算法基础

庄连生

Email: { *lszhuang@ustc.edu.cn* }

***Spring 2019*，USTC**

课程信息

- 课时安排：90学时(60+30)，3.5学分
- 授课时间：3-16周，2 (6, 7)，5 (3, 4), 3C103教室
- 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，1974, Turing Prize
  - *《计算机算法设计与分析》(第3版)，王晓东，电子工业出版社，2007
  - *《算法设计与分析基础》，Anany Levitin著，潘彦译，清华大学出版社，2007
- 课程助教
  - ✓ 沈三景, ssjxjx@mail.ustc.edu.cn，大四
  - ✓ 唐明宇, tmy528@mail.ustc.edu.cn，研一
  - ✓ 白建峰, bjf@mail.ustc.edu.cn，研二

http://staff.ustc.edu.cn/~lszhuang/alg

课程信息

授课形式

- ✓ **课堂讲解**（60学时）：基本理论讲解、基本方法的介绍分析；
- ✓ **上机实践**（30学时）：经典算法的上机实验；
- ✓ 课程作业

考核形式

- ✓ 期末考试**(闭卷)**：60%
- ✓ 课堂作业+上机作业：30%
- ✓ 出勤+课堂纪律：10%

选课要求

- ✓ 学过离散数学、基本概率论
- ✓ 具备编程技能（C、C++、Python等）

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

教学内容

- Part 1 基础知识
  - ✓ 课程学习背景
  - ✓ 算法分析基础
- Part 2 排序和顺序统计学
  - ✓ 归并排序、堆排序、快去排序
  - ✓ 计数排序、基数排序、桶排序
- Part 3 高级数据结构
  - ✓ 红黑树、数据结构的扩张、B树
- Part 4 算法设计策略
  - ✓ 分治法、动态规划法、贪心法、回溯法、分枝限界法
- Part 5 算法研究问题
  - ✓ NP完全问题，有关数论的算法，……

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
  - ✓为什么要学习算法？
  - ✓ 算法相关概念
- 算法分析基础
- 算法设计策略之——分治法

为什么要学习算法

- 算法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！

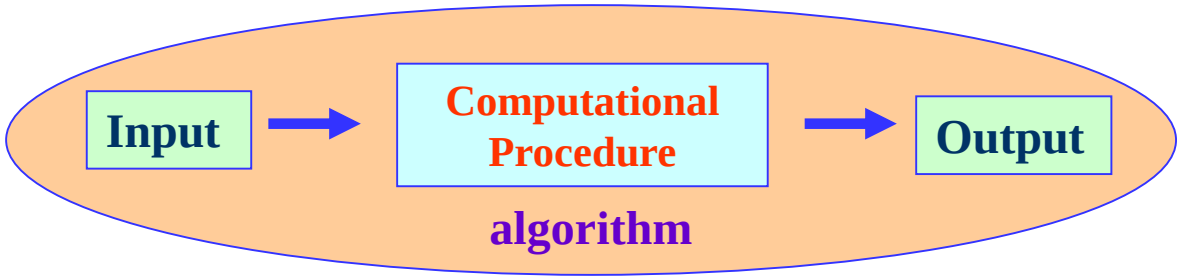
- 学习算法可以开发人们的分析能力

算法是解决问题的一类特殊方法，它是经过对问题的准确理解和定义获取答案的过程。

- 是你获得高薪职位的敲门砖！

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。即，算法就是一系列的计算步骤，用来将输入数据转换成输出结果。



< 31,41,59,26,12,58 >

< 31,41,59,26,12,58 >

<31,41,26,12,58,59>

<31,26,12,41,58,59>

……

“冒泡”排序是个计算过程

<12,26,31,41,58,59>

什么是算法？

- 一个算法通常具有如下特征：
  - ① 输　入：一个算法具有一个或者多个取自指定集合的输入值；
  - ② 输　出：对每一次输入，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：对每一次输入，算法都必须在有限步骤内结束；
  - 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。

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个数组成的一个序列 $<a_1,a_2,\cdots,a_n>$

输出: 对输入系列的一个排列(重排）$<a_1',a_2',\cdots,a_n'>$，使得 $a_1' \le a_2' \le \cdots \le a_n'$

- **排序问题的一个实例：**

Input: <31,41,59,26,41,58>　——　Output: <26,31,41,41,58,59>

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**：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

相关概念：正确算法与不正确算法

正确的算法

如果一个算法对问题每一个输入实例，都能输出正确的结果并停止，则称它为正确的。

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。

作为一种技术的算法

- 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：** $T(n)=c_1n^2$
  - 计算机A: 10^9 指令/s
  - 世界最好的程序员
  - 机器语言

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

$$t=\frac{2\cdot(10^7)^2\text{instruc}}{10^9\text{instruc/s}}=2\times10^5\text{s}\approx55.56\text{h}$$

- **归并排序：** $T(n)=c_2n\lg n$
  - 计算机B: 10^7 指令/s
  - 普通程序员
  - 高级语言+低效编译器

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

$$t=\frac{50\cdot10^7\lg10^7\,\text{instruc}}{10^7\text{instruc/s}}\approx19.38\text{m}$$

算法与程序的区别

- 算法的概念和程序十分相似，但实际上有很大不同。程序并不都满足算法所要求的上述特征，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，算法也常常称为是一个能行过程。一个函数如果可以用一个算法来计算，那么我们称该函数是能行可计算的。
- 如操作系统是一种程序，而不是算法。

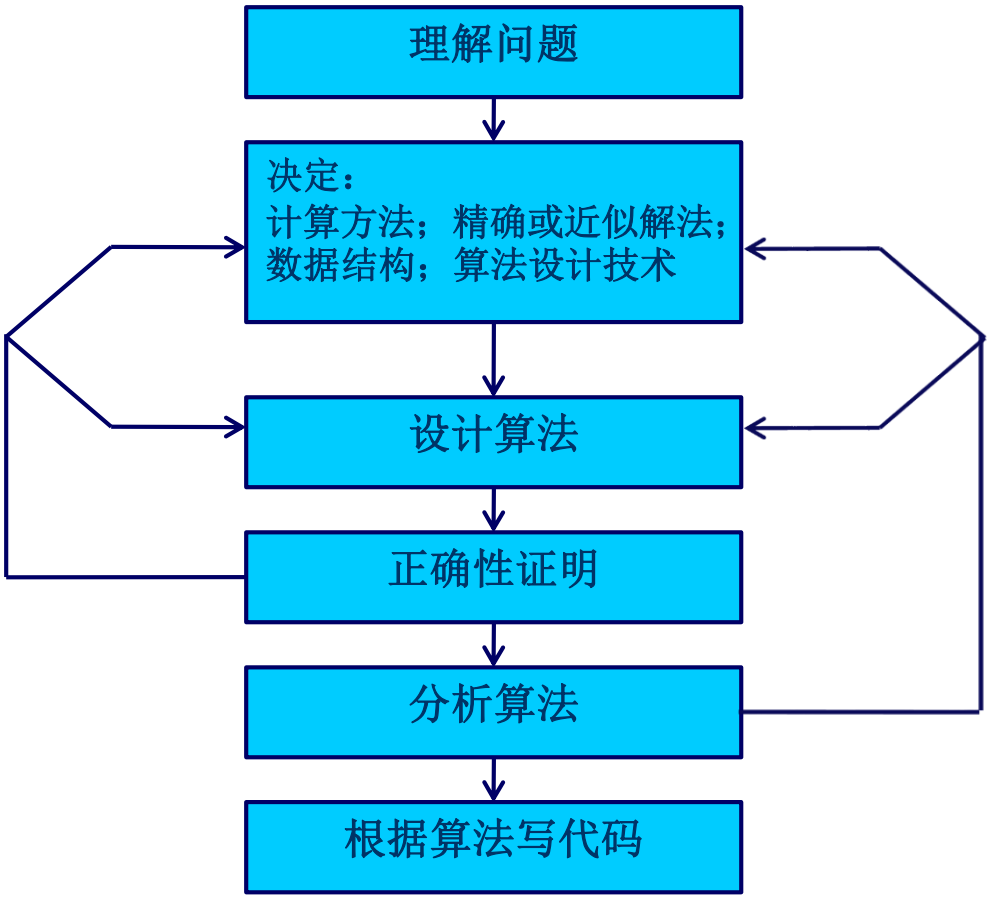
第一讲 算法入门

内容提要：

- 课程学习背景
- 算法分析基础
  - ✓ 问题求解基础
  - ✓ 算法描述方法
  - ✓ 算法分析基本框架
  - ✓举例：插入排序
- 算法设计策略之——分治法

问题求解基础

- 算法设计和分析过程



问题求解基础

- **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 “▷”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: $A[i]$；$A[1..j] = <A[1], A[2], \ldots, A[i]>$
- 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。

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ci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；

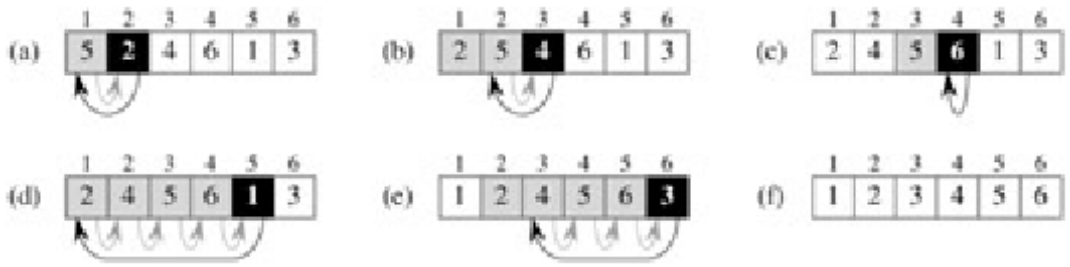
插入排序

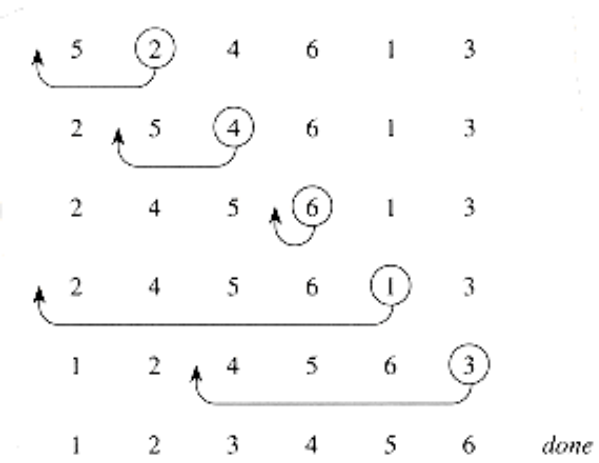
问题： 把一系列数据按非递增的顺序排列

输入： n 个输入数 $<a_1,a_2,\cdots,a_n>$

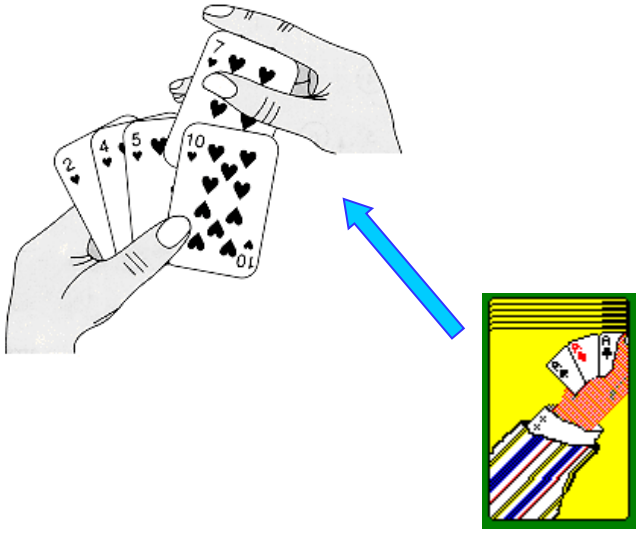
输出： 输入系列的一个排序 $<a_1',a_2',\cdots,a_n'>$，使得 $a_1' \le a_2' \le \cdots \le a_n'$

插入排序





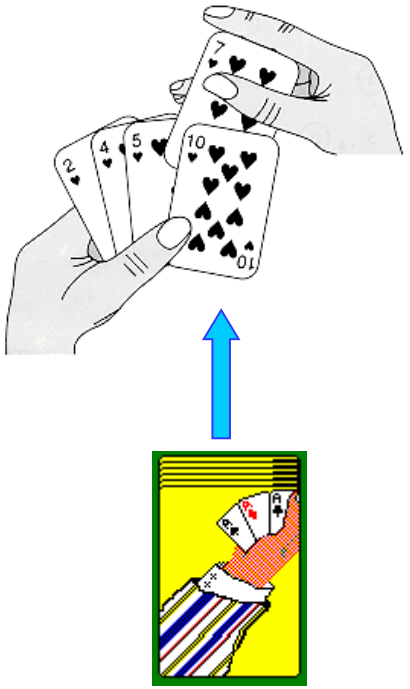

```
INSERTION-SORT(A)
1     for( j = 2; j <=length[A]; j++) // loop header
2     {       key = A[j]
3             // Insert A[j] into the sorted sequence A[1 .. j-1]
4             i←j-1
5             while( i > 0 && A[i] > key)
6             {       A[i+1] = A[i]
7                     i = i-1
8             }
9             A[i+1] = key
10    }     // loop body below
```



插入排序

算法时间效率分析

INSERTION-SORT(*A*)		cost	times
1	for($j = 2; j <= length[A]; j++$)	c_1	n
2	{ $key = A[j]$	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($i > 0$ && $A[i] > key$)	c_5	
6	{ $A[i+1] = A[i]$	c_6	
7	$i = i$-1	c_7	
8	}		
9	$A[i+1] = key$	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：

$$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5(n-1)+c_8(n-1)$$
$$=(c_1+c_2+c_4+c_5+c_8)n-(c_2+c_4+c_5+c_8)=an+b$$

- 如果数组是逆序排序的，则会出现最差情况：

$$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5(\frac{n(n+1)}{2}-1)$$
$$+c_6(\frac{n(n-1)}{2})+c_7(\frac{n(n-1)}{2})+c_8(n-1)=an^2+bn+c$$

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tj=j. 则有：

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

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
  - ✓ 分治法介绍
  - ✓ 归并排序
  - ✓ 分治法分析

算法设计

- 对于一个问题，可以有很多种解决方法。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好得多

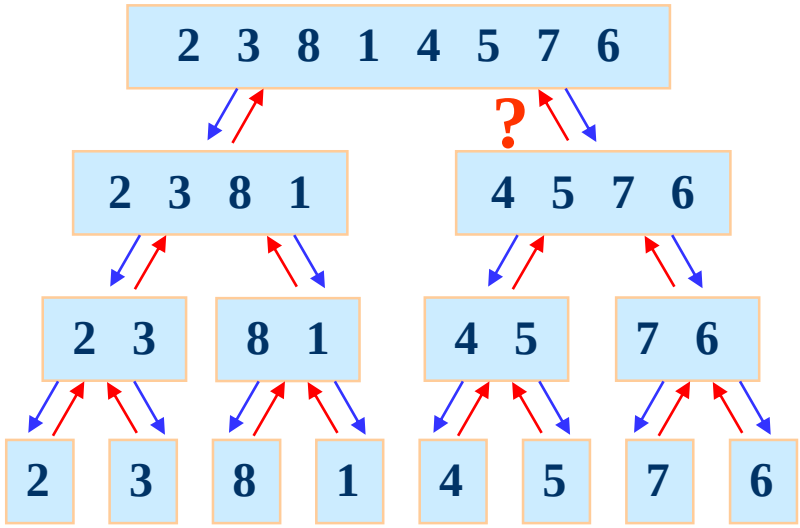
分治法

- **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

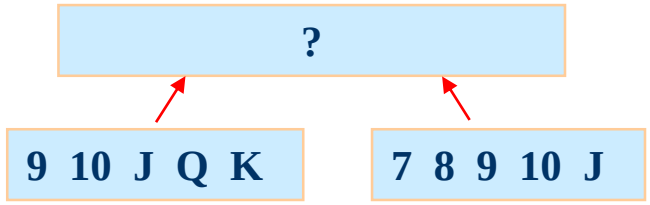
- 归并排序算法（Merge sort algorithm）
  - ① **分解**：把n个元素分成各含n/2个元素的子序列；
  - ② **解决**：用归并排序算法对两个子序列递归地排序；
  - ③ **合并**：合并两个已排序的子序列以得到排序结果。

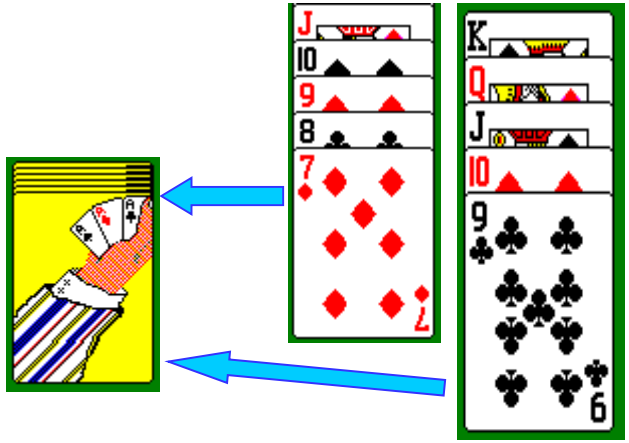
在对子序列排序时，其长度为1时递归结束。单个元素被视为是已排好序的。



归并排序

- 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$p \le q < r$.
  - 该过程假设子数组 $A[p .. q]$ 和 $A[q+1 .. r]$ 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 $A[p .. r]$。
  - 合并过程:





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

归并排序

MERGE(A, p, q, r)

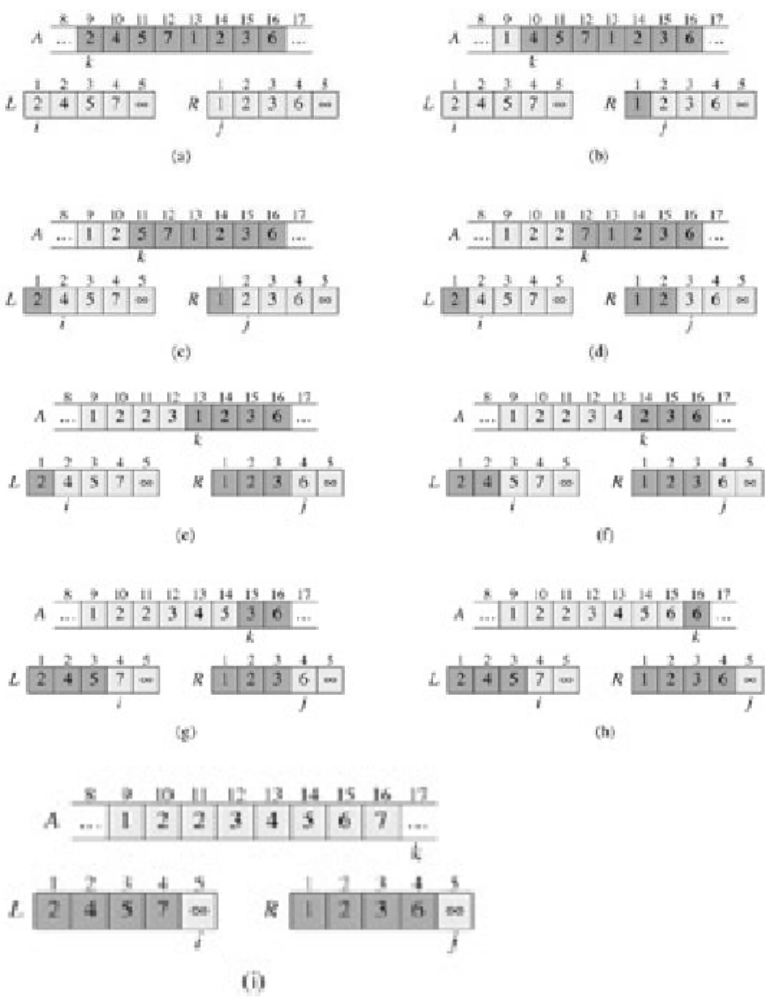
```
1    n1 ← q-p+1
2    n2 ← r-q
3    create arrays L[1 .. n1+1] and R[1 .. n2+1]
4    for i←1 to n1
5          do L[i]←A[p+i-1]
6    for j←1 to n2
7          do R[j]←A[q+j]
8    L[n1+1]←∞      // To avoid having to check whether either pile is empty in each
9    R[n2+1]←∞      // basic step, a sentinel card is put on the bottom of each pile.
10   i←1
11   j←1
12   for k←p to r
13        do if L[i]≤R[j]
14              then A[k]←L[i]
15                   i←i+1
16              else A[k]←R[j]
17                   j←j+1
```

How does the subroutine work? Next Page

归并排序

```
MERGE(A, p, q, r)
1    n1 ← q-p+1
2    n2 ← r-q
3    create arrays L[1 .. n1+1] and R[1 .. n2+1]

4    for i←1 to n1
5          do L[i]←A[p+i-1]
6    for j←1 to n2
7          do R[j]←A[q+j]
8    L[n1+1]←∞
9    R[n2+1]←∞
10   i←1
11   j←1
12   for k←p to r
13        do if L[i]≤R[j]
14              then A[k]←L[i]
15                   i←i+1
16              else A[k]←R[j]
17                   j←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	$r-p+1-x$
17	$j \leftarrow j+1$	c	$r-p+1-x$

$r-p+1$

$= n_1 + n_2 = n$

$1 =< x =< n_1$

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

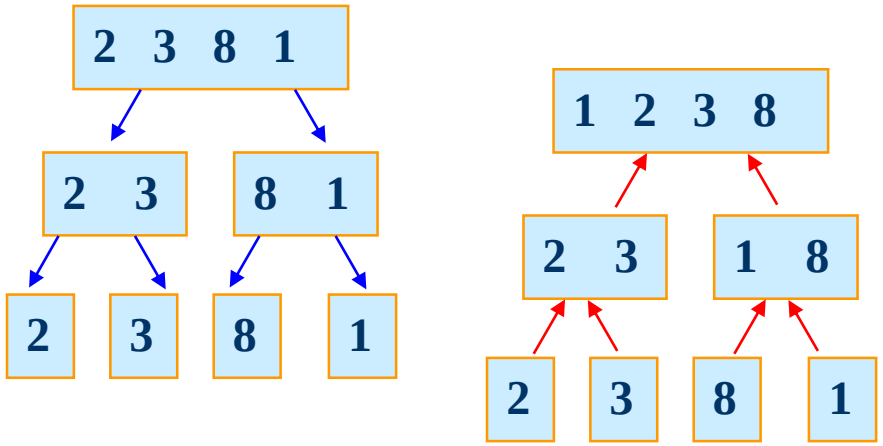
归并排序

- 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$p \geqslant r$，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 ← ⌊(p+r)/2⌋
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 ← ⌊(p+r)/2⌋
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 ← ⌊(1+4)/2⌋ = 2
3        MERGE-SORT(A, 1, 2)
            1  if 1 < 2
            2  Then q ← ⌊(1+2)/2⌋ = 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 ← ⌊(3+4)/2⌋ = 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)
```

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：

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

- 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 ← ⌊(p + r)/2⌋
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$C(n)=\Theta(n)$。

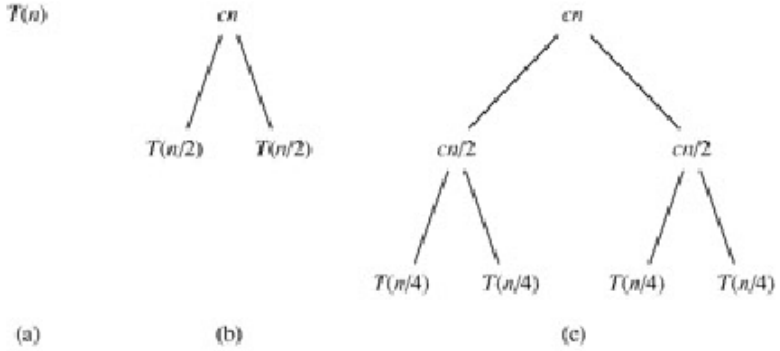
$$T(n)=\begin{cases}\Theta(1) & \text{if } n=1\\ 2T(n/2)+\Theta(n) & \text{if } n>1\end{cases}$$

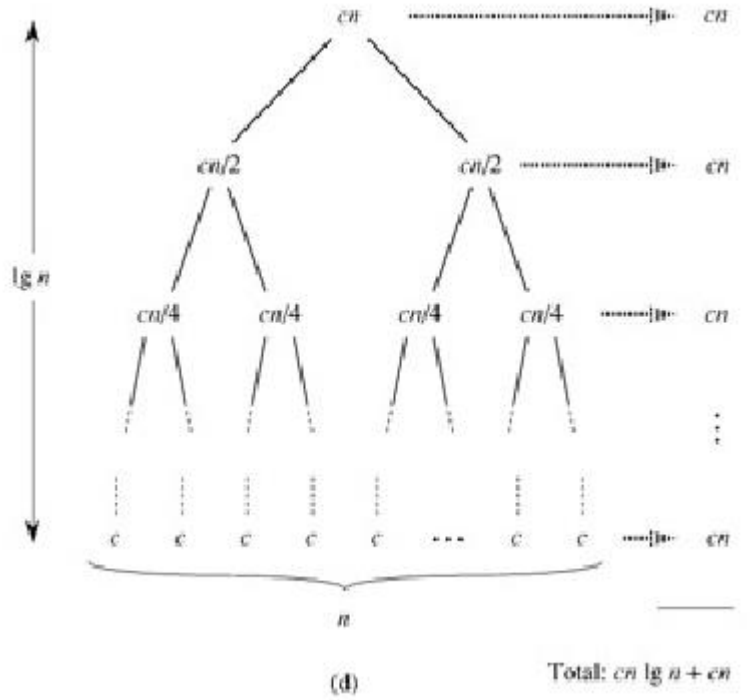
归并排序

- 递归式重写为：

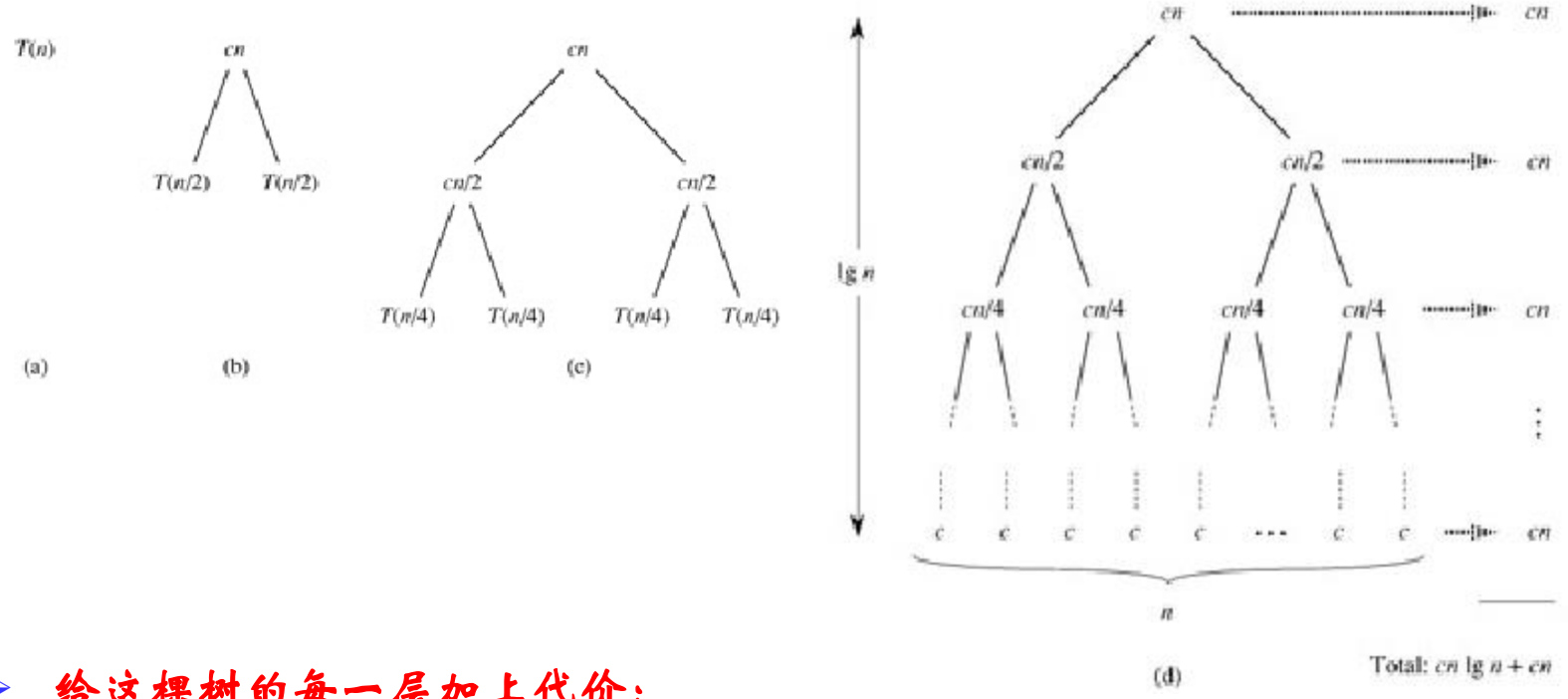
$$T(n)=\begin{cases}\Theta(1) & \text{if } n=1\\ 2T(n/2)+\Theta(n) & \text{if } n>1\end{cases}\quad\Rightarrow\quad T(n)=\begin{cases}\Theta(1) & \text{if } n=1\\ 2T(n/2)+cn & \text{if } n>1\end{cases}$$

- 递归式求解如下：





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