智能软件与工程学院
School of Intelligent Software and Engineering

数据结构与算法
Data Structures and Algorithms

钮鑫涛
Nanjing University
2023 Fall

The slides are mainly adapted from the original ones shared by Chaodong Zheng and Kevin Wayne. Thanks for their supports! We also use some materials from stanford-cs161.

Course Info

- Instructor: 钮鑫涛 (Email: niuxintao@nju.edu.cn)
- Prerequisites: programming and discrete mathematics (some basic probability theory)
- QQ group: 892855425
  - **please show your name, student ID, and department when applying to join the QQ group**
- Course homepage: https://niuxintao.github.io/courses/2023Fall-DS/
- Online Judge：http://172.29.6.1/
- Office hour: Wednesday，2-4 pm，Thursday 10-12 am （拟定南雍楼223)

Teaching Assistants

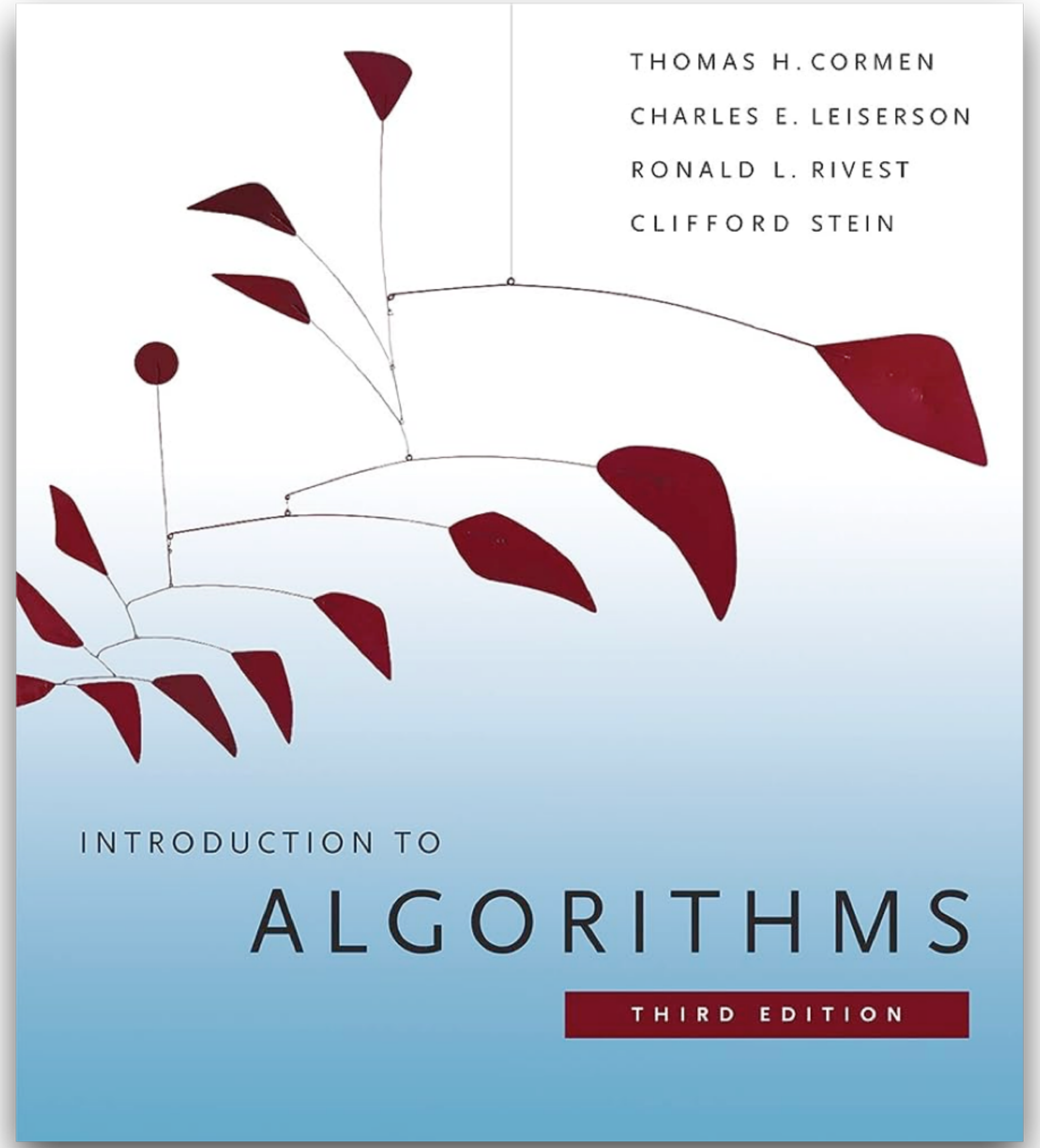
- Heading TA:
  - Hongnan Chen (MG21330010@smail.nju.edu.cn)
- TAs:
  - Coming soon!

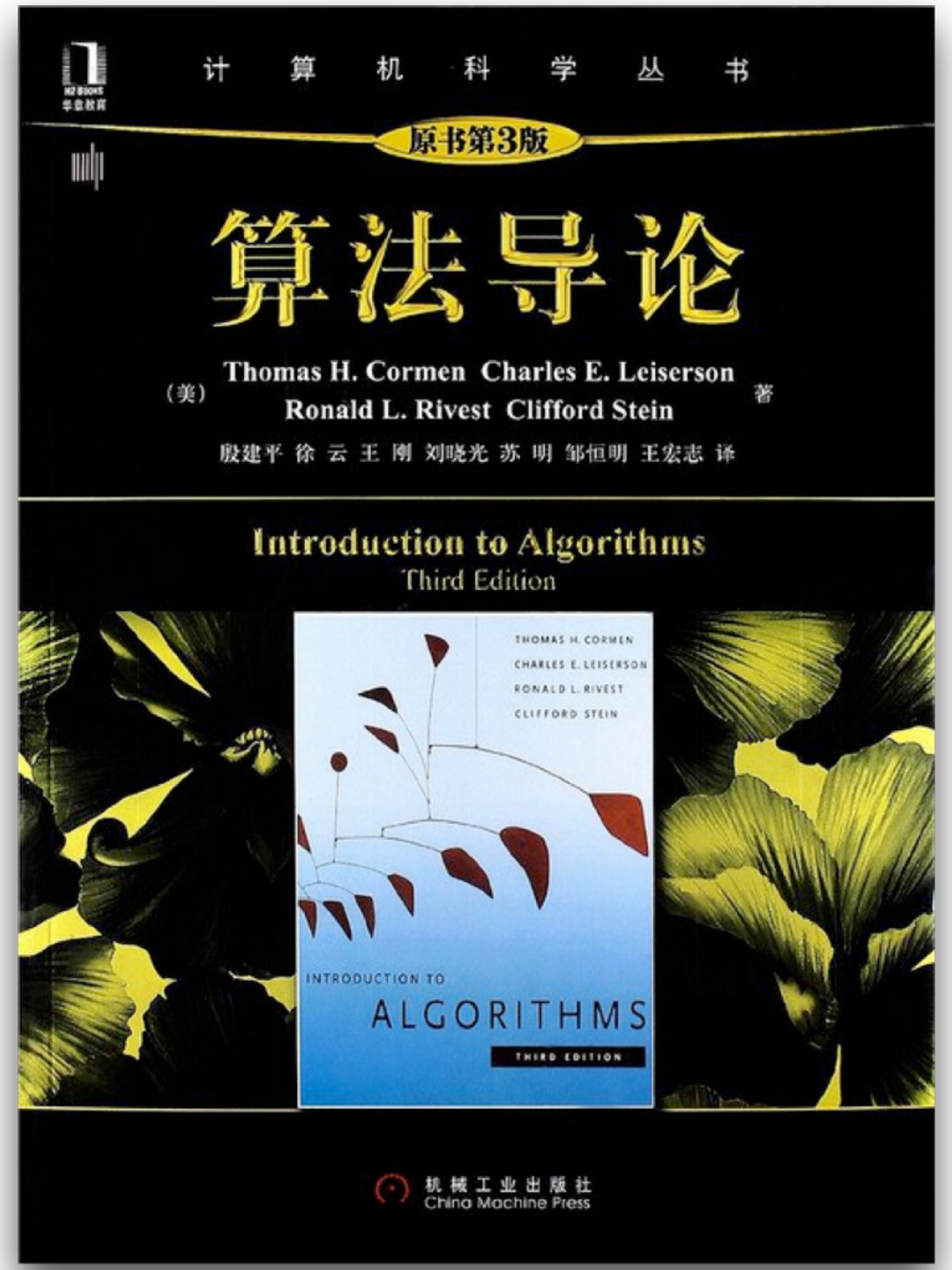
Strongly recommend asking questions in the QQ group for help (We will check them regularly).
Also recommend asking TA questions **personally** during office hours to seek additional help.

Textbook

- “Introduction to Algorithms” by C.L.R.S (中文版：算法导论)

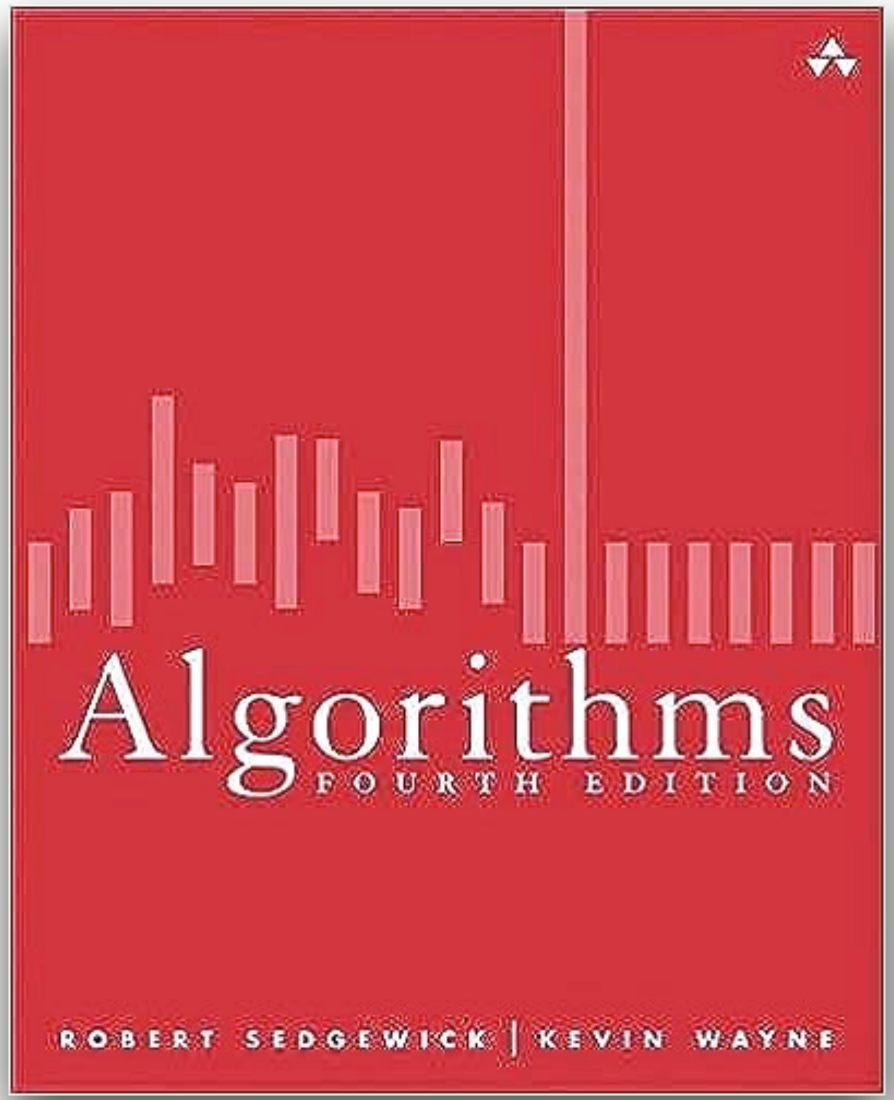
- Version： 3rd edition or 4th edition

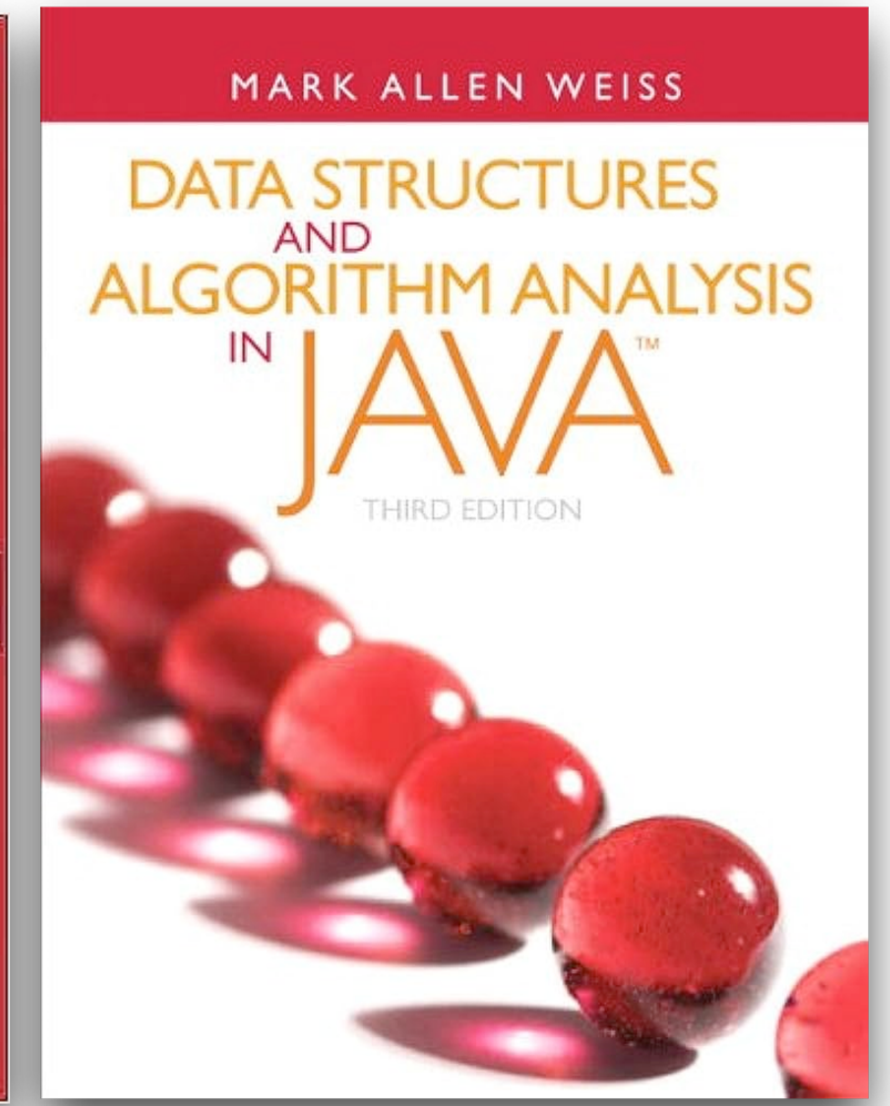


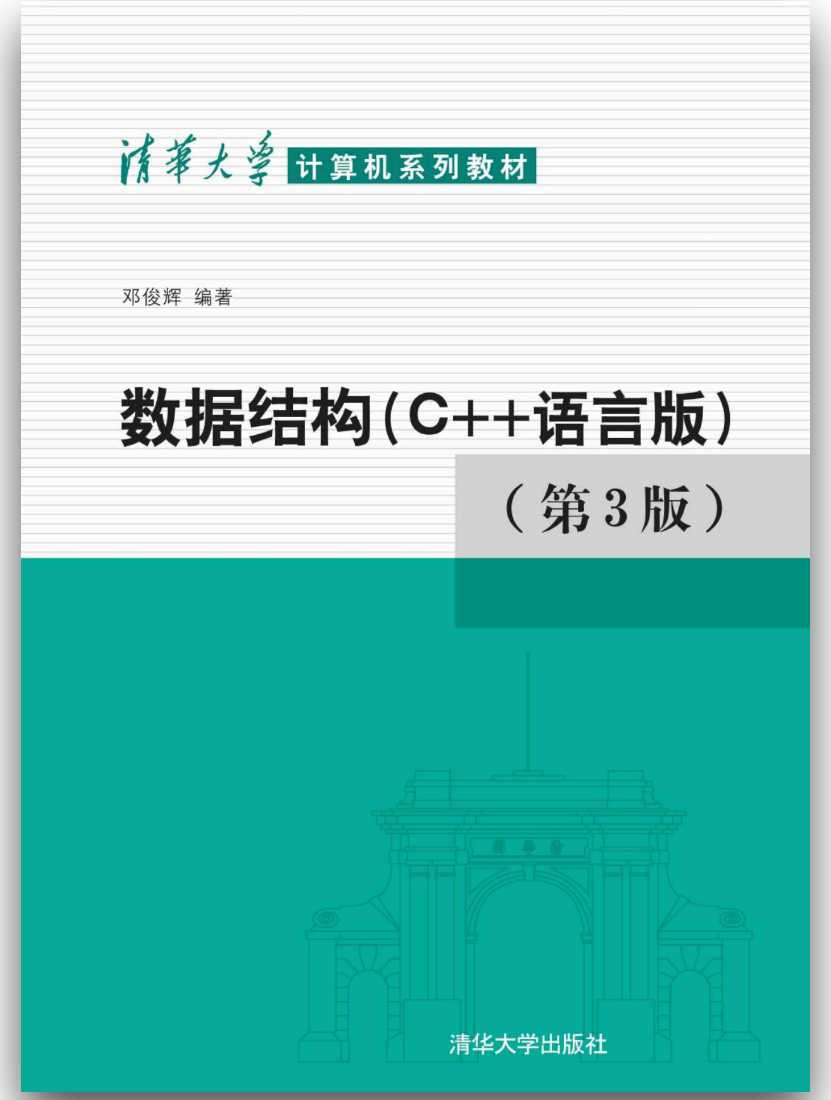


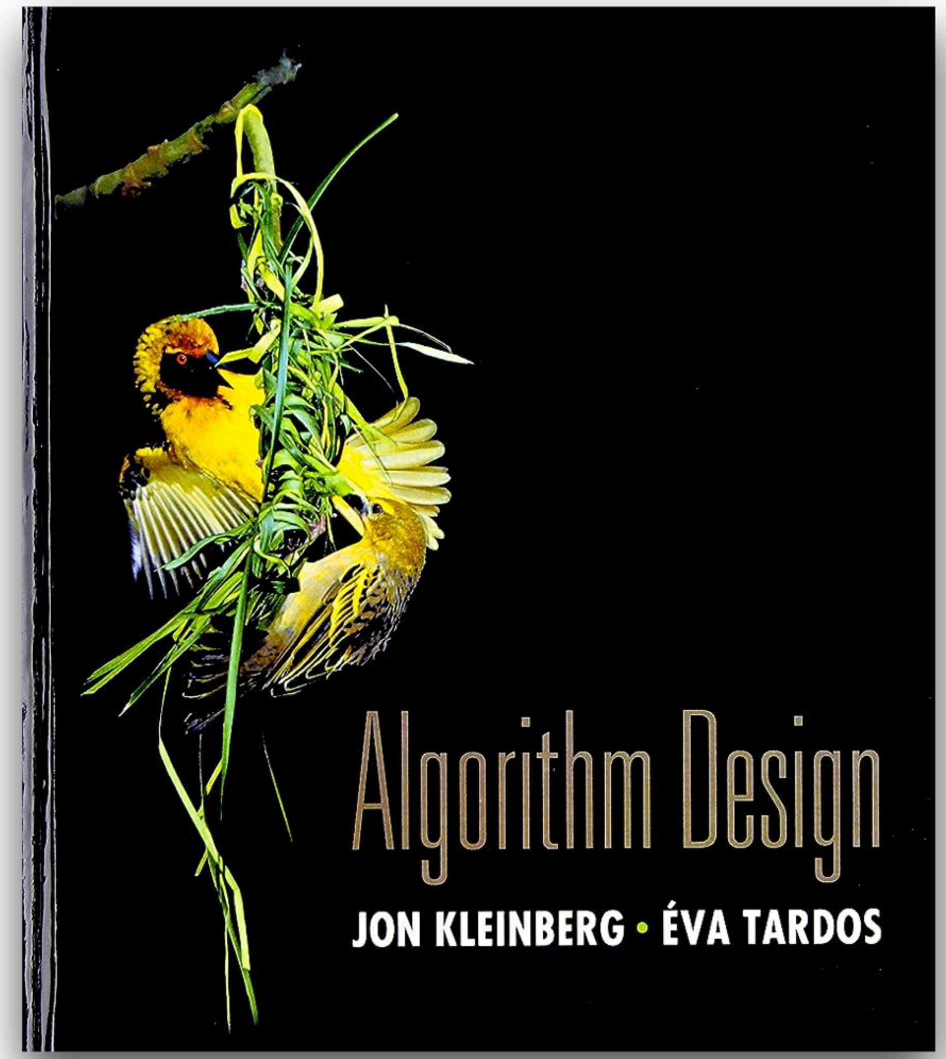
References

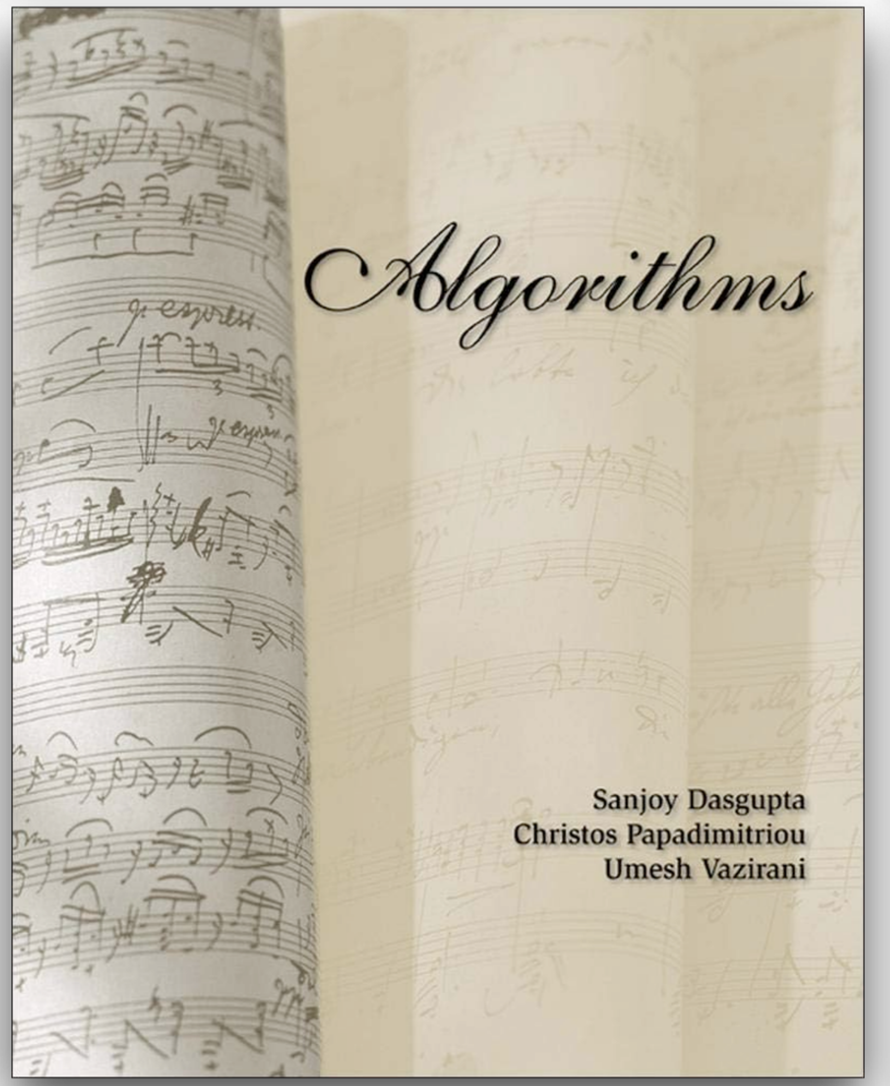
- “Algorithms” by Robert Sedgewick, Kevin Wayne
- “Data structures and algorithm analysis in java” by Mark Allen Weiss
- “数据结构(C++语言版)第3版” by 邓俊辉
- “Algorithm Design” by kleinberg and éva tardos
- “Algorithms” by by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani
- “Algorithms” by Jeff Erickson

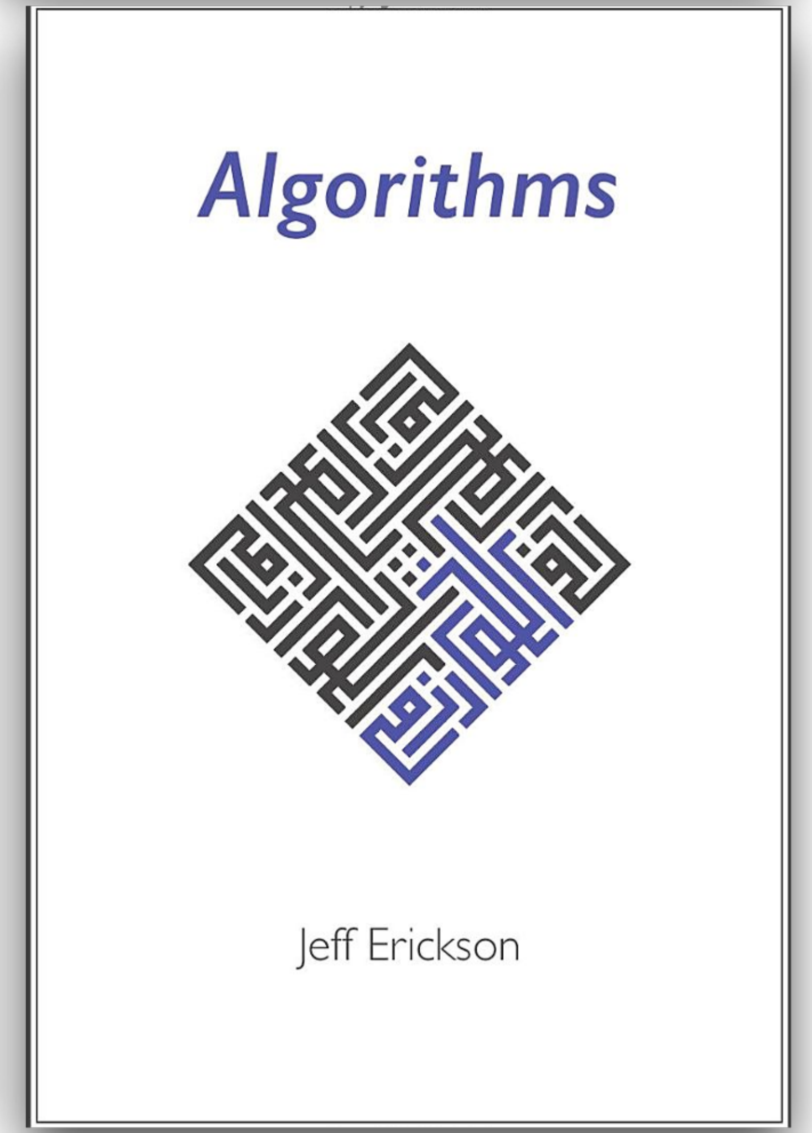












Grading

- Problem Sets + Programming Assignments + Exams
  - Problem sets (PS): weekly, (30%)
  - Programming Assignments (PA): weekly, (30%)
  - Exams: Final Exam (40%)

We ~~may~~ add some computer examination (As part of PA)

More on Online Judge

- Log in (your account ID and your initial password are both your student ID)
  - If you find your account cannot log in, please find TA for help
  - After log in, please change your password
- Programming Assignments are posted and evaluated on this site.
- Only available at Nanjing University

Academic Integrity

- Always try to solve PS and PA independently.
- You may discuss with others if you really need to, but you must list their names in your answers.
- You may not search and/or copy-paste existing solutions (Do not ask Chatgpt for help).

Syllabus

- A collection of common and widely used data structures;
- Basic algorithm design and analysis techniques;
- A collection of classical algorithms;
- Some related advanced topics, if we have time.

General goal: you can correctly and efficiently solve computational problems, by developing/picking appropriate algorithms and data structures.

Quotation

"Algorithms are the life-blood of Computer Science."

—Donald E. Knuth

"Computer science should be called computing science, for the same reason why surgery is not called knife science."

—Edsger Wybe Dijkstra

Quotation

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

—Linus Torvalds

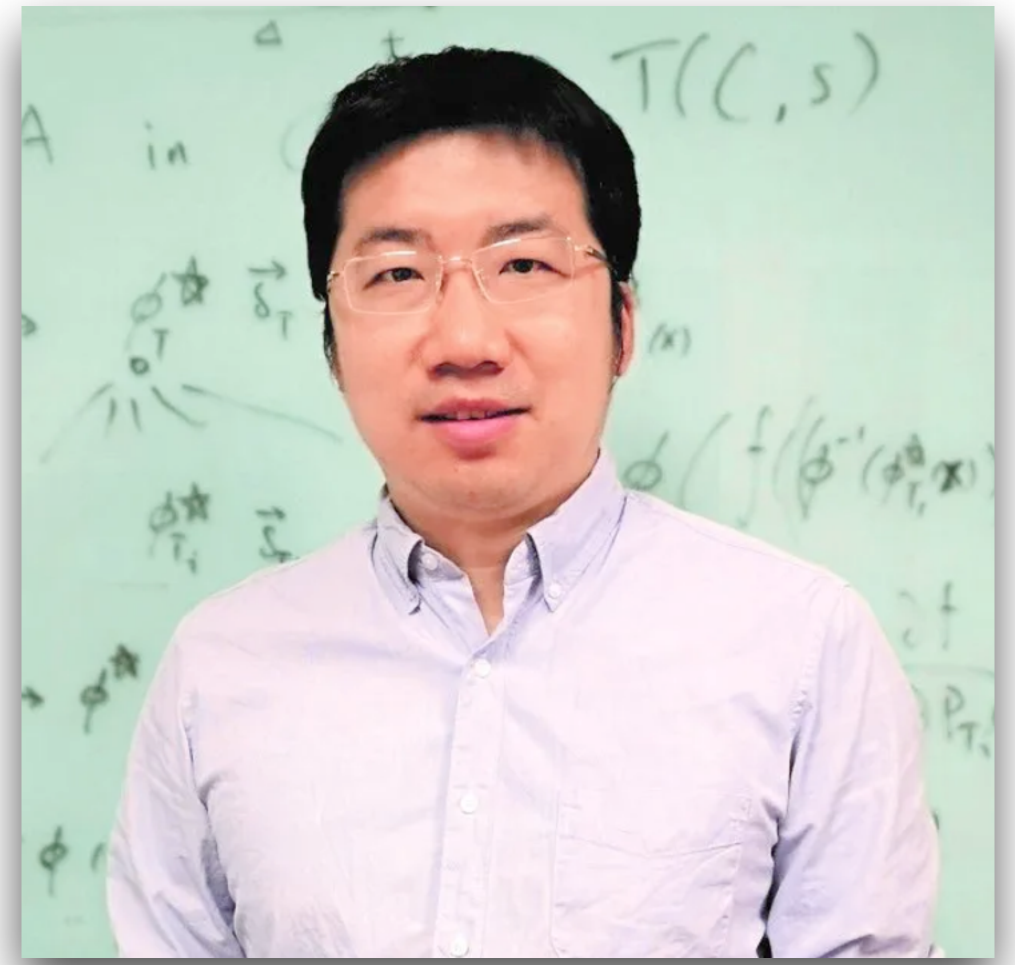
"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing."

— Francis Sullivan

Quotation

“Algorithms + Data Structures = Programs. ”

— Niklaus Wirth



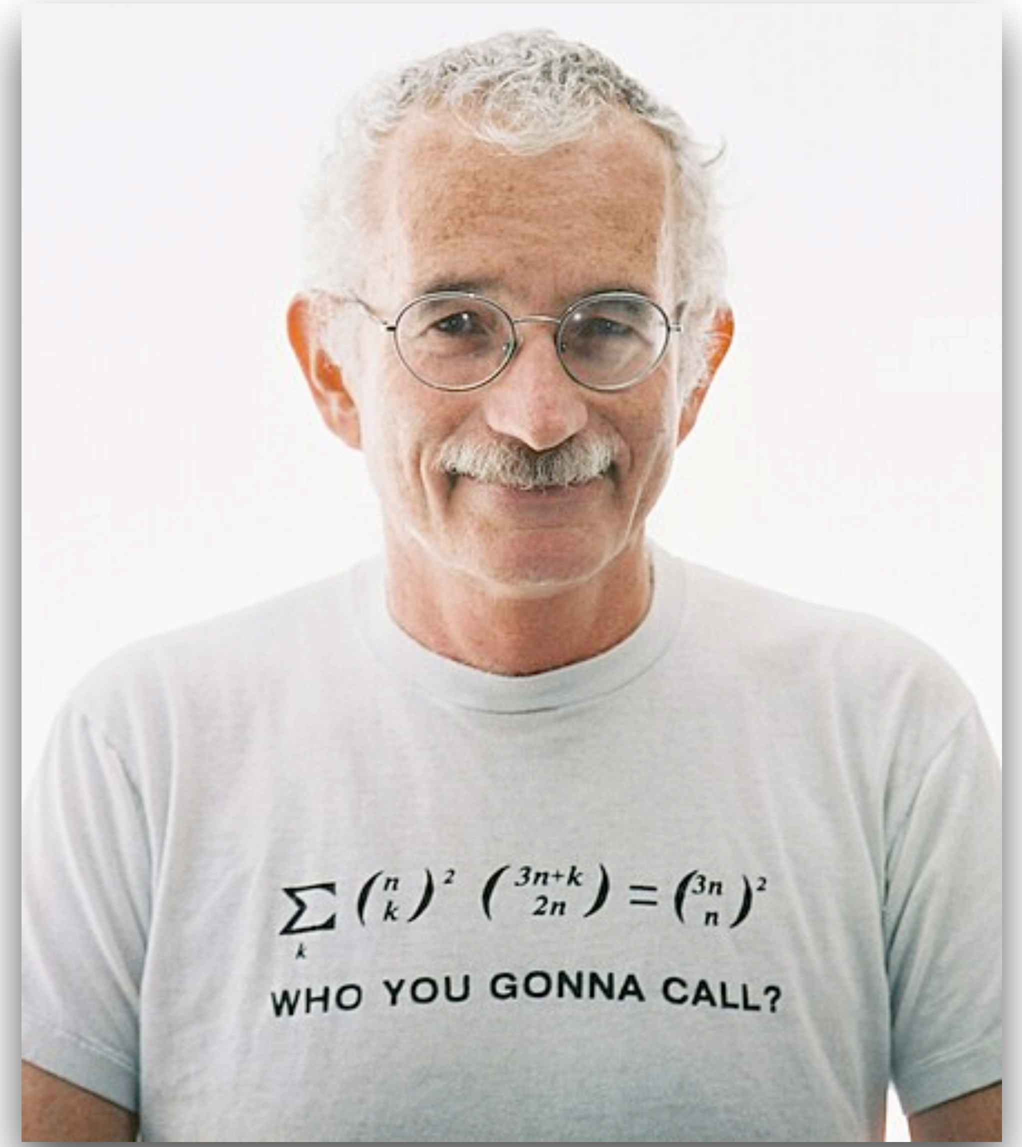
“计算问题因何而易、又因何而难”

— 尹一通

Quotation

"Mathematics my foot! Algorithms are mathematics too, and often more interesting and definitely more useful."

—Doron Zeilberger

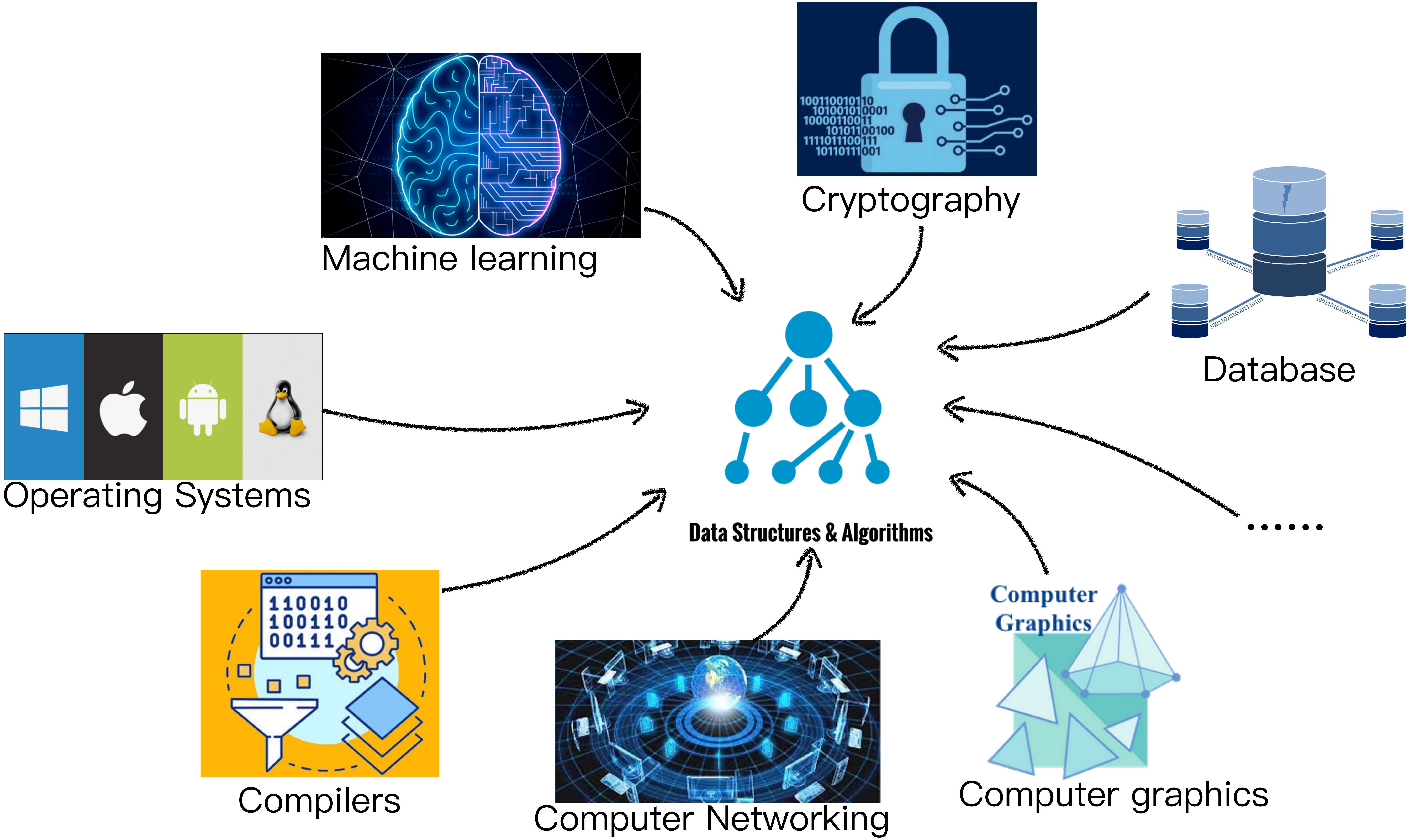


"It's easy to make mistakes that only come out much later, after you've already implemented a lot of code. You'll realize Oh I should have used a different type of data structure. Start over from scratch."

—Guido van Rossum

The importance of this course

Fundamental



The importance of this course

Influential

PERSONALITY TESTS

This Algorithm Knows You Better Than Your Facebook Friends Do

4:09 PM | JAN 12 | By CHRISTIE ASCHWANDEN

ALGORITHMS TAKE CONTROL OF WALL STREET

...led by thousands of little algorithms, and they've created a new ...dictable, and impossible for humans to comprehend. Photo: Mauricio...

THE WALL STREET JOURNAL. TECH

TECHNOLOGY

At UPS, the Algorithm Is the Driver

Turn right, turn left, turn right: inside Orion, the 10-year effort to squeeze every penny f...

By STEVEN ROSENBUSH and LAURA STEVENS

Feb. 16, 2015 8:28 p.m. ET

1832 959 87 COMMENTS

Can maths find you love? eHarmony's ...e algorithm

...maths find you love? The dating ...armony, who claim to have ...esponsible for a staggering half ...n marriages, use algorithms ...that purpose. They claim they ...ponsible for over 500 ...es a day and have data from ...on people looking for love. ...ew are 200 items they collect ...ionnaires from their premium customers. They claim that this data harvests six variables.

Computer Scientists Are Building Algorithms to Tackle COVID-19

Algorithms that can detect infections, dif... common flu, and more

Dave Gershgorn Mar 13 · 3 min read

新华报业网 > 首页 > 正文

新知 | 精准“杀熟”，该如何走出大数据的“坑”

2021/08/31 20:06 交汇点

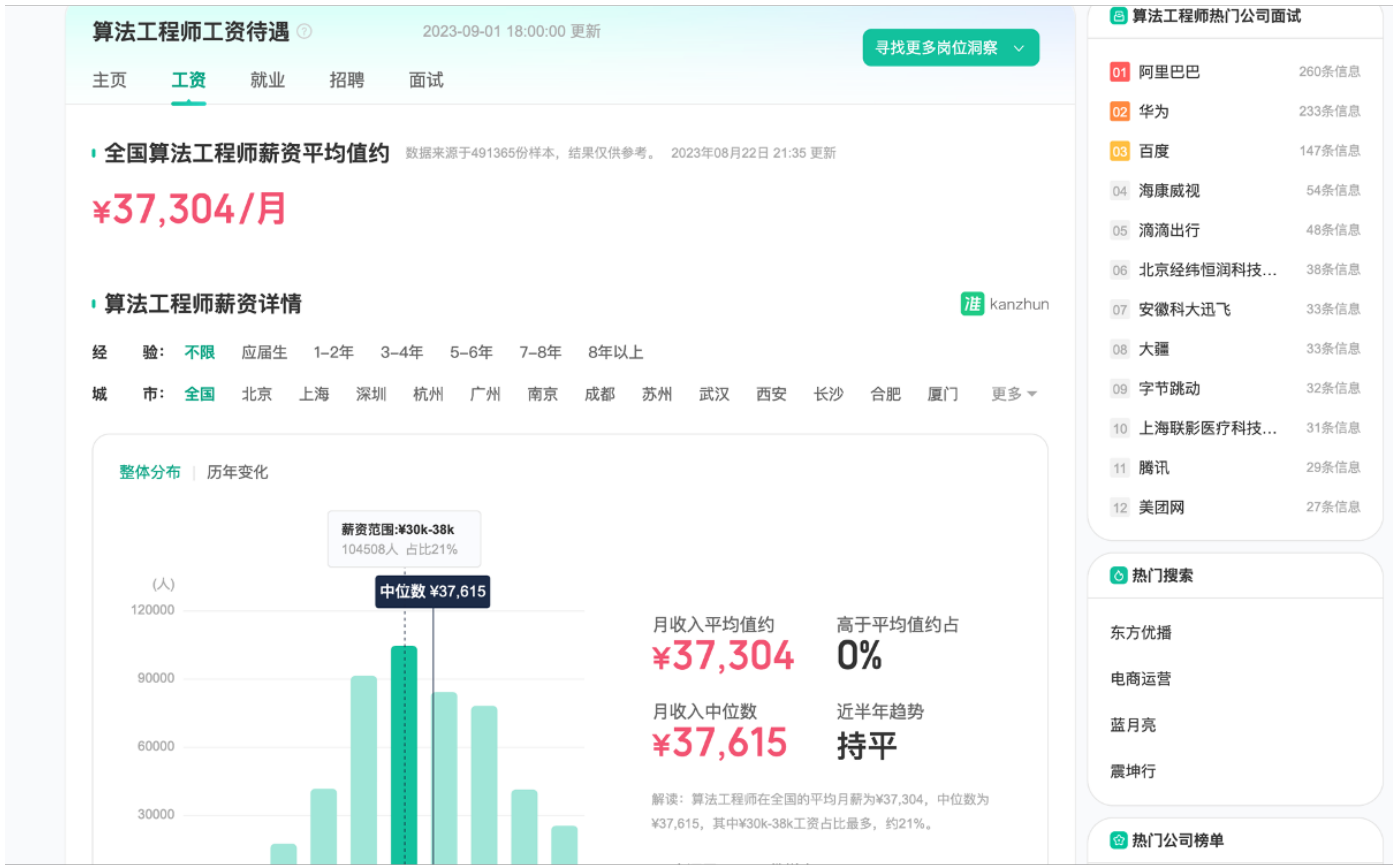
交汇点讯假期临近，在南京工作的蒋女士正计划着回哈尔滨老家看望父母，但在网络平台购买机票时她却发现了其中“猫腻”。同一时间同一航班同一舱位，使用蒋女士自己的账号购买相比用同事的账号购买价格要贵几百元。“前段时间我频繁搜索回家的几条航线，所以应该是被大数据‘杀熟’了。”

为消费能力高者推荐高价产品，同一段车程不同用户网约车平台上显示的车费不同……现实生活中，很多人都像蒋女士一样感觉“被监视”“被涨价”，那么为何会出现杀熟现象？算法如何调整商品售价？个人信息安全如何得到有效保护？《科技周刊》记者邀请相关专家为大家解

智能软件与工程学院
School of Intelligent Software and Engineering

The importance of this course

Profitable



The importance of this course

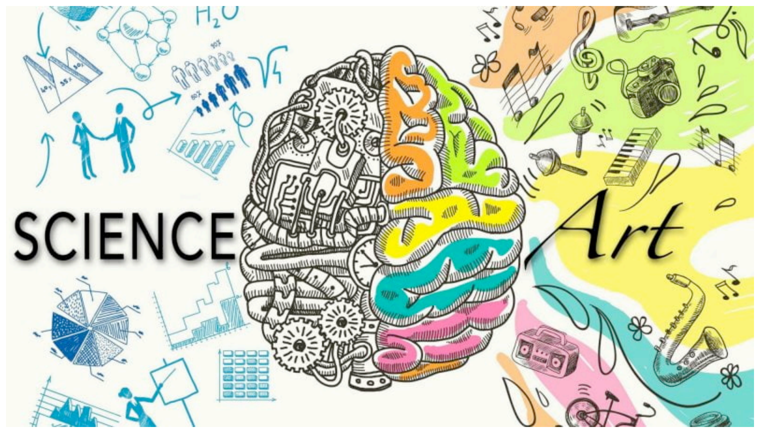
Useful

- Algorithm is the art of problem-solving — you will learn a lot of useful techniques!

- When dealing with industrial problems (with large-scale inputs), having good algorithms makes great impact!

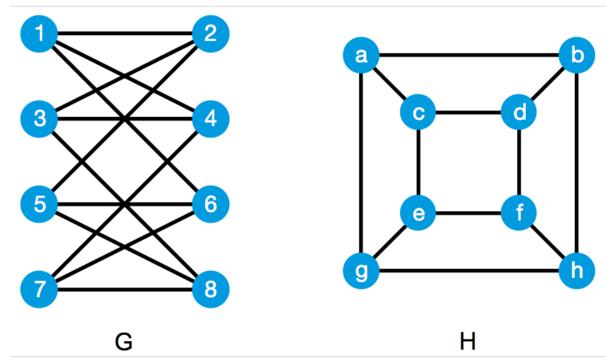
The importance of this course

Last, but not least — Fun



Algorithm design is both an art and a science.

Many surprises!



Many exciting research questions!

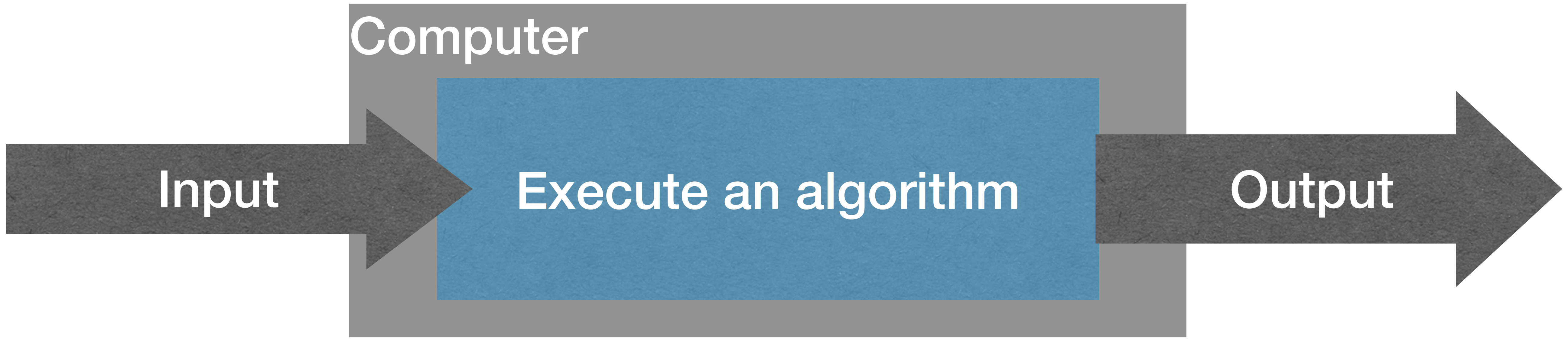
Let’s Start

What is an Algorithm?

- In computer science, an algorithm is any **well-defined** computational **procedure** that takes some value(s) as **input** and produces some value(s) as **output**.
- Another perspective: we can also see an algorithm as a **tool/method** for **solving** a **well-specified** computational **problem**.

Well defined?

- For example, the ***integer sorting problem***:
  - Input: a sequence of n integers $< a_1, a_2, \ldots, a_n >$
  - Output: a reordering $< a'_1, a'_2, \ldots, a'_n >$ of input where $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.



- Counterexamples (ill-defined):
  - Finding a perfect mate
  - Writing a great novel

Well defined?

- For example, the ***integer sorting procedure***:
  - Input: a sequence of n integers $< a_1, a_2, \ldots, a_n >$
  - Output: a reordering $< a'_1, a'_2, \ldots, a'_n >$ of input where $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

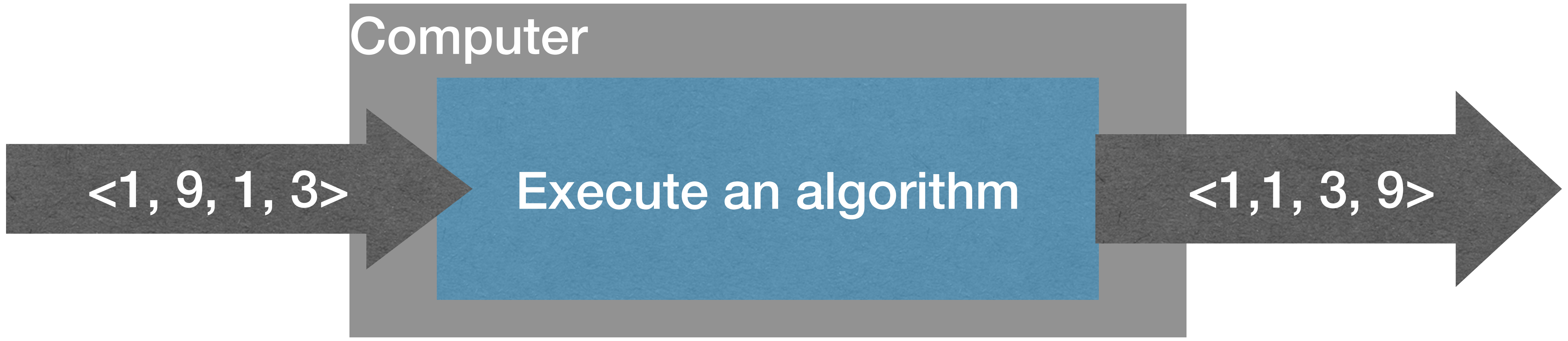
- Step 1 – Set MIN to the first location of $< a_1, a_2, \ldots, a_n >$
- Step 2 – Search the minimum element from the location MIN to the last location of $< a_1, a_2, \ldots, a_n >$
- Step 3 – Swap with value at location MIN
- Step 4 – Increment MIN to point to next element
- Step 5 – Repeat the above steps 2-4 until list is sorted

- One Counterexample:
  - “倒入适量食用油，待油温达到7成热时分次放入鸡丁，将鸡丁炸制成金黄色后捞出，加入适量盐调味”

Instance of one problem

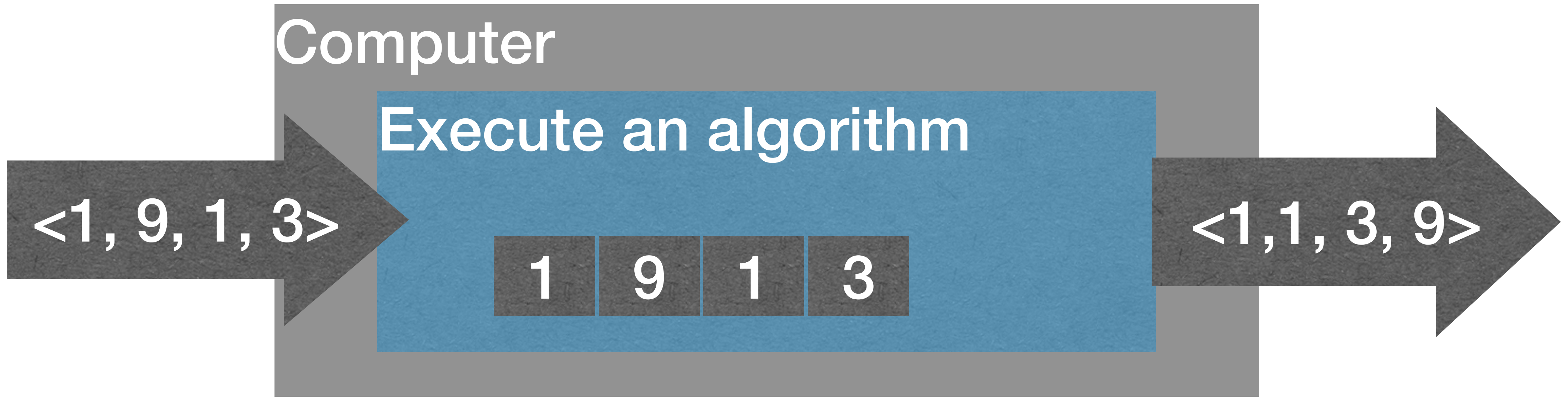
- A particular input of a problem is an instance of that problem.

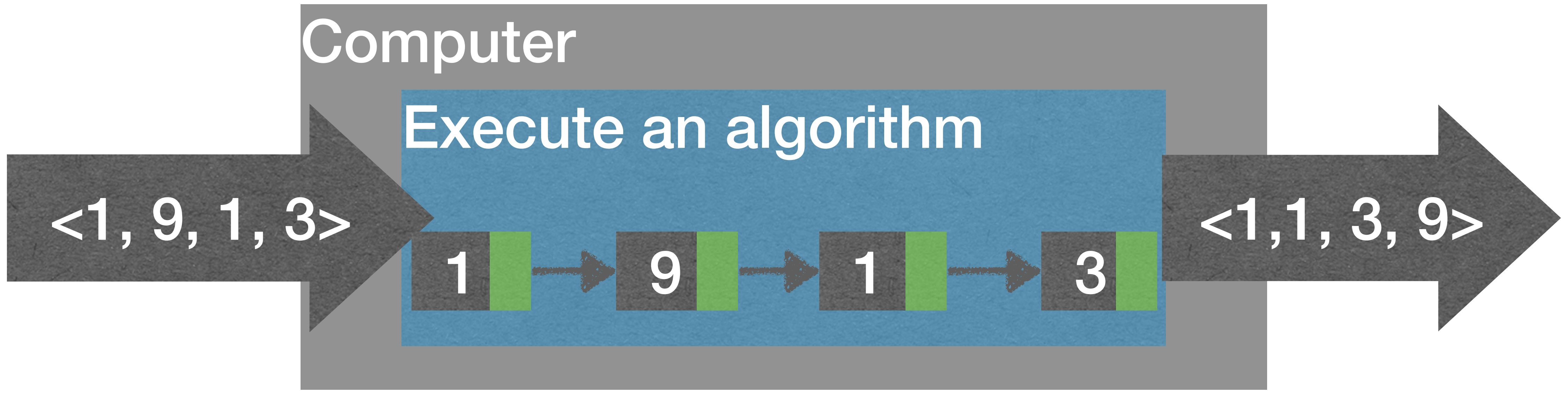
- For example, one instance of ***integer sorting problem***:
  - Sorting the sequence $< 1,9,1,3 >$



What is a data structure?

- A data structure is a way to **store and organize data** in order to facilitate **access** and **modifications**.
    - E.g., *array, linked list.*
- Different types of data usually demand different data structures.
- One type of data could be represented by different data structures.





Picking an appropriate one is **important!**

Algorithm and Data Structures

- Algorithms and Data Structures are closely related
  - An algorithm applies to a particular data structure
  - An Algorithm usually need data structures internally to work as intended.
  - Using the right data structure helps drastically improve an algorithm's performance

Algorithms  Data Structures

hand in hand

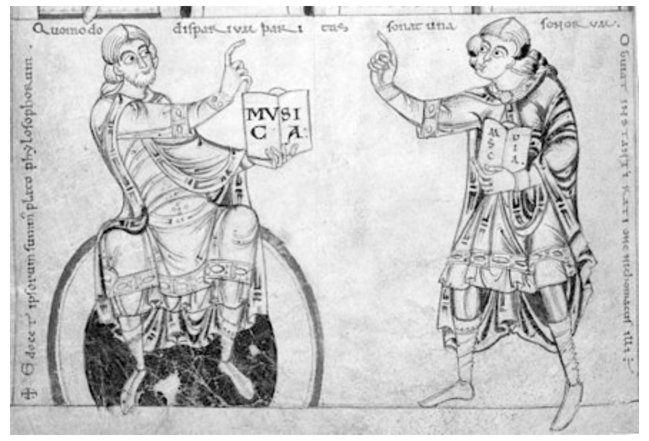
A brief history of Algorithm

300 BC

Euclid's algorithm for finding the greatest common divisor of two numbers

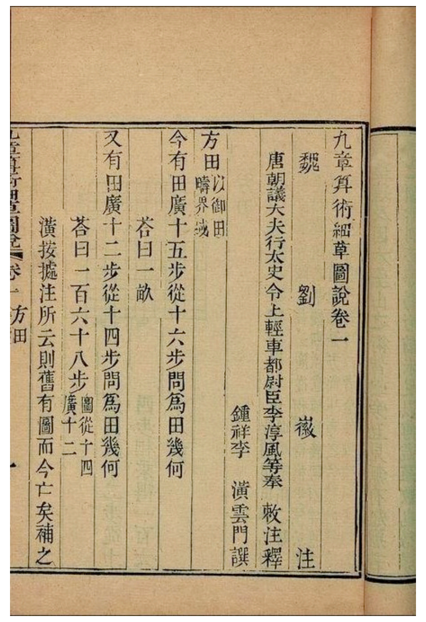
200 BC

The Sieve of Eratosthenes, used by Greek mathematicians to find prime numbers.



150BC

高斯消去法（英语：Gaussian Elimination），是线性代数中的一个算法，以数学家卡尔·高斯命名，但最早出现于中国古籍《九章算术》，成书于约公元前150年, 作者已不可考，后由刘徽做注



820AD

Al-Khawarizmi described algorithms for solving linear equations and quadratic equation

Latin: algorithmus, meaning “calculation method”, is the origin of the word “algorithm”

➡ Data from https://en.wikipedia.org/wiki/Timeline_of_algorithms

A brief history of Algorithm

1540 Lodovico Ferrari discovered a method to find the roots of a quartic polynomial

1614 John Napier develops method for performing calculations using logarithms

1671 Newton–Raphson method which produces successively better approximations to the roots of a real-valued function

1768 Leonard Euler publishes his method for numerical integration of ordinary differential equations

1842 Ada Lovelace writes the first algorithm for a computing engine

1936 Turing machine developed by Alan Turing.

The notion of **algorithm** then is **formalized** by Church and Turing.

➡ Data from https://en.wikipedia.org/wiki/Timeline_of_algorithms

A brief history of Algorithm

- Kruskal's algorithm
- Shor's algorithm
- Peterson's algorithm
- Bron–Kerbosch …rithm
- Fisher–Yates shuffle
- Tabu search
- Floyd–Warshall algorithm
- Dijkstra's algorithm
- Backpropagation
- Daitch–Mokotoff Soundex
- Quicksort
- Simulated annealing
- Kosaraju's algorithm
- Tarjan's strongly connected components algorithm
- Simplex algorithm
- Genetic algorithms
- Knuth–Morris–Pratt algorithm
- Fibonacci search technique
- Hamming distance
- AC-3 algorithm
- C4.5 algorithm
- Gibbs sampling
- QR algorithm
- PageRank Algorithm
- ……

‣ Some algorithms were discovered by undergrads in a course like this!

The goal of algorithm design

- Generally, algorithm designing has two main goals:
  - Does it work (correctness)?
    - An algorithm is correct if for every input instance of the given problem, the algorithm halts with the correct output.
  - Can I do better (efficiency)?
    - A superior algorithm is also correct and solve the given problem, but uses less computing resources (time and memory) than less efficient ones.

An Introductory Example:

Integer Multiplication

Integer Multiplication

- Problem Description: Integer Multiplication
  - Input: Two n-digit nonnegative integers, x and y.
  - Output: The product $x \times y$.

If you want to multiply numbers with different lengths (like 1234 and 56), a simple hack is to just add some zeros to the beginning of the smaller number (for example, treat 56 as 0056).

The Grade-School Algorithm

- Multiply the multiplicand by each digit of the multiplier
- Then add up all the properly shifted results.
  - It requires memorization of the multiplication table for single digits.

Examples:

$$
\begin{array}{r}
123 \\
\times 321 \\
\hline
123 \\
246 \\
369 \\
\hline
39483
\end{array}
\qquad
\begin{array}{r}
123 \\
\times 021 \\
\hline
123 \\
246 \\
000 \\
\hline
2583
\end{array}
\qquad
\begin{array}{r}
99 \\
\times 77 \\
\hline
693 \\
693 \\
\hline
7623
\end{array}
$$

carries

Pseudocode

- We'll typically describe algorithms as procedures written in a **pseudocode**
  - Independent of specific languages, but uses structural conventions of a normal programming language (like C, Java, C++)
  - Intended for human reading rather than machine reading (omit nonessential details and easier to understand)

Pseudocode

- Some conventions:
  - Give a valid name for the pseudocode procedure, specify the input and output variables' names (as well as the types) at the beginning.
  - Use proper Indentation for every statement in a block structure.
  - For a flow control statements use **if-else**. Always end an **if** statement with an **end-if**. Both **if**, **else** and **end-if** should be aligned vertically in same line.

More details can be found at 1. http://www.mscs.mu.edu/~rge/cosc3100/pseudocode.pdf, 2. https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470029757.app1

Pseudocode

- Some conventions:
  - Use ← or := operator for assignment statements, Use = for equality check
  - Array elements can be represented by specifying the array name followed by the index in **square brackets**. For example, A[i] indicates the *i*th element of the array A
  - For looping or iteration use **for** or **while** statements. Always end a for loop with **end-for** and a while with **end-while**.
    - Two or more conditions can be connected with **and** or **or**. Use **not** to negative condition.

More details can be found at 1. http://www.mscs.mu.edu/~rge/cosc3100/pseudocode.pdf, 2. https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470029757.app1

Pseudocode

pseudocode example of the Grade-School Algorithm

```
Procedure  GradeMult(x, y)
In: two n-digit positive integers x, y.
Out: the product p := x · y.

A := split x into an array of its digits // e.g., 1235 -> [1, 2, 3, 5]
B := split y into an array of its digits
product := [1..2n]
for i :=  1 to n:
    carry := 0
    for j :=  1 to n:
        temp := product [i + j - 1] + carry + A[i] * B[j]
        carry := temp / 10
        product [i + j - 1]  :=  temp  mod base
    end for
    product[i+n] := carry
end for
p := transform the product to integer
return p
```

How many operations?

If we count one-digit operations (additions and multiplications):

- At most n^2 multiplications
- and then at most n^2 additions (for carries)
- and then I have to add n different $2n$-digit number —> $2n^2$ additions
- Finally, at most $n^2 + n^2 + 2n^2 = 4 \times n^2$ single digit operations

Why

Constant

Can we do better?

"Perhaps the most important principle for the good algorithm designer is to refuse to be content "

—Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman

"Make It Work, Make It Right, Make It Fast. "

—Kent Beck

Try the recursion?

- Can we divide the integer multiplication into several sub problems which involve multiplications of numbers with fewer digits? If so, we can use recursion.
- A number x with an even number n of digits can be expressed in terms of two n/2-digit numbers, its first half and second half a and b:
  - $x = 10^{n/2} \times a + b$

What if n is an odd number?

- Similarly, $y = 10^{n/2} \times c + d$
- Then, $x \times y = (10^{n/2} \times a + b) \times (10^{n/2} \times c + d) =$
  $10^n \times (a \times c) + 10^{n/2} \times (a \times d + b \times c) + b \times d$ (EQ 1)

Try the recursion?

$$x \times y = 10^n \times (a \times c) + 10^{n/2} \times (a \times d + b \times c) + b \times d \qquad (EQ1)$$

- According to EQ1, instead of directly multiplying x and y, we need to four relevant products: $a \times c$, $a \times d$, $b \times c$, and $b \times d$, both of them have few digits to multiply!
  - Then, 1. tack on n trailing zeroes to $a \times c$; 2. add $a \times d$ and $b \times c$, then tack on n/2 trailing zeroes to the result; 3. Add the above results to $b \times d$.
- For $a \times c$ and other smaller multiplication problems, we can recursively apply the above technique.

A recursive multiplication algorithm

```
Procedure  RecIntMult(x, y)
In: two n-digit positive integers x, y.
Out: the product p := x · y.
//assume n is a power of 2.

if n = 1 then // base case
      return x·y
else
   a, b :=  split x into halves // x = 10^{n/2} · a + b
   c, d :=  split y into halves
   u := RecIntMult(a, c)
   v := RecIntMult(b, d)
   w := RecIntMult(a, d)
   t := RecIntMult(b, c)
   z := w + t
   p := 10^n · u + 10^{n/2} · z + v
   return p
end if
```

Problem

- Is the RecIntMult algorithm faster or slower than the grade-school algorithm?
  - We will learn later, but now, you can implement them and try

Karatsuba Multiplication

- Discovered in 1960 by Anatoly Karatsuba, who at the time was a 23-year-old student!

- One observation of $x \times y = 10^n \times (a \times c) + 10^{n/2} \times (a \times d + b \times c) + b \times d \qquad (EQ1)$:

  - Do we really need $a \times d$ and $b \times c$ separately?

  - No, we only need their sum, that is $a \times d + b \times c$

- Then the question is how can we get $a \times d + b \times c$, without the results of $a \times d$ and $b \times c$?

Karatsuba Multiplication

$$x \times y = 10^n \times (a \times c) + 10^{n/2} \times (a \times d + b \times c) + b \times d \qquad (EQ1)$$

- The solution proposed by Karatsuba is:
  - Recursively compute $a \times c$
  - Recursively compute $b \times d$
  - Then, compute $a + b$ and $c + d$, and recursively compute $(a + b) \times (c + d)$
  - Get $a \times d + b \times c$ by $(a + b) \times (c + d)$ - $a \times c$ - $b \times d$
  - Compute EQ1 by add these results properly (adding trailing zeroes)

Karatsuba Multiplication

```
Procedure  Karatsuba(x, y)
In: two n-digit positive integers x, y.
Out: the product p := x · y.
//assume n is a power of 2.

if n = 1 then // base case
      return x·y
else
   a, b :=  split x into halves // x = 10^{n/2} · a + b
   c, d :=  split y into halves
   u := Karatsuba(a, c)
   v := Karatsuba(b, d)
   w := Karatsuba(a + b, c + d)
   z := w - u - v
   p := 10^n · u + 10^{n/2} · z + v
   return p
end if
```

Karatsuba Multiplication

- Hence, Karatsuba multiplication makes only three recursive calls!
- Saving a recursive call should save on the overall running time, but by how much?
- Is the Karatsuba algorithm faster than the grade-school multiplication algorithm?

More advanced results

- Toom-Cook (1963): instead of breaking into three n/2-sized problems, it should be breaked into five n/3-sized problems.
  - Runs in time $O(n^{1.465})$

The description of O is given later

- Schönhage–Strassen (1971): Runs in time $O(n \times log(n) \times log(log(n)))$
- Furer (2007) Runs in time $O(n \times log(n) \times (2^{O(log^* n)})$
- Harvey and van der Hoeven (2019) Runs in time $O(n \times log(n))$

One more thing: what about incorrect algorithm?

- A Incorrect algorithms might:
  - Never halt on some instances;
  - Halt with incorrect outputs on some instances.

One more thing: what about incorrect algorithm?

- Incorrect (or, imperfect) algorithms can be useful!
  - Correct (perfect) algorithms might be too slow or even do not exist.
  - Imperfect algorithms may output good enough (but not perfect) answers.
  - Imperfect algorithms may never stop in some extreme cases, but halt and output correct answers in most (say 99.9%) cases.

Further reading

- [CLRS] Ch.1
- [AI] Ch.1