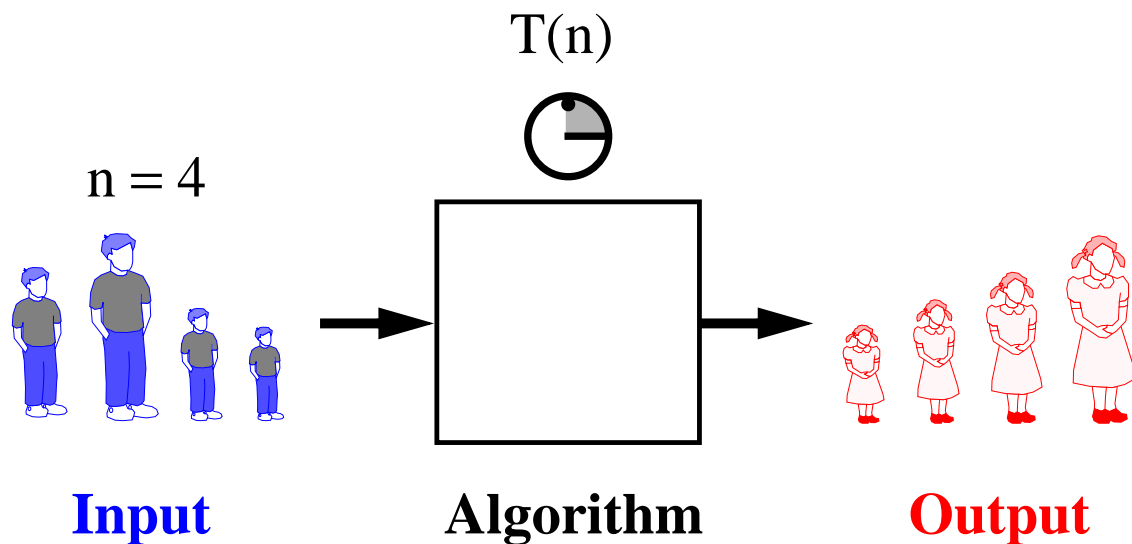


ANALYSIS OF ALGORITHMS

- Quick Mathematical Review
- Running Time
- Pseudo-Code
- Analysis of Algorithms
- Asymptotic Notation
- Asymptotic Analysis



A Quick Math Review

- Logarithms and Exponents
 - properties of **logarithms**:

$$\log_b(xy) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b x^\alpha = \alpha \log_b x$$

$$\log_b a = \frac{\log_a x}{\log_a b}$$

- properties of **exponentials**:

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b / a^c = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c \cdot \log_a b}$$

A Quick Math Review (cont.)

- Floor

$\lfloor x \rfloor =$ the largest integer $\leq x$

- Ceiling

$\lceil x \rceil =$ the smallest integer $\geq x$

- **Summations**

- general definition:

$$\sum_{i=s}^t f(i) = f(s) + f(s+1) + f(s+2) + \dots + f(t)$$

- where f is a function, s is the start index, and t is the end index

- **Geometric progression:** $f(i) = a^i$

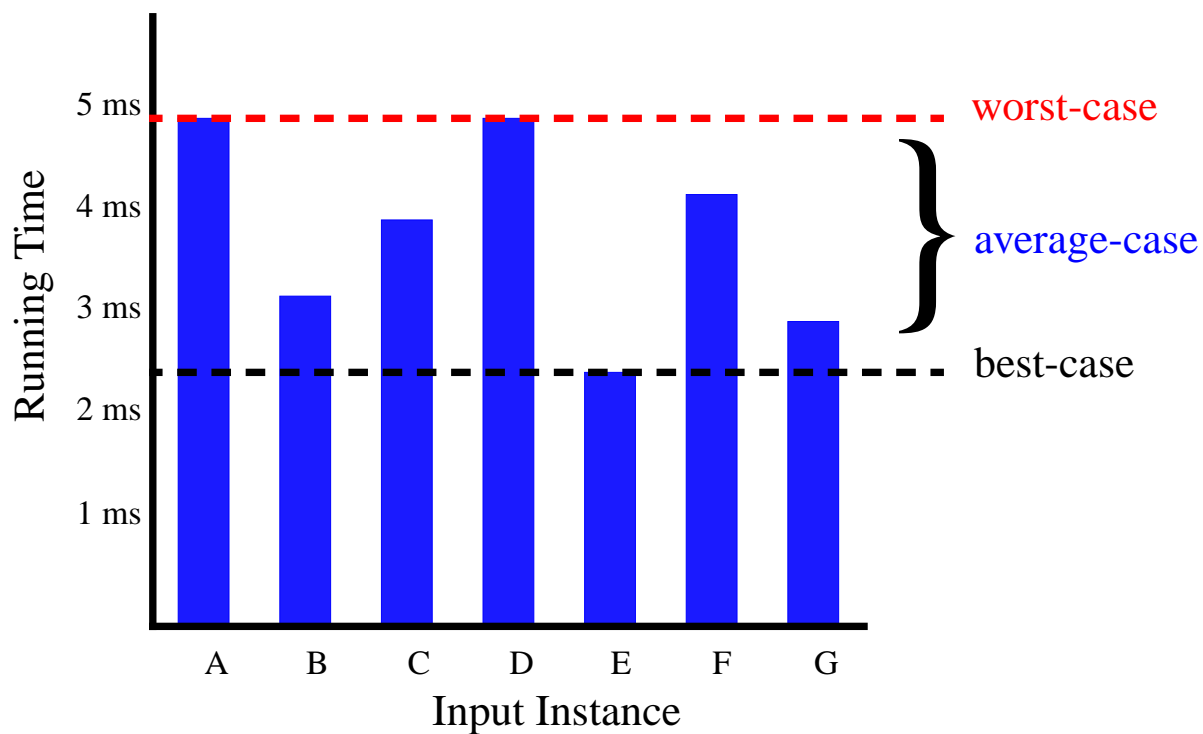
- given an integer $n \geq 0$ and a real number $0 < a \neq 1$

$$\sum_{i=0}^n a^i = 1 + a + a^2 + \dots + a^n = \frac{1 - a^{n+1}}{1 - a}$$

- geometric progressions exhibit exponential growth

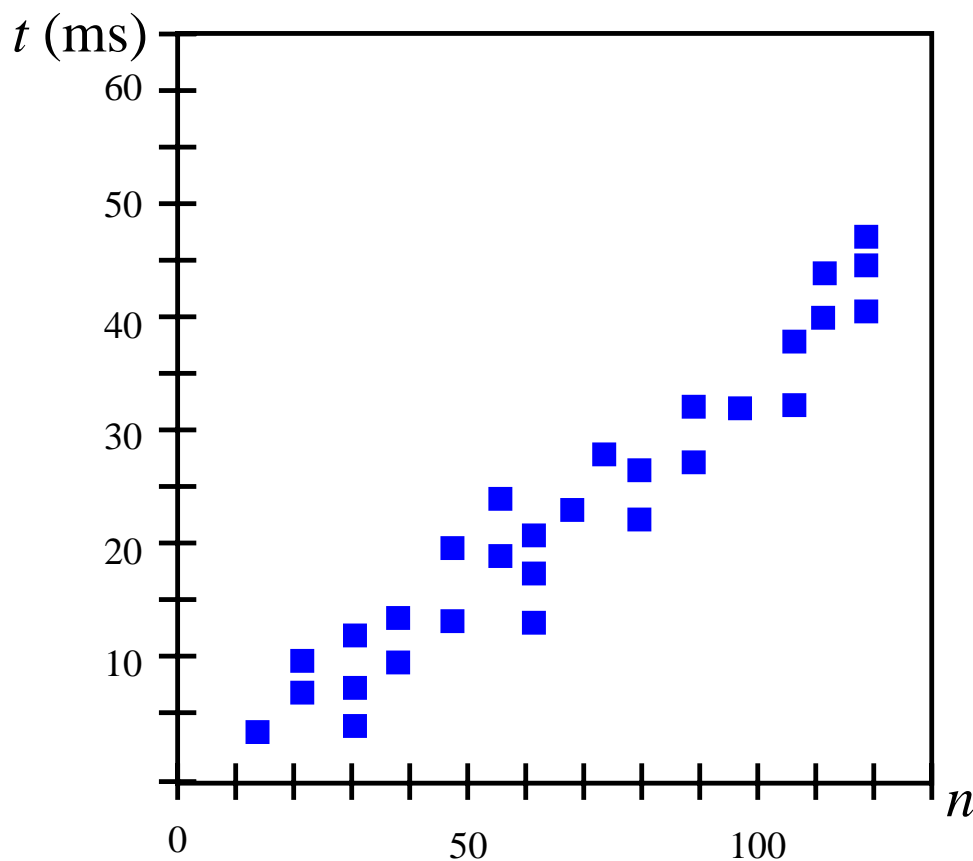
Average Case vs. Worst Case Running Time of an Algorithm

- An algorithm may run faster on certain data sets than on others,
- Finding the **average case** can be very difficult, so typically algorithms are measured by the **worst-case** time complexity.
- Also, in certain application domains (e.g., air traffic control, surgery) knowing the **worst-case** time complexity is of crucial importance.



Measuring the Running Time

- How should we measure the running time of an **algorithm**?
- Experimental Study
 - Write a **program** that implements the algorithm
 - Run the program with data sets of varying size and composition.
 - Use a method like **System.currentTimeMillis()** to get an accurate measure of the actual running time.
 - The resulting data set should look something like:



Beyond Experimental Studies

- Experimental studies have several limitations:
 - It is necessary to **implement** and test the algorithm in order to determine its running time.
 - Experiments can be done only on a **limited set of inputs**, and 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** should be used.
- We will now develop a **general methodology** for analyzing the running time of algorithms that
 - Uses a **high-level description** of the algorithm instead of testing one of its implementations.
 - Takes into account **all possible inputs**.
 - Allows one to evaluate the efficiency of any algorithm in a way that is **independent from the hardware and software environment**.

Pseudo-Code

- Pseudo-code is a description of an algorithm that is more structured than usual prose but less formal than a programming language.
- Example: finding the maximum element of an array.

Algorithm arrayMax(A, n):

Input: An array A storing n 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 $currentMax$

- Pseudo-code is our preferred notation for describing algorithms.
- However, pseudo-code hides program design issues.

What is Pseudo-Code?

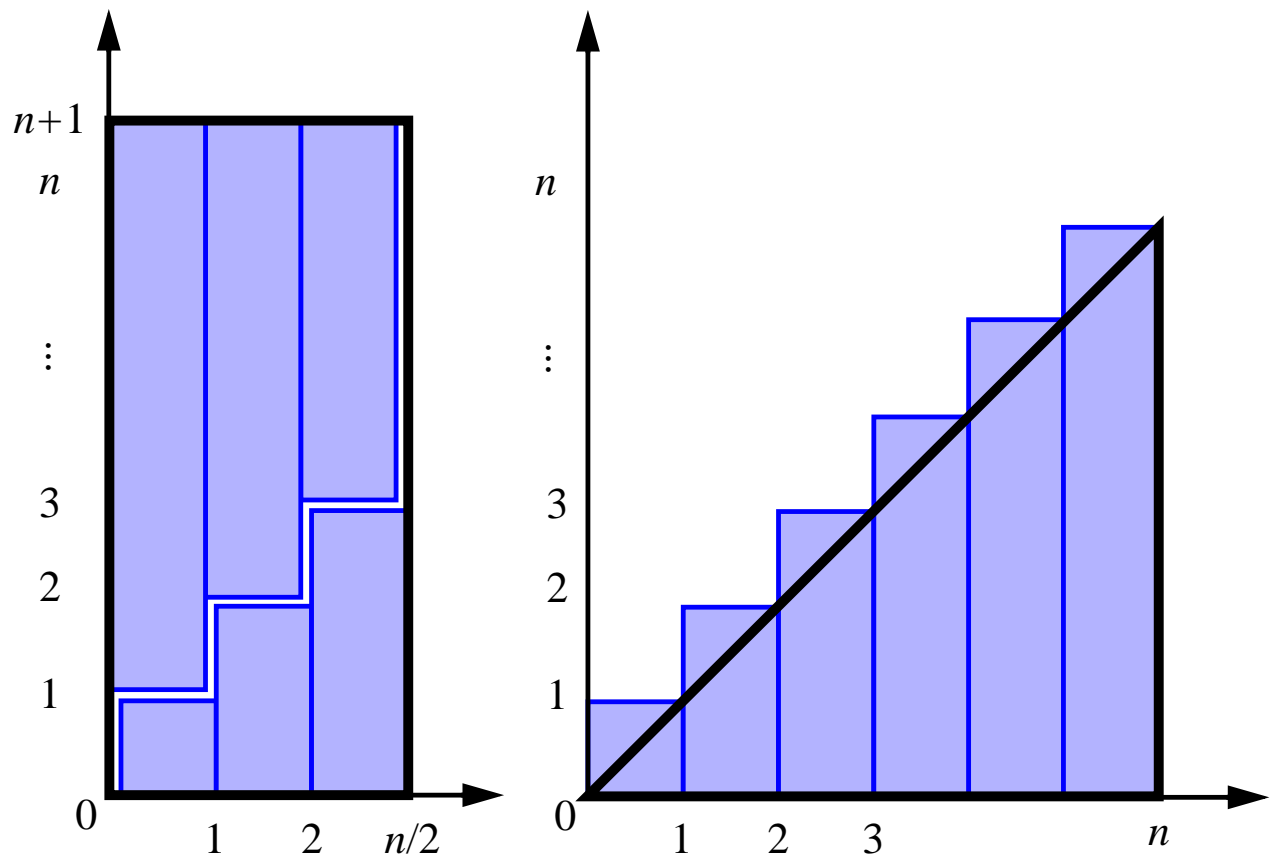
- A mixture of natural language and high-level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm.
 - Expressions: use standard mathematical symbols to describe numeric and boolean expressions
 - use \leftarrow for assignment (“=” in Java)
 - use = for the equality relationship (“==” in Java)
 - Method Declarations:
 - **Algorithm** name(*param1*, *param2*)
 - Programming Constructs:
 - decision structures: **if ... then ... [else ...]**
 - while-loops: **while ... do**
 - repeat-loops: **repeat ... until ...**
 - for-loop: **for ... do**
 - array indexing: **A[i]**
 - Methods:
 - calls: **object method(args)**
 - returns: **return value**

A Quick Math Review (cont.)

- Arithmetic progressions:
 - An example

$$\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n = \frac{n^2 + n}{2}$$

- two visual representations



Analysis of Algorithms

- **Primitive Operations:** Low-level computations that are largely independent from the programming language and can be identified in pseudocode, e.g:
 - calling a method and returning from a method
 - performing an arithmetic operation (e.g. addition)
 - comparing two numbers, etc.
- By inspecting the pseudo-code, we can **count** the number of primitive operations executed by an algorithm.
- Example:

Algorithm arrayMax(A, n):

Input: An array A storing n integers.

Output: The maximum element in A .

$currentMax \leftarrow A[0]$

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

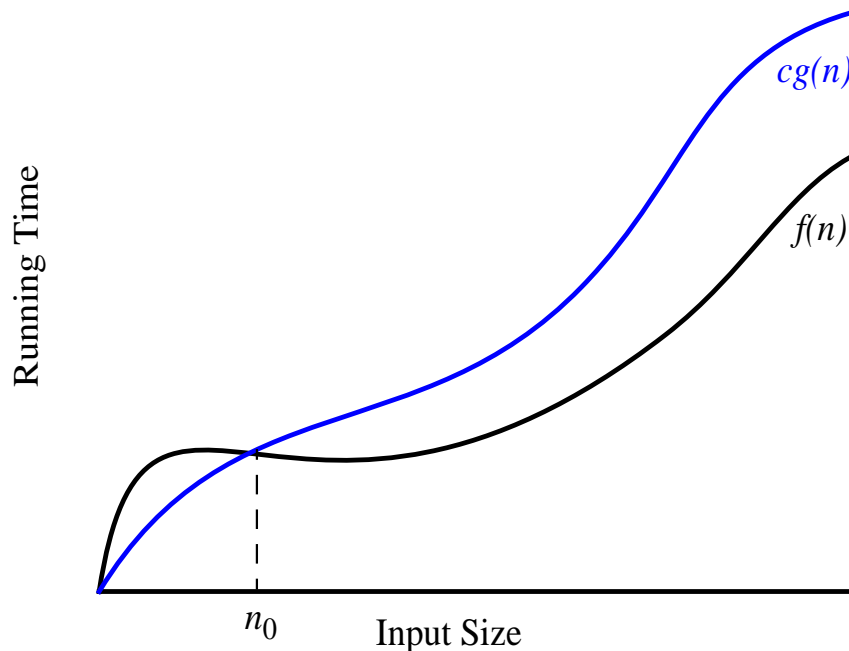
if $currentMax < A[i]$ **then**

$currentMax \leftarrow A[i]$

return $currentMax$

Asymptotic Notation

- Goal: To simplify analysis by getting rid of unneeded information
 - Like “rounding”: 1,000,001 \approx 1,000,000
 - $3n^2 \approx n^2$
- The “Big-Oh” Notation
 - given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if and only if $f(n) \leq c g(n)$ for $n \geq n_0$
 - c and n_0 are constants, $f(n)$ and $g(n)$ are functions over non-negative integers



Asymptotic Notation (cont.)

- **Note:** Even though $7n - 3$ is $O(n^5)$, it is expected that such an approximation be of as small an order as possible.
- **Simple Rule:** Drop lower order terms and constant factors.
 - $7n - 3$ is $O(n)$
 - $8n^2 \log n + 5n^2 + n$ is $O(n^2 \log n)$
- Special classes of algorithms:
 - logarithmic: $O(\log n)$
 - linear $O(n)$
 - quadratic $O(n^2)$
 - polynomial $O(n^k), k \geq 1$
 - exponential $O(a^n), a > 1$
- “Relatives” of the Big-Oh
 - $\Omega(f(n))$: Big Omega
 - $\Theta(f(n))$: Big Theta

Asymptotic Analysis of The Running Time

- Use the Big-Oh notation to express the number of primitive operations executed as a function of the input size.
- For example, we say that the `arrayMax` algorithm runs in $O(n)$ time.
- Comparing the asymptotic running time
 - an algorithm that runs in $O(n)$ time is better than one that runs in $O(n^2)$ time
 - similarly, $O(\log n)$ is better than $O(n)$
 - hierarchy of functions:
 - $\log n \ll n \ll n^2 \ll n^3 \ll 2^n$
- **Caution!**
 - Beware of very large constant factors. An algorithm running in time $1,000,000 n$ is still $O(n)$ but might be less efficient on your data set than one running in time $2n^2$, which is $O(n^2)$

Example of Asymptotic Analysis

- An algorithm for computing prefix averages

Algorithm prefixAverages1(X):

Input: An n -element array X of numbers.

Output: An n -element array A of numbers such that

$A[i]$ is the average of elements $X[0], \dots, X[i]$.

Let A be an array of n numbers.

for $i \leftarrow 0$ **to** $n - 1$ **do**

$a \leftarrow 0$

for $j \leftarrow 0$ **to** i **do**

$a \leftarrow a + X[j]$

$A[i] \leftarrow a / (i + 1)$

return array A

- Analysis ...

Example of Asymptotic Analysis

- A better algorithm for computing prefix averages:

Algorithm prefixAverages2(X):

Input: An n -element array X of numbers.

Output: An n -element array A of numbers such that $A[i]$ is the average of elements $X[0], \dots, X[i]$.

Let A be an array of n numbers.

$s \leftarrow 0$

for $i \leftarrow 0$ **to** $n - 1$ **do**

$s \leftarrow s + X[i]$

$A[i] \leftarrow s / (i + 1)$

return array A

- Analysis ...

Advanced Topics: Simple Justification Techniques

- By Example
 - Find an example
 - Find a counter example
- The “Contra” Attack
 - Find a contradiction in the negative statement
 - Contrapositive
- Induction and Loop-Invariants
 - Induction
 - 1) Prove the base case
 - 2) Prove that any case n implies the next case $(n + 1)$ is also true
 - Loop invariants
 - Prove initial claim S_0
 - Show that S_{i-1} implies S_i will be true after iteration i

Advanced Topics: Other Justification Techniques

- Proof by Excessive Waving of Hands
- Proof by Incomprehensible Diagram
- Proof by Very Large Bribes
 - see instructor after class
- Proof by Violent Metaphor
 - Don't argue with anyone who always assumes a sequence consists of hand grenades
- The Emperor's New Clothes Method
 - "This proof is so obvious only an idiot wouldn't be able to understand it."