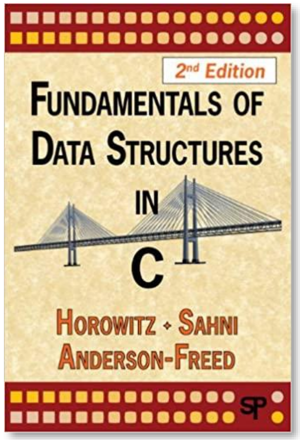
5118006-03 Data Structure

Algorithm Analysis

17 May 2024

Shin Hong

Ch. 1.3.1 Algorithm specification
Ch. 1.5.2 Time Complexity



Problem

- In many domains, there are **key general problems** that ask for output with specific properties when a valid input is given
  - E.g., sorting, searching

- Computing is definiting a generalized solution to a class of problems
  - Computing vs. calculation
  - Programming
    - State precisely the general problem by specifying the input and the desired output, using the appropriate structures
    - Specify the steps of a procedure that takes a valid input and produces the desired output.

George Boole

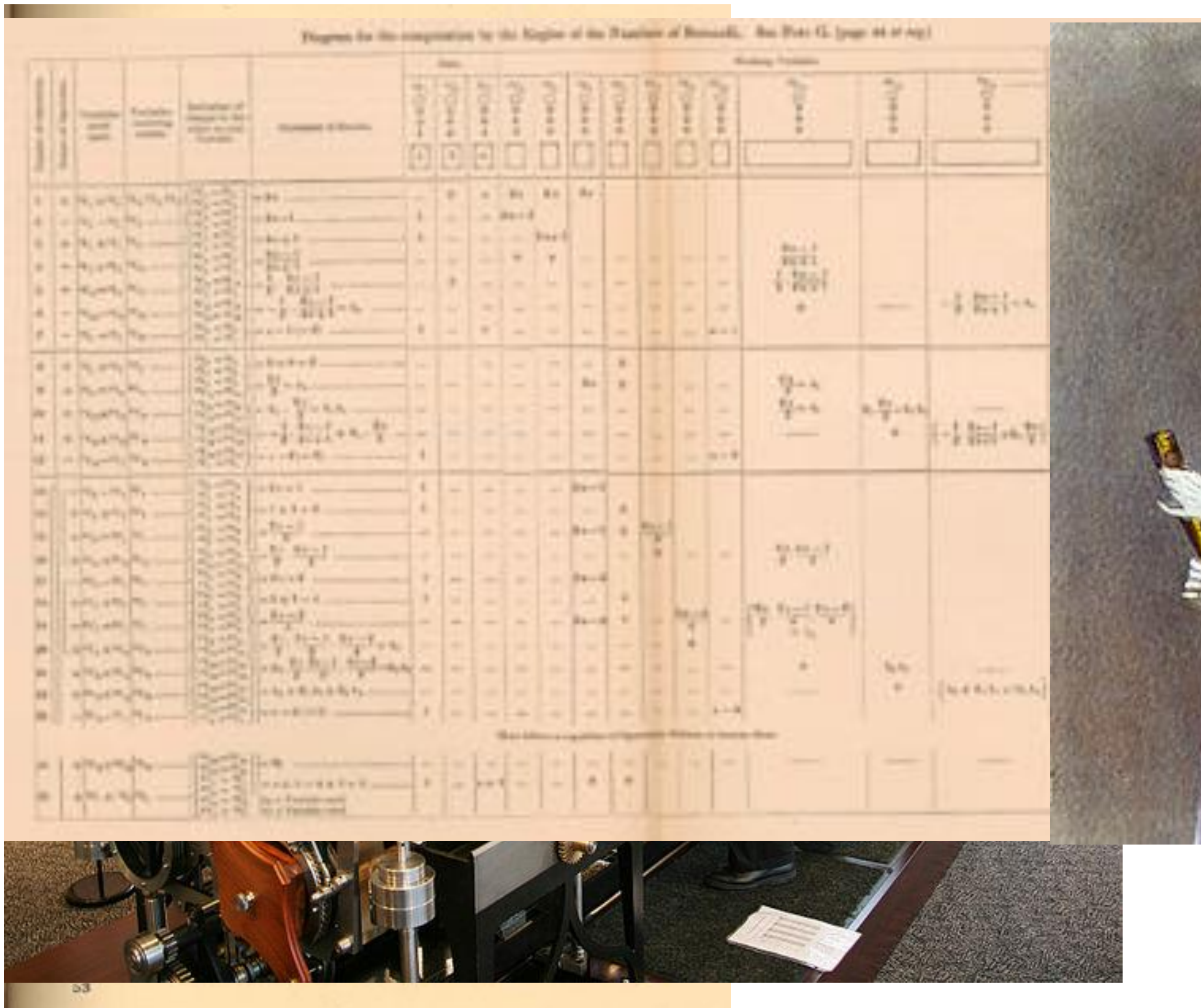
George Boole (1815--1864)

Formulate a calculus of reasoning

- Claim that logic should be considered as a branch of math, rather than a part of philosophy
- Argue that there are math laws to express the operation of human mind
- Showed that Aristotle's syllogistic logic could be rendered as algebraic equitation

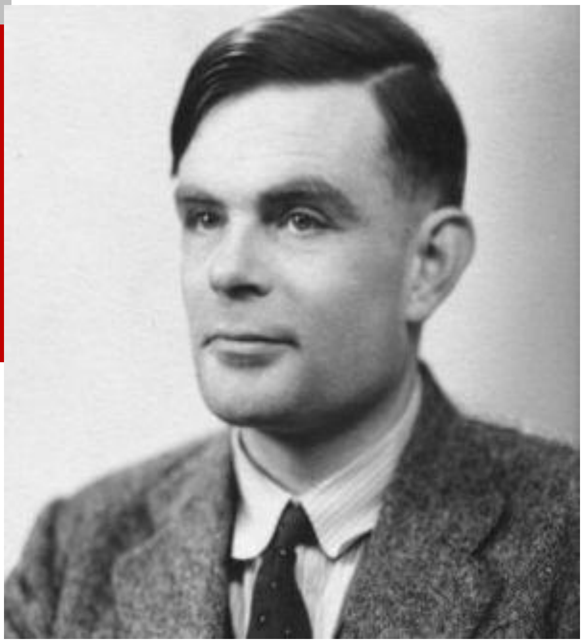
A Brief History of Computing by G. O'Regan

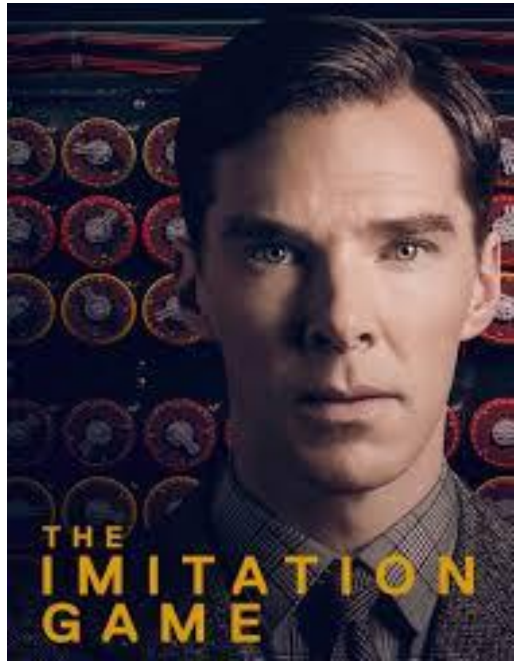
Charles Babbage

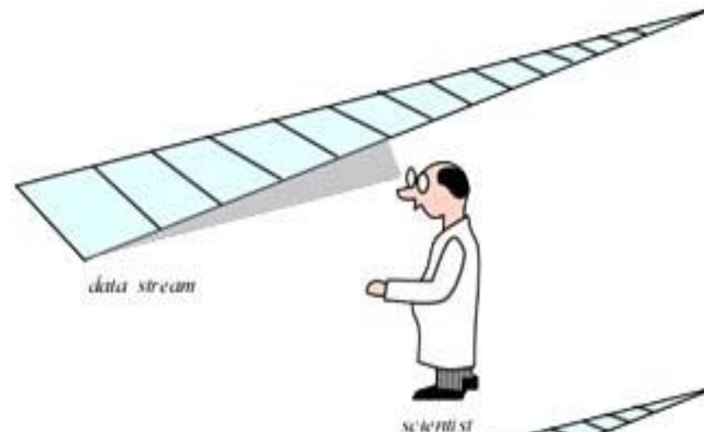


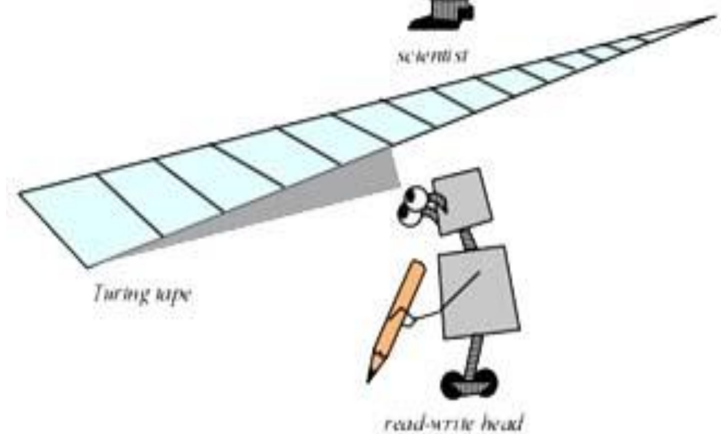
ods to

puter

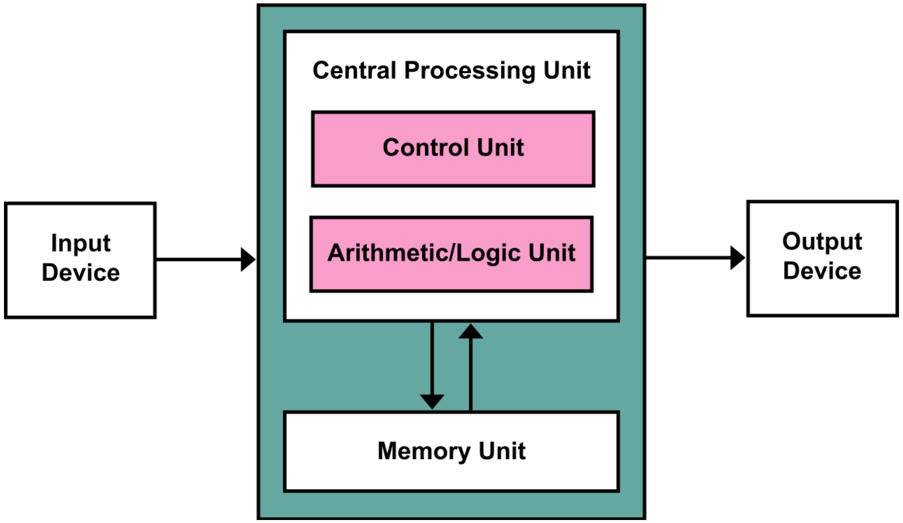



THE
IMITATION
GAME


data stream
scientist


Turing tape
read-write head

Moderm Computer Model



John von N
(1903— 19

- Memory is a map from addresses to values
- A value is either a number or instruction
  - number
  - instruction
    - receive an input
    - produce an output
    - evaluate an expression over memory addresses
    - assign a value to a memory address
    - jump to a memory address
    - finish
- A processor loads and executes instructions from address 0

Algorithm

- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task

- A procedure is an algorithm when it satisfieds the following conditions:
  - **Input**: data is externally supplied to the program
  - **Output**: result is be produced externally
  - **Definiteness**: each instruction is clearly defined
  - **Effectiveness**: each instruction can be performed easily
  - **Finiteness**: for all case, the algorithm must be terminated within a finite step of instruction execution
  - **Generality**: should work for all problems of a certain kind
  - **Correctness**: should produce the correct output for every input

Algorithm Representation (1/2)

- Example : selection sort

- In natural language

  From those integers that are currently unsorted, find the smallest and place it next in the sorted list

- In program code

```
#include <stdio.h>
#include <math.h>
#define MAX_SIZE 101
#define SWAP(x,y,t) ((t) = (x), (x)= (y), (y) = (t))
void sort(int [],int); /*selection sort */
void main(void)
{
   int i,n;
   int list[MAX_SIZE];
   printf("Enter the number of numbers to generate: ");
   scanf("%d",&n);
   if( n < 1 || n > MAX_SIZE) {
     fprintf(stderr, "Improper value of n\n");
     exit(EXIT_FAILURE);
   }
   for (i = 0; i < n; i++) {/*randomly generate numbers*/
      list[i] = rand() % 1000;
      printf("%d  ",list[i]);
   }
   sort(list,n);
   printf("\n Sorted array:\n ");
   for (i = 0; i < n; i++) /* print out sorted numbers */
      printf("%d  ",list[i]);
   printf("\n");
}
```

```
void sort(int list[],int n)
{
   int i, j, min, temp;
   for (i = 0; i < n-1; i++)  {
      min = i;
      for (j = i+1; j < n; j++)
         if (list[j] < list[min])
            min = j;
      SWAP(list[i],list[min],temp);
   }
}
```

Algorithm Representation (2/2)

- In pseudocode

```
Input    list[0..n-1]: a list of n integers
Output list[0..n-1]: a sorted list of the integers
Procedure
for each i = [0.. n-1] begin
   examine list[i] to list[n-1] and find the smallest one as
      list[min] ;
   interchange list[i] and list[min] ;
end
```

Specifying Algorithms in Pseudocode

- Pseudocode is an intermediate step between natural language description and code using a specific programming language
- The form of pseudocode is similar with C++ and Java.
- Programmers can use the description of an algorithm in pseudocode to construct a program in a particular language
- Pseudocode helps us analyze the time required to solve a problem using an algorithm, independent of the actual programming language used to implement algorithm

Performance Analysis

- Quality criteria of program
  - Does the program meet the specifications of the task?
  - Does it work correctly?
  - Is the source code of the program is readable?
  - Does the program have a well-modularized structure?
  - Is the program's running time acceptable for the task?

- The time complexity of a program is the amount of computer time that it needs to run to completion
  - Execution time vs. Time complexity

The Growth of Functions

- In both computer science and in mathematics, there are many times when we care about how fast a function grows.

- In computer science, we want to understand how quickly an algorithm can solve a problem as the size of the input grows.
  - we can compare the efficiency of two different algorithms for solving the same problem
  - we can also determine whether it is practical to use a particular algorithm as the input grows.

Big-*O* Notation (1/3)

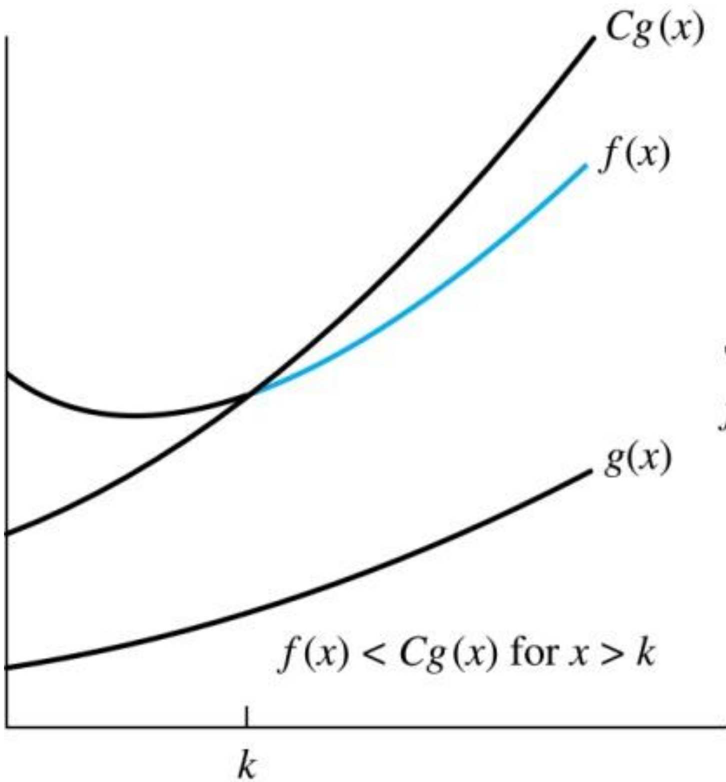
- Let *f* and *g* be functions from the set of real numbers to the set of real numbers. We say that *f*(*x*) is *O*(*g*(*x*)) if there are c onstants *C* and *k* such that

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$. (illustration on next slide)

- This is read as "*f*(*x*) is big-*O* of *g*(*x*)" or "*g* asymptotically do minates *f*."
- The constants C and k are called *witnesses* to the relationsh ip *f*(*x*) is *O*(*g*(*x*)). Only one pair of witnesses is needed.

Big-O Notation (2/3)



$f(x)$ is $O(g(x))$

The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

Big-*O* Notation (3/3)

- If one pair of witnesses is found, then there are infi nitely many pairs
  - We can always make the *k* or the *C* larger and still maintain the i nequality $|f(x)| \leq C|g(x)|$
  - Any pair *C*' and *k*′ where *C* < *C*′ and *k* < *k*' is also a pair of witness es since $|f(x)| \leq C|g(x) \leq C'|g(x)|$ whenever *x* > *k*′ > *k*.

- Usually, we will drop the absolute value sign since we will always deal with functions that take on pos itive values.

Using the Definition of Big-*O* Notation

Example: Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$

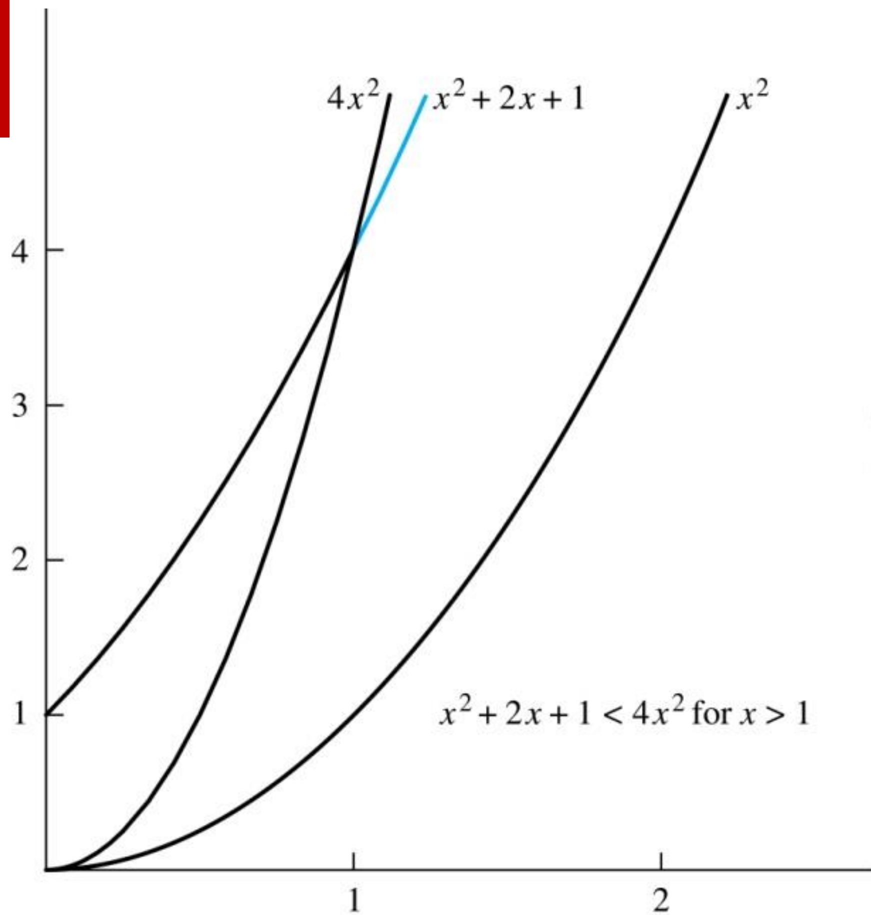
Solution: Since when $x > 1$, $x < x^2$ and $1 < x^2$

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

- Can take $C = 4$ and $k = 1$ as witnesses to show that $f(x)$ is $O(x^2)$

- Alternatively, when $x > 2$, we have $2x \leq x^2$ and $1 < x^2$. Hence, $0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$ when $x > 2$.
  - Can take $C = 3$ and $k = 2$ as witnesses instead.

Illustration of Big-O Notation



$$f(x) = x^2 + 2x + 1$$

is $O(x^2)$

The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in blue.

Big-*O* Notation

- When both $f(x) = x^2 + 2x + 1$ and $g(x) = x^2$ are such that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$, two functions are of the same order

- If $f(x)$ is $O(g(x))$ and *h(x)* is larger than *g(x)* for all positive real numbers, $f(x)$ is $O\big(h(x)\big)$

- If $|f(x)| \le C|g(x)|$ for *k* < *x* and if $|g(x)| < |h(x)|$ for all *x*, $|f(x)| \le C|h(x)|$ if *k* < *x*. Hence, $f(x)$ is $O(h(x))$

- For many applications, the goal is to select the function *g*(*x*) in *O(g(x))* as small (tight) as possible (up to multiplication by a constant, of course)

Using the Definition of Big-*O* Notation

- **Example**: Show that $7x^2$ is $O(x^3)$.
- **Solution**: When $x > 7$, $7x^2 < x^3$. Take $C = 1$ and $k = 7$ as witnesses to establish that $7x^2$ is $O(x^3)$

- **Example**: Show that n^2 is not $O(n)$
- **Solution**: Suppose there are constants C and k for which $n^2 \leq Cn$, whenever $n > k$. Then (by dividing b oth sides of $n^2 \leq Cn$) by n, then $n \leq C$ must hold fo r all $n > k$. A contradiction!

Big-O Estimates for Polynomials

Example: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_o$ where $a_0, a_1, \ldots, a_n$ are real numbers with $a_n \neq 0$. Then $f(x)$ is $O(x^n)$.

Proof: $|f(x)| = |a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_1|$

$$\begin{aligned}
&\leq |a_n| x^n + |a_{n-1}| x^{n-1} + \cdots + |a_1| x^1 + |a_1| \\
&= x^n (|a_n| + |a_{n-1}| / x + \cdots + |a_1| / x^{n-1} + |a_1| / x^n) \\
&\leq x^n (|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_1|)
\end{aligned}$$

- Take $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_1|$ and $k = 1$. Then $f(x)$ is $O(x^n)$.
- The leading term $a_n x^n$ of a polynomial dominates its growth.