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 王智彬
Nanjing University
2025 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

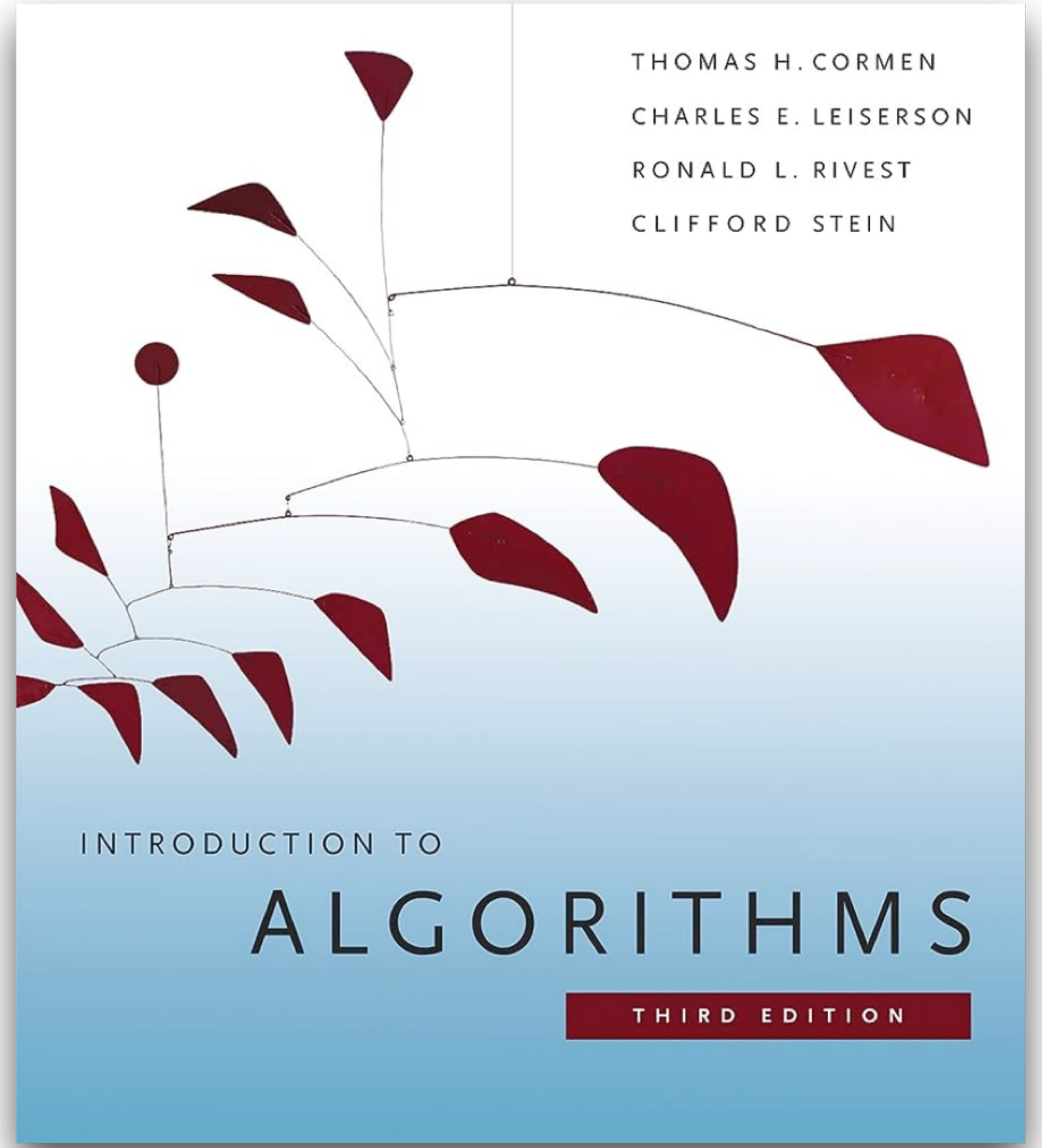
- Instructor: 钮鑫涛 (Email: niuxintao@nju.edu.cn) 王智彬 (wzbwangzhibin@gmail.com)
- Prerequisites: programming and discrete mathematics (some basic probability theory)
- QQ group: 981865070 (1班)， 1057722169（2班）
  - **please show your name, student ID, and department when applying to join the QQ group**
- Course homepage: https://niuxintao.github.io/courses/2025Fall-DS/
- Online Judge：iseoj.nju.edu.cn

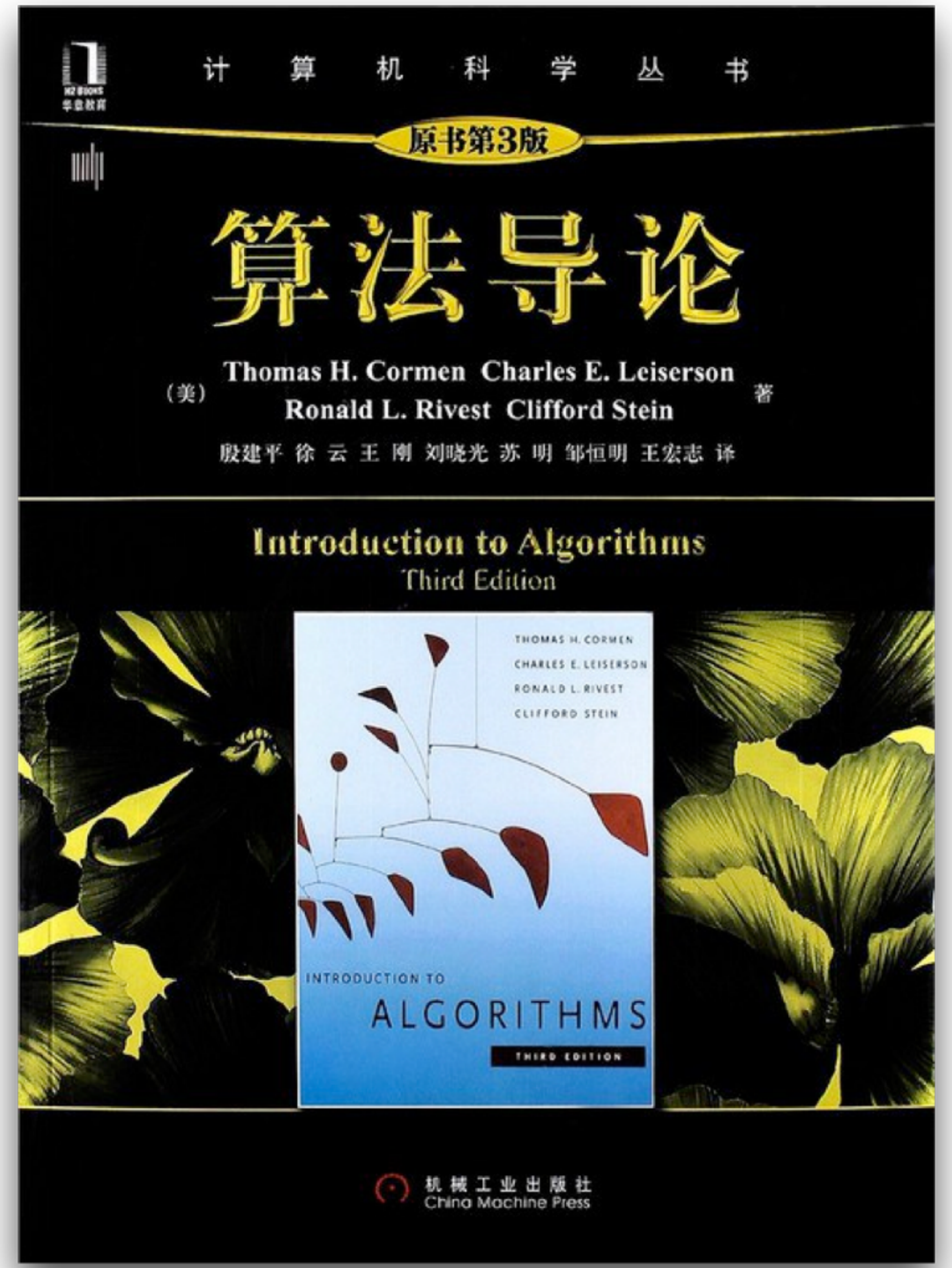
Teaching Assistants

- TA:
  - (1班) 贾学海 602025720002@smail.nju.edu.cn
  - (2班) 赵可泰 502025330076@smail.nju.edu.cn
  - QQ群里还有你们的学长助教（龚亮、计昀）

Textbook

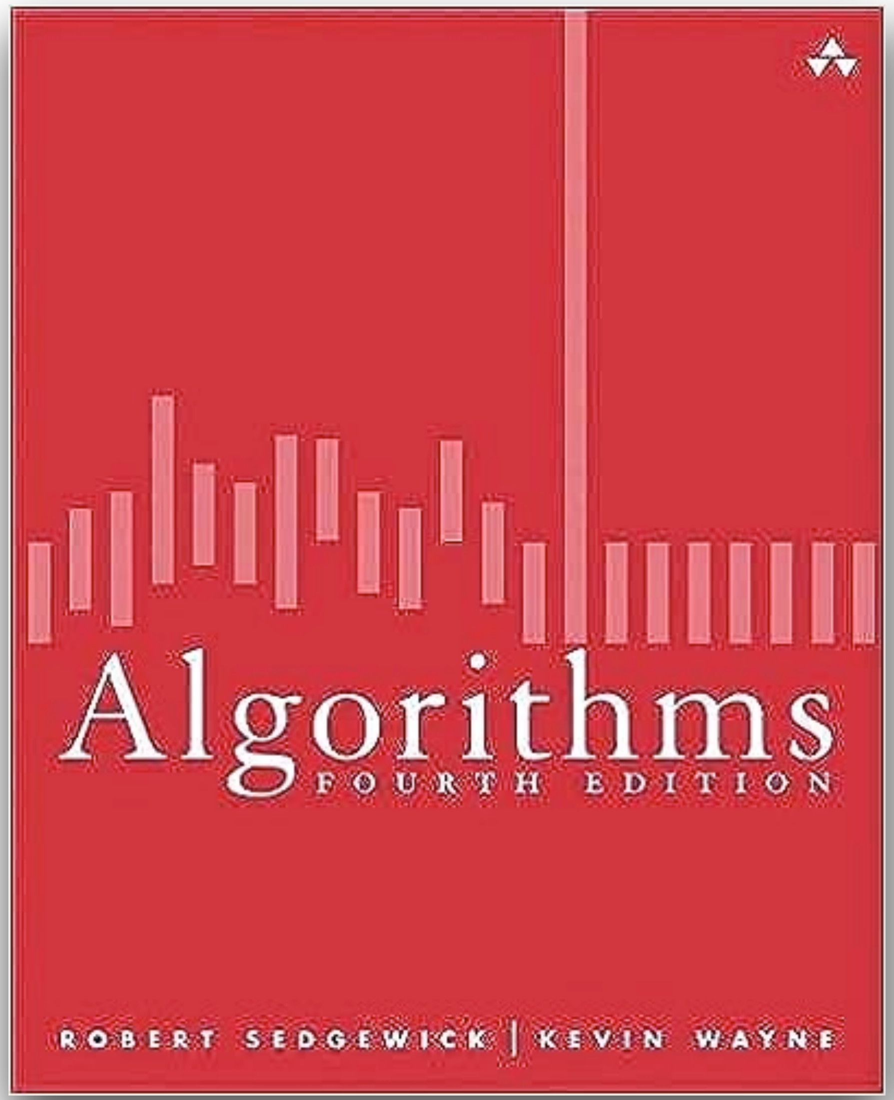
- “Introduction to Algorithms” by C.L.R.S (中文版：算法导论)
- Version： 3rd edition or 4th edition

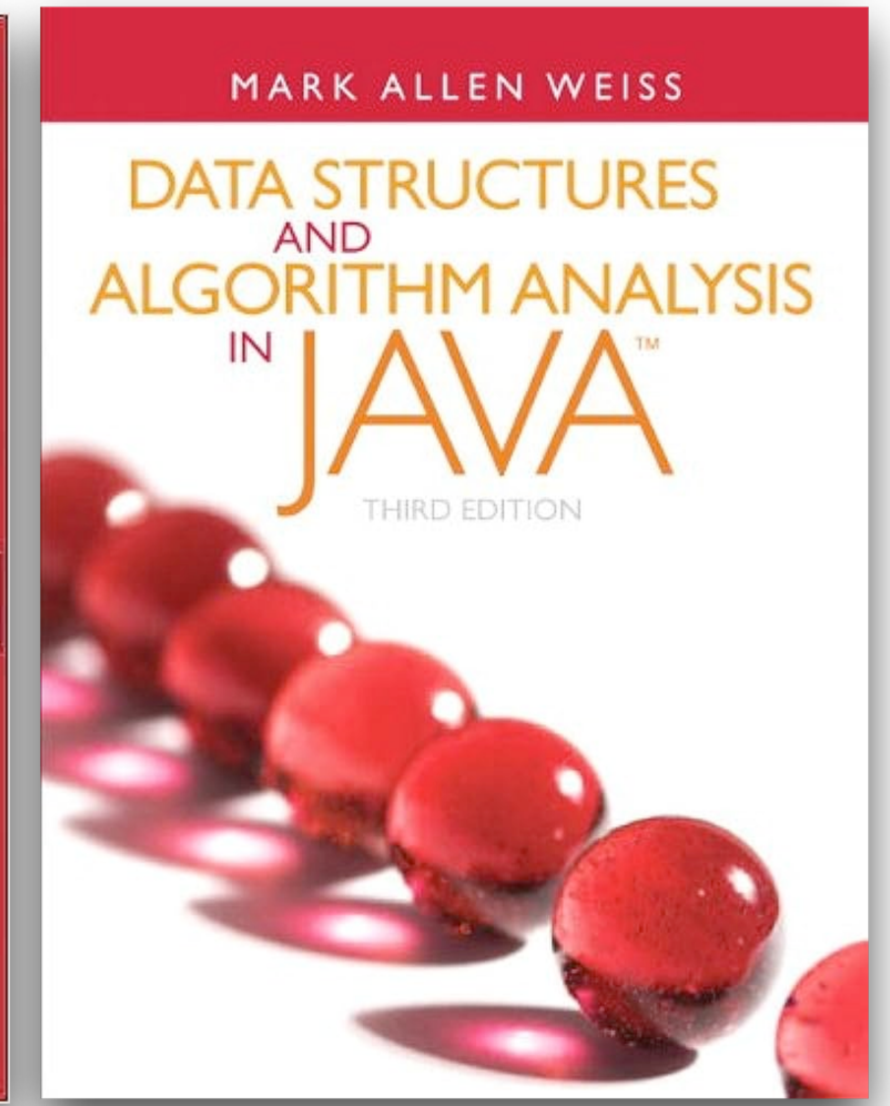


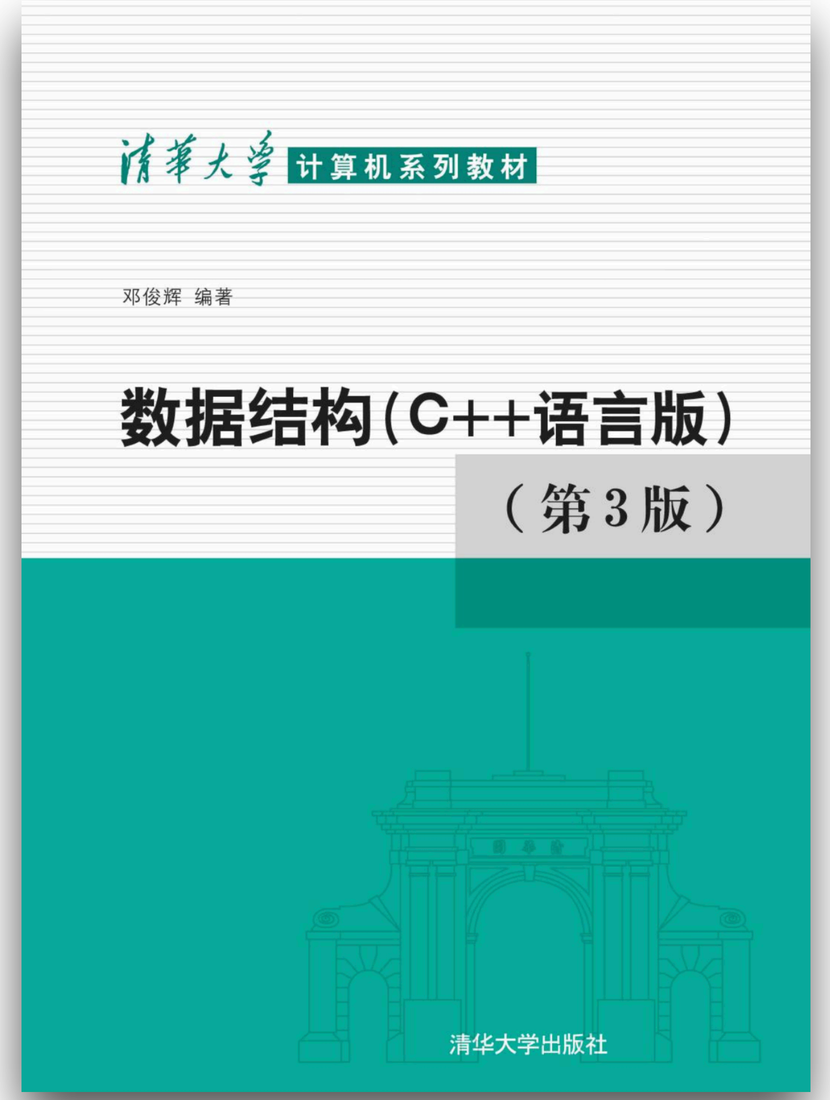


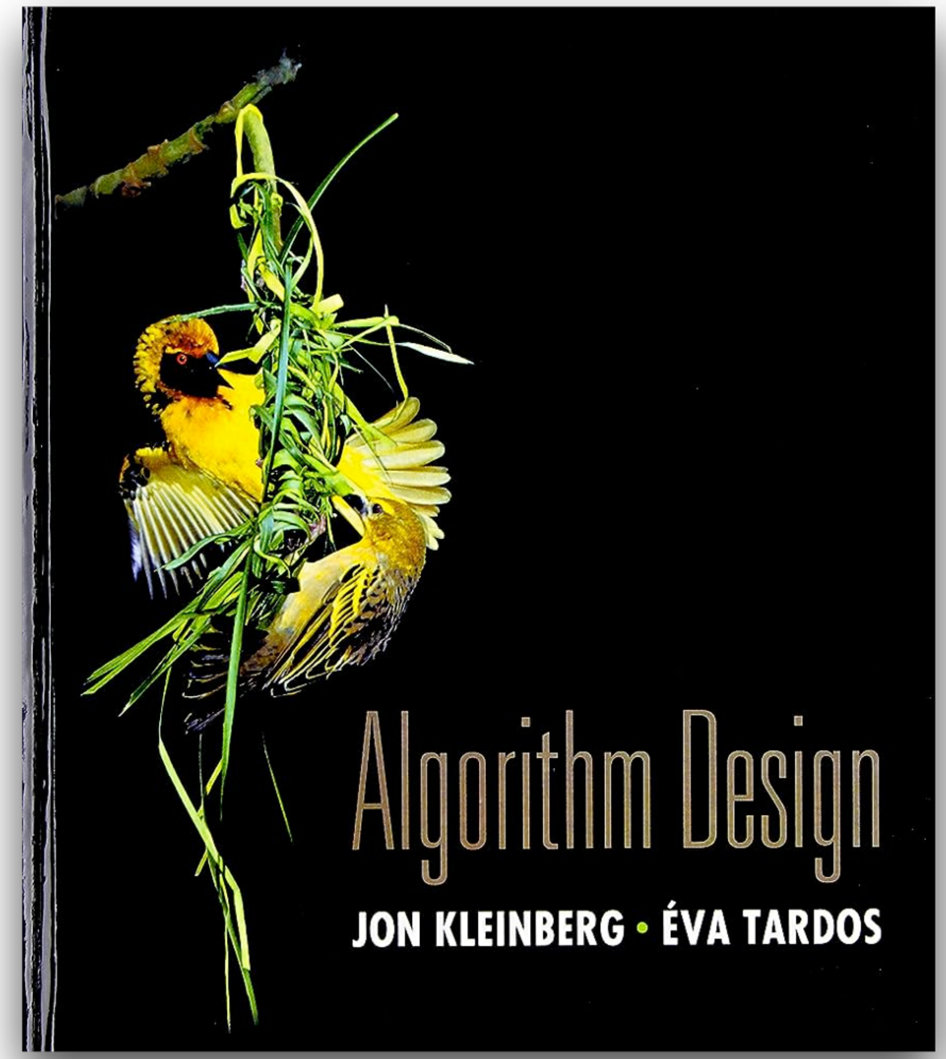
References

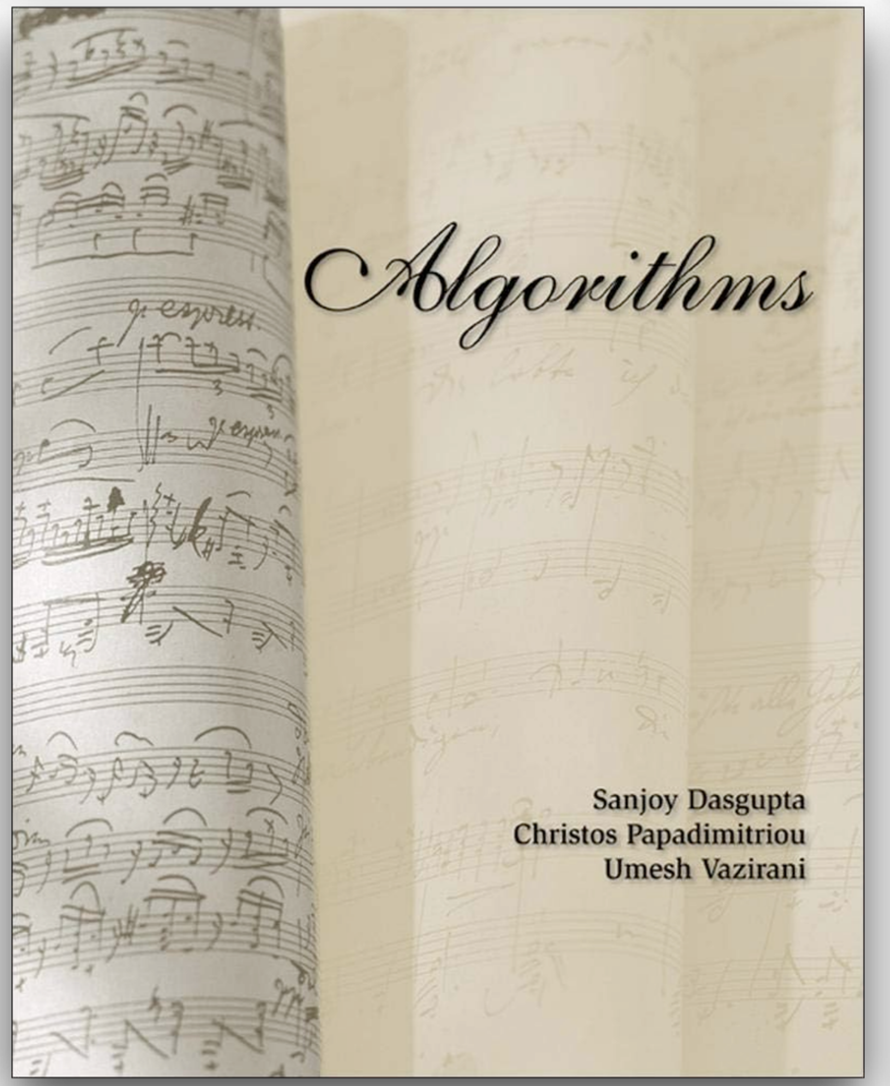
- “Algorithms” by Robert Sedgewick, Kevin Wayne
- “Data structures and algorithm analysis in java” by Mark Allen Weiss
- “数据结构(C++语言版)第3版” by 邓俊辉
- “Algorithm Design” by kleinberg and éva tardos
- “Algorithms” by by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani
- “Algorithms” by Jeff Erickson

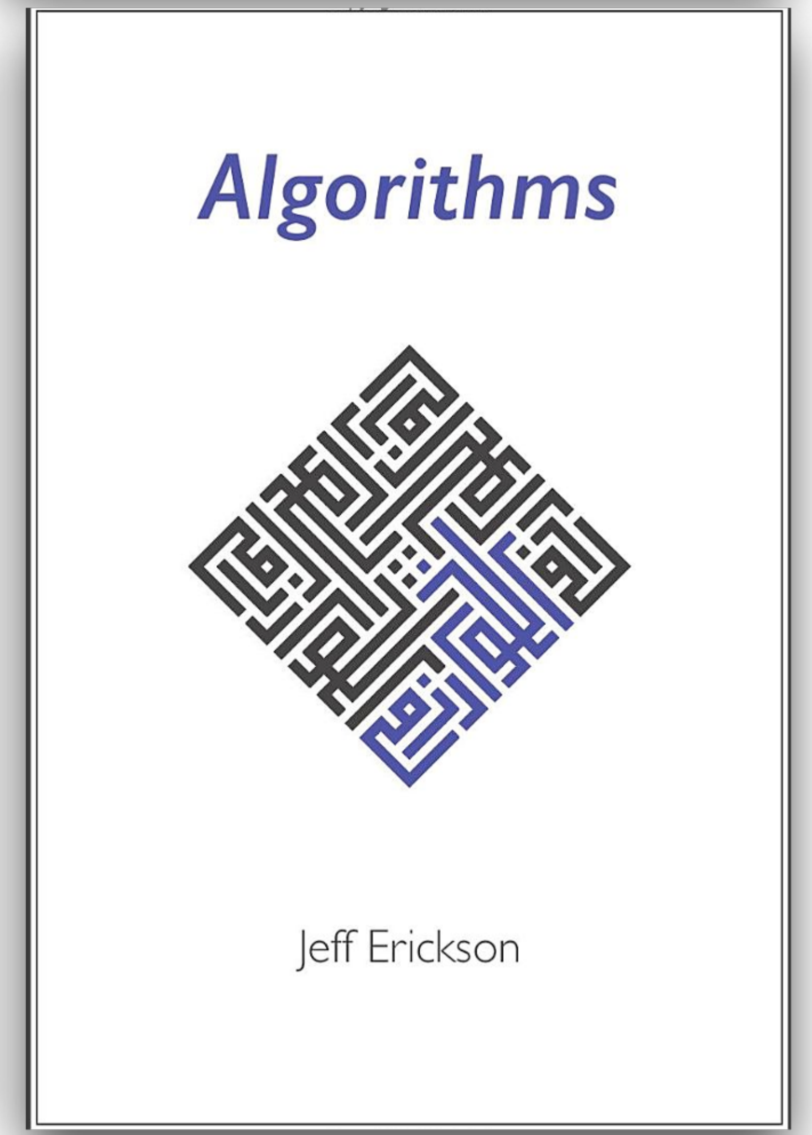












Grading

- Programming Assignments + Computer-based examination + Exams
  - Programming Assignments (PA): weekly (30%)
  - Computer-based Examination: two exams (30%)
  - Exams: Final Exam (40%)

We may adjust the proportions of each component

Academic Integrity

- Always try to solve 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

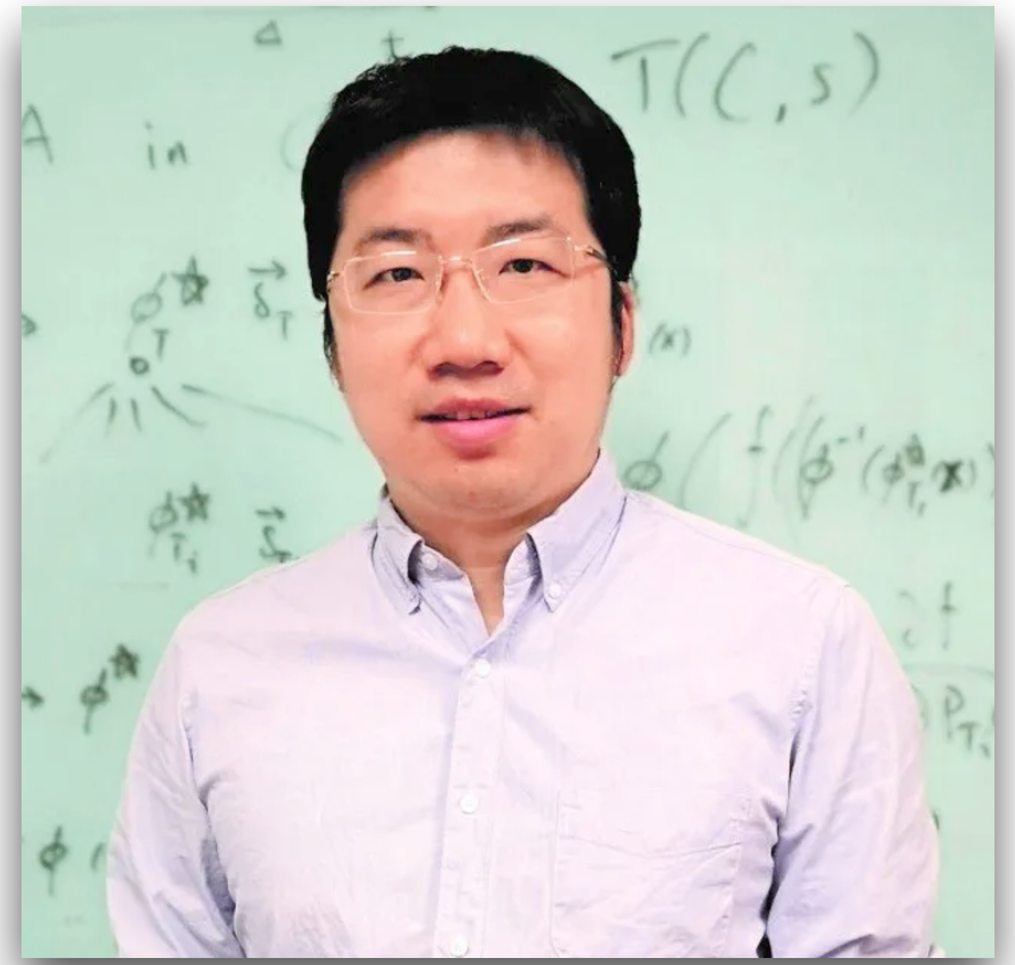
“For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.”

— Francis Sullivan

Quotation

“Algorithms + Data Structures = Programs. ”

— Niklaus Wirth



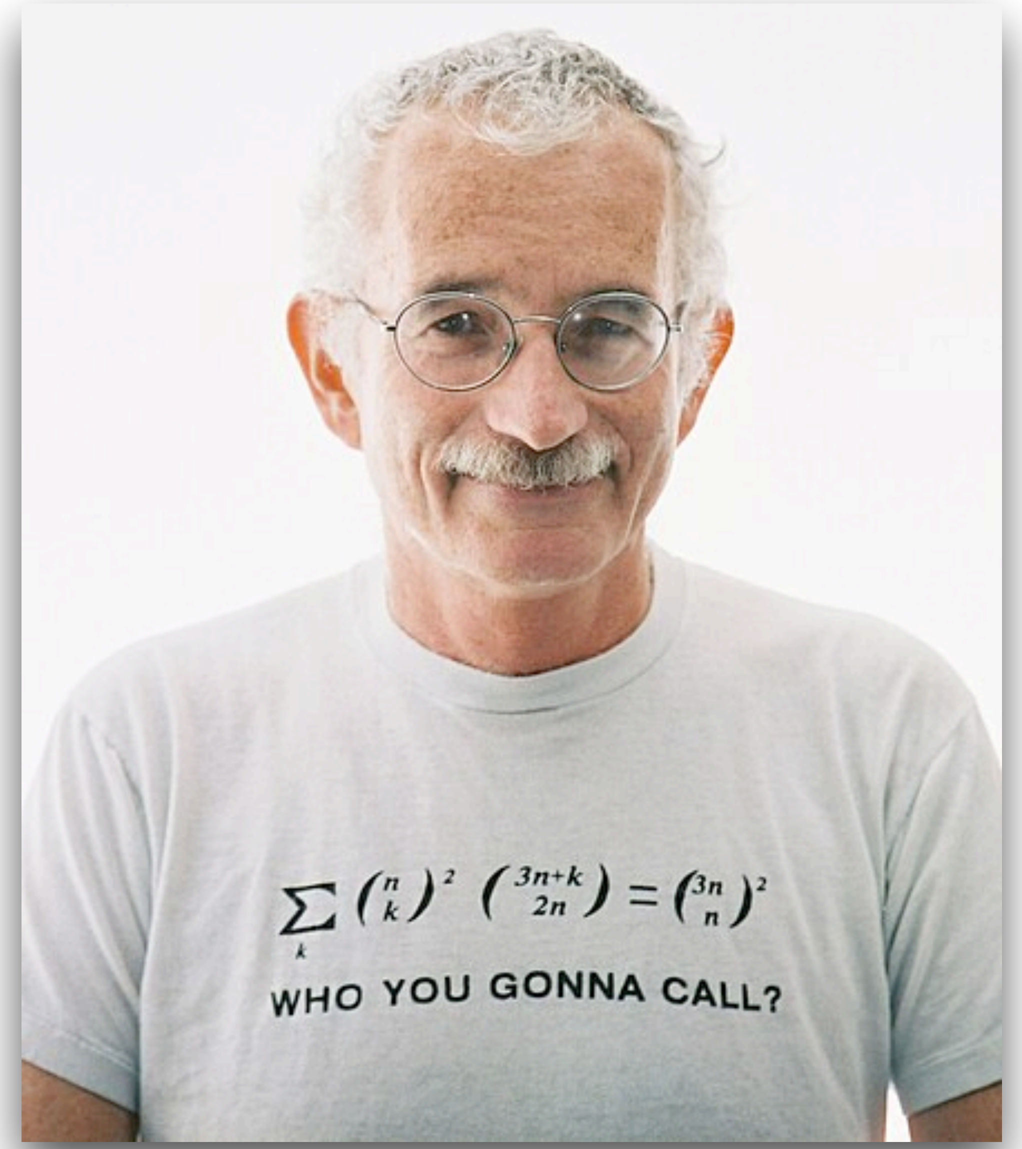
“计算问题因何而易、又因何而难”

— 尹一通

Quotation

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

—Doron Zeilberger

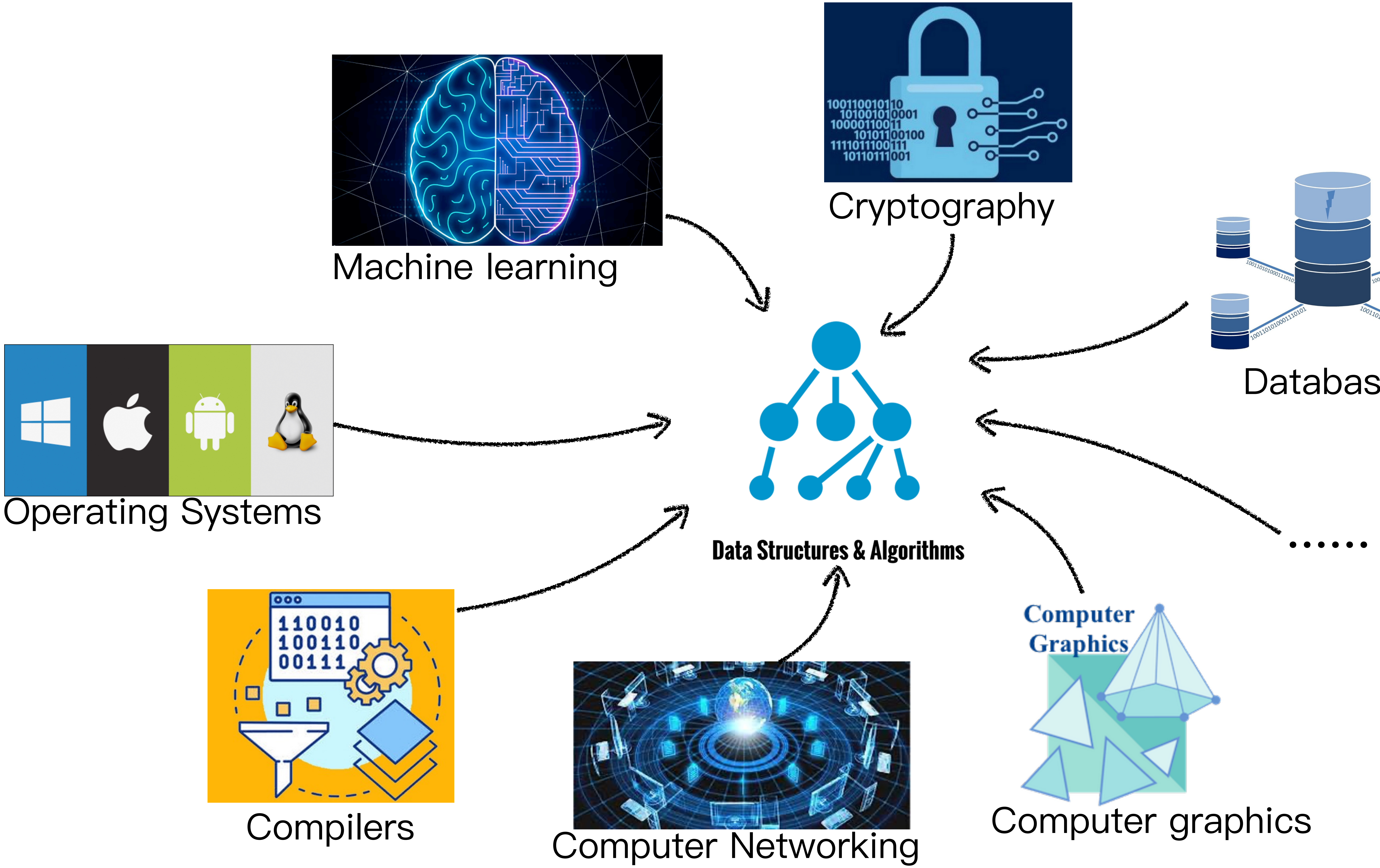


“It's easy to make mistakes that only come out much later, after you've already implemented a lot of code. You'll realize Oh I should have used a different type of data structure. Start over from scratch.”

—Guido van Rossum

The importance of this course

Fundamental



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

The importance of this course

Influential

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

The importance of this course

Profitable

Company	Rating	Role	Open jobs	Average salary	Low	High
Audience	3.3 ★	Algorithm Engineer	9 open jobs	$176,118 / yr	$141K	$223K
ByteDance	3.9 ★	Algorithm Engineer	4,369 open jobs	$221,714 / yr	$175K	$285K
Continental	4 ★	Algorithm Engineer	3,975 open jobs	$140,330 / yr	$113K	$177K
Ford Motor Company	4 ★	Algorithm Engineer	2,761 open jobs	$159,044 / yr	$128K	$200K
Apple	4.2 ★	Algorithms Engineer	4,335 open jobs	$252,725 / yr	$201K	$325K
Google	4.4 ★	Algorithm Engineer	1,731 open jobs	$278,688 / yr	$220K	$360K
Hudson River Trading	4.1 ★	Algorithm Engineer	52 open jobs	$149,716 / yr	$121K	$188K

See 6 salaries from all locations (Audience); See 5 salaries from all locations (others)

全国算法工程师薪资平均值约

数据来源于648675份样本，结果仅供参考。 2024年08月16日 23:25 更新

¥37,236/月

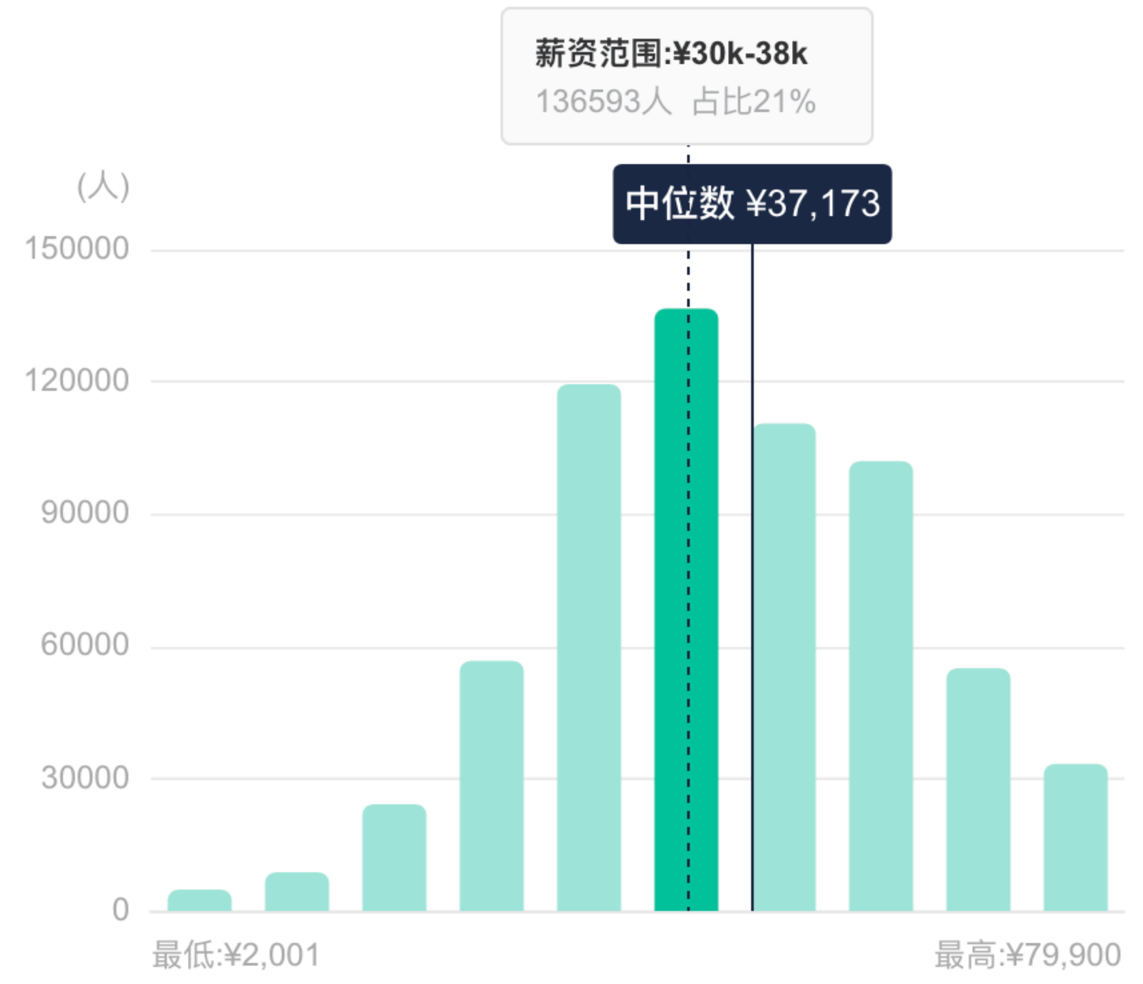
算法工程师薪资详情

kanzhun

经　验： 不限　应届生　1–2年　3–4年　5–6年　7–8年　8年以上

城　市： 全国　北京　上海　深圳　杭州　广州　南京　成都　武汉　苏州　西安　长沙　合肥　厦门　更多

整体分布 | 历年变化



月收入平均值约 **¥37,236**

高于平均值约占 **0%**

月收入中位数 **¥37,173**

近半年趋势 **持平**

解读：算法工程师在全国的平均月薪为¥37,236，中位数为¥37,173，其中¥30k-38k工资占比最多，约21%。

来源于648675份样本

The importance of this course

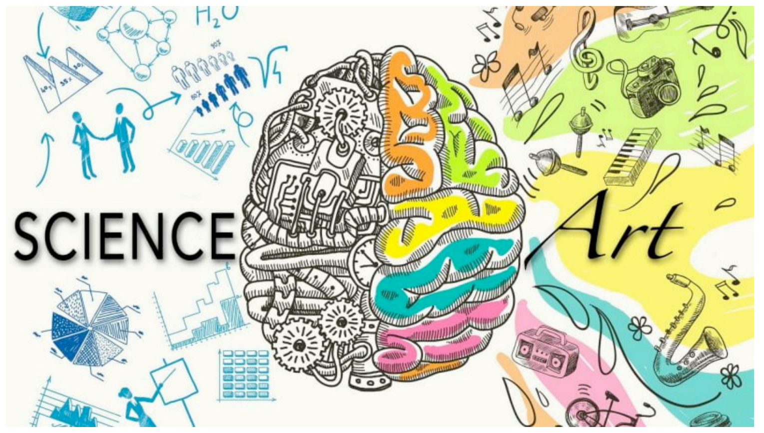
Useful

- 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!

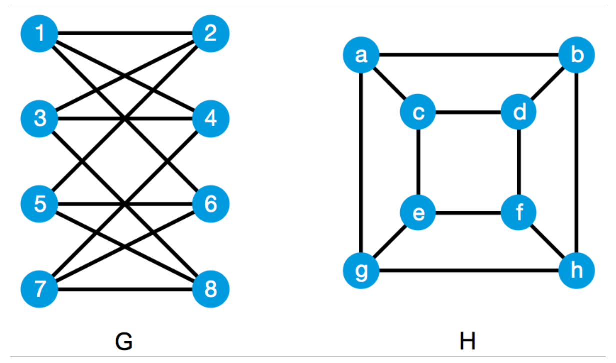
The importance of this course

Last, but not least — Fun



Algorithm design is both an art and a science.

Many surprises!



Many exciting research questions!

Let’s Start

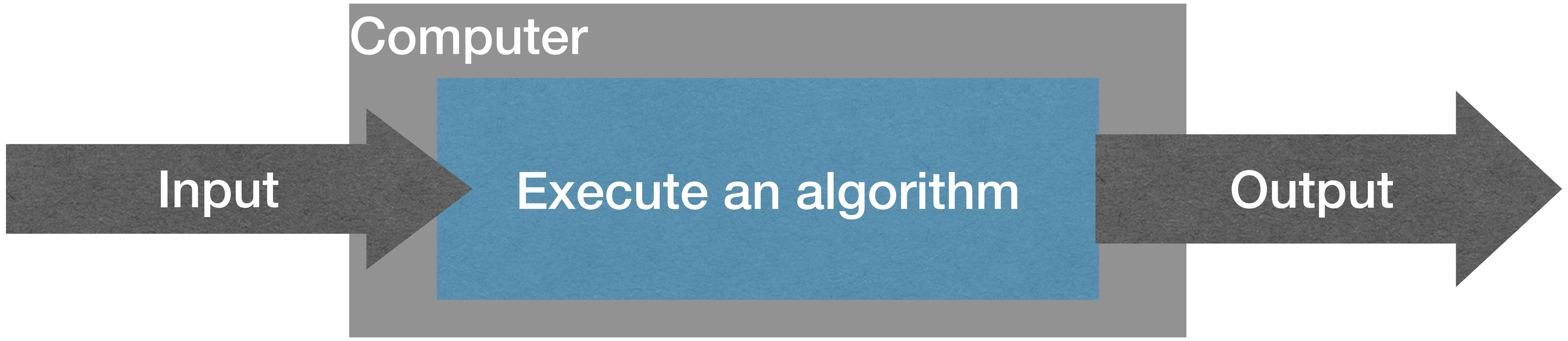
What is an Algorithm?

- 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 $< a_1, a_2, ..., a_n >$
  - Output: a reordering $< a'_1, a'_2, ..., a'_n >$ of input where $a'_1 \le a'_2 \le ... \le a'_n$.

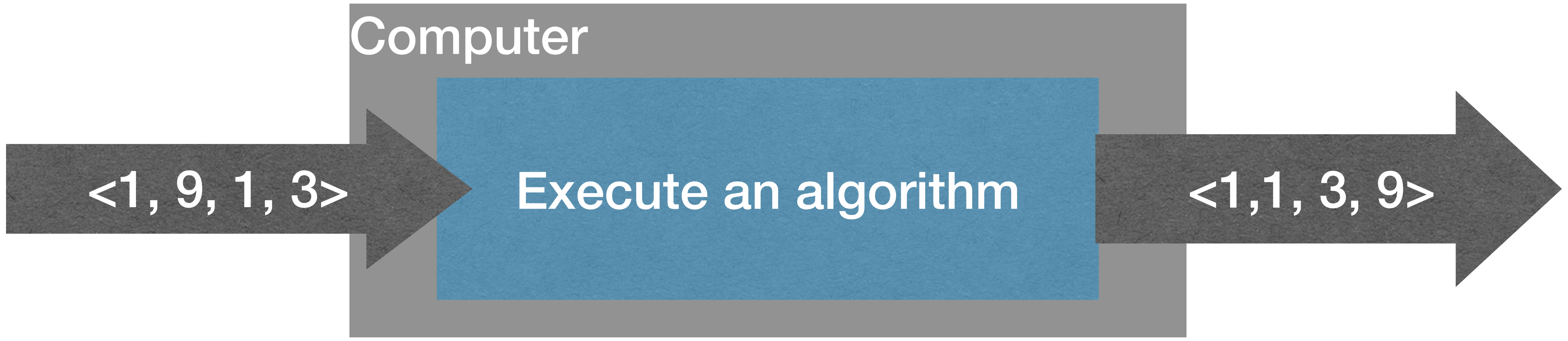
- Step 1 – Set MIN to the first location of $< a_1, a_2, ..., a_n >$
- Step 2 – Search the minimum element from the location MIN to the last location of $< a_1, a_2, ..., a_n >$
- Step 3 – Swap with value at location MIN
- Step 4 – Increment MIN to point to next element
- Step 5 – Repeat the above steps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

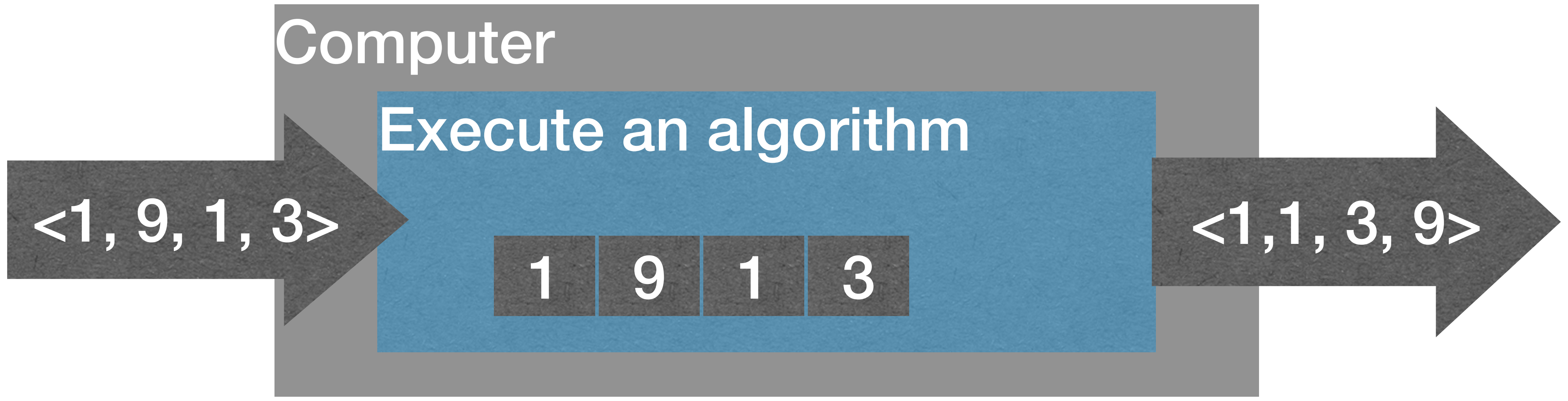
- A particular input of a problem is an instance of that problem.

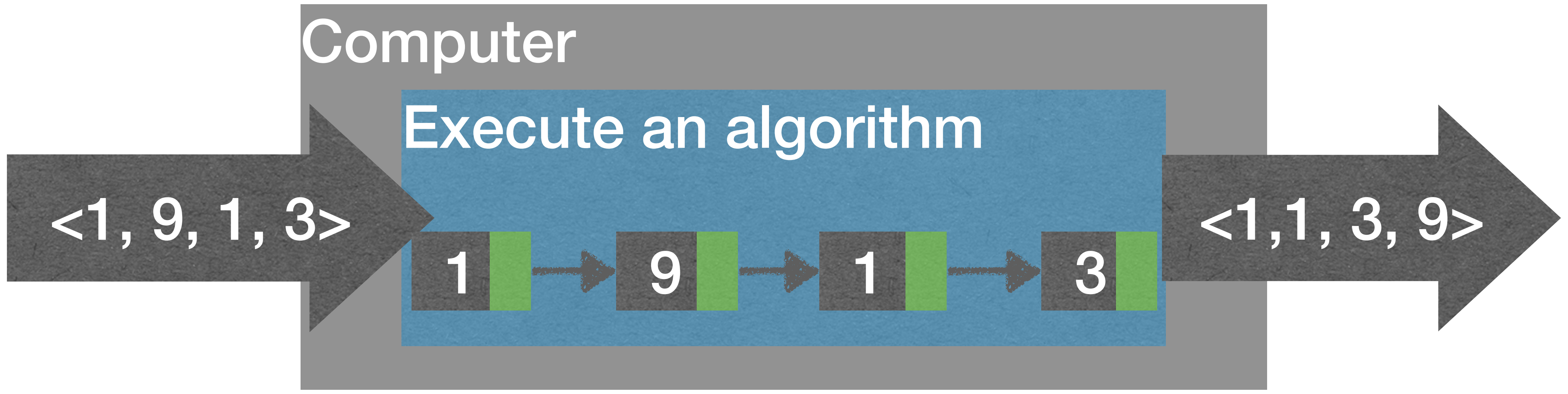
- For example, one instance of ***integer sorting problem***:
  - Sorting the sequence $< 1,9,1,3 >$



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

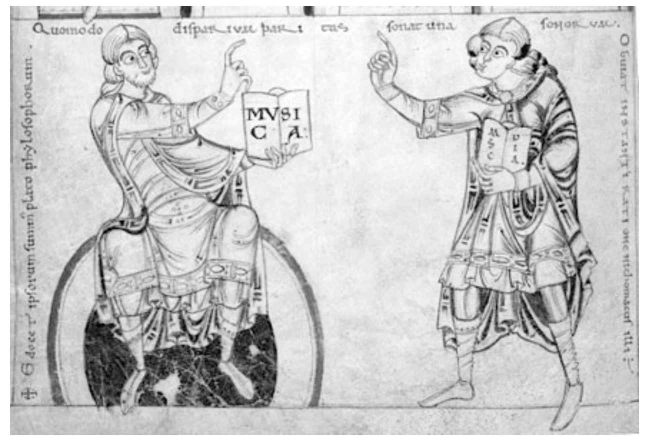
A brief history of Algorithm

300 BC

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

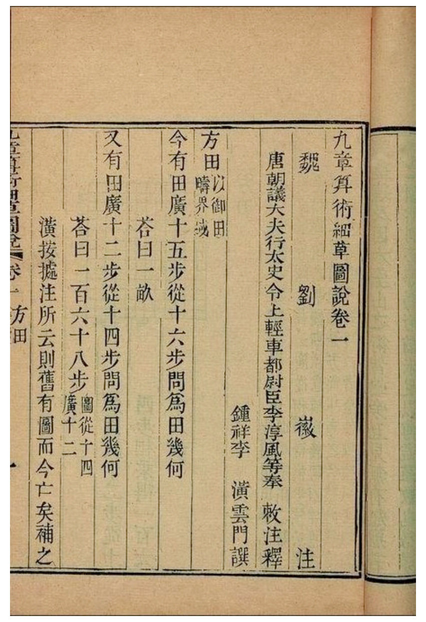
200 BC

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



150BC

高斯消元法（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
AKS primality test
......

- 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:

```
   123
  ×321
 -----
   123
  246
 369
 -----
 39483
```

```
   123
  ×021
 -----
   123
  246
 000
 -----
  2583
```

```
   99
  ×77
 ----
  693
 693
 ----
 7623
```

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 $\leftarrow$ 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 ith 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 10
    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 $\rightarrow$ $2n^2$ additions

Why

- Finally, at most $n^2 + n^2 + 2n^2 = 4 \times n^2$ single digit operations

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.

What if n is an odd number?

- 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:
  - $x = 10^{n/2} \times a + b$
- 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

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

Procedure **RecIntMult(x, y)**

In: two n-digit positive integers x, y //assume n is a power of 2.

Out: the product $p := x \cdot y$

if $n = 1$ **then** // base case

  return $x \cdot y$

else

  $a, b :=$ split x into halves // $x = 10^{n/2} \cdot a + b$

  $c, d :=$ split y into halves

  $u := \textbf{\textit{RecIntMult}}(a, c)$

  $v := \textbf{\textit{RecIntMult}}(b, d)$

  $w := \textbf{\textit{RecIntMult}}(a, d)$

  $t := \textbf{\textit{RecIntMult}}(b, c)$

  $z := w + t$

  $p := 10^n \cdot u + 10^{n/2} \cdot z + v$

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

  - 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 //assume n is a power of 2.

Out: the product $p := x \cdot y$

if $n = 1$ **then** // base case

  return $x \cdot y$

else

  $a, b :=$ split x into halves // $x = 10^{n/2} \cdot a + b$

  $c, d :=$ split y into halves

  $u :=$ ***Karatsuba***(a, c)

  $v :=$ ***Karatsuba***(b, d)

  $w :=$ ***Karatsuba***$(a + b, c + d)$

  $z := w - u - v$

  $p := 10^n \cdot u + 10^{n/2} \cdot z + v$

  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

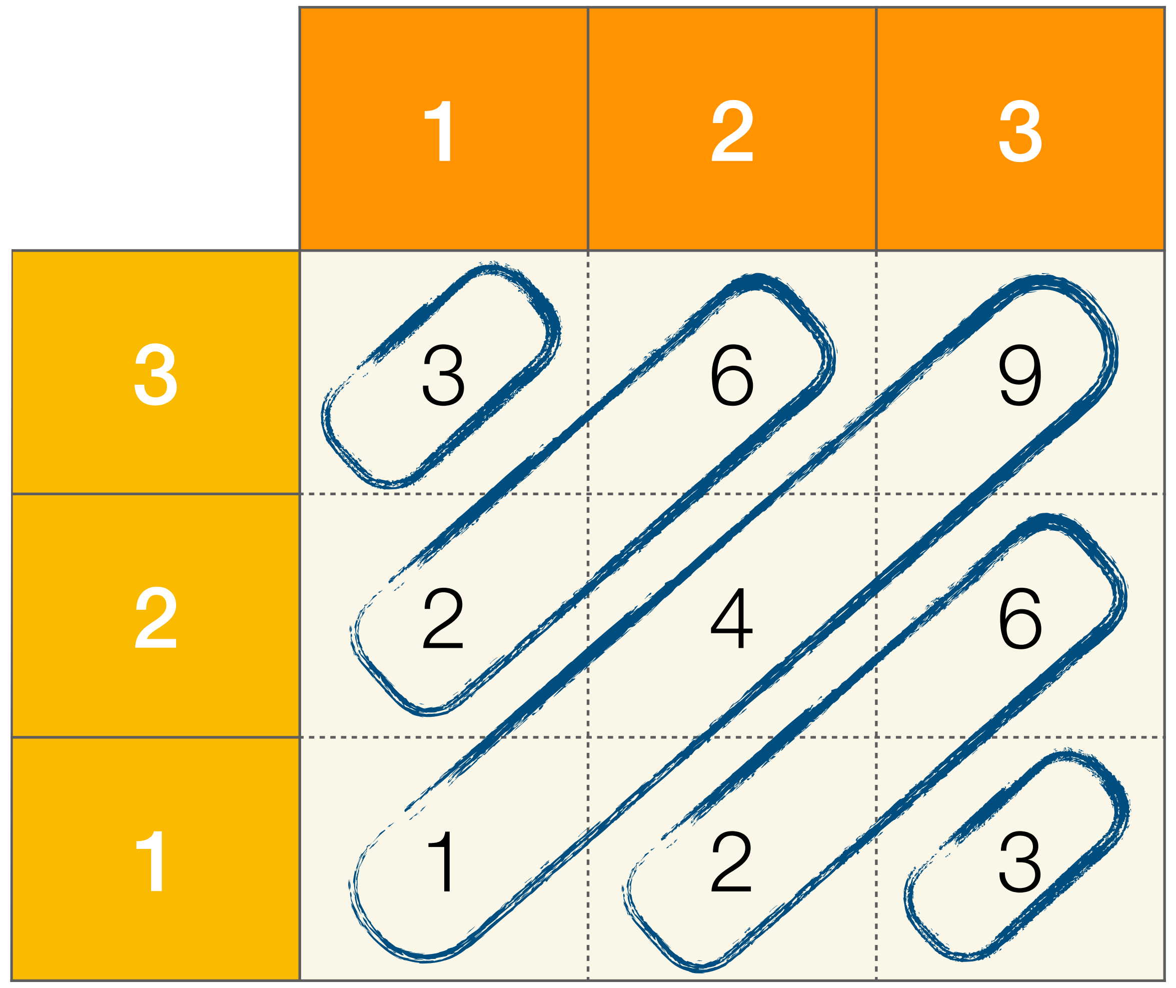
- Schönhage–Strassen (1971): Runs in time $O(n \times \log(n) \times \log(\log(n)))$
- Furer (2007) Runs in time $O(n \times \log(n) \times (2^{O(\log^* n)})$
- Harvey and van der Hoeven (2019) Runs in time $O(n \times \log(n))$

Schönhage–Strassen*

- Intuition：

$$\begin{array}{r} 123 \\ \times 321 \\ \hline 123 \\ 246 \\ 369 \\ \hline 39483 \end{array}$$

Convolution of [1, 2, 3] and [3, 2, 1]



$$\begin{array}{r} 3 \\ 8 \\ 14 \\ 8 \\ 3 \\ \hline 39483 \end{array}$$

$$(x^2 + 2x + 3) * (3x^2 + 2x + 1)$$

coefficients of the polynomial

Schönhage–Strassen*

- Giving $A = (a_1, a_2, \ldots a_n), B = (b_1, b_2, \ldots b_n)$, we need to know the convolution of A and B, that is, the $C = (c_1, c_2, \ldots c_{2n-1})$
- Let $h(x) = c_{2n-1} + c_{2n-2}x + \ldots + c_1 x^{2n-2}$
- Sampling $2n - 1$ points, then we can solve the function to get $c_1, c_2, \ldots c_{2n-1}$
- If the sampling points are a group of complex numbers$\{e^{\frac{2\pi ki}{n}}\}$, this linear function can be very efficiently solved! That is the Fast Fourier Transform (FFT) algorithm!

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.

Halting problem - revisited*

- Halting problem is an undecidable problem, but we can have some approximation algorithms to work in practice.
  - **Bounded Halting Check**: execute a program for a limited number of steps or time, and if the program halts within that bound, it is determined to halt, otherwise returns “unknown”
  - **Ranking function based approach**: maps the state of a program (or loop) to a value, typically a non-negative integer. The key property of a ranking function is that it decreases after each iteration of the loop and is bounded from below (e.g., it cannot go below zero).

Further reading

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