

INTRODUCTION TO

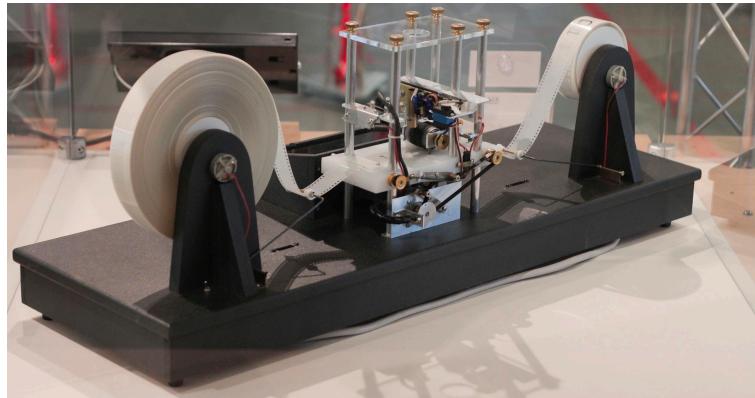
# ALGORITHMS

THIRD EDITION

# 算法 (Algorithm) 是什么？

算法

输入 →



→ 输出

- 良定义的计算过程
- 一系列的计算步骤

算法正确  $\Leftrightarrow$  任意输入实例，算法终止 + 结果正确

# 算法基础

《算法导论》—— 第 1 讲

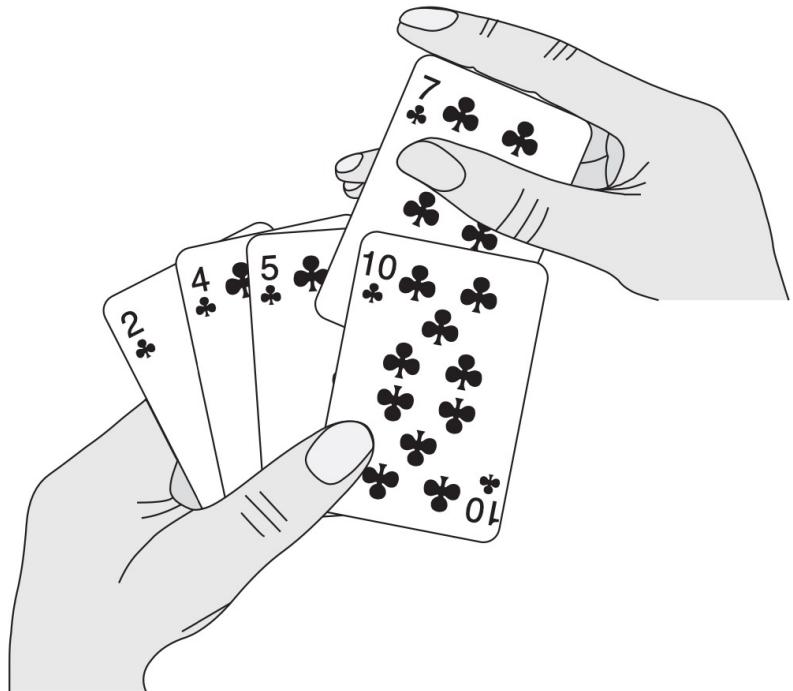
*jiacaicui@163.com*

# 内容提要

- 学习描述和分析算法的框架
  - 使用伪代码来描述算法
  - 使用循环不变式来分析算法的正确性
  - 使用渐进记号（初步）来表示对于算法运行时间的分析
- 两个排序算法：“插入排序” 和 “归并排序”
  - “插入排序”：增量法
  - “归并排序”：分治法
    - 分析分治算法

# 插入排序

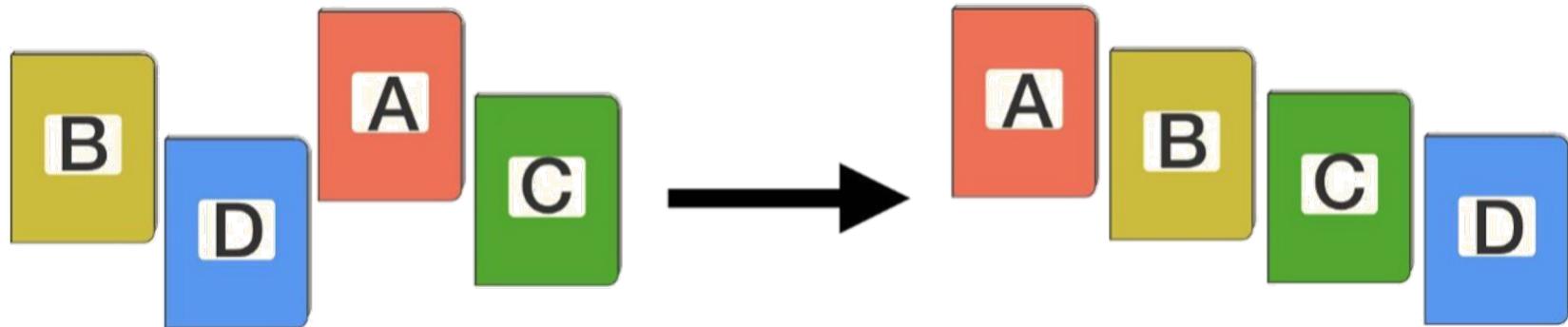
打扑克时的排序算法



# 排序问题

输入： $n$  个数的一个序列  $\langle a_1, a_2, \dots, a_n \rangle$ 。

输出：输入序列的一个排列  $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，满足  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ 。



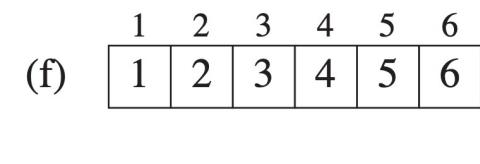
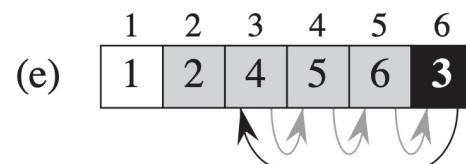
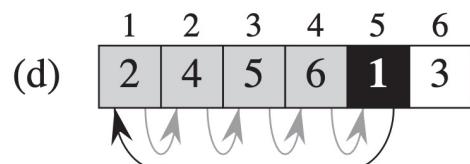
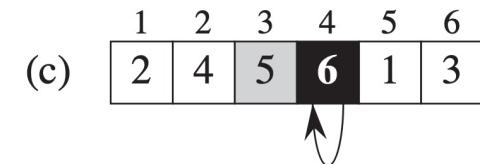
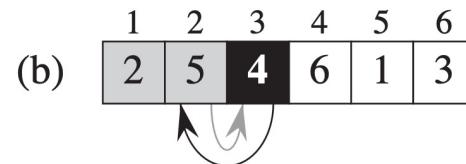
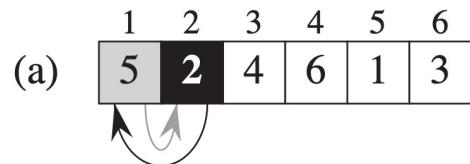
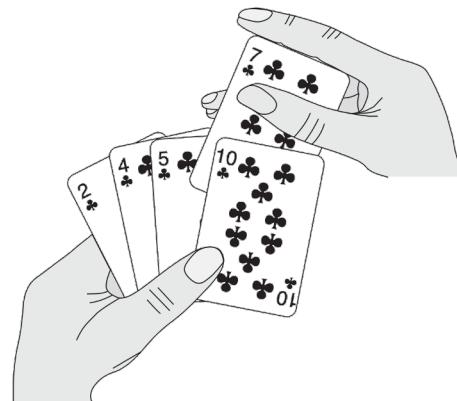
- 这些数字也称为**关键字**，可能还有一些额外的数据和每个关键字绑定在一起，称为**卫星数据**。

# 伪代码 (Pseudocode)

- 伪代码和C/C++, Python, Java这些编程语言很像，转化起来很容易。
- 伪代码是用来面向人类描述算法的。
  - 一些诸如数据抽象、模块化、异常处理等软件工程问题是被忽略的。
- 我们有时候会在伪代码中嵌入英语描述。
  - 因此，伪代码通常是不可机器编译的。
- 伪代码书写约定：
  - 英文版教材20-22页，中文版教材11-12页。

# 插入排序

- 一个对少量元素进行排序的不错的算法。
- 你将乱序的牌堆整理成有序的手牌用的大抵也是这个算法。

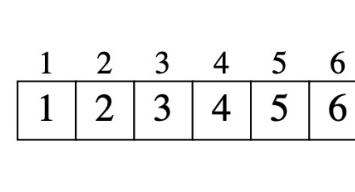
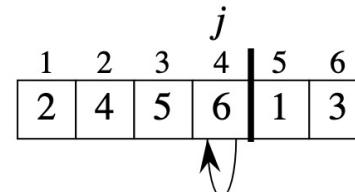
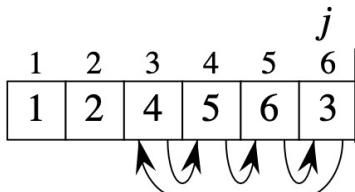
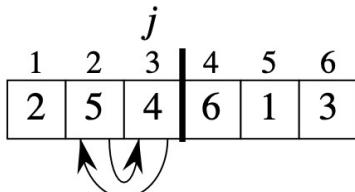
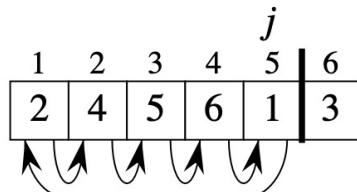
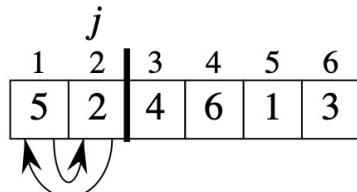
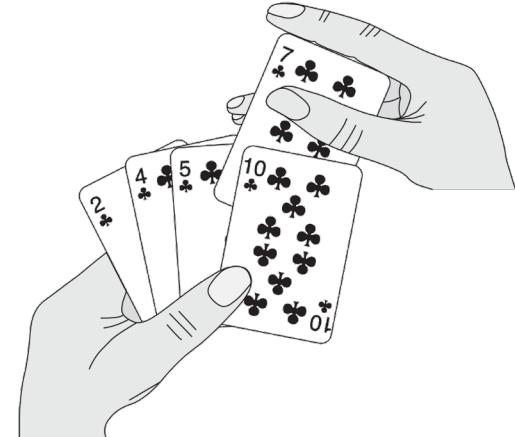


# 插入排序伪代码

INSERTION-SORT( $A$ )

```

1  for  $j = 2$  to  $A.length$ 
2       $key = A[j]$ 
3      // Insert  $A[j]$  into the sorted sequence  $A[1..j - 1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```



# 正确性

- 我们常常使用 “**循环不变式**” (Loop Invariant) 来证明**增量法**的正确性。
- 关于循环不变式，我们必须证明三条性质：
  - **初始化** (Initialization) : 循环的第一次迭代之前，它为真；
  - **保持** (Maintenance) : 如果循环的某次迭代之前它为真，那么下一次迭代之前它仍为真。
  - **终止** (Termination) : 在循环终止时，不变式为我们提供一个有用的性质，该性质有助于证明算法是正确的。
- 本质：数学归纳法 (Mathematics Induction)



# 证明插入排序的正确性

- **循环不变式：**

在 **for** 循环每次迭代开始之前，子数组  $A[1..j - 1]$  由原来在  $A[1..j - 1]$  中的元素组成，且已经排好序。

- **证明：**

**初始化**：第一次迭代开始前  $j = 2$ ,  $A[1..j - 1] = A[1]$  天然有序且是原来元素。

**保持**：若迭代开始前， $A[1..j - 1]$  有序且由原本元素组成；迭代过程中将  $A[j - 1], A[j - 2], A[j - 3]$  等右移，直到**空出合适的位置**放  $A[j]$ ；所以下次迭代开始前， $A[1..j] = A[1..j + 1 - 1]$  有序且由原本元素组成。

**终止**：循环终止的时候  $j = n + 1$ ，循环不变式告诉我们  $A[1..n + 1 - 1] = A[1..n]$  有序且由原本元素组成。

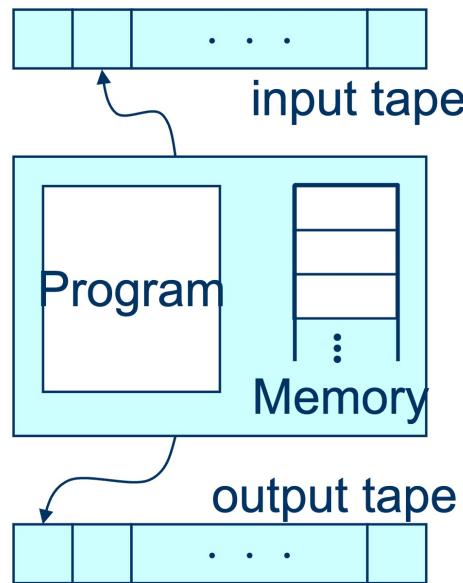
# 分析算法

计算模型与运行时间



# 随机访问机 (RAM) 模型

- 分析算法的运行时表现需要一个具体的计算模型。



- 假定一种通用的单处理器计算模型：Random Access Machine。
- 支持常数项时间的算术指令、数据移动指令和控制指令。

# 运行时间

- 通常把一个算法的运行时间描述成其输入规模的函数。
- 输入规模的粒度是视具体问题而定的。
  - 排序问题：排序元素的个数。
  - 整数乘法问题：乘数的位数。
  - 图问题：图上的结点数和边数。
- 给定输入后，运行时间就是该输入下基本操作的次数。
  - 通常我们会选择某个基本操作（一般作最坏考虑，选择执行次数最多的）作为关键操作。
  - 比如说，基于比较的排序算法将比较操作是为关键操作。

# 分析插入排序

<code>INSERTION-SORT(<math>A</math>)</code>	$cost$	$times$
1 <b>for</b> $j = 2$ <b>to</b> $A.length$	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3       // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ 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 $A[i + 1] = key$	$c_8$	$n - 1$

- $t_j$  = 第5行 **while** 循环条件判断的次数。

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

# 分析插入排序

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

- **最好情况**：输入是顺序的，则  $t_j = 1, j = 2, 3, \dots, n$ 。

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

- **最坏情况**：输入是逆序的，则  $t_j = j, j = 2, 3, \dots, n$ 。

$$\begin{aligned}
 T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5 \left( \frac{n(n+1)}{2} - 1 \right) \\
 &\quad + c_6 \left( \frac{n(n-1)}{2} \right) + c_7 \left( \frac{n(n-1)}{2} \right) + c_8(n-1) \\
 &= \left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\
 &\quad - (c_2 + c_4 + c_5 + c_8).
 \end{aligned}$$

# 最坏情况与平均情况分析

- 通常，我们只关心**最坏情况下**算法的运行时间：
  - 最坏情况下的运行时间给出了任何输入下运行时间的一个上界。
  - 对某些算法，最坏情况经常出现。
  - “平均情况”往往和最坏情况“一样差”。

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

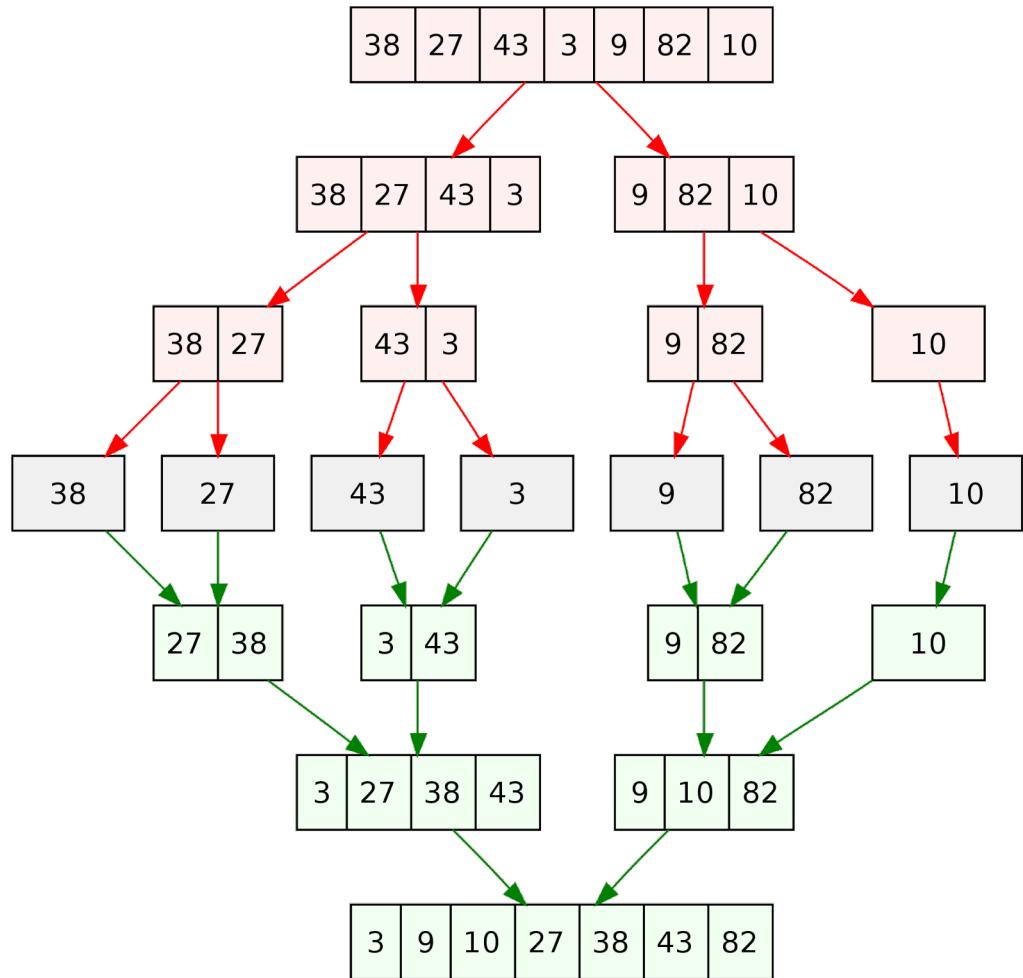
- 平均情况：总是往中间插，则  $t_j \approx \frac{j}{2}, j = 2, 3, \dots, n$ 。
  - $T(n)$  依旧是一个二次函数。

# 增长量级

- 更简化的抽象——**增长量级/增长率** (Order-of-Growth)：
  - 忽略低阶项；
  - 忽略高阶项系数。
- 当  $n$  足够大的时候，低阶项和系数相对来说不如增长率重要。
- 记号： $T(n) = an^2 + bn + c = \Theta(n^2)$ 
  - 下一讲会给出形式化的数学定义，本讲暂时直观的理解含义即可。

# 设计算法

## 分而治之的智慧



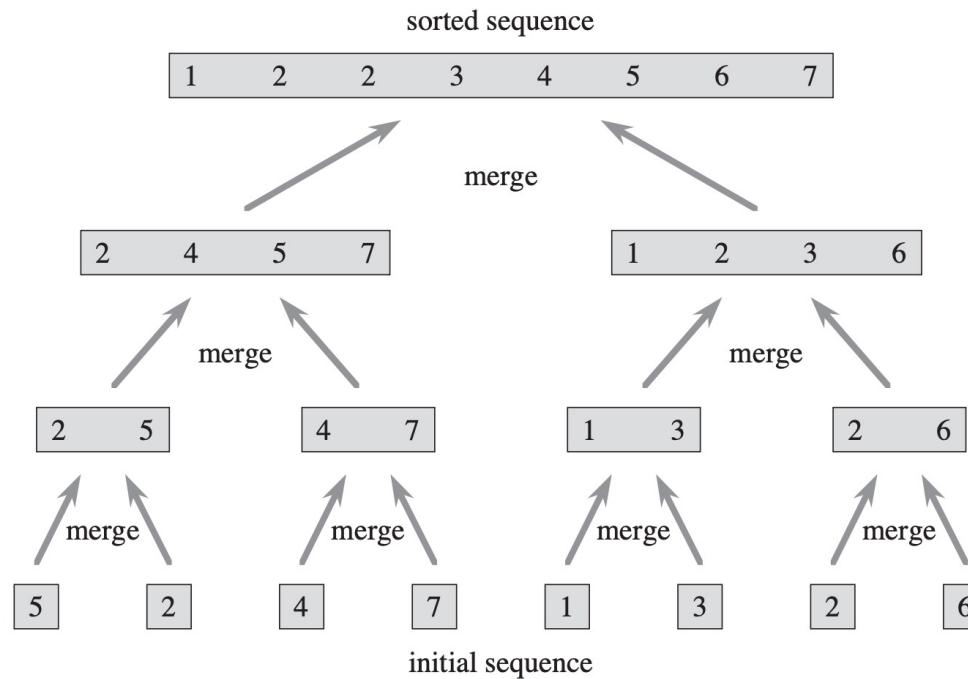
# 增量法与分治法 (Divide and Conquer)

- 插入排序采用的是**增量法**思路：
  - 考虑每多一个输入元素，需要在原来的基础上再做些什么。
  - 手上有已经排好序的  $A[1..j - 1]$ ，适当地放置  $A[j]$ ，得到排好序的  $A[1..j]$ 。
- 另一种方式——“**分而治之**”，**分治法**的思路如下：
  - 分 (Divide)：原问题分解为子问题，**子问题是原问题规模较小的实例**；
  - 治 (Conquer)：解决子问题——**递归地解决或者直接求解（基本情况）**；
  - 合 (Combine)：将子问题的解合并成原问题的解。

# 归并排序

- 归并排序完全遵循分治模式：

- 分：分解待排序的  $n$  个元素的序列成各具  $\frac{n}{2}$  个元素的子序列。
- 治：使用归并排序递归地排列两个子序列。
- 合：合并两个已排序的子序列以产生已排序的答案。



# 归并排序伪代码

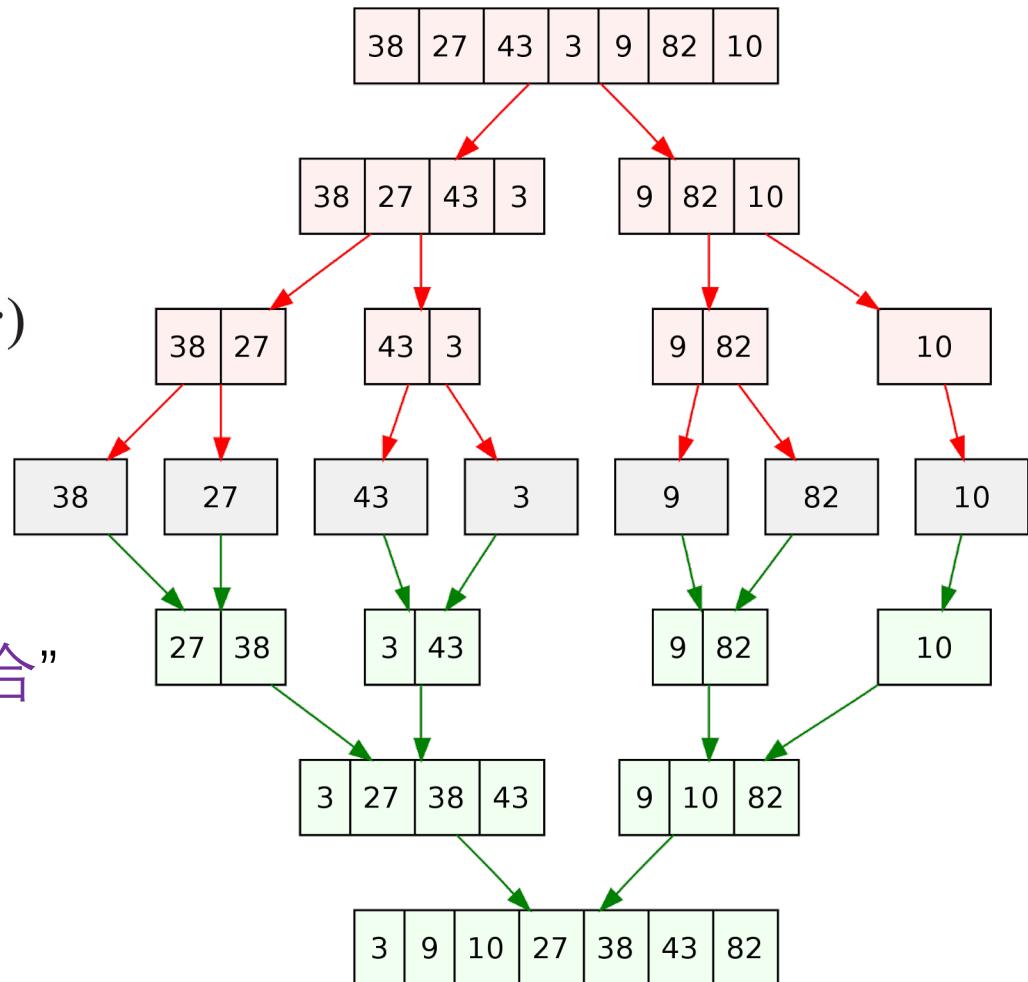
MERGE-SORT( $A, p, r$ )

```

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

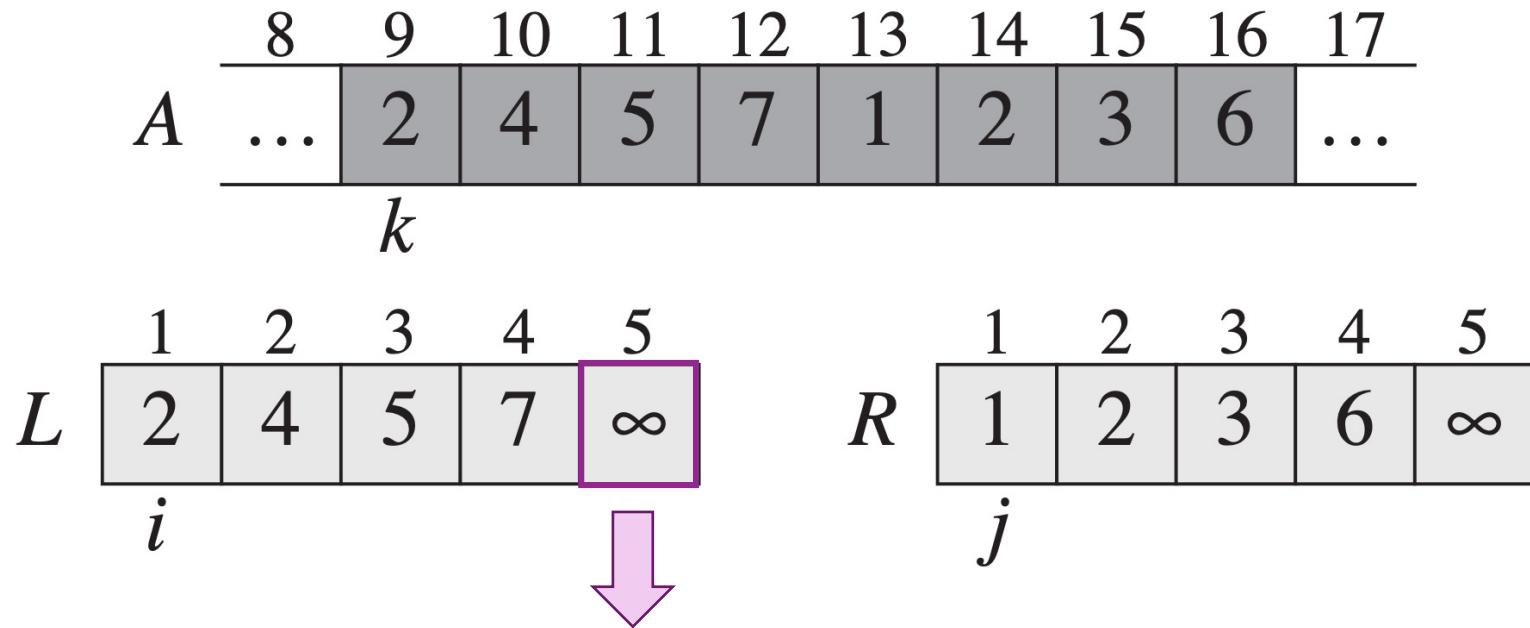
```

- 归并排序是一个“易分难合”的分治算法。
- 怎么“合”？



# 合并算法思路

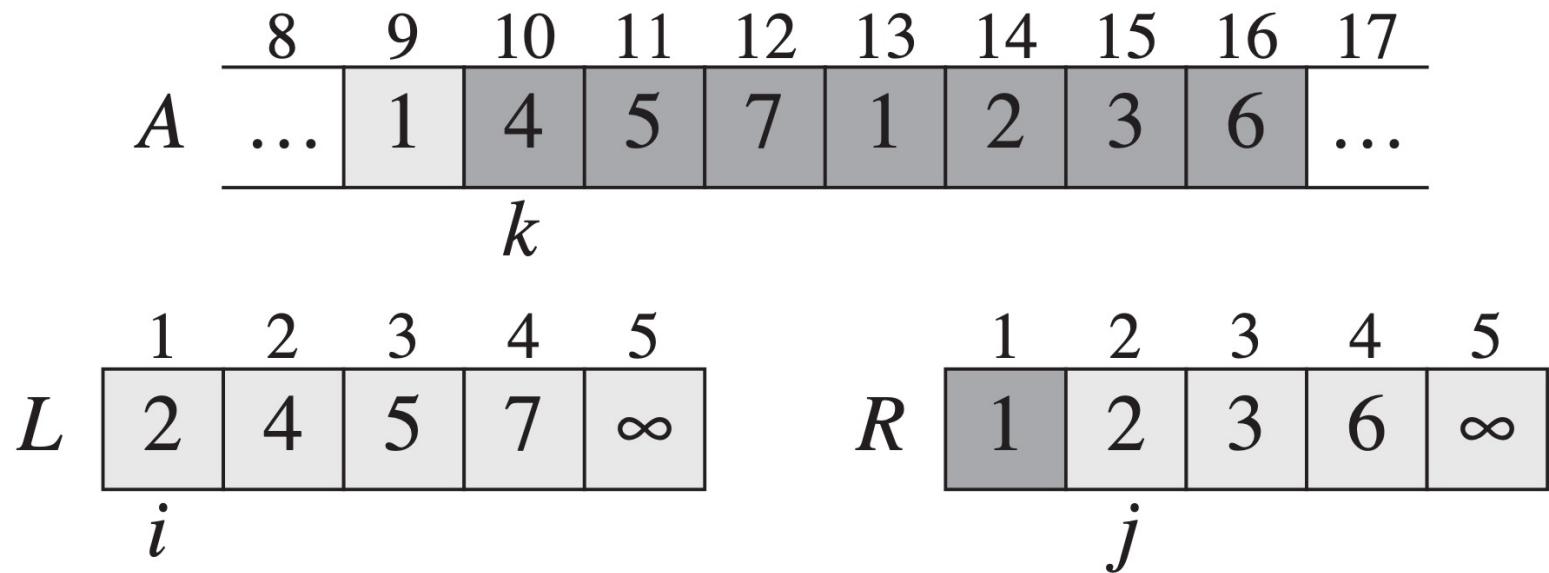
- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



“哨兵”牌，简化代码，避免每次都需要检查牌堆是否为空

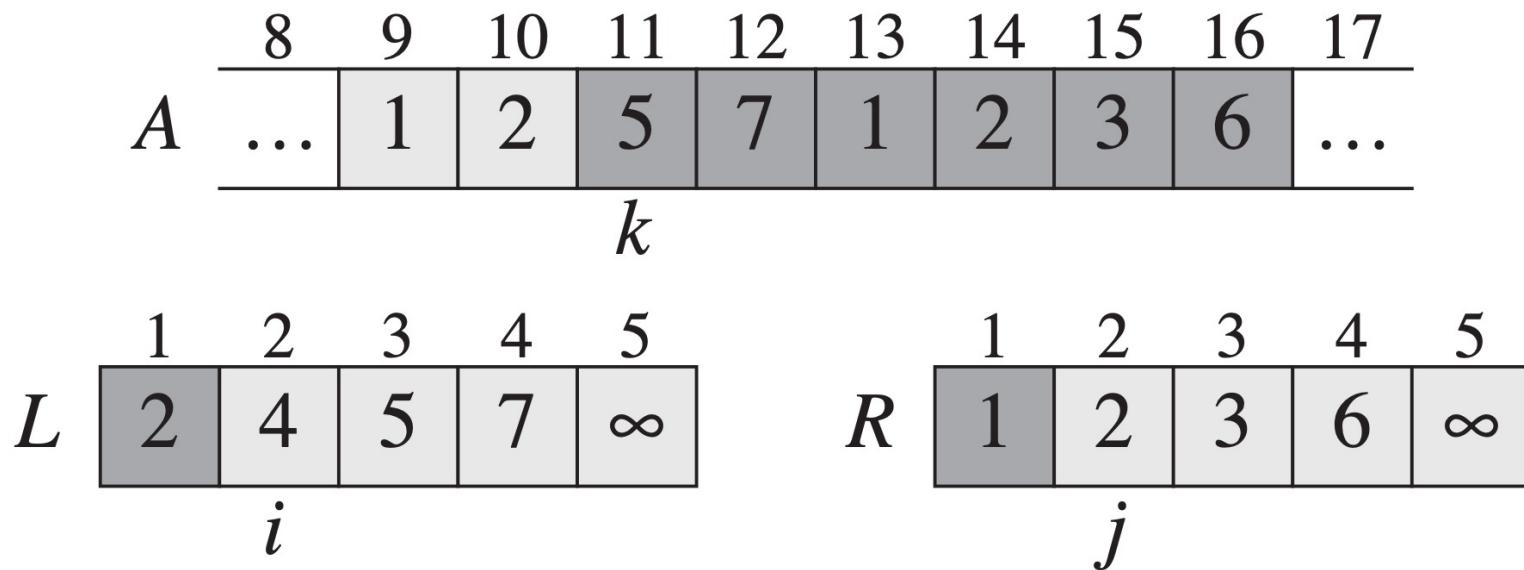
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



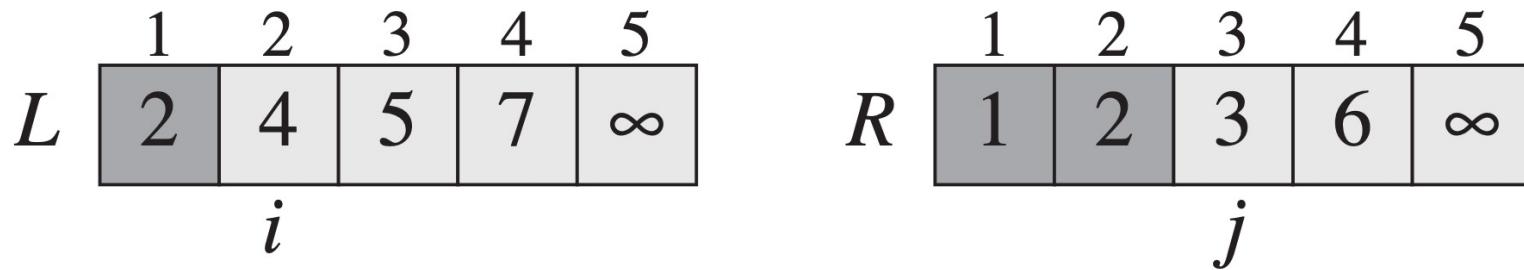
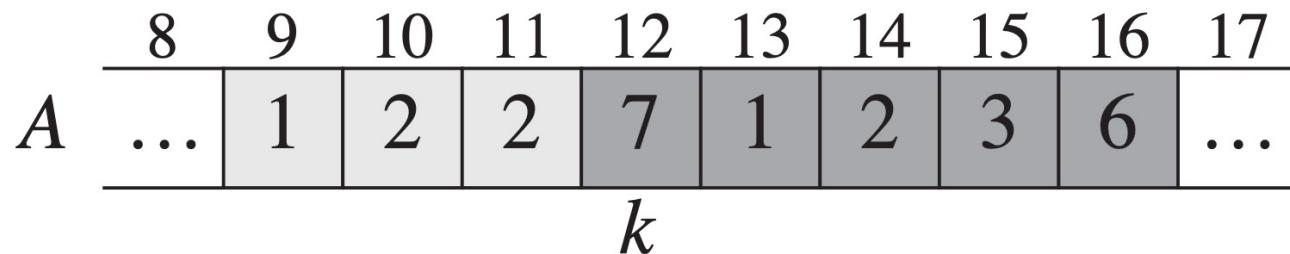
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
  - 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



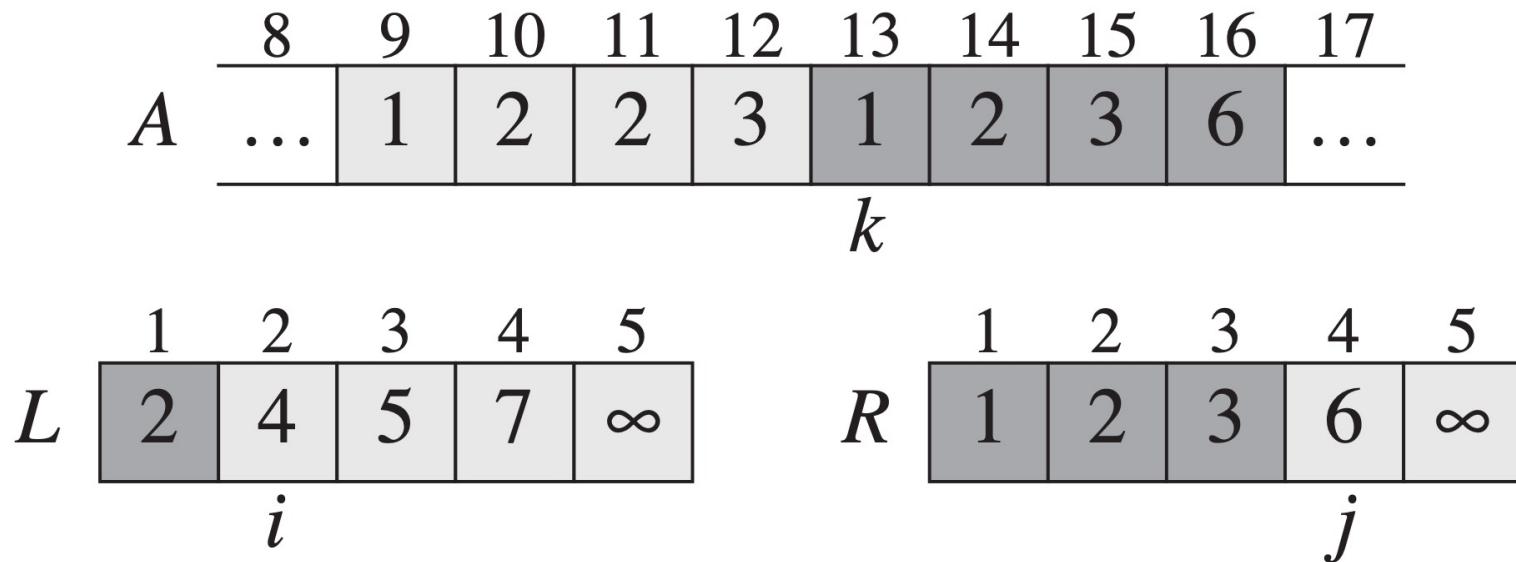
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



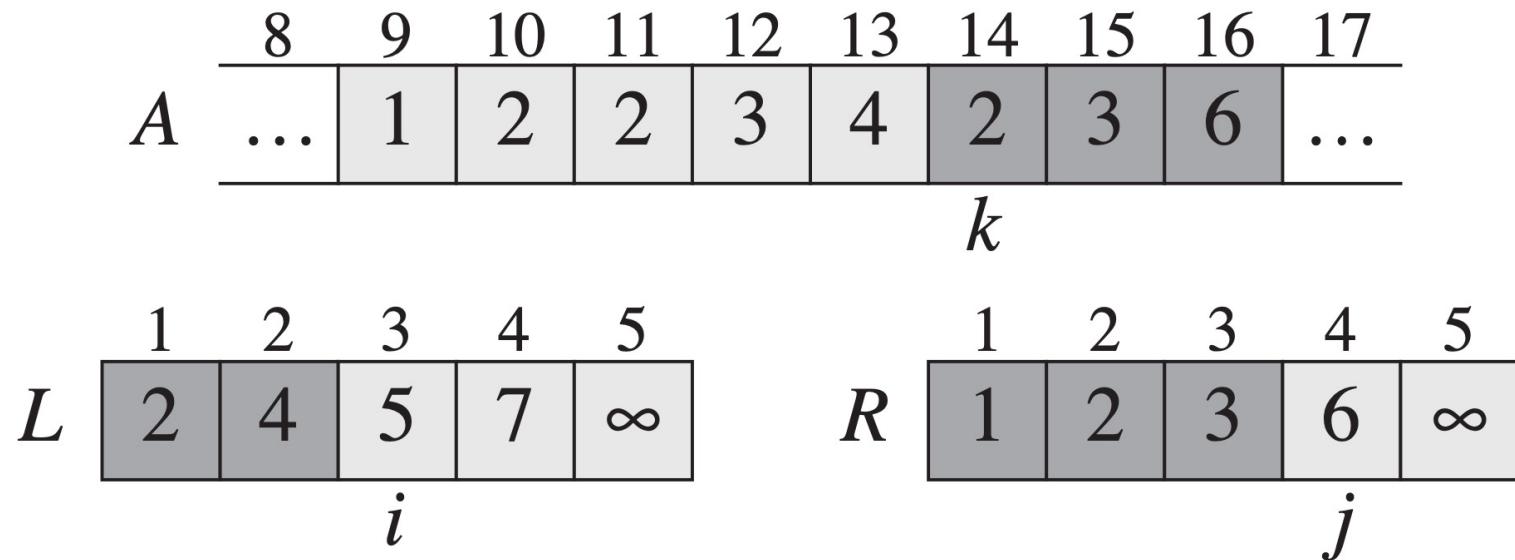
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



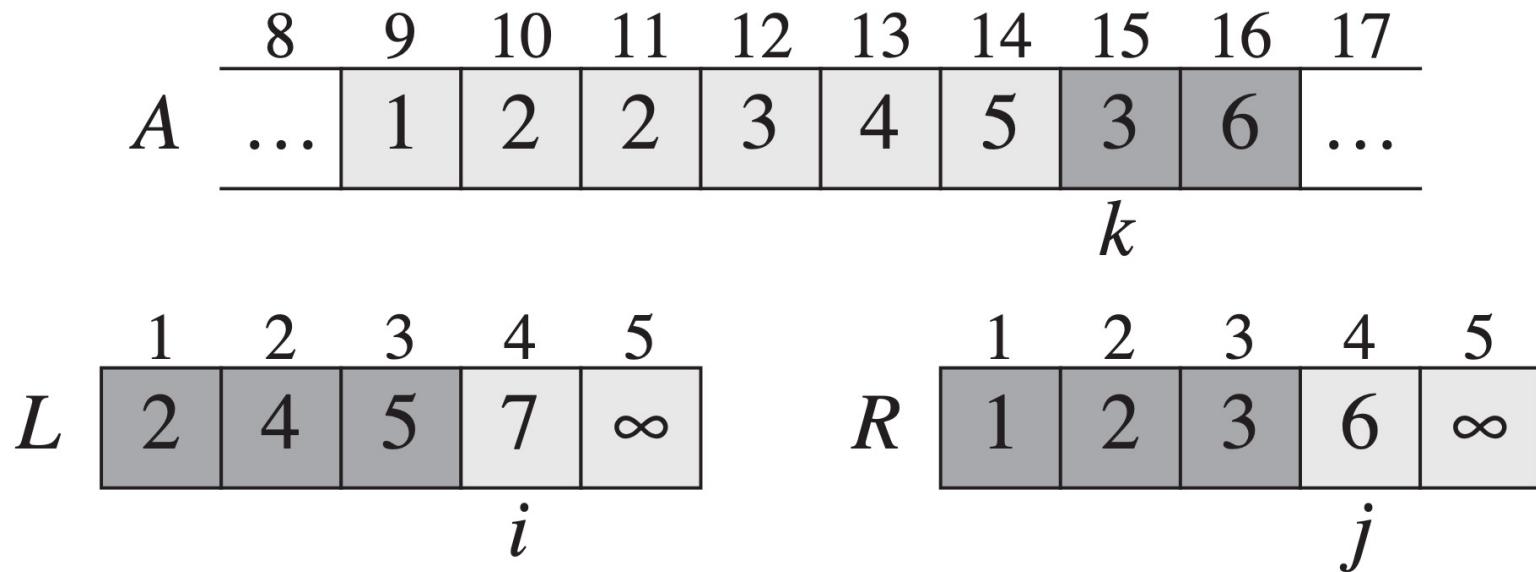
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



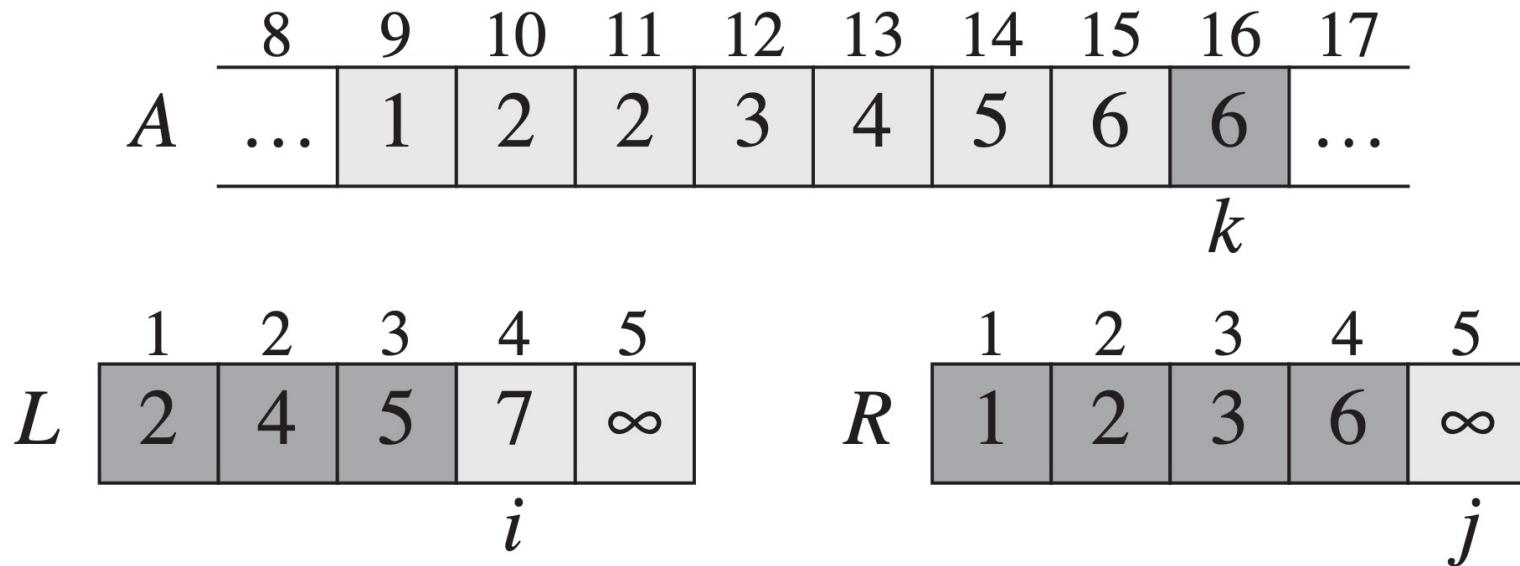
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



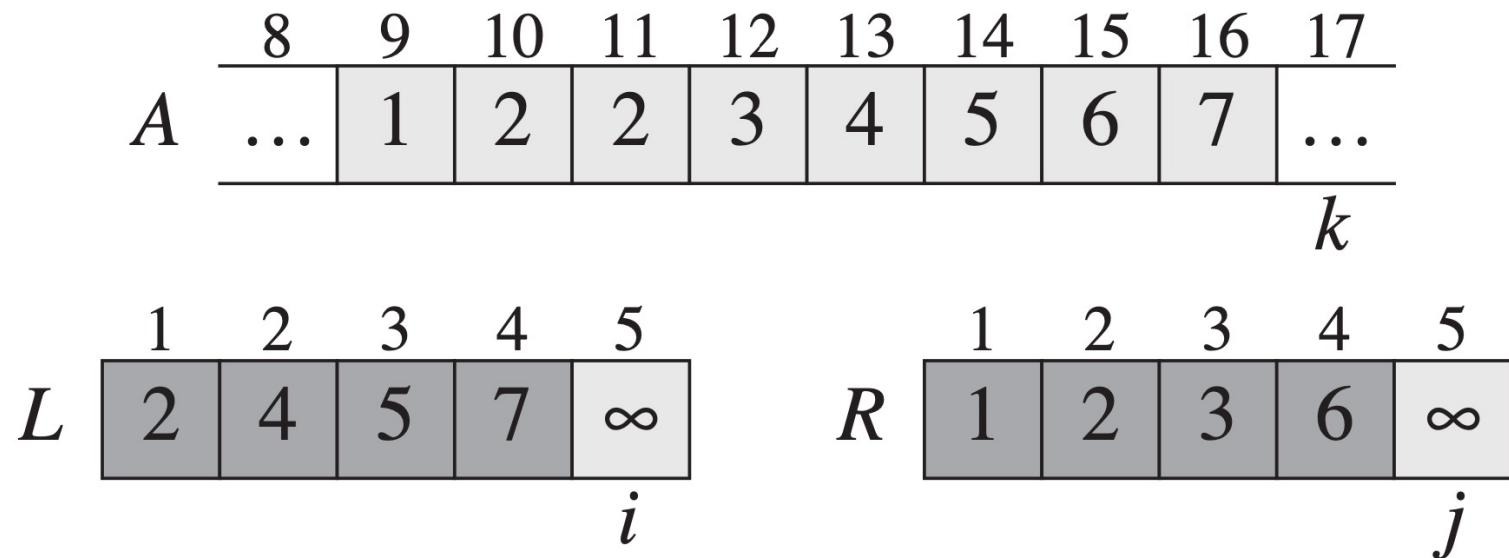
# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



# 合并算法思路

- 如果给你两个已经排好序的牌堆，你会怎么把它们变成排好序的手牌呢？
- 不断地从两个牌堆顶部取较小的牌插在手上已有牌的最后即可。



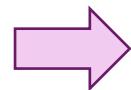
# 合并算法伪代码

MERGE( $A, p, q, r$ )

```

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

“哨兵” 牌，简化代码，避免每次都需要检查牌堆是否为空



# 归并排序的正确性

- 如果合并算法正确，**数学归纳法**显然有归并排序正确。
- 下面只需证明合并算法的正确性即可，利用**循环不变式**：

第 12 ~ 17 行的 **for** 循环的每次迭代时，子数组  $A[p..k - 1]$  按从小到大的顺序包含  $L[1..n_1 + 1]$  和  $R[1..n_2 + 1]$  中的  $k - p$  个最小元素。

- 证明：

**初始化**：循环第一次迭代前， $k = p$ ，不变式显然成立。

**保持**：若迭代开始前结论成立，不妨设  $L[i] \leq R[j]$ ；这时， $L[i]$  是未被复制回数组 A 的最小元素，所以在第 14 行将  $L[i]$  复制到了  $A[k]$  并自增  $i$  和  $k$ ；下一次迭代开始前，不变式依旧成立。

**终止**：终止时  $k = r + 1$ ，循环不变式说明  $A[p, r]$  已完成合并。

# 分析分治算法

- 我们可以用递归式 (recurrence) 来描述递归算法的运行时间。
  - 假设  $T(n)$  是某个算法在问题规模  $n$  下的运行时间。
  - 假设  $n \leq c$  时可以  $\Theta(1)$  直接解决；否则需要以  $D(n)$  的代价将原问题划分成  $a$  个规模为  $\frac{n}{b}$  的子问题递归地解决，然后以  $C(n)$  的代价将子问题的解合成原问题的解。
  - 则可得到递归式：
$$T(n) = \begin{cases} \Theta(1), & n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n), & \text{otherwise} \end{cases}$$
- 我们会在第3讲学习如何求解这类常见的递归式。
  - 不过这周我们可以先解一下归并排序的递归式。

# 归并排序的运行时间

- 运行时间分析：
  - 基础情况： $n = 1$  时自然有序，无需排序，时间为  $\Theta(1)$ ；
  - 分：计算子数组的中间位置，需要常数项时间， $D(n) = \Theta(1)$ ；
  - 治：递归地求解 2 个规模为  $\frac{n}{2}$  的子问题，需要  $2T(\frac{n}{2})$  的时间；
  - 合：在具有  $n$  个元素的数组上运行合并算法需要  $C(n) = \Theta(n)$  的时间。
- 递归式：
$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n > 1 \end{cases}$$
- 根据第3讲会学习的主定理可以很容易得到  $T(n) = \Theta(n \log n)$ 。
  - 不过我们现在也可以算一下。

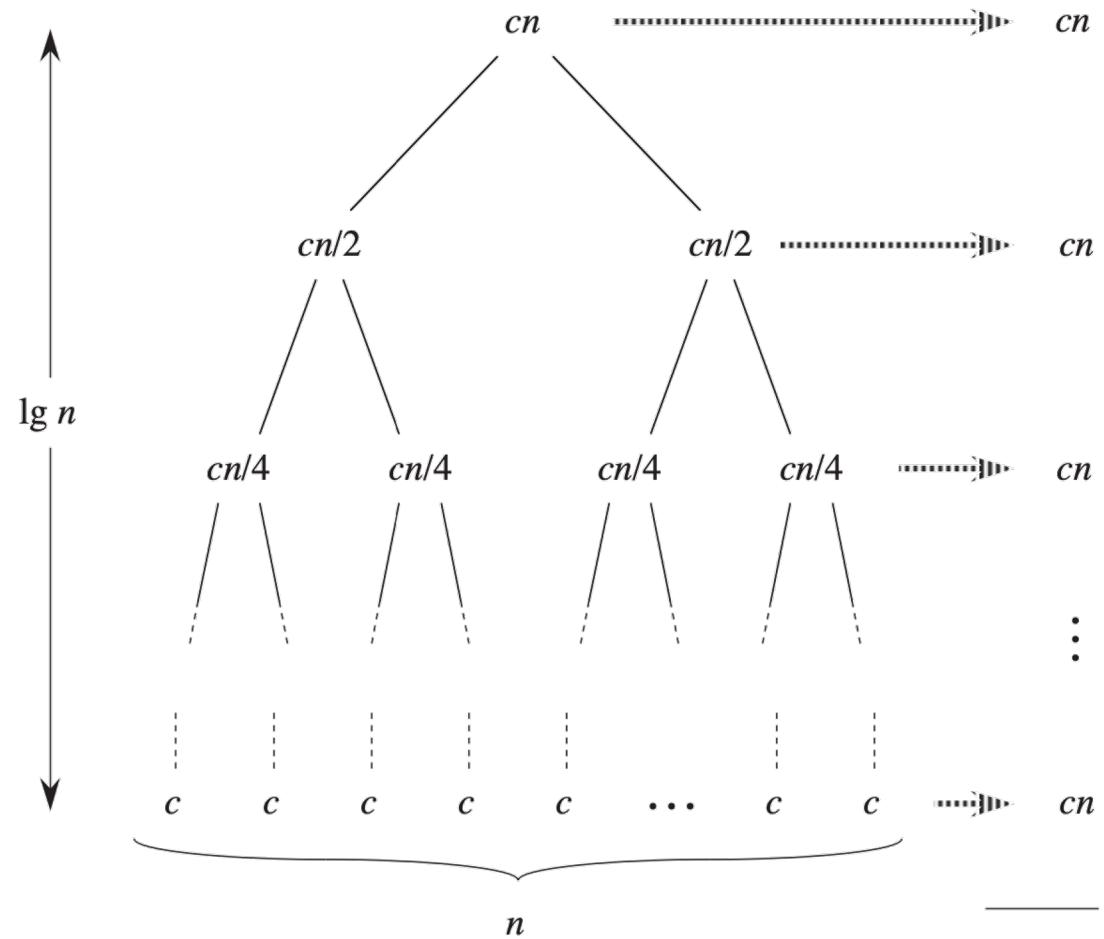
# 归并排序递归式求解

- 为了方便起见，假设  $n$  是 2 的幂，将递归式改写为：

$$T(n) = \begin{cases} c, & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn, & n > 1 \end{cases}$$

- 直接展开递归式并求和可得：

$$\begin{aligned} T(n) &= (\log n + 1) \cdot cn \\ &= \Theta(n \log n) \end{aligned}$$



# 本讲小节

# 内容提要

- 学习描述和分析算法的框架
  - 使用伪代码来描述算法
  - 使用循环不变式来分析算法的正确性
  - 使用渐进记号（初步）来表示对于算法运行时间的分析
- 两个排序算法：“插入排序” 和 “归并排序”
  - “插入排序”：增量法
  - “归并排序”：分治法
    - 分析分治算法

The End!

