计算机算法设计与分析

田暄

西安交通大学软件学院

2020.2

课程简介

- 参考书目：
  - 《计算机算法设计与分析》（第四版）王晓东编著 电子工业出版 社 2012年4月
  - 《算法导论》 第三版 Thomas H.Cormen 等著殷建平等译 机械工业出版社 2013年1月
- 成绩计算：
  - 平时成绩：30%（考勤+作业+上机编程报告）
  - 期末成绩：70%

第一章 算法概述

- 本章要点
- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述
- 熟悉使用伪代码与c++语言描述算法
- 熟悉算法设计实验环境

算法(Algorithm)

- 算法是指解决问题的一种方法或一个过程。
- 算法是若干指令的有穷序列，满足性质：
- (1)**输入**：有外部提供的量作为算法的输入。
- (2)**输出**：算法产生至少一个量作为输出。
- (3)**确定性**：组成算法的每条指令是清晰，无歧义的。
- (4)**有限性**：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序(Program)

- 程序是算法用某种程序设计语言的具体实现。

- 程序可以不满足算法的性质(4)。
- 例如操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
- 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

算法分析基本概念

- 算法运行所需要的计算机资源的量称为**算法的复杂性**。
  - 算法的时间复杂性$T(n)$；
  - 算法的空间复杂性$S(n)$。
  - 其中n是问题的规模（输入大小）。
- 计算算法运行所需资源量的过程称为算法复杂性分析，简称为**算法分析**。

算法的时间复杂性

- （1）**最坏情况**下的时间复杂性

- $T_{\max}(n) = \max\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$
- （2）**最好情况**下的时间复杂性
- $T_{\min}(n) = \min\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$
- （3）**平均情况**下的时间复杂性
- $T_{\mathrm{avg}}(n) = \sum_{size(I)=n} p(I)T(I)$
- 其中I是问题的规模为n的实例，$p(\mathrm{I})$是实 例I出现的概率。

算法渐近复杂性

- $T(n) \to \infty$，as $n \to \infty$;

- $(T(n) - t(n))/ T(n) \to 0$，as $n \to \infty$;

- $t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。

- 在数学上，$t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项。它比$T(n)$ 简单。

渐近分析的记号

- 在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。
- （1）**渐近上界记号O**
- $O(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq f(n) \leq cg(n) \}$
- （2）**渐近下界记号Ω**
- $\Omega(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq cg(n) \leq f(n) \}$

- （3）**非紧上界记号 o**
- $o(g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，存在正数和$n_0>0$使得对所有$n\geq n_0$有：$0 \leq f(n)<cg(n)$ $\}$
- 等价于 $f(n) / g(n) \to 0$ ，as $n\to\infty$。
- （4）**非紧下界记号 ω**
- $\omega(g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，存在正数和$n_0>0$使得对所有$n\geq n_0$有：$0 \leq cg(n) < f(n)$ $\}$
- 等价于 $f(n) / g(n) \to\infty$ ，as $n\to\infty$。
- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

- （5）**紧渐近界记号Θ**
- $\Theta(g(n)) = \{ f(n) \mid$ 存在正常数c_1, c_2和n_0使得对所有$n \geq n_0$有：
  $c_1 g(n) \leq f(n) \leq c_2 g(n) \}$
- **定理1：** $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

渐近分析记号在等式和不等式中的意义

- $f(n) = \Theta(g(n))$的确切意义是：$f(n) \in \Theta(g(n))$。
- 一般情况下，等式和不等式中的渐近记号$\Theta(g(n))$表示$\Theta(g(n))$中的某个函数。
- 例如：$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ 表示
- $2n^2 + 3n + 1 = 2n^2 + f(n)$，其中$f(n)$ 是$\Theta(n)$中某个函数。
- 等式和不等式中渐近记号O, o, Ω和ω的意义是类似的。

渐近分析中函数比较

- $f(n) = O(g(n)) \approx a \le b$;
- $f(n) = \Omega(g(n)) \approx a \ge b$;
- $f(n) = \Theta(g(n)) \approx a = b$;
- $f(n) = o(g(n)) \approx a < b$;
- $f(n) = \omega(g(n)) \approx a > b$.

渐近分析记号的若干性质

- （1）传递性：
- $f(n)=\Theta(g(n))$，$g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n))$；
- $f(n)=O(g(n))$，$g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$；
- $f(n)=\Omega(g(n))$，$g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$；
- $f(n)=o(g(n))$，$g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$；
- $f(n)=\omega(g(n))$，$g(n)=\omega(h(n)) \Rightarrow f(n)=\omega(h(n))$；

- （2）反身性：
- $f(n)=\Theta(f(n))$；
- $f(n)=O(f(n))$；

- $f(n)=\Omega(f(n))$.
- （3）对称性：
- $f(n)=\Theta(g(n)) \Leftrightarrow g(n)=\Theta(f(n))$.
- （4）互对称性：
- $f(n)=O(g(n)) \Leftrightarrow g(n)=\Omega(f(n))$ ；
- $f(n)=o(g(n)) \Leftrightarrow g(n)=\omega(f(n))$ ；

- （5）算术运算：
- $O(f(n))+O(g(n)) = O(\max\{f(\mathrm{n}),g(n)\})$ ；
- $O(f(n))+O(g(n)) = O(f(\mathrm{n})+g(n))$ ；
- $O(f(n))*O(g(n)) = O(f(\mathrm{n})*g(n))$ ；
- $O(cf(n)) = O(f(\mathrm{n}))$ ；
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(\mathrm{n}))$ 。

- 规则$O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$ 的**证明：**
- 对于任意$f_1(n) \in O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n \geq n_1$，有$f_1(n) \leq c_1 f(n)$。
- 类似地，对于任意$g_1(n) \in O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n \geq n_2$，有$g_1(n) \leq c_2 g(n)$。
- 令$c_3=\max\{c_1, c_2\}$， $n_3 =\max\{n_1, n_2\}$，$h(n)= \max\{f(n),g(n)\}$。
- 则对所有的 $n \geq n_3$，有
- $f_1(n) +g_1(n) \leq c_1 f(n) + c_2 g(n)$

  $\leq c_3 f(n) + c_3 g(n)= c_3(f(n) + g(n))$

  $\leq c_3 2 \max\{f(n),g(n)\}$

  $= 2c_3 h(n) = O(\max\{f(n),g(n)\})$.

算法渐近复杂性分析中常用函数

- （1）单调函数
- 单调递增：$m \le n \Rightarrow f(m) \le f(n)$；
- 单调递减：$m \le n \Rightarrow f(m) \ge f(n)$;
- 严格单调递增：$m < n \Rightarrow f(m) < f(n)$;
- 严格单调递减：$m < n \Rightarrow f(m) > f(n)$.
- （2）取整函数
- $\lfloor x \rfloor$：不大于x的最大整数；
- $\lceil x \rceil$：不小于x的最小整数。

取整函数的若干性质

- $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$;
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
- 对于$n \ge 0$，$a,b>0$，有：
- $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
- $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
- $\lceil a/b \rceil \le (a+(b-1))/b$;
- $\lfloor a/b \rfloor \ge (a-(b-1))/b$;
- $f(x)=\lfloor x \rfloor, g(x)=\lceil x \rceil$为单调递增函数。

- **（3）多项式函数**
- $p(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$； $a_d>0$;
- $p(n) = \Theta(n^d)$;

- $f(n) = O(n^k) \Leftrightarrow f(n)$ 多项式有界；
- $f(n) = O(1) \Leftrightarrow f(n) \leq c$;
- $k \geq d \Rightarrow p(n) = O(n^k)$;
- $k \leq d \Rightarrow p(n) = \Omega(n^k)$;
- $k > d \Rightarrow p(n) = o(n^k)$;
- $k < d \Rightarrow p(n) = \omega(n^k)$.

- （4）指数函数
- 对于正整数m,n和实数$a>0$:
- $a^0=1$;
- $a^1=a$;
- $a^{-1}=1/a$;
- $(a^m)^n = a^{mn}$;
- $(a^m)^n = (a^n)^m$;
- $a^m a^n = a^{m+n}$;
- $a>1 \Rightarrow a^n$为单调递增函数;
- $a>1 \Rightarrow \lim_{n\to\infty} \frac{n}{a^n} = 0 \Rightarrow n^b = o(a^n)$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

- $e^x \geq 1 + x$;
- $|x| \leq 1 \Rightarrow 1 + x \leq e^x \leq 1 + x + x^2$;
- $e^x = 1 + x + \Theta(x^2)$, as $x \to 0$;

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$$

- （5）对数函数
- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$l;
- $\log \log n = \log(\log n)$;
- for a>0,b>0,c>0

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n\log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

- $|x| \le 1 \Rightarrow \quad \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots.$
- for $x > -1$, $\quad \frac{x}{1+x} \le \ln(1+x) \le x$
- for any $a > 0$, $\quad \lim_{n\to\infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n\to\infty} \frac{\log^b n}{n^a} = 0$, $\Rightarrow \log^b n = o(n^a)$

- （6）阶层函数

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

- Stirling's approximation

$$n! = \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^{n}\left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \qquad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n \log n)$$

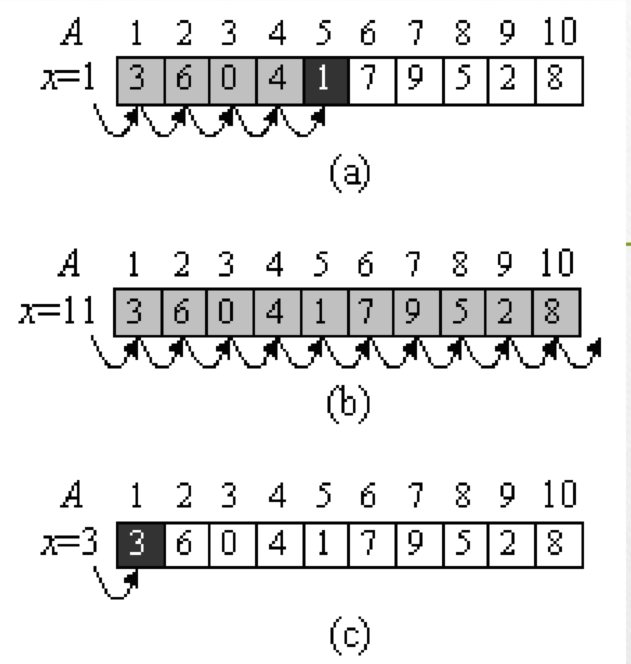
算法分析中常见的复杂性函数

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

算法分析方法

- 例：顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
        if (a[i]==k) return i;
    return -1;
}
```



- 在线性表A中查找。（a）查找值为x=1的元素，从A[1]起依次要进行5次检测，第一次找到值为1的元素。（b）查找值为x=11的元素，从A[1]起依次检测完所有元素（进行10次检测），没有找到值为11的元素——最坏情形。（c）查找值为x=3的元素，从A[1]起仅进行一次检测就找到值为3的元素——最好情形。

- （1） $T_{\max}(n) = \max\{\ T(\mathrm{I})\ |\ \mathrm{size}(\mathrm{I})=n\ \}=O(n)$
- （2） $T_{\min}(n) = \min\{\ T(\mathrm{I})\ |\ \mathrm{size}(\mathrm{I})=n\ \}=O(1)$
- （3）在平均情况下，假设：
- (a) 搜索成功的概率为 $p\ (0 \le p \le 1)$；
- (b) 在数组的每个位置 $i\ (0 \le i < n)$ 搜索成功的概率相同，均为 p/n。

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right) + n \cdot (1 - p)$$

$$= \frac{p}{n}\sum_{i=1}^{n} i + n(1 - p) = \frac{p(n+1)}{2} + n(1 - p)$$

算法分析的基本法则

- **非递归算法：**
- （1）for / while 循环
- 循环体内计算时间*循环次数；
- （2）嵌套循环
- 循环体内计算时间*所有循环次数；
- （3）顺序语句
- 各语句计算时间相加；
- （4）if-else语句
- if语句计算时间和else语句计算时间的较大者。

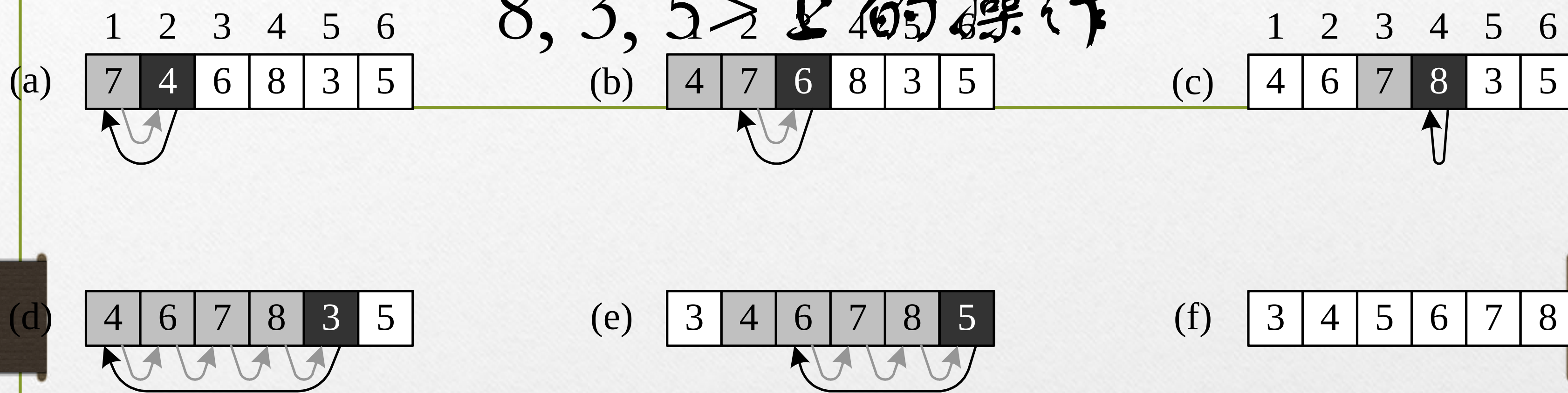
例：插入排序算法

- 1. 问题的理解与描述
- 排序问题的形式化表示为：
- 输入：一组数$<a_1, a_2, ..., a_n>$。
- 输出：输入的一个排列（重排）$<a'_1$
满足$a'_1 \leq a'_2 \leq, ..., \leq a'_n$。

```
INSERTION-SORT（A）
for j ← 2 to length[A]
    do key ← A[j]
      ▷将A[j]插入到排好序的序列A[1… j - 1]中。
      i ← j - 1
      while i > 0 and A[i] > key
         do A[i + 1] ← A[i]
            i ← i - 1
      A[i + 1] ← key
```

```
template<class Type>
void insertion_sort(Type *a, int n)
{
    Type key;                               //  cost      times
    for (int i = 1; i < n; i++){            //   c1        n
        key=a[i];                           //   c2        n-1
        int j=i-1;                          //   c3        n-1
        while( j>=0 && a[j]>key ){          //   c4        sum of ti
        a[j+1]=a[j];                        //   c5        sum of (ti-1)
        j--;                                //   c6        sum og (ti-1)
        }
        a[j+1]=key;                         //   c7        n-1
        }
}
```

INSERTION-SORT 在 $A = \langle 7, 4, 6, 8, 3, 5 \rangle$ 上的操作



数组的下标表示在各方格的上方，存储在数组中的数值表示在各方格内。（a）～（e）第1～8行的for循环的各次重复。每次重复中，黑色方格放的是键$A[j]$，它在第5行中逐一与其左边灰色方格内的元素比较。灰色的箭头指示处在第6行中被右移一格的元素，黑色箭头则指示出键在第8行移动到的位置。（f）最终排好序的数组。

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{i=1}^{n-1}t_i+c_5\sum_{i=1}^{n-1}(t_i-1)+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7(n-1)$$

- 在最好情况下，$t_i=1$, for $1 \le i < n$;

$$T_{\min}(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4(n-1)+c_7(n-1)$$

$$=(c_1+c_2+c_3+c_4+c_7)n-(c_2+c_3+c_4+c_7)=O(n)$$

- 在最坏情况下，$t_i \le i+1$, for $1 \le i < n$;

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

$$T_{\max}(n)\le c_1n+c_2(n-1)+c_3(n-1)+$$
$$c_4\left(\frac{n(n+1)}{2}-1\right)+c_5\left(\frac{n(n-1)}{2}\right)+c_6\left(\frac{n(n-1)}{2}\right)+c_7(n-1)$$
$$=\frac{c_4+c_5+c_6}{2}n^2+\left(c_1+c_2+c_3+\frac{c_4-c_5-c_6}{2}+c_7\right)n-(c_2+c_3+c_4+c_7)$$
$$=O(n^2)$$

- 对于输入数据a[i]=n-i,i=0,1,…,n-1，算法insertion_sort 达到其最坏情形。因此，

$$T_{\max}(n) \geq \frac{c_4+c_5+c_6}{2}n^2+\left(c_1+c_2+c_3+\frac{c_4-c_5-c_6}{2}+c_7\right)n-(c_2+c_3+c_4+c_7)$$

$$=\Omega(n^2)$$

- 由此可见，$T_{\max}(n)=\Theta(n^2)$

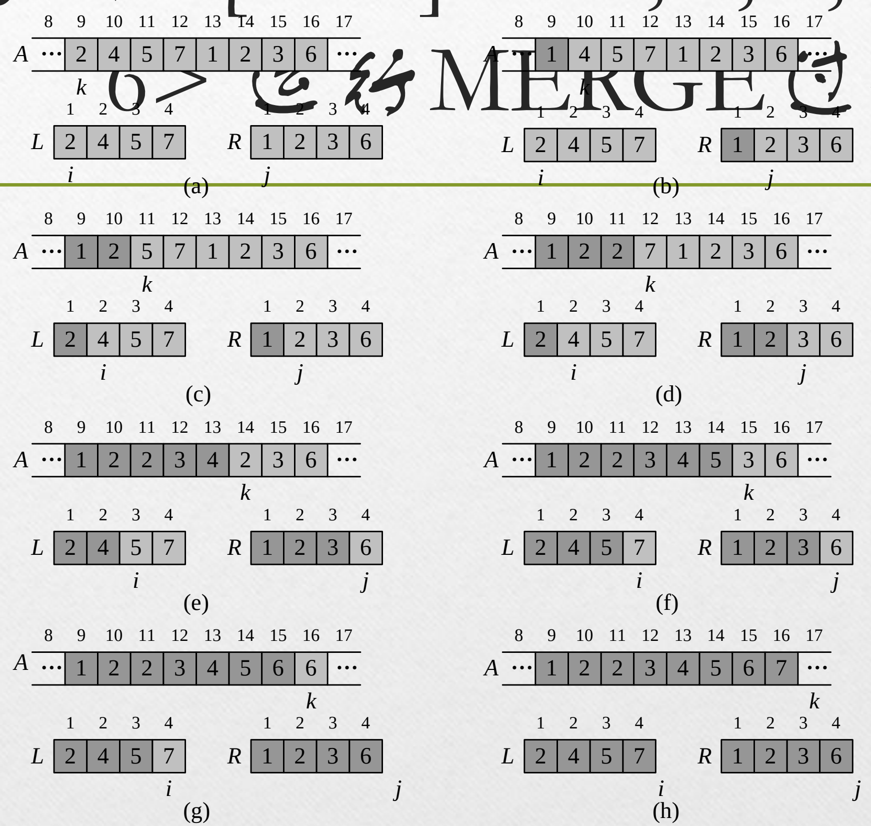
两个有序序列的合并算法

- 1. 问题的理解与描述
- 将两个有序序列合并为一个有序序列问题的形式化表示为：
- **输入：** 序列$A[p\ldots r]$。其中，子序列$A[p\ldots q]$和$A[q+1\ldots r]$是有序的。
- **输出：** $A[p\ldots r]$所有元素的重排，使之有序。

算法的伪代码描述

```
MERGE(A, p, q, r)
n1←q - p + 1
n2 ←r - q
创建数组L[1…n1]和R[1…n2]
for i ← 1 to n1
    do L[i] ← A[p + i - 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
i ←1
j ←1
k ←p
while i ≤ n1 and j ≤ n2
do if L[i] ≤ R[j]
        then A[k] ←L[i]
             i ←i + 1
        else A[k] ←R[j]
             j ←j + 1
    k←k+1
if i < n1
then 将L[i... n1]复制到A[k...r]
if j < n2
then 将R[j... n2]复制到A[k...r]
```

对序列$A[9..16]$ =<2, 4, 5, 7, 1, 2, 3, 6> 运行MERGE过程



- 使用两个辅助数组L和R进行合并操作。在复制后，数组L包含<2, 4, 5, 7>，数组R包含<1, 2, 3, 6>。A中深灰色的位置包含最终值，L和R中浅灰色位置包含的值尚未复制回A中。A的浅灰色位置是将被覆盖的，而L和R中深灰色位置包含的是已经被复制回A的值。(a)～(g)是第12～17行的while循环每次重复之初的A、L、R及其下标k、i、j。(h)是执行了第18～21将L中剩余的元素复制到$A[k...r]$中。此时，子数组$A[9…16]$已经排好序。

- 循环不变量：3. 算法的正确性
- 在第12～17行的while循环的每次重复之初，子数组$A[p\ldots k-1]$包含$L[1\ldots n1]$和$R[1\ldots n2]$中的$k-p$个最小的元素，并排好序。此外，$L[i]$和$R[j]$分别是各自数组中尚未复制回数组A的元素中的最小者。
- 可利用此循环不变量来证明合并算法MERGE是正确的。

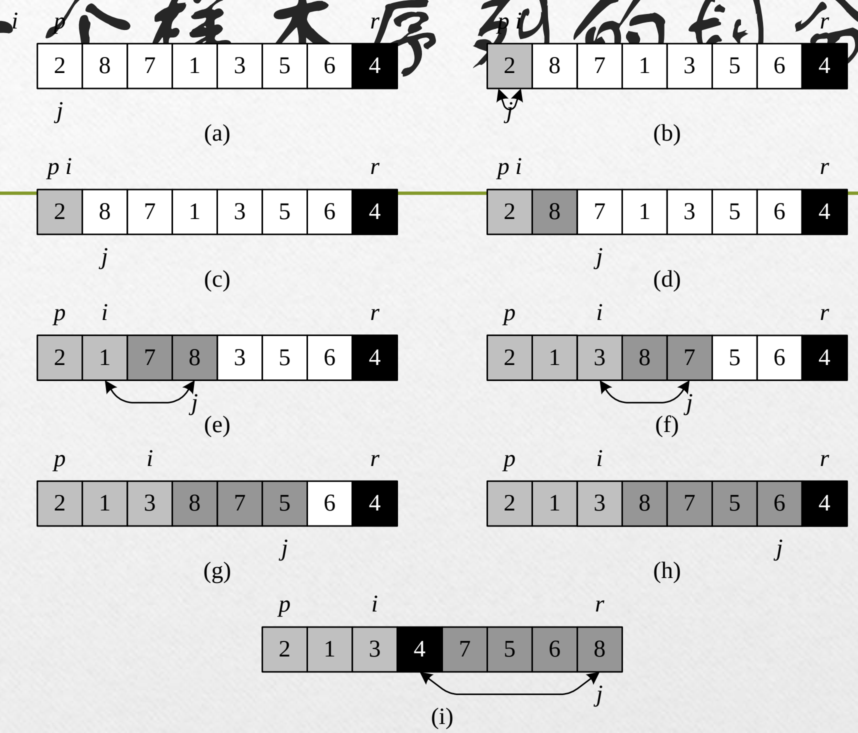
序列的划分

- 1. 问题的理解与描述
- 序列的划分问题描述如下：
- **输入：**序列$A[p...r]$。
- **输出：**下标q（$p \le q \le r$），原序列$A[p...r]$的一个重排：使得$A[p...q]$中的元素值不超过$A[q]$(=原$A[r]$的值)，而$A[q+1...r]$中的元素值均大于$A[q]$。

2. 算法的伪代码描述

- PARTITION(A, p, r)
- 1 $x \leftarrow A[r]$
- 2 $i \leftarrow p - 1$
- 3 **for** $j \leftarrow p$ **to** $r - 1$
- 4 **do if** $A[j] \leq x$
- 5 **then** $i \leftarrow i + 1$
- 6 exchange $A[i] \leftrightarrow A[j]$
- 7 exchange $A[i + 1] \leftrightarrow A[r]$
- 8 **return** $i + 1$

对一个样本序列的划分操作



- （a）初始的序列和变量设置。没有任何元素进入上述两部分。（b）～（h）表示第3～6行的for循环的每一次重复。黑色双向箭头表示第6行的元素交换操作。（i）表示上述循环终止后第7行执行的元素交换操作。

3. 算法的正确性

- 循环不变量：
- 在第3～6行的for循环的每次重复之初，对每一个数组下标k，
- 若 $p \le k \le i$，则$A[k] \le x$。
- 若 $i + 1 \le k \le j - 1$，则$A[k] > x$。
- 若 $k = r$，则$A[k] = x$。
- 我们用此循环不变量来证明算法1-3的正确性。

4. 算法的运行时间

- 假定序列$A[p \ldots r]$中含有n个元素。在此过程中，第$3 \sim 6$行的**for**循环重复了n次，所以该算法的最坏情形运行时间为$\Theta(n)$。