CS483 Design and Analysis of Algorithms

Lecture 1 Introduction and Prologue

Instructor: Fei Li

lifei@cs.gmu.edu with subject: CS483

Office hours:
Room 5326, Engineering Building, Thursday 4:30pm - 6:30pm or by appointments

Course web-site:
http://www.cs.gmu.edu/∼lifei/teaching/cs483_fall09/

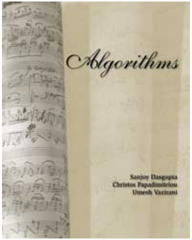
Figures unclaimed are from books "Algorithms" and "Introduction to Algorithms"

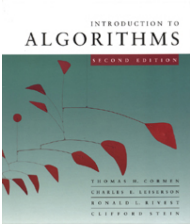
About this Course

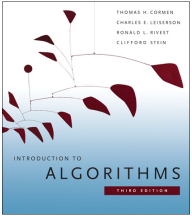
- About this Course
  (From 2007-2008 University Catalog) Analyze computational resources for important problem types by alternative algorithms and their associated data structures, using mathematically rigorous techniques. Specific algorithms analyzed and improved
- Prerequisites
  CS310 (Data Structures) and CS330 (Formal Methods and Models) and MATH125 (Discrete Mathematics I)
- Weekly Schedule
  - When: Monday & Wednesday 3:00pm - 4:15pm
  - Where: Innovation Hall 134

Required Textbooks

1. **Algorithms** by Sanjoy Dasgupta (UCSD), Christos Papadimitriou and Umesh Vazirani (UC-Berkeley).
   A draft of the book can be found at
   `http://www.cs.berkeley.edu/ vazirani/algorithms.html`
2. **Introduction to Algorithms** by Thomas H. Cormen (Dartmouth), Charles E. Leiserson and Ronald L. Rivest (MIT), Clifford Stein (Columbia), 2nd Edition or 3rd Edition (Highly recommended)







How to Reach Me and the TA

1. Instructor: Fei Li
2. Email: lifei@cs.gmu.edu
3. Office: Room 5326, Engineering Building
4. Office hours: Thursday 4:30pm - 6:30pm or by appointments

1. Teaching Assistant: Chen Liang
2. Email: cliang1@gmu.edu
3. Office: Room 4456, Engineering Building
4. Office hours: Tuesday 4:00pm - 6:00pm

Making the Grade

1. Your grade will be determined 45% by the take-home assignments, 20% by a midterm exam, and 35% by a final exam
2. Tentatively, there will be 9 assignments; each assignment deserves 5 points
3. Hand in hard copies of assignments in class. No grace days for late assignment. All course work is to be done independently. Plagiarizing the homework will be penalized by maximum negative credit and cheating on the exam will earn you an F in the course
4. Tentative grading system:
   A (≥ 85), B ($\in [70,\ 85)$), C ($\in [60,\ 70)$), D ($\in [50,\ 60)$), and F (< 50)

Any Questions?

Chapter 0 of DPV — Prologue

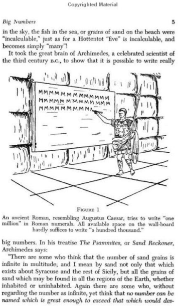
1. What are algorithms?
2. What are asymptotic notations?

Algorithms

Computers + Networks = Hardware (microelectronics) + Software (algorithms)

- Typography versus algorithms
  `http://en.wikipedia.org/wiki/Typography`: "Typography with moveable type was separately invented in 11th-century China, and modular moveable metal type began in 13th-century China and Korea, was developed again in mid-15th century Europe with the development of specialized techniques for casting and combining cheap copies of letter punches in the vast quantities required to print multiple copies of texts."

Decimal Systems and Algorithms


Copyrighted Material

Big Numbers 5

in the sky, the fish in the sea, or grains of sand on the beach were "incalculable," just as for a Hottentot "five" is incalculable, and becomes simply "many"!

It took the great brain of Archimedes, a celebrated scientist of the third century B.C., to show that it is possible to write really

FIGURE 1

An ancient Roman, resembling Augustus Caesar, tries to write "one million" in Roman numerals. All available space on the wall-board hardly suffices to write "a hundred thousand."

big numbers. In his treatise *The Psammites*, or *Sand Reckoner*, Archimedes says:

"There are some who think that the number of sand grains is infinite in multitude; and I mean by sand not only that which exists about Syracuse and the rest of Sicily, but all the grains of sand which may be found in all the regions of the Earth, whether inhabited or uninhabited. Again there are some who, without regarding the number as infinite, yet think that *no number can be named which is great enough to exceed that which would des-*

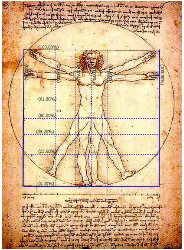
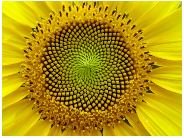
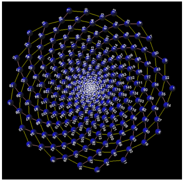
from "One Two Three . . . Infinity: Facts and Speculations of Science" by George Gamow, Dover, 1988

- Decimal system is invented in India around AD 600. With only 10 symbols, arithmetic could be done efficiently by following elementary steps
- Al Khwarizmi (780 - 850) wrote a book on basic methods for adding, multiplying, and dividing numbers, even extracting square roots and calculating digits of π. The term Algorithm derives from his name and is coined after him and the decimal system

Algorithms and Their Asymptotic Notations

1. Algorithm example – Enter Fibonacci
2. Running time — asymptotic notation

Fibonacci Series and Numbers

`http://www.rosicrucian.org` & `http://thelifeportfolio.wordpress.com`

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ \ldots,$$

The Fibonacci numbers F_n is generated by

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & \text{if } n > 1, \\ 1, & \text{if } n = 1, \\ 0, & \text{if } n = 0. \end{cases}$$

The golden ratio $\phi = \frac{1+\sqrt{5}}{2} = 1 + \frac{1}{\phi} \approx 1.618 = \lim_{n\to\infty}\frac{F_{n+1}}{F_n}$

Calculate F_n — First Approach

From the recursive definition

```
function fib1(n)
{
    if (n = 0)
        return 0;
    if (n = 1)
        return 1;

    return fib1(n - 1) + fib1(n - 2);
}
```

Calculate F_n — First Approach

From the recursive definition

```
function fib1(n)
{
    if (n = 0)
        return 0;
    if (n = 1)
        return 1;

    return fib1(n - 1) + fib1(n - 2);
}
```

- Correctness

Calculate F_n — First Approach

From the recursive definition

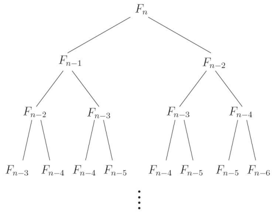
```
function fib1(n)
{
    if (n = 0)
        return 0;
    if (n = 1)
        return 1;

    return fib1(n - 1) + fib1(n - 2);
}
```

- Correctness
- Running time $T(n) = T(n-1) + T(n-2) + 3,\ n > 1$

$$T(200) \geq F_{200} \geq 2^{138}$$

Calculate F_n — Second Approach

Figure 0.1 The proliferation of recursive calls in fib1.

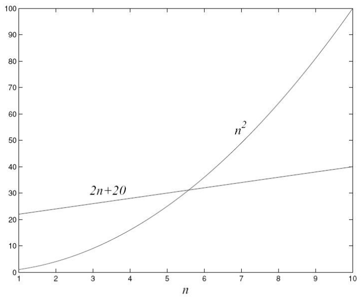



```
function fib2(n)
{
    if (n = 0)
        return 0;

    create an array f[0, ..., n];

    f[0] = 0; f[1] = 1;

    for (i = 2, ..., n)
        f[i] = f[i - 1] + f[i - 2];

    return f[n];
}
```

fib2(n) is linear in n.

Big-$\mathcal{O}$ Notation

Figure 0.2 Which running time is better?



Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. We say $f(n) = O(g)$ (which means that "f grows no faster than g") if *there is a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$*

$$
\begin{aligned}
f &= O(g) \leftrightarrow f(n) \leq c \cdot g(n) \leftrightarrow g = \Omega(f) \\
f &= \Theta(g) \leftrightarrow f = O(g) \;\&\; f = \Omega(g)
\end{aligned}
$$

Exercises

$$\begin{aligned} 14 \cdot n^2 \quad &? \quad n^2 \\ n^a \quad &? \quad n^b, \quad a > b \\ 3^n \quad &? \quad n^5 \\ n \quad &? \quad (\log n)^3 \\ n! \quad &? \quad 2^n \end{aligned}$$

Establish Order of Growth

- **L'Hopital's rule**
  If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ and the derivatives f' and g' exist, then

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

- **Stirling's formula**

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

  where e is the natural logarithm, $e \approx 2.718$. $\pi \approx 3.1415$.

$$\sqrt{2\pi n} \cdot (\frac{n}{e})^n \leq n! \leq \sqrt{2\pi n} \cdot (\frac{n}{e})^{n+\frac{1}{12n}}$$

Some Observations

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

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 \in \Theta(n^k)$$

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

$$\Theta(\log n) < \Theta(n^a) < \Theta(a^n) < \Theta(n!) < \Theta(n^n), \quad \text{where } a > 1$$

Why Does it Matter?

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

from Kleiberg and Tardos, "Algorithms Design"

What Are We Going to Learn from this Course?

Chapter 7:

Goal. Given n men and n women, find a "suitable" matching.

- Participants rate members of opposite sex.
- Each man lists women in order of preference from best to worst.
- Each woman lists men in order of preference from best to worst.

	1st (favorite)	2nd	3rd (least favorite)
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

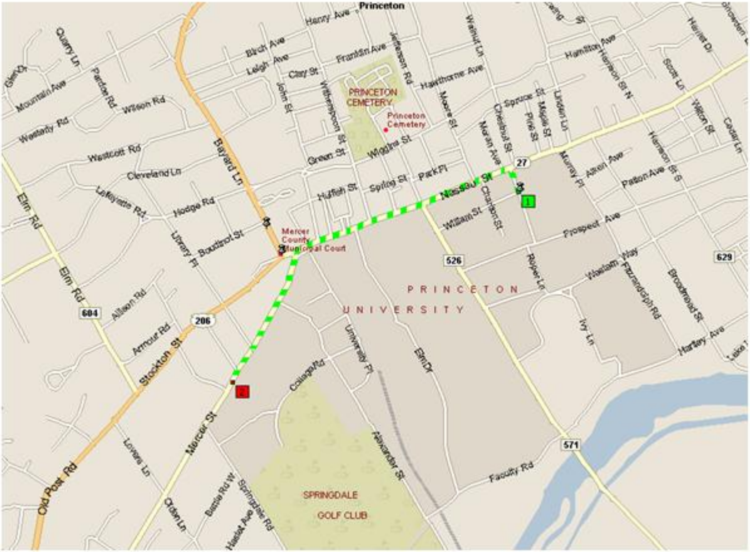
	1st (favorite)	2nd	3rd (least favorite)
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

from Wayne's slides on "Algorithm Design"

What Are We Going to Learn in this Course?

Chapter 4:

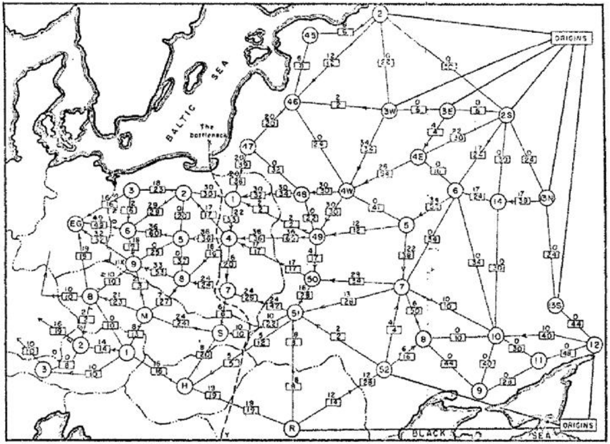


shortest path from Princeton CS department to Einstein's house

What Are We Going to Learn in this Course?

Chapter 7:

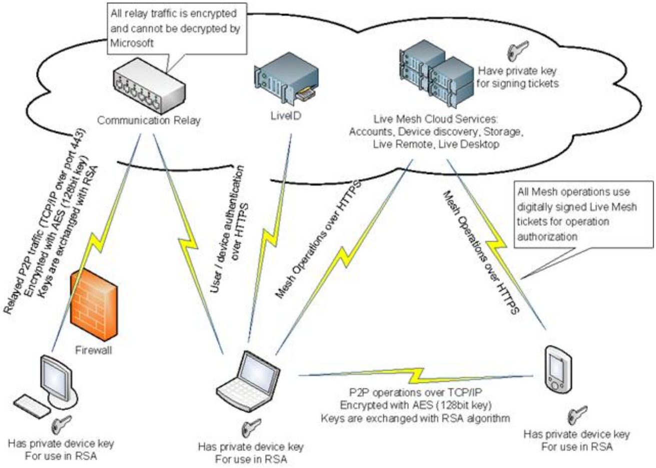
Soviet Rail Network, 1955



Reference: *On the history of the transportation and maximum flow problems.* Alexander Schrijver in Math Programming, 91: 3, 2002.

What Are We Going to Learn in this Course?

Chapter 1:



Course Outcomes

1. An understanding of classical problems in Computer Science
2. An understanding of classical algorithm design and analysis strategies
3. An ability to analyze the computability of a problem
4. Be able to design and analyze new algorithms to solve a computational problem
5. An ability to reason algorithmically

"Size" of Input — Example 1

- Goal: Determine whether all the elements in a given array are distinct.
- Input: An array $A[0, \ldots, n-1]$.
- Output: Returns "true" if all the elements in A are distinct and "false" otherwise.

```
function unique-element(A[0, ..., n - 1])
{
    for (i = 0; i < n - 1; i++)
        for (j = i + 1; j < n; j++)
            if (A[i] == A[j])
                return(false);

    return(true);
}
```

“Size” of Input — Example 2

- Goal: Count binary bits.
- Input: A positive decimal integer *n*.
- Output: The number of binary digits in *n*s binary representation.

```
function count-binary-bit(n)

    counter = 0;

    while (n > 1)

        counter = counter + 1;

        n = ⌊ n / 2 ⌋ ;


    return(counter);
```