CS483 Analysis of Algorithms
Lecture 01*

Jyh-Ming Lien

January 23, 2008

*this lecture note is based on *Algorithms* by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani and *Introduction to the Design and Analysis of Algorithms* by Anany Levitin.

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)

Fibonacci number

Fibonacci's original question

- Fibonacci's original question:
  - Suppose that you are given a newly-born pair of rabbits, one male, one female.
  - Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits.
  - Suppose that our rabbits never die.
  - Suppose that the female always produces one new pair (one male, one female) every month.
- **Question**: How many pairs will there be in one year?

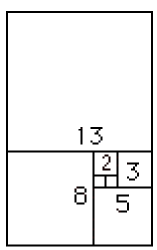
1. Beginning: (1 pair)
2. End of month 1: (1 pair) Rabbits are ready to mate.
3. End of month 2: (_____ pairs)
4. End of month 3: (_____ pairs)
5. End of month 4: (_____ pairs)
6. End of month 5: (_____ pairs)
7. After 12 months, there will be _____ rabits

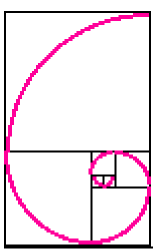
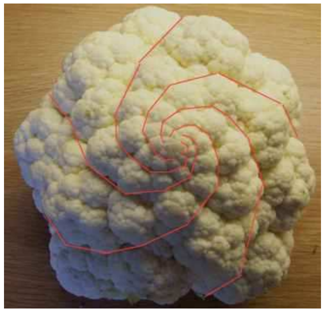
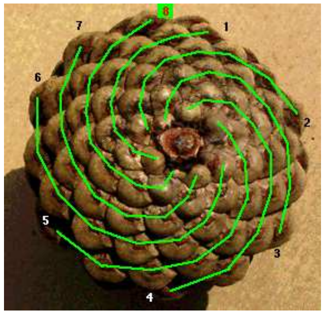
Definition

- Fibonacci numbers $\mathrm{fib}(n)$:

$$\mathrm{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & \text{if } n > 1 \end{cases} \quad (1)$$

- Example: The first 10 Fibonacci numbers are:
$\{0, 1, \_\_\_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_\}$
- Fibonacci numbers have applications in Biology, Visual arts, Music, Simulation, Algorithm analysis and design, etc.




(images from *http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fib.html*)

Our First Algorithm

- Problem: What is $\text{fib}(200)$? What about $\text{fib}(n)$, where n is any positive integer?

Algorithm 0.1: $\text{fib}(n)$

- Questions that we should ask ourselves.

1. Is the algorithm correct?
2. What is the running time of our algorithm?
3. Can we do better?

Analyze Our First Algorithm

- Is the algorithm correct?
  - Yes, we simply follow the definition of Fibonacci numbers
- How fast is the algorithm?
  - If we let the run time of $\mathrm{fib}(n)$ be $T(n)$, then we can formulate

$$T(n) = T(n-1) + T(n-2) + 3 \approx 1.6^n$$

  - $T(200) \geq 2^{139}$
  - The world fastest computer BlueGene/L, which can run 2^{48} instructions per second, will take 2^{91} seconds to compute. (2^{91} seconds = 7.85×10^{10} billion years, Sun turns into a red giant star in 4 to 5 billion years)
  - Can Moose's law, which predicts that CPU get 1.6 times faster each year, solve our problem?
  - No, because the time needed to compute $\mathrm{fib}(n)$ also have the same "growth" rate
    - if we can compute $\mathrm{fib}(100)$ in exactly a year,
    - then in the next year, we will still spend a year to compute $\mathrm{fib}(101)$
    - if we want to compute $\mathrm{fib}(200)$ within a year, we need to wait for 100 years.

Improve Our First Algorithm

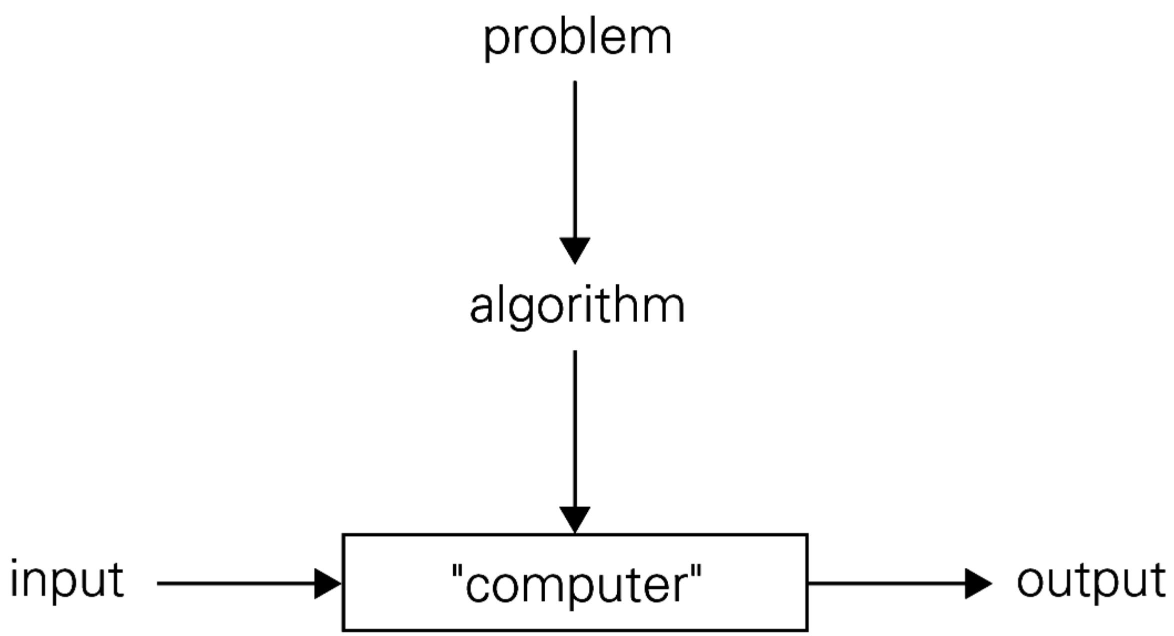
- Can we do better?
- Yes, because many computations in the previous algorithm are repeated.

Algorithm 0.2: $\mathrm{fib}(n)$

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?

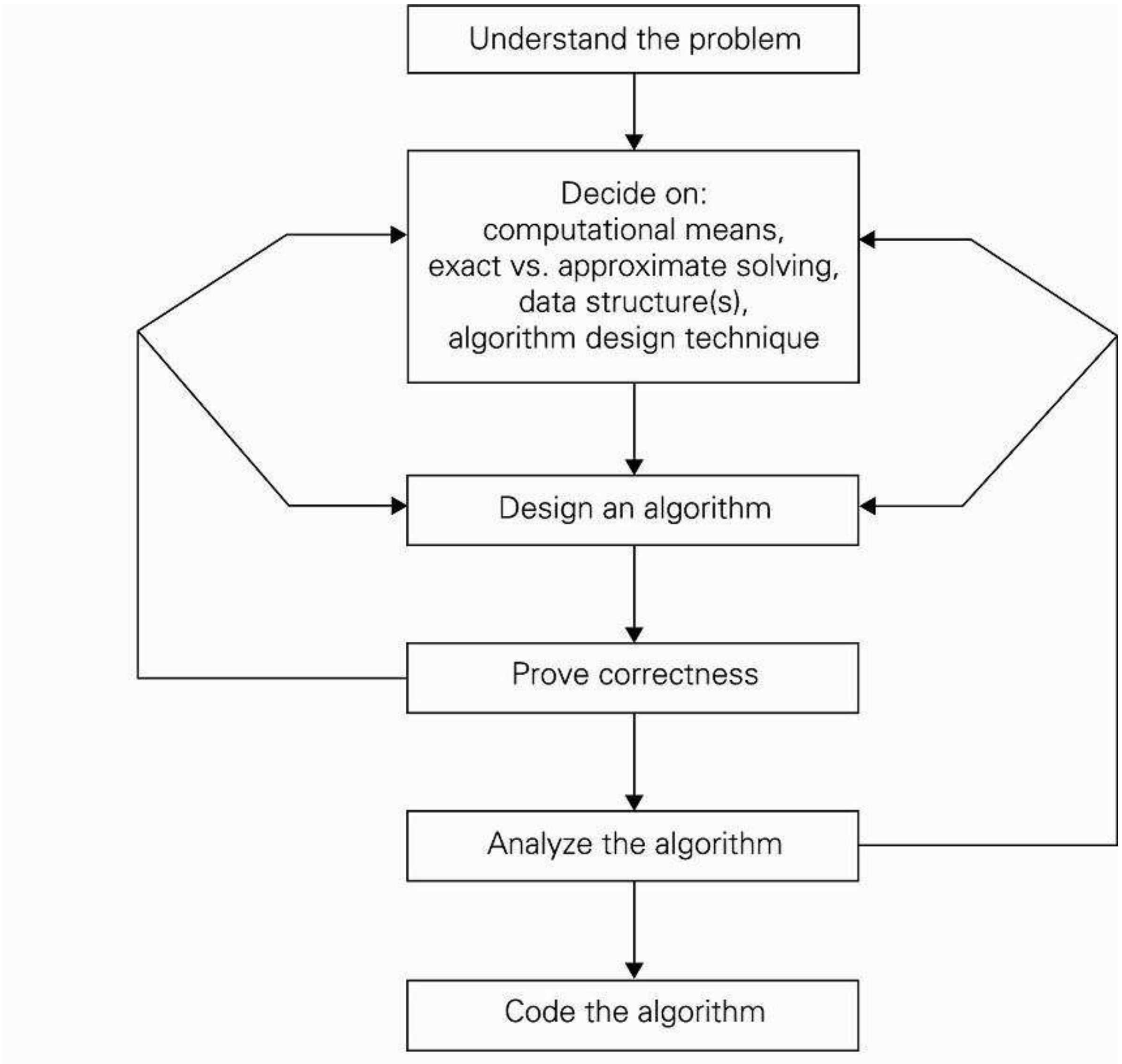
Recipe, process, method, technique, procedure, routine,... with following requirements:

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

- 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

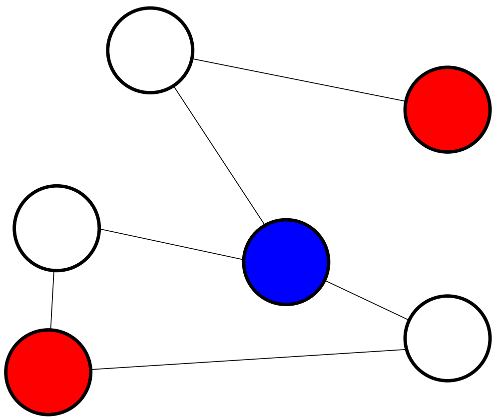
- Examples:

1. sort a list of integers $\{a_1, a_2, \cdots, a_n\}$

2. $$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mk} \end{bmatrix}$$

3. $prime(n)$

4. Graph 3-coloring



Input Size:

1. ____________
2. ____________
3. ____________
4. ____________

Basic operations:

1. ____________
2. ____________
3. ____________
4. ____________

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	$\log_{10}(n)$	$n \log_{10}(n)$	$\log^2_{10}(n)$	$\sqrt{n}$
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

Orders of Growth

- plot the functions (e.g., use matlab or gnuplot)

- Basic efficiency classes

n	n^2	n^3	2^n	$n!$
linear	quadratic	cubic	exponential	factorial

c	$\log_{10}(n)$	$n\log_{10}(n)$	$\sqrt{n}$
constant	logarithmic	n-log-n	square root

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

Example 1: Sequential Search

- Find the value K in a given array $A[1 \cdots n]$

Algorithm 0.3: SEARCH($A[1..n], K$)

for $i \leftarrow [1 \cdots n]$
  do $\begin{cases} \textbf{if } A[i] = K \\ \quad \textbf{then return } (i) \end{cases}$
return (-1)

- Input size
- Worst case (worst case analysis provides an upper bound):

1. When does the worst case happen?

2. What is $C_{worst}(n)$?

Example 1: Sequential Search

- Best case:
  1. When does the best case happen?
  2. What is $C_{best}(n)$?

- Average case:
  1. Average case asks a useful question: what kind of running time to we expect to get when we don't know or know only little about the data?
     - suppose that the probability of $K \in A$ is p
     - suppose that the probability of $K = A[i]$ equals that of $K = A[j]$
  2. When does the best case happen?
  3. What is $C_{best}(n)$?

Example 2: Greatest Common Divisor

Algorithm 0.4: $\mathrm{gcd}(a, b)$

for $i = \{\min(a, b), \cdots, 1\}$

do $\begin{cases} \textbf{if } a\%i = 0 \text{ and } b\%i = 0 \\ \quad \textbf{then return } (i) \end{cases}$

- ☐ Input size=
- ☐ Worst case (worst case analysis provides an upper bound):

Example 2: Greatest Common Divisor

- Best case:
  1. When does the best case happen?
  2. What is $C_{best}(n)$?
- Average case:
  1. Assumptions:
     - Assume that a and b are two randomly chosen integers
     - Assume that all integers have the same probability of being chosen
     - **hint**: The probability that an integer d is a and b's greatest common divisor is $P_{a,b}(d) = \frac{6}{\pi^2 d^2}$
  2. When does the best case happen?
  3. What is $C_{best}(n)$?

Asymptotic Notation

Asymptotic Notation and Basic Efficiency Classes

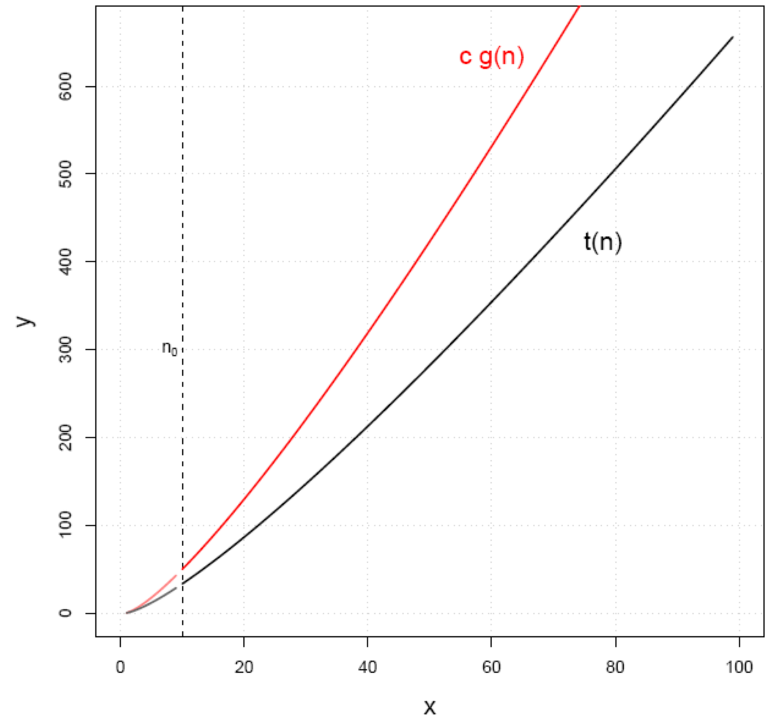
- The main goal of algorithm analysis is to estimate **dominate computation steps** $C(n)$ when the **input size** n is large
- Computer scientists classify $C(n)$ into a set of functions to help them concentrate on trend (i.e., order of growth).
- Asymptotic notation has been developed to provide a 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 $n \geq n_0$
- **Examples**:
  - $10n \in O(n^2)$
    - why?
  - $5n + 20 \in O(n)$
    - why?
  - $2n + 6 \notin O(\log n)$
    - why?

O-notation

- We denote O as an asymptotic **upper** bound



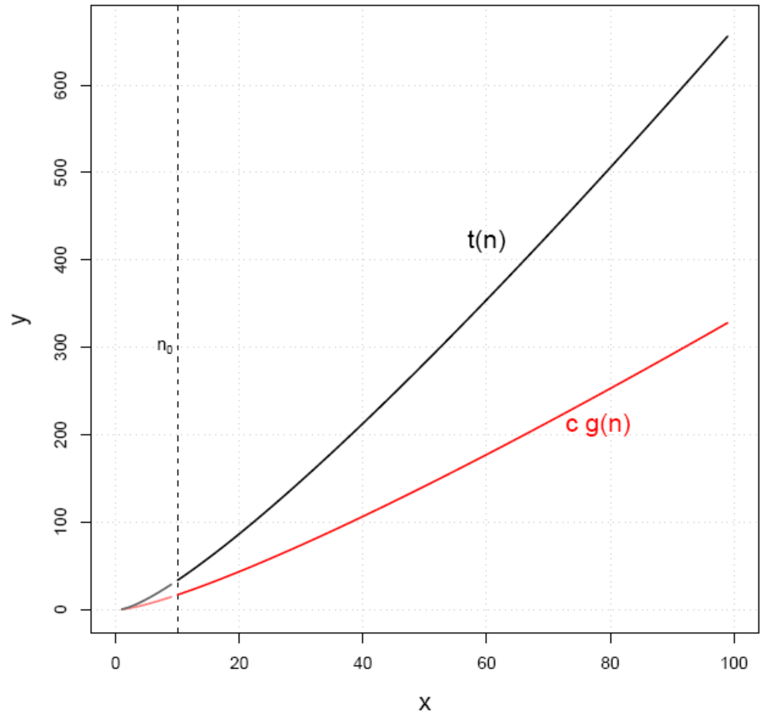
- Try the following commands in **gnuplot**
  – plot $[0 : 20]\ 10 * x, x * x$
  – plot $[0 : 5]\ 5 * x + 20, 10 * x$
  – plot $[0 : 400]\ 2 * x + 6, 100 * log(x)$

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 $n \geq n_0$
- **Examples**:
  - $\frac{n^3}{5} \in \Omega(n^2)$
    - why?
  - $2n - 51 \in \Omega(n)$
    - why?

Ω-notation

- We denote Ω as an asymptotic **lower** bound



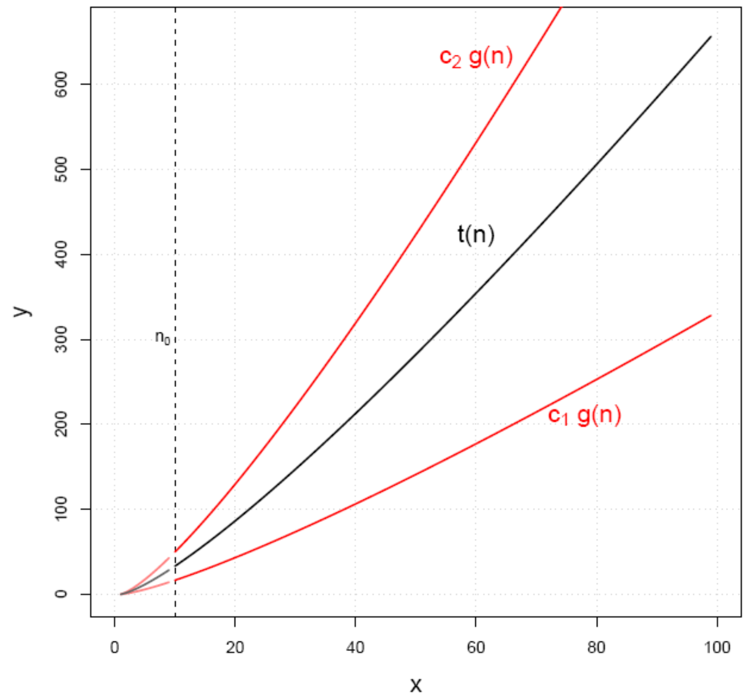
- Try the following commands in **gnuplot**
  - plot $[0:10]\ (x*x*x)/5, x*x$
  - plot $[0:100]\ 2*x-51, x$

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 $n \geq n_0$
- **Examples**:
  - $\frac{1}{2}n(n-1) \in \Theta(n^2)$
    - why?
  - $2n - 51 \in \Theta(n)$
    - why?

Θ-notation

- We denote Θ as an asymptotic **tight** bound



- Try the following commands in **gnuplot**
  - plot $[0 : 10]\ (x * x - x)/2, (x * x)/4, x * x$
  - plot $[0 : 200]\ 2 * x - 51, x, 2 * x$

Useful Property

1. $f(n) \in O(f(n))$

   Proof. □

2. $f(n) \in O(g(n))$ if and only if $g(n) \in \Omega(f(n))$

   Proof. □

3. $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$

   Proof. □

4. $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then
   $f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$

   Proof. □

Comparing Orders of Growth

1. Comparing Orders of Growth

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & t(n) \text{ has a smaller order of growth than } g(n) \\ c > 0 & t(n) \text{ has the same order of growth as } g(n) \\ \infty & t(n) \text{ has a larger order of growth than } g(n) \end{cases}$$

2. Example: Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2

3. Example: Compare the orders of growth of $\log n$ and $\sqrt{n}$

4. Example: Compare the orders of growth of $n!$ and 2^n

Some tools for computing limits

- L'Hôpital's rule

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

- Stirling's formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Orders of growth of some important functions

1. All logarithmic functions $\log_a n$ belong to the same class $\Theta(\log n)$ no matter what the logarithms base $a > 1$ is

   Proof. □

2. All polynomials of the same degree k belong to the same class: $a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 \in \Theta(n^k)$

   Proof. □

3. Exponential functions a^n have different orders of growth for different a's, i.e., $2^n \notin \Theta(3^n)$

   Proof. □

4. order $\log n <$ order $n^{a>0} <$ order $a^n <$ order $n! <$ order n^n

Syllabus

Grading and Important Dates

- **Webpage**: http://cs.gmu.edu/∼jmlien/teaching/08_spring_cs483/
- **TA**: TBA
- **Required Textbook**: Algorithms, by Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani, McGraw-Hill, 2006, ISBN 0073523402.

- Grading
  1. Quizzes and CS Culture assignments 15%
  2. Assignments 25%
  3. Midterm Exam 25%
  4. Final Exam 35%

- Final grade:
  - **A** (≥ 90)
  - **B** (≥ 80)
  - **C** (≥ 70)
  - **D** (≥ 60)
  - **F** (< 60)
- Important Dates.
  - Spring Break (March 10 – 16)
  - Midterm Exam (March 19)
  - Final Exam (May 07)

Policies

- **Quizzes** are mainly for keeping you coming to the class. The quiz will be a closed book exam. You can also have up to **two** opportunities of making up your missed/failed quizzes by turning in two CS culture assignments.
- **CS culture assignment** is a one-page written summary (form available online) of a talk from a CS seminar (see http://cs.gmu.edu/events/) that you attend during the Spring'08 semester.
- **Assignments** must be completed by the stated due date and time. Your assignment score will be halved every extra day after the due date.
- **Exams**. You will be allowed to have one page (letter size) of notes for the midterm and two pages (one sheet) for the final. No copying of anything from the textbook or another person is allowed. You can write some things verbatim. You can also write your notes on the computer and print them. The notes sheet will be handed in with the exam.

Summary

Summary

- ☐ Two important men in algorithms: Al Khwarizmi & Leo Fibonacci
- ☐ Fibonacci number
- ☐ General ideas of design of algorithms
- ☐ Analysis of algorithms: experimental and theoretical
- ☐ Asymptotic notations: O (upper bound), Θ (lower bound), Ω (tight bound)

Please read Chapter 0 Prologue in the textbook.

Assignment

- Chapter 0, Exercise 1
- Chapter 0, Exercise 2
- Due Jan 30 2008, before the class.