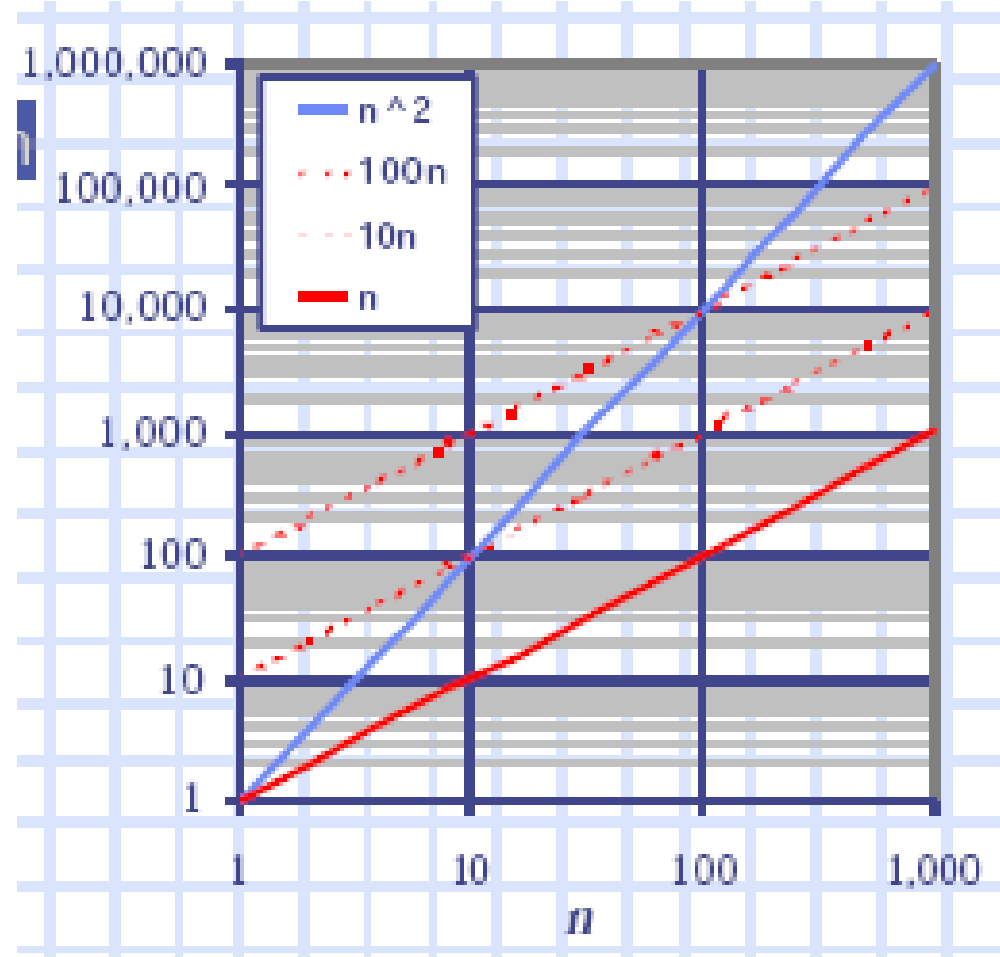
$$f(n) \leq cg(n) \text{ for } n \geq n_0$$

- 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 $7n-2 \leq c \cdot n$ 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 $3n^3 + 20n^2 + 5 \leq c \cdot n^3$ 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 $3 \log n + \log \log n \leq c \cdot \log n$ for $n \geq n_0$

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

Big-Oh Rules

- If $f(n)$ is a polynomial of degree d , then $f(n)$ is $O(n^d)$, i.e.,
 - Drop lower-order terms
 - 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 $O(3n)$ ”