Analysis of Algorithms

Prof. Dianne Foreback

Information on slides originating from *Algorithm Design & Applications* by Michael T. Goodrich and Roberto Tamassia, © 2015 John Wiley & Sons, Inc.Goodrich and Tamassia, ISBN: 978-1118335918.

Reading Material

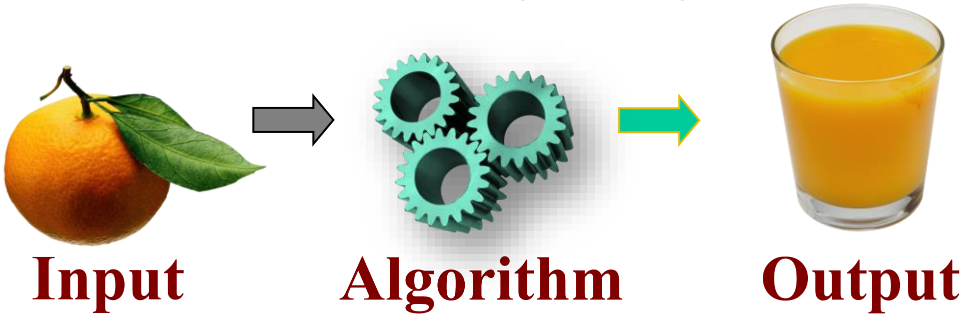
- ***Algorithm Design & Applications*** **by Michael T. Goodrich and Roberto Tamassia**
  - **Chapter 1 Introduction to Section 1.2.2 inclusively**
  - **Appendix A**

Algorithm Definition and Running Time Topics

- Definition of Algorithm
  - Algorithm as abstract entity (math entities)
  - Pseudocode Example (Algo 1.2 arrayMax)
- Correctness and Efficiency of Algorithms
  - Correctness
  - Efficiency (metrics considering in this course)
    - Running Time is primary measure
    - Space measuring
- Primitive Operations
  - What are primitive operations
  - Run time calculations via primitive operations and counting
- Asymptotic Analysis

Algorithm & Data Structure Definition

- **algorithm** is a step-by-step procedure for performing some task in a finite amount of time
  - typically takes input, executes the procedure and yields the solution as output
- **data structure** is a systematic way of organizing and accessing data



Algorithms as Abstract Entities

- considering algorithms abstractly, as math entities
- not tied to a particular language or operating system
- will use pseudocode to represent

Pseudocode Example

Algorithm arrayMax(A, n):

Input: An array A storing $n \geq 1$ integers.

Output: The maximum element in A.

$currentMax \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n - 1$ **do**

  if $currentMax < A[i]$ **then**

    $currentMax \leftarrow A[i]$

return $currentMax$

Algorithm 1.2: Algorithm arrayMax

Pseudocode Details

- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces

- Method declaration

  Algorithm ***method*** (***arg*** [, ***arg***…])

  Input …

  Output …

- Method call

  method (***arg*** [, ***arg***…])

- Return value

  return ***expression***

- Expressions:

  $\leftarrow$ Assignment

  $=$ Equality testing

  n^2 Superscripts and other mathematical formatting allowed

Algorithm Definition and Running Time Topics

- Definition of Algorithm
  - Algorithm as abstract entity (math entities)
  - Pseudocode Example (Algo 1.2 arrayMax)
- Correctness and Efficiency of Algorithms
  - Correctness
  - Efficiency (metrics considering in this course)
    - Running Time is primary measure
    - Space measuring
- Primitive Operations
  - What are primitive operations
  - Run time calculations via primitive operations and counting
- Asymptotic Analysis

Correctness of Algorithms

- correctness – the algorithm must solve the problem designed to solve
  - must be able to reason about the algorithm and explain that it is correct
    - consider precondition, post condition, and corner cases
- efficiency metrics considerations for our class
  - running time (or run time)
  - space – consider how much memory does the algorithm use

Algorithm Definition and Running Time Topics

- Definition of Algorithm
  - Algorithm as abstract entity (math entities)
  - Pseudocode Example (Algo 1.2 arrayMax)
- Correctness and Efficiency of Algorithms
  - Correctness
  - Efficiency (metrics considering in this course)
    - Running Time is primary measure
    - Space measuring
- Primitive Operations
  - What are primitive operations
  - Run time calculations via primitive operations and counting
- Asymptotic Analysis

Primitive Operation

- Assignment
- Method invocation
- Arithmetic operation
- Indexing into an array
- Comparison
- Dereferencing reference pointer
- Return from a method

Calculating Run Time

- Count total number of primitive operations executed in pseudocode
- Convert total to Big-Oh notation

Calculate Run Time Part 1

Algorithm arrayMax(A, n):

Input: An array A storing $n \geq 1$ integers.

Output: The maximum element in A.

$currentMax \leftarrow A[0]$	2 primitive operations
for $i \leftarrow 1$ **to** $n - 1$ **do**	+4 primitive operations
if $currentMax < A[i]$ **then**	+2 primitive operations
$currentMax \leftarrow A[i]$	+2 primitive operations
return $currentMax$	+1 primitive operations

executed multiple times

Algorithm 1.2: Algorithm arrayMax

for i←1 to n-1: assignment of 1 to i (=1 op) , compare i < n-1 (=1 op), hidden in for loop is i = i+1 (= 2op, math op + assignment op) is 4 total primitive operations

Calculate Run Time Part 2 – Worst Case

Algorithm arrayMax(A, n):

Input: An array A storing $n \geq 1$ integers.

Output: The maximum element in A.

$currentMax \leftarrow A[0]$ — 2 primitive operations

for $i \leftarrow 1$ **to** $n-1$ **do** — +4 primitive operations

  if $currentMax < A[i]$ **then** — +2 primitive operations

    $currentMax \leftarrow A[i]$ — +2 primitive operations

return $currentMax$ — +1 primitive operations

executed multiple times

Algorithm 1.2: Algorithm arrayMax

for i←1 to n-1: assignment of 1 to i (=1 op) , compare i < n-1 (=1 op), hidden in for loop is i = i+1 (= 2op, math op + assignment op) is 4 total primitive operations

total worst case run time for algorithm is

2 + 1 + n + 4(n-1) + 1 = 3 + n + 4n – 4 = 5n-1

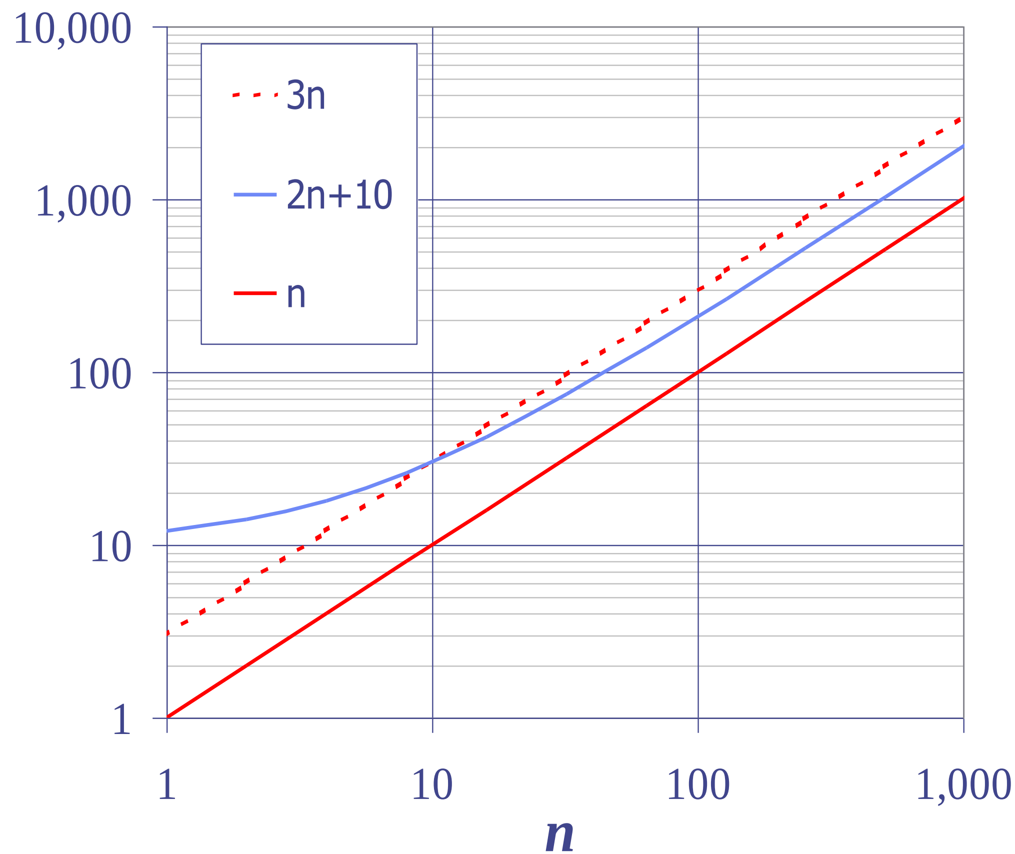
Algorithm Definition and Running Time Topics

- Definition of Algorithm
  - Algorithm as abstract entity (math entities)
  - Pseudocode Example (Algo 1.2 arrayMax)
- Correctness and Efficiency of Algorithms
  - Correctness
  - Efficiency (metrics considering in this course)
    - Running Time is primary measure
    - Space measuring
- Primitive Operations
  - What are primitive operations
  - Run time calculations via primitive operations and counting
- Asymptotic Analysis

Asymptotic Analysis: 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

  $f(n) \leq cg(n)$ 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 $n_0 = 10$



Why use Asymptotic Analysis: Big-Oh

- Want to find a formula for the runtime of an algorithm, $T(n)$, as a function of its input size n
- Gives us an approximation of the run time of an algorithm $T(n)$ based on its input size n
- Easier to reason about runtime of an algorithm with Asymptotic Analysis
- Want the algorithm to be tractable, i.e. solvable in polynomial time (so that it can run on a computer and finish in a "desired" amount of time

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

$$f(n) \geq c\, g(n) \text{ for } n \geq n_0$$

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

$$c'g(n) \leq f(n) \leq c''g(n) \text{ for } n \geq n_0$$

Notation for Relative Rates of Growth

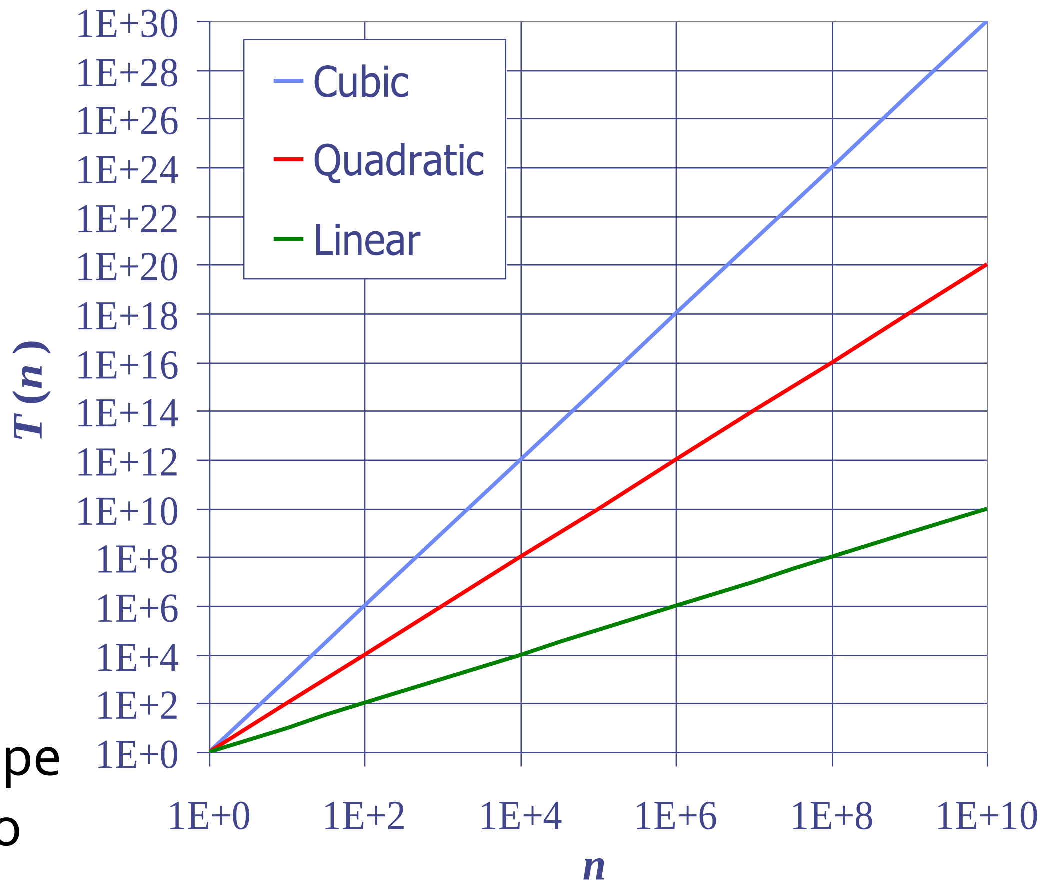
- Notation to describe the runtime performance or space requirements of an algorithm
  - O(n) or “Big-Oh” – upper bound
  - $\Omega(n)$ or “Big-Omega” – lower bound
  - $\Theta(n)$ or “Big-Theta” – tight bound
  - o(n) or “little-oh”
  - $\omega(n)$ or “little-omega”

Big-Oh Cheat Sheet

	Intuitive Meaning	$\lim_{n\to\infty} f(n) / g(n)$
$f(n)=\Theta(g(n))$	$f(n)$ "=" $g(n)$	$0 < c < \infty$
$f(n)=o(g(n))$	$f(n)$ "<" $g(n)$	0
$f(n)=\omega(g(n))$	$f(n)$ ">" $g(n)$	∞
$f(n)=O(g(n))$	$f(n)$ "≤" $g(n)$	$0 \leq c < \infty$
$f(n)=\Omega(g(n))$	$f(n)$ "≥" $g(n)$	$0 < c \leq \infty$

Seven Important Functions

- Seven functions that often appear in algorithm analysis:
  - Constant ≈ $\boldsymbol{1}$
  - Logarithmic ≈ log $\boldsymbol{n}$
  - Linear ≈ $\boldsymbol{n}$
  - N-Log-N ≈ $\boldsymbol{n}$ log $\boldsymbol{n}$
  - Quadratic ≈ $\boldsymbol{n}^2$
  - Cubic ≈ $\boldsymbol{n}^3$
  - Exponential ≈ $\mathbf{2}^n$
- In a log-log chart, the slope of the line corresponds to the growth rate



Math you need to Review

- Summations
- Powers
- Logarithms
- Proof techniques
- Basic probability

- Properties of powers:

  $a^{(b+c)} = a^b a^c$
  $a^{bc} = (a^b)^c$
  $a^b / a^c = a^{(b-c)}$
  $b = a^{\log_a b}$
  $b^c = a^{c*\log_a b}$

- Properties of logarithms:

  $\log_b(xy) = \log_b x + \log_b y$
  $\log_b (x/y) = \log_b x - \log_b y$
  $\log_b xa = a \log_b x$
  $\log_b a = \log_x a / \log_x b$

See Appendix A for more math and formulas to review

Thank You !

Questions ?