算法设计与分析

主讲人：吴庭芳

Email: tfwu@suda.edu.cn

苏州大学

计算机科学与技术学院

课程信息

群名称：2025-2026秋季-算法设计与

群　号：787047803

课程信息

- 课程名称：算法设计与分析
- 课程安排：34 学时
- 授课时间：1-18 周，周二（6-7）
- 教材信息：
  - 《算法导论》(第3版)，Thomas 等著，机械工业出版社，2006

课程信息







- **授课形式**
  - ✓ **课堂讲解**（34学时）：一些算法理论知识的讲解
- **考核形式**
  - ✓ 作业+考勤（20%）
  - ✓ 期中考试（20%，时间：第9-10周，闭卷）
  - ✓ 期末考试（60%，闭卷）
- **课程基础**
  - ✓ 数据结构
  - ✓ 具备编程技能（C、C++、Python等）
  - ✓ 一定的数学基础：高数、离散、概率

教学内容







教学进度表

周次	章节名称	内容提要	授课时数	作业及要求	备注
1-3	算法基础	算法运行时间的渐近表示	6	算法设计、分析与实现	
4-7	分治算法	分治的基本原理、递归式求解、经典分治算法	8	算法设计、分析与实现	
8-11	动态规划	动态规划的基本原理、经典的动态规划算法	8	算法设计、分析与实现	
12-15	贪心算法	贪心算法的基本原理，经典的贪心算法	8	算法设计、分析与实现	
16-17	进阶算法	回溯、网络流、NP、近似算法、随机算法	6	阅读文献、查阅资料	







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 —— 插入排序
- 算法设计策略 —— 归并排序

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

学习算法可以提高分析问题和解决问题能力

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

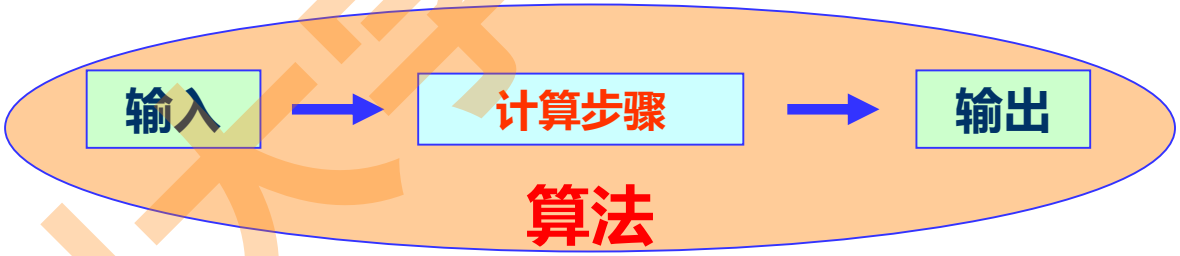
什么是算法?







- **简单说来，算法是问题的程序化解决方案**
- **定义：**算法就是一个定义良好（well-defined）的可计算过程，它取一个或者一组值作为输入，并产生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>

……

冒泡排序算法

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

什么是算法?

- **一个算法通常具有如下五个特征:**

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 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；

④ 确定性：算法的每一个指令步骤都是明确的，不会出现二义性；

⑤ 可行性：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步，即每个计算步骤都可以在有限时间内完成。

相关概念：问题和问题实例







- **问题（Problem）**：问题描述了期望的输入/输出关系，可以用通用语言来描述，例如：
  - 排序问题——将一系列数按非降序进行排列，其形式化定义如下：

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

$$a_1' \le a_2' \le \cdots \le a_n'$$

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的每一个实例，都能输出正确的结果并停止，则称它是正确的

不正确的算法

- ✓ 可能算法根本不会停止
- ✓ 算法停止时输出的不是预期的结果
- ✓ 不正确的算法也并非绝对无用，如果算法的错误率可以控制，也是有用的和有效的（比如 NP 完全问题的近似算法）

作为一种技术的算法

- 对比硬件与软件的影响，**算法的迥异带来的意义可能更明显**！

  比如，对 1000 万个数字进行排序：

- **插入排序：** $T(n)=c_1n^2$
  - 硬件：计算机A：10^9 指令/s
  - 软件：世界最好的程序员
  - 软件：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

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

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







第一讲 算法入门

内容提要：

- 课程学习背景
- 算法分析基础 —— 插入排序
  - ✓ 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 —— 归并排序

问题求解基础







- 算法设计和分析过程

算法描述

- 伪代码是**非真实代码**，书写类似于 C/C++，Java 等实际编程语言，主要用于描述算法的计算步骤而非放在计算机上运行，没有固定的标准
- 与真实代码（real code）的差异：
  - 不需要考虑太多技术细节（比如数据抽象、错误处理等）
  - 对特定算法的描述更加的清晰与精确
  - 用伪代码可以体现算法本质

算法描述方法

- **伪代码书写的一些约定：**
  - ✓ 书写上的“缩进”表示程序块结构；
  - ✓ 循环结构（while, for, repeat）和条件结构 （if, then, else）与 C 语言类似；
  - ✓ “// ” 来表示注释；
  - ✓ 利用 $i = j = e$ 来表示多重赋值，等价于 $j = e$ 和 $i = j$；
  - ✓ 变量是局部于给定过程的；
  - ✓ 数组元素的访问方式：$A[i]$; $A[1..j] = < A[1], A[2], \ldots, A[j] >$；
  - ✓ 复合数据一般被组织成对象，由属性（attribute）所组成；属性的访问是由对象后跟一个点再跟属性形式来表示，如 A.length；
  - ✓ 参数采用按值传递方式；
  - ✓ 布尔操作 “and” 和“or”具有短路能力：例如 “ x and (or) y ”，无论 y 的值如何，必须首先计算 x 的值。

插入排序





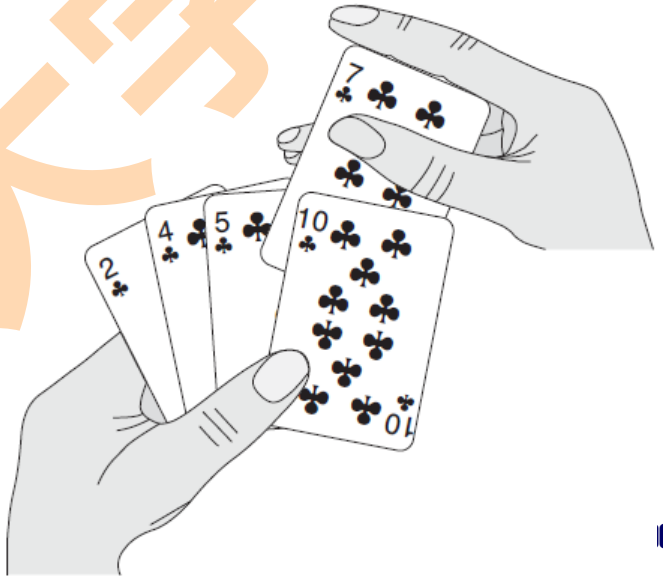


排序问题: 将一系列数据按非降顺序排列

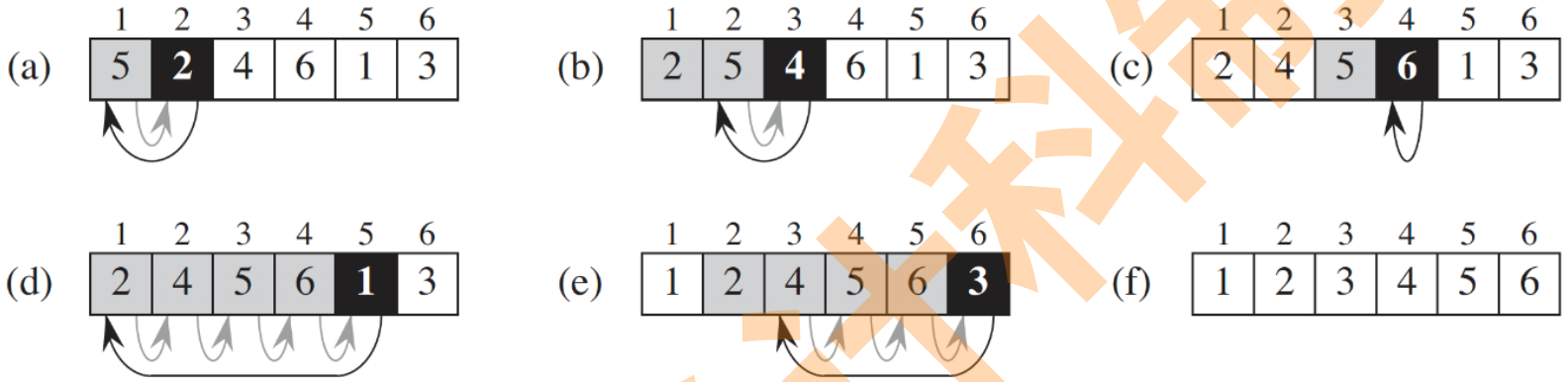
输入： n 个输入数 $<a_1,a_2,\cdots,a_n>$

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

- **思想：通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入**



插入排序




```
INSERTION-SORT(A)
1    for j = 2 to A.length      // 从第二个元素到最后一个元素是未排序序列
2    {      key = A[j]          // 待排序的数据
3           // Insert A[j] into the sorted sequence A[1.. j-1]
4           i = j – 1           // 已排序序列中最右边元素下标
5           while i > 0 and A[i] > key
6           {
7                 A[i+1] = A[i]
8                 i = i - 1
9           }
10          A[i +1] = key       // 直到找到待排序元素大于或者等于已排序元素的位置
11   }  // loop body below
```

插入排序







□ 循环不变式 （Loop invariants）

以 INSERTION-SORT 为例，for 循环中，循环变量为 j，循环过程具有以下性质（j 是当前待排序元素所在位置的下标）：

子数组 A[1.. j-1] 是已经被排序好的序列

这一性质，在 j 被赋予初始值 2，首次进入循环之前成立；而且每次循环之后（j 加了 1）、进入下一次循环之前也成立

把这种在第一次进入循环之前成立，并且每次循环之后还成立的关系称为“循环不变式”或“循环不变关系”

插入排序

利用循环不变式来证明循环的正确性

证明步骤包括以下三步：

① **初始化**：证明初始状态时循环不变式成立，即证明循环不变式在循环开始之前为真

② **保持**：如果循环的某次迭代之前它为真，那么下次迭代之前它仍为真

③ **终止**：证明循环可以在有限次循环之后终止

插入排序

利用循环不变式证明插入排序的正确性

① 初始化：证明循环不变式在循环开始之前为真

第一次循环之前，初始值 $j = 2$，子数组A[1..j-1]实际上只有一个元素，即A[1]，且这个A[1]是A中原来的元素。所以表明第一次循环迭代之前循环不变式成立 —— 初始状态成立

② 保持：证明每次迭代保持循环不变式仍为真

迭代之前，假设A[1..j-1]是已经排好序的序列，待排序的元素A[j]依次与A[j-1]、A[j-2]…进行比较，如果A[j-1]大于A[j]，则依次将其向右移动一位，当遇到开始小于或者等于A[j]的元素时，则A[j]找到了合适的插入位置，插入之后，整个子数组 A[1..j] 已排好序。

之后，执行 j+=1（下次循环开始之前的状态），此时已排序好的子数组“A[1.. j]” 变成了新的A[1.. j-1]。故循环不变式依然成立

插入排序

- **利用循环不变式证明插入排序的正确性**

③ 终止：循环可以有限次终止，当循环结束时，数组 A 中的所有元素都已排好序

循环终止条件是 $j > A.length = n$，所以当循环终止时，有 $j = n+1$

注意：此时循环不变式依然成立，即 A[1.. j-1] 是已排序好的子数组，所以 A 中的 n 个元素已排好序

因此，上述三条性质都成立，根据证明策略，插入排序算法是正确的——确切地说是其中的 for 循环正确，但 for 循环是插入排序过程的主体，所以整个算法正确

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定量的分析，通常是对**计算时间和计算空间**的分析
- 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理论模型，通常采用**随机存取机（RAM）**计算模型。假设：
  - 指令是逐条执行的，没有并发操作
  - 包含常用指令（算术指令、数据移动指令、控制指令），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
- 默认情况下，算法分析一般是指对**算法时间效率**的分析

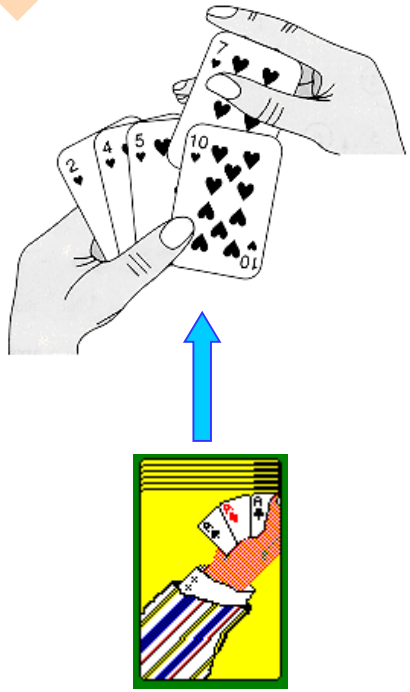
算法分析框架

- **算法运行时间是指在特定输入时，所执行的基本操作数或步数**
  - ✓ 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：**
  - ✓ 算法时间效率采用输入规模 n 为参数的函数 $T(n)$ 来度量
  - ✓ 在输入规模相同情况下，有些算法的时间效率因为输入数据分布的不同会有明显差异。对于这样的算法要区分最坏运行时间、最佳运行时间、以及平均运行时间
  - ✓ 对于大规模输入，通常只关注运行时间函数 $T(n)$ 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

插入排序

□ 算法时间效率分析

INSERTION-SORT(*A*)		cost	times
1	for *j* = 2 to *A.length*	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 and *A*[*i*] > *key*	c_5	$\sum_{j=2}^{n} t_j$
6	{ *A*[*i*+1] = *A*[*i*]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	*i* = *i*-1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
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 $A[1..j\text{-}1]$ 中的每一个元素进行比较，对 $j=2,3,\ldots,n$，有 $t_j=j$，则有：

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

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）
- **函数的增长量级**
  - ① **简化抽象**：忽略每条语句的实际代价，用常量 c_i 来表示；进一步忽略抽象的代价 c_i
  - ② **增长率或增长量级**：利用函数增长率来描述算法效率，并可以用来比较各种算法的相对性能；只考虑公式中最高项，忽略最高项系数和低阶项

插入排序算法分析

- 算法空间复杂度是对一个算法**在运行过程中临时占用存储空间大小**的量度
- 直接插入排序的空间复杂度为 O(1)。直接插入排序是一种原址排序算法，它只需要常数级别的额外空间来存储临时变量，不需要开辟额外的存储空间







第一讲 算法入门

内容提要：

- 课程学习背景
- 算法分析基础 —— 插入排序
- 算法设计策略 —— 归并排序
  - ✓ 分治策略介绍
  - ✓ 归并排序
  - ✓ 分治策略时间分析

分治策略

- **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**：将原问题划分为多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，从而得到原问题的解
- **三个步骤：**

（1）分解（Divide）：将原问题分解成一系列子问题

（2）解决（Conquer）：递归地求解各个子问题；若子问题足够小（比如 n=1），则可以直接求解

（3）合并（Combine）：将子问题的结果合并成原问题的解

归并排序

口 归并排序算法（Merge Sort Algorithm）

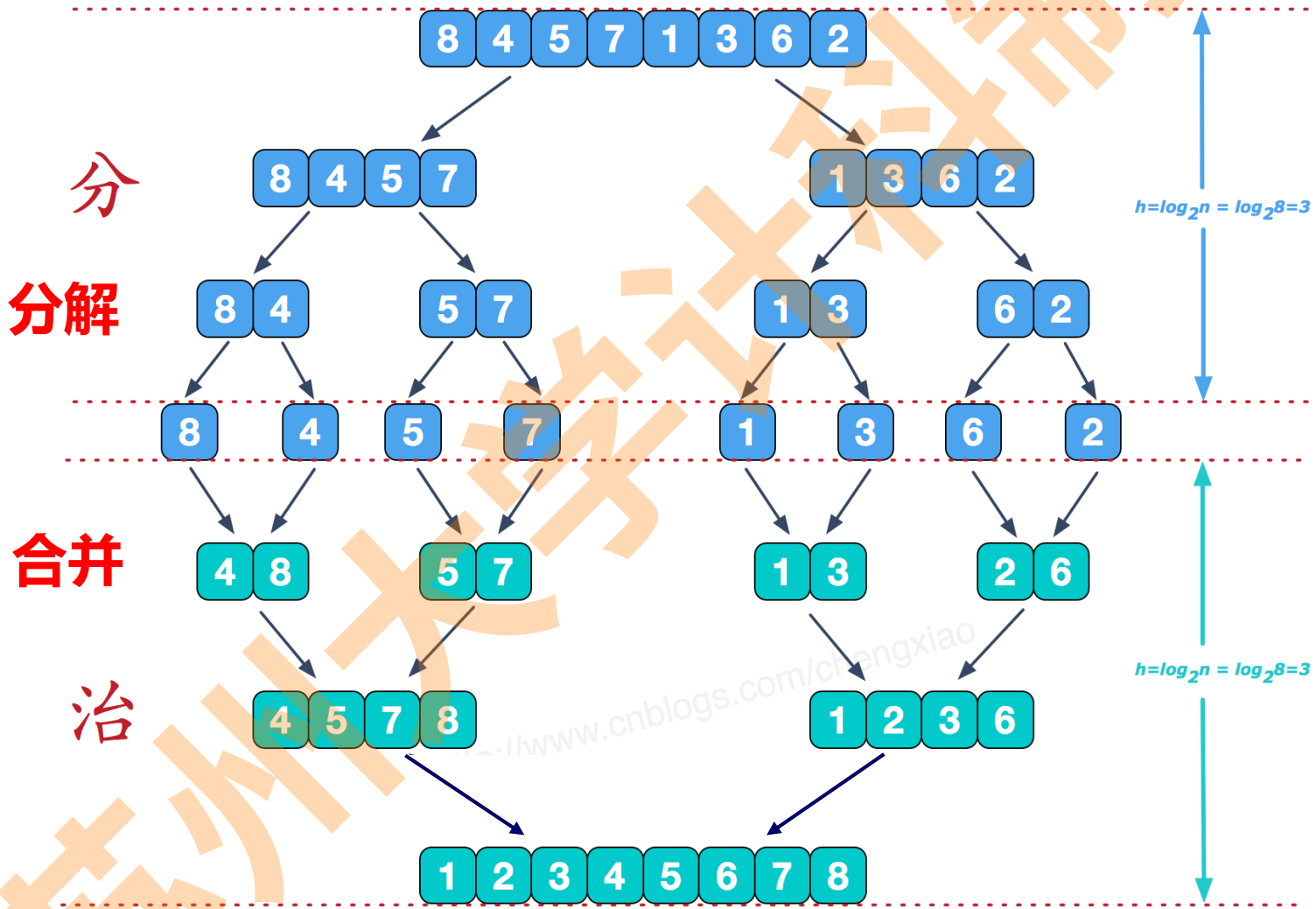
① 分解：把 n 个元素分成各含 $n/2$ 个元素的子序列（左边可能比右边多 1 个数）

② 解决：将步骤 ① 分成的两部分，再分别进行递归分解；直到所有部分的元素个数都为 1

③ 合并：从最底层开始逐步合并两个已排序好的子序列

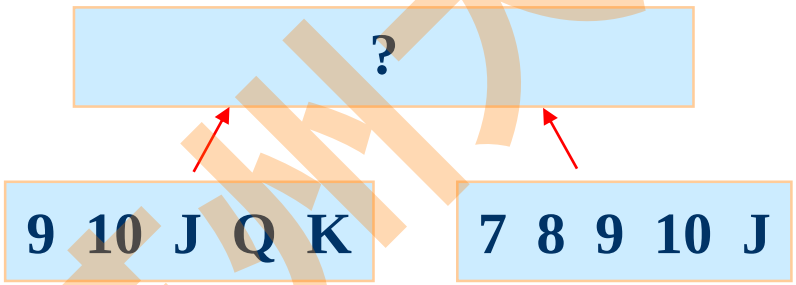
N：在对子序列排序时，其长度为 1 时递归结束，即单个元素被视为已排序好的

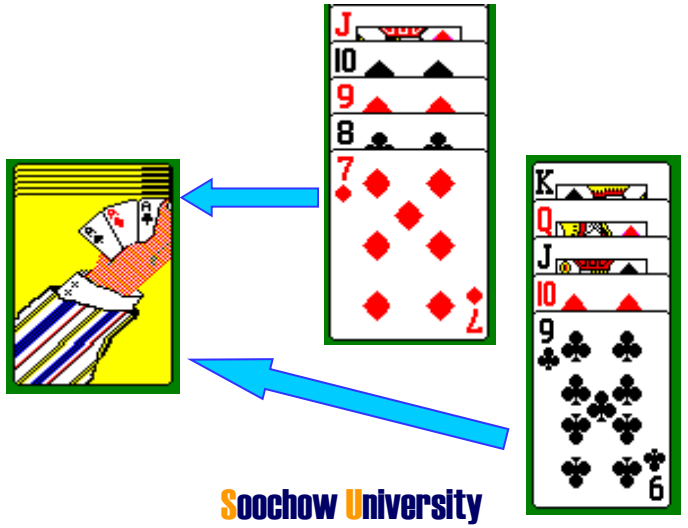
归并排序



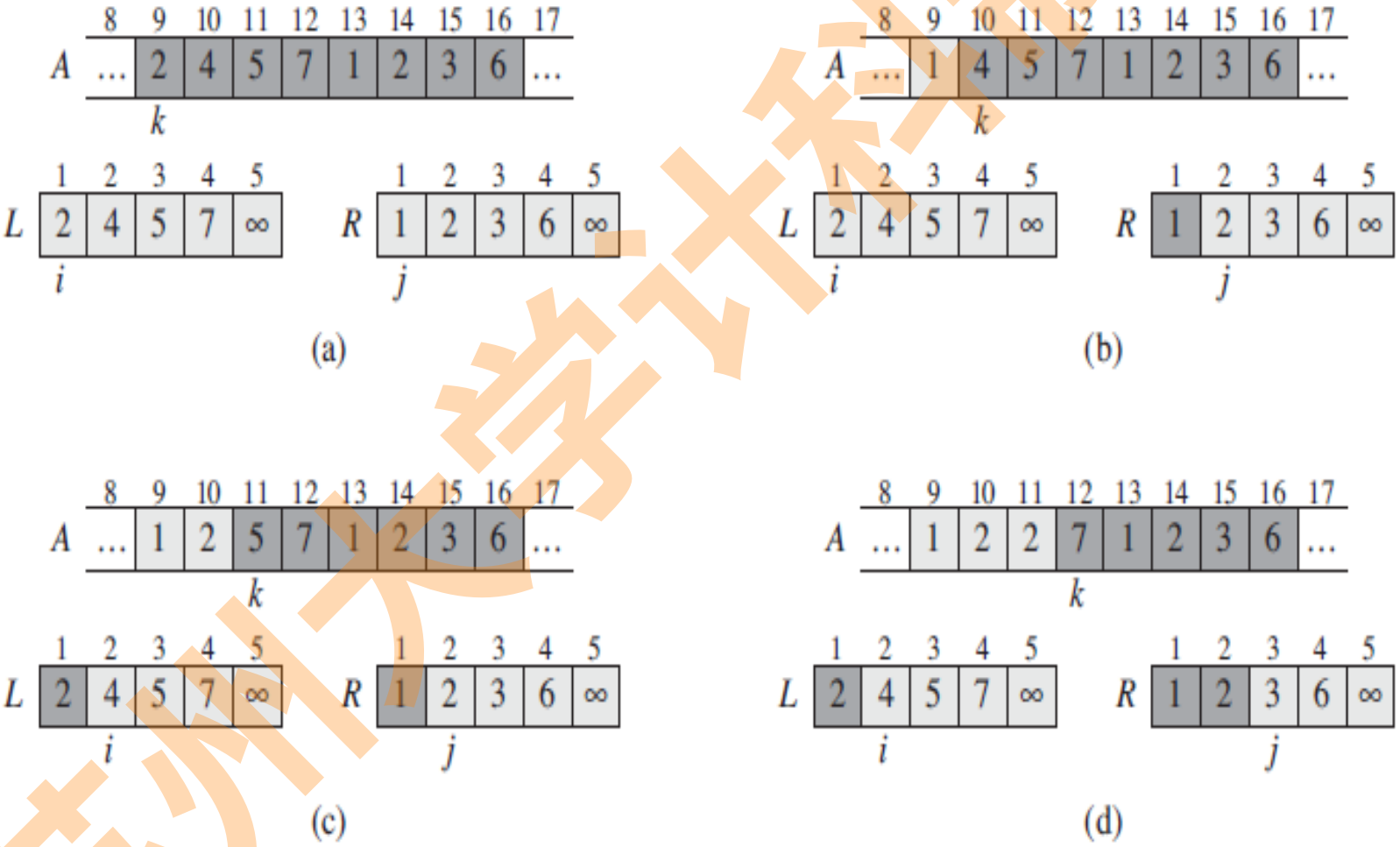
归并排序

- **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 是数组中元素的下标，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来代替当前子数组 $A[p..r]$
  - **合并过程**：

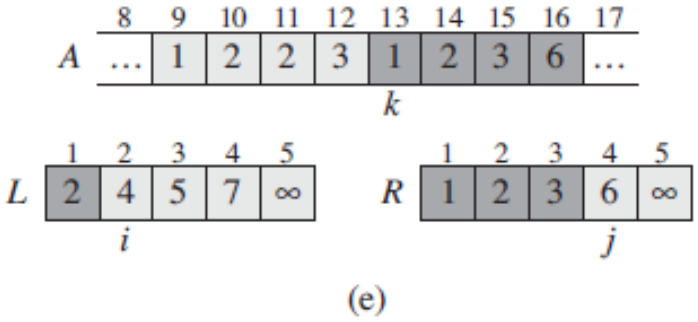




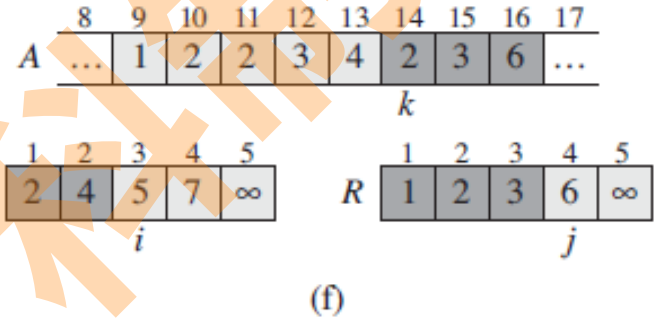
归并排序


8 9 10 11 12 13 14 15 16 17
A … 2 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(a)
8 9 10 11 12 13 14 15 16 17
A … 1 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(b)
8 9 10 11 12 13 14 15 16 17
A … 1 2 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(c)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(d)

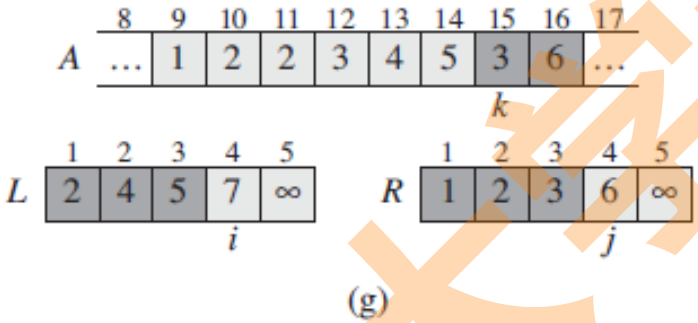
归并排序



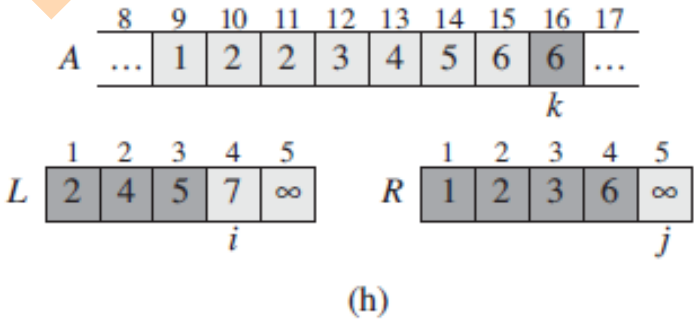
(e)



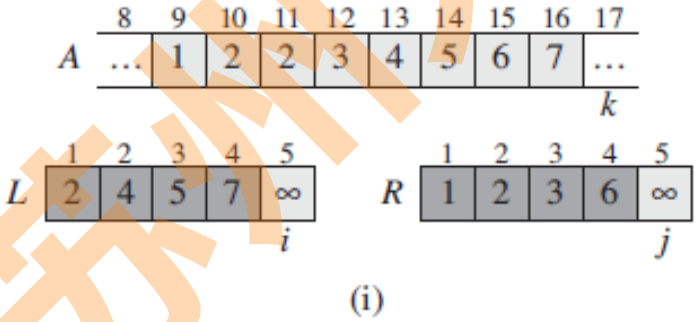
(f)



(g)



(h)



(i)

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        L[i] = A[p+i-1]
6    for j = 1 to n2
7        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       if L[i] ≤ R[j]
14           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)	cost	times
1 $n_1 = q-p+1$	c	1
2 $n_2 = r-q$	c	1
3 create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$	c	1
4 for $i = 1$ to n_1	c	n_1+1
5 do $L[i] = A[p+i-1]$	c	n_1
6 for $j = 1$ to n_2	c	n_2+1
7 do $R[j] = A[q+j]$	c	n_2
8 $L[n_1+1] = \infty$	c	1
9 $R[n_2+1] = \infty$	c	1
10 $i = 1$	c	1
11 $j = 1$	c	1
12 for $k = p$ to r	c	$r-p+2$
13 if $L[i] \le R[j]$	c	$r-p+1$
14 $A[k] = L[i]$	c	n_1
15 $i = i+1$	c	n_1
16 else $A[k] = R[j]$	c	$r-p+1-n_1$
17 $j = j+1$	c	$r-p+1-n_1$

$r-p+1$

$= n_1 + n_2 = n$

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

归并排序







- 将 MERGE 合并过程作为归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，分解步骤首先计算出一个下标 $q=\lfloor (p+r)/2 \rfloor$，将 $A[p..r]$ 分成两个子数组 $A[p..q]$ 和 $A[q+1..r]$

```
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)
```

分治算法分析







- 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求解递归式，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

归并排序算法分析

- 为简化算法分析，假设 n 为 2 的幂次，使得每次分解产生的子序列长度恰为 $n/2$。这一假设不影响递归式解的增长率。
- 归并排序 n 个数的最坏运行时间：
  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 $C(n)=\Theta(n)$

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

归并排序算法分析

- 归并排序算法的空间复杂度：使用一个与原序列同样大小的辅助数组 n + 递归调用时压入栈的数据占用的空间 $\lg n$，即 $n+\lg n$；所以空间复杂度为：$O(n)$
- 递归调用的空间复杂度 = 每次递归的空间复杂度 * 递归深度
- 归并排序算法中，每次递归的空间复杂度是 $O(1)$（为局部变量和形参所开辟的空间），递归调用栈深度为 $\lg n$，空间复杂度就是 $\lg n * 1 = O(\lg n)$







谢谢！