SE2228
Analysis and Design of Algorithms

Introduction

Al-Khwarizmi : About 780 - about 850
(In Turkish : El Harezmi)

Al'Khwarizmi was an Islamic mathematician of Central Asian Turcic origins (Khorasan,Uzbekistan), who wrote on Hindu-Arabic numerals and was among the first to use zero as a place holder in positional base notation.

The word *algorithm* derives from his name. His algebra treatise "*Hisab al-jabr w'al-muqabala*" gives us the word *algebra* and can be considered as the first book to be written on algebra.

About the Course

- Purpose: A rigorous introduction to the design and analysis of algorithms
- In this course, we will look at:
  - *The design and analysis of Algorithms* for solving problems efficiently
  - *Advanced Data structures* for efficiently storing, accessing, and modifying data.

Main Reference Books:

1. *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein
2. The Design and Analysis of Algorithms, A.Levithin.
3. Data Structures and Algorithms in C++,Goodrich,Tamasia,Mount (Other options: Java,Python)

About the Course

- Format:
  - Two lectures/week/Two lab hours
  - Homeworks-Assignments
    - Problem sets
    - Occasional programming assignments
  - One or two Quiz exams+Midterm + Final exam
  - Attendance and participation are effective on the final grade.

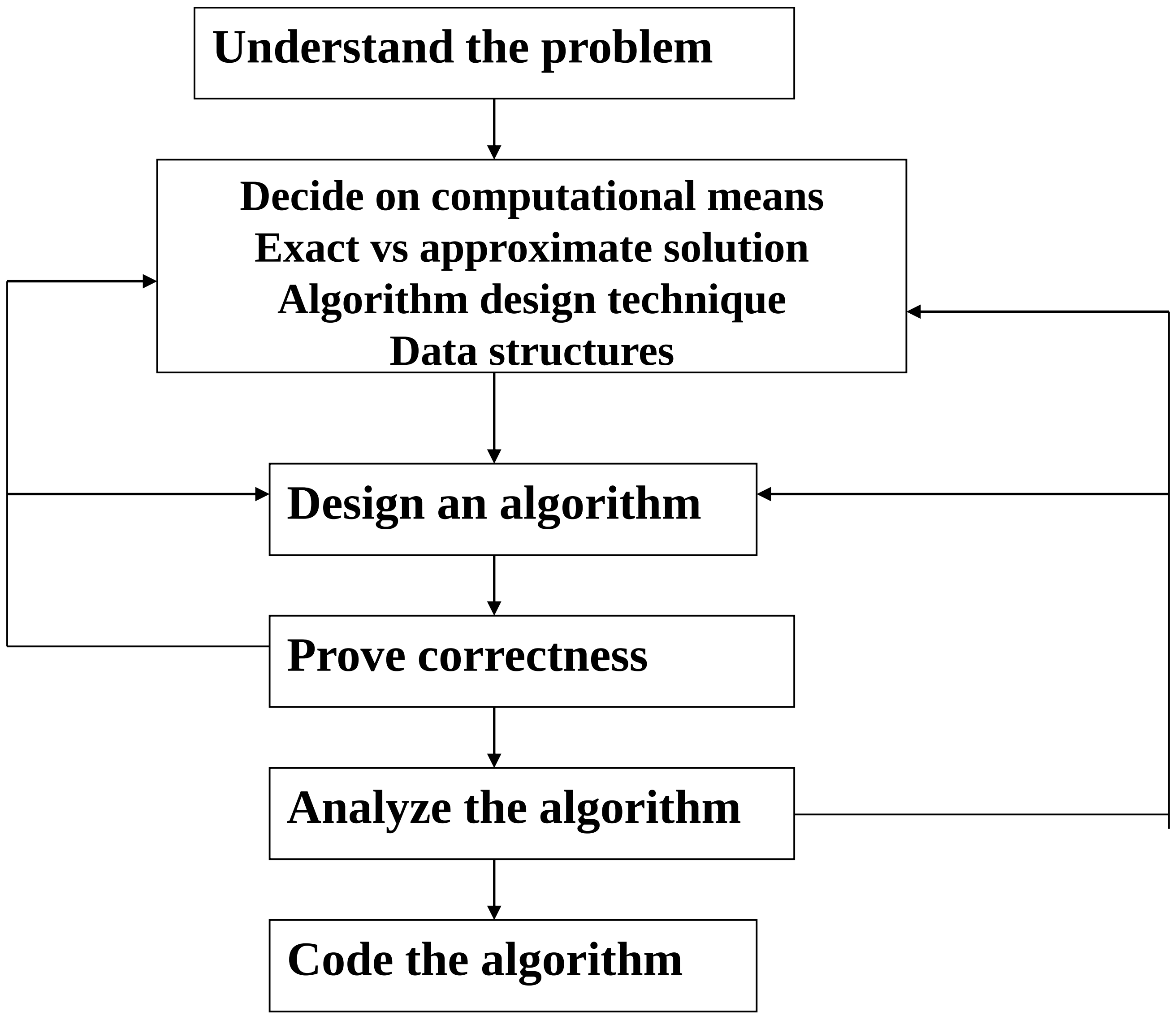
A Review of Basic Concepts

- Computer Science is <u>the study of algorithms</u>, including:
  - Their formal and mathematical properties
  - Their hardware realizations
  - Their linguistic realizations
  - Their applications
- Formal and mathematical properties of algorithms:
  - Correctness , efficiency, computability …
  - A central concept in Computer Science
  - Consequently, it is also a central concept in Computer and Software Engineering.

Problem Solving

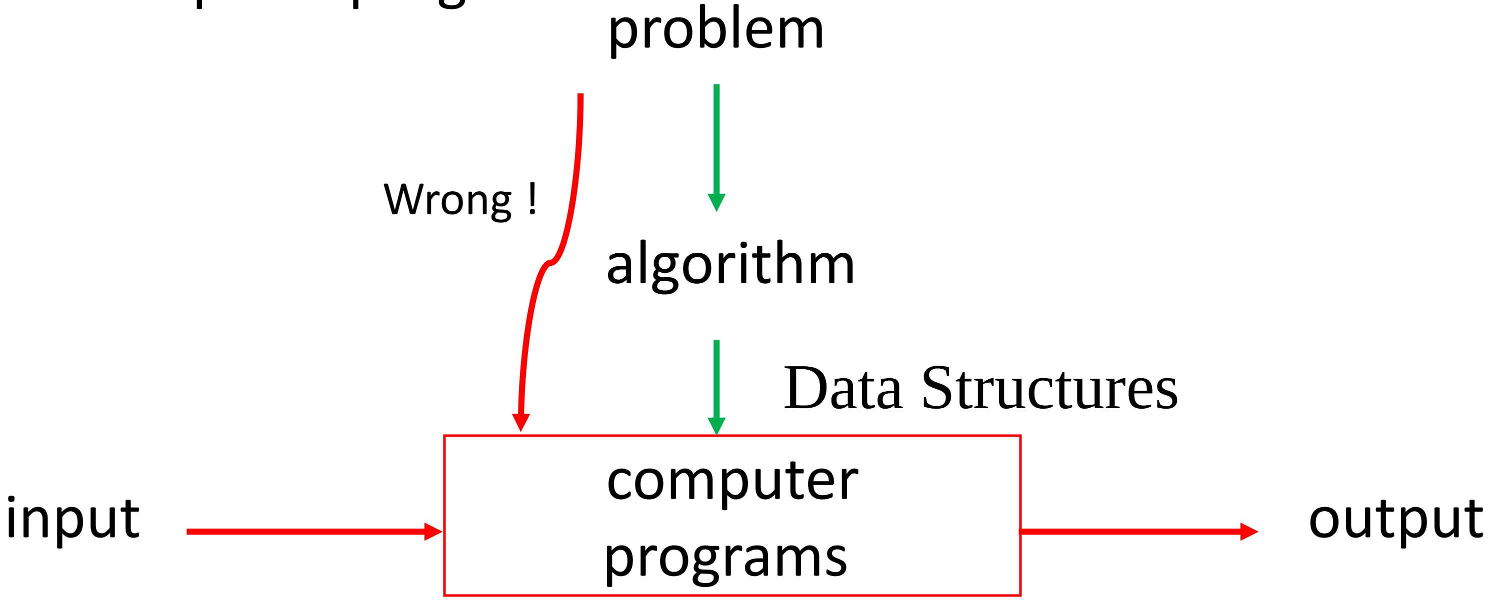
- The single most important skill for a computer scientist/engineer is problem solving.
- Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately.
- Learning about algorithms is an excellent opportunity to practice and improve problem-solving skills

Problem Solving Steps in Computer Science



Algorithms and Data Structures

- An algorithm is a sequence of unambiguous instructions for solving a problem.
- A data structure is a specialized format for organizing, processing, retrieving and storing data . Data structures are intermediary between algorithms and computer programs.



Properties of Algorithms

- An algorithm can be a recipe, process, method, technique, procedure, routine,… with the following requirements:
  - Finiteness: terminates after a finite number of steps
  - Definiteness: the steps are unambiguously specified
  - Input: valid inputs are clearly specified
  - Output: can be proven to produce the correct output given a valid input
  - Effectiveness: steps are sufficiently simple, basic,and “feasible”

Some Important Points

- Each step of an algorithm should be clear and precise
- The range of inputs has to be specified carefully
- The same algorithm can be represented in different ways
- The same problem may be solved by different algorithms
- Different algorithms may take different times to solve the same problem – we may prefer one to the other

Some Important Points

- Algorithm analysis
  - Analysis of resource usage of given algorithms (time , space)
- Efficient algorithms
  - Algorithms that make an efficient usage of resources
  - Efficient algorithms lead to efficient programs
- Algorithm design
  - Methods for designing efficient algorithms

Complexity and Efficiency

- For a given algorithm, the resources needed for its execution determine the complexity of that algorithm.
- A related concept is efficiency: Efficiency of algorithms and complexity of algorithms are the same area.
- The same algorithm can be represented in different ways and there may be several algorithms for solving the same problem
- The complexity of a problem is the complexity of the algorithm that solves this problem.
- How to classify problems according to their computational difficulty?

Complexity and Efficiency

- Suppose we have two algorithms, how can we tell which one is better?
- We could implement both algorithms, run them both
  - Expensive and error prone
- Preferably, we should analyze them mathematically:

  → *Algorithm analysis*
- Algorithm analysis is expected to produce a simple measure of complexity.

Classification of Computational Problems

"How difficult is it to solve a given algorithmic problem?"

- If something can be computed, what resources (time, memory,storage …) will be needed?

  →A computation may be possible but not practical !

- Some problems do not have algorithmic solutions at all:

  →Solvable/unsolvable problems

- Some *solvable* problems are easy problems, some are hard.
- The hard problems have algorithmic solutions that do not complete in a reasonable amount of time:

  tractable / intractable problems <--> easy / hard problems

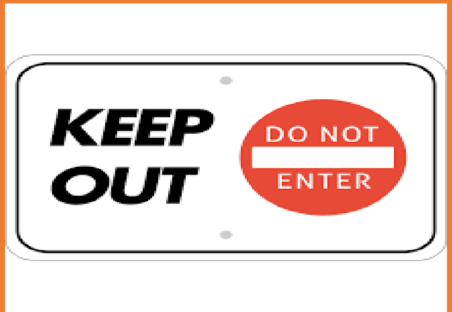
All computational problems

Solvable problems

Unsolvable problems

Tractable (easy) problems

Intractable (hard) problems



Solvable Problems

- Tractable (easy) problems
  - There exists a solution of polynomial time or better efficiency :

    → polynomial-time problems
- Intractable (hard) problems
  - There exists no exact solution better than exponential efficiency :

    → non-polynomial (exponential) time problems

Dealing with Hard Problems

- In order to deal practically with hard computational problems, one must often accept *weakening of the requirement to guarantee computing the correct result*.

This can be done in different ways:

- Accept a "reasonable" solution which is not exactly correct and slightly suboptimal.

  → approximate algorithms.
- Accept that the probability for finding a correct, or optimal solution may be slightly less than 1
  →randomized algorithms.

Categories of Algorithms

This leads us to two independent categories of algorithms:

1. Exact (precise) algorithms vs. approximate algorithms
2. Deterministic algorithms vs. randomized (stochastic) algorithms

Important Computational Problem Types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Numerical problems
- Optimization problems
- Geometric problems
- ……………………..

Review of Fundamental Math

- If you understand the first n slides,
  you will understand the slide n+1

Floor and ceiling functions

- The *floor* function maps any real number x onto the greatest integer less than or equal to x:

$$\lfloor 3.2 \rfloor = \lfloor 3 \rfloor = 3$$

$$\lfloor -5.2 \rfloor = \lfloor -6 \rfloor = -6$$

- The *ceiling* function maps x onto the least integer greater than or equal to x:

$$\lceil 3.2 \rceil = \lceil 4 \rceil = 4$$

$$\lceil -5.2 \rceil = \lceil -5 \rceil = -5$$

Floor and Ceiling :More Examples

$\lfloor X \rfloor$ Floor function: the largest integer $\leq$ X

$\lfloor 2.7 \rfloor = 2$ $\quad$ $\lfloor -2.7 \rfloor = -3$ $\quad$ $\lfloor 2 \rfloor = 2$

$\lceil X \rceil$ Ceiling function: the smallest integer $\geq$ X

$\lceil 2.3 \rceil = 3$ $\quad$ $\lceil -2.3 \rceil = -2$ $\quad$ $\lceil 2 \rceil = 2$

Summations : Arithmetic Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$ ≈ $n^3/3$ for large n

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

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

Proof : Sum of first n Integers

Write out the series twice and add each column

$$\begin{array}{rcccccccccccc} & 1 & + & 2 & + & 3 & + \cdots + & n-2 & + & n-1 & + & n \\ + & n & + & n-1 & + & n-2 & + \cdots + & 3 & + & 2 & + & 1 \\ \hline \end{array}$$

$$(n+1) + (n+1) + (n+1) + \cdots + (n+1) + (n+1) + (n+1)$$

$$= n\,(n+1)$$

Since we added the series twice, we must divide the result by 2

Sum Manipulation Rules

1. $$\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$$

2. $$\sum_{i=l}^{u}(a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

3. $$\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i, \text{ where } l \le m < u$$

Summation Properties

$$\sum_{k=1}^{n} ca_k = c\sum_{k=1}^{n} a_k$$

$$\sum_{k=1}^{n} a_k \pm b_k = \sum_{k=1}^{n} a_k \pm \sum_{k=1}^{n} b_k$$

$$\sum_{k=1}^{n} c = nc$$

$$\sum_{i=a}^{b} 1 = b-a+1$$

Summation Examples-1

Example 1

$$\sum_{k=1}^{3}(2k+1) = (2\times1+1)+(2\times2+1)+(2\times3+1)$$
$$= 3+5+7$$
$$= 15$$

Example 2

$$\sum_{k=0}^{4}(-2)^k = (-2)^0+(-2)^1+(-2)^2+(-2)^3+(-2)^4$$
$$= 1+(-2)+4+(-8)+16$$
$$= 11$$

Example 3

$$\sum_{k=2}^{5}(3k^2-7) = (3\times2^2-7)+(3\times3^2-7)+(3\times4^2-7)+(3\times5^2-7)$$
$$= 5+20+41+68$$
$$= 134$$

Summation Examples-2

1.What is this ? $\sum_{i=1}^{4}\sum_{j=1}^{3} ij$

- The double summation is equivalent to:

```
int sum = 0;
for ( int i = 1; i <= 4; i++ )
    for ( int j = 1; j <= 3; j++ )
        sum += i*j;
```

2. $\sum_{i=1}^{6}\left(i^2+1\right) = \sum_{i=1}^{6} i^2 + \sum_{i=1}^{6} 1$

Summations : Examples-3

Evaluate $\sum^{10}(4i-3)$

$$\sum_{i=1}^{10}(4i-3)=\sum_{i=1}^{10}4i-\sum_{i=1}^{10}3=4\sum_{i=1}^{10}i-3\sum_{i=1}^{10}1=4\frac{10(10+1)}{2}-3(10)$$

=190

Evaluate $\sum_{i=1}^{6}(i^3-i^2)$.

=350

Geometric Series

- Series with a constant ratio between successive terms.

Example-1:

1/2, 1/4, 1/8 , 1/16,...

→Each successive term can be obtained by multiplying the previous term by 1/2.

Example-2 :

1, 2, 4, 8, 16, 32, 64, 128, 256, ...

→Each successive term can be obtained by multiplying the previous term by 2.

Geometric Series

The sum of geometric series with a common ratio r can be expressed as

$$\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}$$

and if $|r| < 1$ then it is also true that

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$$

(For justification,try the formulas using small n,k and r=1/2)

Geometric series

A common geometric series involves the ratios

r = ½ and r = 2. Use these in the expressions on previous page :

$$\sum_{i=0}^{n}\left(\frac{1}{2}\right)^{i} = \frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}} = 2-2^{-n} \qquad \sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^{i} = 2$$

$$\sum_{k=0}^{n} 2^{k} = \frac{1-2^{n+1}}{1-2} = 2^{n+1}-1$$

Combinatorial Formulas

You have **n** items and want to find the number of ways **k** items can be ordered(Selected):

Permutation: Order is important→ AB ≠ BA

The number of possible results :

$$nPr = n!/(n - r)!$$

Combination (Binomial factors,Order is not important→ AB=BA):

$$nCr = n!/r!(n - r)!$$

→Permutation : (Ali,Selin) ≠(Selin,Ali), Who is the first ?

Combination : (Ali,Selin)=(Selin,Ali) , We just need a group of two.

Combinations

You have also seen this in expanding polynomials:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

For example,

$$(x+y)^4 = \sum_{k=0}^{4} \binom{4}{k} x^k y^{4-k}$$

$$= \binom{4}{0} y^4 + \binom{4}{1} xy^3 + \binom{4}{2} x^2 y^2 + \binom{4}{3} x^3 y + \binom{4}{4} x^4$$

$$= y^4 + 4xy^3 + 6x^2 y^2 + 4x^3 y + x^4$$

Exponents

- $X^0 = 1$ by definition
- $X^a X^b = X^{(a+b)}$
- $X^a / X^b = X^{(a-b)}$

  Show that : $X^{-n} = 1 / X^n$

- $(X^a)^b = X^{ab}$

Logarithms

- The log function is the inverse of an exponent

$\rightarrow \log_a X = Y \Leftrightarrow a^Y = X, \; a > 0, X > 0$

E.G: $\log_2 8 = 3$; Since $2^3 = 8$

a is the "base" of logarithm

- $\log_a 1 = 0$ because $a^0 = 1$

logX means $\log_2 X$ (In computer science)

lnX means $\log_e X$

In most algorithm related problems we use base a=2 .

- Warning:Since a positive number to any exponent can never be negative, it follows that you can never take the log of any negative value(and 0).

Logarithms and Exponents : Meaning

$x = 2^y$ if $\log_2 x = y$

- $32 = 2^5$, so $\mathbf{log_2}\, 32 = 5$
- $65536 = 2^{16}$, so $\mathbf{log_2}\, 65536 = 16$

- The exponent of a number says how many times to use the number in a multiplication. e.g. $2^3 = 2 \times 2 \times 2 = 8$

  (2 is used 3 times in a multiplication to get 8)

- A logarithm says how many times one number to multiply to get another number.

  →It asks "what exponent produced this?"

  e.g. $\mathbf{log_2} 8 = 3$ *(2 makes 8 when used 3 times in a multiplication)*

Logarithms : Some Identities

1. If $x>y$,$\log x > \log y$
2. $\log_a(XY) = \log_a X + \log_a Y$
3. $\log_a(X/Y) = \log_a X - \log_a Y$
4. $\log_a(X^n) = n\log_a X$
5. $\log_a b = (\log_2 b)/ (\log_2 a)$
6. $a^{\log_a x} = x$ (Ex: $2^{\log 64} = 64$)

Equivalence of Logarithms

Any base B log is equivalent to base 2 log within a constant factor.

In particular,

$\log_2 x = 3.22 \log_{10} x$

In general, all logarithms are scalar multiples of each others:

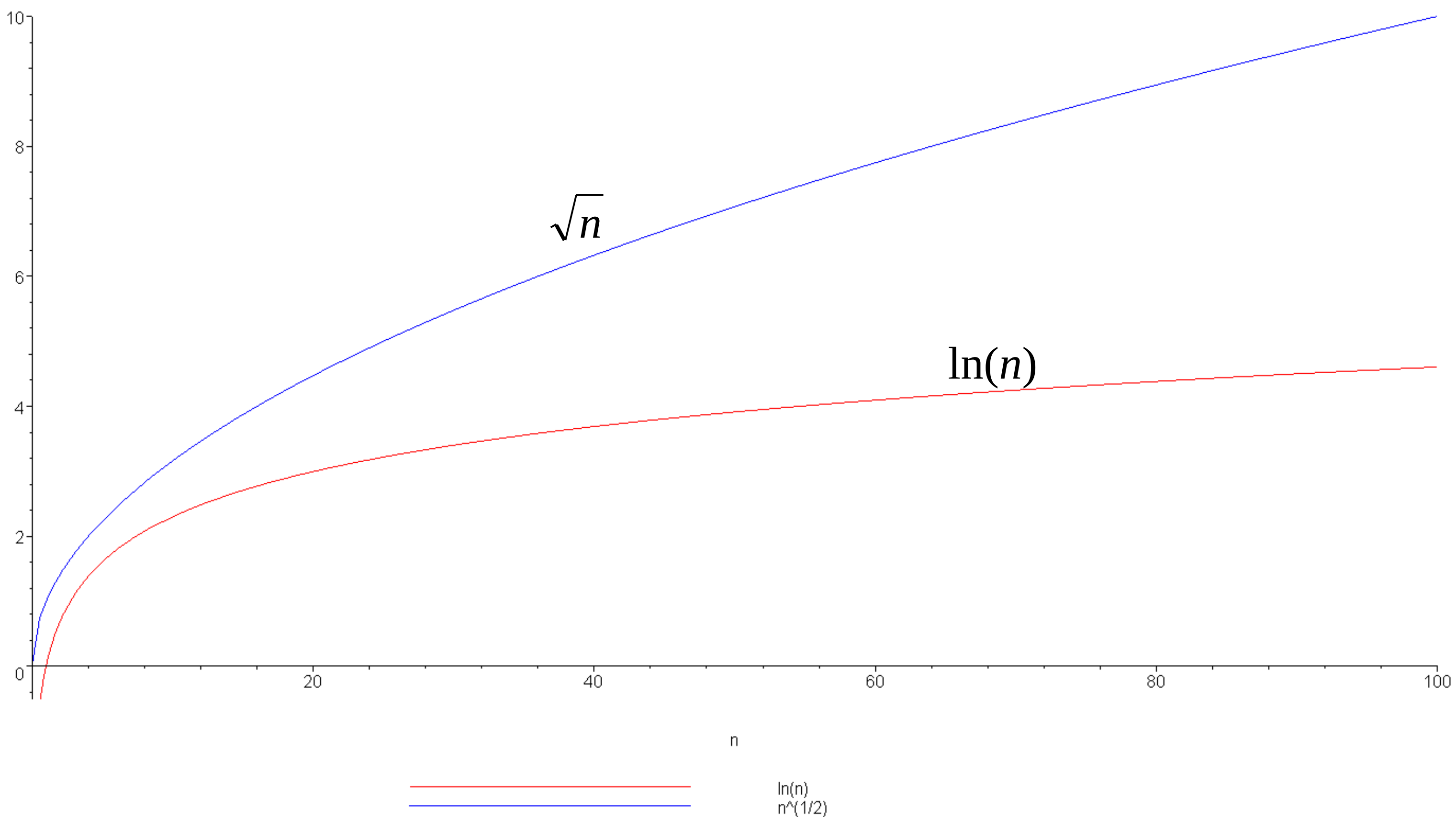
$$\log_B x = (\log_A x) / (\log_A B)$$

This matters in doing math but not CS!

→In algorithm analysis, we tend to not care much about constant factors

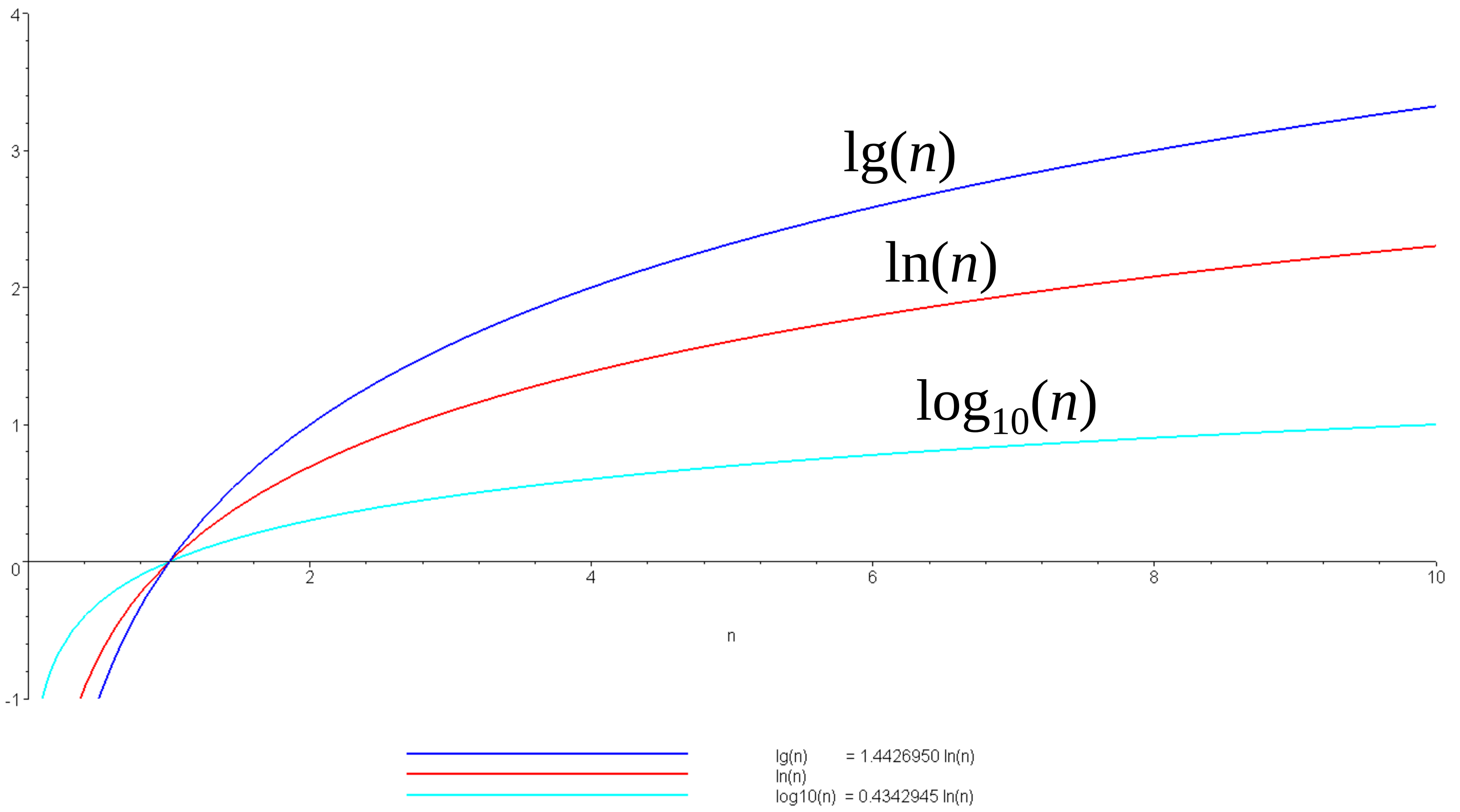
Logarithms : Growth

Example: $f(n) = n^{1/2} = \sqrt{n}$ is strictly greater than $\ln(n)$



Logarithms

A plot of $\log_2(n) = \lg(n)$, $\ln(n)$, and $\log_{10}(n)$



Function Growth and Limits

- $\lim (n) = \infty, n \rightarrow \infty$
- $\lim (n^a) = \infty, n \rightarrow \infty, a > 0$
- $\lim (1 / n) = 0, n \rightarrow \infty$
- $\lim (1 / (n^a)) = 0, n \rightarrow \infty, a > 0$
- $\lim (\log(n)) = \infty, n \rightarrow \infty$
- $\lim (a^n) = \infty, n \rightarrow \infty, a > 0$

Examples

- lim (n/ n^2) = 0, $n \to \infty$
- lim (n^2 / n) = ∞, $n \to \infty$
- lim (n^2 / n^3) = 0, $n \to \infty$
- lim (n^3 / n^2) = ∞, $n \to \infty$
- lim $(n / ((n+1)/2)$ = 2, $n \to \infty$.

Recurrence Relations

Sequences may be defined explicitly:

$x_n = 1/n$

$\rightarrow 1, {}^1/_2, {}^1/_3, {}^1/_4, \ldots$

- A recurrence relationship is a means of defining a sequence based on previous values in the sequence
- Such definitions of sequences are said to be *recursive*

Recurrence Relations

- Recurrence relations are recursive definitions of mathematical functions or sequences.

Examples:

1. $g(n) = g(n-1) + 2n - 1$

   $g(0) = 0$

2. $f(n) = f(n-1) + f(n-2)$

   $f(1) = 1$

   $f(0) = 1$

Solving Recurrence Relations

Example: Find the value of $T(n) = T(n-1) + 1$ for $n=4$,

Initial condition $T(1)=2$

Substituting up from T(1):

- $T(1) = 2$, Initial condition
- $T(2) = T(1) + 1 = 2+1 = 3$
- $T(3) = T(2) + 1 = 3+1 = 4$
- $T(4) = T(3) + 1 = 4+1 = 5$

Subsituting down from T(4):

$$\begin{aligned} T(4) &= T(3) + 1 \\ &= [T(2) + 1] + 1 \\ &= [[T(1) + 1] + 1] + 1 \\ &= 2+1+1+1 = 5 \end{aligned}$$

Solving Recurrence Relations

- Given a function defined by a recurrence relation, we want to eliminate recursion from the function definition and obtain its "closed" form.
- Two main techniques are the iteration method and the Master Theorem method.
- Example: Solving the recurrence relation for g(n) using the iteration method:

$g(n) = g(n-1) + 2n - 1 \quad , \quad g(0) = 0$

$g(n) = \underline{g(n-1)} + 2n - 1$

$\quad = \underline{[g(n-2) + 2(n-1) - 1]} + 2n - 1$

(because $g(n-1) = g(n-2) + 2(n-1) -1$)

Solving Recurrence Relations

$= g(n-2) + 2(n-1) + 2n - 2$

$= [g(n-3) + 2(n-2) -1] + 2(n-1) + 2n - 2$ (because $g(n-2) = g(n-3) + 2(n-2) -1$)

$= g(n-3) + 2(n-2) + 2(n-1) + 2n - 3$

...

$= g(n-i) + 2(n-i+1) +...+ 2n - i$

...

$= g(n-n) + 2(n-n+1) +...+ 2n - n$

$= 0 + 2 + 4 +...+ 2n – n$ (because $g(0) = 0$)

$= 2 + 4 +...+ 2n – n$

$= 2*(1+2+3+......+n) -n$

$= 2*n*(n+1)/2 – n$ (using arithmetic progression formula $1+...+n = n(n+1)/2$)

$= n^2$

This is the closed form solution of the initial recurrence relation.

Recurrence Relations

We will also use functional forms of recurrence relations:

Calculus	SE 2228
$x_1 = 1$	$\mathrm{f}(1) = 1$
$x_n = x_{n-1} + 2$	$\mathrm{f}(n) = \mathrm{f}(n-1) + 2$
$x_n = 2x_{n-1} + n$	$\mathrm{f}(n) = 2\,\mathrm{f}(n-1) + n$

Recurrence Relations :Examples

Given the two recurrence relations

$$x_n = x_{n-1} + 2 \qquad x_n = 2x_{n-1} + n$$

and the initial condition $x_1 = 1$, we would like to find explicit formulae for these sequences.

The solutions are:

$$x_n = 2n - 1 \qquad x_n = 2^{n+1} - n - 2$$

respectively

Recurrence relations

- The previous examples using the functional notation are:

$$f(n) = f(n-1) + 2 \qquad\qquad g(n) = 2\,g(n-1) + n$$

With the initial conditions $f(1) = g(1) = 1$, the solutions are:

$$f(n) = 2n - 1 \qquad\qquad g(n) = 2^{n+1} - n - 2$$

Recurrence Relations

A simple solution may be obtained for the first one by substitution:

$f(n) = f(n-1) + 2$, f(1)=1

f(2)= f(1)+2, 3 (= 2*2-1)

f(3)= f(2)+2 5 (=2*3-1)

f(4)=f(3)+2 7

. . .

. . .

→ f(n)= …………(2n-1)

Recurrence : Example

Consider the following recurrence:

$T(n)=\{3T(n-1)$, if $n>0$,

$= 1$, otherwise

Let us solve using substitution.

$T(n) = 3T(n-1)$

$= 3(3T(n-2))$

$= 3^2T(n-2)$

$= 3^3T(n-3)$

...

...

$= 3^nT(n-n)$

$= 3^nT(0)$

$= 3^n$