# 算法设计与分析

主讲人：吴庭芳

Email: *tfwu@suda.edu.cn*

苏州大学 计算机学院







# 第三讲 分治策略

## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法

# 分治法的基本思想

- **基本思想：** 当问题规模比较大而无法直接求解时，将原始问题分解为几个规模较小、但类似于原始问题的子问题，然后递归地求解这些子问题，最后合并子问题的解以得到原始问题的解
- **分治策略遵循三个步骤：**

  1）分解（Divide）：将原问题分为若干个规模较小、相互独立，形式与原问题一样的子问题

  2）解决（Conquer）：递归地解各个子问题。若子问题足够小，则直接求解；否则“递归”地求解各个子问题，即继续将较大子问题分解为更小的子问题，然后重复上述计算过程

  3）合并（Combine）：将子问题的结果合并成原问题的解

当子问题的规模足够大，需要进一步分解并递归求解时，这种情况称为递归情况 (recursive case)；若子问题的规模变得足够小，不再需要再进一步分解了，这种情况称为基本情况 (base case)（基本情况的子问题可以直接求解）

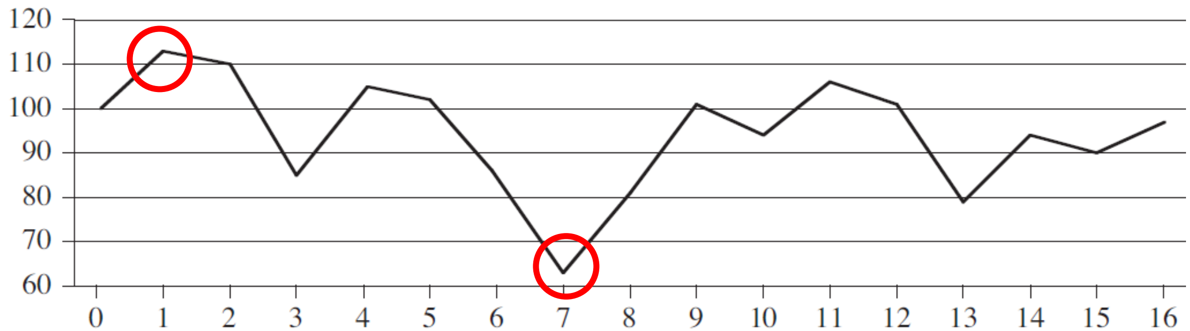
# 第三讲 分治策略

## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法

# 最大子数组问题

- 一个关于炒股的 story：




| 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 |

- 哪段时间最赚钱？即股市有起落，从哪天到哪天的收益最大呢？
- 策略1：低价买进，高价卖出
- 策略2：在最低价格时买进，或在最高价格时卖出
- 策略3：暴力求解，尝试每对可能的买进和卖出日期组合，只要卖出日期在买入日期之后，$n$ 天共有 $\binom{n}{2}=\Theta(n^2)$ 组合

# 最大子数组问题

- **从问题定义到建模求解：**
  - 目的是寻找一段时间，使得从第一天到最后一天的股票价格净变值最大。因此，不再从每日价格的角度去看待输入数据，而是考察每日价格变化：第 $i$ 天价格变化定义为第 $i$ 天和第 $i$-1 天的价格差

| 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 |
| 每日价格变化： |  | 13 | −3 | −25 | 20 | −3 | −16 | −23 | 18 | 20 | −7 | 12 | −5 | −22 | 15 | −4 | 7 |

  - 已知每日价格变化数组 $A$，在 $A$ 中寻找“和”最大的非空连续子数组。这样的连续子数组称为最大子数组
  - 求解炒股问题的算法模型：最大子数组问题

|  | 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 |

maximum subarray

# 最大子数组问题

- **最大子数组怎么求解?**
  - **方法一：****暴力求解法****（brute-force solution）**

    搜索 $A$ 的每一对起止下标区间的和，和最大的子区间就是最大子数组，时间复杂度：$\binom{n-1}{2} = \Theta(n^2)$

① 通常说“一个最大子数组”，而不是“最大子数组”，因为可能有多个子数组达到最大和

② 只有当数组中包含负数时，最大子数组问题才有意义。因为如果所有元素非负，最大子数组就是整个数组

# 最大子数组问题

- 最大子数组怎么求解？
  - 方法二：**使用分治策略的求解方法**

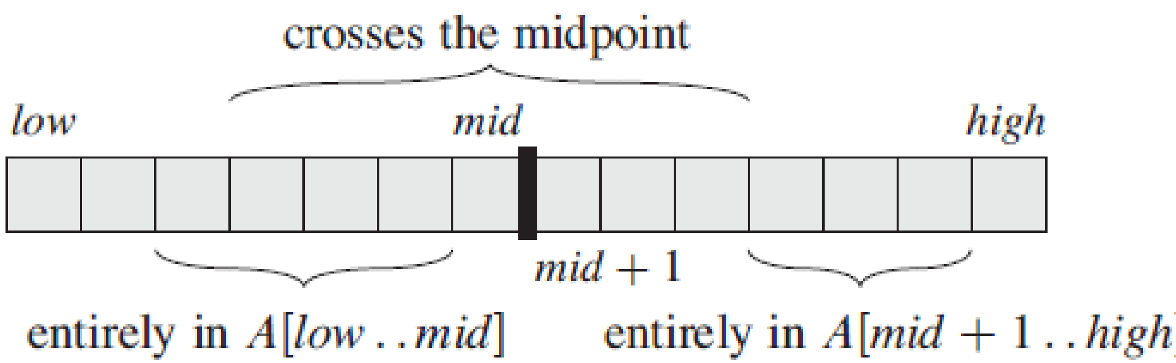
设当前要寻找数组 $A[low..high]$ 的最大子数组

**分治的基本思想是：**

- **分解**：首先将数组 $A[low..high]$ 划分为两个规模尽量相等的子数组，分割点：$mid = (low+high)/2$
- **解决**：然后分别递归求解两个子数组 $A[low..mid]$ 和 $A[mid+1..high]$ 的最大子数组

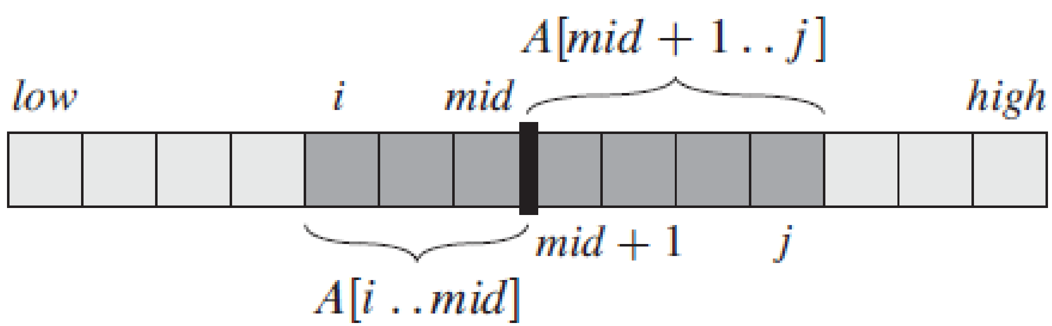
# 最大子数组问题

- **基于上述划分，$A[low..high]$ 的任何连续子数组 $A[i..j]$ 所处的位置必然是下面三种情况之一：**
  - 完全位于左子数组 $A[low..mid]$ 中，因此 $low \le i \le j \le mid$
  - 完全位于右子数组 $A[mid+1..high]$ 中，因此 $mid \le i \le j \le high$
  - 跨越了中点，因此 $low \le i \le mid < j \le high$



# 最大子数组问题

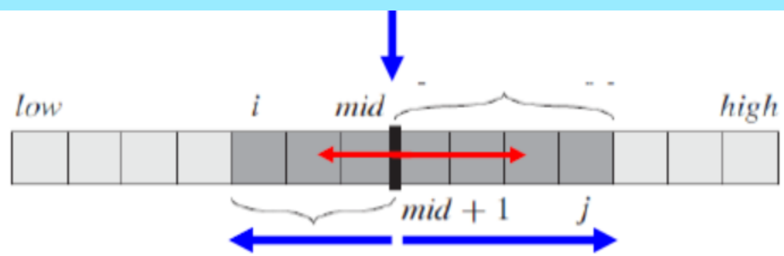
- $A[low..high]$的一个**最大子数组**也是 $A[low..high]$ 的一个连续子数组，因此 $A[low..high]$ 的一个最大子数组所处的位置必然也是上述三种情况之一
- 即：$A[low..high]$ 的一个“最大子数组”必然是：或者**完全位于$A[low..mid]$ 中**、或者**完全位于 $A[mid+1..high]$ 中**、或者是**跨越中点的所有子数组中和最大的那个**



# 最大子数组问题

- **求解过程分析：**
  - 对于完全位于$A[low..mid]$ 和 $A[mid+1..high]$ 中的最大子数组，因为这两个子问题仍是最大子数组问题，因此可以递归进行求解
  - 寻找跨越中点的最大子数组：此问题并非原问题规模更小的实例，因为加入了限制—求出的子数组必须跨越中点

这样的子数组必然跨越中点 ***mid***




**思路：**从 *mid* 出发，分别向左和向右找出和最大的子区间，然后合并这两个区间以得到跨越中点的最大子数组

# 最大子数组问题

- 以下 2 个过程用于求解最大子数组问题：
  - **过程1：**FIND-MAX-CROSSING-SUBARRAY，求跨越中点的最大子数组，将其看做是分治策略中的合并部分

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞ // 目前为止找到的最大和
sum = 0  // 保存A[i..mid]中所有值的和
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

**搜索从 *mid* 开始向左的半个区间，找出左边最大的连续子数组的和**

**同理，搜索从 *mid*+1 开始向右的半个区间，找出右边最大的连续子数组的和**

**返回搜索的结果**

  - **时间复杂度：**(*mid*-*low*+1)+(*high*-*mid*)=*high*-*low*+1=*n*

# 最大子数组问题

- **过程 2：**FIND-MAXIMUM-SUBARRAY, 求最大子数组问题的分治算法

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    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)
```

**求 *A*[*low*..*mid*] 的最大子数组**

**求 *A*[*mid*+1..*high*] 的最大子数组**

**求跨越中点的最大子数组**

**返回三个最大子数组中的最大者作为问题的解**

# 最大子数组问题

- **FIND-MAXIMUM-SUBARRAY的时间分析证明**

令 $T(n)$ 表示求解 $n$ 个元素的最大子数组问题的执行时间

1）当 $n = 1$ 时，$T(1) = \Theta(1)$

2）当 $n > 1$ 时，对 $A[low..mid]$和 $A[mid+1..high]$两个子问题递归求解，每个子问题的规模是 $n/2$，所以每个子问题的时间为 $T(n/2)$，两个子问题递归求解的总时间是 $2T(n/2)$；合并过程 FIND-MAX-CROSSING-SUBARRAY的时间是 $\Theta(n)$

3）找出三个解中的最大值的时间为 $\Theta(1)$

- **算法FIND-MAXIMUM-SUBARRAY执行时间 $T(n)$ 的递归式为：**

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







# 第三讲 分治策略

## 内容提要：

- 分治法基本思想
- 最大子数组问题
- 矩阵乘法的Strassen算法
- 递归式求解方法







# Strassen矩阵乘法

## □ 朴素的矩阵乘法:

已知两个 $n$ 阶方阵：$A=(a_{ij})_{n\times n}$， $B=(b_{ij})_{n\times n}$， 定义乘积 $C=A\cdot B$ 中的元素：

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

实现两个 $n\times n$ 矩阵相乘的过程：

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

**朴素的矩阵相乘的计算时间是 $\Theta(n^3)$**

# Strassen矩阵乘法

- **基于分治策略的矩阵乘算法:**

设 $n=2^k$，两个 $n$ 阶方阵为 $A=(a_{ij})_{n\times n}$， $B=(b_{ij})_{n\times n}$

**N:** 若 $n\neq 2^k$，可通过在 A 和 B 中补 0 使之变成阶是 2 的幂的方阵

✓ **分解**：首先，将 A、B 和 C 分解成 4 个 $n/2\times n/2$ 的子矩阵：

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

✓ **解决**：可以将公式 C=AB 作改写，然后递归求解**8次(*n*/2)×(*n*/2)矩阵相乘**

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

✓ **合并：4次 (*n*/2)×(*n*/2) 矩阵**计算结果相加

# Strassen矩阵乘法

- SQUARE-MATRIX-MULTIPLY-RECURSIVE，矩阵乘法的分治算法

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

**复制矩阵，花费$\Theta(n^2)$**

**使用下标，花费$\Theta(1)$**

**$\Theta(n^2)$**

**$T(n/2)$**

**Note:** 该算法隐藏了一个重要的细节，即如何划分矩阵的问题；常规做法是新建几个新的矩阵，然后在从原矩阵特定位置赋值过来；更高效的做法是利用下标来进行划分和计算

# Strassen矩阵乘法

- **利用下标来划分矩阵：**用表示矩形的方式表示矩阵，即表示出来矩阵的左、右、上、下位置保存到位置数组中

```
void divide_array(int Array[][n], int flag[], int one[], int two[], int three[],int four[])
{
    int left=flag[0];              //矩阵列最小下标
    int right=flag[1];             //矩阵列最大下标
    int top=flag[2];               //矩阵行最小下标
    int bottom=flag[3];            //矩阵行最大下标
   //
    one[0]=left;                           three[0]=left;
    one[1]=(left+right-1)/2;               three[1]=(left+right+1)/2;
    one[2]=top;                            three[2]=(top+bottom+1)/2;
    one[3]=(top+bottom-1)/2;               three[3]=bottom;
   //
    two[0]=(left+right+1)/2;               four[0]=(left+right+1)/2;
    two[1]=right;                          four[1]=right;
    two[2]=top;                            four[2]=(top+bottom+1)/2;
    two[3]=(top+bottom-1)/2;               four[3]=bottom;
}
```







# Strassen矩阵乘法

- **SQUARE-MATRIX-MULTIPLY-RECURSIVE的时间分析证明**

  令 $T(n)$ 表示两个 $n \times n$ 矩阵相乘的计算时间

  1）当 $n = 1$ 时，$T(1) = \Theta(1)$

  2）当 $n > 1$ 时，8次 $n/2 \times n/2$ 矩阵相乘，花费时间为$8T(n/2)$

  4次 $n/2 \times n/2$ 矩阵相加，花费 $\Theta(n^2)$ 时间

- **算法SQUARE-MATRIX-MULTIPLY-RECURSIVE执行时间 $T(n)$ 的递归式为**：

$$T(n) = \begin{cases} \Theta(1) & \text{if } \; n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } \; n > 1 \end{cases} \Longrightarrow T(n) = \Theta(n^3)$$

# Strassen矩阵乘法

- **Strassen方法核心思想:**
  - 令递归树稍微不那么茂盛，即只递归计算 7 次而不是 8次 $n/2\times n/2$ 矩阵相乘
- **Strassen方法具体步骤:**

  1）按上述方法将输入矩阵 A、B 和输出矩阵 C 分解为 $n/2\times n/2$ 的子矩阵。采用下标计算方法，此步骤花费 $\Theta(1)$ 时间

  2）创建10个 $n/2\times n/2$ 的矩阵 $S_1$，$S_2$，…，$S_{10}$，每个矩阵保存步骤 1) 中创建的两个矩阵的和或差，花费时间为 $\Theta(n^2)$

  3）用步骤 1）中创建的子矩阵和步骤 2）中创建的10个矩阵，进行 7 次递归计算矩阵 $P_1$，$P_2$，…，$P_7$，每个矩阵 $P_i$ 都是 $n/2\times n/2$

  4）通过 $P_i$ 矩阵的不同组合进行加减运算，计算出结果矩阵 C 的子矩阵$C_{11}$，$C_{12}$，$C_{21}$，$C_{22}$，花费时间为 $\Theta(n^2)$

# Strassen矩阵乘法







- **Strassen方法细节：**

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

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

**7次(n/2)×(n/2) 矩阵乘法**

**10次(n/2)×(n/2) 矩阵加减法**

$$
\begin{aligned}
C_{11} &= P5 + P4 - P2 + P6 \\
C_{12} &= P1 + P2 \\
C_{21} &= P3 + P4 \\
C_{22} &= P5 + P1 - P3 - P7
\end{aligned}
$$

**8次(n/2)×(n/2) 矩阵加减法**

# Strassen矩阵乘法

- **Strassen方法的时间分析证明**

令 T($n$) 表示两个 $n\times n$ 矩阵的Strassen矩阵乘所需计算时间

1）当 $n = 1$ 时，$T(1) = \Theta(1)$

2）当 $n > 1$ 时，7次 $(n/2)\times(n/2)$ 矩阵相乘，花费时间为 $7T(n/2)$，18次$(n/2)\times(n/2)$ 矩阵加减，花费 $\Theta(n^2)$ 时间

- **Strassen方法执行时间 T($n$) 的递归式为：**

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases} \quad\Rightarrow\quad T(n) = \Theta(n^{\lg 7}) \approx \Theta(n^{2.81})$$







# 第三讲 分治策略

## 内容提要:

- ◻ 分治法基本思想
- ◻ 最大子数组问题
- ◻ 矩阵乘法的Strassen算法
- ◻ 递归式求解方法
  - ✓ 代入法
  - ✓ 递归树法
  - ✓ 主方法

# 递归式求解方法

- **基于分治策略设计的算法计算时间表达式往通常是递归式**
  - ✓ 如归并排序、最大子数组的分治算法运行时间 T(*n*) 的递归式为：

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

边界条件

递归方程

  - ✓ Strassen 矩阵乘法的运行时间 T(*n*) 的递归式为：

$$T(n)=\begin{cases}\Theta(1) & \text{if } n=1 \\ 7T(n/2)+\Theta(n^2) & \text{if } n>1\end{cases}$$

- **那么如何化简递归式，以得到形式简单的限界函数?**

① 代入法

② 递归树法

③ 主方法

**N：** 递归式求解的结果是得到形式简单的渐近限界函数表示（即用 O、Ω、Θ表示的函数式）

# 递归式求解方法

- **预处理——对表达式细节进行简化**

为便于处理，通常对表达式做如下假设和简化处理：

（1）算法的运行时间函数 $\mathrm{T}(n)$ 定义中，一般假定自变量为正整数，因为 $n$ 通常表示输入规模大小

（2）忽略递归式的边界条件，即 $n$ 较小时函数值的表示；原因在于：虽然递归式的解会随着边界值的改变而改变，但此改变不会超过常数因子，对函数的增长阶没有根本影响

（3）对上取整、下取整运算做合理简化，例如：

$$\mathrm{T}(n) = \mathrm{T}(\lceil n/2 \rceil) + \mathrm{T}(\lfloor n/2 \rfloor) + f(n)$$

通常忽略上、下取整函数，就可写作以下简单形式：

$$\mathrm{T}(n) = 2\mathrm{T}(n/2) + f(n)$$

# 代入法

- **代入法求解递归式要点：**

1. 猜测解的形式
2. 用数学归纳法求出解中的常数，并证明猜测解是正确的

例：$T(n)=2T(\lfloor n/2 \rfloor)+n$

解：（1）猜测上式的解为 $T(n)=O(n\lg n)$

（2）代入法要求证明恰当选择常数 $c > 0$，有 $T(n) \le cn\lg n$

假定此上界对所有正数 $m \le n$ 都成立，特别是对于 $m = \lfloor n/2 \rfloor$ 处成立，有 $T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$ 成立（将归纳假设应用于较小值）。将猜测的解代入递归式函数，得到：

$$T(n) \le 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor))+n \le cn\lg(n/2)+n$$
$$= cn\lg n-cn\lg 2+n$$
$$= cn\lg n-cn+n \le cn\lg n$$

其中，只要 $c \ge 1$，最后一步都会成立

# 代入法

- **应用数学归纳法要求归纳假设对边界条件成立：**
  - 一般通过证明边界条件符合归纳证明的基本情况。但可能出现归纳证明基本情况不能满足的问题
  - 假设 $T(1) = 1$ 是递归式 $T(n) = 2T(\lfloor n/2 \rfloor)+n$ 的边界条件，但对于 $n = 1$，归纳证明的基本情况 $T(1) = 1 \le c \times 1 \times \lg 1 = 0$ 不能成立
  - 解决办法：扩展边界条件。因为渐近符号只要求对 $n \ge n_0$，证明 $T(n) \le cn\lg n$，故可以选择适当的 $n_0$，让 $T(n_0)$ 代替作为边界条件
  - 因此，选择 $T(2)$ 和 $T(3)$ 作为边界条件，则 $n_0 = 2$，其中 $T(2) = 4$，$T(3) = 5$
  - 取 $c \ge 2$，对于 $n = 2$，归纳证明的基本情况成立：$T(2) \le c \times 2 \times \lg 2$，$T(3) \le c \times 3 \times \lg 3$，扩展的边界条件符合归纳证明的基本情况







# 代入法

- **代入法求解步骤小结：**
  - 猜测递归式的正确解没有通用的方法
  - 需要一些经验和创造力
  - 启发式方法：递归树
  - 比如要求解的递归式与曾经见过的递归式类似，那么猜测一个类似的解是合理的，例如如下递归式：

$$T(n)=2T(\lfloor n/2 \rfloor+17)+n$$

  - 另外做出好的猜测的方法是先证明递归式较松的上界和下界，然后缩小不确定的范围，例如对于上述递归式，先从下界 $T(n)=\Omega(n)$ 和上界 $T(n)=O(n^2)$ 开始，然后逐渐降低上界，提升下界，直至收敛到渐近紧确界

# 递归树法

- 画出递归树有助于猜测递归式的解
- 递归树中每一个结点代表**一个单一子问题的代价，子问题对应某次递归函数调用**。将树中每一层的代价相加得到每层代价，再将所有层的代价相加得到所有层次的递归调用的总代价
- 但递归树法不够严谨，使用递归树产生好的猜测时，通常需要容忍小量的“不精确”：
  - ✓ floor， ceiling 忽略
  - ✓ $n$ 经常假设为某个整数的幂次方
- 但如果在画递归树和代价求和时非常仔细，也可以用递归树直接证明解的正确性
- 通常作法：用递归树法生成好的猜测解，然后即可用代入法验证猜测解是否正确

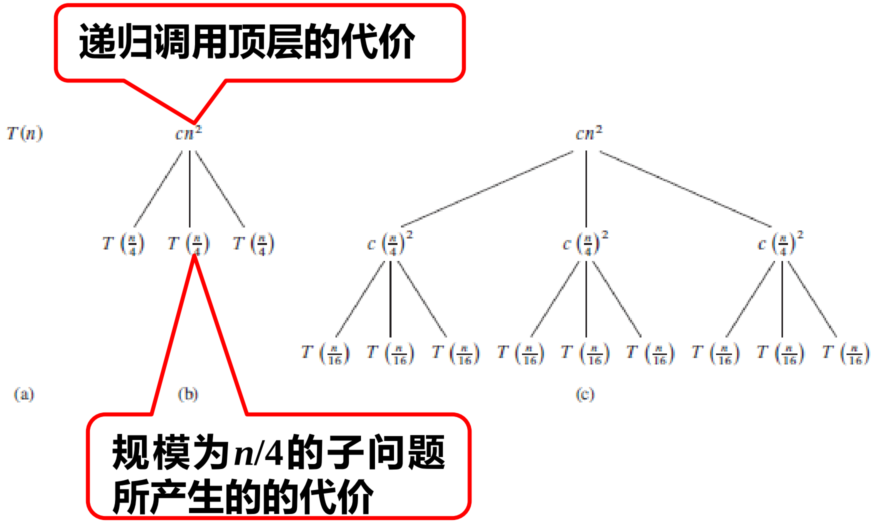
# 递归树法

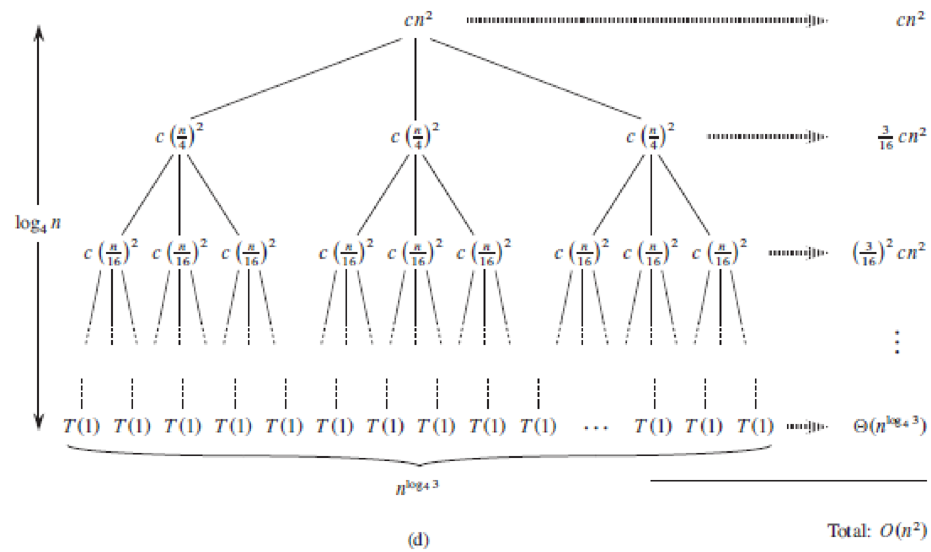
- **例：求解 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$**

  为了方便，假定 $n$ 是 4 的幂次方，从而导致不精确

  **关键：** 1）完全三叉树的深度如何确定？

  2）每个结点的代价？ —> 每层的代价？ —> 总代价？

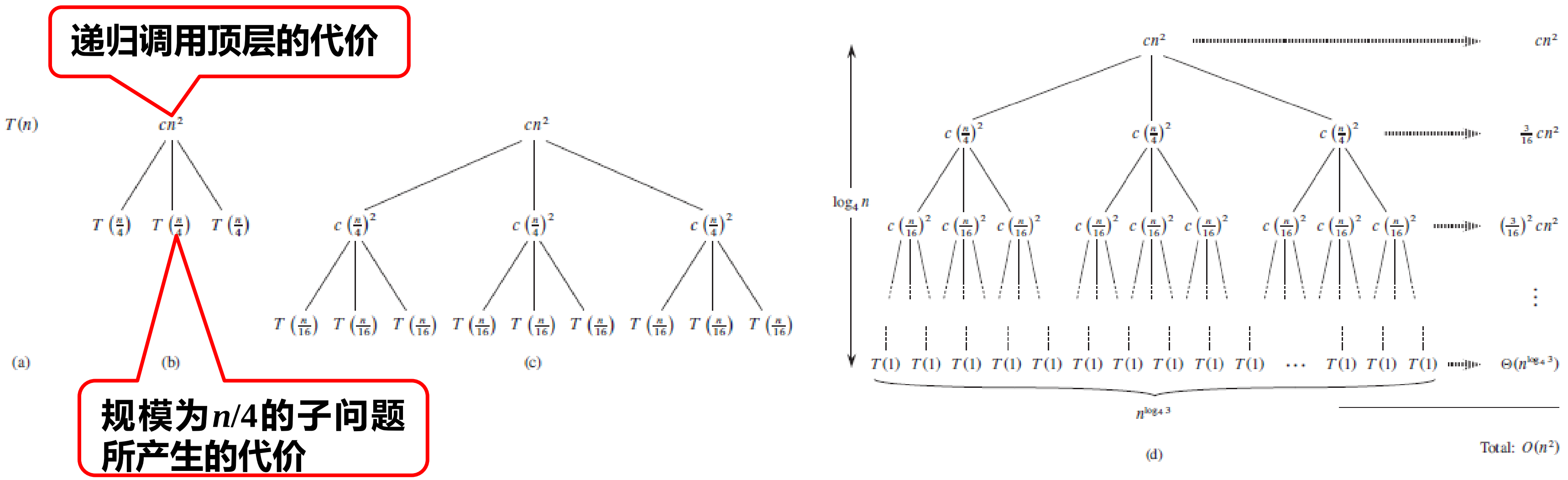



- 递归树的深度为: $\log_4 n+1$ 层
- 深度为 $i$ 的每个结点的代价为：$c(n/4^i)^2$，$i = 0，1，2，…，\log_4 n-1$
- 深度为 $i$ 的每层结点数为：$3^i$

# 递归树法

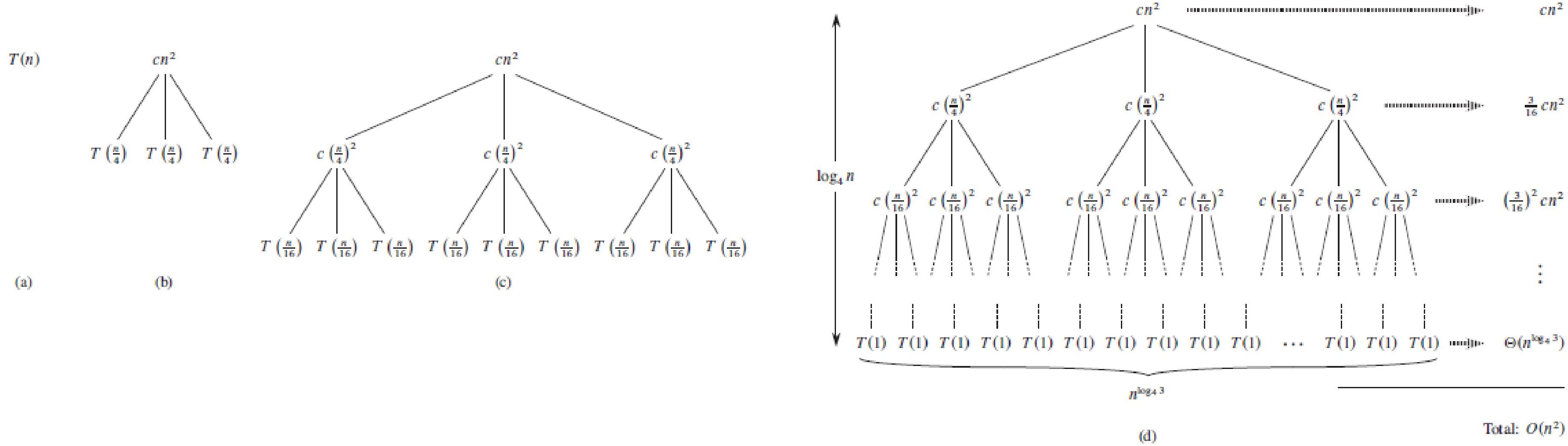
- 例：求解 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$




- 对 $i = 0, 1, 2, \ldots, \log_4 n-1$，深度为 $i$ 层的所有结点的总代价为：$3^i c(n/4^i)^2=(3/16)^i cn^2$
- 递归树的最底层深度为 $\log_4 n$，结点数目为 $3^{\log_4 n} = n^{\log_4 3}$，每个结点代价为 $T(1)$，总代价为 $n^{\log_4 3}T(1)$，即 $\Theta(n^{\log_4 3})$

# 递归树法

- 例：求解 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$




- 递归树中所有层的代价之和，即整棵树的代价:

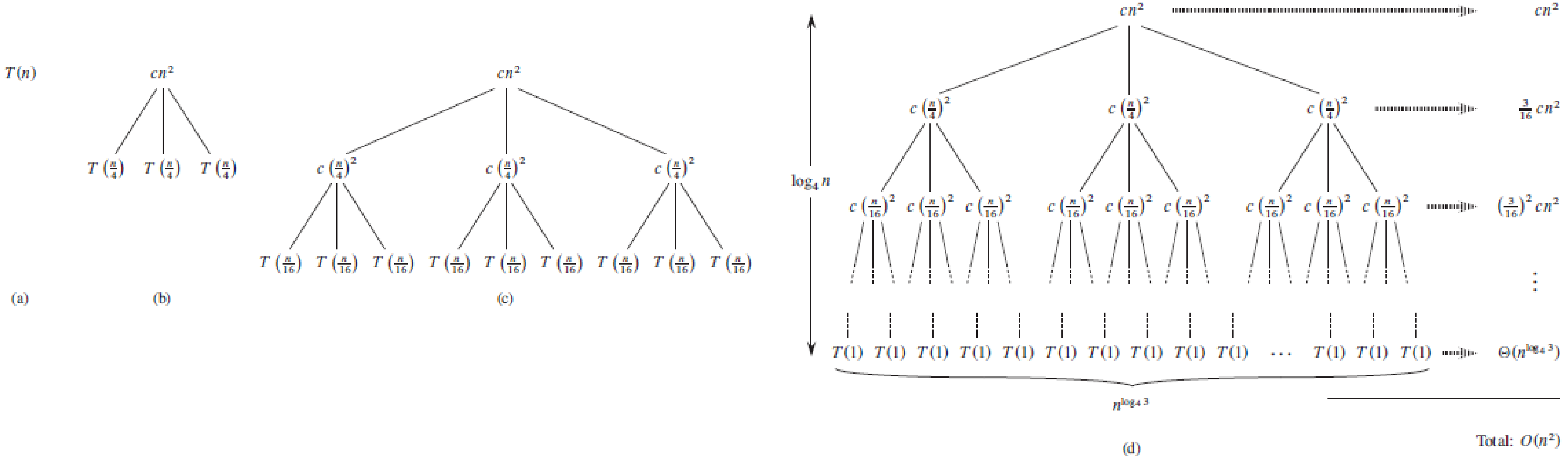
$$T(n) = cn^2 + \frac{3}{16}cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

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

$$= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3})$$

# 递归树法

- 例：求解 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$



- 以无限递减几何级数作为上界，得到:

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

$$= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= \mathrm{O}(n^2)$$

# 递归树法

- 例：求解 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$
  - 接下来，用代入法严格证明递归树法猜测的结果：
  - 猜测：$T(n) = O(n^2)$，即证明存在常数 $d > 0$，有 $T(n) \le dn^2$：

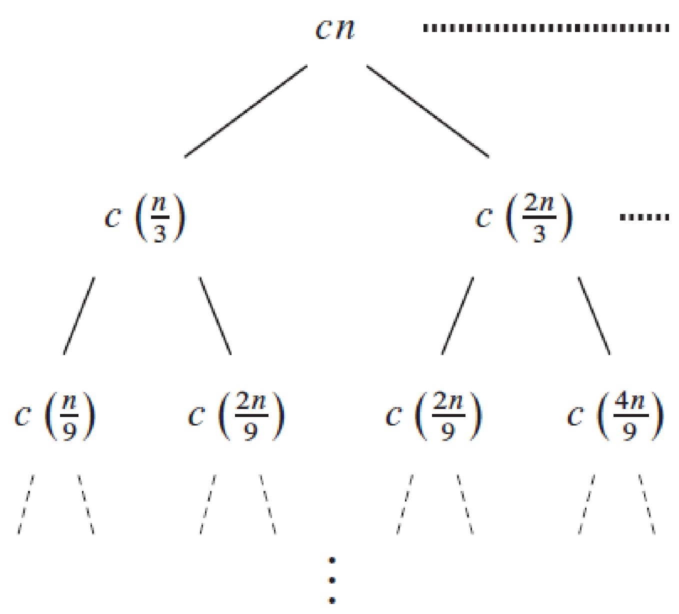
$$\begin{aligned} T(n) &= 3T(n/4) + \Theta(n^2) \\ &\le 3d(n/4)^2 + cn^2 \\ &= (\frac{3}{16}d + c)n^2 \\ &\le dn^2 \end{aligned}$$

，其中 $d \ge \frac{16}{13}c$。$T(n) = O(n^2)$ 得证

  - 此外，由于第一次递归调用的代价就是 $\Theta(n^2)$，因此 $\Omega(n^2)$ 必然是递归式的一个下界

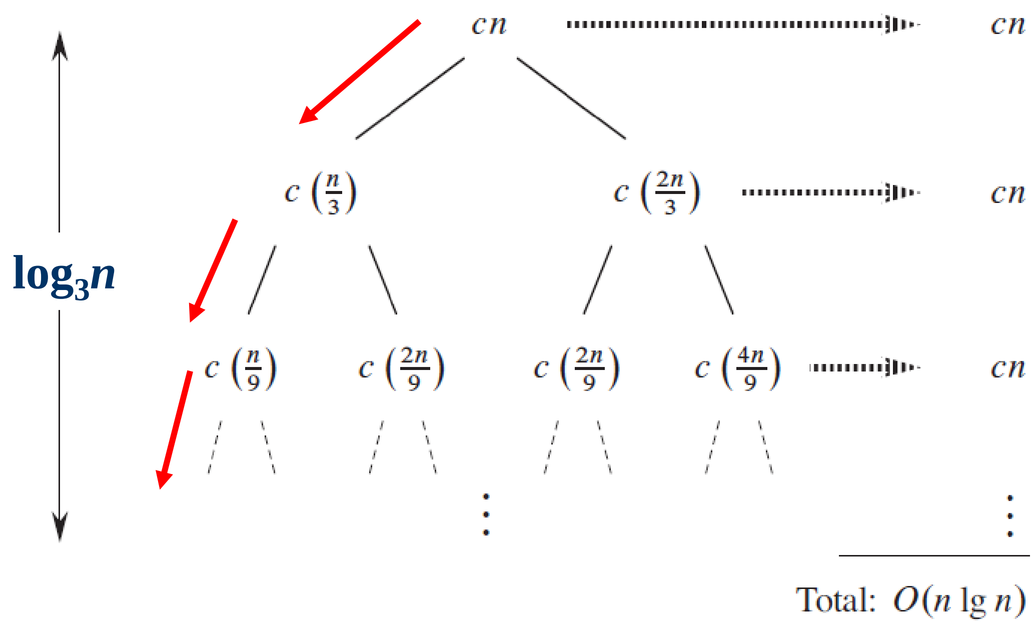
# 递归树法

- 例：求解 $T(n) = T(n/3) + T(2n/3) + O(n)$
  - 这棵递归树是不平衡的，很明显一个父结点的两个子结点的代价不一样，左子结点的输入规模是父结点的 1/3，右子结点的输入规模是父结点的 2/3，因此从根结点到不同的叶结点的路径长度也不一样



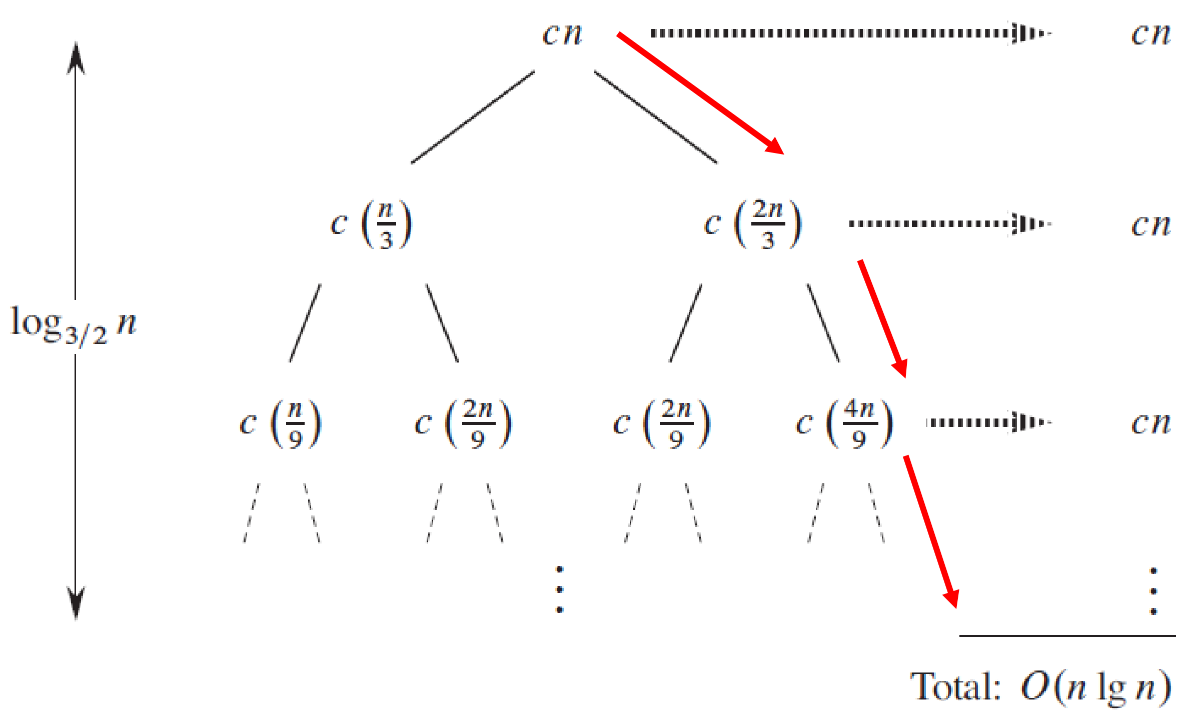
# 递归树法

- **例：求解 T($n$) = T($n$/3) + T(2$n$/3) + O($n$)**
  - 令 $c$ 表示 O($n$) 项中的常数因子
  - 最左边的分枝为从根结点到叶结点的路径，在所有从根结点到叶结点的路径中最短，对应子问题输入规模为 $n \to (1/3)n \to (1/3)^2 n \to \cdots \to 1$，对应子问题代价为 $cn \to (1/3)cn \to (1/3)^2 cn \to \cdots \to T(1)$



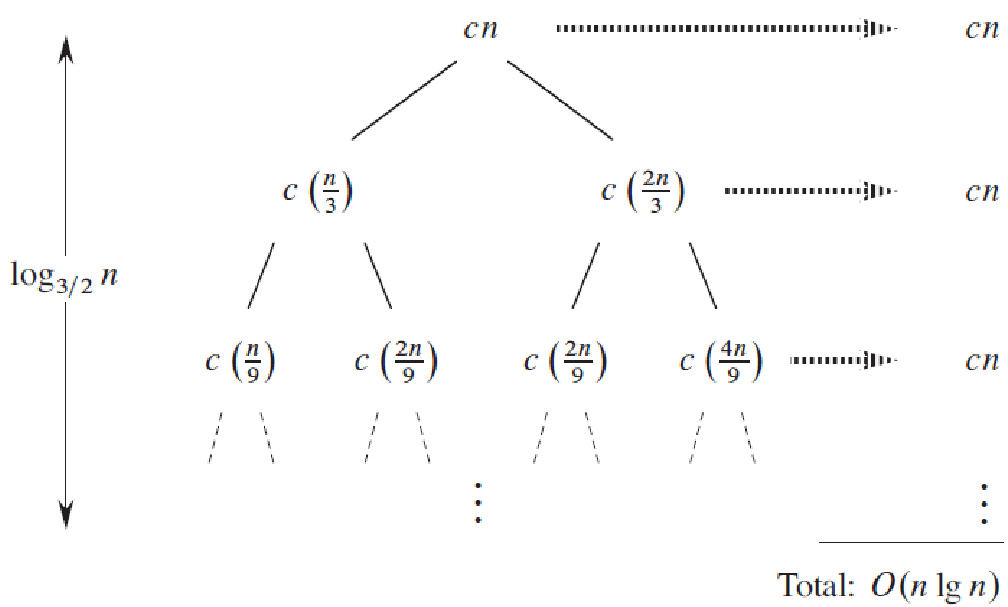
# 递归树法

- **例：求解 $T(n) = T(n/3) + T(2n/3) + O(n)$**
  - 最右边的分枝为从根结点到叶结点的路径，在所有从根结点到叶结点的路径中最长，对应子问题输入规模为 $n \to (2/3)n \to (2/3)^2 n \to \cdots \to 1$ ，对应子问题代价为 $cn \to (2/3)cn \to (2/3)^2 cn \to \cdots \to T(1)$



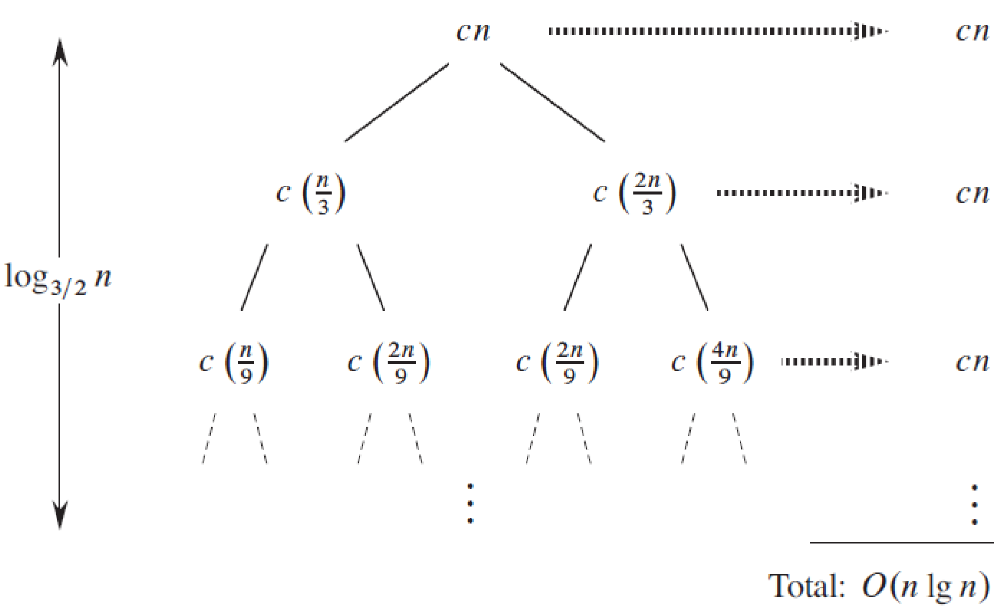
# 递归树法

- 例：求解 **T($n$) = T($n$/3) + T(2$n$/3) + O($n$)**
  - 最右边分枝最长，设递归树高 $h$，$(2/3)^h n=1$，$h=\log_{3/2}n$，因此树高为 $\log_{3/2}n$
  - 期望递归式的解最多是层数乘以每层的代价，即 $O(cn\log_{3/2}n) = O(n\lg n)$



# 递归树法

- 例：求解 T($n$) = T($n$/3) + T(2$n$/3) + O($n$)




猜测 $T(n) = O(n \lg n)$，则

$$
\begin{aligned}
T(n) &= T(n/3) + T(2n/3) + \Theta(n) \\
&\le d\frac{n}{3}\lg\frac{n}{3} + d\frac{2n}{3}\lg\frac{2n}{3} + cn \\
&= dn\lg n + (c - (\lg 3 - \frac{2}{3})d)n \\
&\le dn\lg n
\end{aligned}
$$

，其中 $d \ge \frac{c}{\lg 3 - \frac{2}{3}}$。

# 主方法

- 主方法是用来求解如下形式的递归式：

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

其中 $a \geq 1$ 和 $b > 1$ 是常数，$f(n)$ 是一个渐近正的函数

- 主方法主要针对三种情况，但这样很容易确定许多递归式的解，甚至可以不需要计算
- 上面的递归式不是良好定义的，因为 $n/b$ 可能不是整数，可以用 $\lceil n/b \rceil$ 和 $\lfloor n/b \rfloor$ 来代替 $n/b$，而且这种代替不会对递归式的渐近行为产生影响
- 实际上，在分析分治算法的运行时间时，经常略去下取整和上取整函数，以方便对递归式的分析

# 主方法

- **主方法的正确性依赖于如下的主定理**

**定理 4.1（主定理）：** 假设 $a \geq 1$ 和 $b > 1$ 为常数，$f(n)$为一函数，$T(n)$ 是定义在非负整数上的递归式：$T(n)=aT(n/b)+f(n)$， 其中将 $n/b$ 解释为 $\lceil n/b \rceil$ 和 $\lfloor n/b \rfloor$。那么 $T(n)$ 可能有如下的渐近界：

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 $T(n) = \Theta(f(n))$

# 主方法

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

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a}\lg n) & \text{if } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ and } af(n/b) \leq cf(n) \end{cases} \quad \exists \varepsilon > 0,\ c < 1$$

- **将函数 $f(n)$ 与函数 $n^{\log_b a}$ 进行比较，两个函数较大者决定了递归式的解：**
  - 若函数 $n^{\log_b a}$ 更大，如情况 1，则 $T(n) = \Theta(n^{\log_b a})$
  - 若函数 $f(n)$ 更大，如情况 3，则 $T(n) = \Theta(f(n))$
  - 若两个函数大小相当，如情况 2，则 $T(n) = \Theta(n^{\log_b a}\lg n)$
- **使用主方法需要注意的细节：**
  - 情况 1，$f(n)$ 要多项式意义上小于 $n^{\log_b a}$
  - 情况 3，$f(n)$ 要多项式意义上大于 $n^{\log_b a}$，且满足 $af(n/b) \leq cf(n)$（正则条件）

# 主方法







## 使用主方法举例

**例1：** $T(n)=9T(n/3)+n$

**解：** $a=9$，$b=3$，$f(n)=n$，因此 $n^{\log_b a}=n^{\log_3 9}=n^2$

由于 $f(n)=O(n^{\log_3 9-\varepsilon})$, where $\varepsilon=1 \Rightarrow T(n)=\Theta(n^2)$

**例2：** $T(n)=T(2n/3)+1$

**解：** $a=1$，$b=3/2$，$f(n)=1 \Rightarrow n^{\log_b a}=n^{\log_{3/2} 1}=n^0=1$

$\Rightarrow f(n)=\Theta(n^{\log_b a})=\Theta(1) \Rightarrow T(n)=\Theta(\lg n)$

**例3：** $T(n)=3T(n/4)+n\lg n$

**解：** $a=3$，$b=4$，$f(n)=n\lg n \Rightarrow n^{\log_b a}=n^{\log_4 3}=O(n^{0.793})$

$\Rightarrow f(n)=\Omega(n^{\log_4 3+\varepsilon})$, where $\varepsilon \approx 0.2$,

$af(n/b)=3(n/4)\lg(n/4) \leqslant (3/4)n\lg n=cf(n)$ for $c=3/4$

$\Rightarrow T(n)=\Theta(n\lg n)$

# 主方法

## □ 使用主方法举例

**例4：T($n$)=2T($n$/2)+$n$lg$n$**

**解：**$a=2$，$b=2$，$f(n)=n\lg n \Rightarrow n^{\log_b a}=n$

上述情况可能错误的应用情况 3，因为 $f(n)=n\lg n$ 渐近大于 $n^{\log_b a}=n$。但是它并不是多项式意义上的大于，对任意正常数 ε，比值 $f(n)/n^{\log_b a}=(n\lg n)/n=\lg n$ 都渐近小于 $n^{\varepsilon}$。因此，递归式落入了情况 2 和情况 3 的间隙







# 谢谢！

## Q & A

**作业：4.2-1**

**4.3-6**

**4.4-4**

**4.5-3**