Welcome!

CS456 - Algorithm Design & Analysis

Ashesi University
2024 Semester 1

Instructor : Sampson D Asare (Office : Eng. 205-F)

email: sasare@ashesi.edu.gh

F. I. : Leslie N. Kodjoe

Email: leselie.kodjoe@ashesi.edu.gh

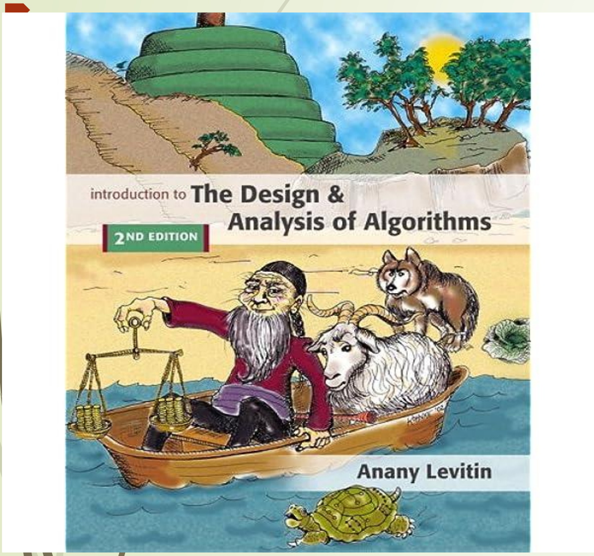
Consultation times: Mondays/Wednesdays : 1 – 2 pm or by Appointment.

Students introduce themselves

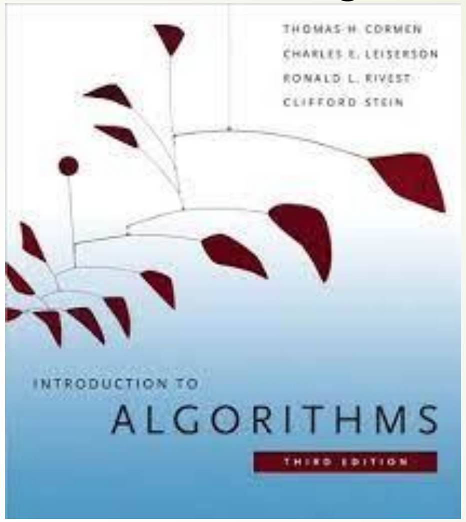
Course Materials

1. **Primary text**: **A. Levitin,**

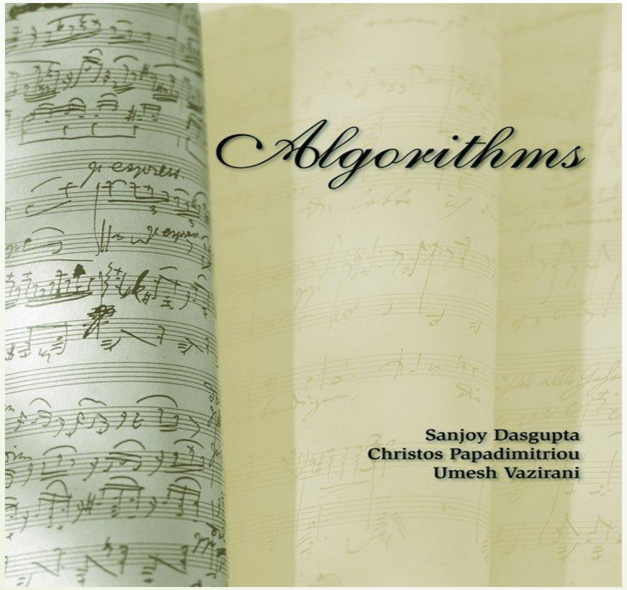
Introduction to the Design and Analysis of Algorithms ISBN **978-0132316811**



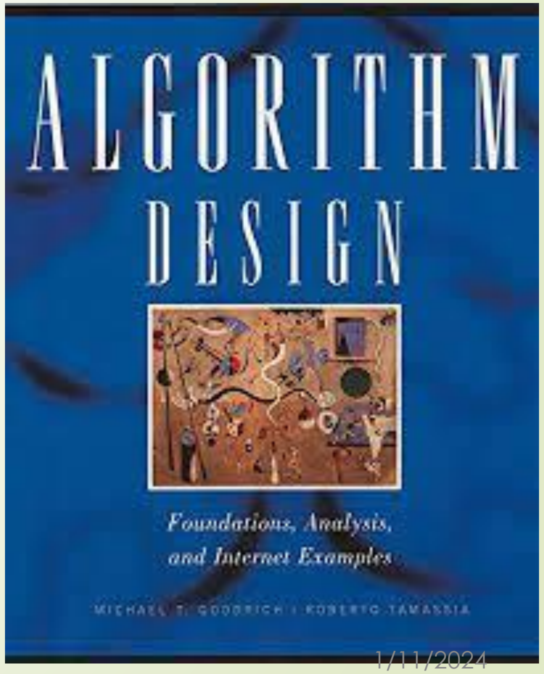
2. T. Cormen, C. E. Leiserson, R. L. Rivest, Clifford Stein, *Introduction to Algorithms*



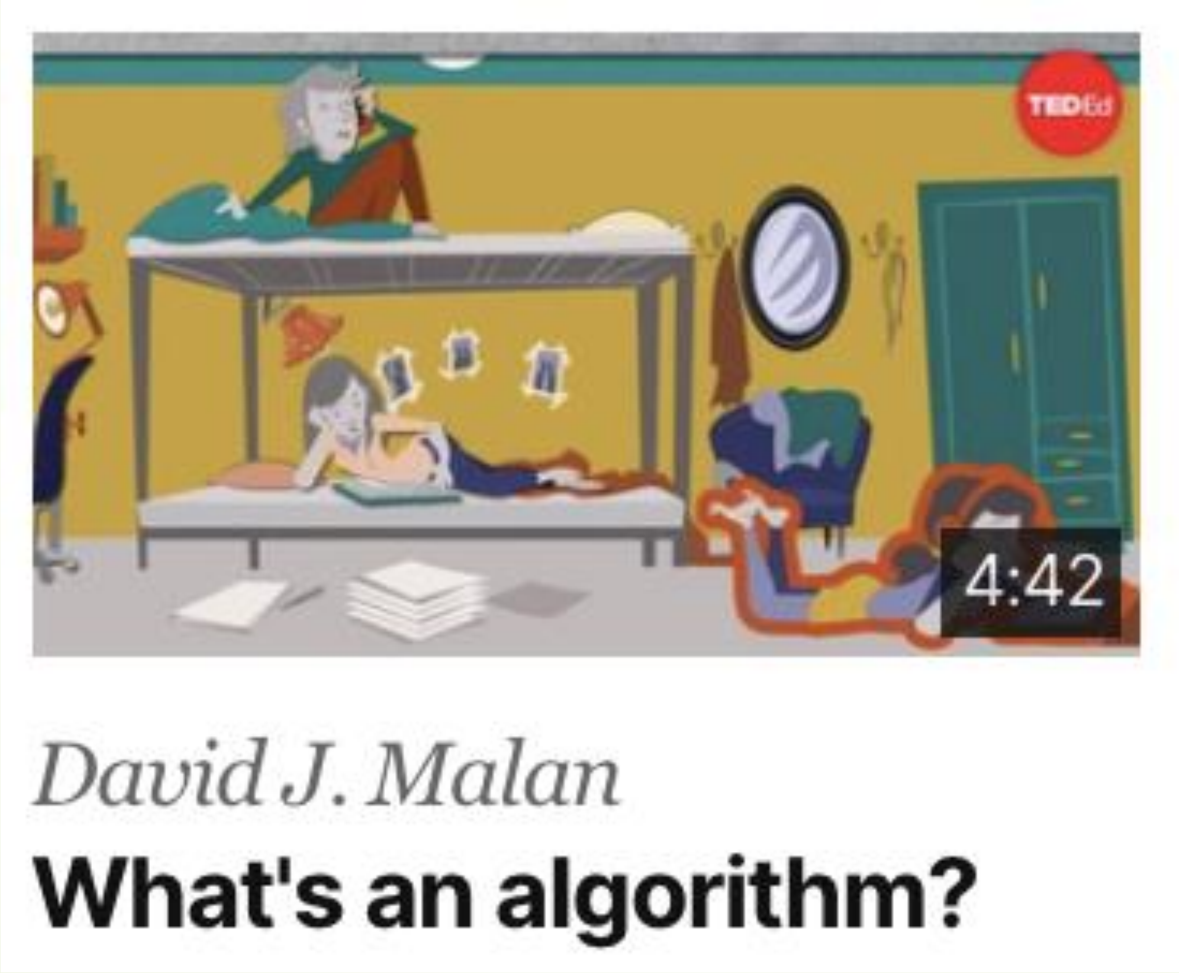
3. S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*



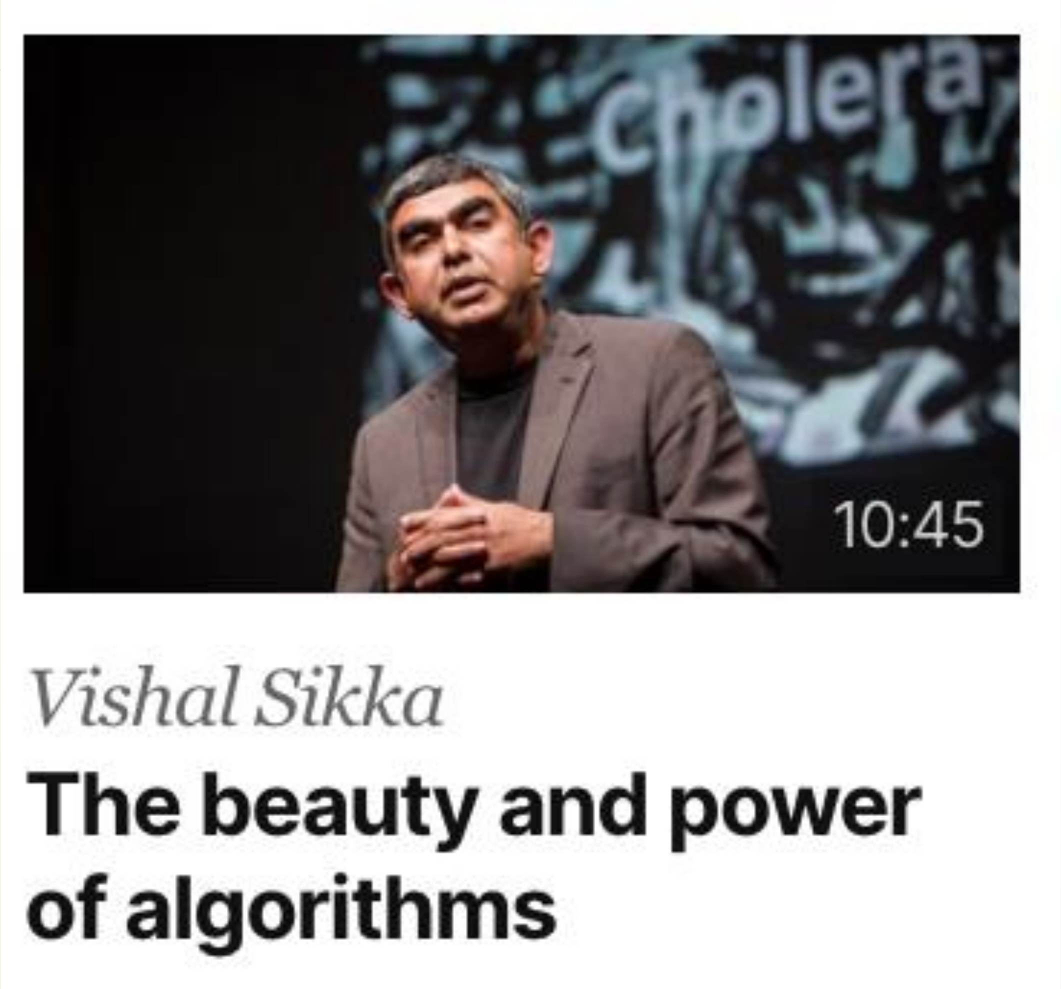
4. Michael T. Goodrich & Roberto Tamassia, ***Algorithm Design and Applications***, ISBN: 978-1118335918, Published by Wiley



Pre-Lecture Video



Another short Video on Algorithms



Small Group Discussions (15 minutes) ...1/2

Discuss:

1. What is an algorithm?
2. What do you understand by the statement that algorithms are "**part math, part design"**?
3. Discuss the meaning of algorithms "**learn and adapt**". What did the presenter mean by this?
4. Think about the algorithms that lie underneath the various systems we interact with in our daily life in this modern world. Is there a particular algorithm that you would really like to understand better? (e.g. GPS, Facebook, Internet , ChatGPT, etc) and why?
5. What is/are your expectation(s) in this Algorithm class? Outline a "**formula/steps by step you will follow to learn in this course**"

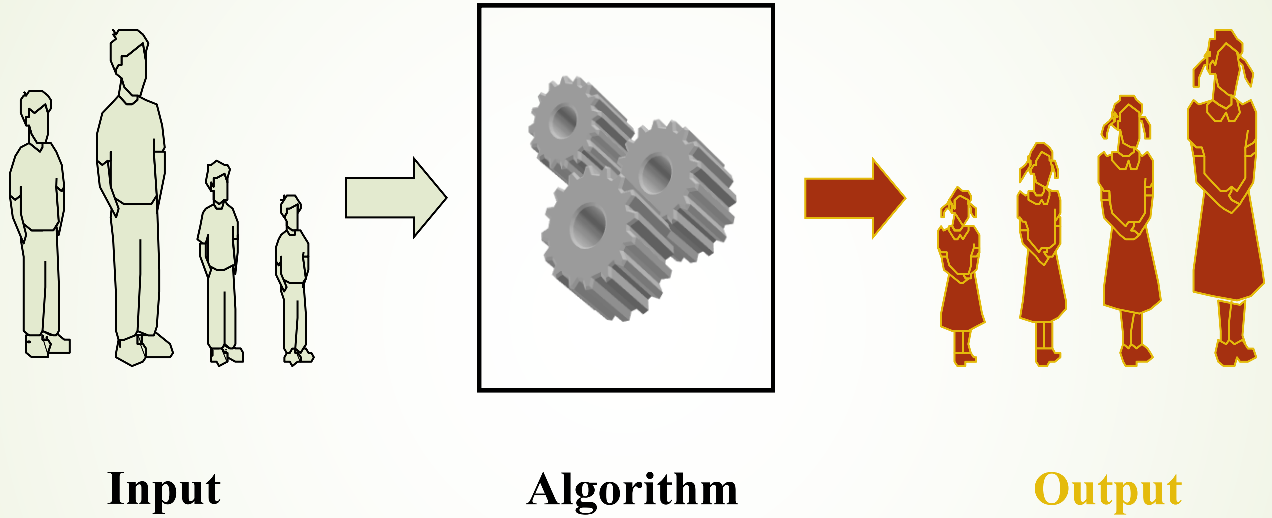
Small Group Discussions – Reporting(15 minutes) …2/2

- Choose one representative to report back briefly (~2-3 minutes) on your discussion

Review of Analysis of Algorithms

- Following slides were adapted from Michael Goodrich and Tamassia book on Data Structures and Algorithms

Analysis of Algorithms



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

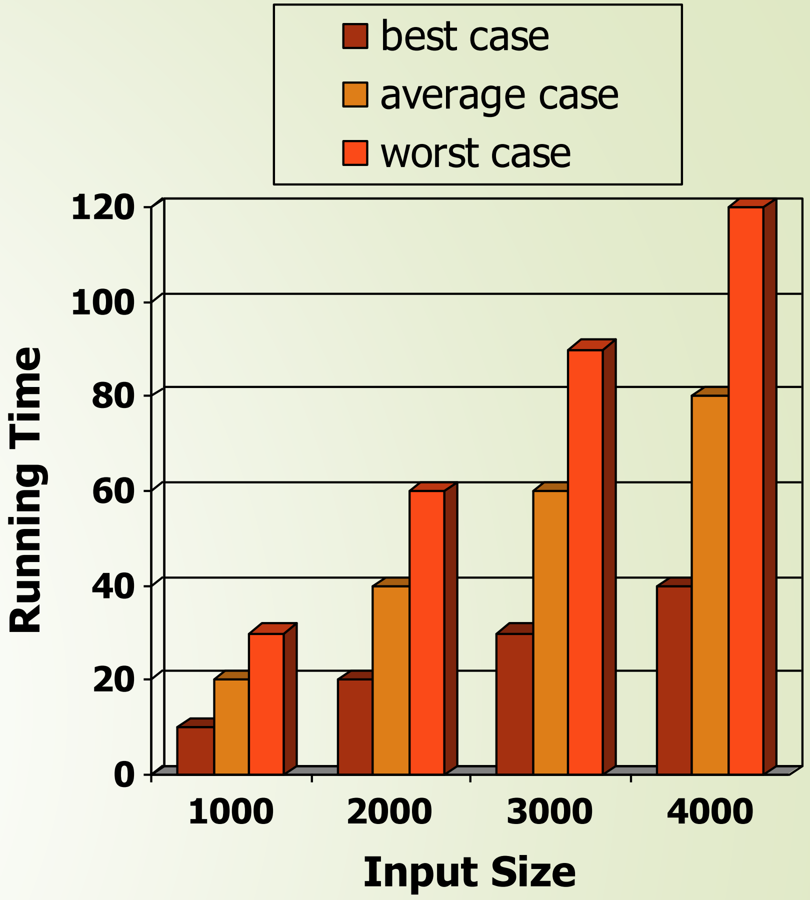
What is a Problem in Algorithm?

- A problem is a **function or a mapping of inputs to outputs**
- Examples of problems are:

1. **Sorting** - sort any random input into desc/asc order,
2. **Searching** – sequential search in a list/binary search in a tree or depth/breadth first search in a graph
3. **String Processing** – pattern match, etc
4. **Graph problems** – TSP/ Shortest part/ Flow /Network/etc.
5. **Combinatorial problems** - graph coloring problem,

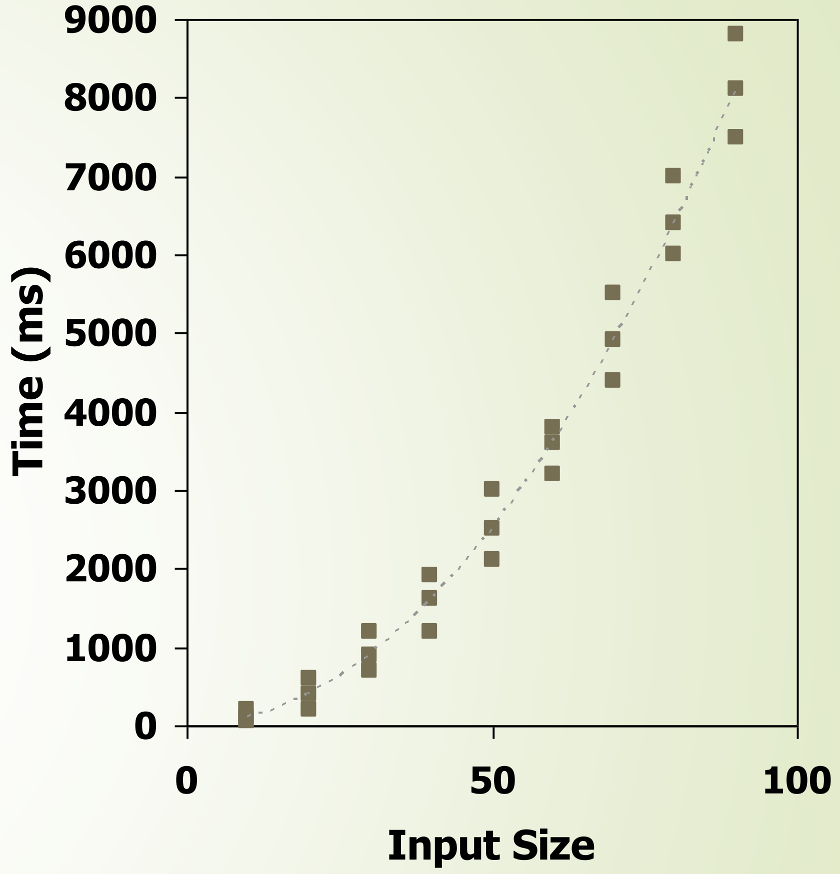
Running Time

- 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 most applications such as games, finance and robotics



A: Experimental Studies

- Write a program/ implement the algorithm
- Run the program with inputs of varying sizes and compositions
- Use a method like System.currentTimeMillis() to get an accurate measure of the actual running time
- Plot the results showing the various time against the inputs



Limitations of Experimental approach

- 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

B:Theoretical Analysis

- Uses a high-level description **(pseudocode)** of the algorithm instead of an actual implementation/experiment
- 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

Pseudocode - to be used to describe Algorithms

- 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

Example: find max element of an array

Algorithm *arrayMax*(*A*, *n*)
  Input array *A* of *n* integers
  Output maximum element of *A*

  currentMax ← *A*[0]
  for *i* ← 1 **to** *n* − 1 **do**
    if *A*[*i*] > *currentMax* **then**
      currentMax ← *A*[*i*]
  return *currentMax*

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

  var.method (*arg* [, *arg*…])
- Return value

  return *expression*
- Expressions

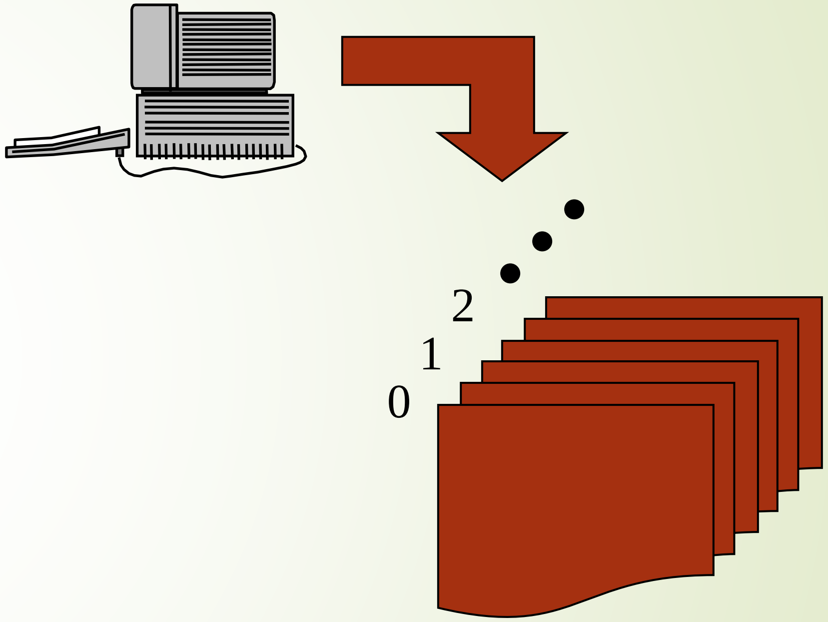
  ← Assignment (like = in Java)

  = Equality testing (like == in Java)

  n^2 Superscripts and other mathematical formatting allowed

The Random Access Memory (RAM) Model

- A **CPU**



- A potentially unbounded bank of **memory** cells, each of which can hold an arbitrary numbers or characters

- Memory cells are numbered and accessing any cell in memory takes unit time.

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 the RAM model

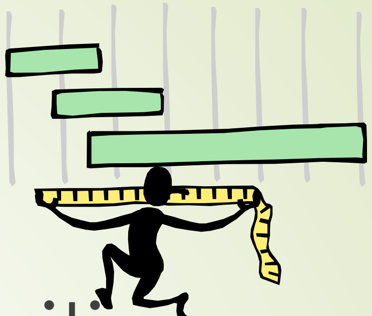
- Examples:
  - Evaluating an expression
    - **X ← (X+1)*3**
  - Assigning a value to a variable
    - **X ← X+1**
  - Indexing into an array
    - **A[0]**
  - Calling a method
    - **jpr.add(obj);**
  - Returning from a method
    - **return A[ind]**

Time Complexity Method 1: Counting Primitive Operations (aka Cost Counting method)

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

Algorithm *arrayMax(A, n)*	max #operations
currentMax $\leftarrow$ *A*[0]	2
for *i* $\leftarrow$ 1 **to** *n* – 1 **do**	***n-1***
if *A*[*i*] > *currentMax* **then**	$2(n-1)$
currentMax $\leftarrow$ *A*[*i*]	$2(n-1)$
return *currentMax*	1
	Total $5n-2$

Estimating Running Time



- Algorithm ***arrayMax*** executes $5n - 2$ 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

  $$a\,(5n - 2) \le T(n) \le b(5n - 2)$$

- Hence, the running time $T(n)$ is bounded by two linear functions (more on this later)

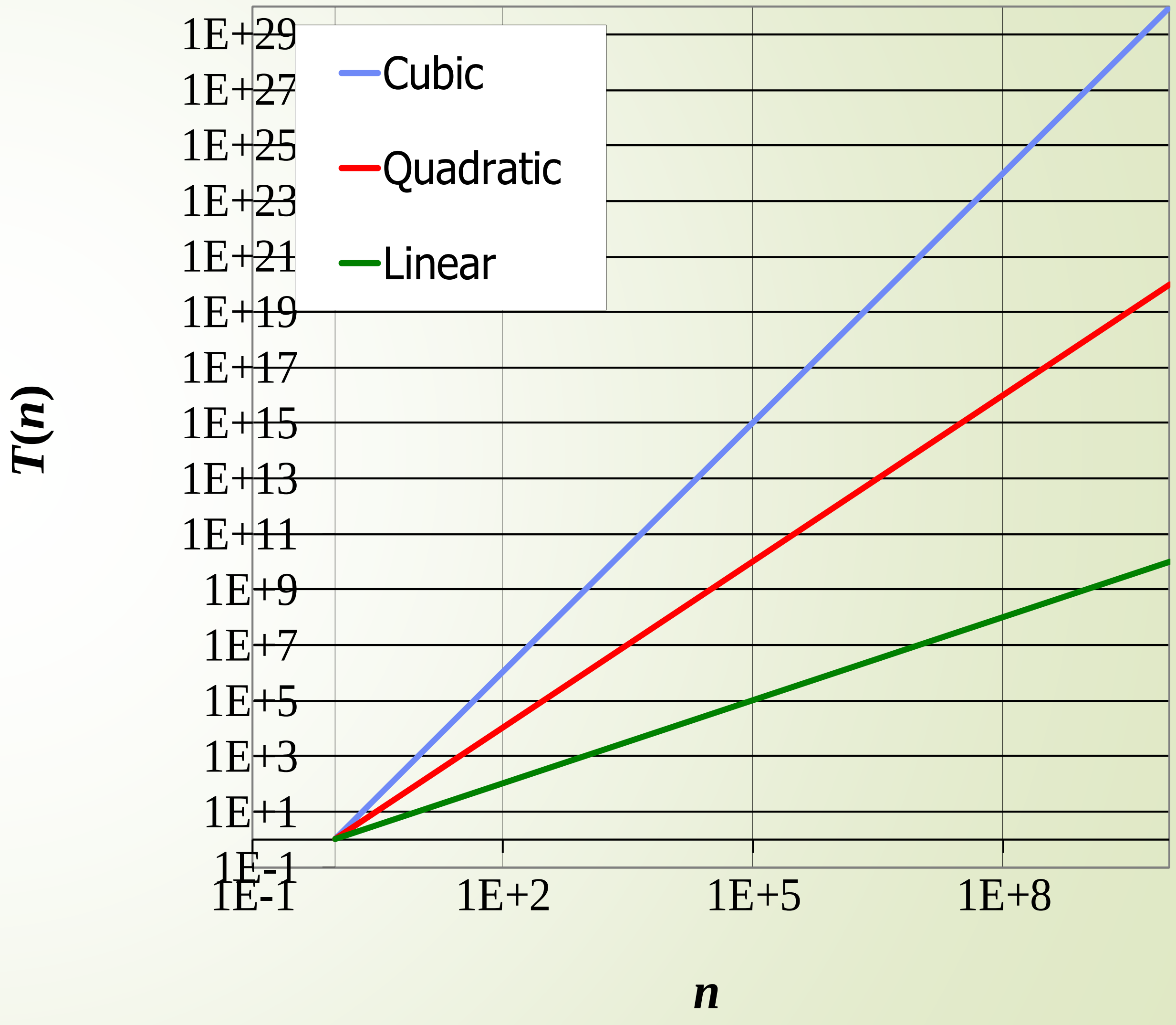
Growth Rate of Running Time - Asymptotic

- 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 ***arrayMax*** [also known as Asymptotic]

Seven Important Asymptotic Functions to note

- Seven functions that often appear in algorithm analysis:
- **Function** **Corresponding Big Oh**
  - Constant ≈ $\mathit{1}$ – $O(1)$
  - Logarithmic ≈ log n – $O(\log N)$
  - Linear ≈ n -- $O(N)$
  - N-Log-N ≈ n log n – $O(N \log N)$
  - Quadratic ≈ n^2 --- $O(N^2)$
  - Cubic ≈ n^3 --- $O(N^3)$
  - Exponential ≈ 2^n --- $O(2^n)$
- In a log-log chart, the slope of the line corresponds to the growth rate of the function



Constant Factors/Lower terms

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