CS 222, Autumn 2017

Algorithm Design and Analysis

Li Jiang

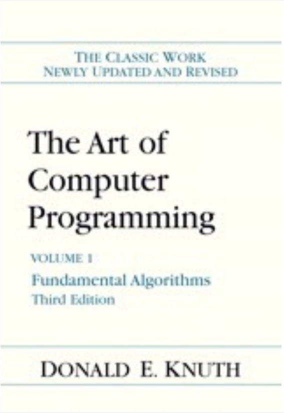
Department of Computer Science and Engineering

Algorithm definitions

" A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation. " — *webster.com*

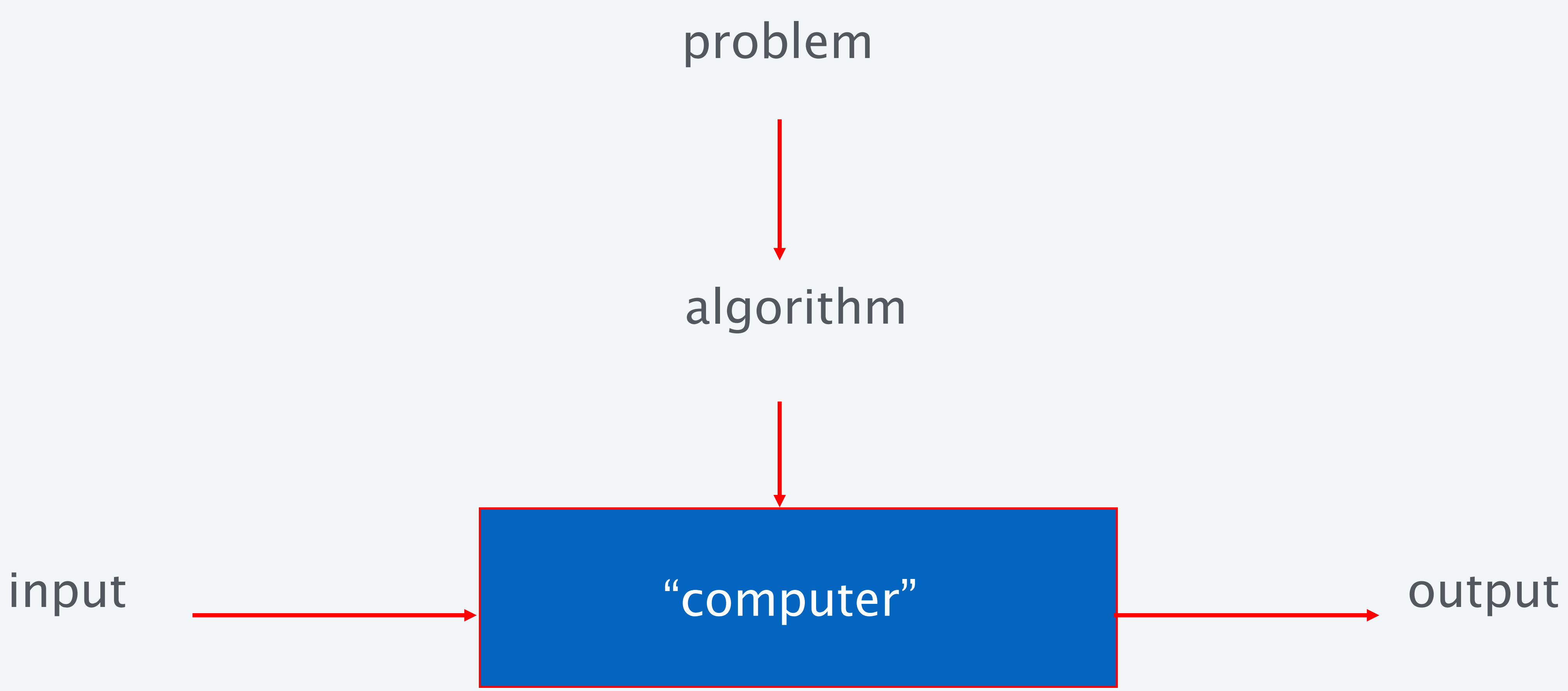
" An algorithm is a finite, definite, effective procedure, with some input and some output. "

— *Donald Knuth*



What is an algorithm?

An ***algorithm*** is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n

Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Euclid's algorithm is based on repeated application of equality

$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Two descriptions of Euclid's algorithm

Step 1 If $n = 0$, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value of the remainder to r

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

```
while n ≠ 0 do
        r ← m mod n
    m← n
    n ← r
return m
```

Consecutive integer checking algorithm

Step 1 Assign the value of min{m,n} to t

Step 2 Divide m by t. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide n by t. If the remainder is 0, return t and stop; otherwise, go to Step 4

Step 4 Decrease t by 1 and go to Step 2

Middle-school procedure

Step 1 Find the prime factorization of m

Step 2 Find the prime factorization of n

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors
and return it as gcd(m,n)

Is this an algorithm?

Sieve of Eratosthenes

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ to n do $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do

if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

$j \leftarrow p * p$

while $j \leq n$ do

$A[j] \leftarrow 0$ //mark element as eliminated

$j \leftarrow j + p$

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

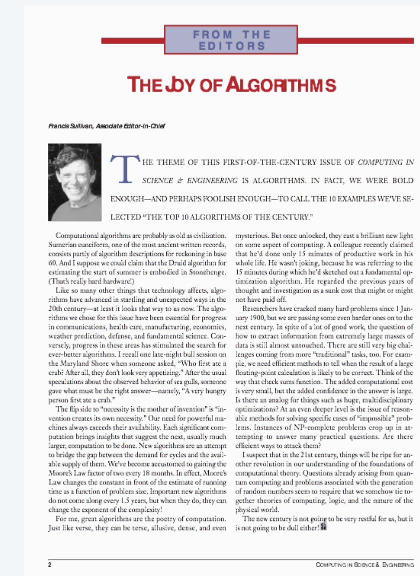
Algorithm etymology

Etymology. [Knuth, TAOCP]

- *Algorism* = process of doing arithmetic using Arabic numerals.
- A misperception: *algiros* [painful] + *arithmos* [number].
- True origin: Abu 'Abd Allah Muhammad ibn Musa al-Khwarizm was a famous 9th century Persian textbook author who wrote *Kitāb al-jabr wa'l-muqābala*, which evolved into today's high school algebra text.


Why study algorithms?

" *For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.* " — *Francis Sullivan*


FROM THE EDITORS

THE JOY OF ALGORITHMS

" *An algorithm must be seen to be believed.* " — *Donald Knuth*

Why study algorithms?

Theoretical importance

- the core of computer science

Practical importance

- A practitioner's toolkit of known algorithms
- Framework for designing and analyzing algorithms for new problems

Algorithms vs. CS 222

Implementation and consumption of classic algorithms.

- Stacks and queues.
- Sorting.
- Searching.
- Graph algorithms.
- String processing.

```
private static void sort(double[] a, int lo, int
hi)
{
   if (hi <= lo) return;
   int lt = lo, gt = hi;
   int i = lo;
   while (i <= gt)
   {
      if      (a[i] < a[lo]) exch(a, lt++, i++);
      else if (a[i] > a[lo]) exch(a, i, gt--);
      else                   i++;
   }

   sort(a, lo, lt - 1);
   sort(a, gt + 1, hi);
}
```

Emphasizes critical thinking, problem-solving, and code.

Design and analysis of algorithms

Design and analysis of algorithms.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomized algorithms.
- Intractability.
- Coping with intractability.
- Data structures.

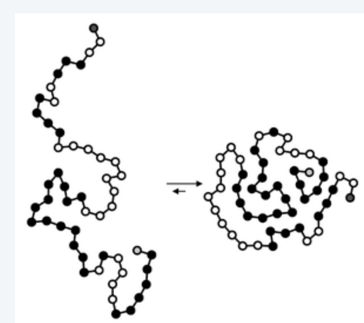
$$
\begin{aligned}
\sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{2}{j-i+1} &= 2 \sum_{i=1}^{N} \sum_{j=2}^{N-i+1} \frac{1}{j} \\
&\leq 2N \sum_{j=1}^{N} \frac{1}{j} \\
&\sim 2N \int_{x=1}^{N} \frac{1}{x}\, dx \\
&= 2N \ln N
\end{aligned}
$$

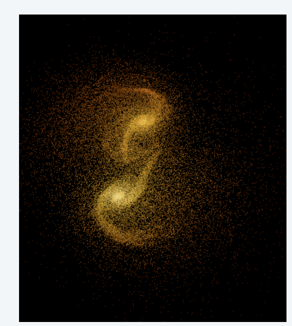
Emphasizes critical thinking, problem-solving, and rigorous analysis.

Why study algorithms?

Internet. Web search, packet routing, distributed file sharing, ...
Biology. Human genome project, protein folding, …
Computers. Circuit layout, databases, caching, networking, compilers, …
Computer graphics. Movies, video games, virtual reality, …
Security. Cell phones, e-commerce, voting machines, …
Multimedia. MP3, JPG, DivX, HDTV, face recognition, …
Social networks. Recommendations, news feeds, advertisements, …
Physics. N-body simulation, particle collision simulation, …

⋮



We emphasize algorithms and techniques that are useful in practice.

Applications

Wide range of applications.

- Caching.
- Compilers.
- Databases.
- Scheduling.
- Networking.
- Data analysis.
- Signal processing.
- Computer graphics.
- Operations research.
- Scientific computing.
- Artificial intelligence.
- Computational biology.
- . . .

We emphasize algorithms and techniques that are useful in practice.

Administrative stuff

Lectures. [Li Jiang, jiangli@cs.sjtu.edu.cn]

- Tue, 10:00-11:40; Thu, 16:00-17:40, 东上院111
- Attendance is required.
- No electronic devices except to aid in learning.
- Office Hour: Tue, afternoon

← viewing lecture slides
taking notes

Precept. [Zishan Jiang, jiang-zs@sjtu.edu.cn]

- Office hour/tutorial: TBA.
- Preceptor works out problems.
- Attendance is recommended.

← precept begins
this week

Prerequisites. Programming Language, Data Structure, Discrete M.

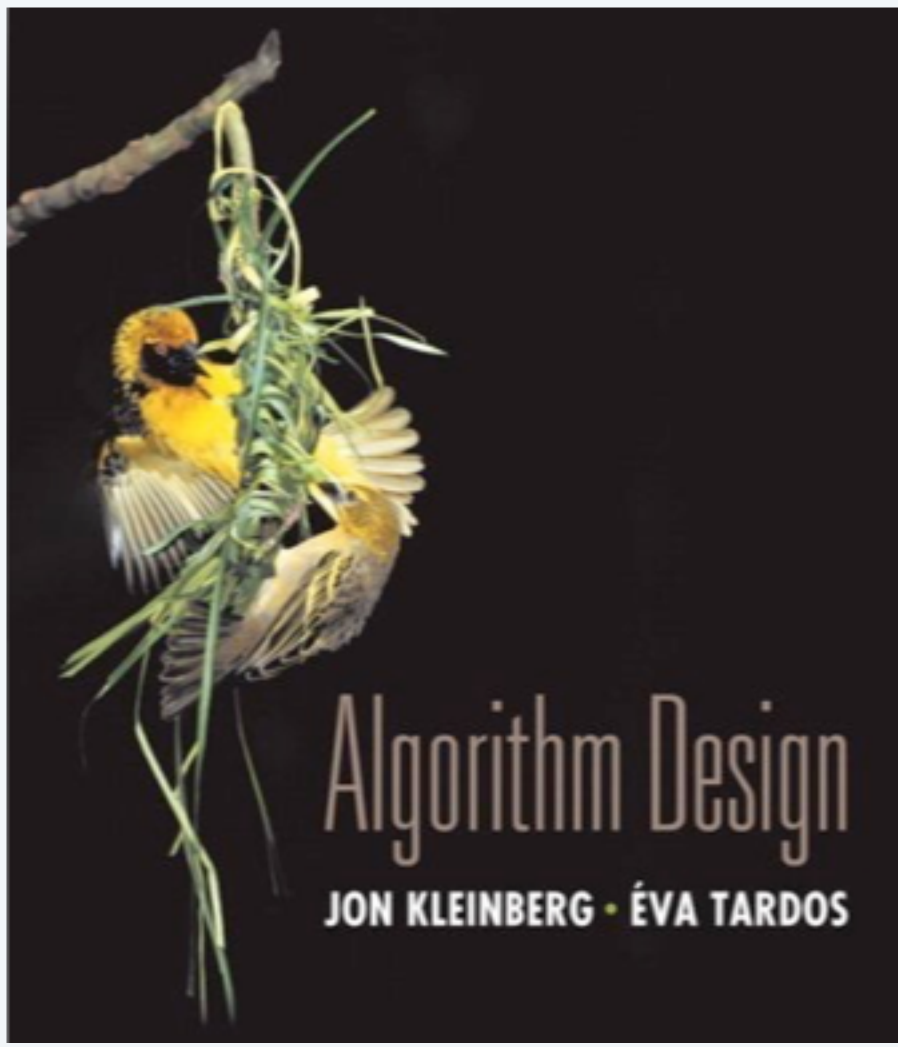
Course website

- Syllabus.
- Office hours.
- Problem sets.
- Lecture slides.
- Electronic submission.
- ...

TBA

Textbook

Required reading. *Algorithm Design* by Jon Kleinberg and Éva Tardos.
Addison-Wesley 2005, ISBN 978-0321295354.



Grades

Problem sets.

- "Weekly" problem sets, due via electronic submission. ← problem set is due In class Monday
- Graded for correctness, clarity, conciseness, rigor, and efficiency.
- Use LaTeX template for writing solutions.

Course grades.

- Primarily based on problem sets.
- Staff discretion used to adjust borderline cases.
- “Weekly" problem sets, due Monday 10am in class. 20%
- Class participation, staff discretion for borderline cases. 10%
- Final exams. 50%
- Course project.20%

Collaboration

Collaboration policy. [see syllabus for full details; ask if unsure]

- Course materials (textbook, slides, handouts) are always permitted.
- No external resources, e.g., can't Google for solutions.

"Collaboration permitted" problem sets.

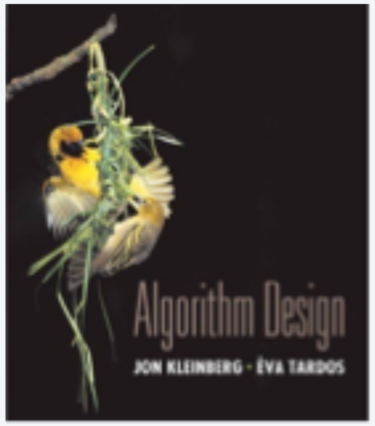
- You may discuss ideas with classmates.
- You must write up solutions on your own, in your own words.

"No collaboration" problem sets.

- You may discuss ideas with course staff.

Where to get help?

Textbook. Read the textbook—it's good!



Piazza. Online discussion forum.

- Low latency, low bandwidth.
- Mark as private any solution-revealing questions.

piazza

https://piazza.com/class/j7i92eh9s1q5n0

Office hours.

- High bandwidth, high latency.
- Tue, 13:00-16:00.

www.cs.princeton.edu/courses/archive/spring13/cos423

Reference

Algorithm:

- T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, MIT Press, 2009
- S. Dasgupta, C. Papadimitriou, U. Vazirani, Algorithm, McGraw-Hill, 2007.
- J. Kleinberg, and E. Tardos, Algorithm Design, Pearson-Addison Wesley, 2005.
- Henming Zou, The Way of Algorithms, China Machine Press, 2010.

Computational Complexity:

- Theory of Computational Complexity, by Ding-Zhu Du, and Ker-I Ko, published by John Wiley & Sons, Inc., 2000.
- Computational Complexity: A Modern Approach, by Sanjeev Arora and Boaz Barak, Cambridge University Press, 2006.

Approximation:

- D.P. Williamson and D.B. Shmoys, The Design of Approximation Algorithms, 2011.
- D.Z Du, K-I. Ko, and X.D. Hu, Design and Analysis of Approximation Algorithms, 2012.

Questions?