

INTRODUCTION TO

# ALGORITHMS

THIRD EDITION

# 分治策略

《算法导论》—— 第3讲

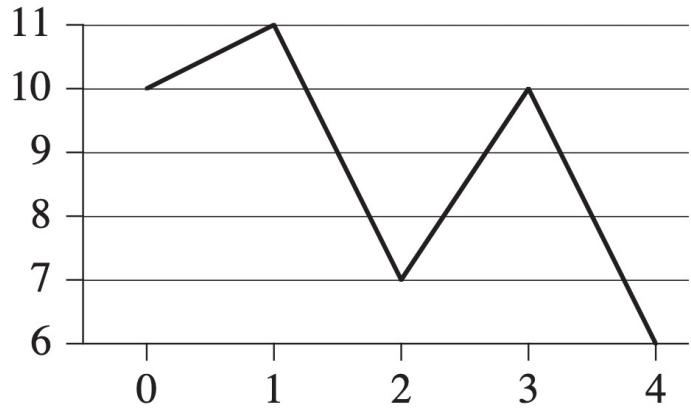
*jiacaicui@163.com*

# 内容提要

- 复习分治 (Divide and Conquer) 算法设计
  - 分 (Divide) 、治 (Conquer) 、合 (Combine)
- 学习两个经典的分治算法
  - 最大子数组问题的分治算法
  - 矩阵乘法的 Strassen 算法
- 学习递归式 (Recurrence) 的求解方法
  - 代入法
  - 递归树法
  - 主方法
    - 证明主定理

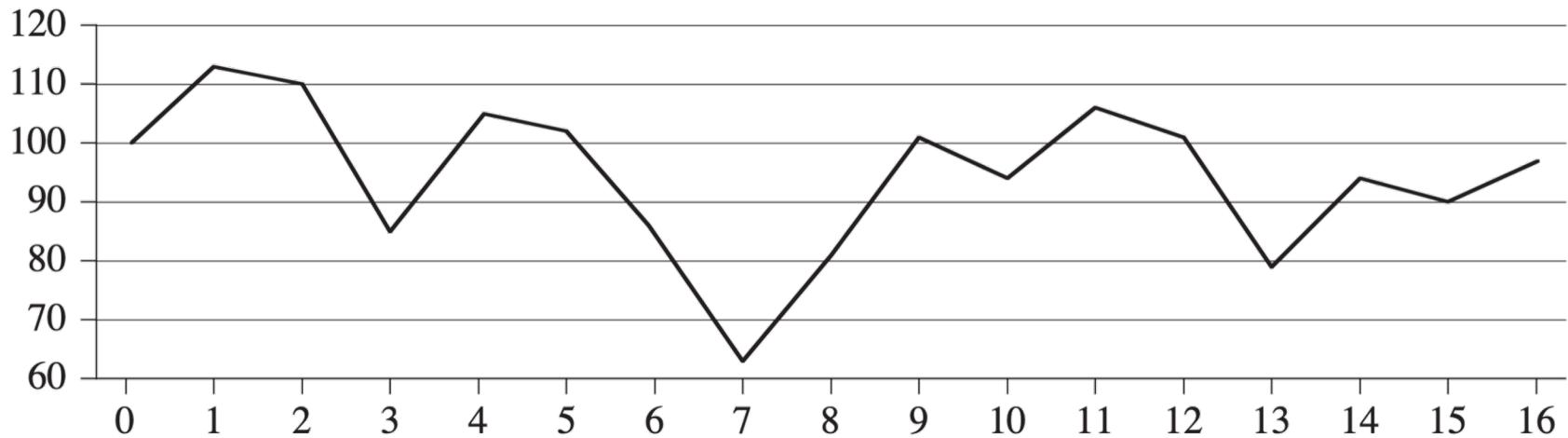
# 最大子数组问题

购买股票的马后炮



# 引例：股票购买问题

输入：一段时间内的股票价格。



输出：购进和售出的时间。

暴力思路：排查所有  $\binom{n}{2} = \Theta(n^2)$  种可能，至少需要  $\Omega(n^2)$  的代价。

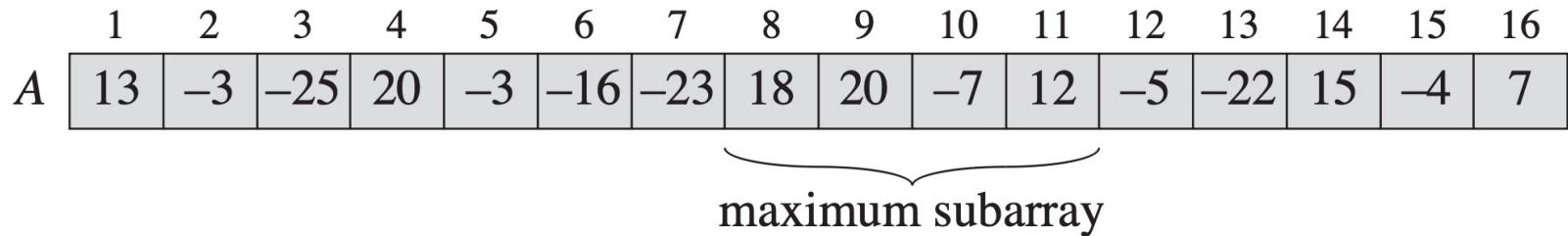
# 引例：股票购买问题

输入：一段时间内的股票价格。

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

输出：购进和售出的时间。

问题变换：考虑每日价格的变化，寻找**差价最大的**两天就是寻找**价格变化总和最大的区间**。



- 问题变换需要  $\Theta(n)$ ，只需要能优于  $\Omega(n^2)$  解决**最大子数组** (maximum subarray) 问题即可超越暴力算法。

# 最大子数组问题

输入：数组  $A[1..n]$ 。

输出： $(i, j, s)$  使得  $A[i..j]$  为最大子数组，且和为  $s$ 。

- 分治策略：
  - 分 (Divide)：取数组中点  $mid$ ，最大子数组要么完全位于左侧或者右侧，要么跨越中点。
  - 治 (Conquer)：完全位于某一侧可递归解决，需  $2T(\frac{n}{2})$ ；跨越中点则从中点向两侧扩张遍历解决，需  $\Theta(n)$ 。
  - 合 (Combine)：三种情况取和最大的即可，需  $\Theta(1)$ 。
- 综上，代价  $T(n) = 2 T\left(\frac{n}{2}\right) + \Theta(n)$ ，不难得出  $T(n) = n \log n$ 。

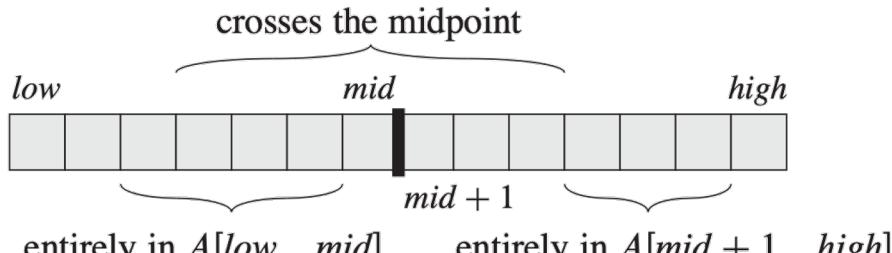
# 寻找最大子数组伪代码

FIND-MAXIMUM-SUBARRAY( $A, low, high$ )

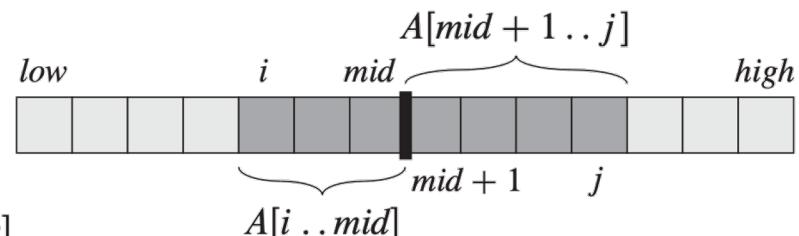
```

1  if  $high == low$ 
2      return ( $low, high, A[low]$ )           // base case: only one element
3  else  $mid = \lfloor (low + high)/2 \rfloor$ 
4      ( $left-low, left-high, left-sum$ ) =
            FIND-MAXIMUM-SUBARRAY( $A, low, mid$ )
5      ( $right-low, right-high, right-sum$ ) =
            FIND-MAXIMUM-SUBARRAY( $A, mid + 1, high$ )
6      ( $cross-low, cross-high, cross-sum$ ) =
            FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )
7      if  $left-sum \geq right-sum$  and  $left-sum \geq cross-sum$ 
8          return ( $left-low, left-high, left-sum$ )
9      elseif  $right-sum \geq left-sum$  and  $right-sum \geq cross-sum$ 
10         return ( $right-low, right-high, right-sum$ )
11     else return ( $cross-low, cross-high, cross-sum$ )

```



(a)



(b)

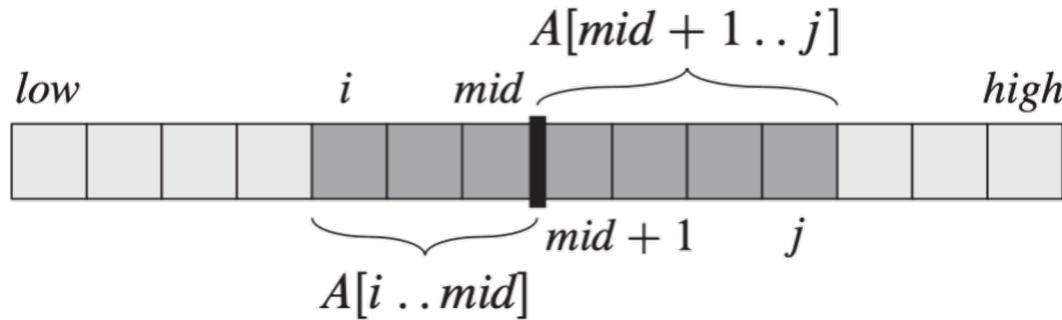
# 寻找跨越中点的最大子数组

FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )

```

1  left-sum = -∞
2  sum = 0
3  for  $i = mid$  downto  $low$ 
4      sum = sum +  $A[i]$ 
5      if sum > left-sum
6          left-sum = sum
7          max-left = i
8  right-sum = -∞
9  sum = 0
10 for  $j = mid + 1$  to  $high$ 
11     sum = sum +  $A[j]$ 
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)

```



- 核心思路：中点左侧和右侧都最大即可。

# 思考

- 利用分治方法，我们得到了一个渐近复杂性优于暴力算法的算法。
- 通过归并排序和最大子数组问题，我们对于分治方法的强大能力有了一些了解。
- 有时，对某个问题，分治方法能给出渐近最快的算法；而其他时候，我们不用分治甚至能做得更好。
  - 最大子数组问题存在线性时间的算法，见练习3-1的问题2。

# 矩阵乘法

超越平凡的复杂度

$$\begin{matrix} A \\ \left[ \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \right] \end{matrix} \times \begin{matrix} B \\ \left[ \begin{matrix} 5 & 6 \\ 7 & 8 \end{matrix} \right] \end{matrix} = \begin{matrix} \left[ \begin{matrix} 19 & 22 \\ 43 & 50 \end{matrix} \right] \\ \underbrace{\hspace{10em}}_{\text{↓}} \end{matrix}$$
$$\begin{aligned} 1 \times 6 + 2 \times 8 &= 22 \\ 1 \times 5 + 2 \times 7 &= 19 \\ 3 \times 5 + 4 \times 7 &= 43 \\ 3 \times 6 + 4 \times 8 &= 50 \end{aligned}$$

↑                      ↑  
8 multiplications

# 矩阵乘法问题的平凡算法

**输入**：两个  $n \times n$  的方阵  $A = (a_{ij})$  和  $B = (b_{ij})$ 。

**输出**：乘积矩阵  $C = A \times B$ ，其中  $c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$ 。

**SQUARE-MATRIX-MULTIPLY( $A, B$ )**

```

1   $n = A.\text{rows}$ 
2  let  $C$  be a new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 

```

- 平凡算法复杂度为  $\Theta(n^3)$ ，Strassen 利用分治改进至  $\Theta(n^{\log 7})$ 。

# 一个简单的分治算法

- 分：将每个  $n \times n$  的方针拆分成 4 个  $\frac{n}{2} \times \frac{n}{2}$  的方阵。

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- 治：递归地计算 8 次  $\frac{n}{2} \times \frac{n}{2}$  的矩阵乘法，需要  $8 \cdot T(\frac{n}{2})$ ；平凡地计算 4 次矩阵加法，需要  $\Theta(n^2)$ 。

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

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

- 合：利用下标操作即可自动完成合的部分。

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- 总代价： $T(n) = 8 \cdot T\left(\frac{n}{2}\right) + \Theta(n^2)$ ，算得  $T(n) = n^3$ 。

# 直接递归分治算法的伪代码

SQUARE-MATRIX-MULTIPLY-RECURSIVE( $A, B$ )

```

1   $n = A.\text{rows}$ 
2  let  $C$  be a new  $n \times n$  matrix
3  if  $n == 1$ 
4       $c_{11} = a_{11} \cdot b_{11}$ 
5  else partition  $A, B$ , and  $C$  as in equations above
6       $C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$ 
         +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$ 
7       $C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$ 
         +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$ 
8       $C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$ 
         +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$ 
9       $C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$ 
         +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$ 
10 return  $C$ 
```

$$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + \Theta(n^2) \Rightarrow T(n) = n^3 \quad \text{并没有优于平凡算法 !}$$

# Strassen 方法

核心思想：令递归树稍微不那么茂盛一点儿，即只递归进行 7 次，而不是 8 次  $\frac{n}{2} \times \frac{n}{2}$  的矩阵乘法。

步骤：

- 分：将每个  $n \times n$  的方针拆分成 4 个  $\frac{n}{2} \times \frac{n}{2}$  的方阵，下标计算需  $\Theta(1)$ 。
- 治：创建 10 个  $\frac{n}{2} \times \frac{n}{2}$  的矩阵  $S_1, S_2, \dots, S_{10}$ ，每个矩阵保存“分”步骤中矩阵的和或者差，需要  $\Theta(n^2)$  时间；利用  $S_1, S_2, \dots, S_{10}$  递归地计算 7 个矩阵乘积  $P_1, P_2, \dots, P_7$ ，需要  $7 \cdot T(\frac{n}{2})$  的时间。
- 合：通过  $P_1, P_2, \dots, P_7$  之间的加减运算得出  $C_{11}, C_{12}, C_{21}, C_{22}$ ，需  $\Theta(n^2)$ 。

分析： $T(n) = 7 \cdot T\left(\frac{n}{2}\right) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log 7}) = O(n^{2.81})$ 。

# Strassen 算法细节

- 计算备用的和式 -  $\Theta(n^2)$  :

$$\begin{array}{lll} S_1 = B_{12} - B_{22}, & S_4 = B_{21} - B_{11}, & S_7 = A_{12} - A_{22}, \\ S_2 = A_{11} + A_{12}, & S_5 = A_{11} + A_{22}, & S_8 = B_{21} + B_{22}, \\ S_3 = A_{21} + A_{22}, & S_6 = B_{11} + B_{22}, & S_9 = A_{11} - A_{21}, \\ & & S_{10} = B_{11} + B_{12}. \end{array}$$

- 计算备用的乘积 -  $7 \cdot T(\frac{n}{2})$  :

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}, \\ P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}, \\ P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}, \\ P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}, \\ P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}, \\ P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}, \\ P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}. \end{aligned}$$

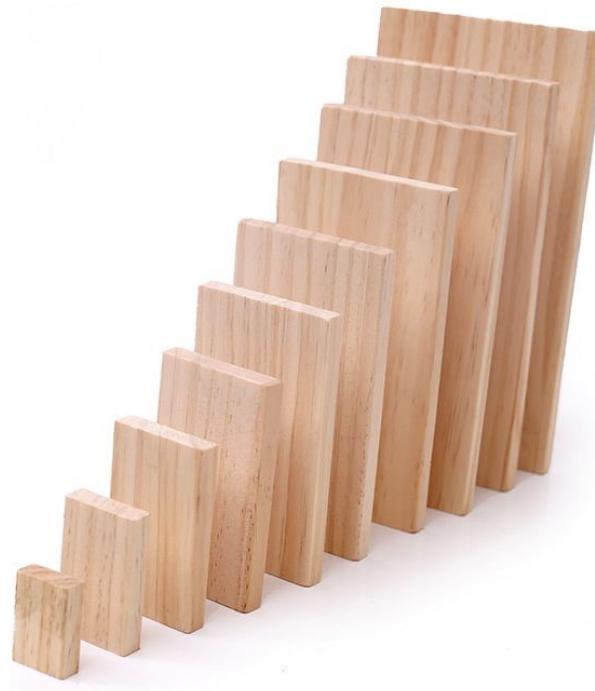
# Strassen 算法细节

- 计算结果 -  $\Theta(n^2)$ ：
    - $C_{11} = P_5 + P_4 - P_2 + P_6$

- $$\left. \begin{array}{l} \bullet C_{12} = P_1 + P_2 \\ \bullet C_{21} = P_3 + P_4 \\ \bullet C_{22} = P_5 + P_1 - P_3 - P_7 \end{array} \right\} \text{数学验算即可证得正确性}$$

# 代入法

求解递归式最严谨的方法



# 代入法

步骤：

1. 猜测解的形式。
  2. 用数学归纳法求出解中的常数，并证明解是正确的。
- 

例：确定  $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$  的上界。

- 猜测解为  $T(n) = O(n \log n)$ ，下面证明恰当选择常数  $c > 0$ ，可有  $T(n) \leq cn \log n$ 。
  - 首先假定此界对于所有的正数  $m < n$  都成立，特别是对于  $m = \left\lfloor \frac{n}{2} \right\rfloor$ ，有  $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \leq c \left\lfloor \frac{n}{2} \right\rfloor \log\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ ，**代入递归式**，有
- $$\begin{aligned} T(n) &\leq 2c \left\lfloor \frac{n}{2} \right\rfloor \log\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\leq cn \log \frac{n}{2} + n \\ &= cn \log n + (1 - c)n \\ &\leq cn \log n \end{aligned}$$
- 其中，只要  $c \geq 1$ ，最后一步就成立。
  - 边界条件只需要选取一个较小的  $n_0$  和一个合适的  $c$  即可轻松构造。

# 技巧1：好的猜测

- 没有**猜测**的通法，需要经验和创造力，不过一般可以用一些**启发式方法**来辅助猜测。
  - 如果递归式的结构类似，那么猜测一个类似的解是合理的。

例： $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + n$ ，当  $n$  充分大时， $\left\lfloor \frac{n}{2} \right\rfloor + 17$  和  $\left\lceil \frac{n}{2} \right\rceil$  差别不大，都接近  $n$  的一半，因此可以猜测  $T(n) = O(n \log n)$ 。

- 逐步调整法：先猜测一个宽松的上下界，然后逐渐收紧。

例： $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ ，先猜测  $T(n) = \Omega(n)$  和  $T(n) = O(n^2)$ ，然后  
再缩紧到  $T(n) = \Theta(n \log n)$ 。

## 技巧2：强化结论

- 归纳失败的时候，可以从先前的假设中减去一个低阶项，证明一个**更强的假设**。
  - 证明一个更强的假设，在归纳步骤的时候也就可以使用这个更强的假设了。**看似增加了证明义务，同时也增强了可以使用的归纳假设条件。**

例： $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ ， 猜测  $T(n) = O(n)$ 。

- 假设  $T(n) \leq cn$ ， 则  $T(n) \leq c \left\lfloor \frac{n}{2} \right\rfloor + c \left\lceil \frac{n}{2} \right\rceil + 1 = cn + 1 \not\leq cn$ 。
- 新的猜测：** $T(n) \leq cn - d$ ， 则

$$T(n) \leq c \left\lfloor \frac{n}{2} \right\rfloor - d + c \left\lceil \frac{n}{2} \right\rceil - d + 1 = cn - d + (1 - d) \leq cn - d$$

- 只要  $d \geq 1$ ， 最后一步就成立。

# 技巧3：换元法

- 一些小小的代数变换可以将一个不熟悉的递归式编程你熟悉的形  
式。

例： $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log n$

- 令  $m = \log n$ ，有  $T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$ ；
- 令  $S(m) = T(2^m)$ ，则  $S(m) = 2S\left(\frac{m}{2}\right) + m$ ；
- 不难发现  $S(m) = \Theta(m \log m)$ ；
- 则  $T(n) = T(2^m) = S(m) = \Theta(m \log m) = \Theta(\log n \log \log n)$ 。

} 换元

还元

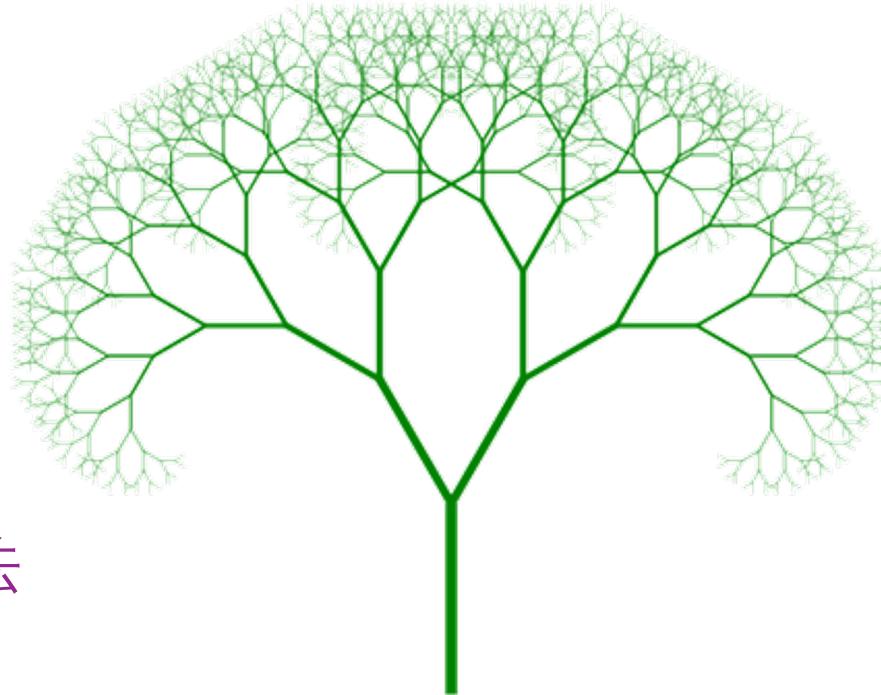
# 滥用渐近记号的陷阱

例： $T(n) = 2 T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$ ，伪证  $T(n) = O(n)$ 。

- 猜测  $T(n) \leq cn$ ，并论证  $T(n) \leq 2c \left\lfloor \frac{n}{2} \right\rfloor + n \leq cn + n = O(n)$ 。
- 错误：并未论证出与归纳假设严格一致的形式！
- 需要严格的显式论证到  $T(n) \leq cn$  才行，否则数学归纳法的“多米诺骨牌”无法正常地“排山倒海”。
- 上述伪证是对于渐近记号的滥用。

# 递归树法

求解递归式最直观的方法



# 递归树法

步骤：

1. 使用递归树猜测解的形式；

2. 使用代入法证明。

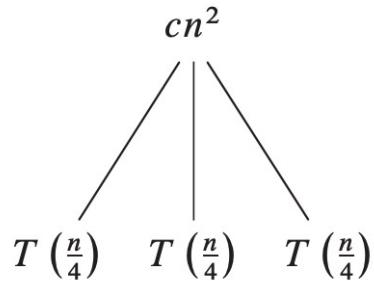
- 作为一种猜解手段的递归树可以画得不那么精确；
- 如果纯用递归树证明的话，树本身需要特别精确。
- 使用递归树法需要有一定的计数和求和功底。

# 例1：一棵较为精确的递归树

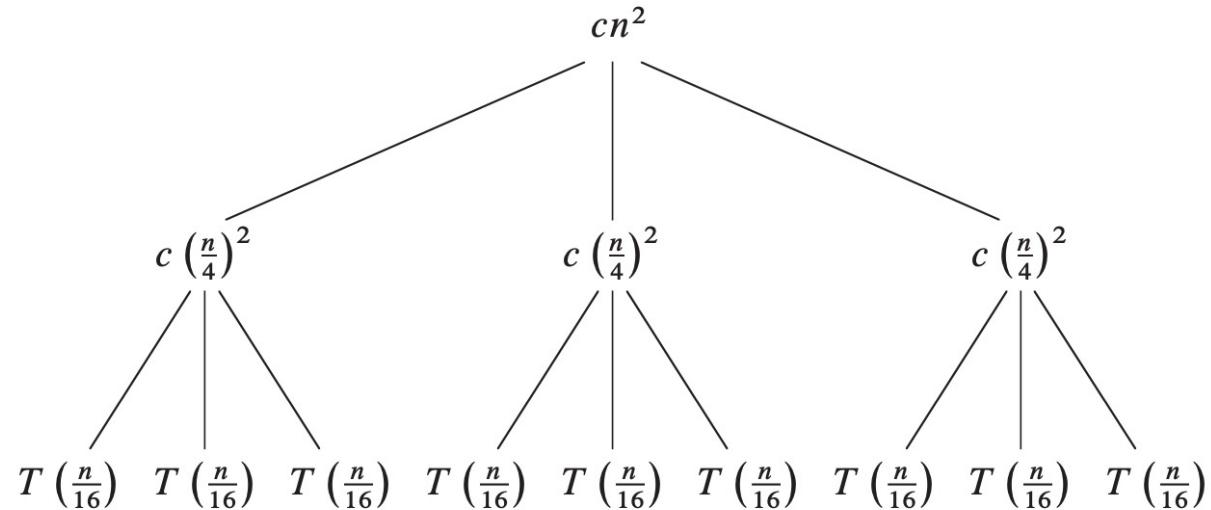
求解： $T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$ 。

- 忽略舍入，改写渐近符号，为  $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$  创建一棵递归树。

(a)



(b)

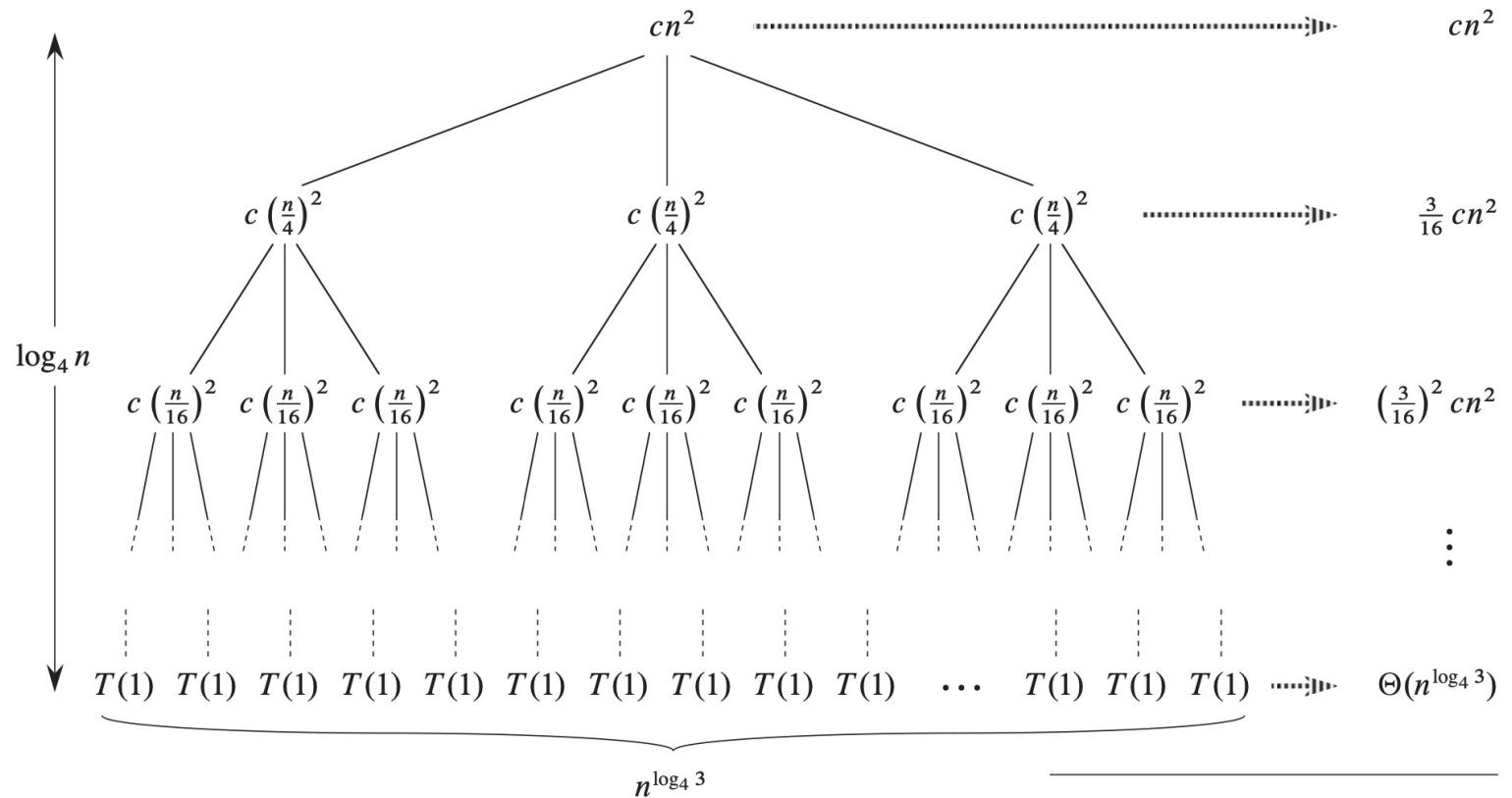


(c)

# 例1：一棵较为精确的递归树

求解： $T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$ 。

- 忽略舍入，改写渐近符号，为  $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$  创建一棵递归树。

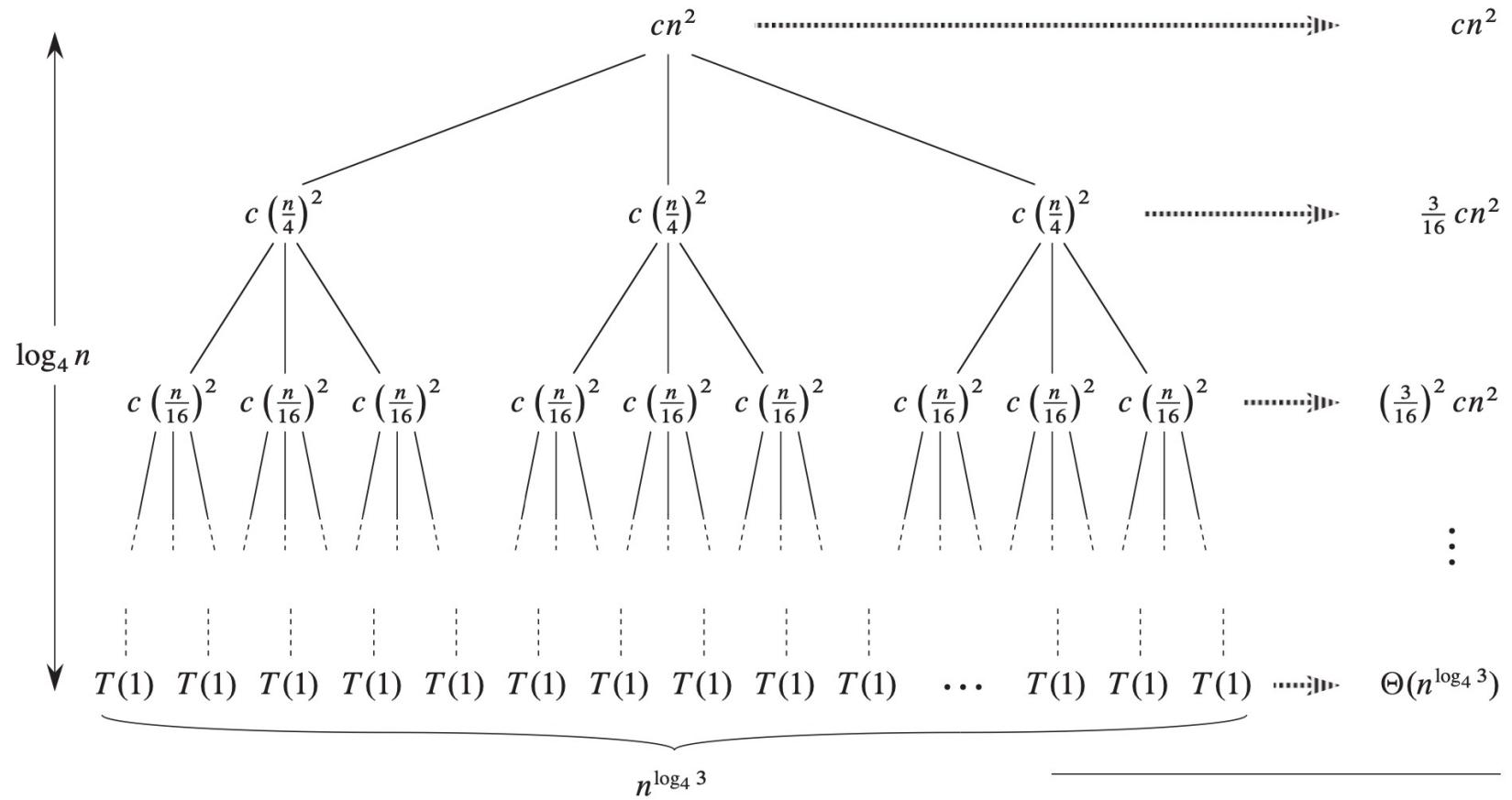


# 例1：递归树细节指标的计算

$$T(n) = 3T\left(\frac{n}{4}\right) + cn^2$$

- 深度为  $i$  (深度从 0 开始) 的结点对应规模为  $\frac{n}{4^i}$  的子问题, 当  $\frac{n}{4^i} = 1 \Leftrightarrow i = \log_4 n$  时, 问题规模为 1, 因此递归树有  $\log_4 n + 1$  层。
- 第  $i$  层有  $3^i$  个结点, 代表  $3^i$  个规模为  $\frac{n}{4^i}$  的子问题, 每个子问题划分的代价为  $c\left(\frac{n}{4^i}\right)^2$ , 因此这一层的总代价为  $\left(\frac{3}{16}\right)^i cn^2$ 。
- 树的最底层深度为  $\log_4 n$ , 共有  $3^{\log_4 n} = n^{\log_4 3}$  个叶子, 每个叶子的代价为  $T(1)$ , 因此, 总代价为  $n^{\log_4 3}T(1) = \Theta(n^{\log_4 3})$ 。

# 例1：求和每层代价得出总代价



$$T(n) = \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) = \frac{1 - \left(\frac{3}{16}\right)^{\log_4 n}}{1 - \frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) = \Theta(n^2)$$

# 例1：代入法严格证明

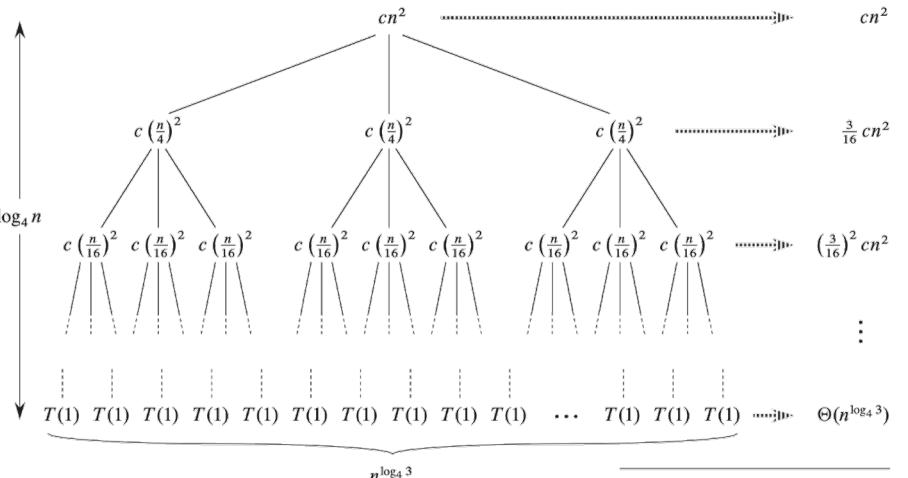
$$T(n) = 3T\left(\frac{n}{4}\right) + cn^2$$

- $T(n) \geq cn^2$  显然成立，所以  $T(n) = \Omega(n^2)$ ，下面证明  $T(n) = O(n^2)$ 。
- 假设对于  $m < n$  有  $T(m) \leq dm^2$ ，特别地，有  $T\left(\frac{n}{4}\right) \leq \frac{d}{16}n^2$ ，则

$$T(n) \leq \frac{3}{16}dn^2 + cn^2 = \left(\frac{3}{16}d + c\right)n^2 \leq dn^2$$

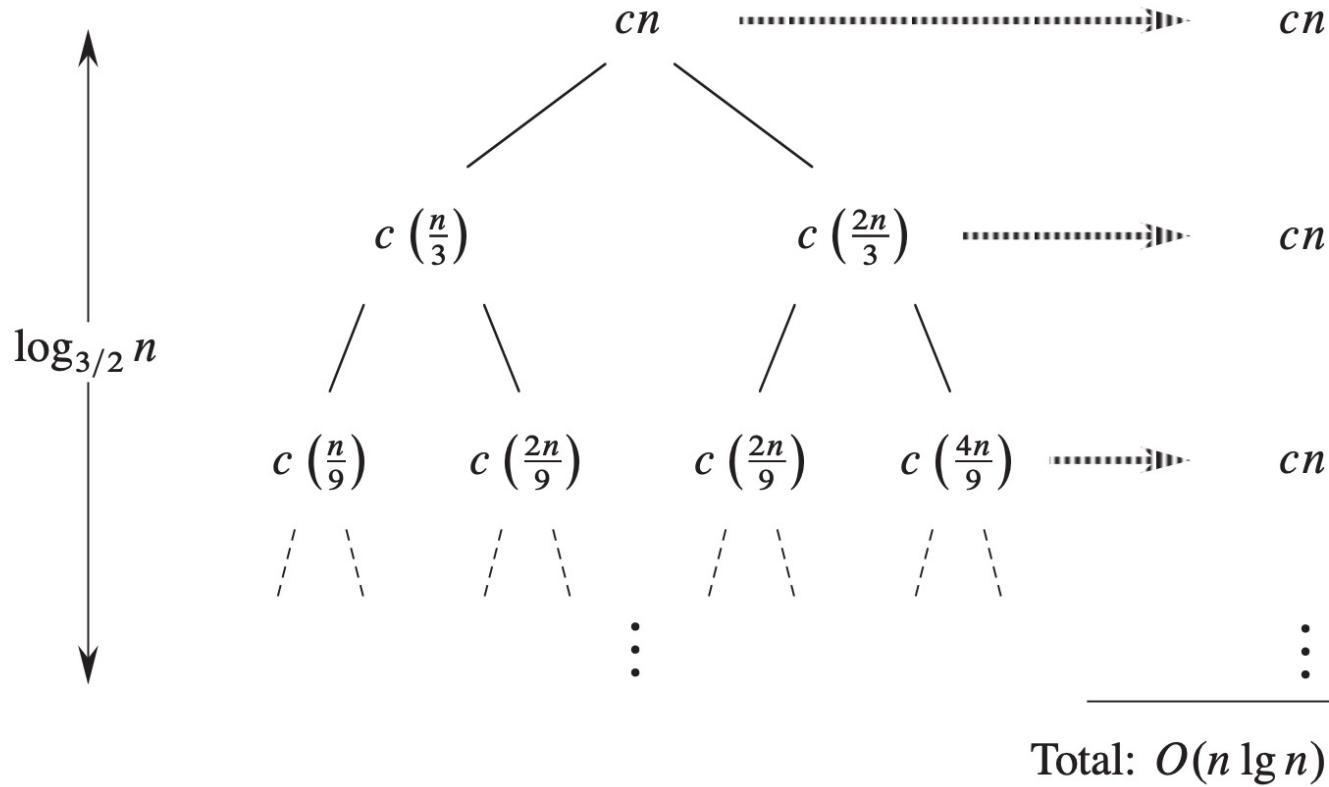
- 当  $d \geq \frac{16}{13}c$  时，最后一步推导成立。

- 于是我们严格证明了  $T(n) = \Theta(n^2)$ 。



# 例2：递归树估算

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$



# 例2：递归树估算

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

- 这个递归式的递归树并不是完全二叉树，中间缺失了许多内结点。
- 最长路径为  $n \rightarrow \frac{2}{3}n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \dots \rightarrow 1$  这一条，令  $\left(\frac{2}{3}\right)^k n = 1$  可知树的高度为  $k = \log_{\frac{3}{2}} n$ 。
- 直觉上，我们期望递归式的解最多是层数乘以每层的代价，即  $cn \log_{\frac{3}{2}} n = O(n \log n)$ ，但这是过高的估计，因为树是不完全的。
- 考虑完全二叉树，最多有  $2^k = n^{\frac{\log_3 2}{2}} = \omega(n \log n)$  个叶子，但这也是过高的估计。

# 例2：代入法严格证明

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

- 证明： $T(n) = O(n \log n)$ 。
- 假设  $T(n) \leq dn \log n$  对于小于  $n$  的数都成立，则

$$T(n) \leq d \frac{n}{3} \log \frac{n}{3} + d \frac{2n}{3} \log \frac{2n}{3} + cn = dn \log n - \left[ d \left( \log 3 - \frac{2}{3} \right) - c \right] n \leq dn \log n$$

- 只要  $d \geq \frac{c}{\log 3 - \frac{2}{3}}$ ，最后一步就成立。

# 主方法

求解递归式最简单的方法

分而治之

# 定理4.1 (主定理 Master Theorem)

令  $a \geq 1$  和  $b > 1$  是常数,  $f(n)$  是一个函数,  $T(n)$  是定义在自然数集上的递归式:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad (\text{主递归式})$$

其中, 我们将  $\frac{n}{b}$  解释为  $\left\lfloor \frac{n}{b} \right\rfloor$  或者  $\left\lceil \frac{n}{b} \right\rceil$ 。那么  $T(n)$  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  $T(n) = \Theta(n^{\log_b a} \log n)$ 。
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  $af\left(\frac{n}{b}\right) \leq cf(n)$ , 则  $T(n) = \Theta(f(n))$ 。

# 理解主定理

- 主定理为形如  $T(n) = aT\left(\frac{n}{b}\right) + f(n)$  的递归式提供了一种“菜谱”式的求解方法。
- 直觉：
  - $f(n)$  与  $n^{\log_b a}$  的**较大者**决定了递归式的解；
    - 如果  $n^{\log_b a}$  更大，如情况1，则解为  $\Theta(n^{\log_b a})$ ；
    - 如果  $f(n)$  更大，如情况3，则解为  $\Theta(f(n))$ ；
    - 如果两者大小相当，如情况2，则乘上一个对数因子，解为  $\Theta(n^{\log_b a} \log n)$ 。
  - 这里的较大指的是“**多项式意义**”上的较大。
  - 情况3还需要满足**正则条件**  $af\left(\frac{n}{b}\right) \leq cf(n)$ ，多项式函数一般都满足于这个条件。

# 使用主方法——情况1

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- 情况1：若对某个常数  $\varepsilon > 0$  有  $f(n) = O(n^{\log_b a - \varepsilon})$ ， 则  $T(n) = \Theta(n^{\log_b a})$ 。

例：Strassen 算法的递归式 -  $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$

- $\Theta(n^2) = O(n^{\log_2 7 - \varepsilon})$ ， 符合情况1，  $T(n) = n^{\log 7}$ 。

# 使用主方法——情况2

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- 情况2：若  $f(n) = \Theta(n^{\log_b a})$ ， 则  $T(n) = \Theta(n^{\log_b a} \log n)$ 。

例：最熟悉的递归式 -  $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

- $\Theta(n) = \Theta(n^{\log_2 2})$ ， 符合情况2，  $T(n) = \Theta(n \log n)$ 。

# 使用主方法——情况3

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- 情况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  $af\left(\frac{n}{b}\right) \leq cf(n)$ ，则  $T(n) = \Theta(f(n))$ 。

例： $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$ 。

- $n \log n = \Omega(n^{\log_4 3 + \varepsilon})$  且  $3f\left(\frac{n}{4}\right) = 3 \frac{n}{4} \log \frac{n}{4} \leq \frac{3}{4} n \log n = \frac{3}{4} f(n)$ ，符合情况3， $T(n) = n \log n$ 。

# 主方法补充

- 主定理的三种情况之间是有间隙的，并不是所有的主递归式都恰好落在这三种情况中。
  - 例如： $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$ 。
- 主定理情况2拓展：如果  $f(n) = \Theta(n^{\log_b a} \log^k n)$ ，其中  $k \geq 0$ ，那么主递归式的解为  $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$ 。（见练习3-2问题6）
  - 这个拓展可以解决上面的例子，得到  $T(n) = \Theta(n \log^2 n)$ ，但并没有填满主定理的空隙。
- 主定理中的情况3被过分强调了，正则条件  $af\left(\frac{n}{b}\right) \leq cf(n)$  本身就意味着  $f(n) = \Omega(n^{\log_b a+\varepsilon})$  了。（见练习3-2问题7）

# 证明主定理

对于  $b$  的幂证明主定理



# 问题化简

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- 假定  $T(n)$  仅定义在  $b$  的幂上，即对于  $n = 1, b, b^2, \dots$  定义；
- 教材 4.6.2 节有关于取整函数的详细讨论，但取整一般不影响数量级，因此这里为了简化问题，仅讨论  $b$  的幂次。
- 稍微滥用一下渐近记号；
- 渐进记号是对于  $\{1, 2, \dots, n, \dots\}$  定义的，而不是  $\{1, b, b^2, \dots, b^n, \dots\}$ ；
- 对于  $b$  的幂次证明了  $T(n) = O(n)$  并不代表  $T(n) = O(n)$ ，因为有可能：
- $T(n) = \begin{cases} n, & n = 1, 2, 4, 8, \dots \dots \\ n^2, & \text{otherwise} \end{cases}$
- 但为了方便，我们就只考虑  $\{1, b, b^2, \dots, b^n, \dots\}$  的情况。

# 证明过程：

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

证明过程分3步：

- 引理4.2：将求解主递归式的问题归约成为一个求和表达式的求值问题。
- 引理4.3：确定这个和式的界。
- 引理4.4：将引理4.2和引理4.3合二为一，证明  $n$  为  $b$  的幂的情况下主定理。

## 引理4.2 (从主递归式到和式)

令  $a \geq 1$  和  $b > 1$  是常数， $f(n)$  是一个定义在  $b$  的幂上的非负函数。 $T(n)$  是定义在  $b$  的幂上的递归式：

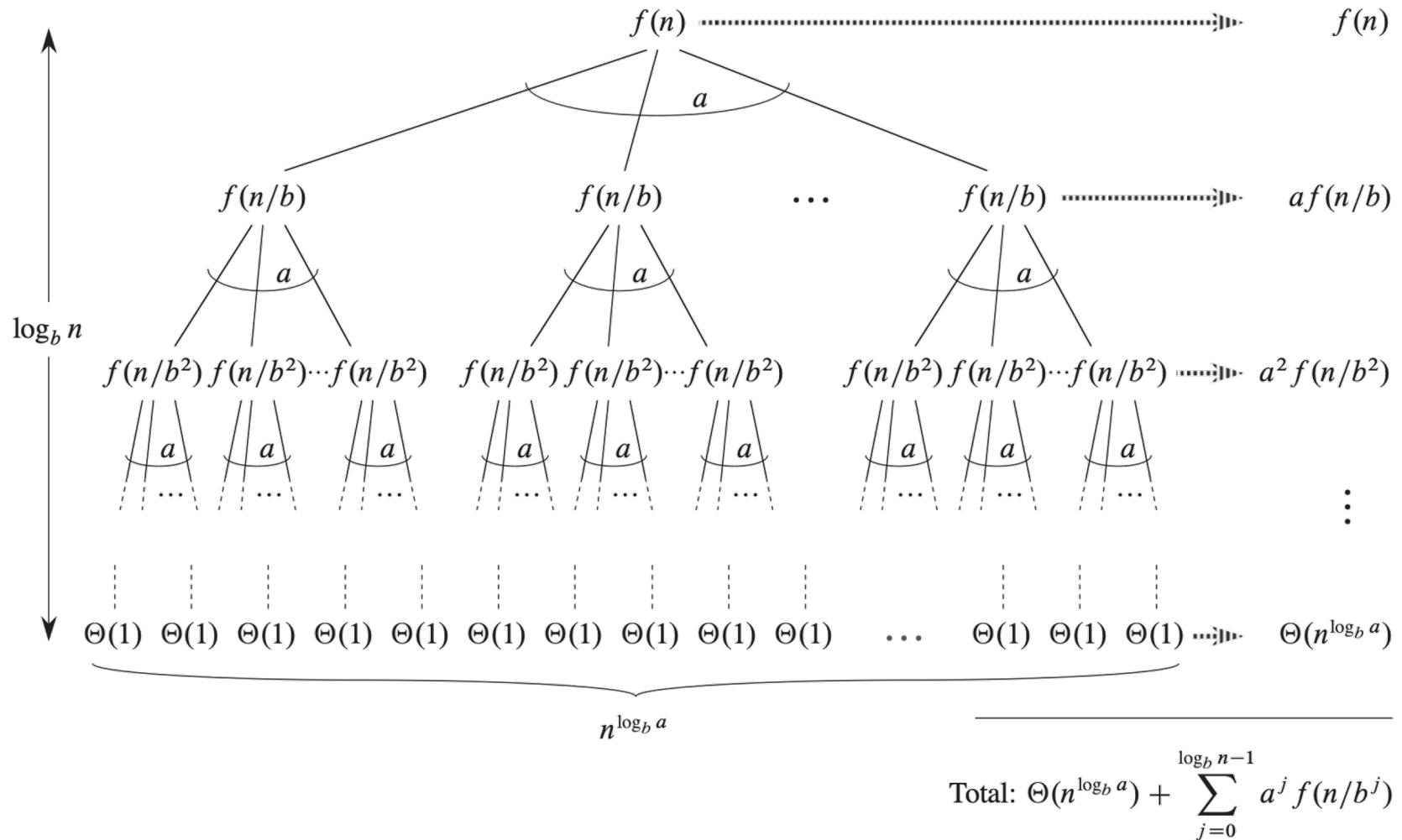
$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ aT\left(\frac{n}{b}\right) + f(n), & n = b^i \end{cases}$$

其中  $i$  是正整数。那么

$$T(n) = \Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

# 引理4.2证明

- 利用递归树将递归式转化成和式：



# 引理4.3 (确定和式的上界)

令  $a \geq 1$  和  $b \geq 1$  是常数,  $f(n)$  是一个定义在  $b$  的幂上的非负函数。 $g(n)$  是定义在  $b$  的幂上的函数:

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

对  $b$  的幂,  $g(n)$  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  $g(n) = O(n^{\log_b a})$ 。
2. 若  $f(n) = \Theta(n^{\log_b a})$ , 则  $g(n) = \Theta(n^{\log_b a} \log n)$ 。
3. 若对某个常数  $c < 1$  和所有足够大的  $n$  有  $af\left(\frac{n}{b}\right) \leq cf(n)$ , 则  $g(n) = \Theta(f(n))$ 。

# 引理4.3证明 - 情况1

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f\left(\frac{n}{b^j}\right)$$

- $f(n) = O(n^{\log_b a - \varepsilon}) \Rightarrow f\left(\frac{n}{b^j}\right) = O\left(\left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right)$ , 代入即有 :

$$\begin{aligned} g(n) &= O\left(\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right) = O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n-1} \left(\frac{ab^\varepsilon}{b^{\log_b a}}\right)^j\right) \\ &= O\left(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n-1} b^{\varepsilon j}\right) = O\left(n^{\log_b a - \varepsilon} \left(\frac{n^\varepsilon - 1}{b^\varepsilon - 1}\right)\right) = O(n^{\log_b a - \varepsilon} \cdot n^\varepsilon) \\ &= O(n^{\log_b a - \varepsilon}) \end{aligned}$$

# 引理4.3证明 – 情况2

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

- $f(n) = \Theta(n^{\log_b a}) \Rightarrow f\left(\frac{n}{b^j}\right) = \Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a}\right)$ , 代入即有 :

$$g(n) = \Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right) = \Theta\left(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^{\log_b a}}\right)^j\right)$$

$$= \Theta\left(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1\right) = \Theta(n^{\log_b a} \log_b n) = \Theta(n^{\log_b a} \log n)$$

# 引理4.3证明 – 情况3

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

- $g(n) \geq a^0 f\left(\frac{n}{b^0}\right) = f(n)$  , 显然有  $g(n) = \Omega(f(n))$  。
- 由  $n$  足够大时,  $a f\left(\frac{n}{b}\right) \leq c f(n)$  , 迭代有  $f(n) \geq \frac{a}{c} f\left(\frac{n}{b}\right) \geq \left(\frac{a}{c}\right)^2 f\left(\frac{n}{b^2}\right) \geq \dots \geq \left(\frac{a}{c}\right)^j f\left(\frac{n}{b^j}\right)$  , 即  $a^j f\left(\frac{n}{b^j}\right) \leq c^j f(n)$  , 代入有 :

$$g(n) \leq \sum_{j=0}^{\log_b n - 1} c^j f(n) + O(1) \leq f(n) \sum_{j=0}^{\infty} c^j + O(1) = \frac{f(n)}{1 - c} + O(1) = O(f(n))$$

- 综上,  $g(n) = \Theta(f(n))$  。

# 引理4.4 ( $n$ 为 $b$ 的幂情况下的主定理)

令  $a \geq 1$  和  $b > 1$  是常数,  $f(n)$  是一个定义在  $b$  的幂上的非负函数。 $T(n)$  是定义在  $b$  的幂上的递归式：

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ aT\left(\frac{n}{b}\right) + f(n), & n = b^i \end{cases}$$

其中,  $i$  是正整数。那么对  $b$  的幂,  $T(n)$  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  $T(n) = \Theta(n^{\log_b a} \log n)$ 。
3. 若对某个常数  $\varepsilon > 0$ , 有  $f(n) = \Omega(n^{\log_b a + \varepsilon})$ , 并且对某个常数  $c < 1$  和所有足够大的  $n$ , 有  $af\left(\frac{n}{b}\right) \leq cf(n)$ , 则  $T(n) = \Theta(f(n))$ 。

# 引理4.4证明

利用引理4.3中的界对引理4.2中的和式进行求值：

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

- 情况1： $T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a}) = \Theta(n^{\log_b a})$ 。
- 情况2： $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_b a} \log n)$ 。
- 情况3： $T(n) = \Theta(n^{\log_b a}) + \Theta(f(n)) = \Theta(f(n))$ 。

引理4.4处理好向下取整和向上取整就可以拓展到所有正整数。

# 拓展：Akra-Bazzi 方法

$$T(x) = \begin{cases} \Theta(1), & \text{if } 1 \leq x \leq x_0 \\ \sum_{i=1}^k a_i T(b_i x) + f(x), & \text{if } x > x_0 \end{cases}$$

其中：

- $x \geq 1$  是一个实数，
- $x_0$  是一个常数， 满足对  $i = 1, 2, \dots, k$ ，  $x_0 \geq \frac{1}{b_i}$  且  $x_0 \geq \frac{1}{1-b_i}$ ，
- 对  $i = 1, 2, \dots, k$ ，  $a_i$  是一个正常数，  $b_i$  是一个常数且  $0 < b_i < 1$ ，
- $k \geq 1$  是一个整数常数且  $f(x)$  是一个非负函数， 满足 **多项式增长条件**：存在正常数  $c_1$  和  $c_2$ ， 使得对所有  $x \geq 1$ ，  $i = 1, 2, \dots, k$  以及所有满足  $b_i x \leq u \leq x$  的  $u$ ， 有  $c_1 f(x) \leq f(u) \leq c_2 f(x)$ 。

# 拓展：Akra-Bazzi 方法

$$T(x) = \begin{cases} \Theta(1), & \text{if } 1 \leq x \leq x_0 \\ \sum_{i=1}^k a_i T(b_i x) + f(x), & \text{if } x > x_0 \end{cases}$$

- 关于多项式增长条件：若  $|f'(x)|$  的上界是  $x$  的某个多项式，则  $f(x)$  满足多项式增长条件。例如： $f(x) = x^\alpha \log^\beta x$  满足多项式增长条件。

寻找满足  $\sum_{i=1}^k a_i b_i^p = 1$  的实数  $p$ （这样的  $p$  总是存在的），那么递归式的解为：

$$T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{f(u)}{u^{p+1}} du\right)\right)$$

- 可解决一些主方法所不能解决的问题： $T(n) = T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + \Theta(n)$ 。

# 本讲小结

# 内容提要

- 复习分治 (Divide and Conquer) 算法设计
  - 分 (Divide) 、治 (Conquer) 、合 (Combine)
- 学习两个经典的分治算法
  - 最大子数组问题的分治算法
  - 矩阵乘法的 Strassen 算法
- 学习递归式 (Recurrence) 的求解方法
  - 代入法
  - 递归树法
  - 主方法
    - 证明主定理

The End!

