算法设计与分析

第二章 算法分析的数学基础

哈尔滨工业大学（深圳）
李穆

本讲内容

2.1 复杂性函数阶的计算

2.2 和式的估计与界限

2.3 递归方程

如何描述算法的效率

- 记录算法实现的程序在机器上实际运行的时间？
  - 实现代码的语言的效率差别很大
  - 代码的优化程度
  - 机器的运算速度，指令集
  - 。。。

对比不同算法的实际运行时间非常困难

增长的阶

- 算法增长的阶也称为增长率，增长量级
- 一个算法比另一个算法“效率高”，如果它的运算时间随着输入规模的增长比另一个算法增长的慢
- 比方说我们用$\Theta(n^2)$表示插入排序的最坏运行时间，不是说他的运行时间是n^2，而是增长率和n^2相同

增长函数

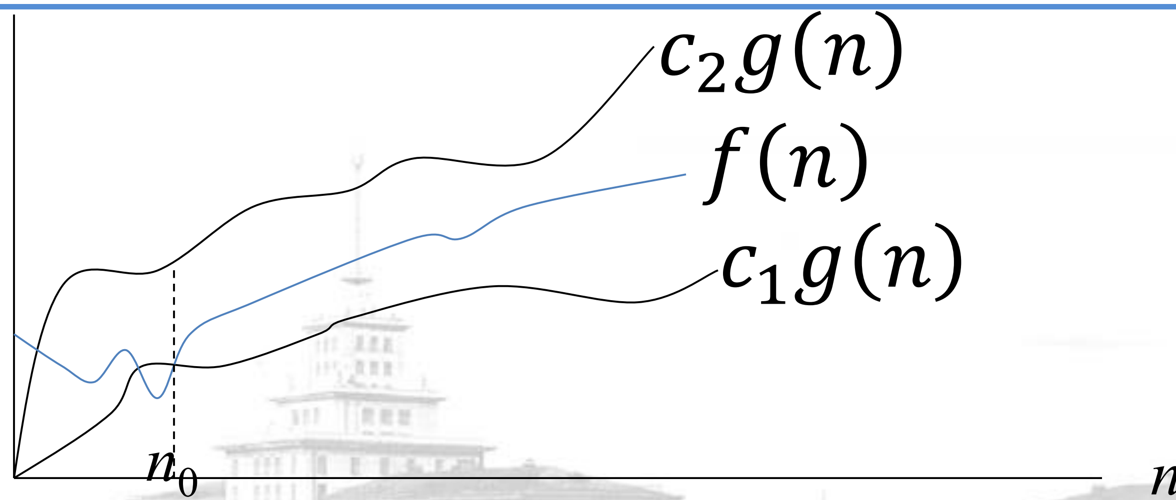
- 渐进效率:
  - 输入规模非常大
  - 忽略低阶项和常系数
  - 只考虑最高阶（增长的阶）
- 典型的增长阶:
  - $\Theta(1), \Theta(\lg n), \Theta(\sqrt{n}), \Theta(n), \Theta(n\lg n),$
  - $\Theta(n^2), \Theta(n^3), \Theta(2^n), \Theta(n!)$
- 增长的记号: $O, \Theta, \Omega, o, \omega$.

同阶函数集合

对于给定的函数$g(n)$，

$\Theta(g(n)) = \{f(n) | \exists \underline{c_1, c_2 > 0 和 n_0}, \underline{对于 \forall n} \geq n_0,$
$c_1 g(n) \leq f(n) \leq c_2 g(n)\}$，称$\Theta(g(n))$是与$g(n)$**同阶**的函数集合。

- 如果$f(n) \in \Theta(g(n))$，$f(n)$与$g(n)$同阶
- 如果$f(n) \in \Theta(g(n))$，记作$f(n) = \Theta(g(n))$



$f(n) = \Theta(g(n))$渐进紧界

$\Theta(g(n))$函数的例子

例1. 证明$\frac{1}{2}n^2 - 3n = \Theta(n^2)$。

证明：只需找到$c_1, c_2 > 0$和n_0，且满足$c_1 n^2 \le 1/2n^2 - 3n \le c_2 n^2$

两边同除n^2，得到$c_1 \le 1/2 - 3/n \le c_2$

对于任意$n \ge 1, c_2 \ge 1/2$；且对于任意$n \ge 7, c_1 \le 1/14$

因此$c_1 = 1/14$，$c_2 = 1/2, n_0 = 7$

例2. 证明$6n^3 \ne \Theta(n^2)$。

证明：如果存在$c_1, c_2 > 0$和n_0，且满足当$n > n_0$时，有

$c_1 n^2 \le 6n^3 \le c_2 n^2$，即$c_1/6 \le n \le c_2/6$

$c_2 > 0$ 是一个正常数，所以对任意大的n，不等式 $n \le c_2/6$不成立

$\Theta(g(n)) = \{f(n) | \exists c_1, c_2 > 0$和$n_0$, 对于$\forall n \ge n_0, c_1 g(n) \le f(n) \le c_2 g(n)\}$，称$\Theta(g(n))$是与$g(n)$的同阶函数集合。

$\Theta(g(n))$函数的例子

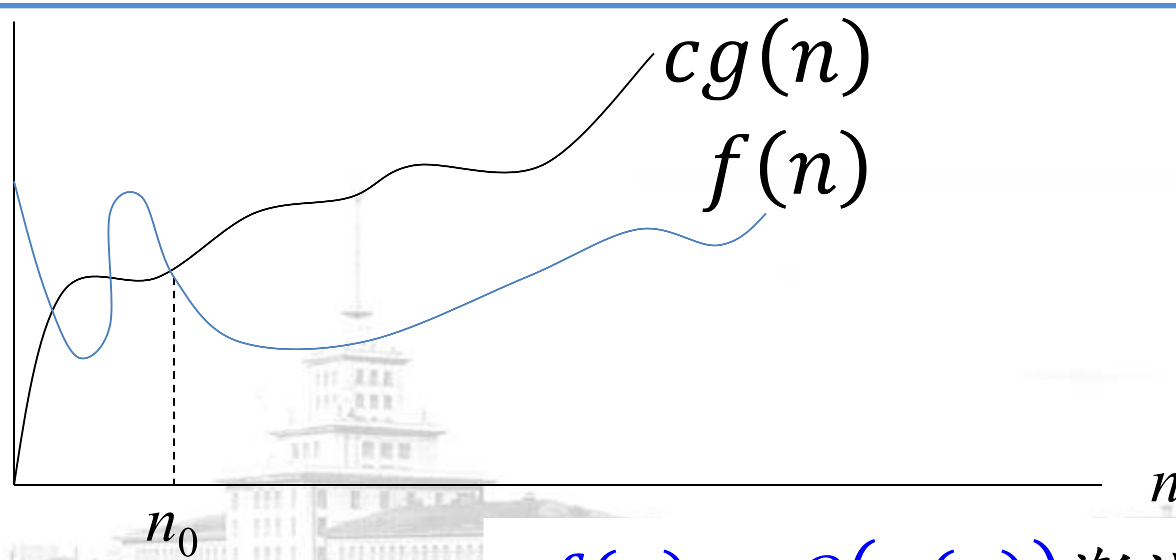
- 通常$f(n) = an^2 + bn + c = \Theta(n^2)$，其中$a, b, c$是常数且 $a > 0$
- $p(n) = \sum_{i=0}^{d} a_i n^i$，其中$a_i$是常数且$a_d > 0$.
  - $p(n) = \Theta(n^d)$
- $\Theta(n^0)$或者$\Theta(1)$，常数时间复杂性

低阶函数集合

对于给定的函数$g(n)$，

$O(g(n)) = \{f(n) | \exists c > 0$和$n_0$，对于$\forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$称$O(g(n))$是与$g(n)$**低阶**的函数集合。

– 如果$f(n) \in O(g(n))$，记为$f(n) = O(g(n))$



$f(n) = O(g(n))$渐进上界

$\Theta(g(n))$和$O(g(n))$的关系

- $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$
- Θ标记强于O标记
- $\Theta(g(n)) \subseteq O(g(n))$
- $an^2 + bn + c = \Theta(n^2)$，且$= O(n^2)$
- $an + b = O(n^2)$。为什么？
- $n = O(n^2)$!!!
- O标记，表示渐进上界
- Θ标记，表示渐进紧界
- 一些讨论:
  - 当我们谈到插入排序的最坏运行时间是$O(n^2)$，这个结论适用于所有的输入，即使对于已经排序的输入也成立，因为$O(n) \in O(n^2)$。
  - 然而插入排序的最坏运行时间$\Theta(n^2)$不能应用到每个输入，因为对于已经排序的输入，$\Theta(n) \neq \Theta(n^2)$。

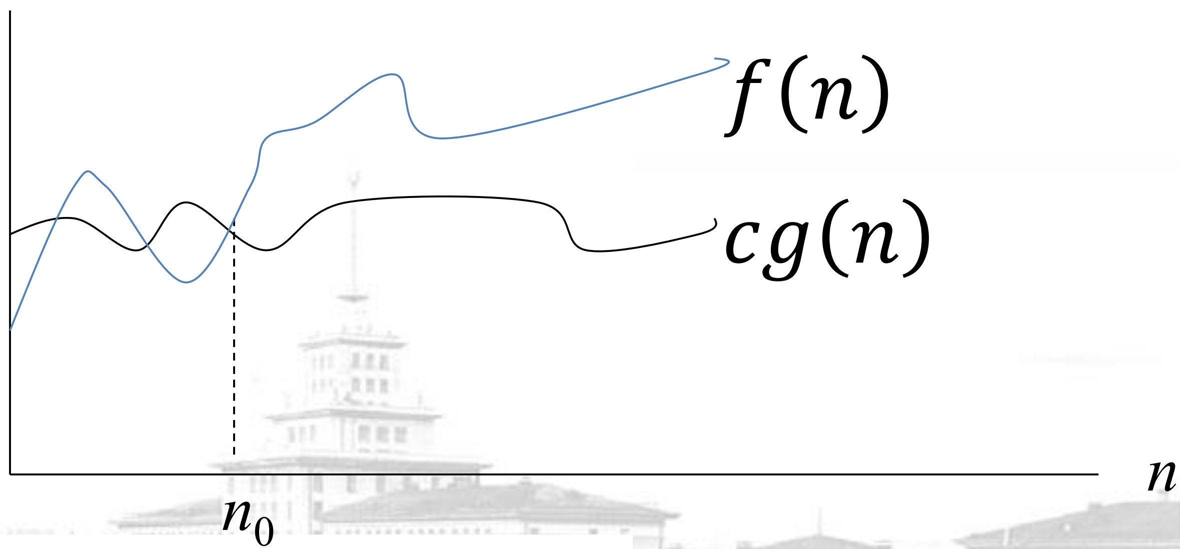
如果$f(n) = O(n^d)$,则称$f(n)$是多项式界限的。

高阶函数集合

对于给定的函数$g(n)$，

$\Omega(g(n)) = \{f(n) | \exists c$和$n_0 > 0$, 对于$\forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$称$\Omega(g(n))$是$g(n)$的**高阶**函数集合。

- 如果$f(n) \in \Omega(g(n))$，记为$f(n) = \Omega(g(n))$



$f(n) = \Omega(g(n))$渐进下界

考虑一下这个0合适吗？

关于Ω标记

- 用来描述运行时间的最好情况
- 对所有输入实例都正确
- 比如，对于插入排序
  - 最好情况下的运行时间是 $\Omega(n)$
  - 最坏情况下的运行时间是 $\Omega(n^2)$
  - 但说插入排序的运行时间是$\Omega(n^2)$则有误
- 可以用来描述问题
  - 排序问题的时间复杂性是$\Omega(n)$

O, Θ, Ω标记的关系

- 对于$f(n)$和$g(n)$，$f(n) = \Theta(g(n))$当且仅当$f(n) = O(g(n))$且$f(n) = \Omega(g(n))$。
- O：渐进上界
- Θ：渐进紧界
- Ω：渐进下界

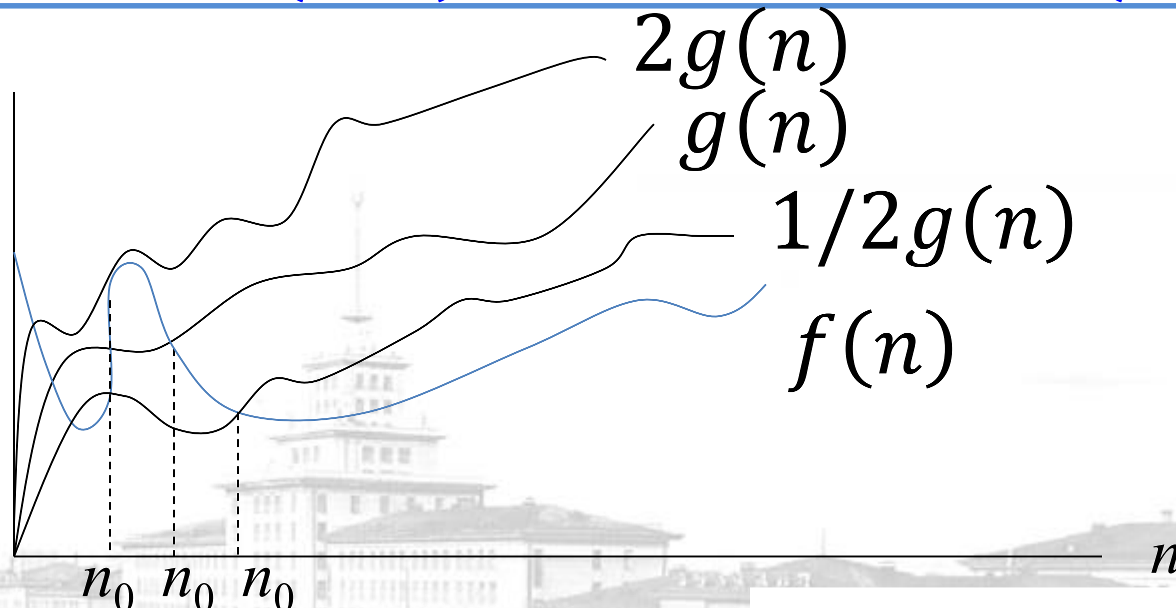
严格低阶函数

$O(g(n)) = \{f(n) | \exists c > 0$和$n_0$, 对于$\forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$，我们称为 $g(n)$ 是 $f(n)$ 的上界（upper bound）

对于给定的函数$g(n)$，

$o(g(n)) = \{f(n) |$ 对于$\forall c > 0$, 存在$\exists n_0 > 0$, 对于$\forall n \geq n_0$, $0 \leq f(n) < cg(n)\}$ 称$o(g(n))$是$g(n)$的**严格低阶**函数集合。

– 记作$f(n) \in o(g(n))$，或者简写为$f(n) = o(g(n))$



$f(n) = o(g(n))$

$o(g(n)) = \{f(n) | $对于$\forall c > 0$, 存在$\exists n_0 > 0$, 从而对所有$n \geq n_0, 0 \leq f(n) < cg(n)\}$

例1：证明$2n = o(n^2)$

证明：对$\forall c > 0$，要证$2n < cn^2$，

因为$n > 0$，必有$2 < cn$，即$n > \frac{2}{c}$。

所以，当$n_0 = \left\lceil \frac{2}{c} \right\rceil$时，对$\forall c > 0$，$n > n_0$，都有$0 < 2n < cn^2$。

例2：证明$2n^2 \neq o(n^2)$

分析：要证明$2n^2 = o(n^2)$不成立，只需要证明其不符合严格低阶函数的定义，即存在$\exists c > 0$，使得对于$\forall n > n_0$，$0 < 2n^2 < cn^2$不成立。

证明：当$c = 1 > 0$，对于$\forall n > n_0$，$0 < 2n^2 < cn^2$不成立。

关于o标记

- O标记可能是或不是紧的
  - $2n^2 = O(n^2)$ 是紧的，但$2n = O(n^2)$不是紧的。
- o标记用于标记上界但不是紧的情况
  - $2n = o(n^2)$，但是$2n^2 \neq o(n^2)$。
- 区别: 某个正常数c在O标记中， 但所有正常数c在o标记中。

$$f(n) = o\big(g(n)\big) \Longrightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$$

证明：由于$f(n) = o\big(g(n)\big)$，对任意$c > 0$，存在$n_0 > 0$, 当$n \geq n_0$时，

$$0 \leq f(n) < cg(n),$$

即$0 \leq \frac{f(n)}{g(n)} < c$。于是$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$。

严格高阶函数集合

$\Omega(g(n)) = \{f(n) | \exists c$和$n_0 > 0$, 对于$\forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$我们称 $g(n)$ 是 $f(n)$ 的渐进下界 (lower bound)

对于给定的函数$g(n)$，

$\omega(g(n)) = \{f(n) |$对于$\forall c > 0$, 存在$\exists n_0 > 0$, 对于$\forall n \geq n_0$, $0 \leq cg(n) < f(n)\}$ 称$\omega(g(n))$是$g(n)$的**严格高阶**函数集合。

- 记作$f(n) \in \omega(g(n))$，或者简写为$f(n) = \omega(g(n))$

- ω标记，类似o标记，表示不紧的下界。
  - $\frac{n^2}{2} = \omega(n)$，但$\frac{n^2}{2} \neq \omega(n^2)$
- $f(n) = \omega(g(n))$当且仅当$g(n) = o(f(n))$
- $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

渐进符号的性质

- 传递性：所有五个标记
  - $f(n) = \Theta(g(n))$ 且 $g(n) = \Theta(h(n)) \rightarrow f(n) = \Theta(h(n))$
  - $f(n) = O(g(n))$ 且 $g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$
  - $f(n) = \Omega(g(n))$ 且 $g(n) = \Omega(h(n)) \rightarrow f(n) = \Omega(h(n))$
  - $f(n) = o(g(n))$ 且 $g(n) = o(h(n)) \rightarrow f(n) = o(h(n))$
  - $f(n) = \omega(g(n))$ 且 $g(n) = \omega(h(n)) \rightarrow f(n) = \omega(h(n))$
- 自反性：Θ, O, Ω
  - $f(n) = \Theta(f(n))$
  - $f(n) = O(f(n))$
  - $f(n) = \Omega(f(n))$

为什么没有o, ω

- 对称性：Θ
  - $f(n) = \Theta(g(n))$当且仅当$g(n) = \Theta(f(n))$ 为什么？
- 反对称性：
  - $f(n) = O(g(n))$当且仅当$g(n) = \Omega(f(n))$
  - $f(n) = o(g(n))$ 当且仅当$g(n) = \omega(f(n))$

为什么？

不同的增长记号对比

① **同阶函数集合**： $\Theta(g(n)) = \{f(n) | \exists c_1, c_2 > 0和n_0, 对于\ \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

② **低阶函数集合**： $O(g(n)) = \{f(n) | \exists c和n_0 > 0, 对于\ \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$

③ **严格低阶函数集合**： $o(g(n)) = \{f(n) | \forall c > 0,\ \exists n_0 > 0, 对于\forall n \geq n_0,\ 0 \leq f(n) < cg(n)\}$

④ **高阶函数集合**： $\Omega(g(n)) = \{f(n) | \exists c和n_0 > 0, 对于\ \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$

⑤ 严格高阶函数集合： $\omega(g(n)) = \{f(n) | \forall c > 0, \exists n_0 > 0, 对于\forall n \geq n_0,\ 0 \leq cg(n) < f(n)\}$

注意！

并非所有的函数都是可比的，即对于函数$f(n)$和$g(n)$，可能$f(n) \neq O(g(n))$， $f(n) \neq \Omega(g(n))$。
例如：n和$n^{1+\sin(n)}$。

随堂测验

多选题 1分

设置

1、某算法的时间复杂度为$o(n^2)$，表明该算法的（C）

A 问题规模是n^2

B 执行时间等于n^2

C 当n足够大时，执行时间不大于n^2

D 问题规模与n^2成正比

提交

多选题 2分

设置

2、以下算法的时间复杂度为（**D**）

```
void func(int n){
    int i=1;
    while(i<=n)
        i = i*2;
}
```

A $\Theta(n)$

B $\Theta(n^2)$

C $\Theta(n\log_2 n)$

D $O(\log_2 n)$

提交

多选题 3分

设置

3、多选题：已知一个数组a的长度为n，求问下面这段代码的时间复杂度（**AB**）

```
for (i = 0, length = 1; i < n – 1; i ++){
    for (j = i + 1; j < n && a[j-1] <= a[j]; j ++)
        if(length < j – i + 1)
            length = j – i + 1;
}
```

A $\Omega(n)$

B $O(n^2)$

C $\Theta(n^2)$

D $O(n)$

提交

单选题 1分

设置

4、 $f(n)+o(f(n))=\Theta(f(n))$ 是否正确（A）

A. 正确

B. 错误

提交

本讲内容

2.1 复杂性函数阶的计算

2.2 和式的估计与界限

2.3 递归方程

为什么需要和式的估计与界限

```
For l = 2 To n Do
    For i = 1 To n-l+1 Do
        j = i + l -1;
        m[i,j] = ∞;
        For k = i To i-1 Do
            q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j;
            If  q < m[i,j] Then m[i,j] = q
```

和式的估计

1. 线性和

$$\sum_{k=1}^{n}(ca_k+b_k)=c\sum_{k=1}^{n}a_k+\sum_{k=1}^{n}b_k$$

2. 级数

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$ 算术级数

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1}-1}{x-1} \ (x \neq 1)$$ 几何级数

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \ (|x| < 1)$$

$$\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0$$

$$\sum_{k=0}^{n-1}(a_k - a_{k+1}) = a_0 - a_n$$

$$\sum_{k=1}^{n-1}\frac{1}{k(k+1)} = \sum_{k=1}^{n-1}\left(\frac{1}{k} - \frac{1}{k+1}\right) = 1 - \frac{1}{n}$$

$$\lg\left(\prod_{k=1}^{n} a_k\right) = \sum_{k=1}^{n} \lg a_k$$

(第二)数学归纳法

例1. 证明$\sum_{k=0}^{n} 3^k = O(3^n)$

证明：

对于$c \geq \frac{3}{2}$，存在n_0，当$n > n_0$时，$\sum_{k=0}^{n} 3^k \leq c3^n$

当$n = 0$时，$\sum_{k=0}^{n} 3^k = 1 \leq c = c3^n$

设$n \leq m$时成立，令$n = m + 1$，则

$$\sum_{k=0}^{m+1} 3^k = \sum_{k=0}^{m} 3^k + 3^{m+1} \leq c3^m + 3^{m+1} = c3^{m+1}\left(\frac{1}{3} + \frac{1}{c}\right)$$
$$\leq c3^{m+1}$$

对于给定的函数$g(n)$，

$O(g(n)) = \{f(n) | \exists c > 0$和$n_0$, 对于$\forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$

数学归纳法

例2. 证明$\sum_{k=1}^{n} k = O(n)$。

证明：

当$n = 1$时，$\sum_{k=1}^{n} k = 1 \le c1 = O(1)$只需$c$ 大于等于1

设$n = m$时成立，令$n = m + 1$，则

错误

$$\sum_{k=1}^{m+1} k = \sum_{k=1}^{m} k + (m+1) \le cm + (m+1) = (c+1)m + 1 = O(m)$$

要证明的应该是上式$\le c(m+1)$，即$cm + (m+1) \le c(m+1)$

则 $c \ge m + 1$才满足条件。

错在$O(n)$的常数c随n的增长而变化，不是常数。

要证明$\sum_{k=1}^{n} k = O(n)$，需证明：对某个$c > 0$，$\sum_{k=1}^{n} k \le cn$。

对于给定的函数$g(n)$，

$$O(g(n)) = \{f(n) | \exists c > 0 和 n_0, 对于 \forall n \ge n_0, 0 \le f(n) \le cg(n)\}$$

确定级数中各项的界

例1. $\sum_{k=1}^{n} k \le \sum_{k=1}^{n} n = n^2$。

例2. $\sum_{k=1}^{n} a_k \le n \times max\{a_k\}$。

例3. 设对于所有$k \ge 0, a_0 \ge 0, 0 \le \frac{a_{k+1}}{a_k} \le r < 1$，求$\sum_{k=0}^{n} a_k$的上界。

解：

$$\frac{a_1}{a_0} \le r \Rightarrow a_1 \le a_0 r$$

$$\frac{a_2}{a_1} \le r \Rightarrow a_2 \le a_1 r \le a_0 r^2$$

$$\frac{a_3}{a_2} \le r \Rightarrow a_3 \le a_2 r \le a_1 r^2 \le a_0 r^3$$

$$\dots$$

$$\frac{a_k}{a_{k-1}} \le r \Rightarrow a_k \le a_{k-1} r \le \cdots \le a_1 r^{k-1} \le a_0 r^k$$

于是，$\sum_{k=0}^{n} a_k \le \sum_{k=0}^{\infty} a_0 r^k = a_0 \sum_{k=0}^{\infty} r^k = a_0 \frac{1}{1-r} \quad (|r| < 1)$。

例4. 求$\sum_{k=1}^{\infty}\left(\frac{k}{3^k}\right)$的上界。

解：使用例3的方法

$$\frac{\frac{k+1}{3^{k+1}}}{\frac{k}{3^k}}=\frac{1}{3}\times\frac{k+1}{k}=\frac{1}{3}\times\left(1+\frac{1}{k}\right)\leq\frac{2}{3}=r<1$$

于是，

$$\sum_{k=1}^{\infty}\frac{k}{3^k}\leq\sum_{k=1}^{\infty}a_1r^k=\sum_{k=1}^{\infty}\frac{1}{3}\times\left(\frac{2}{3}\right)^k=\frac{1}{3}\times\left(\frac{1}{1-\frac{2}{3}}-1\right)=\frac{2}{3}$$

分裂求和

例1. 用分裂求和的方法求$\sum_{k=1}^{n} k$的下界。

解：

$$\sum_{k=1}^{n} k = \sum_{k=1}^{\lceil n/2 \rceil} k + \sum_{k=\lceil n/2 \rceil+1}^{n} k \geq \sum_{k=1}^{\lceil n/2 \rceil} 0 + \sum_{k=\lceil n/2 \rceil+1}^{n} \frac{n}{2} \geq \left(\frac{n}{2}\right)^2$$

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

例2. 求 $\sum_{k=0}^{\infty}\frac{k^2}{2^k}$ 的上界。

解：当$k \geq 3$时，

$$\frac{(k+1)^2/_{2^{k+1}}}{k^2/_{2^k}} = \frac{(k+1)^2}{2k^2} \leq \frac{8}{9}$$

于是

$$\sum_{k=0}^{\infty}\frac{k^2}{2^k} = \sum_{k=0}^{2}\frac{k^2}{2^k} + \sum_{k=3}^{\infty}\frac{k^2}{2^k} \leq 0 + \frac{1}{2} + 1 + \sum_{k=3}^{\infty}\frac{9}{8}\left(\frac{8}{9}\right)^k$$

$$\leq \frac{3}{2} + \sum_{k=0}^{\infty}\frac{9}{8}\left(\frac{8}{9}\right)^k = \frac{3}{2} + \frac{9}{8} * \frac{1}{1-\frac{8}{9}} = \frac{93}{8} = O(1)$$

例3. 求调和级数$H_n=\sum_{k=1}^{n}\frac{1}{k}$的上界。

解：

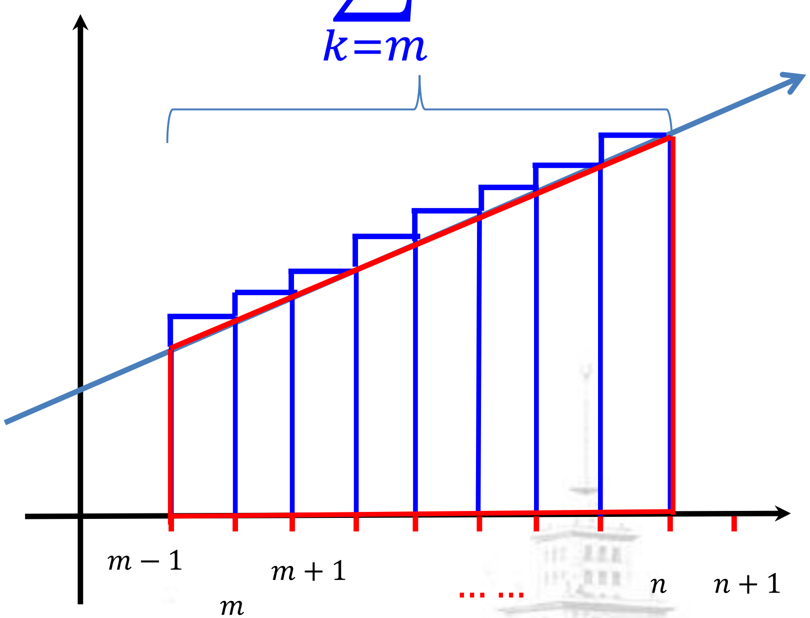
$$\sum_{k=1}^{n}\frac{1}{k}=\frac{1}{1}+\left(\frac{1}{2}+\frac{1}{3}\right)+\left(\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}\right)+\left(\frac{1}{8}+\frac{1}{9}+\frac{1}{10}+\frac{1}{11}+\frac{1}{12}+\frac{1}{13}+\frac{1}{14}+\frac{1}{15}\right)+\cdots\cdots$$

$$\le\sum_{i=0}^{\lceil\lg n\rceil}\sum_{j=0}^{2^i-1}\frac{1}{2^i+j}\le\sum_{i=0}^{\lceil\lg n\rceil}\sum_{j=0}^{2^i-1}\frac{1}{2^i}\le\sum_{i=0}^{\lceil\lg n\rceil}1\le\lg n+1=O(\lg n)$$

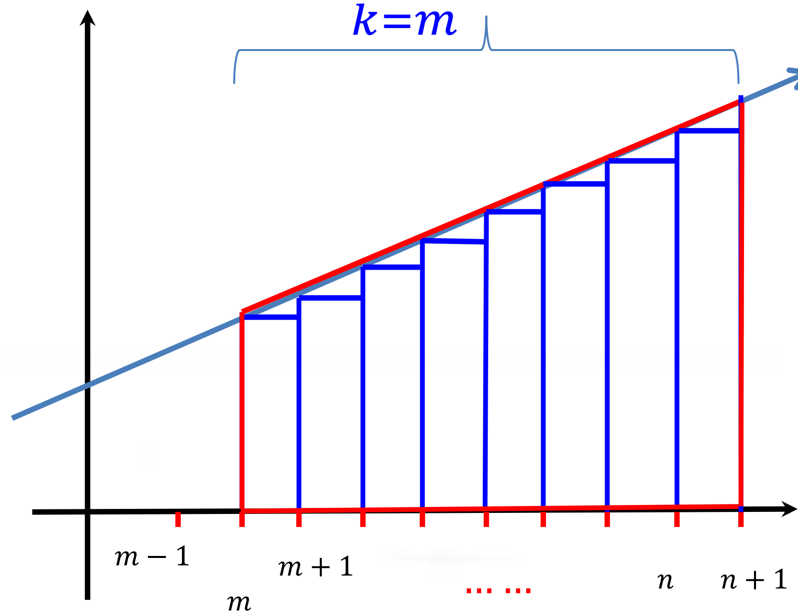
积分求和的近似

例1. 如果$f(k)$单调递增，则$\int_{m-1}^{n} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)dx$ 。

$$\sum_{k=m}^{n} f(k)$$



$$\sum_{k=m}^{n} f(k)$$



$$\int_{m-1}^{n} f(x)dx$$

注意长方形的方向

$$\int_{m}^{n+1} f(x)dx$$

例2. 如果$f(k)$单调递减，则$\int_{m}^{n+1} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx$ 。

例3.

$$\ln(n+1) = \ln x|_{1}^{n+1} = \int_{1}^{n+1} \frac{dx}{x} \leq \sum_{k=1}^{n} \frac{1}{k}$$

$$\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} = \ln x|_{1}^{n} = \ln(n)$$

单选题 1分

设置

4、$1+\frac{1}{2}+\cdots+\frac{1}{n}=o(n)$ 是否正确（A）

A. 正确

B. 错误

提交

本讲内容

2.1 复杂性函数阶的计算

2.2 和式的估计与界限

2.3 递归方程

递归方程

- 例$T(n) = 2 * T\left(\frac{n}{2}\right) + 11 * n$表示了一种递归关系
- 递归方程描述了$T(n)$与$T(< n)$之间的关系
- 挑战：

  给定关于$T(n)$的一个递推关系，

  找到关于$T(n)$的封闭表示

- 例$T(n) = O(n\lg n)$

递归方程的初始条件

- 递归方程需有基本情况或初始条件。
- $T(n) = 2 * T\left(\frac{n}{2}\right) + 11 * n$ with $T(1) = 1$

不同于

- $T(n) = 2 * T\left(\frac{n}{2}\right) + 11 * n$ with $T(1) = 1000000000$
- 递归方程描述了$T(n)$与$T(< n)$之间的关系
- 然而，$T(1) = O(1)$，因此，我们可以忽略具体的值

忽略边界条件：
对于一个常量规模的输入，算法运行时间为常量，对于足够小的n，$T(n)$为常量，改变$T(1)$不会改变函数的增长阶。

求解递归方程的三个主要方法

- 替换（代入）方法:
  - 首先猜想,
  - 然后用数学归纳法证明。
- 迭代（递归树）方法:
  - 画出递归树,
  - 把方程转化为一个和式,
  - 然后用估计和的方法来求解。
- Master定理方法:
  - 求解型为$T(n) = a * T\left(\frac{n}{b}\right) + f(n)$的递归方程。

这一部分对应于《算法导论》第三版的4.3-4.5节

替换方法

例1. 求解$T(n) = 2 * T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$，$T(1) = 1$的上界。

解：

- 根据经验，猜测其解为 $T(n) = O(n\lg n)$，要求证明：

  $\exists c, n_0 > 0, \forall n \geq n_0, T(n) \leq cn\lg n$。（$O$定义所得）

- 归纳法证明（第二数学归纳法）

（1）假设对于正整数$m < n$都成立，当$m = n/2$，那么

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \leq c \left\lfloor \frac{n}{2} \right\rfloor \lg\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

（2）验证$m = n$ 时，

$$T(n) = 2 * T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \leq 2\left(c \left\lfloor \frac{n}{2} \right\rfloor \lg\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right) + n \leq c\, n \lg\left(\frac{n}{2}\right) + n$$

$$= cn \lg n - cn \lg 2 + n = cn \lg n - cn + n$$

$$\leq cn \lg n，当\ c \geq 1。$$

替换方法

例1. 求解$T(n) = 2 * T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + n$，$T(1) = 1$的上界。

解：

- 初始条件不成立时，往后推，看是否成立
- $T(1) = c \lg 1 = 0$，与$T(n) = 1$矛盾；
- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4 \leq c * 2 * \lg 2$，只需$c > 2$，成立。

渐进符号只要求：

$\exists c, n_0 > 0$，当 $n \geq n_0$时，有$T(n) \leq cn \lg n$ 。

对于大多数递归式而言，扩展边界条件使得归纳假设对较小的n成立，是一种简单直接的方法。

考虑一下向下取整符号可不可以去掉

替换方法

猜测方法I：联想已知的$T(n)$

例2. 求解$T(n) = 2 * T\left(\frac{n}{2} + 17\right) + n$。

解：猜测：$T(n) = 2 * T\left(\frac{n}{2} + 17\right) + n$与$T(n) = 2 * T\left(\frac{n}{2}\right) + n$只相差一个常数17

当n充分大时，$T\left(\frac{n}{2} + 17\right)$和$T\left(\frac{n}{2}\right)$的差别并不大，因为$\frac{n}{2} + 17$与$\frac{n}{2}$相差小。我们可以猜测$T(n) = O(n \lg n)$。

证明：用数学归纳法

替换方法

例2. 求解$T(n) = 2 * T\left(\frac{n}{2} + 17\right) + n$。

解：

（1）不验证初始条件了。

（2）假设$m < n$都成立，即

$$T(n) \le cn\lg n$$

（3）验证$m = n$时，

$$T(n) = 2 * T\left(\frac{n}{2} + 17\right) + n \le 2c\left(\frac{n}{2} + 17\right) * \lg\left(\frac{n}{2} + 17\right) + n$$

$= (cn + 34c) * (\lg(n + 34) - 1) + n$当$n$足够大时，有$n + 34 < 1.5n$

$\le (cn + 34c) * (\lg(1.5n) - 1) + n$

$= (cn + 34c) * (\lg n + \lg 1.5 - 1) + n$

$= cn\lg n + ((\lg 1.5 - 1)c + 1)n + 34c(\lg 1.5 - 1) + 34c\lg n$

$\le cn\lg n$

猜测方法II：先证明较松的上下界，然后缩小不确定性范围

例3. 求解$T(n) = 2 * T\left(\frac{n}{2}\right) + n$。

解：首先证明 $T(n) = \Omega(n)$，$T(n) = O(n^2)$

然后逐阶地降低上界、提高下界。

$\Omega(n)$ 的上一个阶是$\Omega(n\lg n)$，

$O(n^2)$的下一个阶是$O(n\lg n)$。

细微差别的处理

- 问题：猜测正确，数学归纳法的归纳步似乎证不出来
- 解决方法：从猜测中减去一个低阶项，可能就可以了

例4. 求解$T(n)=T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)+T\left(\left\lceil\frac{n}{2}\right\rceil\right)+1$。

解：（1）我们猜测$T(n)=O(n)$

证明：$T(n)\le c\left\lfloor\frac{n}{2}\right\rfloor+c\left\lceil\frac{n}{2}\right\rceil+1=cn+1\neq cn$

因为低阶项的存在，证明不出$T(n)=O(n)$

（2）减去一个低阶项，猜测$T(n)\le cn-b$， $b\ge 0$是常数

证明：设当$m<n$时成立$T(m)\le cm-b$成立。那么$m=n$时

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

$$=cn-2b+1=cn-b-b+1\le cn-b$$（只要$b\ge 1$）

c必须充分大，以满足边界条件。

为什么是“减去” 低阶项？

避免陷阱

例5. 求解$T(n) = 2 * T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + n$。

解：猜测$T(n) = O(n)$

证明：用数学归纳法证明$T(n) \le cn$

$$T(n) = 2 * T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + n \le 2\left(c\left\lfloor\frac{n}{2}\right\rfloor\right) + n \le (c+1)n = O(n)$$

错误

错在哪里？

错误在于没有与归纳假设严格一致的形式，即$T(n) \le cn$。从$T(n) \le (c+1)n$不可能得到$T(n) \le cn$。因为对于任何$c > 0$，我们都得不到$(c+1)n \le cn$。

变量替换方法：
经变量替换把递归方程变换为熟悉的方程。

例6. 求解$T(n) = 2 * T(\sqrt{n}) + \lg n$。只考虑$\sqrt{n}$是整数的情形。

解：令$m = \lg n$，则$n = 2^m$，$T(2^m) = 2 * T\left(2^{\frac{m}{2}}\right) + m$

令$S(m) = T(2^m)$，则$T\left(2^{\frac{m}{2}}\right) = S\left(\frac{m}{2}\right)$。

于是，$S(m) = 2S\left(\frac{m}{2}\right) + m$。

显然，$S(m) = O(m\lg m)$，即$T(2^m) = O(m\lg m)$。

由于$2^m = n$，$m = \lg n$，$T(n) = O(\lg n \lg(\lg n))$。

迭代（递归树）方法

方法：

- 画出递归树
- 循环地展开递归方程
- 把递归方程转化为和式
- 然后可使用求和技术求解

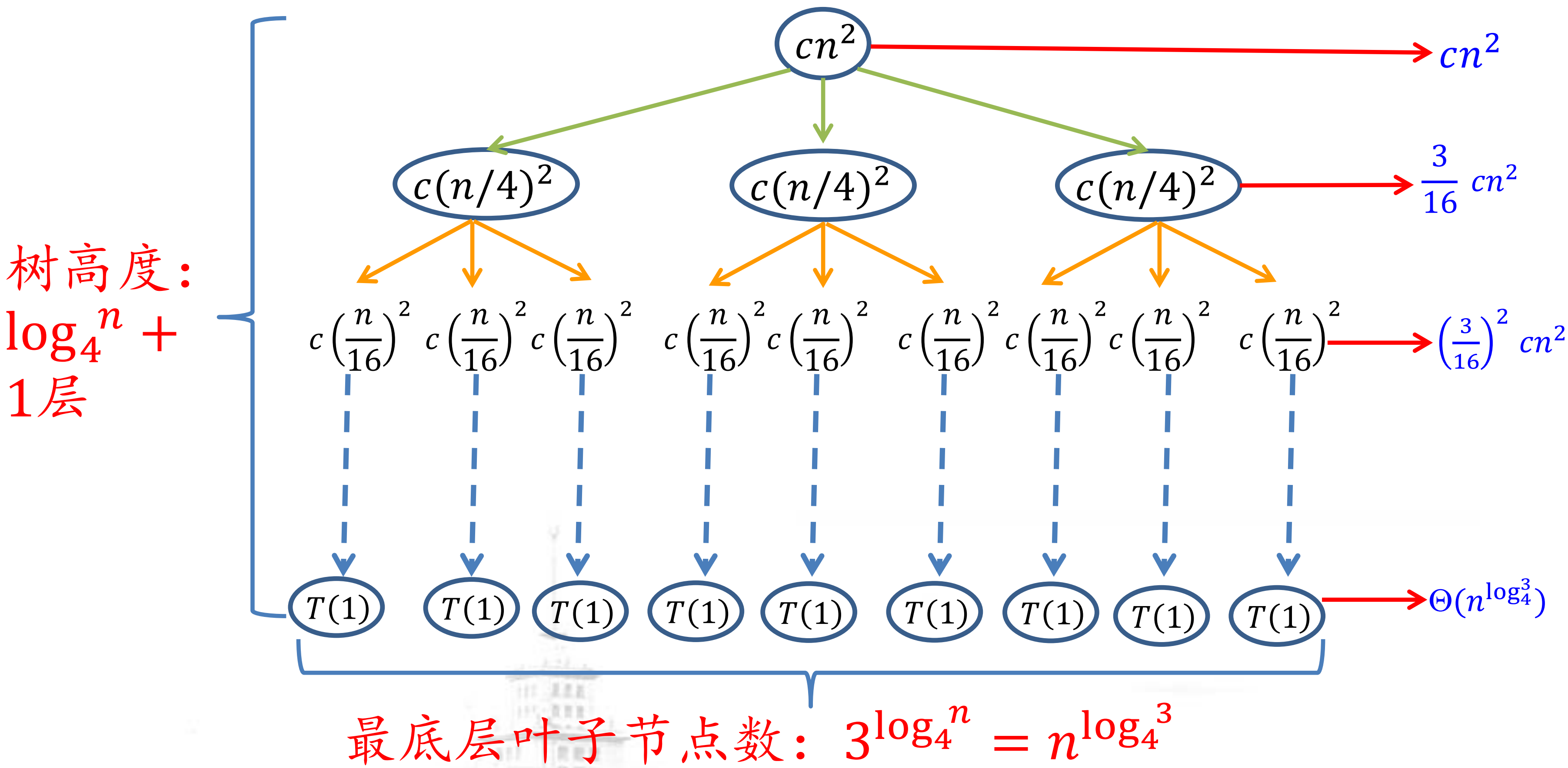
递归树最适合用来生成好的猜测，然后即可用代入法来验证猜测是否正确。

递归树

- 根结点表示递归调用顶层的代价
- 内部节点，表示不同层次递归调用产生的代价，即（合并）子问题的代价
- 树的分枝数量取决于子问题的数量
- 叶节点表示边界条件值

递归树

例1. $T(n) = 3 * T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$。假定$n$是4的幂。



总的代价之和=迭代树右侧所有代价之和！！！

递归树

例1. $T(n) = 3 * T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$。

解：

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

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

根据：$\sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x}$

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

我们得出$T(n) = O(n^2)$

用代入法来验证。

发现什么规律？

例2. $T(n)=n+3T(\lfloor n/4 \rfloor)$。

$$T(n)=n+3T(\lfloor n/4 \rfloor)=n+3\left(\lfloor n/4 \rfloor+3T(\lfloor n/16 \rfloor)\right)$$ 定义所得

$$=n+3\left(\lfloor n/4 \rfloor+3\left(\lfloor n/16 \rfloor+3T(\lfloor n/64 \rfloor)\right)\right)$$

$$=n+3\lfloor n/4 \rfloor+3^2\left\lfloor n/4^2 \right\rfloor+3^3\left\lfloor n/4^3 \right\rfloor+\cdots+3^iT\left(\left\lfloor n/4^i \right\rfloor\right)$$

令 $n/4^i=1 \Rightarrow 4^i=n \Rightarrow i=\log_4^n$

$$=n+3\lfloor n/4 \rfloor+3^2\left\lfloor n/4^2 \right\rfloor+3^3\left\lfloor n/4^3 \right\rfloor+\cdots+3^{\log_4^n}T(\lfloor 1 \rfloor)$$

$$\le \sum_{i=0}^{\log_4^n-1} 3^i\, n/4^i+\Theta\left(n^{\log_4^3}\right)\le n\sum_{i=0}^{\infty}\left(3/4\right)^i+\Theta(n^{\log_4^3})=$$

$$n\times\frac{1}{1-\frac{3}{4}}+\Theta(n^{\log_4^3})=4n+\Theta(n^{\log_4^3})=O(n)$$

Master定理方法

目的：求解 $T(n)=aT(^n/_b)+f(n)$ 型方程，$a\geq 1$， $b>1$ 常数，$f(n)$ 是渐近正函数。

方法：记住三种情况，则不用笔纸即可求解上述方程。

Master 定理：设 $a\geq 1$ 和 $b>1$ 是常数， $f(n)$ 是一个函数，

$T(n)$ 是定义在非负整数集上的函数

$$T(n)=aT(^n/_b)+f(n)。$$

那么 $T(n)$ 可以如下求解：

（1）若 $f(n)=O(n^{\log_b{}^{a}-\varepsilon})$， $\varepsilon>0$ 是常数，则 $T(n)=\Theta(n^{\log_b{}^{a}})$。

（2）若 $f(n)=\Theta(n^{\log_b{}^{a}})$ **（同阶）**，则 $T(n)=\Theta(n^{\log_b{}^{a}}\lg n)=\Theta(f(n)\lg n)$ 。

（3）若 $f(n)=\Omega(n^{\log_b{}^{a}+\varepsilon})$， $\varepsilon>0$ 是常数，且对某个常数 $c<1$ 和所有充分大的 n，有 $af(^n/_b)\leq cf(n)$，则 $T(n)=\Theta(f(n))$。

Master 定理

$$T(n) = aT\left(n/_b\right) + f(n)$$

Master定理适用于，通过解一个问题的子问题，来解一个问题，并且子问题规模相同

a:　　子问题数量

b:　　子问题从原问题缩小的比例，

n/b为子问题规模

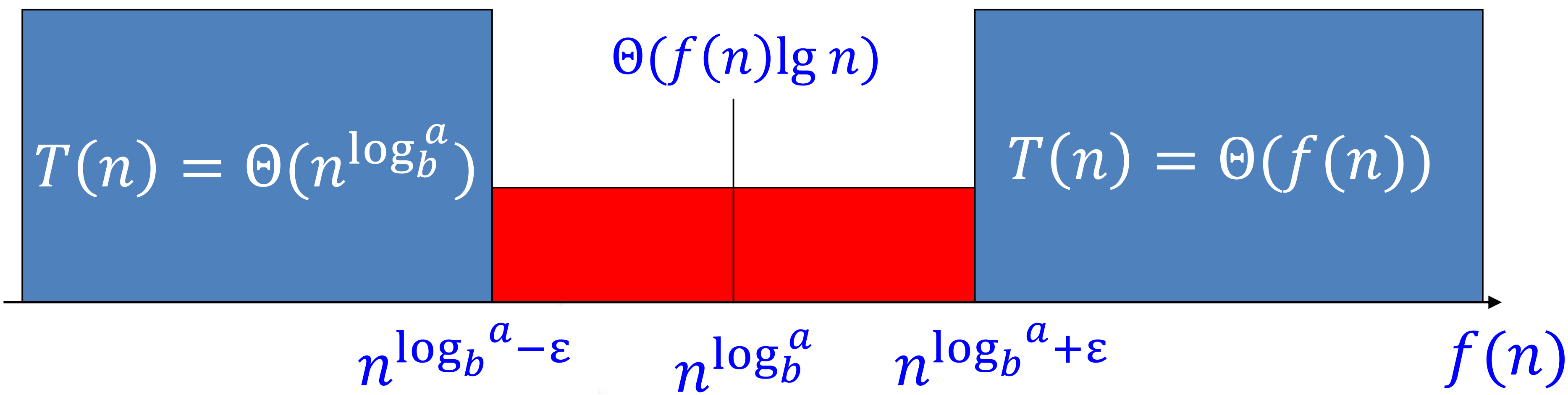
$f(n)$: 将问题分解和子问题解整合的代价

直观地：我们用$f(n)$与$n^{\log_b a}$比较：

（1）若$n^{\log_b a}$多项式地大，则$T(n) = \Theta(n^{\log_b a})$。

（2）若$f(n)$多项式地大，则$T(n) = \Theta(f(n))$。

（3）若$f(n)$与$n^{\log_b a}$同阶，则$T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(f(n)\lg n)$。



对于红色部分，Master定理无能为力

更进一步：

（1）在第一种情况，$f(n)$不仅小于$n^{\log_b^a}$，必须多项式地小于，

即对于一个常数$\varepsilon > 0$，$f(n) = O\left(\frac{n^{\log_b^a}}{n^{\varepsilon}}\right)$。

（2）在第三种情况，$f(n)$不仅大于$n^{\log_b^a}$，必须多项式地大于，
即对于一个常数$\varepsilon > 0$，$f(n) = \Omega\left(n^{\log_b^a} \cdot n^{\varepsilon}\right)$。

怎么证明？参考书中4.6节采用递归树证明。

Master定理的使用

（1）若$f(n)=O(n^{\log_b^a-\varepsilon})$，$\varepsilon>0$是常数，则$T(n)=\Theta(n^{\log_b^a})$

（2）若$f(n)=\Theta(n^{\log_b^a})$，则$T(n)=\Theta(n^{\log_b^a}\lg n)$。

例1. 求解$T(n)=9T(^n/_3)+n$。

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

因为$f(n)=n=O(n^{\log_b^a-\varepsilon})$，$\varepsilon=1$

所以$T(n)=\Theta(n^{\log_b^a})=\Theta(n^2)$

例2. 求解$T(n)=T(^{2n}/_3)+1$。

解：$a=1$，$b=^3/_2$，$f(n)=1$，$n^{\log_b{}^a}=n^{\log_{3/2}{}^1}=n^0=1$，

因为$f(n)=1=\Theta(1)=\Theta(n^{\log_b{}^a})$，

所以$T(n)=\Theta(n^{\log_b{}^a}\lg n)=\Theta(\lg n)$

Master定理的使用（续）

（3）若$f(n)=\Omega(n^{\log_b{}^a+\varepsilon})$，$\varepsilon>0$是常数，且对于所有充分大的$n$，$af(^n/_b)\le cf(n)$，$c<1$是常数，则$T(n)=\Theta(f(n))$。

例3. 求解$T(n)=3T(^n/_4)+n\lg n$。

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

（1）$f(n)=n\lg n\ge n^{\log_b{}^a+\varepsilon}$，$\varepsilon\approx 0.2$

（2）对所有n， $af(^n/_b)=3\times\frac{n}{4}\lg\frac{n}{4}=\frac{3}{4}n\lg\frac{n}{4}\le\frac{3}{4}n\lg n=$

$cf(n)$，$c=\frac{3}{4}$

于是$T(n)=\Theta(f(n))=\Theta(n\lg n)$。

Master定理的使用（续）

例4. 求解$T(n) = 2T(^n/_2) + n\lg n$。

解： $a = 2$，$b = 2$，$f(n) = n\lg n$，$n^{\log_b a} = n^{\log_2 2} = n$

$f(n) = n\lg n$大于$n^{\log_b a} = n$，但不是多项式地大于，Master 定理不适用于该$T(n)$。

作业