CS483 Analysis of Algorithms
Lecture 02

Jyh-Ming Lien

June 8, 2017

A Brief History

- In ancient Europe, numbers are represented by Roman numerals, e.g., MDCCCCIIII.
- Decimal system is invented in India around AD 600, e.g., 1904.
- Al Khwarizmi (AD 840), one of the most influential mathematicians in Baghdad, wrote a textbook in Arabic about adding, multiplying, dividing numbers, and extracting square roots and computing π using decimal system.

(image of Al Khwarizmi from http://jeff560.tripod.com/)

A Brief History (Cont.)

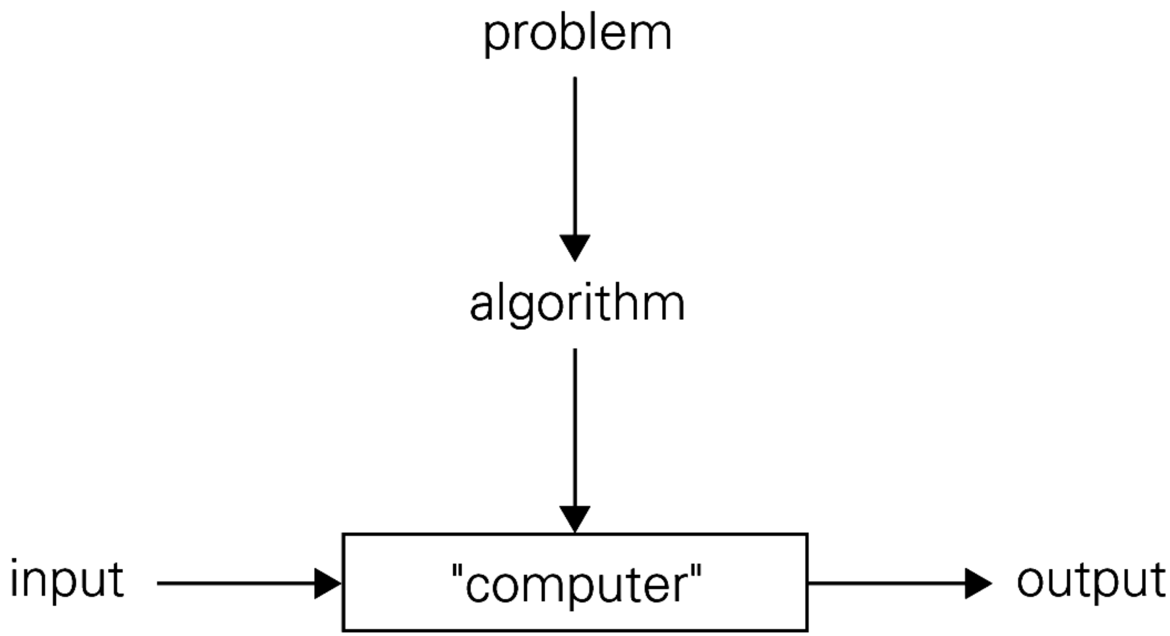
- Many centuries later, decimal system was adopted in Europe, and the procedures in Al Khwarizmi's book were named after him as "Algorithms." One of the most important mathematicians in this process was a man named "Leonard Fibonacci."
- Today, one of his most well known work is *Fibonacci /Fee-boh-NAH-chee/ number* (AD 1202).

(image of Leonardo Fibonacci from http://www.math.ethz.ch/fibonacci)

Design Algorithms

Process of Designing An Algorithm

- **Definition**: “An algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state” - *from wikipedia, the free encyclopedia*



What is an algorithm?

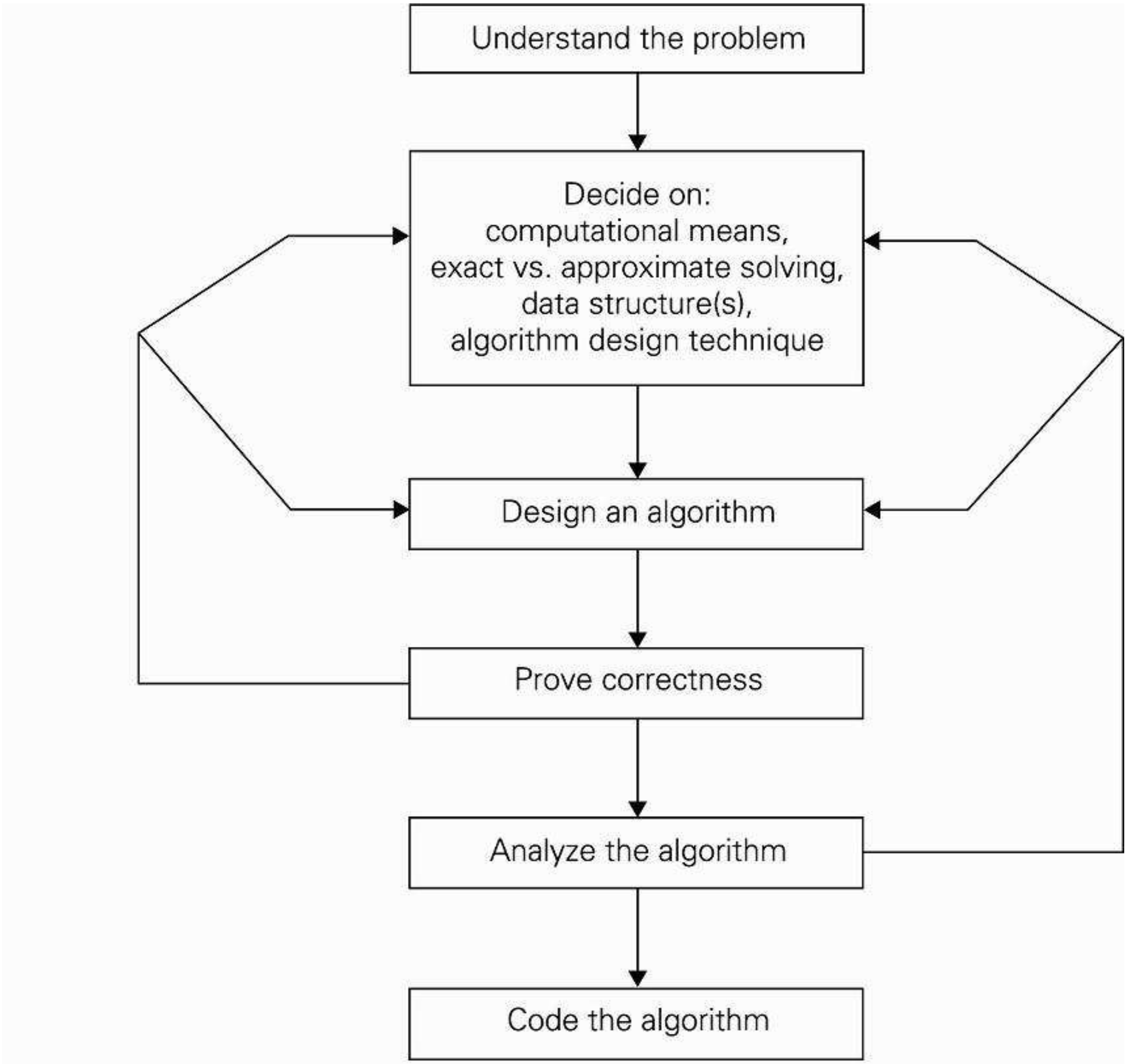
Recipe, process, method, technique, procedure, routine,... with following requirements:

1. Finiteness
   terminates after a finite number of steps
2. Definiteness
   rigorously and unambiguously specified
3. Input
   valid inputs are clearly specified
4. Output
   can be proved to produce the correct output given a valid input
5. Effectiveness
   steps are sufficiently simple and basic

Why study algorithms?

- Theoretical importance
  - the core of computer science (or the core the entire western civilization!)
- Practical importance
  - A practitioners toolkit of known algorithms (i.e., standing on the shoulders of giants)
  - Framework for designing and analyzing algorithms for new problems (i.e, so you know that your problem will terminate before the end of the world)

How to design algorithms?



Analysis of Algorithms

Analysis of Algorithms

- When we design an algorithm, we should ask ourselves:
  1. Is the algorithm correct?
  2. How efficient is the algorithm?
     - Time efficiency
     - Space efficiency
  3. Can we do better?
- Approaches
  1. theoretical analysis
  2. empirical analysis

Empirical analysis of time efficiency

- A typical way to estimate the running time
  - Select a specific (typical) sample of inputs
  - Use wall-clock time (e.g., milliseconds)
    or
    Count actual number of basic operation's executions
  - Analyze the collected data (e.g., plot the data)
- Problems with empirical analysis
  - difficult to decide on how many samples/tests are needed
  - computation time is hardware/environmental dependent
  - implementation dependent

Theoretical analysis of time efficiency

- Provide *machine independent* measurements
- Estimate the bottleneck of the algorithm
- The size of the input increases $\rightarrow$ algorithms run longer $\Rightarrow$. Typically we are interested in how efficiency scales w.r.t. input size
- To measure the running time, we could
  1. count all operations executed.
  2. or determine the number of the **basic operation** as a function of **input size**
- **Basic operation**: the operation that contributes most towards the running time

Theoretical analysis of time efficiency

- We can approximate the run time using the following formula:

$$T(n) \approx c_{op} C(n) \ ,$$

where n is the input size, $C(n)$ is the number of the basic operation for n, and c_{op} is the time needed to execute one single basic operation.

- **Examples**: Given that $C(n) = \frac{1}{2}n(n-1)$, How much time an algorithm will take if the input size n doubled?

- Theoretical analysis focuses on "order of growth" of an algorithm. (Given the input size n)

Orders of Growth

- Some of the commonly seen functions representing the number of the basic operation $C(n) =$

  1. n
  2. n^2
  3. n^3
  4. $\log_{10}(n)$
  5. $n \log_{10}(n)$
  6. $\log_{10}^2(n)$
  7. $\sqrt{n}$
  8. 2^n
  9. $n!$

- Can you order them by their growth rate?

Orders of Growth

- Test functions using some values

n	n^2	n^3	2^n	$n!$
10	10^2	10^3	1024	3.6×10^6
100	10^4	10^6	1.3×10^{30}	9.3×10^{157}
1000	10^6	10^9	1.1×10^{301}	
10000	10^8	$10^1 2$		

n	$\log_{10}(n)$	$n \log_{10}(n)$	$\log_{10}^2(n)$	$\sqrt{n}$
10	1	10	1	3.16
100	2	200	4	10
1000	3	3000	9	31.6
10000	4	40000	16	100

- Now, we can order the functions by their growth rate

Best-, average-, worst-cases

For some algorithms efficiency depends on form of input:

- Worst case: $C_{worst}(n) \rightarrow$ maximum over inputs of size n
- Best case: $C_{best}(n) \rightarrow$ minimum over inputs of size n
- Average case: $C_{avg}(n) \rightarrow$ "average" over inputs of size n

  1. Number of times the basic operation will be executed on typical input
  2. NOT the average of worst and best case
  3. Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs

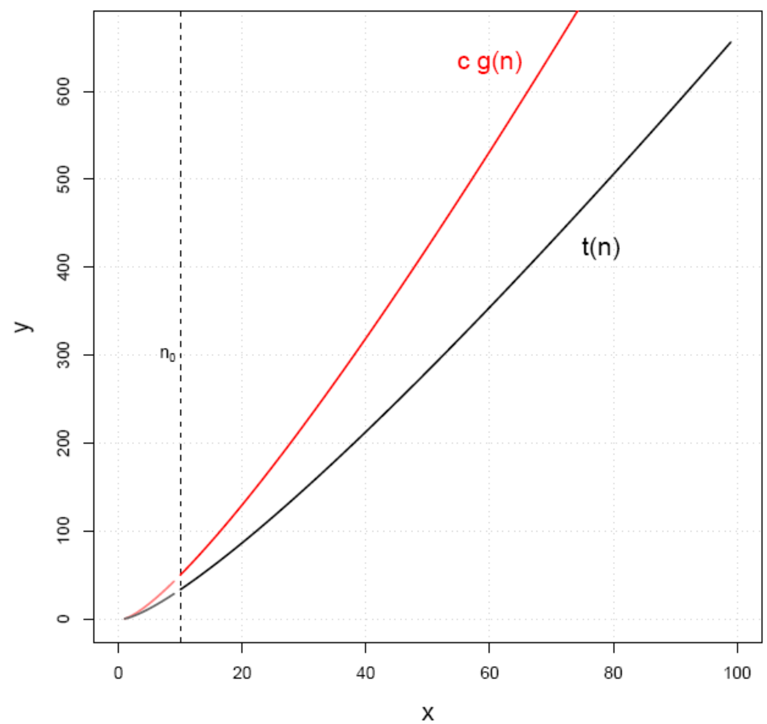
Asymptotic Notation

Asymptotic Notation and Basic Efficiency Classes

- The main goal of algorithm analysis is to estimate **dominate computation steps** $C(n)$ when the **input size** n is large
- Computer scientists classify $C(n)$ into a set of functions to help them concentrate on trend (i.e., order of growth).
- Asymptotic notation has been developed to provide a tool for studying order of growth
  - $O(g(n))$: a set of functions with the same or smaller order of growth as $g(n)$
    - $2n^2 - 5n + 1 \in O(n^2)$
    - $2^n + n^{100} - 2 \in O(n!)$
    - $2n + 6 \notin O(\log n)$
  - $\Omega(g(n))$: a set of functions with the same or larger order of growth as $g(n)$
    - $2n^2 - 5n + 1 \in \Omega(n^2)$
    - $2^n + n^{100} - 2 \notin \Omega(n!)$
    - $2n + 6 \in \Omega(\log n)$
  - $\Theta(g(n))$: a set of functions with the same order of growth as $g(n)$
    - $2n^2 - 5n + 1 \in \Theta(n^2)$
    - $2^n + n^{100} - 2 \notin \Theta(n!)$
    - $2n + 6 \notin \Theta(\log n)$

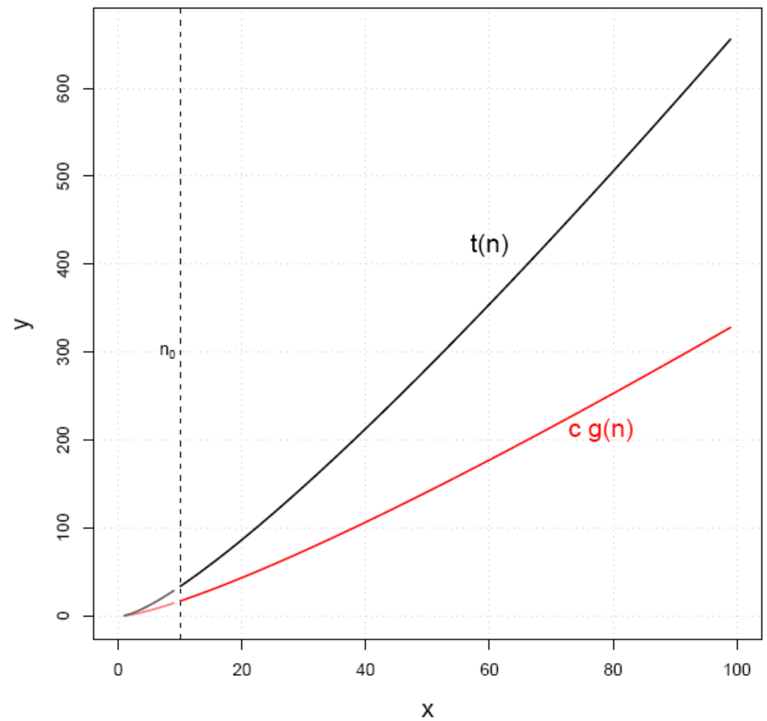
O-notation

- **Definition**: $f(n)$ is in $O(g(n))$ if "order of growth of $f(n)$" $\leq$ "order of growth of $g(n)$" (within constant multiple)
  - there exist positive constant c and non-negative integer n_0 such that $f(n) \leq cg(n)$ for every $n \geq n_0$
- We denote O as an asymptotic **upper** bound



Ω-notation

- **Definition**: $f(n)$ is in $\Omega(g(n))$ if "order of growth of $f(n)$" $\geq$ "order of growth of $g(n)$" (within constant multiple)
  - there exist positive constant c and non-negative integer n_0 such that $f(n) \geq cg(n)$ for every $n \geq n_0$
- We denote Ω as an asymptotic **lower** bound



Θ-notation

- **Definition**: $f(n)$ is in $\Theta(g(n))$ if $f(n)$ is bounded above and below by $g(n)$ (within constant multiple)
  - there exist positive constant c_1 and c_2 and non-negative integer n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for every $n \geq n_0$
- We denote Θ as an asymptotic **tight** bound