课程回顾

- 递归式（举例：阶乘、斐波那契数列、Ackermann函数、汉诺塔问题、全排列问题）
- 分治法求解：分解、解决、合并
- 求解递归式（计算时间复杂度）：
  - 代入法：猜测解+数学归纳法证明（注意边界条件！）
    - 注意事项：做出好的猜测、细节修正、避免陷阱、变量代换
  - 递归树法
  - 主方法

递归式求解——递归树法

- 递归树中每个结点表示一个单一子问题的代价，子问题对应某次递归函数调用
- 树中每层的代价求和得到每层代价，将所有层的代价求和，得到所有层次递归调用总代价
- 递归树是展开过程的形象化，从$T(n)$逐步展开直到$T(1)$

递归式求解——递归树法 (续)

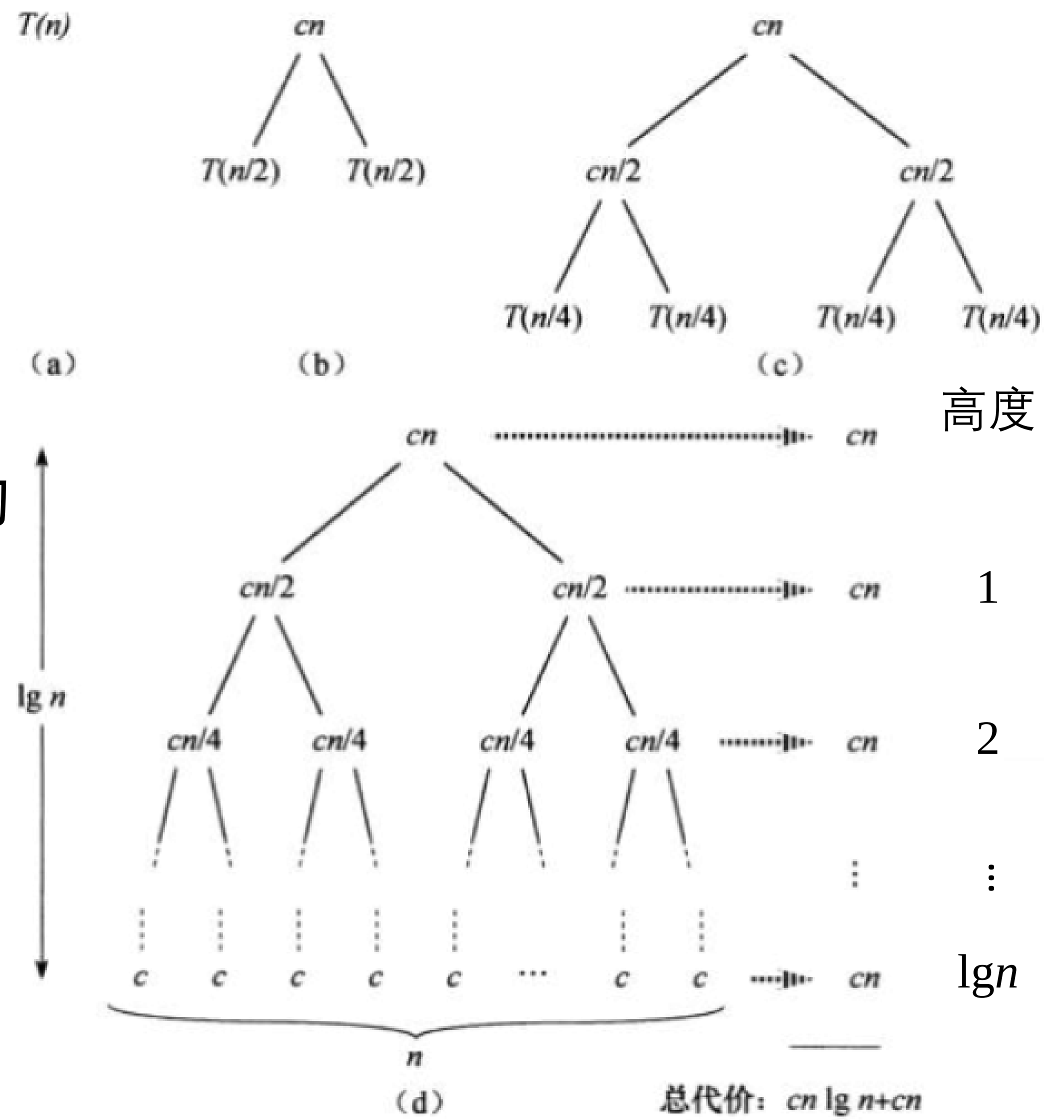
■例1：（教材p21）
归并排序
$T(n)=2T(n/2)+cn$
（不妨设$n=2^k$）

➢树高（从根到叶的最长简单路径长度）：$\lg n$

➢总层数：$\lg n+1$

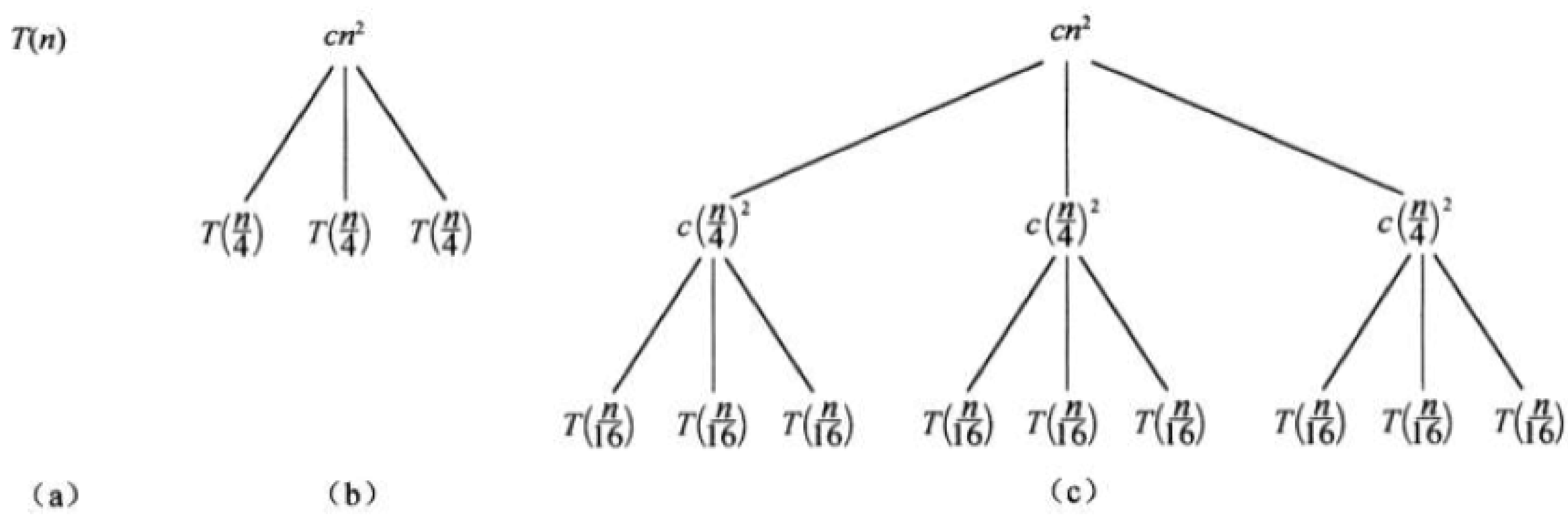
➢每层代价：cn

➢总代价：$cn\lg n+cn$



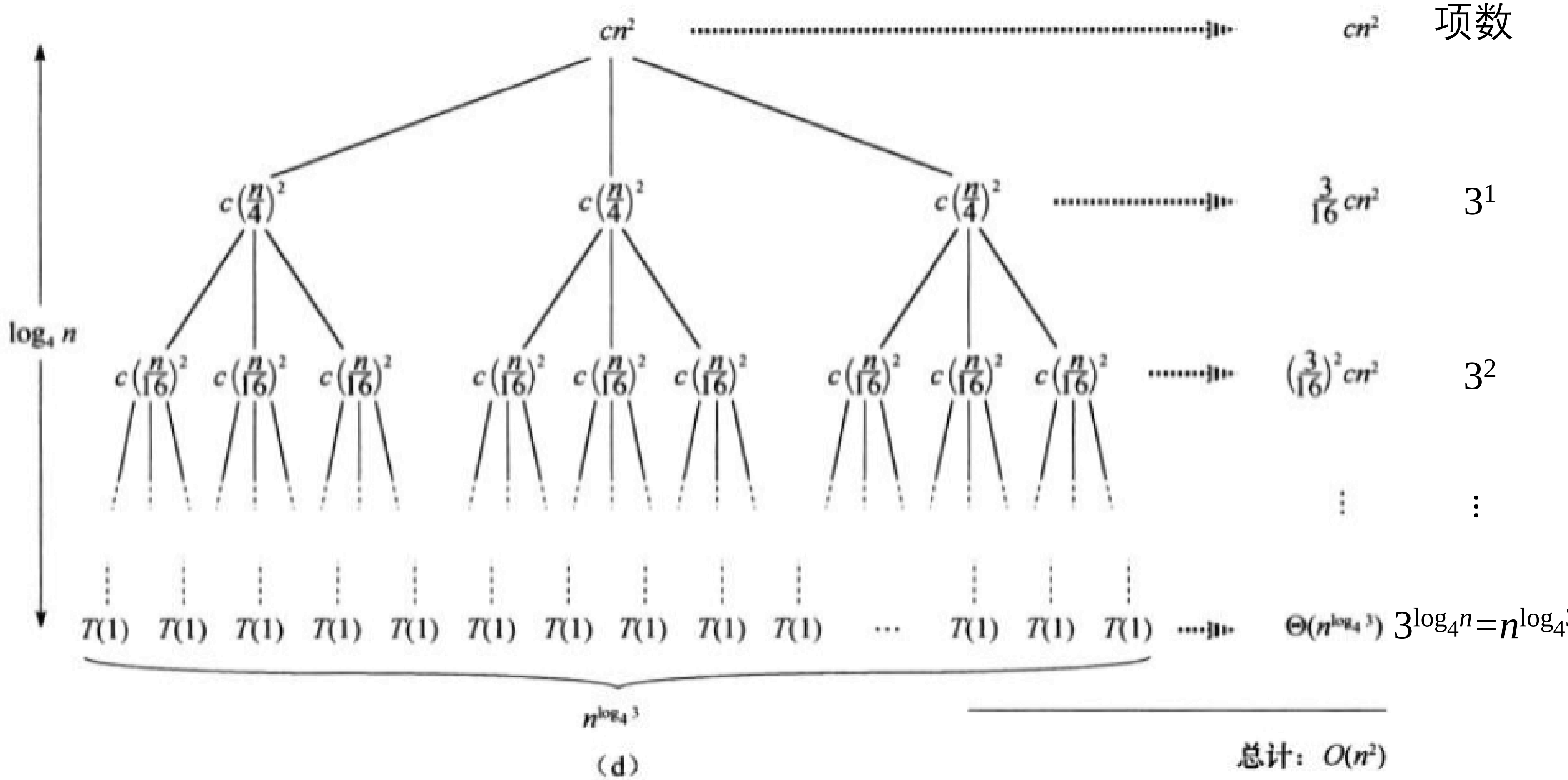
递归式求解——递归树法 (续)

■例2：（教材p51）$T(n)=3T(\lfloor n/4 \rfloor)+cn^2$ （设n=4^k）



递归式求解——递归树法 (续)

■例2：（教材p51）$T(n)=3T(\lfloor n/4 \rfloor)+cn^2$ （设$n=4^k$）



递归式求解——递归树法 (续)

■例2： （教材p51）$T(n)=3T(\lfloor n/4\rfloor)+cn^2$ （设n=4^k）

$$
\begin{aligned}
T(n) &= cn^2+\frac{3}{16}cn^2+\left(\frac{3}{16}\right)^2cn^2+\ldots+\left(\frac{3}{16}\right)^{\log_4 n-1}cn^2+\Theta(n^{\log_4 3})\\
&= \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2+\Theta(n^{\log_4 3})\\
&= \frac{1-(3/16)^{\log_4 n}}{13/16}cn^2+\Theta(n^{\log_4 3})\\
&< \frac{16}{13}cn^2+\Theta(n^{\log_4 3}) \qquad //0<(3/16)^{\log_4 n}\le 1\\
&= O(n^2)
\end{aligned}
$$

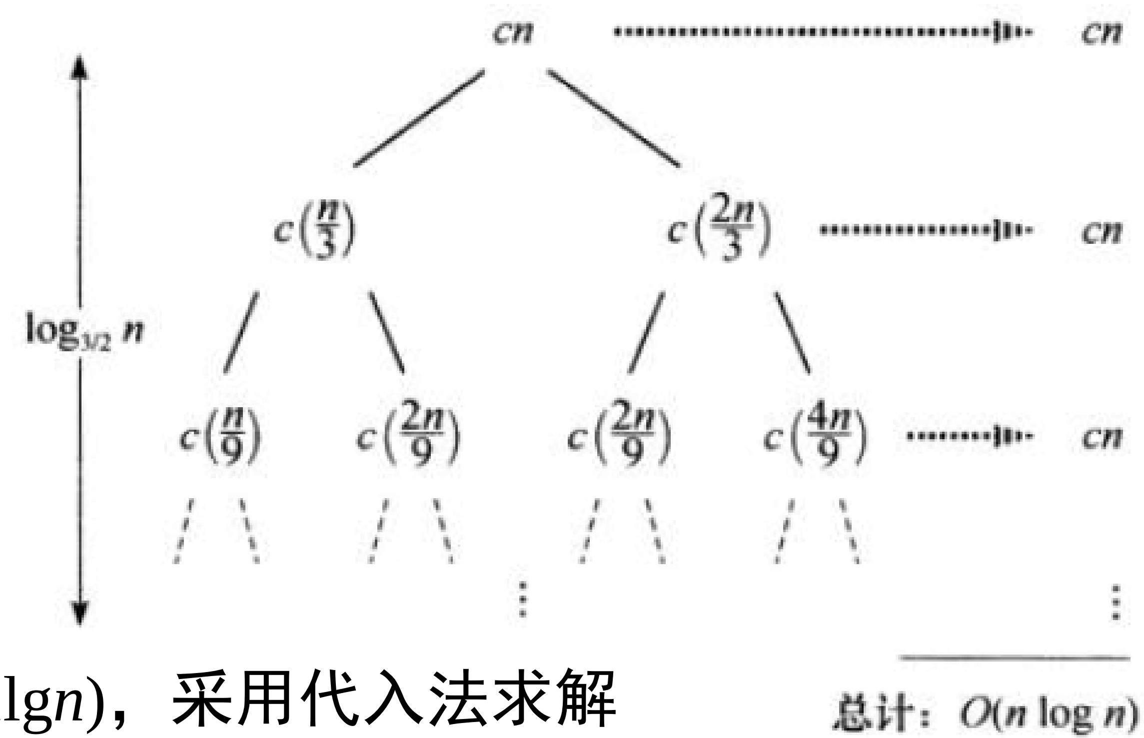
递归式求解——递归树法 (续)

■例3：（教材p52）更复杂的例子：树不一定是满二叉树，叶子深度不尽相同

$T(n)=T(n/3)+T(2n/3)+O(n)$

树高k满足：

$(2/3)^k \cdot n=1$



猜测$T(n)=O(n\lg n)$，采用代入法求解

递归式求解——主方法

- The master method，通用法，万能法
- 可迅速求解
  - $T(n)=aT(n/b)+f(n)$ //常数$a\geq 1$，$b>1$，$f(n)$为渐近正函数
  - 将规模为n的问题划分为a个子问题，每个子问题规模为n/b，每个子问题时间为$T(n/b)$，划分和合并的时间为$f(n)$
  - 注：n/b不一定为整数，实际中应当用$\lceil n/b \rceil$或$\lfloor n/b \rfloor$替换，但不影响渐近性质

递归式求解——主方法 (续)

■定理4.1（教材p53-54，主定理） 令$a \geq 1$和$b > 1$是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将n/b解释为$\lceil n/b \rceil$或$\lfloor n/b \rfloor$，有如下渐近界：

1. 若对某个常数$\varepsilon > 0$有$f(n) = O(n^{\log_b a - \varepsilon})$，则$T(n) = \Theta(n^{\log_b a})$
2. 若$f(n) = \Theta(n^{\log_b a})$，则$T(n) = \Theta(n^{\log_b a} \lg n)$
3. 若对某个常数$\varepsilon > 0$有$f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对某个常数$c < 1$和所有足够大的$n$有$af(n/b) \leq cf(n)$，则$T(n) = \Theta(f(n))$

递归式求解——主方法 (续)

■定理意义：比较$f(n)$和$n^{\log_b a}$，直观上**两函数较大者决定$T(n)$**

1. $n^{\log_b a}$比$f(n)$大一个多项式因子n^{ε}：$T(n) = \Theta(n^{\log_b a})$
2. 两者相同，乘以对数因子lg$n$：
   $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n f(n))$
3. $f(n)$比$n^{\log_b a}$大一个多项式因子n^{ε}，以及满足“正则”条件：$T(n) = \Theta(f(n))$

■注：**三种情况并未覆盖所有可能的$f(n)$**，存在间隙

递归式求解——主方法 (续)

$T(n) = aT(n/b) + f(n)$
子问题个数a，子问题规模n/b
求解与合并时间$f(n)$与$n^{\log_b a}$比较

■例1：

$$T(n) = 9T(n/3) + n$$

- a=9，b=3，$f(n)$=n，$n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$
- $f(n) = O(n^{\log_3 9-1}) = O(n)$, $\varepsilon = 1$
- $n^{\log_b a}$更大，主定理第1种情况：$T(n)=\Theta(n^2)$

递归式求解——主方法 (续)

■例2：

$$T(n) = T(2n/3) + 1$$

$T(n) = aT(n/b) + f(n)$

子问题个数a，子问题规模n/b

求解与合并时间$f(n)$与$n^{\log_b a}$比较

- a=1，b=3/2，$f(n)$=1，$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = \Theta(1)$
- $f(n) = 1 = \Theta(1)$
- $n^{\log_b a}$与$f(n)$同级别，主定理第2种情况：$T(n)=\Theta(\lg n)$

递归式求解——主方法 (续)

■例3：

$T(n) = 3T(n/4) + n\lg n$

$T(n) = aT(n/b) + f(n)$
子问题个数a，子问题规模n/b
求解与合并时间$f(n)$与$n^{\log_b a}$比较

- a=3，b=4，$f(n)$=nlg$n$，$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
- $f(n) = n\lg n = \Omega(n) = \Omega(n^{\log_4 3+\epsilon})$, $\varepsilon \approx 0.2$
- c=3/4且n足够大时，$af(n/b)$=3(n/4)lg(n/4)≤(3/4)nlg$n$=$cf(n)$
- $f(n)$更大，主定理第3种情况：$T(n)$=Θ(nlg$n$)

递归式求解——主方法 (续)

$T(n) = aT(n/b) + f(n)$
子问题个数*a*，子问题规模*n*/*b*
求解与合并时间*f*(*n*)与$n^{\log_b a}$比较

■例4：

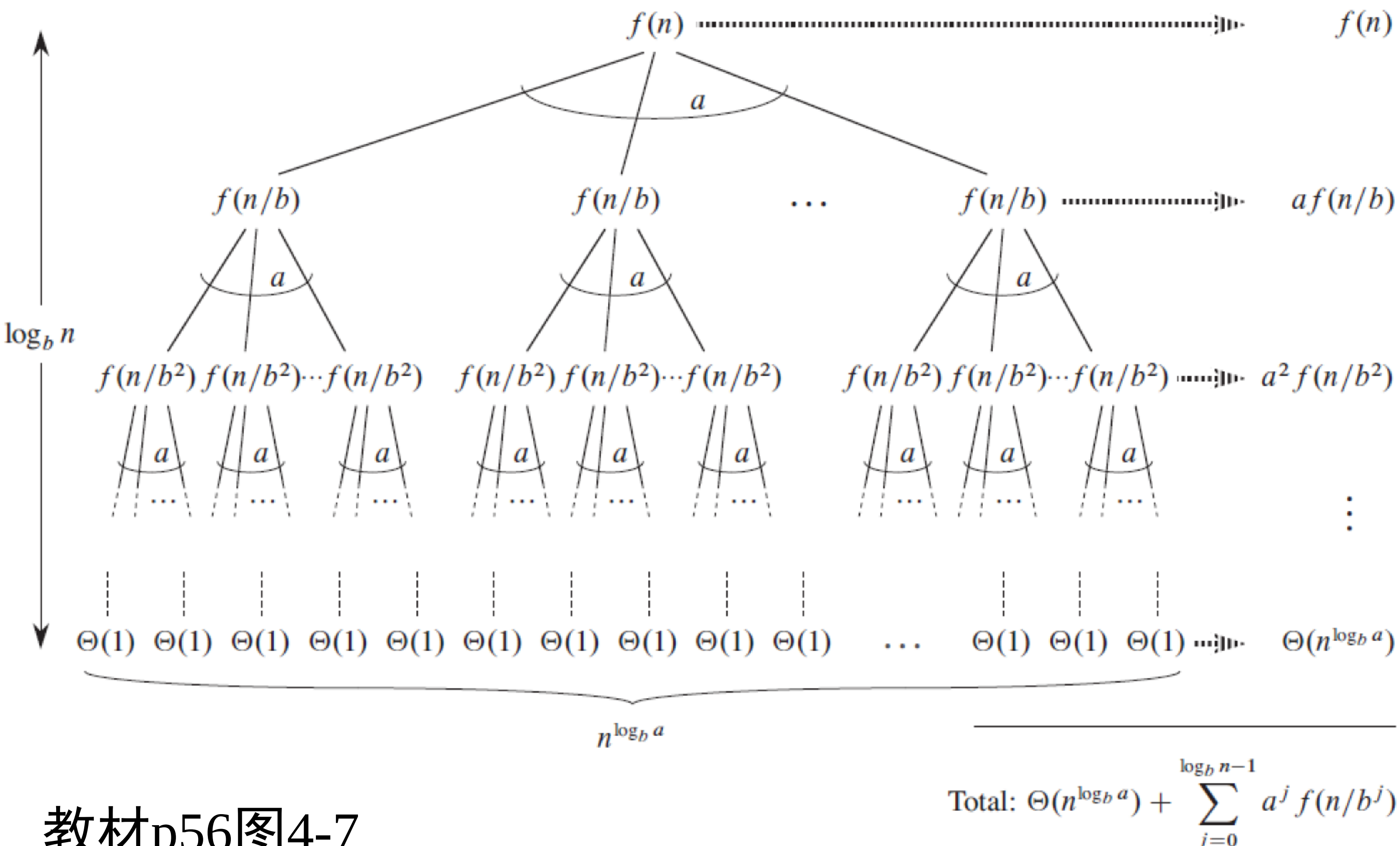
$$T(n) = 2T(n/2) + n\lg n$$

- *a*=2，*b*=2，*f*(*n*)=*n*lg*n*，$n^{\log_b a} = n^{\log_2 2} = \Theta(n)$
- $f(n) = n\lg n = \Omega(n)$，但找不到ε使得$n\lg n = \Omega(n^{1+\varepsilon})$

$$\lim_{n\to\infty} \frac{n^{\varepsilon}}{\lg n} = \infty, \quad \varepsilon > 0$$

- 该递归式落入了情况2和3的间隙，无法使用主方法

递归式求解——主方法 (续)



教材p56图4-7

分治法的适用条件

1. 该问题的规模缩小到一定的程度就可以容易地解决

    - 因为问题的计算复杂性一般是随着问题规模的增加而增加，因此大部分问题满足这个特征

2. 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质

    - 这条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了递归思想的应用

分治法的适用条件 (续)

3. 利用该问题分解出的子问题的解可以合并为该问题的解
   - 能否利用分治法完全取决于问题是否具有这条特征，如果具备了前两条特征，而不具备第三条特征，则可以考虑贪心算法或动态规划
4. 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题
   - 这条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用动态规划较好

分治法求解——二分搜索

- 给定已按升序排好序的n个元素$A[1..n]$，现要在这n个元素中找出一特定元素x
- 适用条件分析：
  - ➢ 该问题的规模缩小到一定的程度就可以容易地解决 ✔
    - 分析：如果n=1即只有一个元素，则只要比较这个元素和x就可以确定x是否在数组中
  - ➢ 该问题可以分解为若干个规模较小的相同问题 ✔
    - 分析：比较x和A的中间元素$A[mid]$，若$x=A[mid]$，则x在A中的位置就是mid；如果$x<A[mid]$，由于A是递增排序的，因此假如x在A中的话，x必然排在$A[mid]$的前面，所以只要在$A[low..mid-1]$查找x即可；如果$x>A[mid]$，同理只要在$A[mid+1..high]$查找x即可。无论如何查找x，其方法都和在A中查找x一样，只不过是查找的规模缩小了
  - ➢ 分解出的子问题的解可以合并为原问题的解 ✔

分治法求解——二分搜索 (续)

- 给定已按升序排好序的n个元素$A[1..n]$，现要在这n个元素中找出一特定元素x
- 适用条件分析：
  - ➢分解出的各个子问题是相互独立的 ✔
    - 分析：显然此问题分解出的子问题相互独立，即在$A[low..mid\text{-}1]$和$A[mid+1..high]$ 查找x是独立的子问题

分治法求解——二分搜索 (续)

■适用范围：顺序表、有序序列

■基本思想

1. 设$A[low..high]$是当前查找区间，当$low \le high$时进行查找（否则A数组为空）
2. 确定中点位置 $mid = \lfloor (low + high)/2 \rfloor$
3. 将待查值x与$A[mid]$比较
   ① 若$x=A[mid]$：查找成功，返回位置mid
   ② 若$x<A[mid]$：查找左子数组$A[low..mid\text{-}1]$
   ③ 若$x>A[mid]$：查找右子数组$A[mid+1..high]$

分治法求解——二分搜索 (续)

```
BINARY_SEARCH(A, x, low, high)
if low ≤ high
    mid ← ⌊(low + high)/2⌋
    if x = A[mid]
        return mid
    if x < A[mid]
        return BINARY_SEARCH(A, x, low, mid-1)
    else return BINARY_SEARCH(A, x, mid+1, high)
return 0
```

■最坏情况运行时间：$T(n)=T(n/2)+\Theta(1)$，由主方法易得$T(n)=\Theta(\lg n)$

分治法求解——最大子数组问题

■ （教材p39）寻找数组$A[1..n]$中，和最大的非空连续子数组

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

最大子数组

图 4-3 股票价格变化值的最大子数组问题。本例中，子数组 $A[8..11]$的和是 43，是 A 的所有连续子数组中和最大的

- ➢暴力求解：穷举所有非空连续子数组，即穷举所有可能的起始、终止位置：$C_n^2 = \dfrac{n(n-1)}{2} = \Theta(n^2)$
- ➢可在$O(n^3)$时间解决，改进方法可在$O(n^2)$时间解决

分治法求解——最大子数组问题 (续)

■分治法适用条件分析：

- ➢该问题的规模缩小到一定程度就可容易解决 ✔
  - 分析：如果n=1即只有一个元素，只要输出该元素值即可
- ➢该问题可以分解为若干个规模较小的相同问题 ✔
  - 分析：将数组$A[low..high]$对半划分，中间位置为mid，A的非空连续子数组$A[i..j]$肯定在三种情况之一：

1. 完全位于子数组$A[low..mid]$中：$low \le i \le j \le mid$
2. 完全位于子数组$A[mid+1..high]$中：$mid+1 \le i \le j \le high$
3. 跨越了中点mid：$low \le i \le mid < j \le high$

前两种情况相当于求解规模更小的原问题；第3种情况易于计算

分治法求解——最大子数组问题 (续)

■分治法适用条件分析：

- 分解出的子问题的解可以合并为原问题的解 ✔
  - 分析：将三种情况得到的最大和进行比较即可
- 分解出的各个子问题是相互独立的 ✕
  - 分析：求和过程可能多次计算，并非独立

存在更好的解法：动态规划、贪心算法
但分治法依旧可以实现

分治法求解——最大子数组问题 (续)

■ 寻找跨越中点*mid*的最大子数组

$A[max\_left..max\_right]$： $low \le max\_left \le mid < max\_right \le high$

```
FIND_MAX_CROSSING_SUBARRAY(A, low, mid, high)
left_sum ← -∞
sum ← 0
for i ← mid downto low do
    sum ← sum + A[i]
    if sum > left_sum
        left_sum ← sum,  max_left ← i
right_sum ← -∞
sum ← 0
for j ← mid+1 to high do
    sum ← sum + A[j]
    if sum > right_sum
        right_sum ← sum,  max_right ← j
return (max_left, max_right, left_sum + right_sum)
```

$\Theta(n)$

分治法求解——最大子数组问题 (续)

```
FIND_MAXIMUM_SUBARRAY(A, low, high)
if high = low
    return (low, high, A[low])
else mid ← ⌊(low + high)/2⌋
    (left_low, left_high, left_sum) ←
        FIND_MAXIMUM_SUBARRAY(A, low, mid)   //T(n/2)
    (right_low, right_high, right_sum) ←
        FIND_MAXIMUM_SUBARRAY(A, mid+1, high) //T(n/2)
    (cross_low, cross_high, cross_sum) ←
        FIND_MAX_CROSSING_ SUBARRAY(A, low, mid, high) //Θ(n)
 if left_sum ≥ right_sum and left_sum ≥ cross_sum
    return (left_low, left_high, left_sum)
 elseif right_sum ≥ left_sum and right_sum ≥ cross_sum
    return (right_low, right_high, right_sum)
else return (cross_low, cross_high, cross_sum)
```

分治法求解——最大子数组问题 (续)

■算法分析

$$T(n) = 2T(n/2) + \Theta(n)$$

- 与归并排序类似
- 主方法求解：a=2，b=2，$f(n)$=cn，$n^{\log_b a} = n^{\log_2 2} = \Theta(n)$
- $f(n) = \Theta(n)$
- $n^{\log_b a}$与$f(n)$同级别，主定理第2种情况：$T(n)$=$\Theta(n\lg n)$

分治法求解——矩阵乘法

■方阵乘法问题：设矩阵A=(a_{ij})和B=(b_{ij})都是$n \times n$的方阵，则定义乘积矩阵C=A·B中的元素c_{ij}为：

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

■传统方法：$O(n^3)$

- 计算每一个元素c_{ij}都要做n次乘法和n-1次加法，因此计算矩阵C所有n^2个元素所需时间为$O(n^3)$

是否有更快的求解方法？考虑分治法

分治法求解——矩阵乘法 (续)

■不妨假设n=2^k，将矩阵A, B, C都分块成4个大小相等的子矩阵

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

■此时

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

- 8次规模为n/2的矩阵乘法，4次包含n^2/4个元素的矩阵加法，时间复杂度：$T(n)=8T(n/2)+\Theta(n^2)= \Theta(n^3)$

减少划分的子问题数量，即降低矩阵乘法次数

分治法求解——Strassen算法

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

■将分块矩阵进行如下操作：

$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12})B_{22} \\ P_3 &= (A_{21} + A_{22})B_{11} \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}) \end{aligned} \Rightarrow \begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

子问题数量降低为7，$T(n)=7T(n/2)+\Theta(n^2)=\Theta(n^{\lg 7})$

分治法求解——大整数乘法

■考虑两个n位大整数$A=a_1a_2a_3\ldots a_n$和$B=b_1b_2b_3\ldots b_n$相乘（a_i和b_j分别表示A第i位的数字和B第j位的数字），可列竖式计算：

$$
\begin{array}{r}
a_1\ a_2 \ldots a_n \\
\times \qquad\qquad\qquad b_1\ b_2 \ldots b_n \\
\hline
(d_{10})\ d_{11}d_{12}\ldots d_{1n} \\
(d_{20})\ d_{21}d_{22}\ldots d_{2n}\quad\ \ \\
\ldots\ldots \qquad\qquad\qquad \\
(d_{n0})\ d_{n1}d_{n2}\ldots d_{nn}\qquad\qquad \\
\hline
\Sigma \qquad\qquad\qquad\qquad\qquad\qquad
\end{array}
$$

➢计算n^2次乘法和n-1次加法，时间复杂度$\Theta(n^2)$

分治法求解——大整数乘法 (续)

- 与矩阵乘法类似考虑采用分治法解决：将一个n位大整数划分为两个$n/2$位的大整数，即$A=A_1A_2$和$B=B_1B_2$（$A_1=a_1a_2\ldots a_{n/2}$，$A_2=a_{n/2+1}a_{n/2+2}\ldots a_n$，$B_1=b_1b_2\ldots b_{n/2}$，$B_2=b_{n/2+1}b_{n/2+2}\ldots b_n$）

$$A\cdot B = A_1\cdot B_1\cdot 10^n + (A_1\cdot B_2+A_2\cdot B_1)\cdot 10^{n/2} + A_2\cdot B_2$$

  - 例：A=2135，B=4014，划分后A_1=21，A_2=35，B_1=40，B_2=14

$$A\cdot B = (21\cdot 10^2+35)\cdot(40\cdot 10^2+14)$$

$$= 21\cdot 40\cdot 10^4+(21\cdot 14+35\cdot 40)\cdot 10^2+35\cdot 14$$

  - $T(n)=4T(n/2)+\Theta(n)=\Theta(n^2)$没有改进，减少子问题数量！

分治法求解——大整数乘法 (续)

- 将子问题数量从4降到3：
  - $C_1=A_1\cdot B_1$，$C_2=A_2\cdot B_2$，$C_3=(A_1+A_2)\cdot(B_1+B_2)$
  - $A\cdot B = C_1\cdot 10^n + (C_3-C_1-C_2)\cdot 10^{n/2} + C_2$
  - $T(n)=3T(n/2)+\Theta(n)=\Theta(n^{\lg 3})$
- 如果将大整数分成更多段，用更复杂的方式把它们组合起来，将有可能得到更优的算法
- 最终这个思想导致快速傅利叶变换(Fast Fourier Transform)的产生（FFT可参见教材Chap 30）

分治法求解——快速排序

- （教材p95）快速排序尽管最坏情况时间复杂度是$\Theta(n^2)$，但平均情况时间复杂度为$\Theta(n\lg n)$
- 基于比较的排序算法时间下界为：

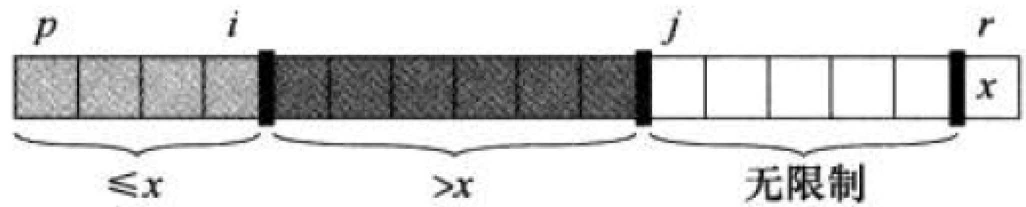
$$\begin{aligned}\lg(n!) &= \lg(\sqrt{2\pi n}(n/e)^n(1+\Theta(1/n))) \\ &= \lg(2\pi n)/2 + n\lg(n/e) + \lg(1+\Theta(1/n)) \\ &= \lg(2\pi)/2 + (\lg n)/2 + n\lg n - n\lg e + \lg(1+\Theta(1/n)) \\ &\approx n\lg n - 1.44n + \Theta(\lg n)\end{aligned}$$

- 快速排序平均情况：$1.39n\lg n+O(n)$，因系数较小称为“快排”

分治法求解——快速排序 (续)

■算法描述

```
QUICKSORT(A, p, r)
if p < r
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q-1)
    QUICKSORT(A, q+1, r)
```




```
PARTITION(A, p, r)
x ← A[r]  //划分元/主元(pivot element)
i ← p - 1
for j ← p to r-1 do
    if A[j] ≤ x
        i ← i + 1,  exchange A[i] with A[j]
exchange A[i+1] with A[r]
return i+1
```

循环不变式：

1. $A[p..i] \le x$
2. $A[i+1..j-1] > x$
3. $A[r] = x$

该划分使得：

$A[p..q-1] \le A[q] < A[q+1..r]$

分治法求解——快速排序 (续)

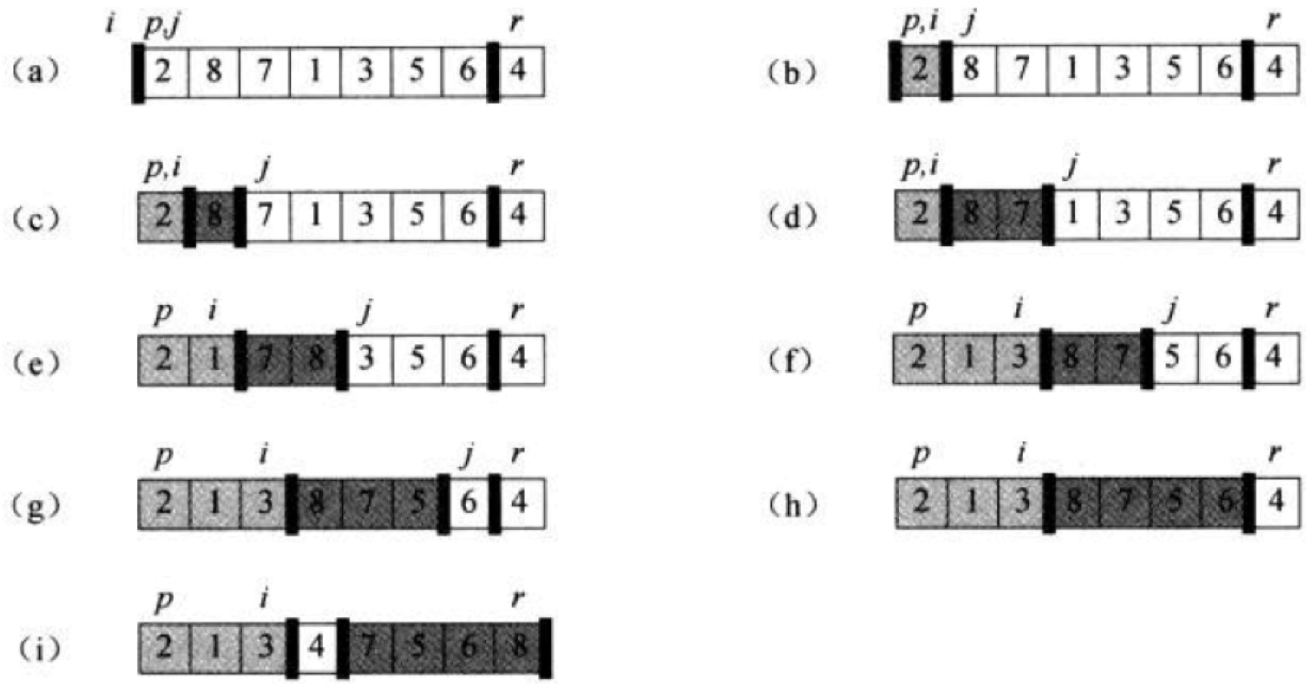


图 7-1　在一个样例数组上的 PARTITION 操作过程。数组项 $A[r]$是主元 x。浅阴影部分的数组元素都在划分的第一部分，其值都不大于 x。深阴影部分的元素都在划分的第二部分，其值都大于 x。无阴影的元素则是还未分入这两个部分中的任意一个。最后的白色元素就是主元 x。(a)初始的数组和变量设置。数组元素均未被放入前两个部分中的任何一个。(b)2 与它自身进行交换，并被放入了元素值较小的那个部分。(c)～(d)8 和 7 被添加到元素值较大的那个部分中。(e)1 和 8 进行交换，数值较小的部分规模增加。(f)数值 3 和 7 进行交换，数值较小的部分规模增加。(g)～(h)5 和 6 被包含进较大部分，循环结束。(i)在第 7～8 行中，主元被交换，这样主元就位于两个部分之间