

算法分析与设计

Algorithm Analysis and Design

主讲人：甘文生 PhD

Email: wsgan001@gmail.com

暨南大学网络空间安全学院

Fall 2021

Jinan University, China

个人简介

甘文生，工学博士，暨南大学副教授，硕士生导师

<https://faculty.jnu.edu.cn/xxkxjsxy/gws/list.htm>

<https://dblp.org/pid/145/5903.html>

学习经历

2016.03-2019.12，哈尔滨工业大学，计算机科学与技术/博士，导师：赵涵捷 教授/校长

2017.09-2019.03，美国伊利诺伊大学芝加哥分校，联合培养博士，导师：Philip S. Yu 教授

2013.09-2015.12，哈尔滨工业大学，计算机技术/硕士，导师：Jerry Chun-Wei Lin 教授

2009.09-2013.06，华南师范大学，计算机科学与技术/学士，推荐免试

研究方向

- 1) 数据科学/挖掘/治理、云计算、大数据技术；
- 2) (涉及数据、系统、网络等) 安全与隐私保护；
- 3) 区块链与物联网；
- 4) 数字孪生、流程挖掘等。

学术服务

- 1) SCI期刊JIT (Journal of Internet Technology) 的副主编 (AE)
- 2) 20余个领域知名SCI期刊的审稿人 (TKDE、TKDD、TCYB、TIST等)
- 3) 多个国际会议的程序委员会委员TPC

个人简介

合作方向：数据挖掘、大数据技术、区块链、数字孪生等

项目经历

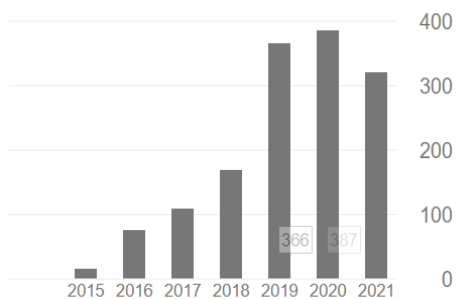
1. 2021 广东省琶洲实验室项目，大数据的价值挖掘与赋能应用 (**主持**)
2. 2021 广州市科技计划项目，大数据的精简式价值挖掘技术研究 (**主持**)
3. 2021 NSFC国家自然科学基金，基于时序数据的效用挖掘及其隐私保护技术 (**主持**)
4. 2016 NSFC国家自然科学基金，基于多源异构不确定数据.... (**排名第2**)
5. 2017 深圳市基础研究计划，电子商务数据安全的关键性技术研究 (**排名第3**)
6. 哈工大-腾讯联合项目，序列高效用规则的研究 (**排名第2**)

科研成果

在数据科学、隐私安全、大数据等领域，发表**100余篇**高水平学术论文，其中**50余篇SCI期刊论文**（如IEEE TKDE、ACM TKDD、ACM TDS、IEEE TCYB、ACM TOIT、ACM TMIS等）和**40余篇EI会议论文**。**学术影响力为：Citation 1500, H-index为22, H10-index为35**。获得公示或授权美国/PCT/中国发明专利2/2/11件。参与数据挖掘知名开源系统**SPMF**、隐私保护/安全挖掘开源系统**PPSF**等开源。

引用次数

| | 总计 | 2016 年至今 |
|--------|------|----------|
| 引用 | 1450 | 1432 |
| h 指数 | 22 | 22 |
| i10 指数 | 35 | 35 |



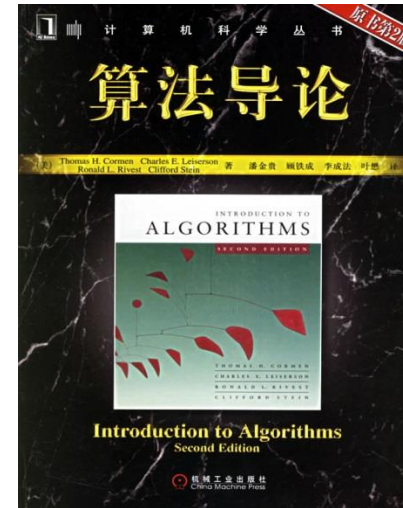
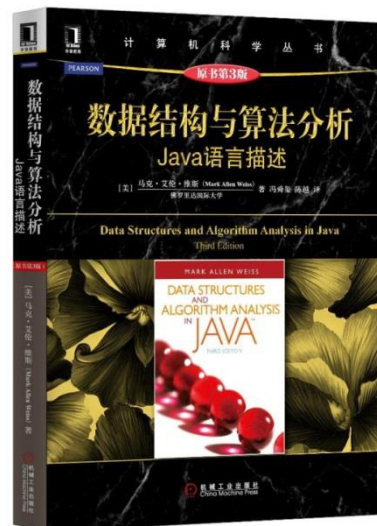
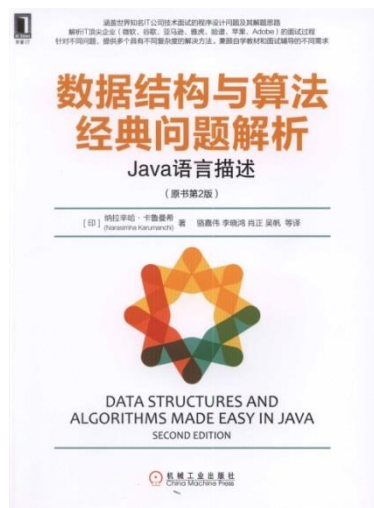
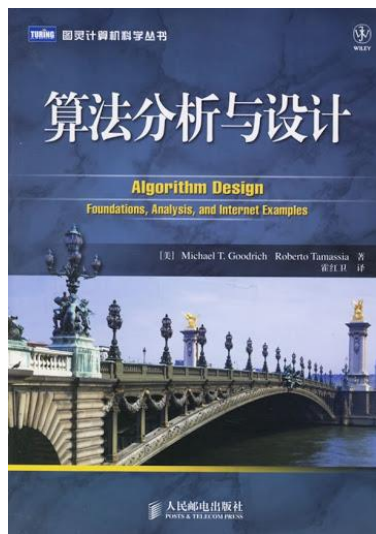
课程信息

- 课程名称：算法分析与设计 (专业核心课)， N329教室
- 课程安排：54学时(45学时+9学时)， 3学分
- 授课时间：第3-15周， 第16周论文汇报， 第17周 (复习/答疑)
- 教材信息：

- * 《算法导论》(第2版)， Thomas 等著， 潘金贵等译， 机械工业出版社， 2006.9
- * 《The Art of Computer Programming》, Donald E Knuth (高德纳)， 1974, Turing Prize
- * 《算法设计与分析》 屈婉玲、刘田、张立昂、王捍贫编著， 清华大学出版社， 2011
- * 《计算机算法设计与分析》(第3版)， 王晓东， 电子工业出版社， 2007
- * 《算法》(红皮书)， 普林斯顿大学
- * 《算法设计与分析基础》， Anany Levitin著， 潘彦译， 清华大学出版社， 2007

.....

参考教材



课程信息

□ 授课形式

- ✓ 课堂讲解（45学时）：基本理论讲解、基本方法的介绍分析；
- ✓ 上机实践（9学时）：3~4个经典算法的上机实验；
- ✓ 课程作业：大约2次的小作业，1~2次大作业；

□ 考核形式

- ✓ 期末考试（**闭卷**）：60%
- ✓ Project报告：30%
- ✓ 出勤+课堂纪律：10%

□ 选课要求

- ✓ 学过数据结构、离散数学、概率论
- ✓ 具备编程技能（C、C++、Java、Python等）



课程信息

□ 联系方式

- ✓ 甘文生 老师
- ✓ 办公室：番禺校区实验楼D1栋103室
- ✓ 电 话：13691774876（办）
- ✓ Email: wsgan001@gmail.com

□ 课程助教

- ✓ XXX 同学
- ✓ 实验室：番禺校区实验楼D1栋
- ✓ Email: XXX@jnu.edu.cn

相关基础：数据结构、算法、程序设计

□ 数据结构

✓ 研究数据的逻辑结构、物理结构及其操作的学科

程序 = 数据结构 + 算法 + 语言 + 程序设计方法

- 数据结构是算法实现的**基础**，算法总是依赖于数据结构来**实现**
- 算法是灵魂，数据结构是加工对象，(低级或高级) 语言是工具，编程需采用合适的方法。
- 算法在很大程度上受到数据结构的限制，甚至在某些情况下数据结构起决定性的作用。

编程的灵魂：数据结构+算法

□ 数据结构

- ✓ 表示要处理的数据（包括输入的数据和输出的数据）
- ✓ 再设计相应的算法来实现程序的功能
- ✓ 最后使用某一门程序设计语言来进行编码
- ✓ 综合起来，便构成一个实实在在的程序

□ 算法

- ✓ 算法是解决问题的抽象方法和步骤
- ✓ 同一个算法在不同的语言中具有不同的实现形式
- ✓ 依赖于数据结构的形式和程序设计语言的语法格式

□ 数据结构是算法实现的**基础**，算法总是依赖于数据结构来**实现**

□ **人工智能**的三要素：数据、**算法**、算力

算法的要素

- ❑ 算法由操作、控制结构、数据结构三要素组成
- ❑ 算术运算：加、减、乘、除
- ❑ 关系比较：大于、小于、等于、不等于
- ❑ 逻辑运算：与、或、非
- ❑ 数据传送：输入、输出、赋值

- ❑ 三种基本的控制结构
 - ✓ ① 顺序结构
 - ✓ ② 选择结构
 - ✓ ③ 循环结构

算法为王

一般认为，人工智能技术发挥作用的三要素

- 数据
- 算法
- 算力

大数据时代



大数据 \neq 大价值



机器学习
算法!

有效的数据分析

大数据算法，Hadoop/Spark大数据处理

教学内容

□ Part 1 基础知识

- ✓ 课程学习背景
- ✓ 算法分析基础

□ Part 2 排序算法

- ✓ 归并排序、堆排序、快速排序
- ✓ 计数排序、基数排序、桶排序

□ Part 3 算法设计策略

- ✓ 递归与分治法、动态规划法、贪心法、回溯法、分枝限界法

□ Part 4 算法研究问题

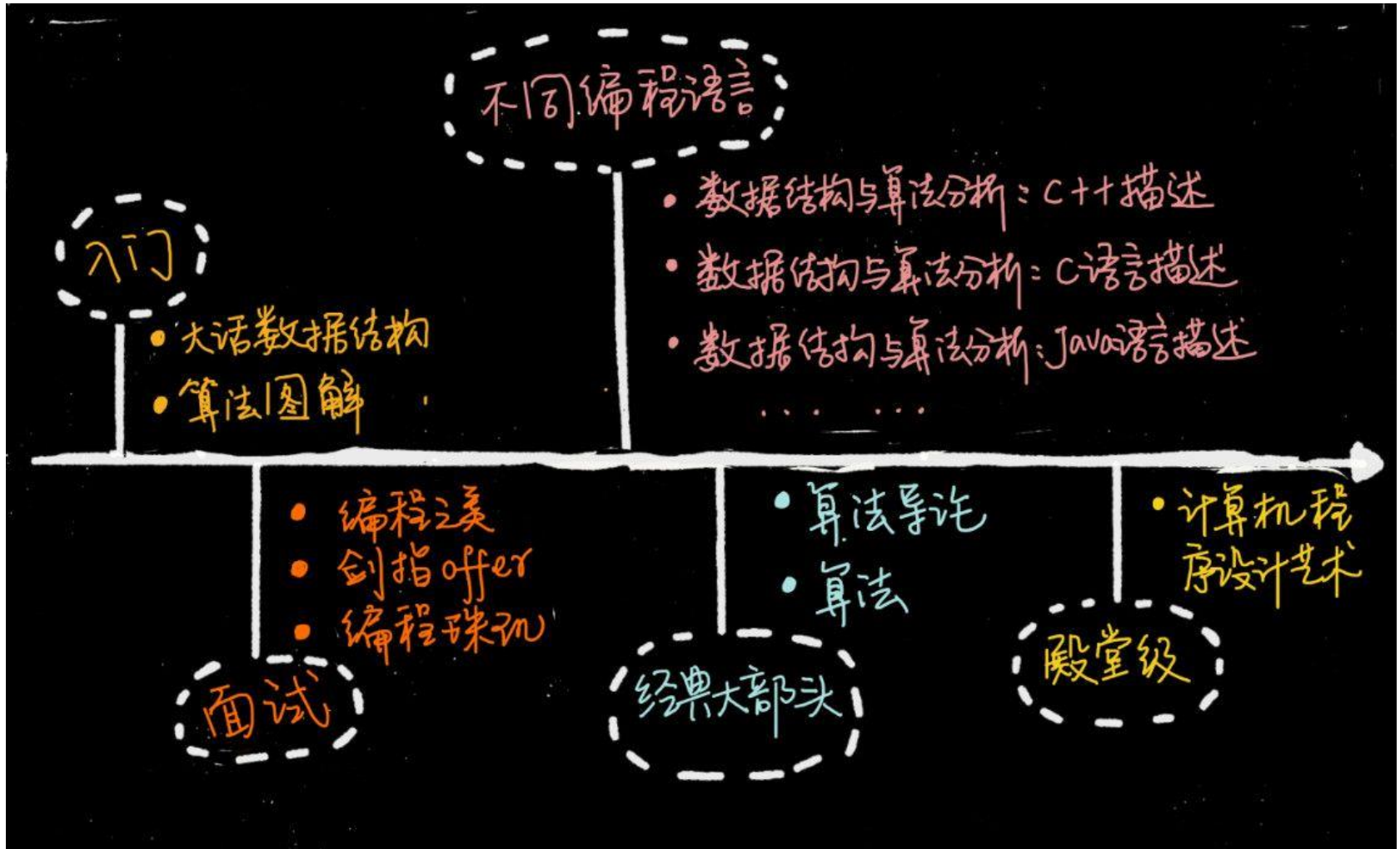
- ✓ NP完全问题，近似算法、随机算法、有关数论的算法，

教学目标

- A survey of algorithmic design techniques.
- Abstract thinking.
- How to develop new algorithms for any problem that may arise.
- Be a great thinker and designer.
- ~~Not: A list of algorithms~~
 - ~~Learn their code~~
 - ~~Trace them until work~~
 - ~~Implement them~~
 - ~~be a mundane programmer~~



教学目标



教学目标

- ❑ 学会发现问题，具备抽象描述、解决实际问题的能力
- ❑ 掌握算法的基本原理、设计技巧
- ❑ 掌握算法的复杂度分析
- ❑ 学会算法设计与分析的典型方法，并进行算法的设计
 - ✓ 递归与分治法、动态规划法、贪心法、回溯法、分枝限界法
 - ✓ NP完全问题，近似算法、随机算法
- ❑ 具备分析算法效率的能力
- ❑

学习方法：多读、多思考、多编程

第一讲 算法入门

内容提要：

- 课程学习背景
- 算法分析基础
- 算法设计策略之——分治法

两个例子：“插入排序”和“归并排序”

第一讲 算法入门

内容提要:

□ 课程学习背景

- ✓ 为什么学习算法?
- ✓ 算法是什么
- ✓ 算法的描述
- ✓ 算法的设计要求

□ 算法分析基础

□ 算法设计策略之——分治法

为什么学习算法

□ **算法Algorithm**是计算机科学的基石，是改造世界的有力工具！

“微积分以及在微积分基础上建立起来的数学分析体系成就了现代科学，而算法则成就了现代世界” —— David Berlinski, 2000

互联网是20世纪最伟大的发明之一，改变了世界，改变了我们的生活！各种算法在支撑着整个互联网的正常运行，互联网的信息传输需要路由选择算法，互联网的信息安全需要加密算法，互联网的信息检索需要模式匹配算法，互联网的信息存储需要排序算法，……，没有算法也就没有互联网！

□ 学习算法可以开发人们的分析能力

算法是解决问题的一类特殊方法，是对问题的准确理解和定义后获取答案的过程。

□ 是你获得高薪职位的敲门砖！

为什么学习算法

❑ **算法Algorithm**是计算机科学的基石，是科技发展的推力！

“图灵奖”于1966年开始设立，是ACM(美国计算机协会)在计算机科学技术领域中所授予的最高奖项

有超过1/3的Turing奖获奖者，其成果与算法有关！

❑ 1972, Edsger W. Dijkstra (求最短路径的**Dijkstra算法**)



❑ 1974, Donald E. Knuth (高德纳, Stanford)

- ✓ 算法最早的奠基人之一(计算机程序设计艺术)
- ✓ 现代“算法”与“数据结构”名词及内涵的提出,
- ✓ KMP算法, LR(k)文法, Tex编辑器等



❑ 1980, C. Anthony R. Hoare (英)

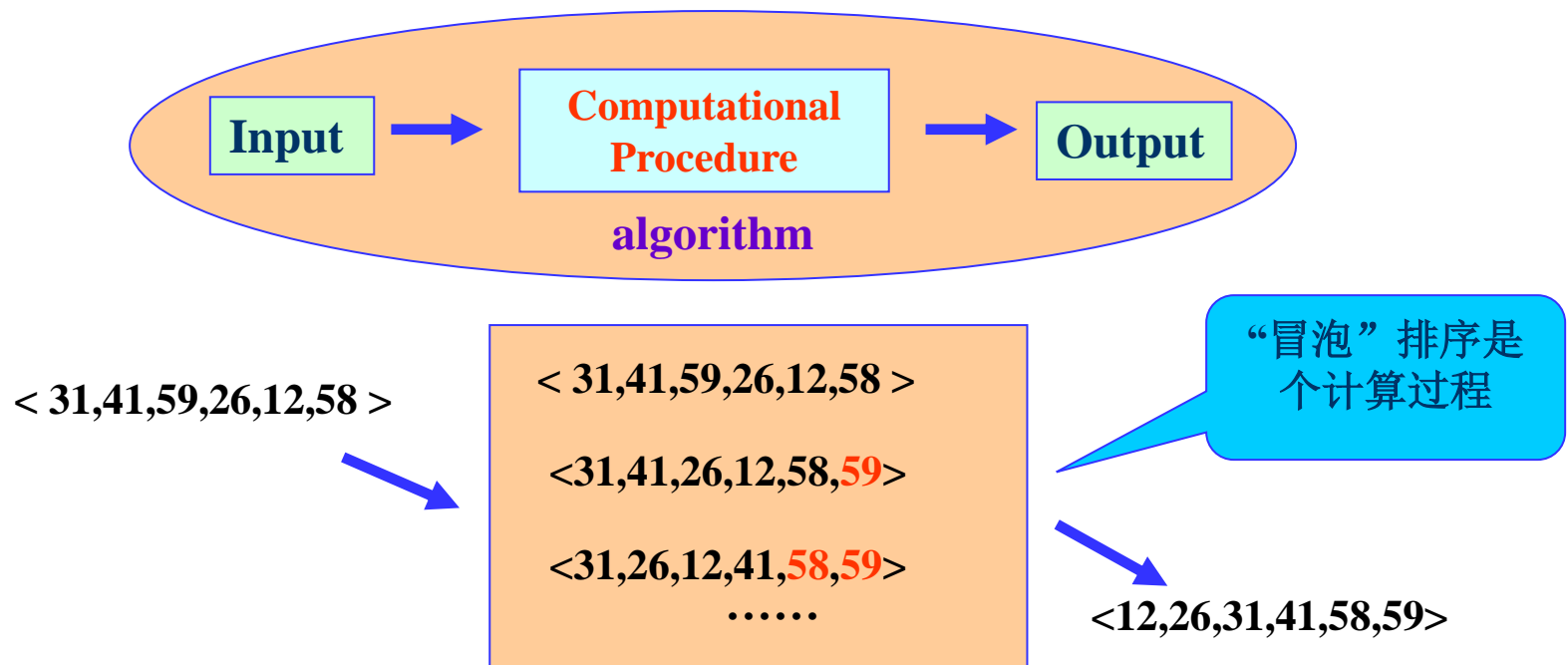
- ✓ **Quick-sort算法**
- ✓ 程序设计 (CASE、While语句等)



什么是算法？

□ 简单说来，**算法就是问题的程序化解决方案。**

□ **定义：**算法就是一个定义良好的可计算过程，它取一个或者一组值作为输入，并产生出一个或者一组值作为输出。因此，算法是一系列的计算步骤，用来将输入数据转换成输出结果。



什么是算法？

□ 一个算法通常具有如下特征：

- ① 输入：一个算法具有零个或者多个取自指定集合的输入值；
- ② 输出：对算法的每一次输入，算法具有一个或多个与输入值相联系的输出值；
- ③ 确定性：算法的每一个指令步骤都是明确的；
- ④ 有限性/有穷性：对算法的每一次输入，算法都必须在有限步骤（即有限时间）内结束；
- ⑤ 正确性：对每一次输入，算法应产生出正确的输出值；
- ⑥ 通用性：算法的执行过程可应用于所有同类求解问题，而不是仅适用于特殊的输入。

思考：算法与程序的区别？

算法与程序的区别

- 算法的概念和程序很相似，但实际上不同。程序并不都满足算法的所有特征，如有限性特征。算法代表了对特定问题的求解，而程序则是算法在计算机上的实现。因此，算法可以称为是一个执行过程。一个函数若能用一个算法来计算，那么我们称该函数是能行可计算的。操作系统是一种程序，而不是算法。

算法与程序：(1). 一个程序不一定满足有穷性。例操作系统，只要整个系统不遭破坏，它将永远不会停止，即使没有作业需要处理，它仍处于动态等待中。因此，操作系统不是一个算法。(2). 程序中的指令必须是机器可执行的，而算法中的指令则无此限制。(3). 算法代表了对问题的解，而程序则是算法在计算机上的特定的实现。一个算法若用程序设计语言来描述，则它就是一个程序。

相关概念： 问题和问题实例

□ 问题 (Problem)： 规定了输入与输出之间的关系，可以用通用语言来描述；

□ 问题实例： 某一个问题的实例包含了求解该问题所需的输入；

□ 排序问题——将一系列数按非降顺序进行排序

输入： 由 n 数组成的一个序列 $\langle a_1, a_2, \dots, a_n \rangle$

输出： 对输入系列的一个排列(重排) $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，使得 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

□ 排序问题的一个**实例**：

Input: $\langle 31, 41, 59, 26, 41, 58 \rangle$ **Output:** $\langle 26, 31, 41, 41, 58, 59 \rangle$

相关概念： 问题和问题实例

□ 算法可求解的问题：

- ✓ 人类基因项目
- ✓ 在电子商务中的应用
- ✓ 在互联网中的应用（图像检索、视频检索.....）
- ✓

□ 重要问题类型：排序、查找、字符串处理、图问题、组合问题、几何问题、数值问题等。

相关概念：输入实例与问题规模

□ 输入实例：实际问题的具体计算例子

如，排序问题的3个输入实例：

① 13,5,6,37,8,92,12

② 43,5,23,76,25

③ 53,67,32,42,22,33,4,39,56

□ 问题规模：算法的输入实例大小。

如，上面排序问题的3个输入实例的规模大小分别为7,5,9

算法的描述

□ 基于文字描述

- ✓ 优点：容易理解
- ✓ 缺点：冗长、二义性

□ 基于流程图描述

- ✓ 优点：流程直观
- ✓ 缺点：缺少严密性、灵活性

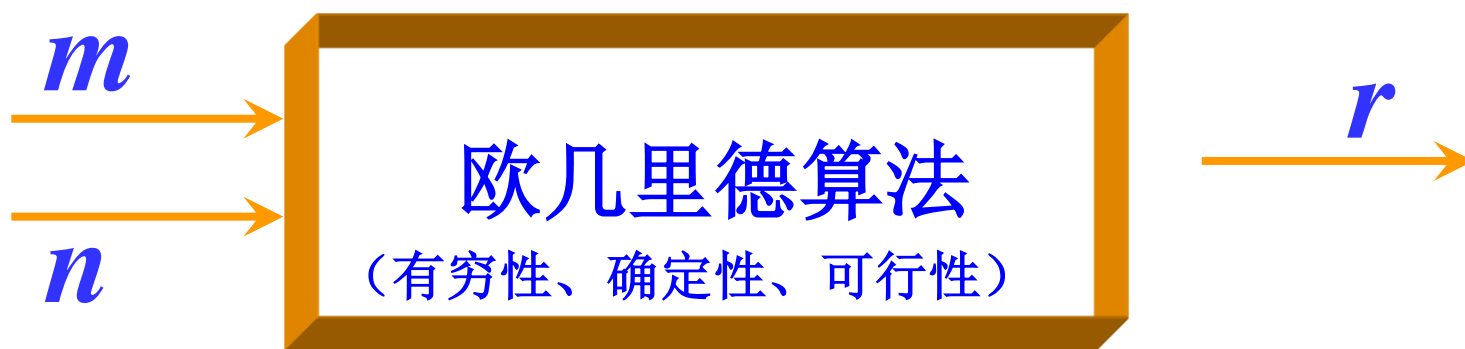
□ 基于程序设计语言

- ✓ 优点：能由计算机执行
- ✓ 缺点：抽象性差，对语言要求高

□ 伪代码

算法的描述

例子：欧几里德算法——辗转相除法，求两个自然数 m 和 n 的最大公约数。具体做法：用较大数除以较小数，再用出现的余数（第一余数）去除除数，再用出现的余数（第二余数）去除第一余数，如此反复，直到最后余数是0为止。如果是求两个数的最大公约数，那么最后的除数就是这两个数的最大公约数



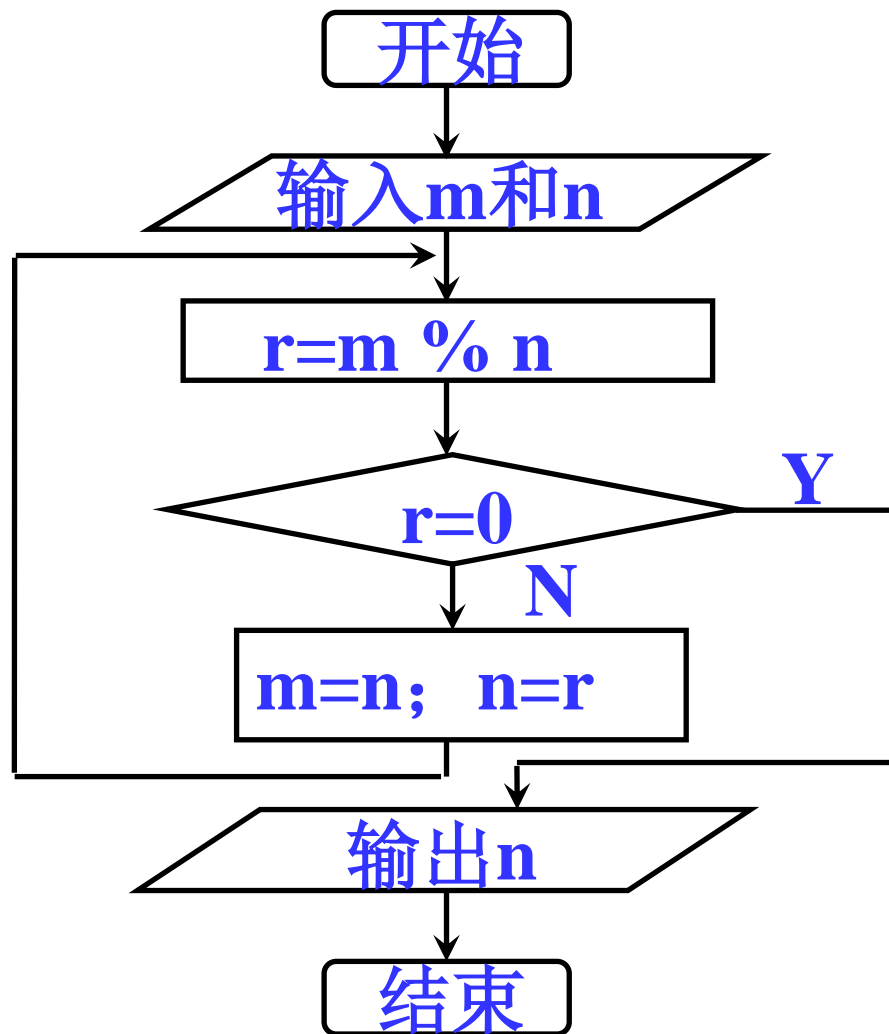
算法描述——基于自然语言

步骤1: 将 m 除以 n 得到余数 r ;
步骤2: 若 r 等于0, 则 n 为最大公约数,
 算法结束; 否则执行步骤3;
步骤3: 将 n 值放在 m 中, 将 r 值放在 n 中, 重新执行步骤1;

- 优点: 容易理解
- 缺点: 冗长、二义性
- 使用方法: 粗线条描述算法思想
- 注意事项: 避免写成自然段

算法描述——基于流程图

- 优点：流程直观
- 缺点：缺少严密性、灵活性
- 使用方法：描述简单算法
- 注意事项：注意抽象层次



算法描述——基于程序设计语言

```
#include <iostream.h>
```

```
int CommonFactor(int m, int n) {  
    int r = m % n;  
    while (r != 0) {  
        m = n;  
        n = r;  
        r = m % n;  
    }  
    return n;  
}
```

□ 优点：能由计算机执行

□ 缺点：抽象性差，对语言要求高

□ 使用方法：算法需要验证

□ 注意事项：将算法写成子函数

```
void main( ) {  
    cout<<CommonFactor(63, 54)<<endl;  
}
```

算法描述——基于伪代码

❑ 伪代码 (Pseudocode) 介于自然语言和程序设计语言之间的方法，它采用某一程序设计语言的基本语法，操作指令可以结合自然语言来设计。

❑ 优点：

- ✓ 表达能力强
- ✓ 抽象性强
- ✓ 容易理解

```
1.  $r = m \% n$ ;  
2. 循环直到  $r$  等于0  
    2.1  $m = n$ ;  
    2.2  $n = r$ ;  
    2.3  $r = m \% n$ ;  
3. 输出  $n$ ;
```

算法设计的要求：一个好算法

□ 正确性(correctness)

算法能满足具体问题的需求；对输入、输出和处理过程等有明确的无歧义的描述；程序不包含语法错误；对输入实例能有正确的输出结果。[软件测试]

- ✓ 层次a：程序不含语法错误；
- ✓ 层次b：程序对于几组输入数据能得出满足要求的结果；
- ✓ 层次c：程序对于精心选择的典型、苛刻的几组输入数据能够得出满足规格说明要求的结果；
- ✓ 层次d：程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

相关概念：正确算法与不正确算法

□ 正确的算法

- ✓ 算法对问题每一个输入实例，都能输出正确的结果并停止，则称为正确的。

□ 不正确的算法

- ✓ 可能根本不会停止；
- ✓ 停止时给出的不是预期的结果；
- ✓ 如果算法的错误率可以控制，也是**有用的**。

□ 近似算法

- ✓ 对所有输入都停止
- ✓ 产生近似正确的解或产生不多的不正确解

□ 调试程序 \neq 程序正确性证明

- ✓ 程序调试只能证明程序有错，
- ✓ 不能证明程序无错误！

算法设计的要求: 一个好算法

□ 可读性(readability)

易于阅读与理解, 晦涩难懂的程序容易隐藏较多的错误, 难以调试、修改和维护 [详细的文档与注释]

□ 健壮性(robustness)

对于非法的输入数据, 能适当地做出反应或进行处理, 不会产生莫名其妙的结果 (异常中断)

□ 效率与低存储等需求

- ✓ 效率指算法执行的时间;
- ✓ 存储量需求指算法执行过程中需要的最大存储空间;
- ✓ 均与问题的规模有关。

算法设计：一个好算法

□ 如果计算机无限快、存储器都是免费的,算法研究是否还需要?

① Yes! 证明方案是正确的, 可以给出正确结果。

② Yes! 希望自己的实现符合良好的软件工程实践要求, 采用最容易的实现方法。

□ 算法对于当代计算机非常重要! 类比硬件与软件的不同, 算法的迥异带来的意义可能更明显! 比如, 对100万个数字进行排序:

◆ **插入排序:** $T(n) = c_1 n^2$

- 计算机A: 10^9 指令/s
- 世界最好的程序员
- 机器语言

$$T(n) = 2n^2$$

$$t = \frac{2 \cdot (10^7)^2 \text{instruc}}{10^9 \text{instruc/s}} = 2 \times 10^5 \text{s} \approx 55.56 \text{h}$$

◆ **归并排序:** $T(n) = c_2 n \lg n$

- 计算机B: 10^7 指令/s
- 普通程序员
- 高级语言+低效编译器

$$T(n) = 50n \lg n$$

$$t = \frac{50 \cdot 10^7 \lg 10^7 \text{instruc}}{10^7 \text{instruc/s}} \approx 19.38 \text{m}$$

第二讲 算法分析基础

内容提要：

□ 课程学习背景

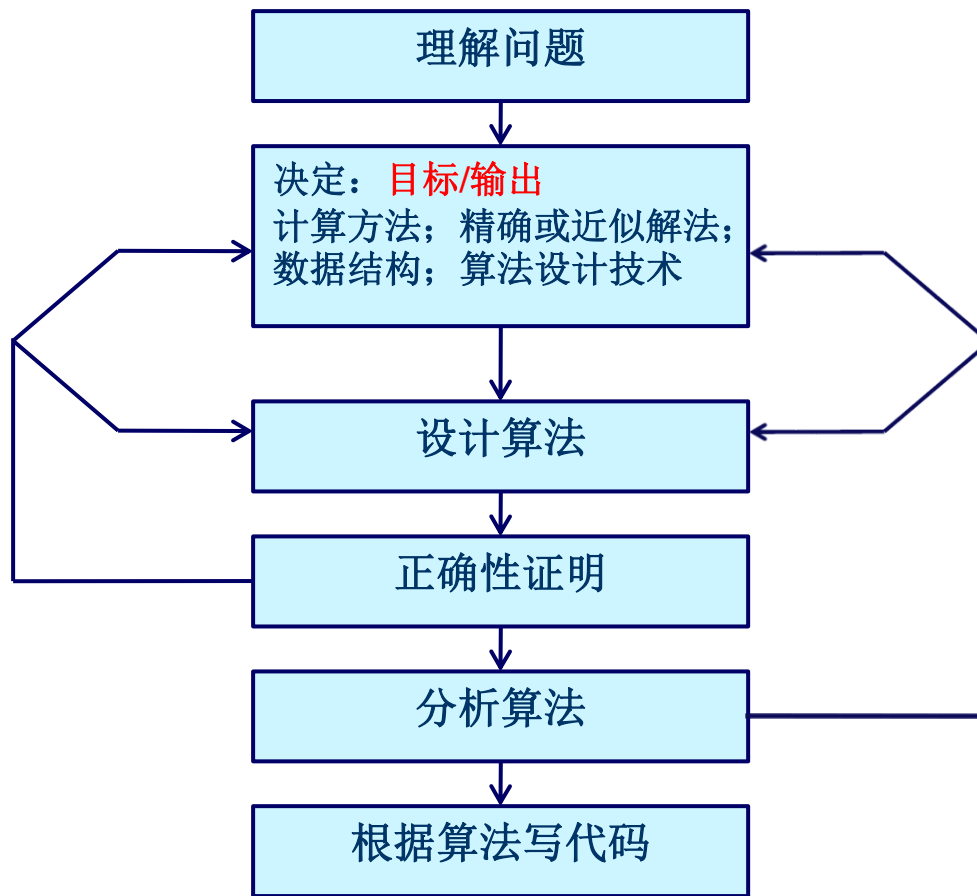
□ 算法分析基础

- ✓ 问题求解的基础
- ✓ 算法描述的方法
- ✓ 算法分析的基本框架
- ✓ 举例：插入排序

□ 算法设计策略之——分治法

问题求解的基础

□ 算法设计和分析过程



问题求解的基础

□ 求解过程说明：

- ✓ 理解问题：是否属于已知问题？确定算法输入范围；
- ✓ 了解计算设备的性能：清楚速度和计算资源的限制；
- ✓ 在精确解法和近似解法之间做选择：有时近似算法是唯一选择；
- ✓ 确定适当的数据结构
- ✓ 算法的设计技术：这是本课程学习重点和目标；
- ✓ 算法的描述：自然语言、流程图、伪代码；
- ✓ 算法的正确性证明：证明对于所有合法输入均能产生正确结果；
- ✓ 算法的分析：分析执行效率（**时间**和空间）、简单性、一般性；
- ✓ 为算法写代码：利用编程语言以计算机程序行使实现；

算法描述的方法

□ 基于文字描述

- ✓ 优点：容易理解
- ✓ 缺点：冗长、二义性

□ 基于流程图描述

- ✓ 优点：流程直观
- ✓ 缺点：缺少严密性、灵活性

□ 基于程序设计语言

- ✓ 优点：能由计算机执行
- ✓ 缺点：抽象性差，对语言要求高

□ 伪代码

算法描述的方法

- ❑ 伪代码拥有自然语言和类编程语言特性，经常被用于算法描述；
- ❑ 与真实代码(real code)的差异：
 - ✓ 对特定算法的描述更加的清晰与精确；
 - ✓ 不需要考虑太多技术细节（数据抽象、模块、错误处理等）；
 - ✓ 用伪代码可以体现算法本质；
 - ✓ 永远不会过时；

算法描述的方法

□ 伪代码的一些约定:

- ✓ 书写上的“缩进”(缩排)表示程序中的分程序(程序块)结构;
- ✓ 循环结构(while, for, repeat)和条件结构(if, then, else)与Pascal, C语言类似;
- ✓ “//” or “ \triangleright ”来表示注释;
- ✓ 利用 $i \leftarrow j \leftarrow e$ 来表示多重赋值, 等价于 $j \leftarrow e$ 和 $i \leftarrow j$.
- ✓ 变量是局部于给定过程的。
- ✓ 数组元素的访问方式: $A[i]$; $A[1 .. j] = \langle A[1], A[2], \dots, A[j] \rangle$
- ✓ 符合数据一般组织成对象, 由属性(attribute)或域(field)所组成; 域的访问是由域名后跟方括号括住的对象名形式来表示, 如length[A];
- ✓ 参数采用按值传递方式;
- ✓ 布尔操作“and”和“or”具有短路能力: 如“ x and (or) y ”: 无论 y 的值如何, 必须首先计算 x 的值。

算法分析框架

- ❑ 算法分析是指对一个算法所需要的资源进行预测，通常是对**计算时间和空间**的预测。算法分析的目的是为了从多个候选算法中选择一个最有效的算法，或去掉较差的算法。
- ❑ 进行算法分析之前，首先要确立有关实现技术的模型，通常采用**随机存取机（RAM）**计算模型。假设：
 - ✓ 指令时逐条执行的，没有并发操作；
 - ✓ 包含常用指令，每条指令执行时间为常量；
 - ✓ 数据类型有整数类型和浮点实数类型
 - ✓ 不对存储器层次进行建模
- ❑ 默认情况下，算法分析一般是指对算法**时间效率**的分析。

算法分析框架

□ 算法运行时间是指在特定输入时，所执行的基本操作数。

✓ 输入数据的规模和分布是影响算法运行时间的两个主要因素。

□ 算法时间效率分析框架：

✓ 算法时间效率用算法输入规模 n 为参数的函数来度量；

✓ 对输入规模相同情况下，有些算法的时间效率会有明显差异。对于这样的算法要区分最坏运行时间、最佳运行时间、平均运行时间；

✓ 对于大规模输入，通常只关注运行时间效率函数的增长率，即只关注函数的高阶项，而忽略低阶项和高阶项系数。

度量算法效率的方法

□ 事后统计的方法

□ 事前分析估算的方法

- ✓ 依据的算法选用何种策略
- ✓ 问题的规模
- ✓ 书写程序的语言
- ✓ 编辑程序所产生的机器代码的质量
- ✓ 机器执行指令的速度

□ **算法的存储量**：包括输入数据所占空间、程序本身所占空间和辅助变量所占空间。

□ **空间复杂度**：通常指辅助变量所占空间，是对一个算法在运行过程中临时占用的存储空间大小的量度。

时间复杂度的渐进表示法

算法中基本语句重复执行的次数是问题规模 n 的某个函数 $f(n)$,算法的时间量度记作:

$$T(n)=O(f(n))$$

表示随着 n 的增大,算法执行的时间的增长率和 $f(n)$ 的增长率相同,称渐近时间复杂度。

- ◆ 算法中重复执行次数和算法的执行时间成正比的语句
- ◆ 对算法运行时间的贡献最大

n 越大算法的执行时间越长

- ◆ 排序: n 为记录数
- ◆ 矩阵: n 为矩阵的阶数
- ◆ 多项式: n 为多项式的项数
- ◆ 集合: n 为元素个数
- ◆ 树: n 为树的结点个数
- ◆ 图: n 为图的顶点数或边数


算法的时间复杂度

算法的执行时间 = \sum 原操作的执行次数 * 原操作的执行时间

语句的**频度**指的是该语句重复执行的次数。一个算法中**所有语句的频度之和**构成该算法的运行时间。

例1:

```
for(j=1; j<=n; ++j)
    for(k=1; k<=n; ++k)
        ++x;
```

- 
- 语句“j=1”的频度是1
 - 语句“k=1、++j”的频度是n
 - 语句“j<=n”的频度是n+1
 - 语句“k<=n”的频度是n(n+1)
 - 语句“++x、++k”的频度是n²

算法运行时间为： $3*n^2+4n+2$

分析算法时间复杂度的基本方法

- ❑ 找出语句频度最大的那条语句作为基本语句
- ❑ 计算基本语句的频度得到问题规模 n 的某个函数 $f(n)$
- ❑ 取其数量级用符号“ O ”表示

```
x = 0; y = 0;
```

```
for (k = 0; k < n; k ++ )
```

```
    x ++;
```

```
for (i = 0; i < n; i++ )
```

```
    for (j = 0; j < n; j++ )
```

```
        y ++;
```

$T(n) = O(n^2)$

$f(n)=n^2$

时间复杂度——嵌套最深层语句

```
void exam ( float x[ ][ ], int m, int n ) {  
    float sum [100];int i,j;  
    for (i = 0; i < m; i++ ) {  
        sum[i] = 0.0;  
        for (j = 0; j < n; j++ )  
            sum[i] += x[i][j];  
    }  
  
    for ( i = 0; i < m; i++ )  
        printf("%.1f\n", sum [i]);  
}
```

$$T(n) = O(m*n)$$

$$f(n)=m*n$$

时间复杂度——嵌套最深层语句

例2: $N \times N$ 矩阵相乘

```
for(i=1; i<=n; i++)
```

```
    for(j=1; j<=n; j++)
```

```
    { c[i][j]=0;
```

```
        for(k=1; k<=n; k++)
```

```
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
```

```
    }
```

→ $T(n) = O(n^3)$

算法中的基本操作语句为 $c[i][j]=c[i][j]+a[i][k]*b[k][j];$



例3:

```
for( i=1; i<=n; i++)
```

```
    for (j=1; j<=i; j++)
```

```
        for (k=1; k<=j; k++)
```

```
            x=x+1;
```

定理1.1

若 $f(n)=a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$
是 m 次多项式, 则 $T(n) = O(n^m)$

忽略所有低次幂项和
最高次幂系数, 体现
出增长率的含义

$$\text{语句频度} = \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j 1 = \sum_{i=1}^n \sum_{j=1}^i j = \sum_{i=1}^n \frac{i(i+1)}{2}$$

$$\begin{aligned} &= \frac{1}{2} \left(\sum_{i=1}^n i^2 + \sum_{i=1}^n i \right) \\ &= \frac{1}{2} \left(\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right) \\ &= \frac{n(n+1)(n+2)}{6} \end{aligned}$$

时间复杂度——嵌套最深层语句

例4：分析以下程序段的时间复杂度

```
i=1;           ①  
while (i<=n)  
    i=i*2;      ②
```

$$2^{f(n)} \leq n \quad \text{即 } f(n) \leq \log_2 n, \quad \text{取最大值 } f(n) = \log_2 n$$

所以该程序段的时间复杂度 $T(n) = O(\log_2 n)$

时间复杂度分析

有的情况下，算法中基本操作重复执行的次数与**输入数据集**不同而不同

例5：顺序查找，在数组 $a[i]$ 中查找值等于 e 的元素，返回其所在位置。

```
for (i=0; i< n; i++)  
    if (a[i]==e) return i+1;  
return 0;
```

最好情况：1次

最坏情况：n

平均时间复杂度为： $O(n)$

算法分析框架

- 对于规模为 n 的任何输入，一般考察算法的最坏运行时间：
 - ✓ 最坏情况运行时间是在任何输入情况下的一个上界；
 - ✓ 对于某些算法来说，最坏情况出现还是比较频繁的，如信息检索（信息经常不存在）；
 - ✓ 大致上看，“平均情况”通常和最坏情况一样差。
- 平均运行时间（期望运行时间）
 - ✓ 概率分析技术(probabilistic analysis)
 - ✓ 随机化算法(randomized algorithm)
- 函数的**增长率**
 - ✓ 抽象简化。忽略每条语句的真实代价，用常量 c_i 来表示；进一步忽略了抽象的代价；
 - ✓ 增长率或增长量级。只考虑公式中的最高项，忽略最高项系数和低阶项；

算法分析&评价

经常涉及的渐进增长率/阶， n 为问题的规模

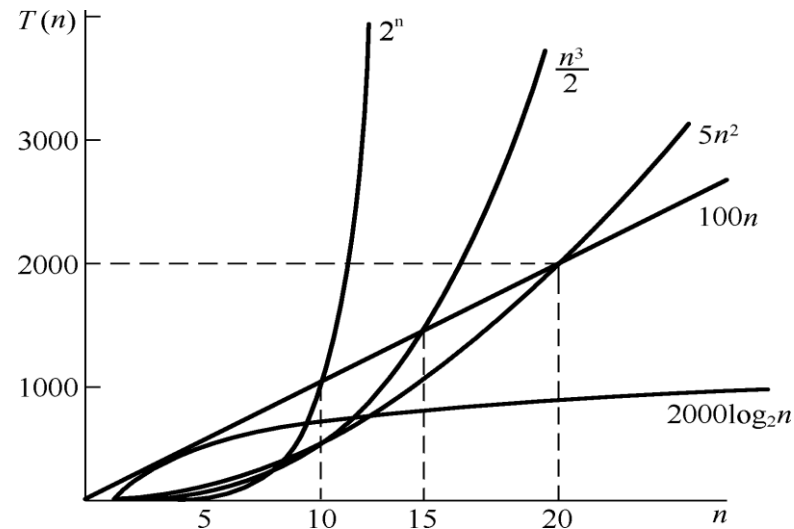
- $O(1)$ ——常量阶 $O(n)$ ——线性阶
- $O(n^2)$ ——平方阶 $O(n^k)$ ——多项式阶
- $O(\log n)$ ——对数阶 $O(2^n)$ ——指数阶

注意：

常量阶： $O(1) = O(10)$

多项式阶： $2n^3 + 3n^2 + 4n + 5 = O(n^3)$

应当尽量选择多项式阶 $O(n^k)$ 的算法



常见函数的增长率

算法分析&评价

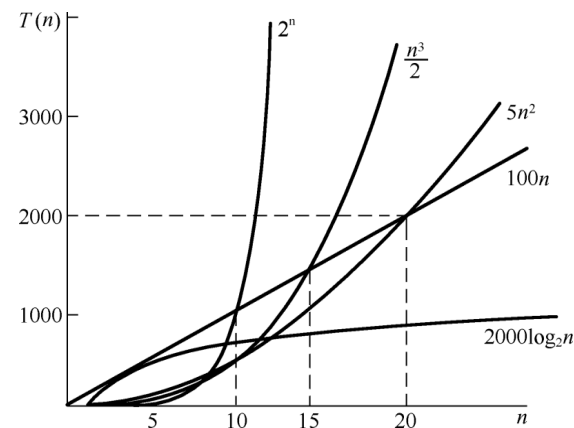
多项式时间 vs. 指数时间

□ 设每秒可做某种基本运算 10^9 次, $n=60$

| | 算法1 | 算法2 | 算法3 | 算法4 | 算法5 | 算法6 |
|-----|---------------------|-----------------------|------------------------|----------|--------|-------------------------|
| 复杂度 | n | n^2 | n^3 | n^5 | 2^n | 3^n |
| 时间 | $6 \times 10^{-8}s$ | $3.6 \times 10^{-6}s$ | $2.16 \times 10^{-4}s$ | 0.013min | 3.66世纪 | 1.3×10^{13} 世纪 |

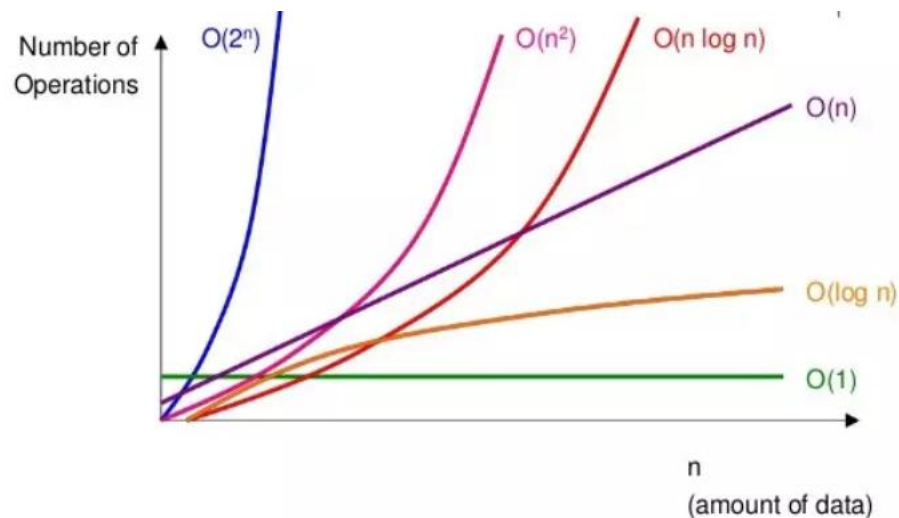
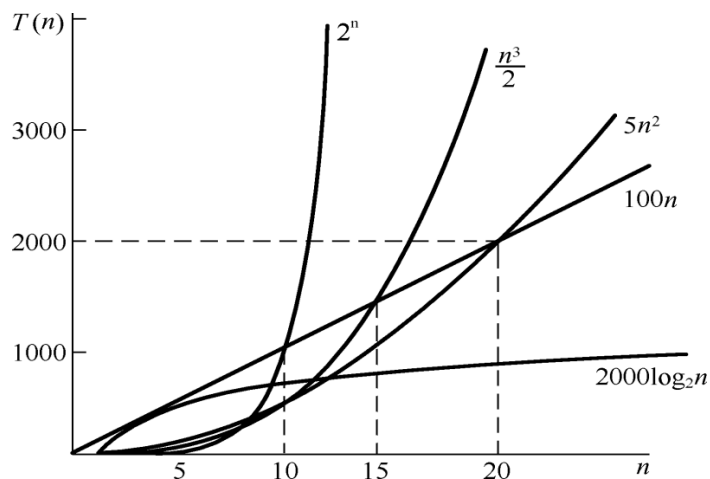
□ 两个结论

- ✓ 多项式时间的算法之间虽有差距, 一般可接受
- ✓ 指数量级时间的算法对于较大的 n 无实用价值
- ✓ 一般避免设计指数阶的算法



算法分析&评价

当n很大时，指数时间算法和多项式时间算法所需时间的非常悬殊



时间复杂度 $T(n)$ 按数量级递增顺序

复杂度低

复杂度高

| | | | | | | | | |
|--------|---------------|--------|-----------------|----------|----------|-----|----------|----------|
| 常数阶 | 对数阶 | 线性阶 | 线性对数阶 | 平方阶 | 立方阶 | ... | K次方阶 | 指数阶 |
| $O(1)$ | $O(\log_2 n)$ | $O(n)$ | $O(n \log_2 n)$ | $O(n^2)$ | $O(n^3)$ | | $O(n^k)$ | $O(2^n)$ |

时间复杂度的多种表示

- ❑ Θ , 既是上界也是下界(tight), 等于的意思。
- ❑ O , 表示上界(tightness unknown), 小于等于的意思。
- ❑ o , 表示上界(not tight), 小于的意思。
- ❑ Ω , 表示下界(tightness unknown), 大于等于的意思。
- ❑ ω , 表示下界(not tight), 大于的意思。

总结: O 是渐进上界, Ω 是渐进下界。 Θ 需同时满足大 O 和 Ω , 称为确界(必须同时符合上界和下界)。 O 极其有用, 因为它表示了最差性能。

| Letter (字母) | Bound (限制) | Growth (增长) |
|------------------------|---------------------------------------|-----------------------------------|
| (theta) Θ | upper and lower, tight ^[1] | equal ^[2] |
| (big-oh) O | upper, tightness unknown | less than or equal ^[3] |
| (small-oh) o | upper, not tight | less than |
| (big omega) Ω | lower, tightness unknown | greater than or equal |
| (small omega) ω | lower, not tight | greater than |

渐进空间复杂度

□ 研究**算法的空间效率**，只需要分析除输入和算法之外的额外空间。若所需额外空间相对于输入数据量来说是常数，则称此算法为原地工作，否则，它应当是规模的一个函数。

□ **空间复杂度(space complexity)**: 算法所需存储空间的度量，记作:

$$S(n)=O(f(n))$$

其中n为问题的规模(或大小)

□ 算法要占据的空间

- 算法本身要占据的空间，输入/输出，指令，常数，变量等
- 算法要使用的辅助空间
- 若额外空间相对于输入数据量是常数，则称该算法**原地工作**

渐进空间复杂度

例子：将一维数组a中的n个数逆序存放到原数组中。

$$S(n) = O(n)$$

【算法1】

```
for(i=0; i<n/2; i++)  
{ t=a[i];  
  a[i]=a[n-i-1];  
  a[n-i-1]=t;  
}
```

【算法2】

```
for(i=0; i<n; i++)  
  b[i]=a[n-i-1];  
for(i=0; i<n; i++)  
  a[i]=b[i];
```

$$S(n) = O(1)$$

原地工作

排序算法的时间复杂度

Array Sorting Algorithms

| Algorithm | Time Complexity | | | Space Complexity |
|-----------------------|---------------------|------------------------|-------------------|------------------|
| | Best | Average | Worst | Worst |
| <u>Quicksort</u> | $\Omega(n \log(n))$ | $\Theta(n \log(n))$ | $O(n^2)$ | $O(\log(n))$ |
| <u>Mergesort</u> | $\Omega(n \log(n))$ | $\Theta(n \log(n))$ | $O(n \log(n))$ | $O(n)$ |
| <u>Timsort</u> | $\Omega(n)$ | $\Theta(n \log(n))$ | $O(n \log(n))$ | $O(n)$ |
| <u>Heapsort</u> | $\Omega(n \log(n))$ | $\Theta(n \log(n))$ | $O(n \log(n))$ | $O(1)$ |
| <u>Bubble Sort</u> | $\Omega(n)$ | $\Theta(n^2)$ | $O(n^2)$ | $O(1)$ |
| <u>Insertion Sort</u> | $\Omega(n)$ | $\Theta(n^2)$ | $O(n^2)$ | $O(1)$ |
| <u>Selection Sort</u> | $\Omega(n^2)$ | $\Theta(n^2)$ | $O(n^2)$ | $O(1)$ |
| <u>Tree Sort</u> | $\Omega(n \log(n))$ | $\Theta(n \log(n))$ | $O(n^2)$ | $O(n)$ |
| <u>Shell Sort</u> | $\Omega(n \log(n))$ | $\Theta(n(\log(n))^2)$ | $O(n(\log(n))^2)$ | $O(1)$ |
| <u>Bucket Sort</u> | $\Omega(n+k)$ | $\Theta(n+k)$ | $O(n^2)$ | $O(n)$ |
| <u>Radix Sort</u> | $\Omega(nk)$ | $\Theta(nk)$ | $O(nk)$ | $O(n+k)$ |
| <u>Counting Sort</u> | $\Omega(n+k)$ | $\Theta(n+k)$ | $O(n+k)$ | $O(k)$ |
| <u>Cubesort</u> | $\Omega(n)$ | $\Theta(n \log(n))$ | $O(n \log(n))$ | $O(n)$ |

插入排序

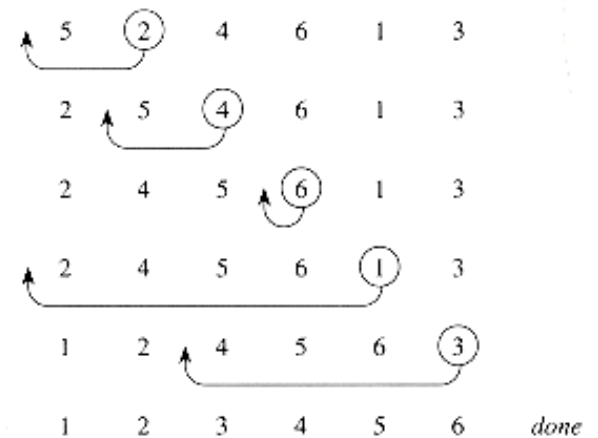
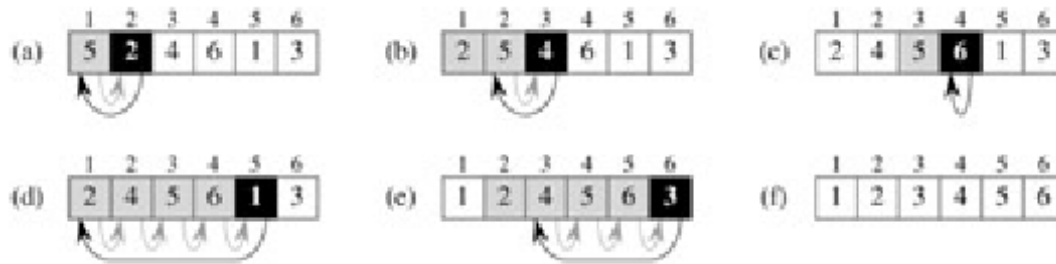
问题：把一系列数据按非递增的顺序排列

输入： n 个输入数 $\langle a_1, a_2, \dots, a_n \rangle$

输出： 输入系列的一个排序 $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，使得 $a'_1 \leq a'_2 \leq \dots \leq a'_n$



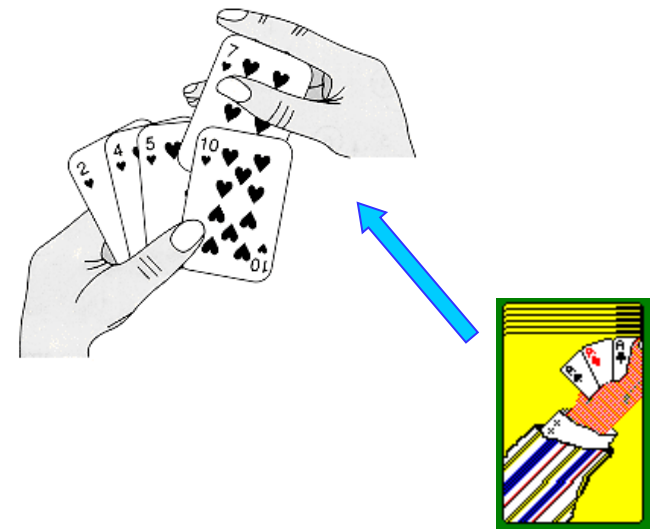
插入排序



INSERTION-SORT(A)

```

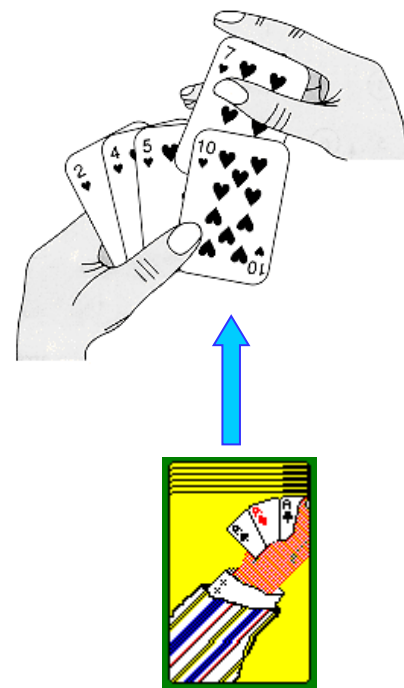
1  for( $j = 2; j \leq \text{length}[A]; j++$ ) // loop header
2  {
3       $\text{key} = A[j]$ 
4      // Insert  $A[j]$  into the sorted sequence  $A[1 .. j-1]$ 
5       $i \leftarrow j-1$ 
6      while( $i > 0 \ \&\& \ A[i] > \text{key}$ )
7      {
8           $A[i+1] = A[i]$ 
9           $i = i-1$ 
10     }
11      $A[i+1] = \text{key}$ 
12 } // loop body below
    
```



插入排序

□算法时间效率分析

| INSERTION-SORT(A) | cost | times |
|--|-------|-------|
| 1 for ($j = 2; j \leq \text{length}[A]; j++$) | c_1 | n |
| 2 { $\text{key} = A[j]$ | c_2 | $n-1$ |
| 3 // Insert $A[j]$ into the sorted sequence $A[1 .. j-1]$ | 0 | $n-1$ |
| 4 $i = j-1$ | c_4 | $n-1$ |
| 5 while ($i > 0 \ \&\& \ A[i] > \text{key}$) | c_5 | |
| 6 { $A[i+1] = A[i]$ | c_6 | |
| 7 $i = i-1$ | c_7 | |
| 8 } | | |
| 9 $A[i+1] = \text{key}$ | c_8 | $n-1$ |
| 10 } | | |



总运行时间：

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8 (n-1)$$

插入排序

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

➤ 如果数组是排好序的，则会出现**最好情况**：

$$\begin{aligned} T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) = an + b \end{aligned}$$

➤ 如果数组是逆序排序的，则会出现**最差情况**：

$$\begin{aligned} T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) = an^2 + bn + c \end{aligned}$$

此时必须将每个元素A[j]与整个已排序的子数组A[1..j-1]中的每一个元素进行比较，对j=2,3,...,n,有t_j=j. 则有：

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1, \quad \sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

一些原则上的算法设计建议

□ 保证正确性、可靠性、健壮性、可读性。

- 1) 当心那些视觉上不易分辨的操作符发生书写错误。
- 2) 算法中的变量(指针、数组)在当成右值使用,被引用前,一定要有确切的含义,或是被赋值或是经模块接口传递信息。
- 3) 算法中要当心变量发生上溢或下溢,数组的下标越界。
- 4) 写算法时就要考虑,可能出现错误的情况,提示执行错误处理算法。
- 5) 编写算法时区别问题的循环条件和停止条件,不要误用。
- 6) 注意算法中循环体,或条件体的内容,不要误把循环体内的操作写互循环体外或者出现相反的错误。

一些原则上的算法设计建议

□ 提高算法的效率。

- 1) 以提高算法的全局效率为主，提高局部效率为辅。
- 2) 在优化算法的效率时，应当先找出限制效率的“瓶颈”。
- 3) 多数情况下，时间效率和空间效率可能对立，此时应当分析哪个更重要，作出适当的折衷。
- 4) 可以考虑先选取合适的数据结构，再优化算法。
- 5) 递归过程的实现决定了递归算法的效率往往很低，费时和费内存空间。在解决问题时，如果能使用递推法解决应考虑用递推法，其效率更高些。
- 6) 注意多用数学方法，可以大大提高算法效率。
- 7) 另外还有一些细节上的问题,如:乘、除运算的效率比加、减法运算低。

第三讲 算法设计策略

内容提要:

- 课程学习背景
- 算法分析基础
- 算法设计策略之——分治法
 - ✓ 递归与分治法
 - ✓ 归并排序
 - ✓ 分治法分析

算法设计

- 对于一个问题，可以有很多中解决方法。因此，算法设计策略也有很多。
- 排序问题：
 - ◆ Bubble sort: bubbling
 - ◆ Insertion sort: incremental approach (增量靠近)
 - ◆ Merge sort: divide-and conquer (分而治之)
 - ◆ Quick sort: location (元素定位)
 -
- 分治法最差的时间比插入排序法差得多

分治法

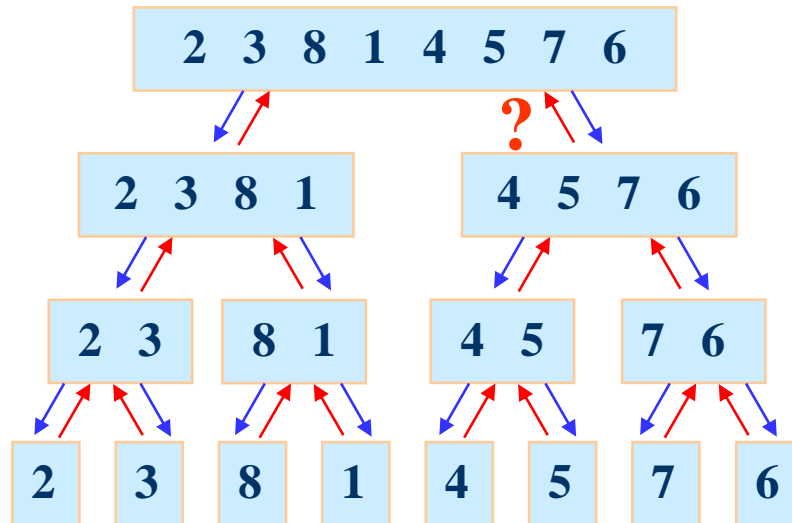
- **核心思想：**分而治之，各个击破；
- **递归结构：**为了解决一个给定的问题，算法要一次或多次地递归调用其自身来解决相关的子问题。
- **分治策略：**将原问题划分为 n 个规模较小而结构与原问题相似的子问题；递归地解决这些子问题，然后再合并其结果，就得到原问题的解。
- **三个步骤：**
 - (1) **分解 (Divide)：**将原问题分成一系列子问题；
 - (2) **解决 (Conquer)：**递归地解各个子问题。若子问题足够小，则直接求解；
 - (3) **合并 (Combine)：**将子问题的结果合并成原问题的解

归并排序

□ 归并排序算法 (Merge sort algorithm)

- ① 分解：把 n 个元素分成各含 $n/2$ 个元素的子序列；
- ② 解决：用归并排序算法对两个子序列递归地排序；
- ③ 合并：合并两个已排序的子序列以得到排序结果。

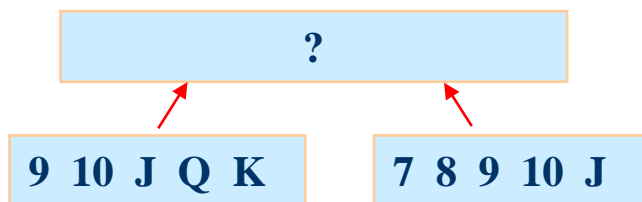
在对子序列排序时，其长度为1时递归结束。单个元素被视为是已排好序的。



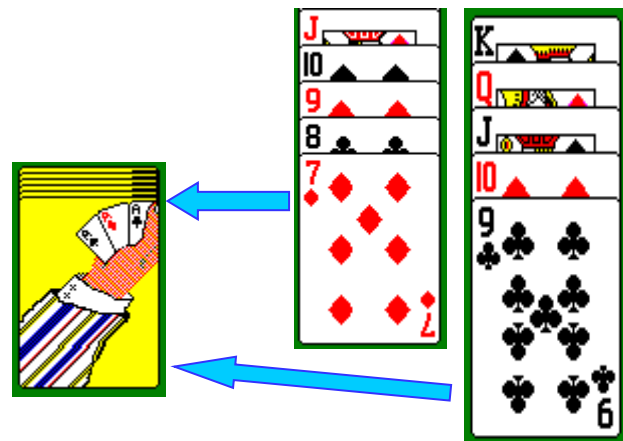
归并排序

□ $\text{MERGE}(A, p, q, r)$: 归并排序算法的关键步骤, 合并步骤中合并两个已经排序好的子序列。

- ◆ A 是个数组, p, q, r 数组中元素的下标, 且 $p \leq q < r$.
- ◆ 该过程假设子数组 $A[p \dots q]$ 和 $A[q+1 \dots r]$ 是有序的, 并将它们合并成一个已排好序的子数组代替当前子数组 $A[p \dots r]$ 。
- ◆ 合并过程:



The procedure takes time $\Theta(n)$, $n=r-p+1$ 。



归并排序

MERGE(A, p, q, r)

```
1   $n_1 \leftarrow q-p+1$ 
2   $n_2 \leftarrow r-q$ 
3  create arrays  $L[1 .. n_1+1]$  and  $R[1 .. n_2+1]$ 
4  for  $i \leftarrow 1$  to  $n_1$ 
5      do  $L[i] \leftarrow A[p+i-1]$ 
6  for  $j \leftarrow 1$  to  $n_2$ 
7      do  $R[j] \leftarrow A[q+j]$ 
8   $L[n_1+1] \leftarrow \infty$     // To avoid having to check whether either pile is empty in each
9   $R[n_2+1] \leftarrow \infty$     // basic step, a sentinel card is put on the bottom of each pile.
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13     do if  $L[i] \leq R[j]$ 
14         then  $A[k] \leftarrow L[i]$ 
15              $i \leftarrow i+1$ 
16         else  $A[k] \leftarrow R[j]$ 
17              $j \leftarrow j+1$ 
```

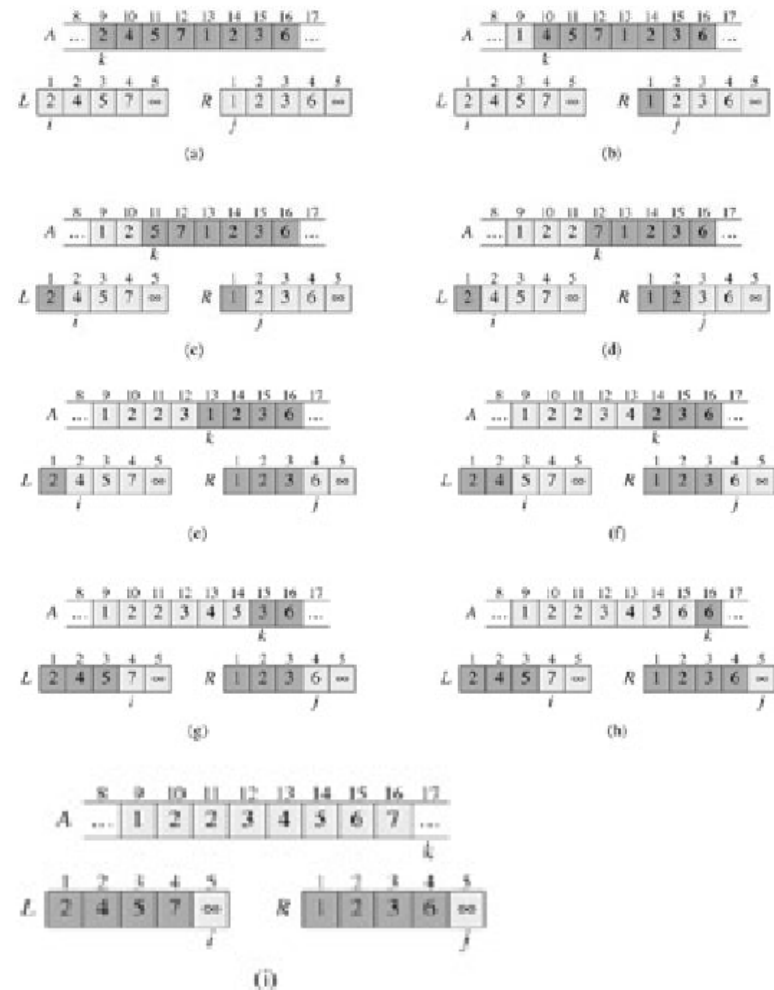
How does the subroutine work?
Next Page

归并排序

MERGE(A, p, q, r)

```

1   $n_1 \leftarrow q-p+1$ 
2   $n_2 \leftarrow r-q$ 
3  create arrays  $L[1 .. n_1+1]$  and  $R[1 .. n_2+1]$ 
4  for  $i \leftarrow 1$  to  $n_1$ 
5      do  $L[i] \leftarrow A[p+i-1]$ 
6  for  $j \leftarrow 1$  to  $n_2$ 
7      do  $R[j] \leftarrow A[q+j]$ 
8   $L[n_1+1] \leftarrow \infty$ 
9   $R[n_2+1] \leftarrow \infty$ 
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13     do if  $L[i] \leq R[j]$ 
14         then  $A[k] \leftarrow L[i]$ 
15              $i \leftarrow i+1$ 
16         else  $A[k] \leftarrow R[j]$ 
17              $j \leftarrow j+1$ 
    
```



归并排序

| MERGE(A, p, q, r) | | cost | times |
|-------------------|---|------|---------|
| 1 | $n_1 \leftarrow q-p+1$ | c | 1 |
| 2 | $n_2 \leftarrow r-q$ | c | 1 |
| 3 | create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$ | c | 1 |
| 4 | for $i \leftarrow 1$ to n_1 | c | n_1+1 |
| 5 | do $L[i] \leftarrow A[p+i-1]$ | c | n_1 |
| 6 | for $j \leftarrow 1$ to n_2 | c | n_2+1 |
| 7 | do $R[j] \leftarrow A[q+j]$ | c | n_2 |
| 8 | $L[n_1+1] \leftarrow \infty$ | c | 1 |
| 9 | $R[n_2+1] \leftarrow \infty$ | c | 1 |
| 10 | $i \leftarrow 1$ | c | 1 |
| 11 | $j \leftarrow 1$ | c | 1 |
| 12 | for $k \leftarrow p$ to r | c | $r-p+2$ |
| 13 | do if $L[i] \leq R[j]$ | c | $r-p+1$ |
| 14 | then $A[k] \leftarrow L[i]$ | c | x |
| 15 | $i \leftarrow i+1$ | c | x |
| 16 | else $A[k] \leftarrow R[j]$ | c | $r-$ |
| 17 | $j \leftarrow j+1$ | c | $r-$ |
| | $p+1-x$ | | |
| | $p+1-x$ | | |

$$r-p+1 \\ = n_1 + n_2 = n$$

$$1 \leq x \leq n_1$$

$$\Theta(n_1+n_2)=\Theta(n)$$

归并排序

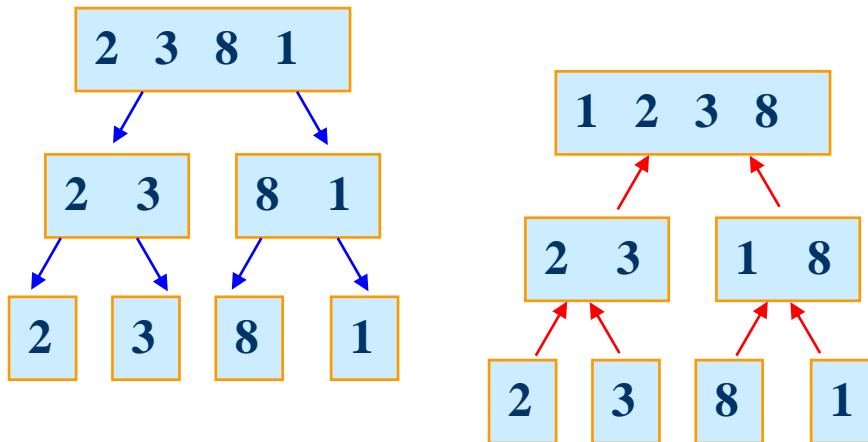
- 将MERGE过程作为合并排序中的一个子过程来使用。下面过程MERGE-SORT(A, p, r)对子数组 $A[p..r]$ 进行排序. 如果 $p \geq r$, 则该子数组中至多只有一个元素, 当然就是已排序的。否则, 分解步骤就计算出一个下标 q , 将 $A[p..r]$ 分成 $A[p..q]$ 和 $A[q+1..r]$, 各含 $\lceil n/2 \rceil$ 个元素。

```
MERGE-SORT( $A, p, r$ )
1  if  $p < r$ 
2      Then  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3          MERGE-SORT( $A, p, q$ )
4          MERGE-SORT( $A, q+1, r$ )
5  MERGE( $A, p, q, r$ )
```

归并排序

MERGE-SORT(A, p, r)

```
1  if  $p < r$ 
2    Then  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3    MERGE-SORT( $A, p, q$ )
4    MERGE-SORT( $A, q+1, r$ )
5    MERGE( $A, p, q, r$ )
```



MERGE-SORT($A, 1, 4$)

1 if $1 < 4$

2 Then $q \leftarrow \lfloor (1+4)/2 \rfloor = 2$

3 MERGE-SORT($A, 1, 2$)

1 if $1 < 2$

2 Then $q \leftarrow \lfloor (1+2)/2 \rfloor = 1$

3 MERGE-SORT($A, 1, 1$)

1 if $1 < 1$

4 MERGE-SORT($A, 2, 2$)

1 if $2 < 2$

5 MERGE($A, 1, 1, 2$)

4 MERGE-SORT($A, 3, 4$)

1 if $3 < 4$

2 Then $q \leftarrow \lfloor (3+4)/2 \rfloor = 3$

3 MERGE-SORT($A, 3, 3$)

1 if $3 < 3$

4 MERGE-SORT($A, 4, 4$)

1 if $4 < 4$

5 MERGE($A, 3, 3, 4$)

5 MERGE($A, 1, 2, 4$)

Jinan University, China

分治法分析

□ 当一个算法含有对其自身的递归调用时，其运行时间总可以用一个递归方程（或递归式）来表示。该方程通过描述子问题与原问题的关系，来给出总的运行时间。我们可以利用数学工具来解递归式，并给出算法性能的界。

□ 分治法给出的时间：

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

- $D(n)$ 是把原问题分解为子问题所花的时间；
- $C(n)$ 是把子问题的解合并为原问题的解所花的时间；
- $T(n)$ 是一个规模为 n 的问题的运行时间。

```
MERGE-SORT( $A, p, r$ )
1   if  $p < r$ 
2       Then  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3           MERGE-SORT( $A, p, q$ )
4           MERGE-SORT( $A, q+1, r$ )
5           MERGE( $A, p, q, r$ )
```

分治法分析

□ 为简化算法分析，通常假设 n 为2的幂次，使得每次分解产生的子序列长度恰为 $n/2$ 。这一假设并不影响递归式解的增长量级。

□ 合并排序 n 个数最坏运行时间：

① 当 $n=1$ 时，合并排序一个元素的时间是个常量；

② 当 $n>1$ 时，运行时间分解如下：

➤ 分解：仅仅是计算出子数组的中间位置，需要常量时间， $D(n) = \Theta(1)$ ；

➤ 解决：递归地求解两个规模为 $n/2$ 的子问题，时间为 $2T(n/2)$ ；

➤ 合并：MERGE过程的运行时间为 $C(n) = \Theta(n)$ 。

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

归并排序

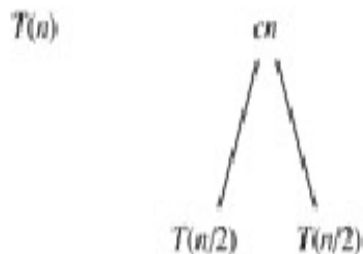
➤ 递归式重写为：

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

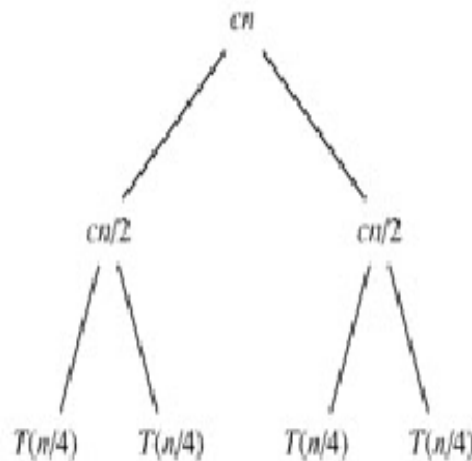


$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

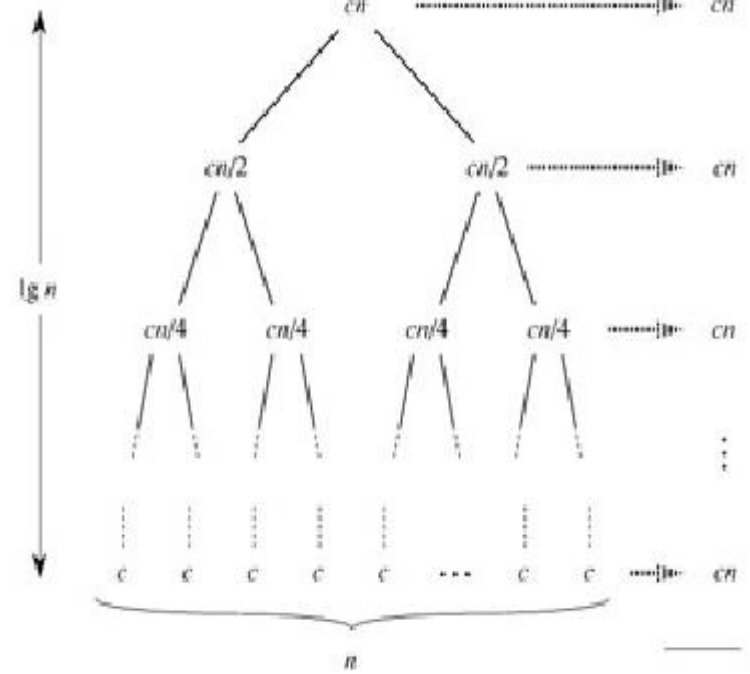
➤ 递归式求解如下：



(a)



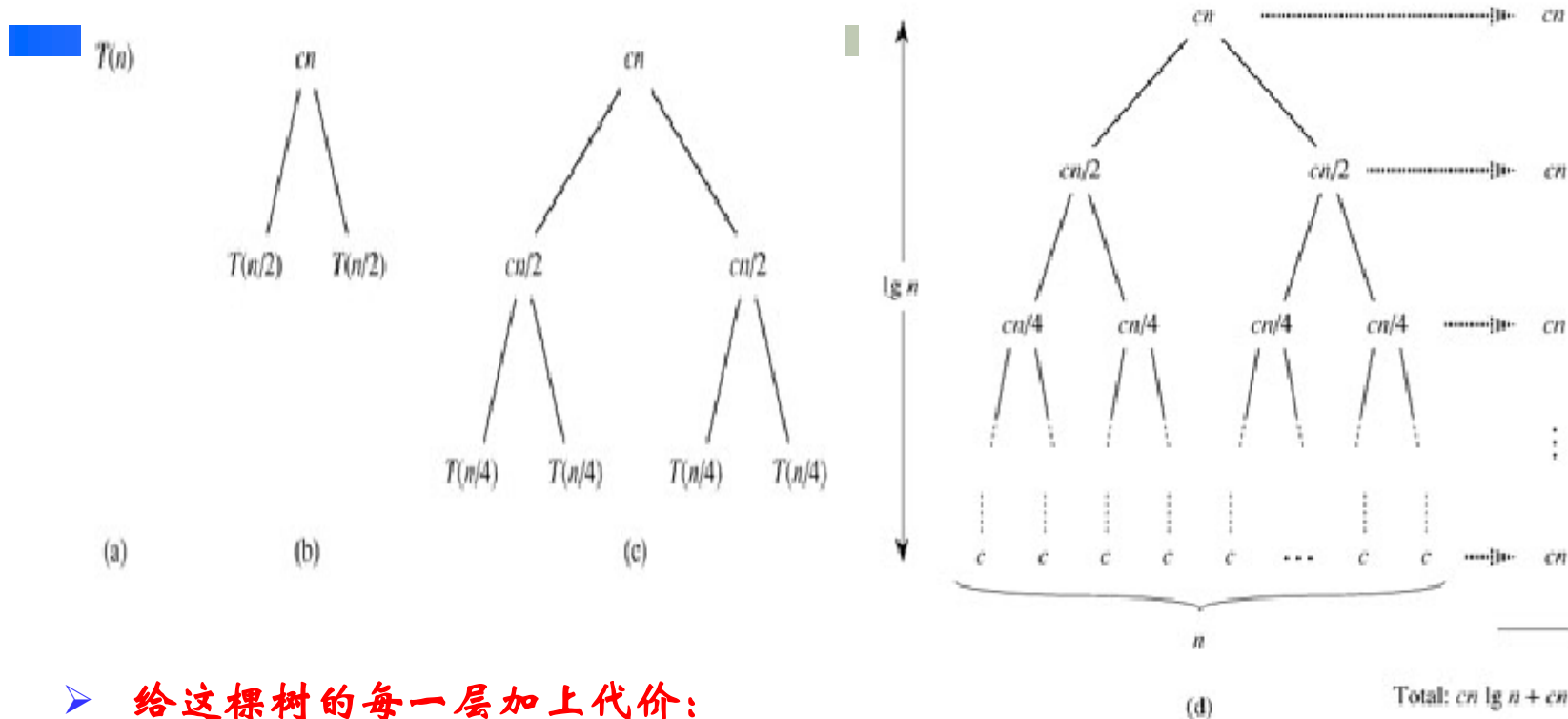
(b)



(c)

Total: $cn \lg n + cn$

分治法分析



- 给这棵树的每一层加上代价：

最底层之下的第 i 层有 2^i 个节点，每一个代价是 $c(n/2^i)$ 第 i 层总代价是 $2^i c(n/2^i) = cn$ 。
最底层有 n 个节点，每一个节点的代价是 c ，该层总代价为 cn ；

- 树总共层数是 $\lg n + 1$ 层，每一层代价都是 cn ，所以总代价为：

$$cn(\lg n + 1) = cn \lg n + cn = \Theta(n \lg n)$$

- 最坏情况明显比插入排序要好 $\Theta(n^2)$

谢谢！

Q & A

作业：

1. 3个不同算法的复杂度分析
2. 写出选择排序的算法

