

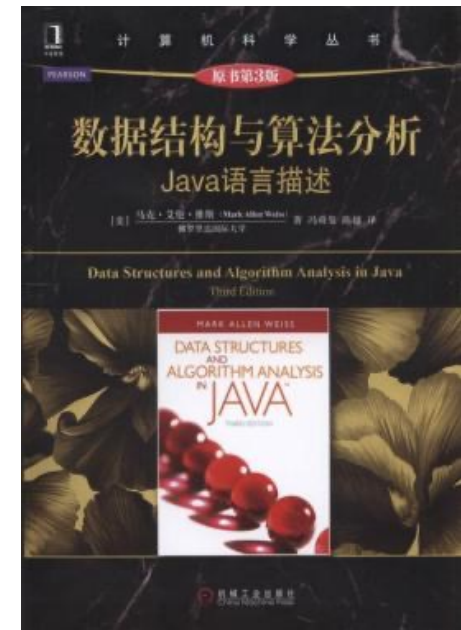
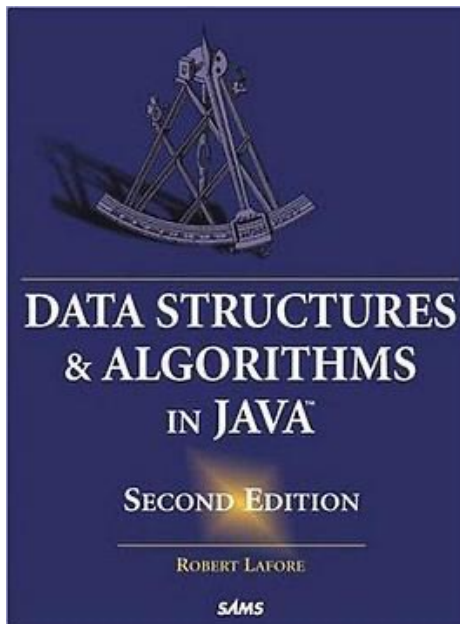
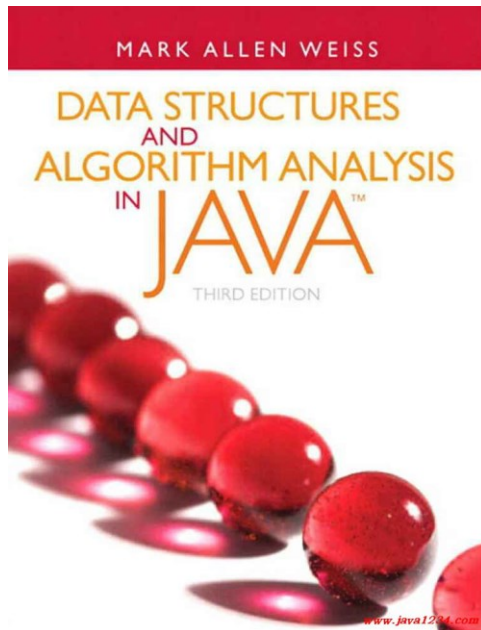
# Recap

现在不是放松的时候，越是离考试近，就越不能放松，不要骄傲自满，沉下心来做学问才会出成绩，你是知道的。好好努力！

- Classes & Labs
- Course content
- Grading
- Tools to be used in this course
- Motivation



# Topic 1 – Intro to Data Structures and Algorithm



# What is a data structure?

- A data structure is a **conceptual** structure for organizing information
- There are many different ways of organizing information
- Different structures have different advantages and disadvantages
- In other words, we will learn to use different structures to store data
- At the same time, we will analyze their advantages and disadvantages

# Data Structure

- What are the **advantages** and **disadvantages** of this data structure?
- Easy to add to
- Difficult to find stuff
- Wasteful of space



# Data Structure

- What are the **advantages** and **disadvantages** of this data structure?
- Easy to find stuff
- Easy to maintain
- Laborious to add stuff
- Difficult to expand

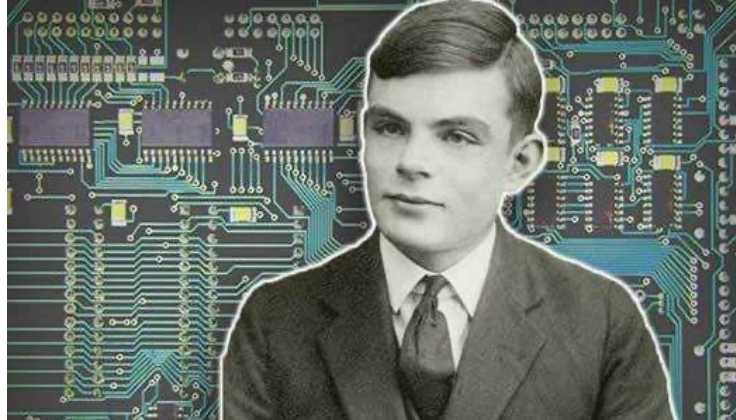


# What is an algorithm?

- An algorithm is a precise **step-by-step** plan for solving a problem using a **finite sequence of instructions**
- But how do we know if a set of instructions is comprehensive?
- It proved extremely difficult for mathematicians to pin down a precise definition for 'algorithm'

# What is an algorithm?

- Alan Turing finally managed to provide a comprehensive definition for 'algorithm' in the 1930s: **a program that can run on a Turing machine**



# Algorithm

- An algorithm is:
  - A well-defined, step-by-step procedure
  - That is guaranteed to terminate
- There are literally thousands of published algorithms
- We will cover some of the algorithms from the textbook
- It's almost always better to find an existing algorithm than to re-invent it yourself
  - Always reference your sources



# Analysis of algorithms

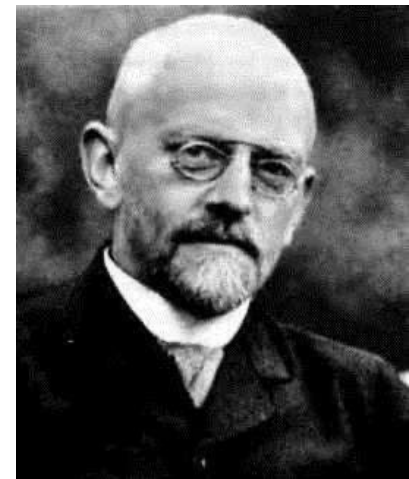
- Analysis of algorithms is a relatively small part of this course, but it's an important part
  - Most of the time, programming efficiency is the *wrong goal*—you should be making your programs clearer, not faster
  - But sometimes your program is too slow, or it is part of an application where speed is critical
  - Besides, using well-known algorithms often make your program easier to understand

# Analysis of algorithms

- Analysis can tell you how fast an algorithm will run, and how much space it will require
- When speed is important, a proper choice of algorithm is essential
  - Coding details matter almost not at all

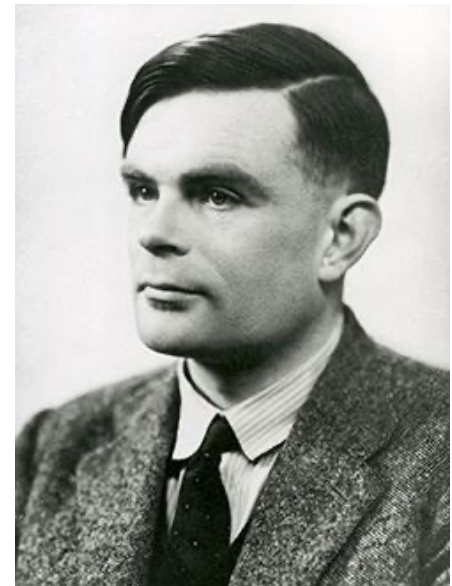
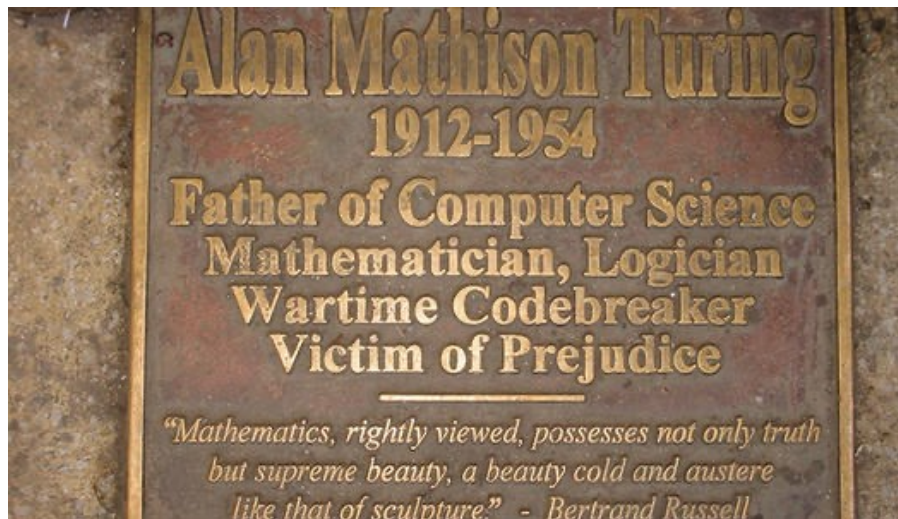
# Hilbert's Entscheidungsproblem

- David Hilbert set up a research program in the 1920s to define the foundations of mathematics
- He wondered whether **there was an algorithm that could decide whether any statement was true or false**
- This is called the *Entscheidungsproblem* (German for decision problem)
- Hilbert believed there was no such thing as an unsolvable problem in math
  - “We must know, we shall know”



# Alan Turing (1912 – 1954)

- Alan Turing was an English mathematician and logician who is now considered the father of computer science
- In school Turing fell in love with an older male student whose subsequent death shattered his belief in God and turned him into an atheist
- He was elected as a fellow in Cambridge based on the strength of his final year dissertation



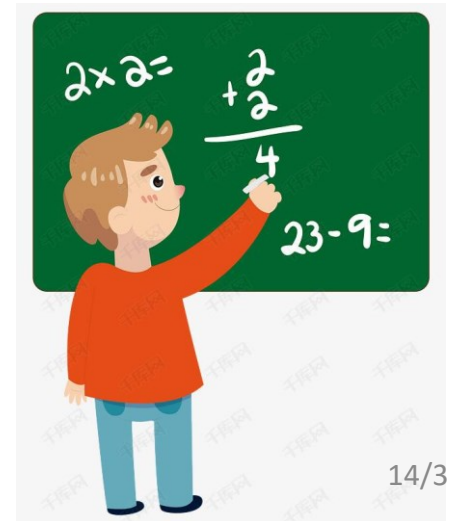
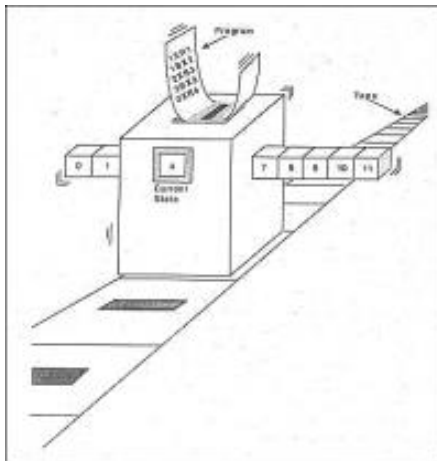
# What is an algorithm?

- Alan Turing wanted to prove that no algorithm could ever resolve the *Entscheidungsproblem*
- But what exactly is an algorithm?
- Turing observed some computers at work
- At the time women were often employed as **‘computers’** as they represented a source of cheap labour and had mathematical training



# Computers at work

- The '**computer**' makes marks on a sheet of paper
- She shifts her attention from what she had written earlier to what she is writing now
- She keeps a small amount of information in her working memory
- Turing wanted to strip out all of the detail such as whether the computer is using a pencil or a pen

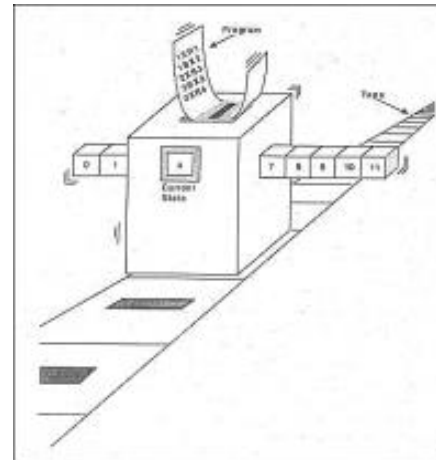
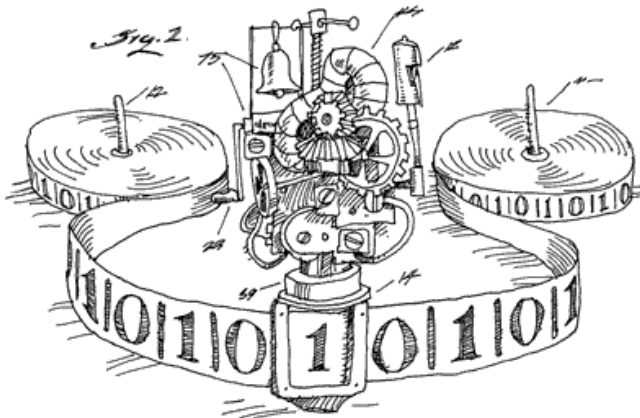


# Turing Machine

- Alan Turing's greatest contribution was in coming up with a model of computation that provides a complete description of all of the physical steps involved in following an algorithm
- Turing's model of computation is now called a Turing machine (he called them a-machines)
- Turing accounted for **every single step down to the manipulation of the smallest piece of information**
- In doing so he had managed to reconcile the field of logic with mechanical processes occurring in the physical world

# Turing Machine

- It is important to note that a Turing Machine is just a thought experiment involving an abstract model of computation
- It has an infinitely long tape and a read/write head that can move along the tape changing the values along the way
- This simplistic theoretical machine can be used to simulate the computation of any algorithm





# Halting

- Turing realised that for every algorithm that is running there are only two possible outcomes
  - Either the algorithm will terminate at some stage (called **halting**)
  - Or else it will run forever
  - Obviously, it has to be one or the other!

## Halts

```
for(int i = 0; i<10; i++){  
    System.out.println("Still looping...");  
}
```

## Loops forever

```
while(true==true){  
    System.out.println("Still looping...");  
}
```

- But from just looking at an algorithm is it possible to predict in advance which category it will fall into?

# How Turing did it

- Turing realised that he could use the halting / looping distinction to expose the limitations of Turing machines
- **He proved the *Entscheidungsproblem* was unsolvable** by showing that you can seize up any Turing machine by feeding it a description of itself and giving it the following instructions:
  - *Computer,*
  - *Take this description of an algorithm and check to see if it halts or loops forever. If it halts at some point then go into an infinite loop. However, if you find that it loops forever then halt.*




# Feed the computer itself...



*“10101001010001  
111010100101000  
100111011011001  
010010101001011  
0101010...”.*

# Turing's proof of halting problem

- Assume there exists Oracle that knows a TM (K) will halt or not
  - $X(Y)$ 
    - takes an input program Y
    - Asks the Oracle whether the input program Y halts or not
    - If Y halts  $\Rightarrow$  X loops forever
    - If Y loops forever  $\Rightarrow$  X halts
  - If feed X itself to X:  $X(X)$ 
    - Oracle will predict the input X halts/loops
    - If predicted X halts, X loops
    - If predicted X loops, X halts
  - The results is contradictory, therefore, the assumption is invalid
- 

# Turing's influence



- Turing's efforts to solve Hilbert's *Entscheidungsproblem* led to several breakthroughs
  1. Turing had to specify a model of computation (called the Turing machine) to allow the idea of an **algorithm** to be comprehensively defined for the first time
  2. To feed the machine itself, Turing had to show that machines, programs and data could all be represented as input data
  3. As a result, Turing realised that it is possible to have one Turing machine which is capable of simulating all other machines (**the Universal Turing Machine**)
  4. By showing that no algorithm can solve the *Entscheidungsproblem*, Turing showed that there are some questions which are **unsolvable**

# Machine, Program, and Data

- Before Turing it was assumed that the machine, program and data were all different
- Turing's model allowed data, algorithm and machine to be treated as abstract patterns
- Computers represent patterns of information using binary code
  - 11011010110101
- The Church-Turing thesis speculates that the entire universe can be simulated by a Universal Turing Machine



# Incomputability

- So, **not all problems can be solved by an algorithm**
  - In fact, there are infinitely more problems than there are algorithms to solve them
- We will study problems that can be solved by algorithms in reasonable amounts of time
- Next year you will study problems that cannot be solved, in the module - Theory of Computation

# Summary

- **Algorithm:** A step-by-step procedure for solving a problem via a computational process
- **Program:** An implementation of an algorithm in some programming languages
- **Data Structure:** A conceptual system for organizing the data needed to solve some problems

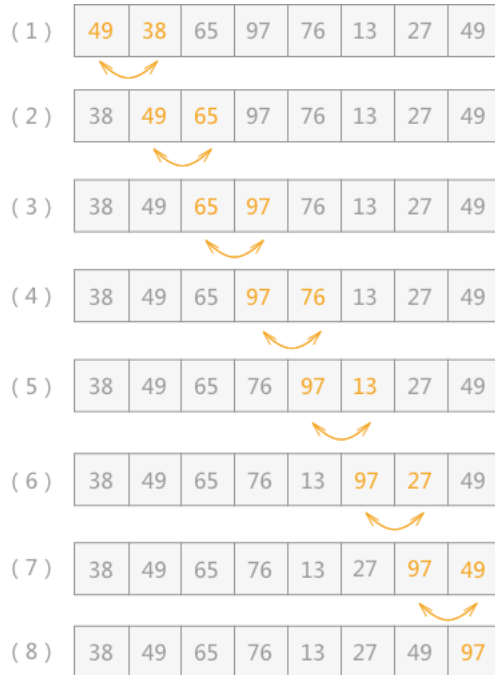


# Summary

- Algorithm is like your idea, and program is its implementation.



## Algorithm



## Program

```
public class BubbleSort {  
    public void bubbleSort(Integer[] arr, int  
n) {  
        if (n <= 1) return;  
        for (int i = 0; i < n; ++i) {  
            boolean flag = false;  
            for (int j = 0; j < n - i - 1; ++j) {  
                if (arr[j] > arr[j + 1]) {  
                    int temp = arr[j];  
                    arr[j] = arr[j + 1];  
                    arr[j + 1] = temp;  
                    flag = true;  
                }  
            }  
            if (!flag) break;  
        }  
        public static void main(String[] args) {  
            Integer arr[] = {2, 4, 7, 6, 8, 5, 9};  
            SortUtil.show(arr);  
            BubbleSort bubbleSort = new  
BubbleSort();  
            bubbleSort.bubbleSort(arr, arr.length);  
            SortUtil.show(arr);  
        }  
    }  
}
```

# Example: Luhn's algorithm



- Also known as the “mod 10” algorithm,
  - a checksum formula used to validate credit card numbers
1. From the rightmost digit, which is the check digit, and moving left, double the value of every second digit. If the result of this doubling operation is greater than 9 (e.g.,  $8 \times 2 = 16$ ), then add the digits of the product (e.g., 16:  $1 + 6 = 7$ , 18:  $1 + 8 = 9$ ) or alternatively subtract 9 from the product (e.g., 16:  $16 - 9 = 7$ , 18:  $18 - 9 = 9$ ).
  2. Take the sum of all the digits.
  3. If the total modulo 10 is equal to 0 (if the total ends in zero) then the number is valid according to the Luhn formula; else it is not valid.

# Luhn's algorithm



- Assume an example of an account number "79927398713":

7	9	9	2	7	3	9	8	7	1	3
	x2		x2		x2		x2		x2	
	18		4		6		16		2	
7	9	9	4	7	6	9	7	7	2	3

$$7 + 9 + 9 + 4 + 7 + 6 + 9 + 7 + 7 + 2 + 3 = 70$$

# Luhn's algorithm



- The algorithm was created by IBM scientist Hans Peter Luhn and filed in a U.S. patent on January 6, 1954
- The Luhn algorithm will detect any single-digit error, as well as almost all transpositions of adjacent digits.
- It will not, however, detect transposition of the two-digit sequence 09 to 90 (or vice versa). It will detect twin errors except  $22 \leftrightarrow 55$ ,  $33 \leftrightarrow 66$  and  $44 \leftrightarrow 77$

# Luhn's algorithm



- Other, more complex check-digit algorithms (such as the Verhoeff algorithm and the Damm algorithm) can detect more transcription errors
- Because the algorithm operates on the digits in a right-to-left manner and zero digits affect the result only if they cause shift in position, zero-padding the beginning of a string of numbers does not affect the calculation. Therefore, systems that pad to a specific number of digits (by converting 1234 to 0001234 for instance) can perform Luhn validation before or after the padding and achieve the same result.

Quiz!



