算法设计与分析

第二章 算法分析的数学基础

户保田
e-mail:hubaotian@hit.edu.cn

哈尔滨工业大学（深圳）计算机学院

本讲内容

2.1 计算复杂性函数的阶

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

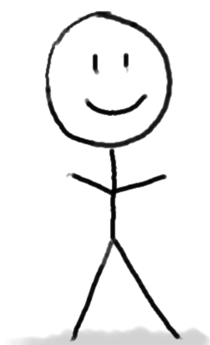
- 记录算法实现的程序在机器上实际运行的时间？
    - 实现代码的语言的效率差别很大
    - 代码的优化程度
    - 机器的运算速度，指令集
    - …

对比不同算法的实际运行时间非常困难！

增长的阶

- 核心思想

关注算法的运行时间是如何随着**问题的规模**（也即输入大小n）而变化的



Algorithm designer

Here is my algorithm!

```
Algorithm:
  Do the thing
  Do the stuff
  Return the answer
```

Here is an input!

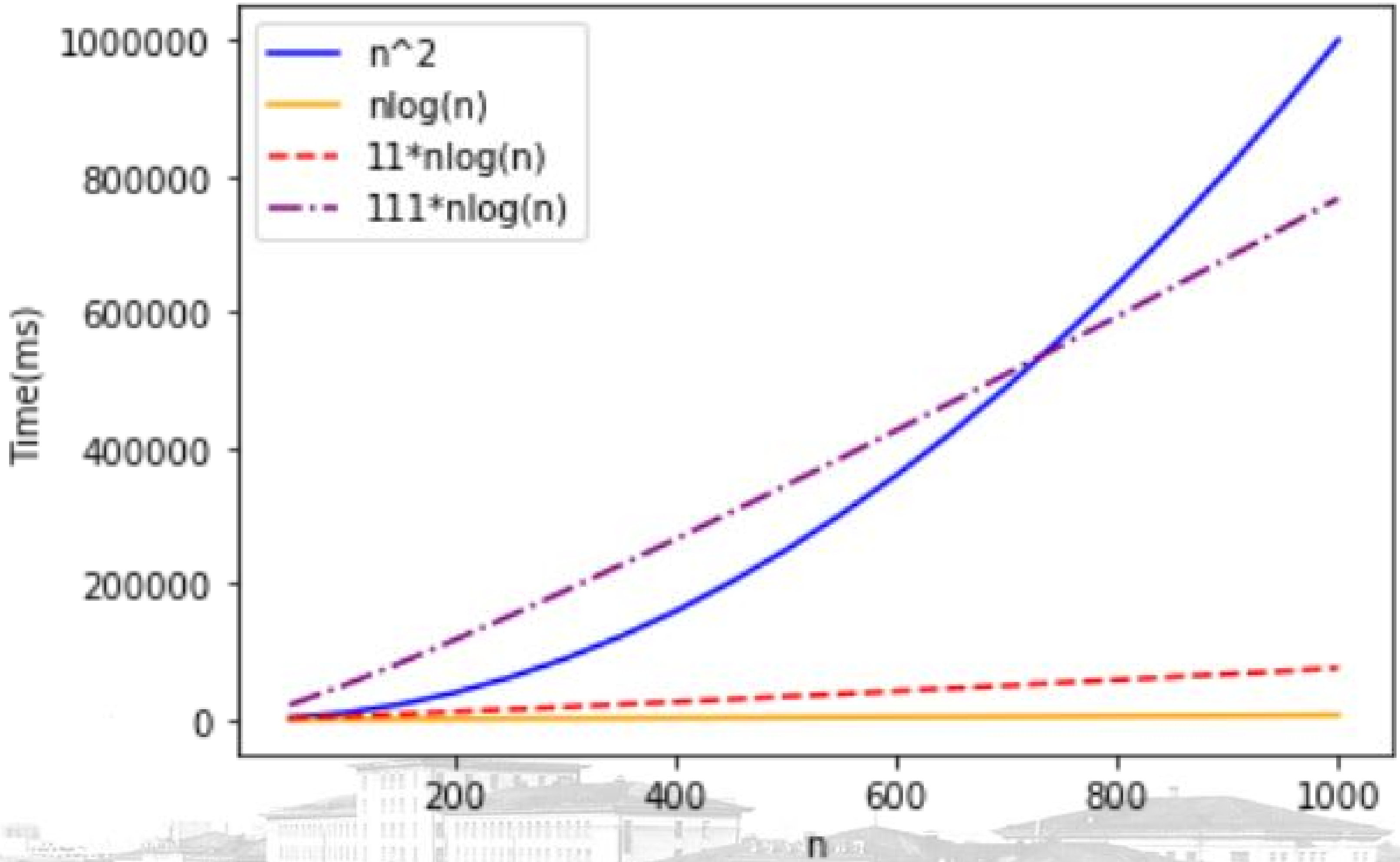
增长的阶

- 算法增长的阶也即**增长率**或**时间复杂度**（time complexity)
- 只与输入**问题的规模**相关
- 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复杂度，**不是说他的运行时间是n^2**

增长的阶

- 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, Θ, Ω, o, ω

增长的阶



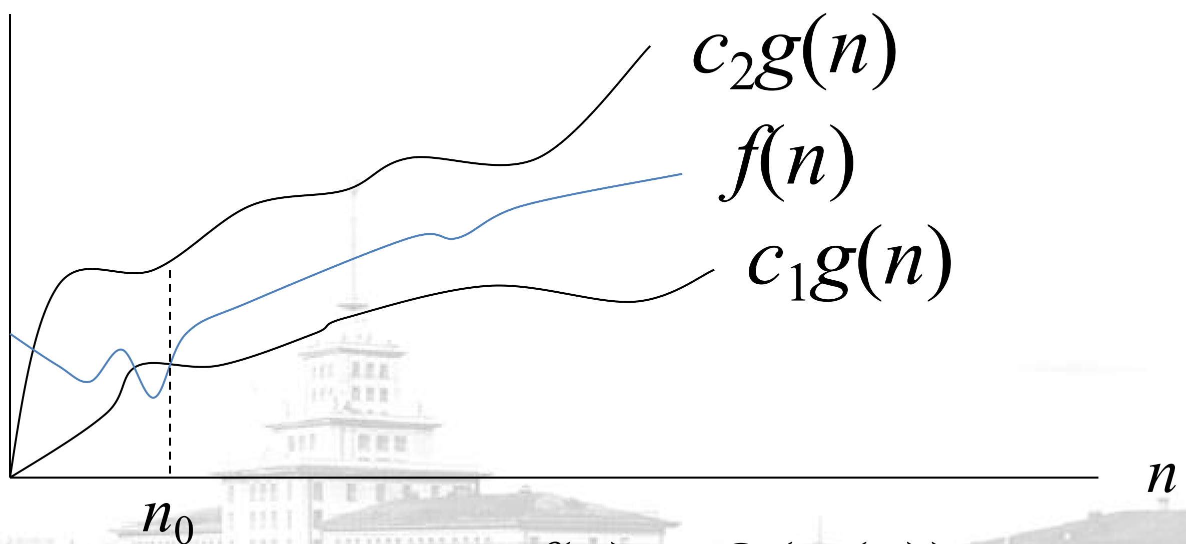
n^2
nlog(n)
11*nlog(n)
111*nlog(n)
1000000
800000
600000
400000
200000
0
Time(ms)
200
400
600
800
1000
n

同阶函数集合

$\Theta(g(n))=\{f(n) \mid \exists c_1, c_2>0,\ n_0,\ \forall n>n_0,\ c_1 g(n) \le f(n) \le c_2 g(n)\}$

$\Theta(g(n))$称为$g(n)$的同阶函数集合。

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



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

$\Theta(g(n))$ 函数的例子

$\Theta(g(n))=\{f(n) \mid \exists c_1, c_2>0, n_0, \forall n>n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$,$\Theta(g(n))$称为$g(n)$同阶的函数集合。

- **证明** $1/2n^2 - 3n = \Theta(n^2)$
  - $c_1 n^2 \leq 1/2n^2 - 3n \leq c_2 n^2$
  - $c_1 \leq 1/2 - 3/n \leq c_2$
  - 对于任意$n \geq 1$, $c_2 \geq ½$; 且对于任意$n \geq 7$，$c_1 \leq 1/14$
  - 因此 $c_1 = 1/14$, $c_2 = ½$, $n_0 = 7$.
- **证明** $6n^3 \neq \Theta(n^2)$

如果存在c_1、$c_2>0$，n_0使得当$n \geq n_0$时，$c_1 n^2 \leq 6n^3 \leq c_2 n^2$ 。也即 $c_1/6 \leq n \leq c_2/6$, 显然与 $\forall n>n_0$， $c_1 g(n) \leq f(n) \leq c_2 g(n)$ 矛盾

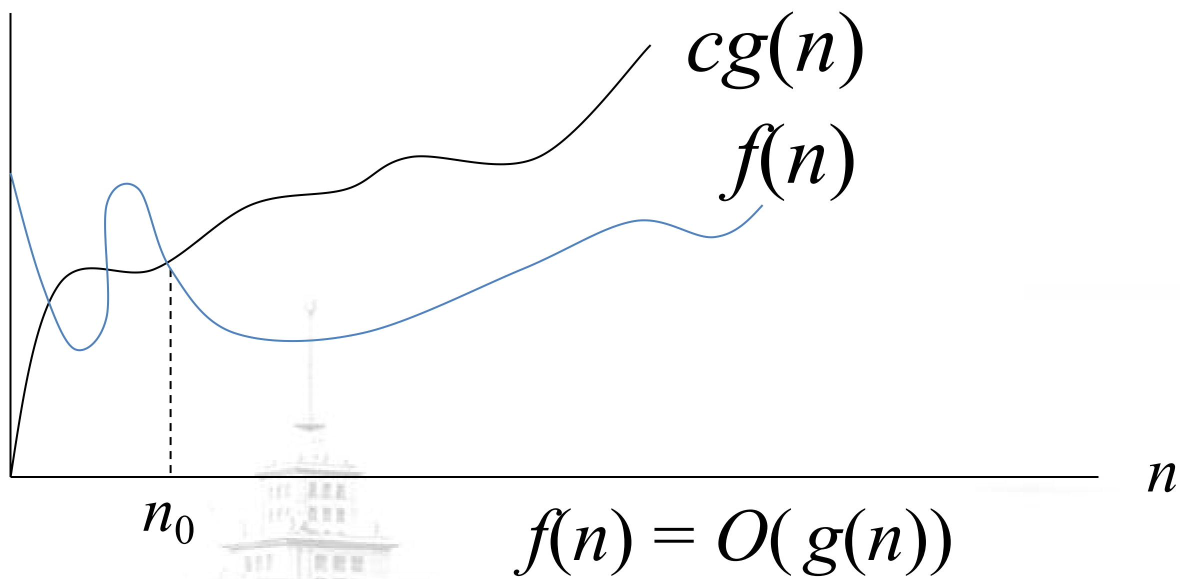
$\Theta(g(n))$ 函数的例子

- 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$a_d>0$, 则$p(n) = \Theta(n^d)$
- $\Theta(n^0)$或者$\Theta(1)$，常数时间复杂性

低阶函数集合

$O(g(n))=\{f(n) \mid \exists c>0,\ n_0,\ \forall n>n_0,\ 0\leq f(n) \leq cg(n)\}$，$O(g(n))$称为$g(n)$的低阶函数集合。

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



$\Theta(g(n))$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？
- $n = O(n^2)$!!!

如果$f(n)=O(n^k)$, 则称$f(n)$是多项式界限的。

- O标记, 表示渐进上界
- Θ标记, 表示渐进紧界

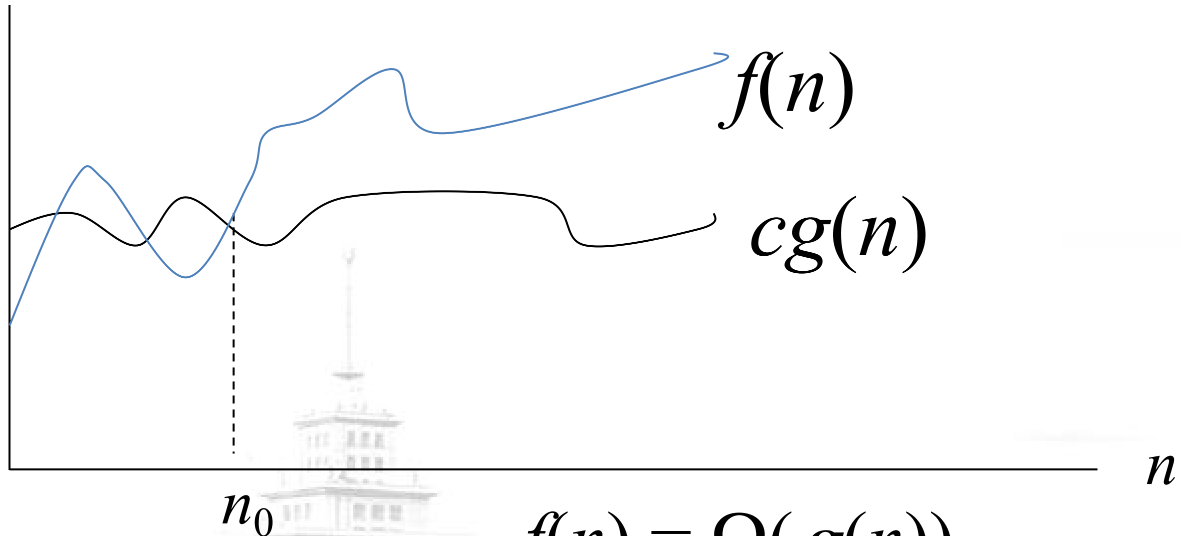
- 一些讨论:
  - 当我们谈到插入排序的最坏运行时间是$O(n^2)$, 这个结论适用于所有的输入，即使对于已经排序的输入也成立，因为$O(n) \in O(n^2)$
  - 然而插入排序的最坏运行时间$\Theta(n^2)$不能应用到每个输入，因为对于已经排序的输入, $\Theta(n) \neq \Theta(n^2)$

高阶函数集合

$\Omega(g(n))=\{f(n) \mid \exists c>0, n_0, \forall n>n_0, 0 \le cg(n) \le f(n)\}$,

$\Omega(g(n))$称为$g(n)$的高阶函数集合

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



$f(n) = \Omega(g(n))$

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

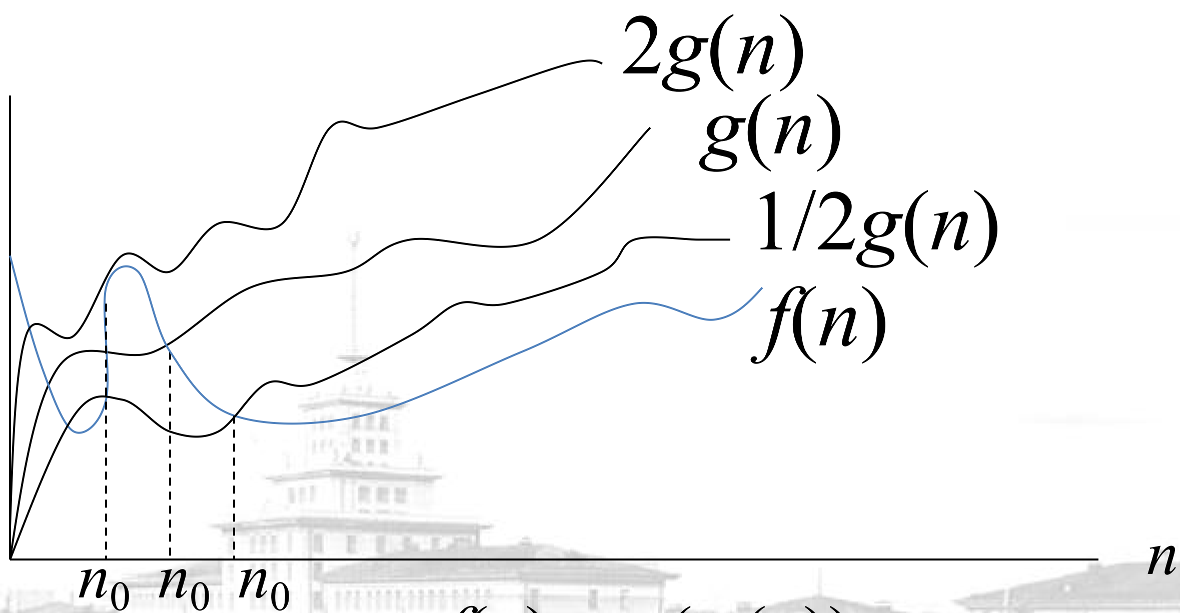
严格低阶函数

O(g(n))={f(n) | $\exists c>0$, n_0, $\forall n>n_0$, 0≤ f(n) ≤ cg(n)}，O(g(n))称为g(n)的低阶函数集合。

$o(g(n))=\{f(n) \mid \forall c>0,\ \exists n_0>0,\ \forall n\geq n_0,\ 0\leq f(n) < cg(n)\}$，

$o(g(n))$称为$g(n)$的严格低阶函数集合。

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



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

$o(g(n))=\{f(n) \mid \forall c>0, \exists n_0>0, \forall n \geq n_0, 0 \leq f(n) < cg(n)\}$，$o(g(n))$称为$g(n)$的严格低阶函数集合。

例 1．证明$2n = o(n^2)$

证．对$\forall c>0$，欲$2n < cn^2$，必$2 < cn$，即$\frac{2}{c} < n$。所以，当$n_0 = \frac{2}{c}$时，$2n < cn^2$对$\forall c>0$，$n \geq n_0$。

例 2．证明$2n^2 \neq o(n^2)$

证．当$c=1>0$时，对于任何n_0，当$n \geq n_0$，$2n^2 < cn^2$都不成立

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

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

证. 由于 $f(n)=o(g(n))$，对任意$\varepsilon>0$, 存在n_0， 当$n \geq n_0$时，

$$0 \leq f(n) < \varepsilon g(n),$$

即$0 \leq \dfrac{f(n)}{g(n)} \leq \varepsilon$. 于是，$\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.

严格高阶函数集合

$\Omega(g(n))=\{f(n)|\ \exists c>0,\ n_0, \forall n>n_0,\ 0 \le cg(n) \le f(n)\}$，$\Omega(g(n))$称为$g(n)$的高阶函数集合

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

$\omega(g(n))$称为$g(n)$的严格高阶函数集合

- 记作$f(n) \in \omega(g(n))$, 或者简记为$f(n) = \omega(g(n))$
- ω标记, 类似o标记, 表示不紧的下界
  - $n^2/2 = \omega(n)$, 但$n^2/2 \ne \omega(n^2)$
- $f(n) = \omega(g(n))$当且仅当$g(n)=o(f(n))$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

渐进符号的性质

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

不同的增长记号对比

同阶函数集合：$\Theta(g(n))=\{f(n) \mid \exists c_1, c_2>0,\ n_0,;\ \ \forall n>n_0, c_1g(n)\leq f(n)\leq c_2g(n)\}$

低阶函数集合：$O(g(n))=\{f(n) \mid \exists c，\ n_0>0；\ \forall n\geq n_0; 0\leq f(n) \leq cg(n)\}$

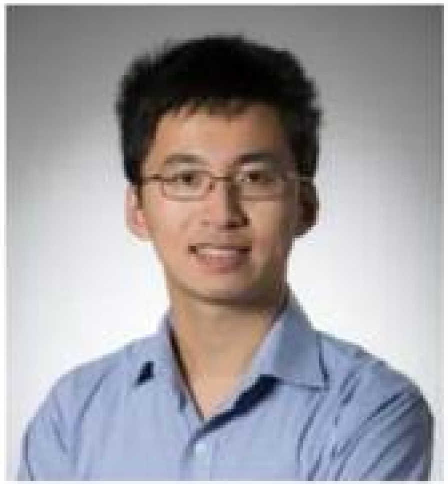
严格低阶函数集合：$o(g(n))=\{f(n) \mid \exists n_0>0；\ \forall c>0，\ n\geq n_0; 0\leq f(n) < cg(n)\}$

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

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

注意

*并非所有函数都是可比的，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 $n^{1+\sin n}$.



Best Paper Award at ACM-SIAM Symposium on Discrete Algorithms (SODA) 2021

Professor Richard Peng and Professor Santosh Vempala received the best paper award at ACM-SIAM Symposium on Discrete Algorithms (SODA) 2021 for their breakthrough work on Solving Sparse Linear Systems Faster than Matrix Multiplication.

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \qquad\quad \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m \end{cases}$$

高斯消元法

$O(n^3)$

迭代法

$O(n^{2.331645})$

https://mp.weixin.qq.com/s/2tKj47rHEZ4FL99r5k0bZw

本讲内容

2.1 计算复杂性函数的阶

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

和式的估计

1. 线性和

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

和式的估计

2. 级数

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

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

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

和式的估计

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

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

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

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

数学归纳法

例1. 证明 $\sum_{k=0}^{n} 3^k = O(3^n)$

O(g(n))={f(n) | $\exists c>0$, n_0, $\forall n>n_0$，$0 \le f(n) \le cg(n)$}，O(g(n))称为g(n)的低阶函数集合。

证明：只需证$\exists c$，n_0，当$n > n_0$时，$\sum_{k=0}^{n} 3^k \le c3^n$成立，取$c = \frac{3}{2}$

- 当$n = 0$时，$\sum_{k=0}^{n} 3^k = 1 \le c = c3^n = \frac{3}{2}$
- 假设$n \le m$时成立
- 当$n = m + 1$时，则：$\sum_{k=0}^{m+1} 3^k = \sum_{k=0}^{m} 3^k + 3^{m+1}$

$$\le c3^m + 3^{m+1} = c3^{m+1}\left(\frac{1}{3} + \frac{1}{c}\right)$$

$$\le c3^{m+1}$$

数学归纳法

O(g(n))={f(n) | ∃c>0, n_0, ∀n>n_0，0≤ f(n) ≤ cg(n)}，O(g(n))称为g(n)的低阶函数集合。

例2. 证明 $\sum_{k=1}^{n} k = O(n)$

证明:

- 当$n = 1$时，$\sum_{k=1}^{n} k = 1 = O(1)$
- 假设$n = m$时成立
- 当$n = m + 1$时，则： $\sum_{k=1}^{m+1} k = \sum_{k=1}^{m} k + (m+1)$

$$= O(m) + (m+1) = O(m)$$

错误

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$c > 0$，$\sum_{k=1}^{n} k \le cn$

确定级数中各项的界

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

$$\sum_{k=1}^{n} a_k \le n \times max\{a_k\}$$

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

解：

$$\frac{a_1}{a_0} \le r \Rightarrow a_1 \le a_0 r$$
$$\frac{a_2}{a_1} \le r \Rightarrow a_2 \le a_1 r \le a_0 r^2$$
$$\frac{a_3}{a_2} \le r \Rightarrow a_3 \le a_2 r \le a_1 r^2 \le a_0 r^3$$
$$\cdots$$
$$\frac{a_k}{a_{k-1}} \le r \Rightarrow a_k \le a_{k-1} r \le \cdots \le a_1 r^{k-1} \le a_0 r^k$$

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

确定级数中各项的界

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

解：使用例3的方法

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

于是,

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

确定级数中各项的界

例1. 用分裂求和的方法求$\sum_{k=1}^{n} k$的下界。

解:

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

确定级数中各项的界

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:

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

分裂求和

例3. 求$H_n = \sum_{k=1}^{n} \frac{1}{k}$的上界

解:

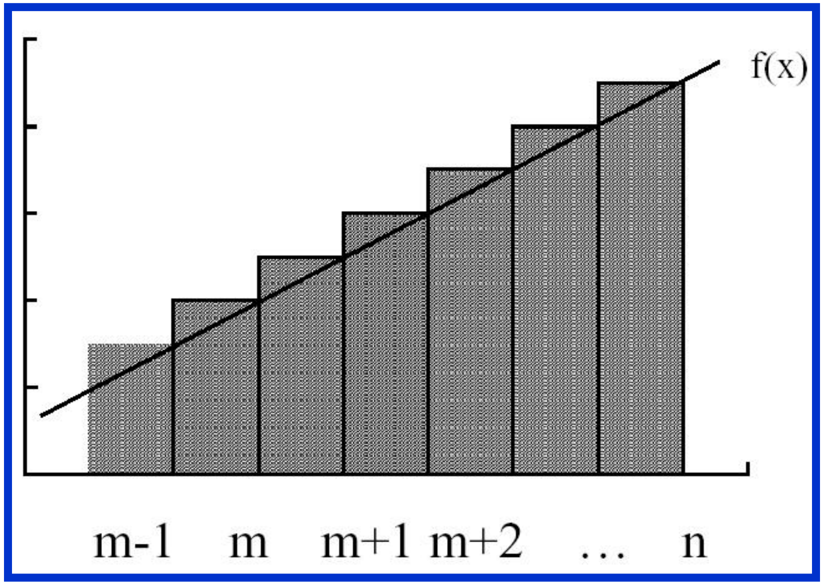
$$\sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \left(\frac{1}{2} + \frac{1}{3}\right) + \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}\right) +$$

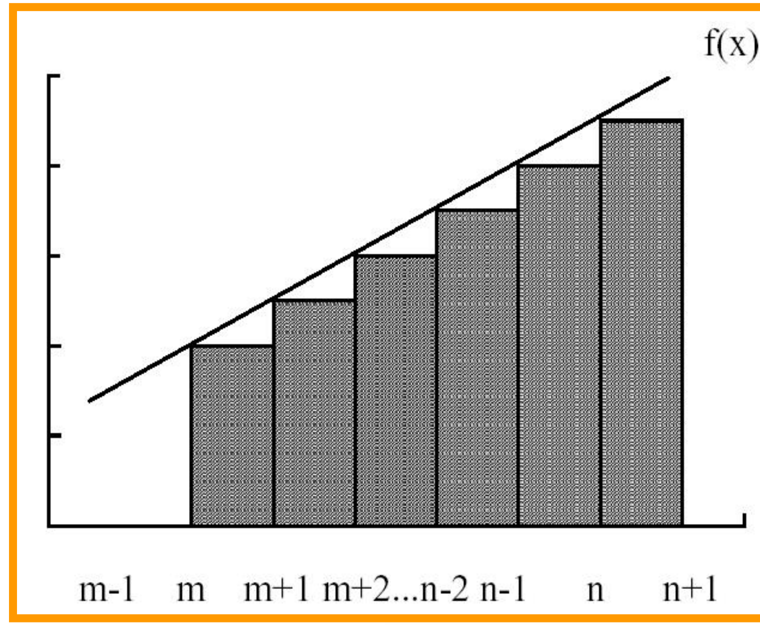
$$\left(\frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \frac{1}{11} + \frac{1}{12} + \frac{1}{13} + \frac{1}{14} + \frac{1}{15}\right) + \cdots\cdots$$

$$\le \sum_{i=1}^{\lfloor \log n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j} \le \sum_{i=1}^{\lfloor \log n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i} \le \sum_{i=1}^{\lfloor \log n \rfloor} 1 \le \log n + 1 = O(\log n)$$

积分求和的近似

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





$$\sum_{k=m}^{n} f(k) = \sum_{k=m}^{n} f(k)\Delta x \ge \int_{m-1}^{n} f(x)dx,\ f(m-1) < f(n), \Delta x = 1$$

$$\sum_{k=m}^{n} f(k) = \sum_{k=m}^{n} f(k)\Delta x \le \int_{m}^{n+1} f(x)dx$$

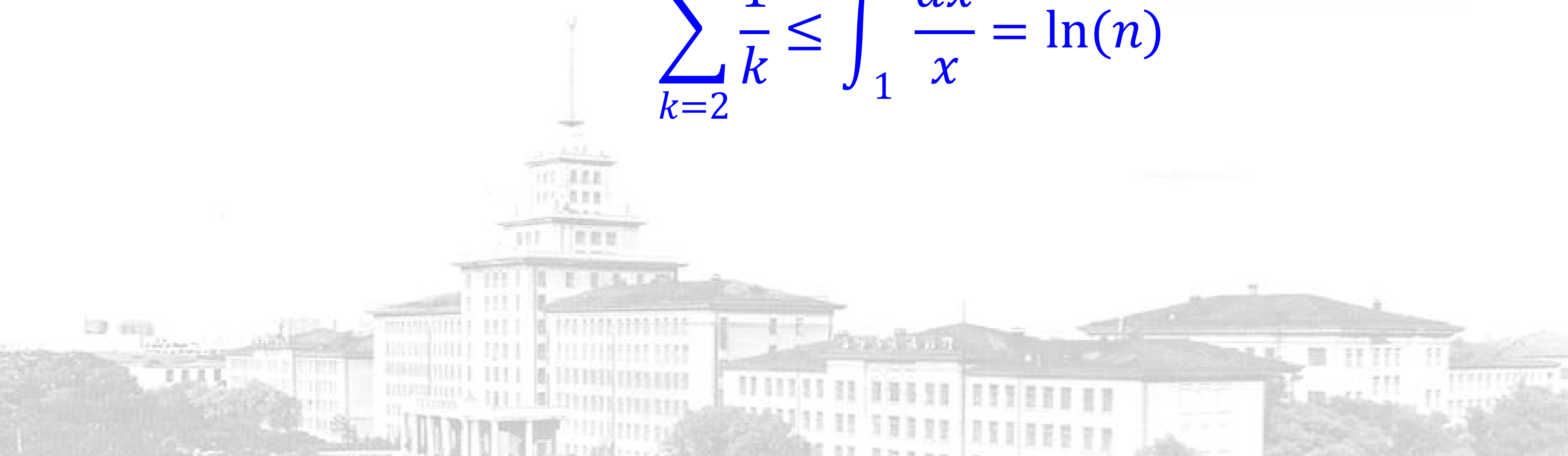
积分求和的近似

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

例3.

$$\sum_{k=1}^{n} \frac{1}{k} \ge \int_1^{n+1} \frac{1}{x} dx = \ln(n+1)$$

$$\sum_{k=2}^{n} \frac{1}{k} \le \int_1^{n} \frac{dx}{x} = \ln(n)$$



本讲内容

2.1 计算复杂性函数的阶

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：

  Given a recurrence relation for $T(n)$,

  find a closed-form repression for $T(n)$

- 例 $T(n)=O(n\log n)$

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 $T(1) = 1$

is not the same as

- $T(n) = 2 * T\left(\frac{n}{2}\right) + 11 * n$ with $T(1) = 100000000$
- 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

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$T(n)=aT(n/b)+f(n)$的递归方程

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
- $T(1) = c_1 \log 1 = 0$，与$T(n) = 1$矛盾
- $T(2) = 4 \leq c_2 2 \log 2$，只需$c > 2$，成立

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

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$O(n\lg n)$

替换方法

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guess中减去一个低阶项，可能work.

替换方法

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：

- $T(n)\le c\left\lfloor\frac{n}{2}\right\rfloor+c\left\lceil\frac{n}{2}\right\rceil+1=cn+1\ne cn$
- 证明不出$T(n)=O(cn)$

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

证明：假设当$\le n-1$时成立，则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。

替换方法

避免陷阱

错误

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

解：猜测$T(n) = O(n)$,用数学归纳法证明$T(n) \leq cn$

$$T(n) \leq 2\left(c\left\lfloor \frac{n}{2} \right\rfloor\right) + n \leq cn + n = O(n)$$

错在哪里：过早地使用了$O(n)$而陷入了陷阱，应该在证明了$T(n) \leq cn$之后，才可用。从$T(n) \leq cn + n$不可能得到$T(n) \leq cn$。因为对于任何$c > 0$，我们都得不到$cn + n \leq cn$。

替换方法

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$T(n) = 2 * T(\sqrt{n}) + \lg n$

解:

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

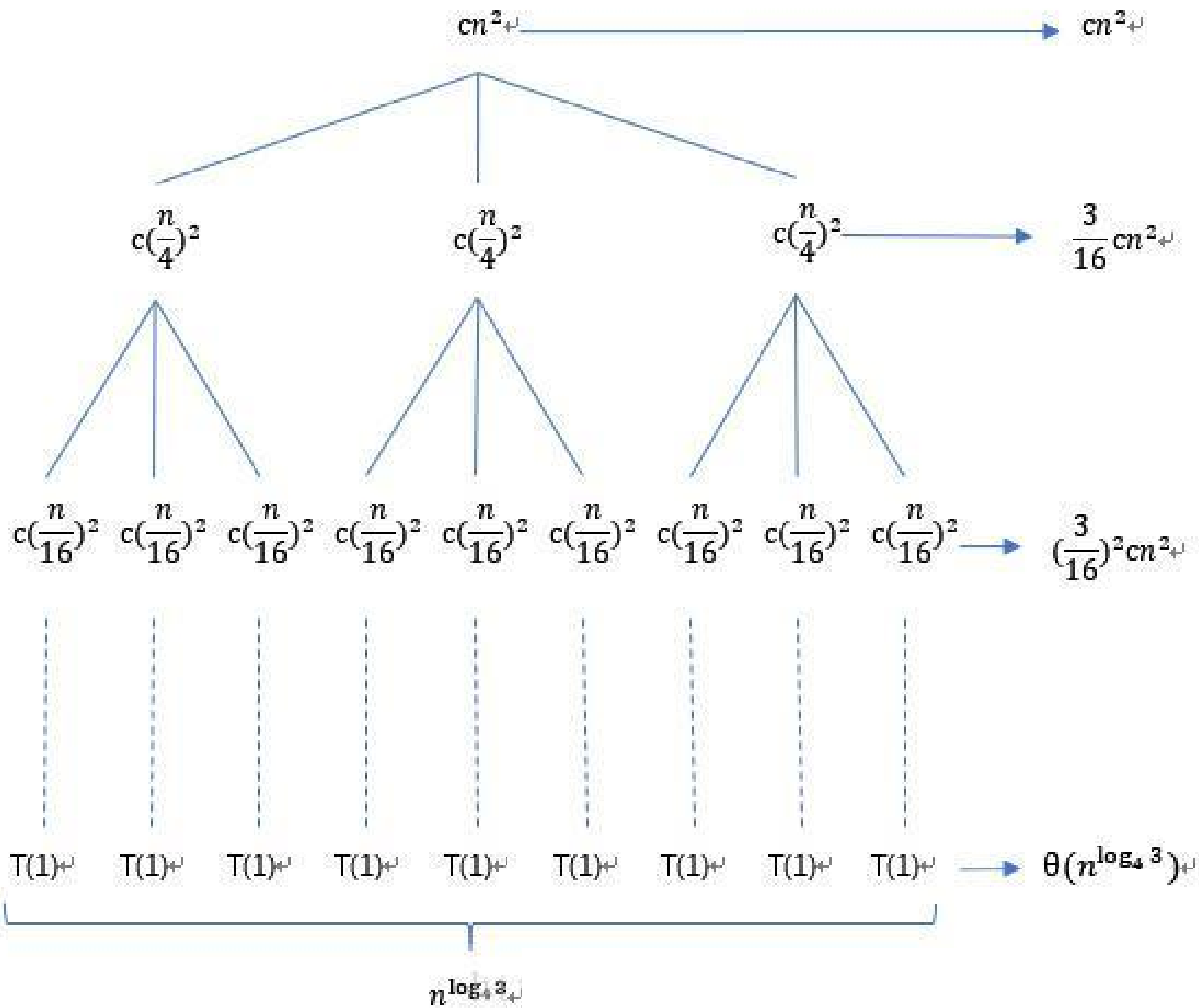
- 画出递归树
- 循环地展开递归方程
- 把递归方程转化为和式
- 然后可使用求和技术解之

递归树

- 根结点表示递归调用顶层的代价
- 内部节点，表示合并其子问题的代价
- 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. $T(n) = 3 * T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$



递归树

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

解:

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

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

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

我们得出$T(n) = O(n^2)$

递归树

例2. $T(n) = n + 3T(\lfloor {}^{n}/_{4} \rfloor)$

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

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

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

令 $\boxed{{}^{n}/_{4^i} = 1 \Rightarrow 4^i = n \Rightarrow i = \log_4^n}$

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

$$\leq \sum_{i=0}^{\log_4^n} 3^i\, {}^{n}/_{4^i} + O(n) \leq n \sum_{i=0}^{\infty} \left({}^{3}/_{4}\right)^i = n \times \frac{1}{1-\frac{3}{4}} = 4n = O(n)$$

Master定理方法

目的：求解 $T(n)=aT\left(n/b\right)+f(n)$ 型方程，$a\geq 1, b>0$ 是常数，$f(n)$ 是正函数

方法：记住三种情况，则不用笔纸即可求解上述方程.

Master 定理

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 $T(n) = aT\left(\frac{n}{b}\right) + f(n)$．$T(n)$ 可以如下求解：

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

(2)．若 $f(n) = \theta\left(n^{\log_b a}\right)$，则 $T(n) = \theta\left(n^{\log_b a} \lg n\right)$．

(3)．若 $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$，$\varepsilon > 0$ 是常数，且对于所有充分大的 n

$af\left(\frac{n}{b}\right) \leq cf(n)$，c<1 是常数，则 $T(n) = \theta(f(n))$．

Master 定理

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

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

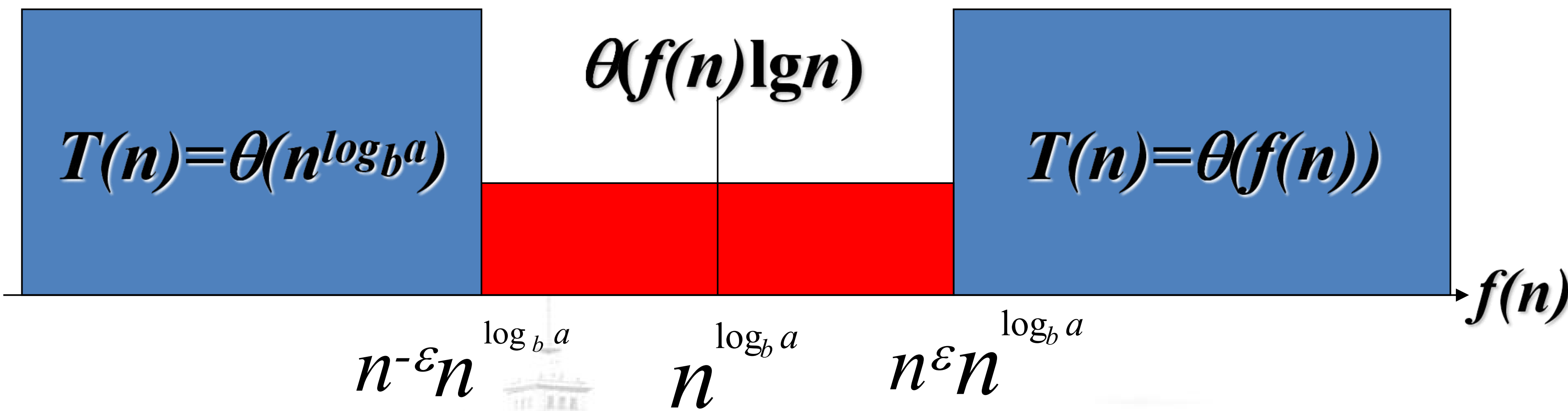
a ： 子问题数量
b ： 子问题从原问题缩小的比例
f(n)： 将子问题的解，整合的代价

*直观地：我们用 $f(n)$ 与 $n^{\log_b a}$ 比较

(1). 若 $n^{\log_b a}$ 大，则 $T(n)=\theta\left(n^{\log_b a}\right)$

(2). 若 $f(n)$ 大，则 $T(n)=\theta(f(n))$

(3). 若 $f(n)$ 与 $n^{\log_b a}$ 同阶，则 $T(n)=\theta\left(n^{\log_b a}\lg n\right)=\theta(f(n)\lg n)$.



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 $\varepsilon > 0$， $f(n) = O(\frac{n^{\log_b a}}{n^{\varepsilon}})$.

(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，即对一个常数 $\varepsilon > 0$， $f(n) = \Omega(n^{\log_b a} \cdot n^{\varepsilon})$.

Master定理的使用

例 1. 求解 $T(n)=9T\left(n/3\right)+n$.

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

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

$$\because f(n)=n=O\left(n^{\log_b a-\varepsilon}\right),\quad \varepsilon=1$$

$$\therefore T(n)=\theta\left(n^{\log_b a}\right)=\theta\left(n^2\right)$$

例 2. 求解 $T(n)=T\left(2n/3\right)+1$.

(2). 若 $f(n)=\theta\left(n^{\log_b a}\right)$， 则 $T(n)=\theta\left(n^{\log_b a}\lg n\right)$.

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

$$f(n)=1=\theta(1)=\theta\left(n^{\log_b a}\right),\quad T(n)=\theta\left(n^{\log_b a}\lg n\right)=\theta(\lg n)$$

Master定理的使用（续）

例 3． 求解 $T(n)=3T\left(n/4\right)+n\lg n$

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

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

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

(2) 对所有 **n**，$af\left(n/b\right)=3\times\frac{n}{4}\lg\frac{n}{4}=\frac{3}{4}n\lg\frac{n}{4}\le\frac{3}{4}n\lg n=cf(n)$，$c=\frac{3}{4}$．

于是，$T(n)=\theta(f(n))=\theta(n\lg n)$

例 4． 求解 $T(n)=2T\left(n/2\right)+n\lg n$．

Master定理失效

解： $a=2$，$b=2$，$f(n)=n\lg n$，$n^{\log_b a}=n$．$f(n)=n\lg n$ 大于 $n^{\log_b a}=n$，但不是多项式**地**大于，Master 定理不**适**用于该 $T(n)$．

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（**ABCD**）

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

A $O(n)$

B $O(n^2)$

C $O(n\log_2 n)$

D $O(\log_2 n)$

提交

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

3、已知两个长度分别为m和n的升序链表，若将他们合并为一个长度为m+n的降序链表，则最坏情况下的时间复杂度是（AD）

A. $\Omega(\max(m,n))$

B. $O(m\times n)$

C. $O(\min(m,n))$

D. $O(\max(m,n))$

提交

多选题 1分

设置

4、下列说法正确的是（AB）

A. 如果函数f(n)是O(g(n))，g(n)是O(h(n))，那么f(n)是O(h(n))

B. 如果函数f(n)是O(g(n))，g(n)是O(h(n))，那么f(n)+g(n)是O(h(n))

C. 如果a>b>1,$\log_a n$是O($\log_b n$)，但$\log_b n$不一定是O($\log_a n$)

D. 函数f(n)是O(g(n))，当常数a足够大时，一定有函数g(n)是O(af(n))

提交

单选题 1分

设置

5、 $f(n)+o(f(n))=\theta(f(n))$ 是否正确（A)

A. 正确

B. 错误

提交

单选题 1分

设置

6、$1+\frac{1}{2}+\cdots+\frac{1}{n}=o(n)$ 是否正确（A）

A. 正确

B. 错误

提交