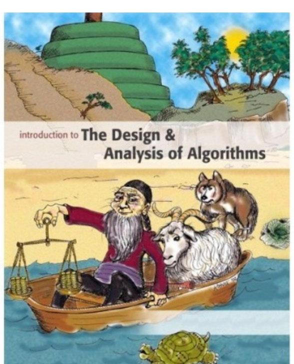
Introduction to

Algorithm Design and Analysis

[1] Model of Computation



Yu Huang

http://cs.nju.edu.cn/yuhuang
Institute of Computer Software
Nanjing University

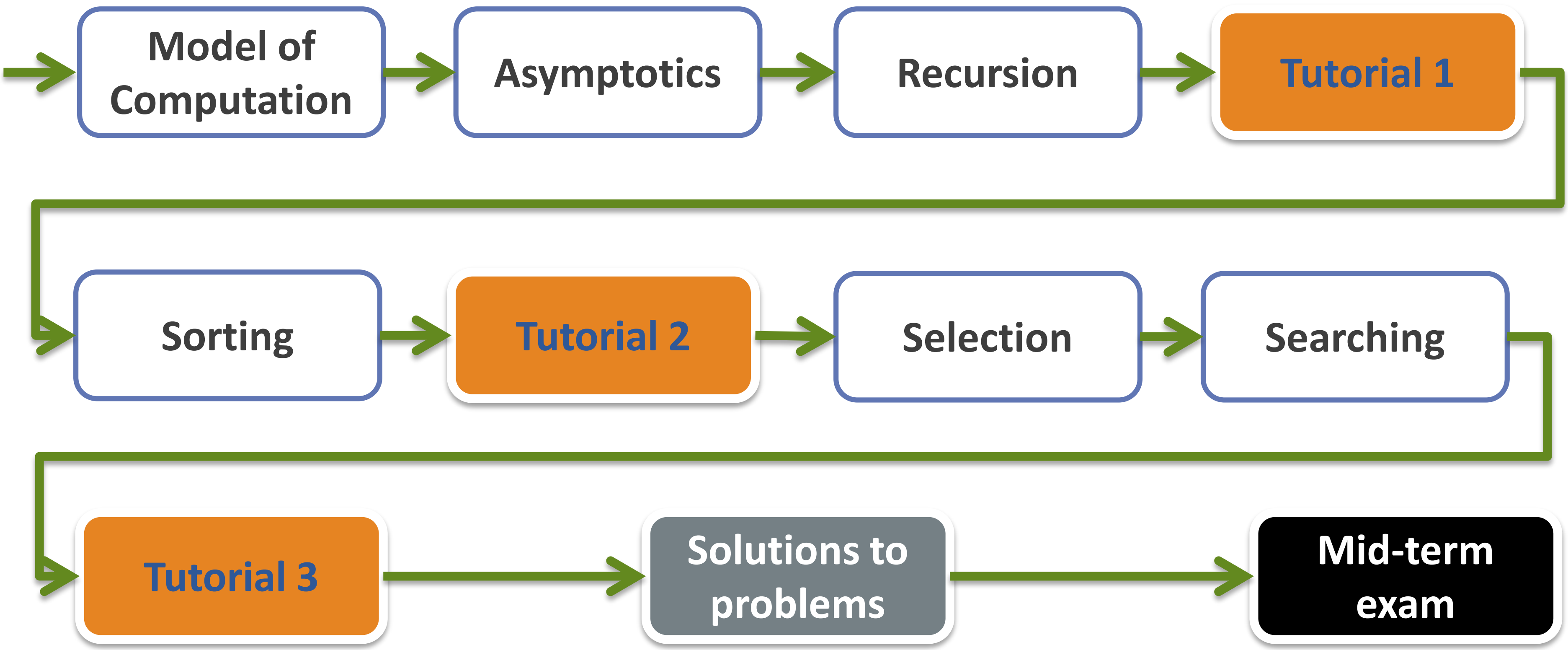
Course Information

- Syllabus
- Textbook
- Website

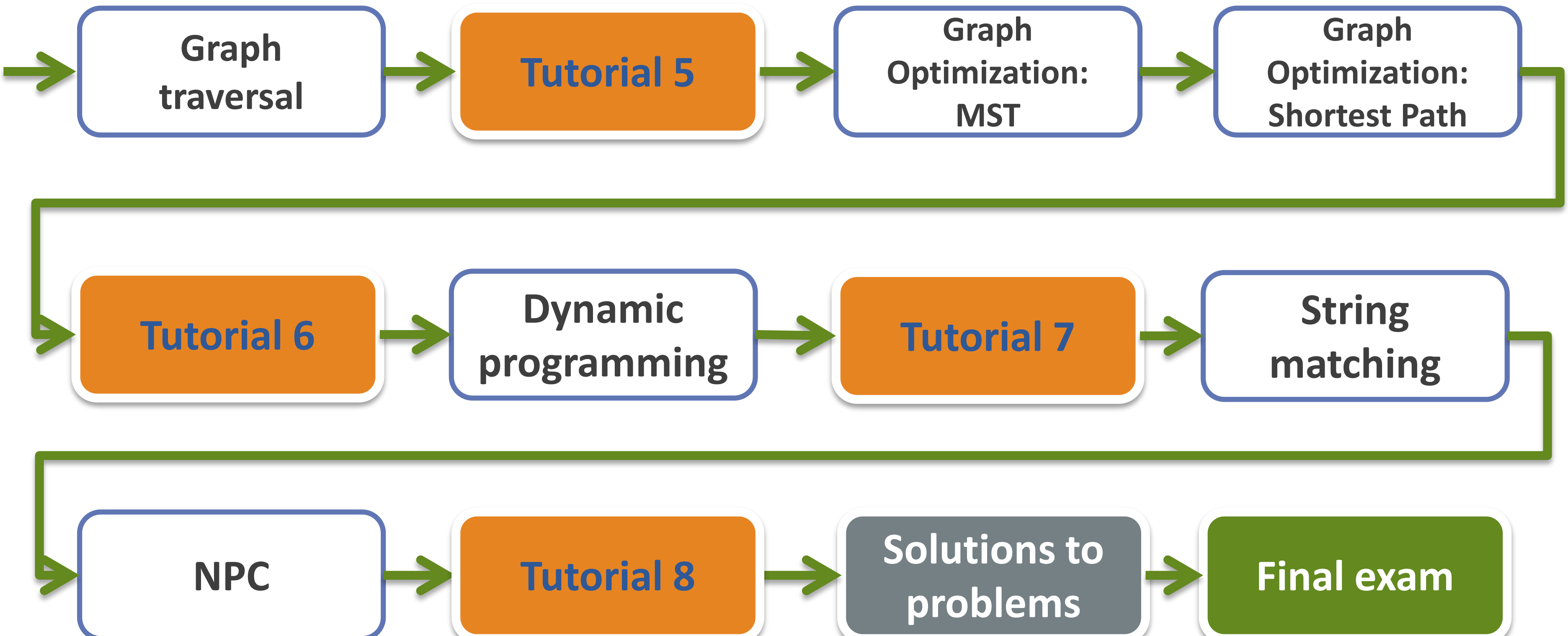
Syllabus

- Model of Computation
- Algorithm design & analysis techniques
- Computation complexity

Syllabus

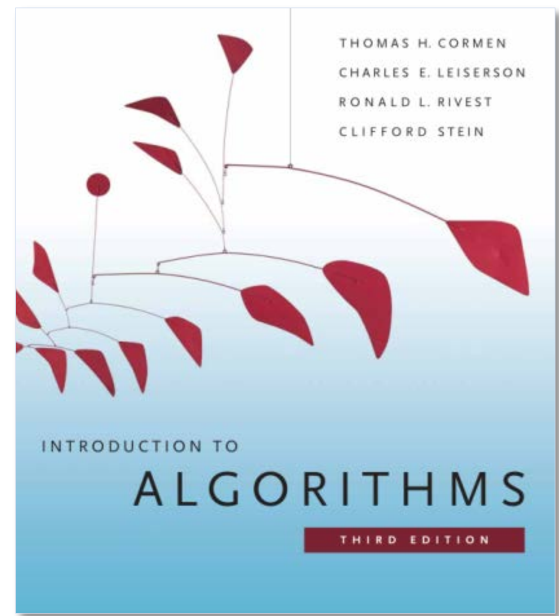

Model of Computation
Asymptotics
Recursion
Tutorial 1
Sorting
Tutorial 2
Selection
Searching
Tutorial 3
Solutions to problems
Mid-term exam

Syllabus


Graph traversal
Tutorial 5
Graph Optimization: MST
Graph Optimization: Shortest Path
Tutorial 6
Dynamic programming
Tutorial 7
String matching
NPC
Tutorial 8
Solutions to problems
Final exam

Textbooks

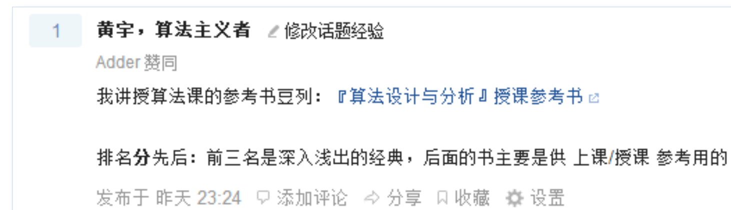
- **Course outline: LADA**
  - **L**ectures on **A**lgorithm **D**esign & **A**nalysis (slides)
- **Course contents**
  - Introduction to Algorithms (CLRS)

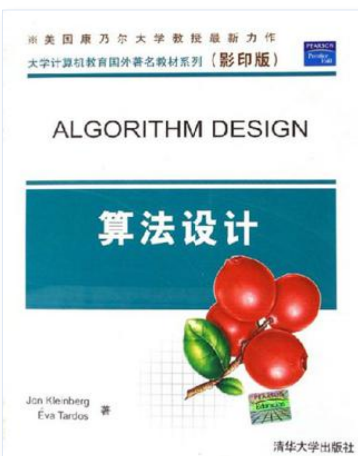


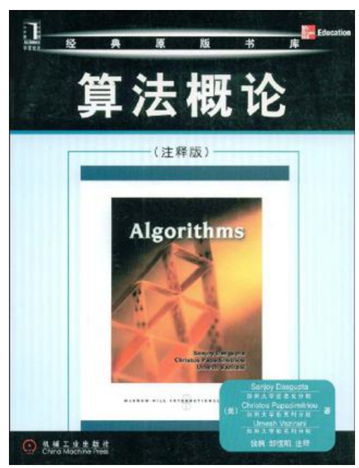
Textbooks

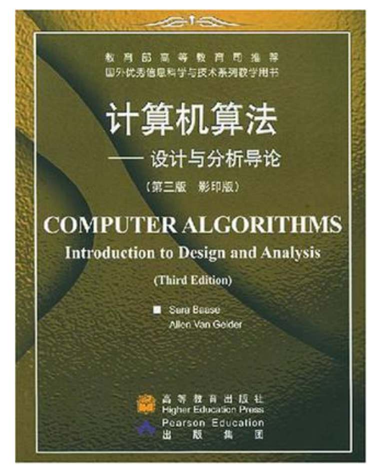
- **Further reading**
  - o Algorithms
  - o Algorithm Design
  - o Computer Algorithms*

See the "douban list" for more info:
http://book.douban.com/doulist/1155824/



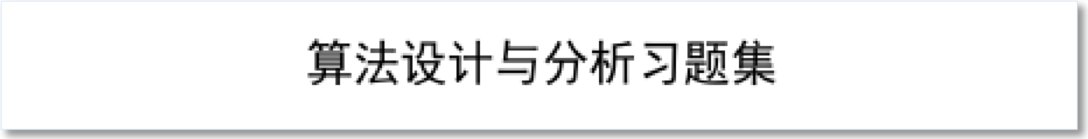




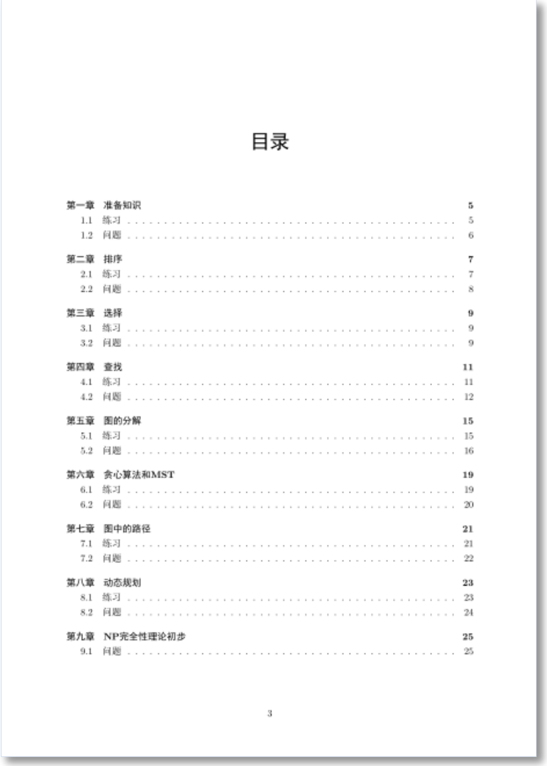


Problem Sets

- **Exercises**
  - Course contents
- **Problems**
  - Problem solving

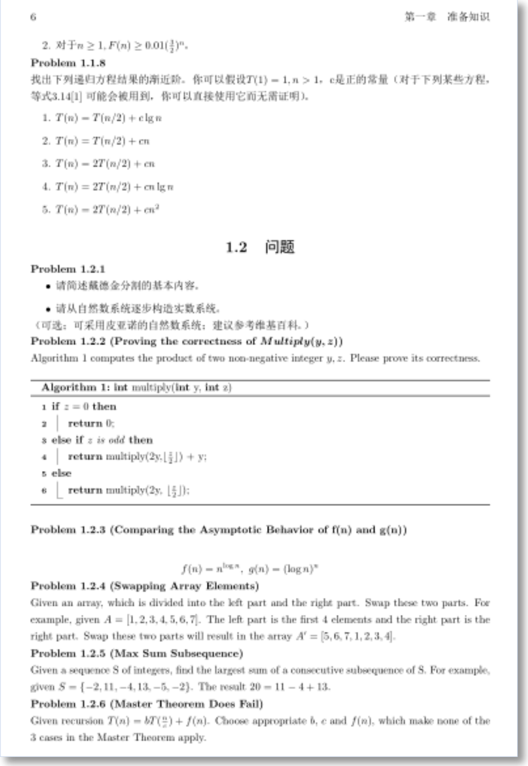


算法设计与分析习题集



目录

3



6 第一章 准备知识

2. 对于$n \geq 1, F(n) \geq 0.01(\frac{3}{2})^n$。

Problem 1.1.8
找出下列递归方程结果的渐近阶。你可以假设$T(1) = 1, n > 1$，c是正的常量（对于下列某些方程，等式3.14[1] 可能会被用到，你可以直接使用它而无需证明）。

1. $T(n) = T(n/2) + c\lg n$
2. $T(n) = T(n/2) + cn$
3. $T(n) = 2T(n/2) + cn$
4. $T(n) = 2T(n/2) + cn\lg n$
5. $T(n) = 2T(n/2) + cn^2$

1.2 问题

Problem 1.2.1
- 请简述戴德金分割的基本内容。
- 请从自然数系统逐步构造实数系统。

（可选：可采用皮亚诺的自然数系统；建议参考维基百科。）

Problem 1.2.2 (Proving the correctness of $Multiply(y, z)$)
Algorithm 1 computes the product of two non-negative integer y, z. Please prove its correctness.

```
Algorithm 1: int multiply(int y, int z)
if z = 0 then
    return 0;
else if z is odd then
    return multiply(2y, ⌊z/2⌋) + y;
else
    return multiply(2y, ⌊z/2⌋);
```

Problem 1.2.3 (Comparing the Asymptotic Behavior of f(n) and g(n))

$$f(n) = n^{\log n},\ g(n) = (\log n)^n$$

Problem 1.2.4 (Swapping Array Elements)
Given an array, which is divided into the left part and the right part. Swap these two parts. For example, given $A = [1, 2, 3, 4, 5, 6, 7]$. The left part is the first 4 elements and the right part is the right part. Swap these two parts will result in the array $A' = [5, 6, 7, 1, 2, 3, 4]$.

Problem 1.2.5 (Max Sum Subsequence)
Given a sequence S of integers, find the largest sum of a consecutive subsequence of S. For example, given $S = \{-2, 11, -4, 13, -5, -2\}$. The result $20 = 11 - 4 + 13$.

Problem 1.2.6 (Master Theorem Does Fail)
Given recursion $T(n) = bT(\frac{n}{c}) + f(n)$. Choose appropriate b, c and $f(n)$, which make none of the 3 cases in the Master Theorem apply.

Websites

QQ group: 2105 15746

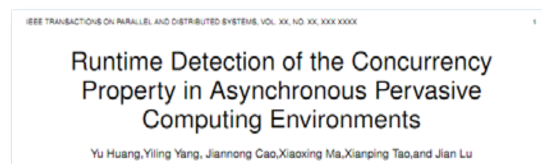
QA site: http://bigoh.net

Algorithm – Design & Analysis

- **Algorithm - the spirit of computing**
  - Model of computation
- **Algorithm by example**
  - Greatest common divisor
  - Sequential search
- **Algorithm design & analysis**
  - Correctness
  - Worst-case / average-case cost analysis

Computer and Computing

- **Problem 1**
  - o Why the computer seems to be able to do anything?
    - Scientific computing, document processing, computer games, EBooks, Movies, Computer games, …

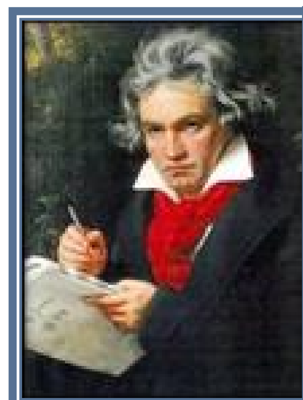

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. XX, NO. XX, XXX XXXX

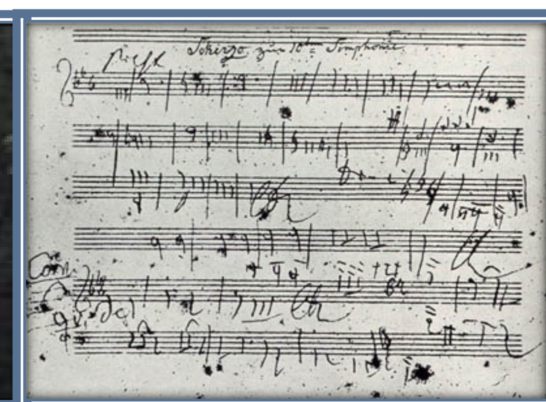
Runtime Detection of the Concurrency Property in Asynchronous Pervasive Computing Environments

Yu Huang, Yiling Yang, Jiannong Cao, Xiaoxing Ma, Xianping Tao, and Jian Lu

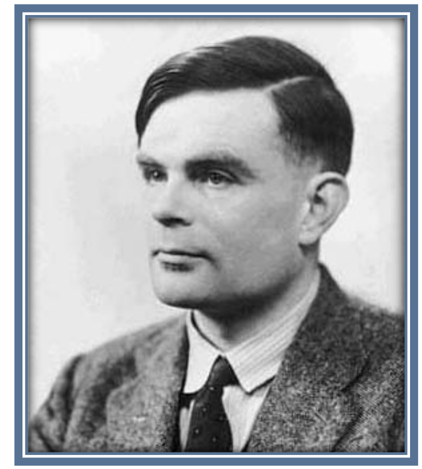
Computer and Computing

- **Problem 2**
  - o What can / cannot be efficiently done by a computer?
    - manage millions of songs vs. music composition



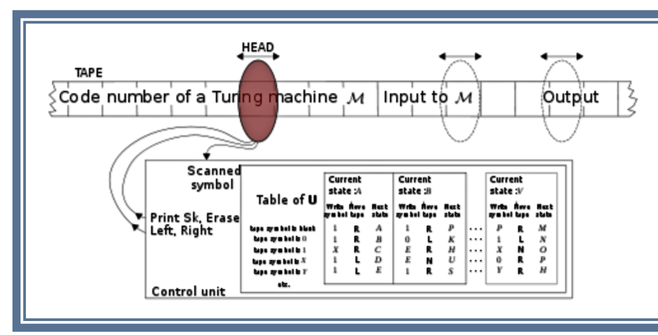


Computer and Computing

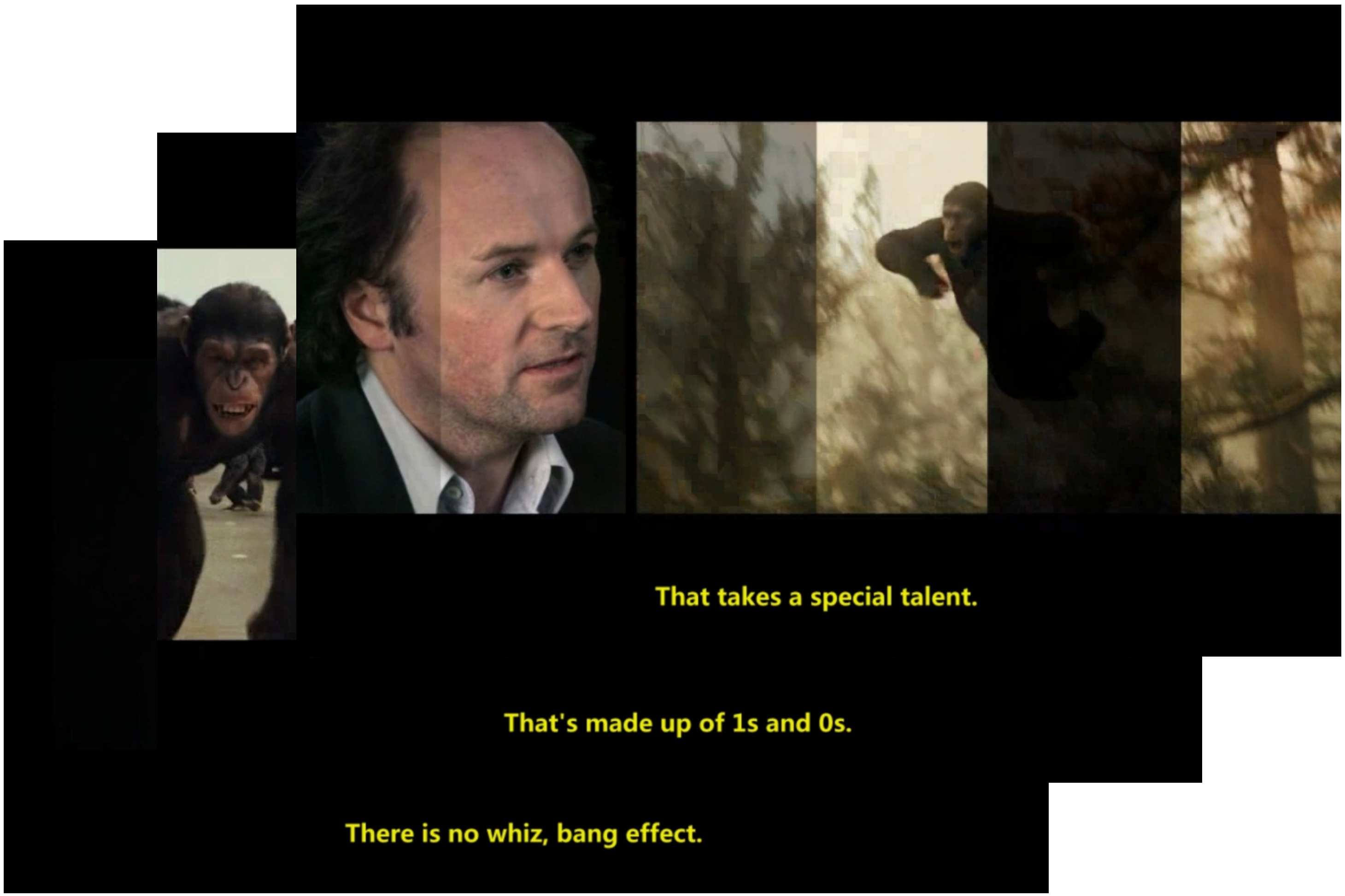


- **Computing**
  - o Encoding everything into `0's and `1's
  - o Operations over '1's and '0's
  - o Decoding the '1's and '0's

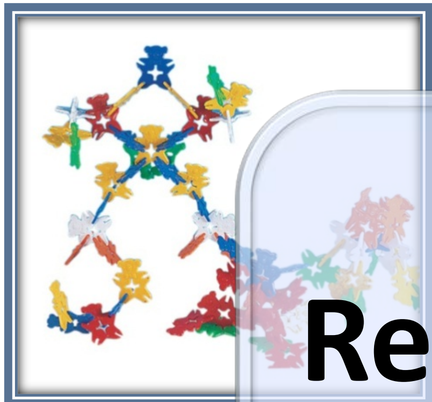
- **Turing machine**
  - o An abstract/logical computer

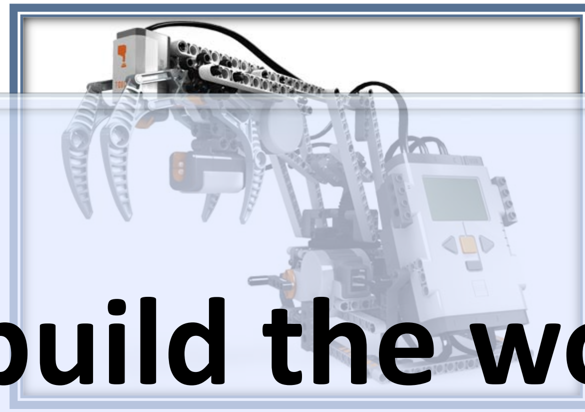


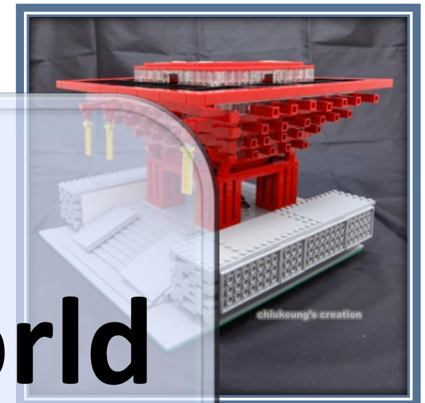
Computing in Everyday Life



Algorithm



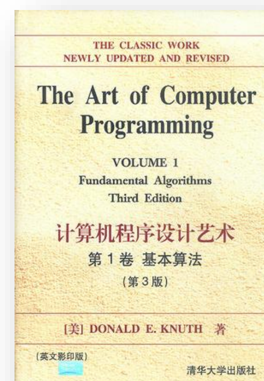




Rebuild the world with 0s and 1s

Algorithm

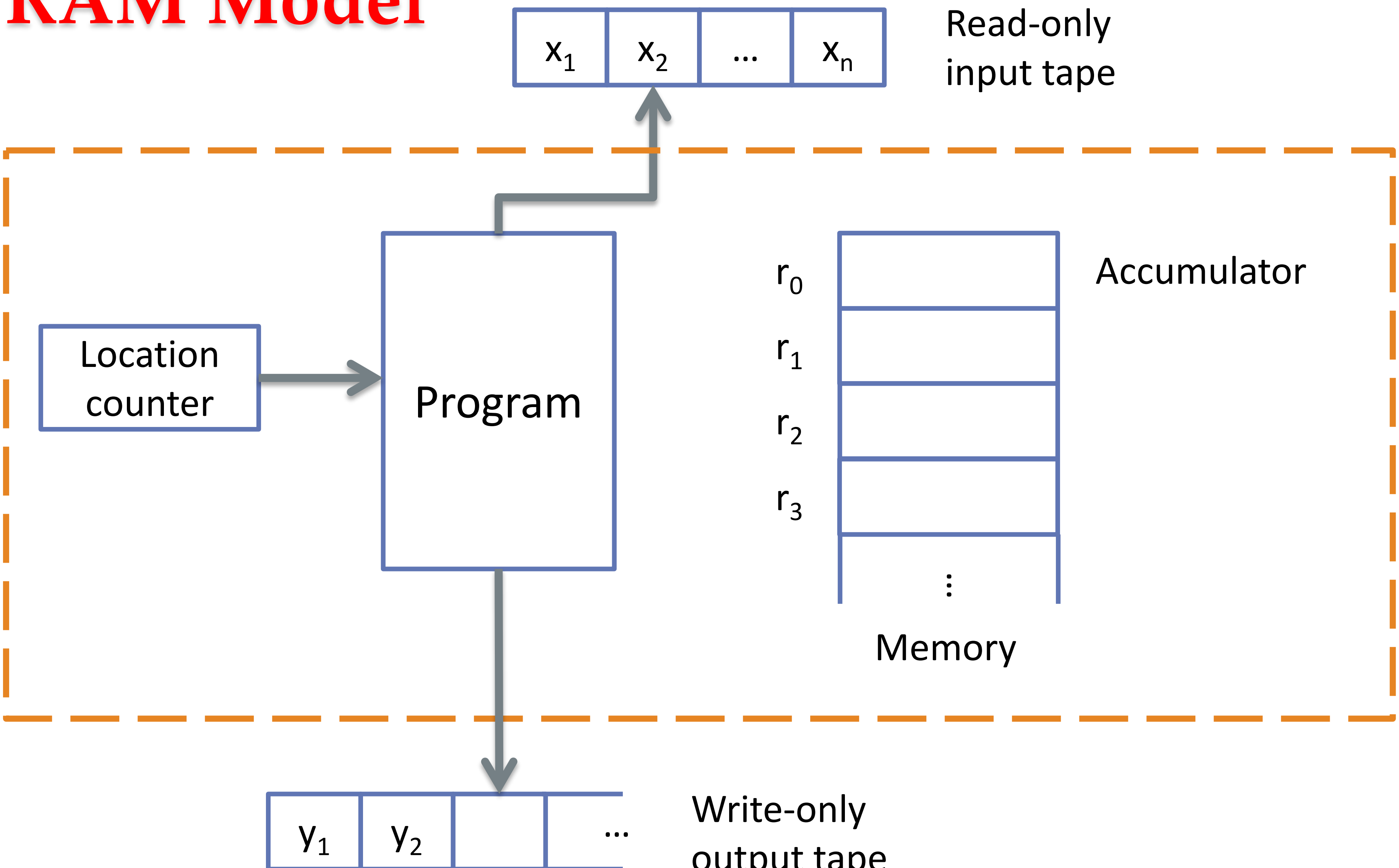
- **Algorithm is the spirit of computing**
  - o To solve a specific problem (so called an *algorithmic problem*)
  - o Combination of basic operations
    - in a precise and elegant way
- **Essential issues**
  - o Model of computation
  - o Algorithm design
  - o Algorithm analysis



Model of Computation

- **Problems**
  - Why the algorithms we learn can run almost everywhere?
  - Why the algorithms we learn can be implemented in any language?
- **Machine- and language- independent algorithms, running on an abstract machine**
  - Turing machine: over-qualify
  - RAM model: simple but powerful

RAM Model



x_1
x_2
...
x_n
Read-only input tape
Location counter
Program
r_0
r_1
r_2
r_3
Accumulator
Memory
y_1
y_2
...
Write-only output tape

The RAM Model of Computation

- **Each *simple operation* takes one time step**
  - E.g., key comparison, +/-, memory access, …
- **Non-simple operations should be decomposed**
  - Loop
  - Subroutine
- **Memory**
  - Memory access is a simple operation
  - Unlimited memory

Tradeoff:
accuracy
vs.
ease of use

Further Reading

“哼，你让他们成楔形攻击队形不就行了？”秦始皇轻蔑地看着冯·诺伊曼。牛顿不知从什么地方掏出六面小旗．三白三黑，冯·诺伊曼接过来分给三名士兵，每人一白一黑，说：“白色代表0，黑色代表1。好，现在听我说，出，你转身看着入1和入2，如果他们都举黑旗，你就举黑旗，其他的情况你都举白旗，这种情况有三种：入1白，入2黑；入1黑，入2白；入1、入2都是白。”

“不需要，我们组建一千万个这样的门部件，再将这些部件组合成一个系统，这个系统就能进行我们所需要的运算，解出那些预测太阳运行的微分方程。这个系统，我们把它叫做……嗯，叫做……”

“计算机。”汪淼说。

“啊——好！”冯·诺伊曼对汪淼竖起一根指头，“计算机，这个名字好，整个系统实际上就是一部庞大的机器，是有史以来最复杂的机器！”

刘慈欣，《三体、牛顿、冯·诺依曼、秦始皇、三日连珠》，《三体》第一部

To Create an Algorithm

- **Algorithm design**
  - o Composition of simple operations, to solve an algorithmic problem
- **Algorithm analysis**
  - o Amount of work done / memory used
    - In the worst/average case
  - o Advanced issues
    - Optimality, approximation ratio, …

Algorithm by Example

- **Algorithmic Problem 1**
  - Find the greatest common divisor of two non-negative integers *m* and *n*

- **Algorithmic Problem 2**
  - Is a specific key *K* stored in array E[1..n]?

Probably the Oldest Algorithm

- **Euclid Algorithm**

Problem

- Find the greatest common divisor of two non-negative integers *m* and *n*

Specification

Input: non-negative integer m, n
Output: gcd(m, n)

Euclid algorithm

[E1] n divides m, the remainder -> r
[E2] if r = 0 then return n
[E3] n -> m; r-> n; goto E1

Euclid algorithm – recursive version

Euclid(m,n)
[E1] if n=0 then return m
[E2] else return Euclid(n, m mod n)

Sequential Search

Problem
- Search an array for a specific key

Specification

Input: K, E[1..n]

Output: Location of K (1,2,…,n; -1: K is not in E[])

Sequential searchEuclid algorithm

```
Int seqSearch(int[] E, int n, int K)
  int ans, index;
  ans=-1;
  for (index=1; index<=n; index++)
    if (K==E[index])
      ans=index;
      break;
  Return ans;
```

Algorithm Design

- **Criteria**
  - Defining correctness
- **Main challenge**
  - For proving correctness
- **Our strategy**
  - Mathematical induction
  - …

Specification

Input: non-negative integer m, n
Output: gcd(m, n)

Main challenge

- The output is **always** correct, for **any** legal input.
- Infinite possible inputs

Mathematical induction

- Weak principle
- Strong principle

For Your Reference

- **Mathematical induction**

The Weak Principle of Mathematical Induction

- If the statement p(b) is true and the statement p(n-1) => p(n) is true for all n>b, then p(n) is true for all integers n>=b.

The Strong Principle of Mathematical Induction

- If the statement p(b) is true, and the statement {p(b) and p(b+1) and … and p(n-1) => p(n)} is true, for all n>b, then p(n) is true for all integers n>=b.

Correctness of the Euclid Algorithm

- **Induction on n**
  - o Base case
    - n = 0: for any m, Euclid(m, 0) = m;
    - n = 1: for any m, Euclid(m, 1) = 1;
    - n = 2: …
  - o Assumption
    - For any n<=N_0, Euclid(m, n) is correct;
  - o Induction
    - Euclid(m, N_0+1) = Euclid(N_0+1, m mod (N_0+1));

gcd(m, N_0+1) = gcd(N_0+1, m mod (N_0+1))

Notes on Mathematical Induction

“Notes on Structured Programming”, E.W. Dijkstra

I have mentioned **mathematical induction** explicitly, because it is the only pattern of reasoning that I am aware of, that eventually enables us to cope with loops and recursive procedures

Algorithm Analysis

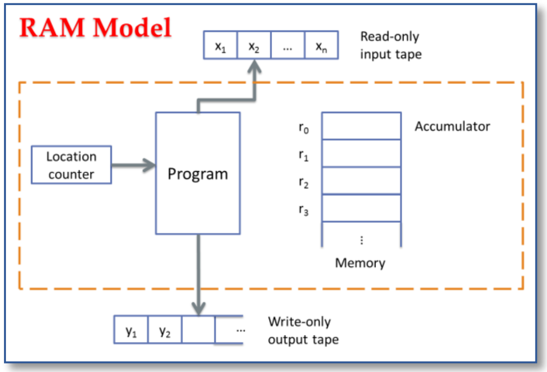
- **Criteria**
  - Performance metrics
- **Worst case**
  - Best case?
- **Average case**
  - Average cost?
- **Advanced topics**
  - Lower bound, optimality, …

Algorithm Analysis

- **How to measure**
  - Not too general
    - Giving essential indication in comparison of algorithms
  - Not too precise
    - Machine independent
    - Language independent
    - Programming paradigm independent
    - Implementation independent

Algorithm Analysis

- **Criteria**
  - Critical operation
  - How many critical operation are conducted
- **For example**



Algorithmic problem	Critical operation
Sorting, selection, searching String matching	Comparison (of keys)
Graph traversal	Processing a node/edge
Matrix multiplication	Multiplication

Algorithm Analysis

- **Amount of work done**
  - o usually depends on size of the input
  - o usually does not depend on size of the input only

Worst-case Complexity

- **W(n)**
  - Upper bound of cost
    - For any possible input
  - $W(n) = \max_{I \in D_n} f(I)$

Average-case Complexity

- **A(n)**
  - Weighted average
  - $A(n) = \sum_{I \in D(n)} \Pr(I)\, f(I)$
- **A special case**
  - Average cost
    - Total cost of all inputs, averaged over the input size
  - $Average(n) = \frac{1}{|D(n)|} \sum_{I \in D(n)} f(I)$

Average-case Cost of *SeqSearch*

- **Case 1: K is in E[]**
  - Assumptions:
    1. Assuming that K is in E[]
    2. Assuming no same entries in E[]
    3. Each possible input appears with equality (thus, K in the i^{th} location with probability $\frac{1}{n}$)

  - $$A_{succ}(n) = \sum_{i=0}^{n-1} \Pr(I_i|succ)\, t(I_i)$$
    $$= \sum_{i=0}^{n-1} \frac{1}{n}(i+1)$$
    $$= \frac{n+1}{2}$$

Average-case Cost of *SeqSearch*

- **Case 2: K may (or may not) be in E[]**
  - Assume that K is in E[] with probability q
  - $\mathrm{A}(n) = \Pr(succ)\, A_{succ}(n) + \Pr(fail)\, A_{fail}(n)$
    $= q\,\frac{n+1}{2} + (1-q)n$

How to make reasonable assumptions?

Algorithm Analysis

- **Advanced topics**
  - Lower bound (Selection)
  - Optimality (Greedy, DP)
  - Computation complexity
  - Approximate / online / randomized algorithms

Thank you!

Q & A

Yu Huang

http://cs.nju.edu.cn/yuhuang