第一章 算法分析的数学基础

东南大学计算机学院 金嘉晖

本章内容

- 算法复杂性的度量
- 复杂性函数的阶
- 和的估计与界限
- 递归方程

解决算法问题的步骤

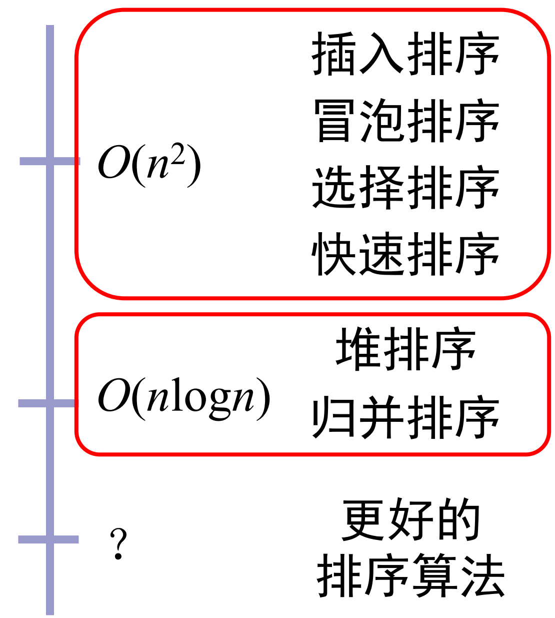
- 理解问题的背景和领域
- 形式化描述问题
- 求解问题（即算法设计）
- 评价算法
  - 算法的优化目标是否达到（准确性）
  - 求解的性能是否好（算法分析）

算法的正确性分析

- 一个算法是正确的，如果它对于每一个输入都最终停止,而且产生正确的输出
  - 不正确算法：
    - 不停止(在某个输入上)
    - 对所有输入都停止，但对某输入产生不正确结果
  - 近似算法
    - 对所有输入都停止
    - 产生近似正确的解或产生不多的不正确解
  - 调试程序 ≠ 程序正确性证明
    - 程序调试只能证明程序有错，
    - 不能证明程序无错误!

算法计算复杂度分析

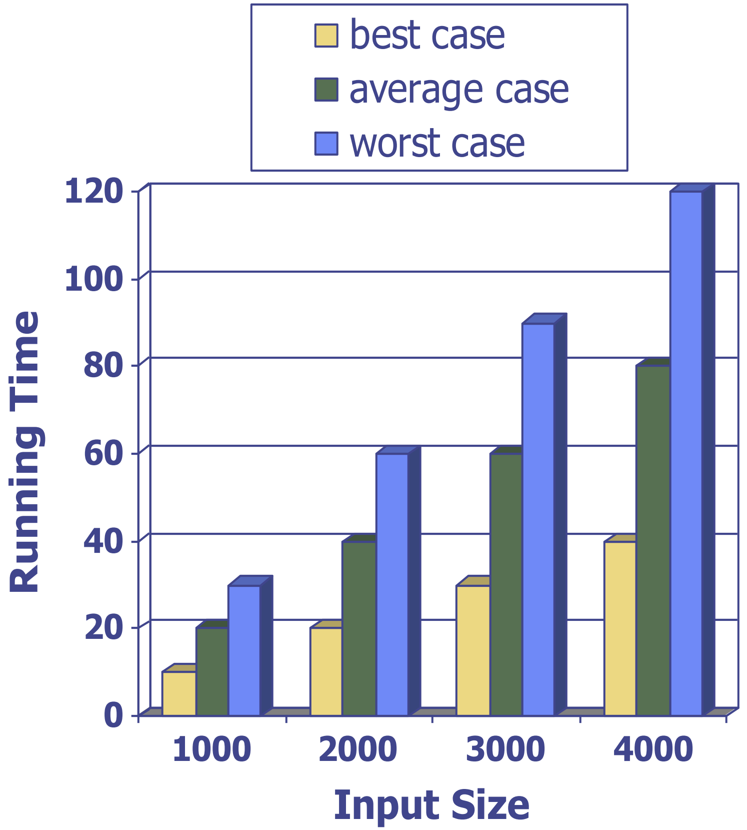
- **问题**
  - **哪个排序算法效率最高?**
  - **是否可以找到更好的排序算法?**
  - **排序问题计算难度如何?**
  - **问题计算复杂度的估计方法**



哪个排序算法效率最高?
如何分析排序问题计算难度?

算法的分析（运行时间）

- 大多数算法是将输入转化成输出的过程
- 算法的运行时间通常随着输入数据的规模而发生变化
- 虽然平均的运行时间能较准确刻画算法性能，但是平均时间较难计算
- 我们通常考虑最坏情况下执行时间
  - 比较容易计算
  - 对于计算/存储密集型应用（如游戏、商业分析、机器人）而言，非常重要



评估算法的执行效率

- 两种评估方法：
  - **经验(Empirical)**：对各种算法编程，用不同实例进行实验；
  - **理论(Theoretical)**：以数学化的方式确定算法所需要资源数与实例大小之间函数关系。

算法效率→算法的快慢

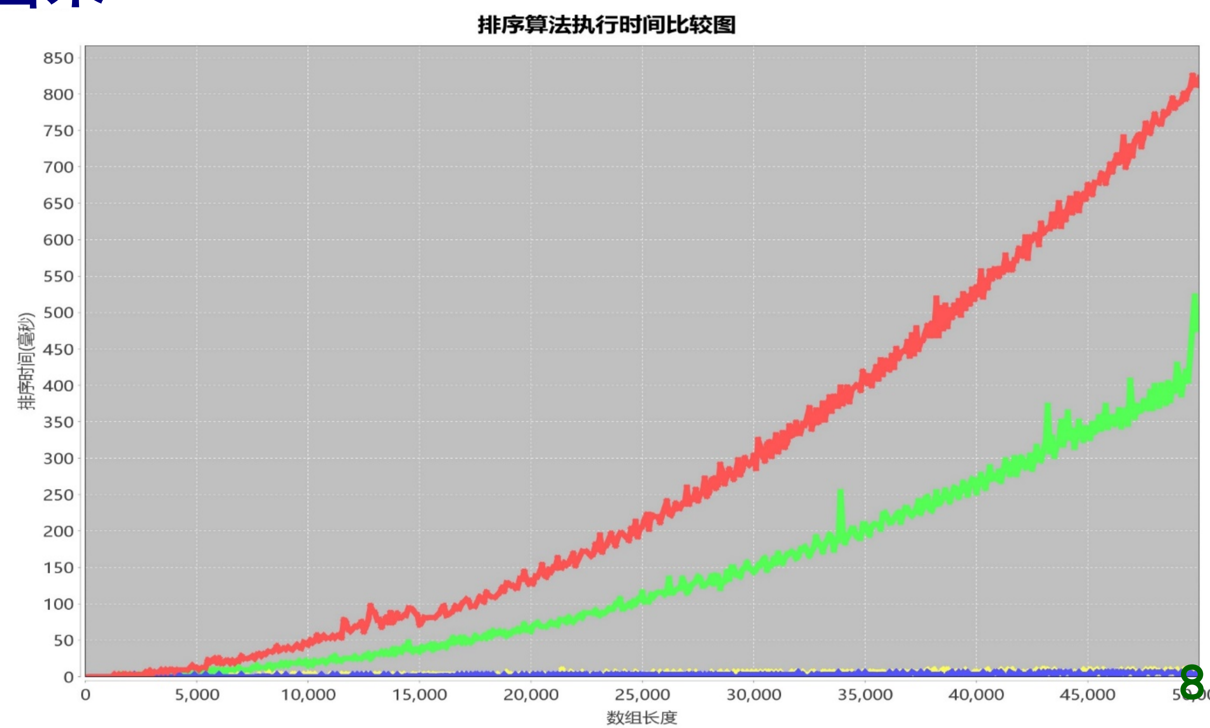
★ 时间/空间

某些方法实例中某些构件数的数量

- 排序：以参与排序的项数表示实例大小；
- 讨论图时，常用图的节点/边来表示

基于经验的评估方法

- 实验方案
  - 生成n个随机数并进行排序（n=100,200,…,25000）
  - 记录各个算法的排序时间
  - 用图的形式画出来



经验法存在的问题

- 经验法的问题
  - 依赖于计算机
  - 依赖于语言/编程技能
  - 需要一定的编程/调试时间
  - 只能评估部分实例的效率

理论法优点：既不依赖于计算机，也不依赖于语言/编程技能。节省了无谓编程时间；可研究任何在实例上算法效率

基于理论的评估方法

- 执行时间的估计
  - 定义评估函数$T(n)$，n为输入数据规模
  - 如果存在一个正的常数c，而该算法对每个大小为n的实例的执行时间都不超过$cT(n)$ 秒
  - →该算法的开销在$T(n)$级内。

为什么定义常数c?

Apple I
CPU MOS 6502
@ 1 MHz

2017

iMac Pro
CPU Intel Xeon W
@ 3.2GHz 八核

算法好坏的衡量尺度

- 最初，用所需计算时间来衡量算法的好坏
- 但不同的机器相互之间无法比较
- 故需要用独立于具体计算机的客观衡量标准
  - 问题的规模
  - 基本运算
  - 算法的计算量函数

算法好坏的衡量尺度

- 时间复杂度
  - 基本运算（原子操作）执行次数
- 空间复杂度
  - 需要的存储空间大小

算法好坏的衡量尺度

- 问题的规模
  - 一个或多个整数，作为输入数据量的测度
  - 数组的长度 (数据项的个数)
    - 问题：在一个数组中寻找X
  - 矩阵的最大维数 (阶数)
    - 问题：求两个实矩阵相乘的结果
- 输入规模通常用n来表示
  - 也可有两个以上的参数，如图中的顶点数和边数 (图论中的问题)

算法好坏的衡量尺度

- 基本运算
  - 解决给定问题时占支配地位的运算
  - 在一个表中寻找数据元素x
    - x与表中的一个项进行比较
  - 两个实矩阵的乘法
    - 实数的乘法(及加法)C=AB则$c_{ij}=\sum a_{ik}*b_{kj}$
  - 将一个数组进行排序
    - 数组中的两个数据项进行比较

算法好坏的衡量尺度

- 基本运算
  - 通常情况下，讨论一个算法优劣时，我们只讨论基本运算的执行次数
  - 因为它是占支配地位的，而其它的运算可以忽略不计

算法好坏的衡量尺度

- 算法的计算量函数
  - 用输入规模的某个函数来表示算法的基本运算量
  - 该函数称为算法的时间复杂性(度)，一般用T(n)或T(n,m)等表示
    - $T(n)=5n$，$T(n)=3n*\log n$，
    - $T(n)=4n^3$，$T(n)=2^n$，
    - $T(n,m)=2(n+m)$

最坏情况时间复杂性

- 规模为n的所有输入中，基本运算执行次数最多的时间复杂性
  - 在一个顺序表中寻找数据元素x
    - 顺序查找：最坏情况为O(n)；
    - 二分查找：最坏情况为O(logn)

平均情况时间复杂性

- 规模为n的所有输入的算法时间复杂度的平均值（一般均假设每种输入情况以等概率出现）
  - 在一个顺序表中寻找数据元素x
    - 顺序查找：平均情况仍为O(n)；
    - 二分查找：平均情况仍为O(logn)

算法的伪代码

- 算法的抽象表示
- 比自然语言更加准确
- 比程序语言更加自由
- 经常喜欢用数学符号来描述
- 允许隐藏编程的细节（比如变量的类型、模板定义等）

Example: 找出数组中最大的元素

Algorithm *arrayMax*(*A*, *n*)
  Input array *A* of *n* integers
  Output maximum element of *A*

  currentMax ← *A*[0]
  for *i* ← 1 **to** *n* − 1 **do**
    if *A*[*i*] > *currentMax* **then**
      currentMax ← *A*[*i*]
  return *currentMax*

伪代码的组成

- 控制语句
  - if … then … [else …]
  - while … do …
  - repeat … until …
  - for … do …
  - 可以用缩进代替花括号
- 算法声明Method declaration

  Algorithm *method* (*arg* [, *arg*…])

  Input …

  Output …

- 方法调用

  var.method (*arg* [, *arg*…])
- 返回值

  return *expression*
- 表达式

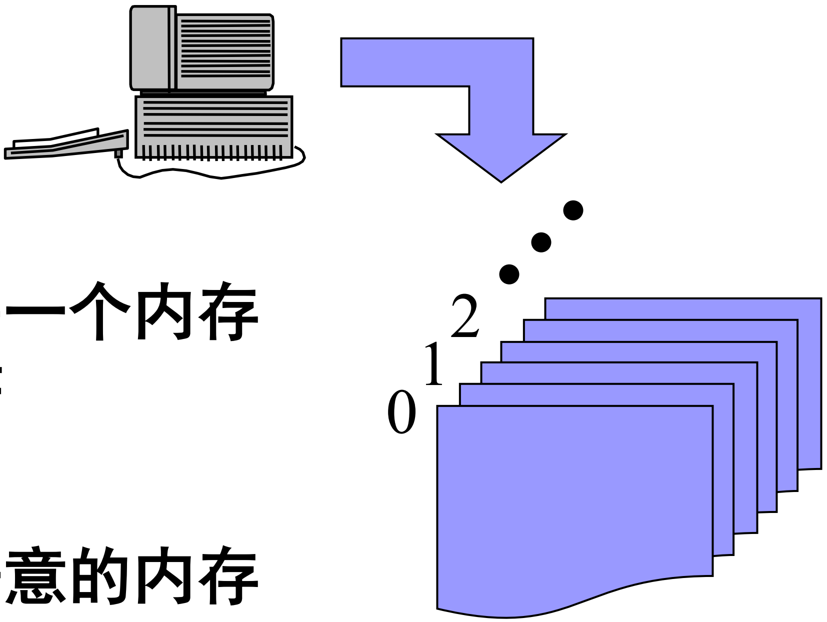
  ← 赋值 (like = in Java)

  = 是否相等 (like == in Java)

  n^2 允许上标或其它数学表达

计算机的抽象模型

- 拥有一个CPU
- 一个无限大的内存空间，每一个内存单元可以存放数字或者字符
- 内存单元是连续编号的，任意的内存单元都可以在一个时间片内被访问到
- Random Access Machine （RAM）



基本运算

- 算法中最为基本的运算
- 在伪代码中很容易识别
- 与编程语言无关
- 假设在每个基本运算都在RAM模型中花费一定的时间
- 无需精确衡量执行多少时间

- 例如:
  - 计算表达式
  - 赋值
  - 访问数组元
  - 调用一个方法
  - 返回一个值

数基本运算的次数（理论分析）

- 通过分析算法的伪代码，可以得到基本运算的次数和算法输入规模之间的函数关系

Algorithm *arrayMax*(A, n)	# operations
$currentMax \leftarrow A[0]$	2
for $i \leftarrow 1$ to $n-1$ do	$2+n$
if $A[i] > currentMax$ then	$2(n-1)$
$currentMax \leftarrow A[i]$	$2(n-1)$
{ increment counter i }	$2(n-1)$
return $currentMax$	1
Total	$7n-1$

估算算法的运行时间

- 算法 *arrayMax* 在最坏情况下执行了 $7n-1$ 次基本运算，令：
  - a 为执行速度最快的基本运算
  - b 为执行速度最慢的基本运算
- 令 $T(n)$ 为*arrayMax*的最长运行时间，则

$$\boldsymbol{a\,(7n-1) \le T(n) \le b(7n-1)}$$

- 所以$T(n)$ 值的上界和下界可以确定

运行时间的增长率

- 改变软硬件环境可以影响 $T(n)$ 的值
- 但不能影响$T(n)$的增长率
- 对于算法*arrayMax* 而言，$T(n)$的增长率是线性的，不随软硬件的变化而改变

如何分析算法的增长率呢？

$T(N, I, A)$的概念

- **计算量函数依赖于问题的规模(N)，输入(I)和算法(A)本身，用C表示。**

  $C=F(N, I, A)$是一个三元函数。

时间T　　S空间复杂度

两者类似，但S简单

例子：利用插入排序对数组{5, 7, 1, 3, 6, 2, 4}排序

N ⇒ 7

I ⇒

5	7	1	3	6	2	4

A ⇒ `void insertSort(T[] a)`

$T(N, I, A)$的概念

- **计算量函数依赖于问题的规模(N)，输入(I)和算法(A)本身，用C表示。**

  $C=F(N, I, A)$是一个三元函数。

能否将计算量函数$T(N, I, A)$简化

简化：将A隐去

通常研究$T(N,I)$在一台抽象计算机上运行所需时间

$T(N, I)$的概念

- 设抽象计算机的元运算有k种，记为$O_1, \ldots, O_k$，每执行一次所需时间为$t_1, \ldots, t_k$。
- 对算法A，用到元运算O_i的次数为e_i与N，I相关。

$$T(N,I)=\sum_{i=1}^{k} t_i e_i(N,I)$$

能否将计算量函数$T(N, I)$简化

$T(N)$的概念

- 不可能规模N的每种合法输入I都去统计$e_i(N, I)$，对于 I 分别考虑：

最坏情况、最好情况、平均情况

$$T_{\max}(N)=\max_{I\in D_N} T(N,I)=\max_{I\in D_N}\sum_{i=1}^{k} t_i e_i(N,I)=\sum_{i=1}^{k} t_i e_i(N,I^*)=T(N,I^*)$$

$$T_{\min}(N)=\min_{I\in D_N} T(N,I)=\min_{I\in D_N}\sum_{i=1}^{k} t_i e_i(N,I)=\sum_{i=1}^{k} t_i e_i(N,\tilde{I})=T(N,\tilde{I})$$

$$T_{avg}(N)=\sum_{I\in D_N} P(I)T(N,I)=\sum_{I\in D_N} P(I)\sum_{i=1}^{k} t_i e_i(N,I)$$

合法输入集

$T(N)$的概念

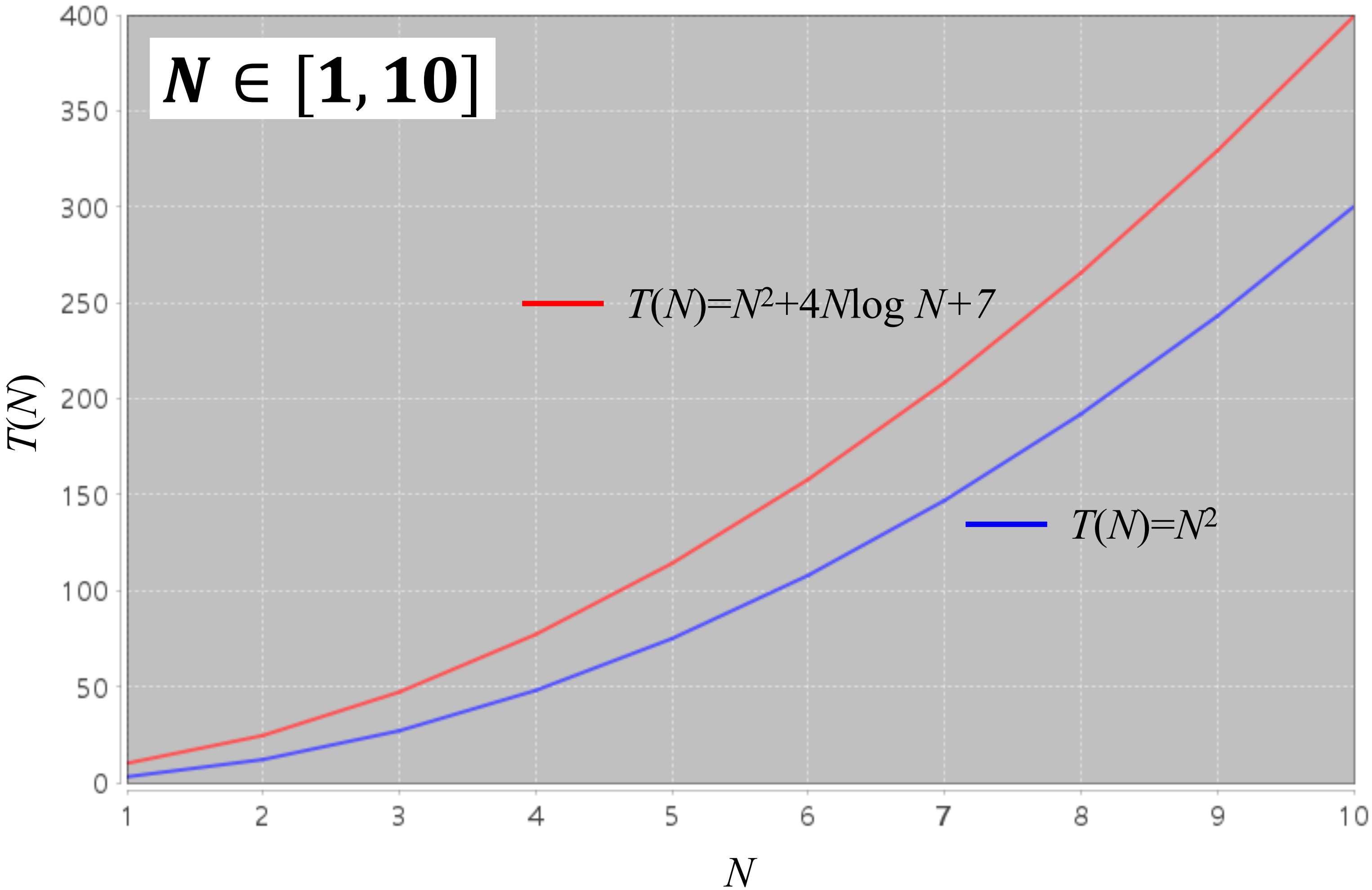
- **进一步简化：**假设算法中用到的所有不同基本运算各执行一次需要的时间都是一个单位时间。
- 用输入规模的某个函数来表示算法的基本运算量，称为**算法的时间复杂性(度)**。

用$T(N)$或$T(N, M)$来表示，例如：

- $T(N)=5N+3$
- $T(N)=3N\log N+2N$
- $T(N)=4N^3+3N+2$
- $T(N)=2^N$
- $T(N, M)=2(N+M)$

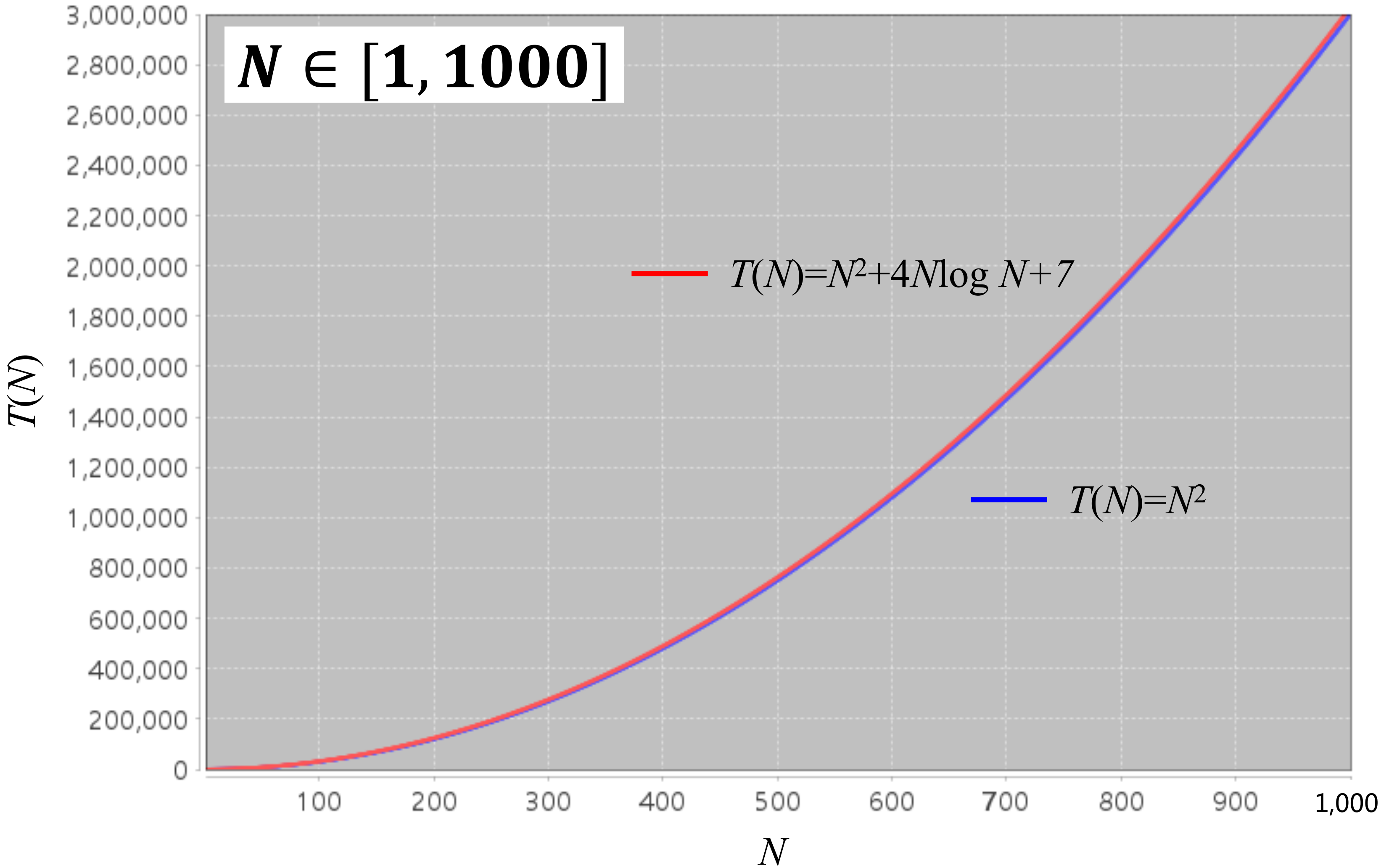
能否将计算量函数$T(N)$简化

复杂性渐进性态



$N \in [1, 10]$
$T(N)=N^2+4N\log N+7$
$T(N)=N^2$
$T(N)$
N
400
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8
9
10

复杂性渐进性态



$N \in [1, 1000]$
$T(N)=N^2+4N\log N+7$
$T(N)=N^2$
3,000,000
2,800,000
2,600,000
2,400,000
2,200,000
2,000,000
1,800,000
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
100
200
300
400
500
600
700
800
900
1,000
$T(N)$
N

复杂性渐进性态

- 设 $T(N)$ 是前面定义的算法 A 复杂性函数。
  - N 递增到无限大，$T(N)$ 递增到无限大
  - 如存在 $\tilde{T}(N)$，使 $N\to\infty$ 时，有 $\dfrac{T(N)-\tilde{T}(N)}{T(N)}\to 0$

称 $\tilde{T}(N)$ 是 $T(N)$ 当 $N\to\infty$ 的渐进性态。

- 在数学上，$\tilde{T}(N)$ 是 $T(N)$ 当 $N\to\infty$ 的渐进表达式，通常 $\tilde{T}(N)$ 是 $T(N)$ 中略去低阶项所留下的主项。
  - $\tilde{T}(N)$ 比 $T(N)$ 简单。

复杂性渐进性态

- 例如： $T(N)=3N^2+4N\log N+7$

  $\tilde{T}(N)=3N^2$

- 由 $\dfrac{T(N)-\tilde{T}(N)}{T(N)}=\dfrac{4N\log N+7}{3N^2+4N\log N+7}\to 0$
- 因为 $N\to\infty$， $T(N)\to\tilde{T}(N)$
- 所以有理由用 $\tilde{T}(N)$ 来替代 $T(N)$ 来度量A。

复杂性渐进性态

- 当比较两个算法的渐近复杂性的阶不同时，只要确定各自的阶，即可判定哪个算法效率高。

等价于

  - 只要关心$\widetilde{T}(N)$的阶即可，不必考虑其中常数因子。

- 简化算法复杂性分析的方法和步骤，只要考察问题规模充分大时，算法复杂性在渐近意义下的阶。

练习：按照渐近阶从低到高的顺序排列以下表达式：

$$n!,\ 4n^2,\ \log n,\ 3^n,\ 20n,\ 2,\ n^{2/3}$$

复杂性渐进性态

- 当比较两个算法的渐近复杂性的阶不同时，只要确定各自的阶，即可判定哪个算法效率高。

等价于

  - 只要关心$\widetilde{T}(N)$的阶即可，不必考虑其中常数因子。

- 简化算法复杂性分析的方法和步骤，只要考察问题规模充分大时，算法复杂性在渐近意义下的阶。

$$\widetilde{T}(N)=3N^2$$

是否可以去掉系数?

能否将计算量函数$\tilde{T}(N)$简化

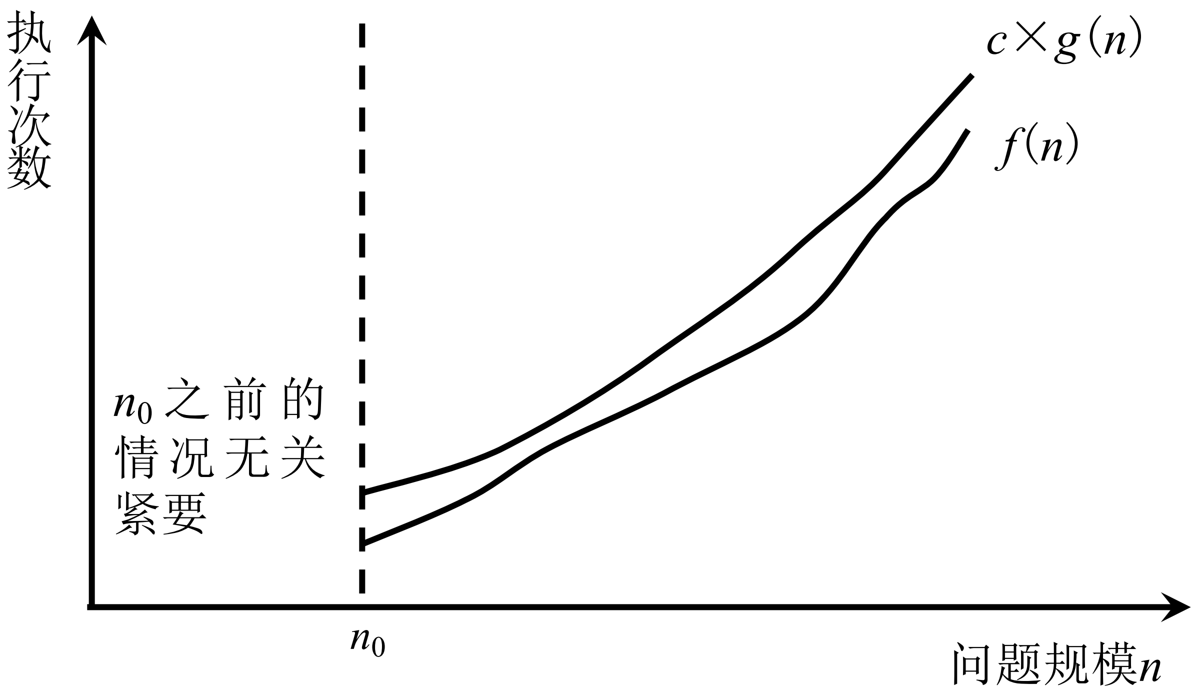
渐近分析的记号

- 渐近上界记号 O
- 渐近下界记号 Ω
- 紧渐近界记号 Θ
- 非紧上界记号 o
- 非紧下界记号 ω

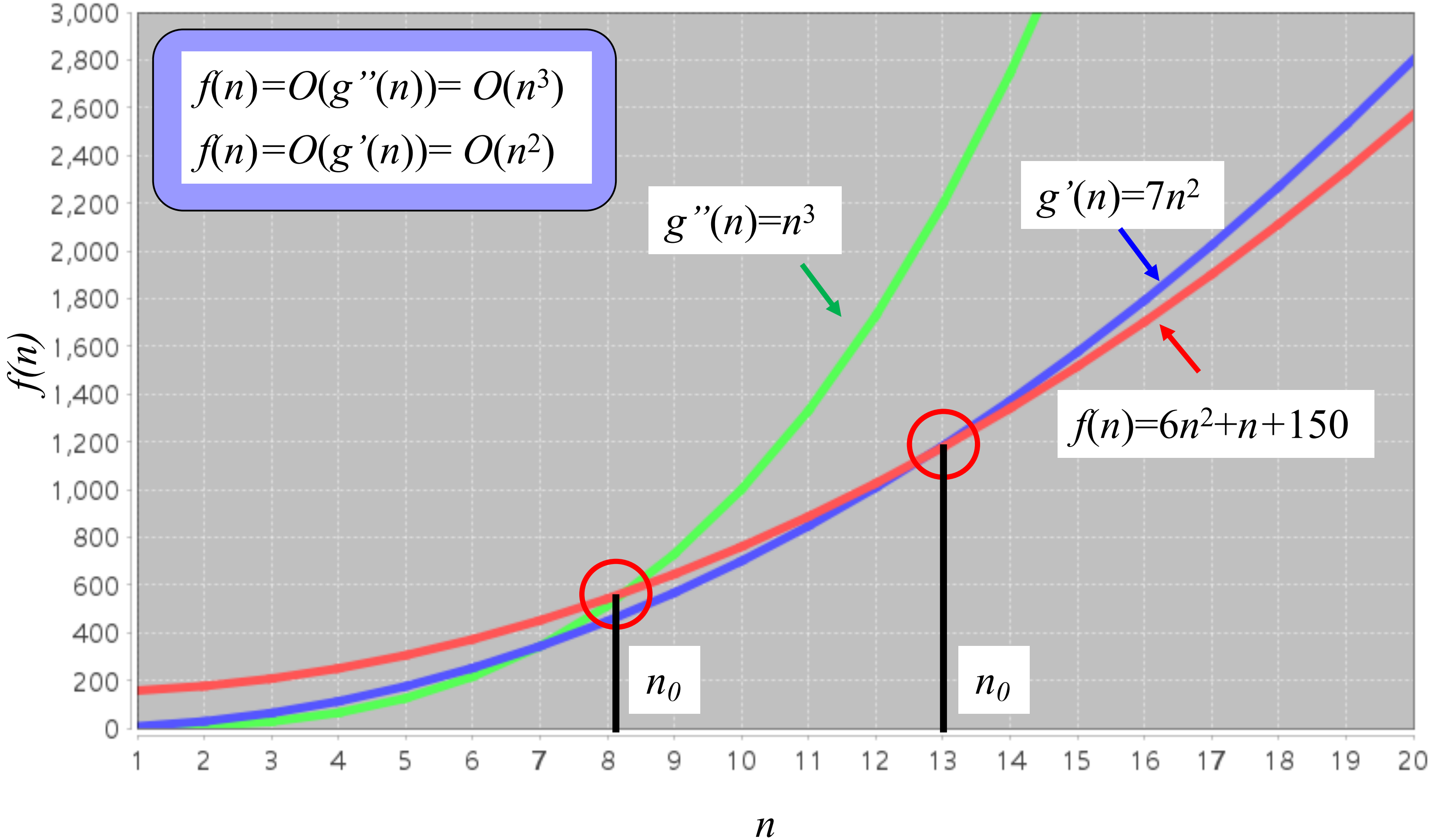
下面的讨论中，对所有n，$f(n)\geqslant 0$，$g(n)\geqslant 0$

渐近上界记号O

- 渐近上界记号O
  - 若存在两个正的常数c和n_0，使得对所有$n \geqslant n_0$，都有：$f(n) \leqslant c \times g(n)$，则称$f(n) = O(g(n))$



渐近上界记号O



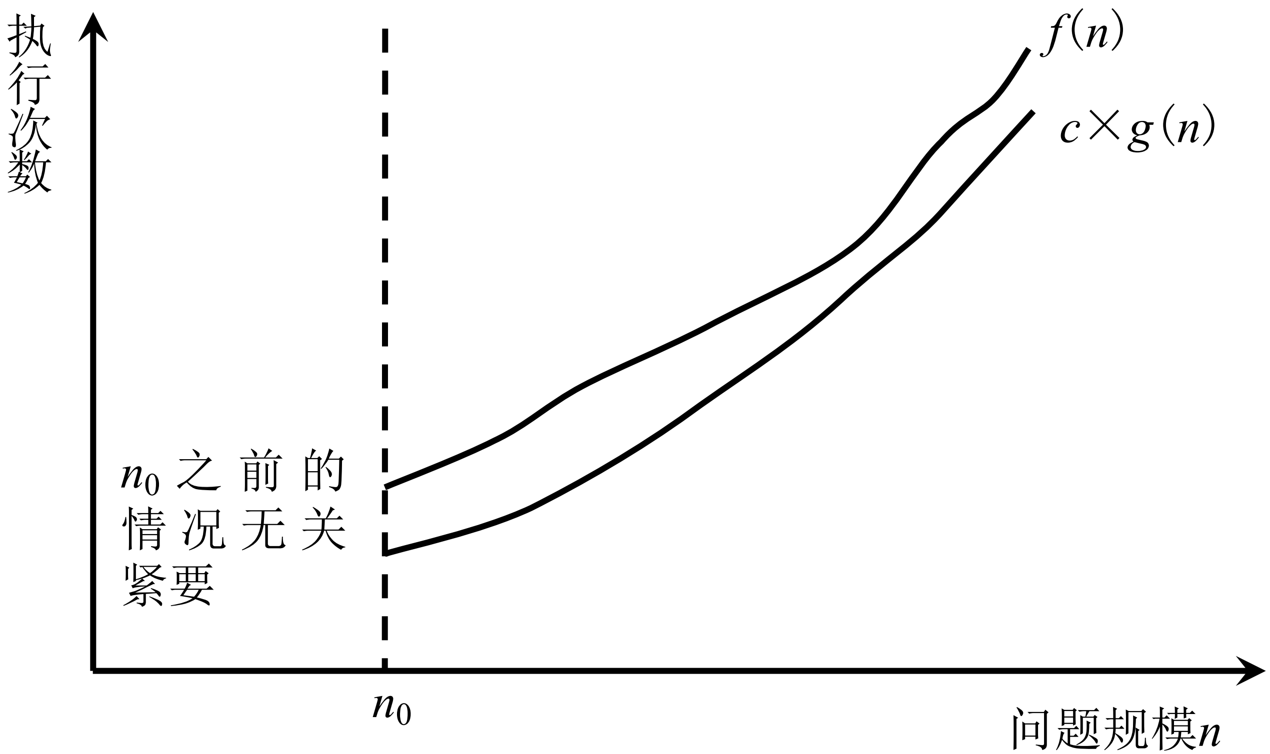
$f(n)=O(g''(n))= O(n^3)$
$f(n)=O(g'(n))= O(n^2)$
$g''(n)=n^3$
$g'(n)=7n^2$
$f(n)=6n^2+n+150$
n_0
n_0
$f(n)$
n
3,000
2,800
2,600
2,400
2,200
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

渐近上界记号O

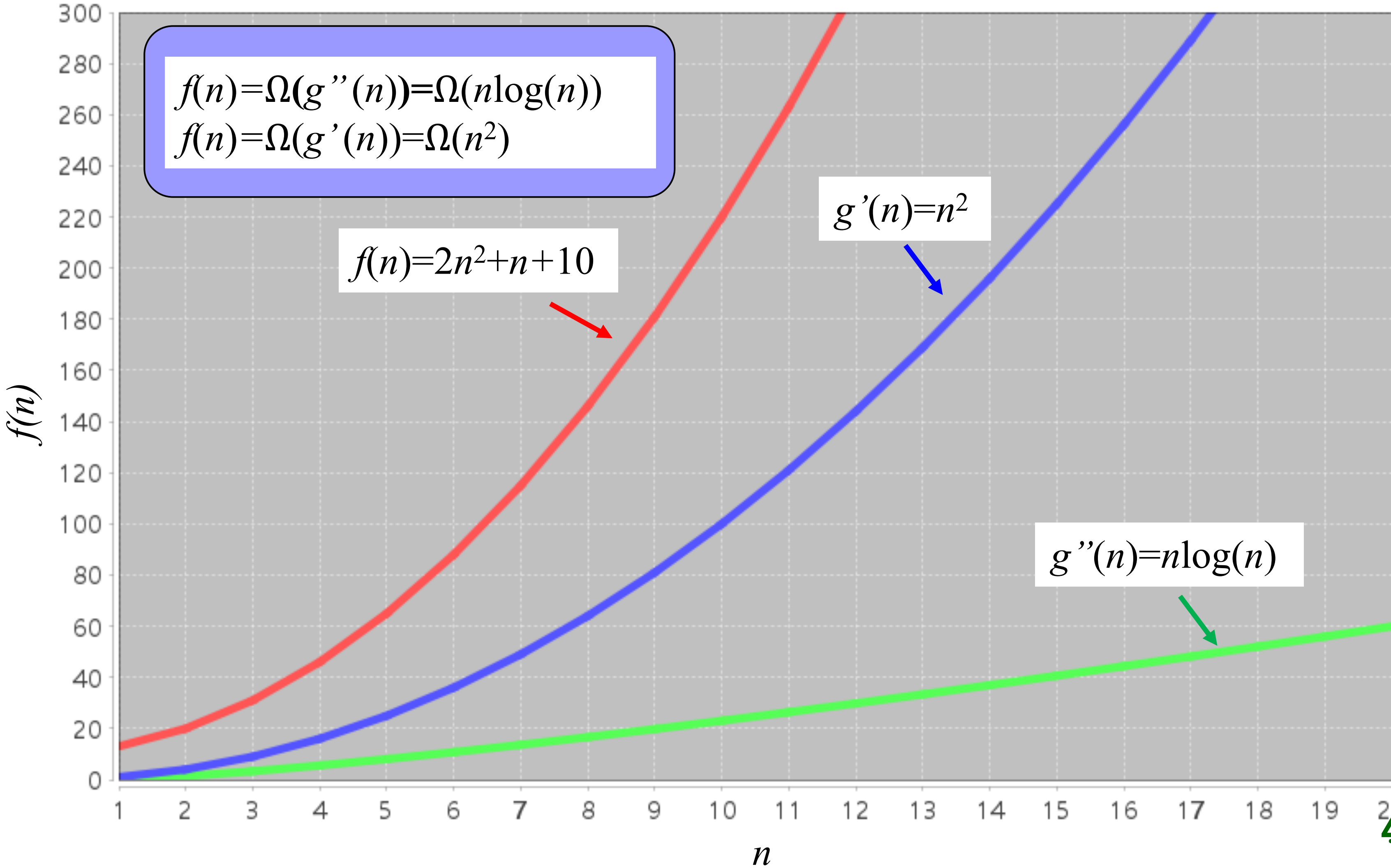
- 练习：求下列函数的渐近上界
  - $3n^2 + 10n$
  - $n^2/10 + 2^n$
  - $21 + 1/n$
  - $\log n^3$
  - $10\log 3^n$

渐近下界记号Ω

- 渐近下界记号Ω
  - 若存在两个正的常数c和n_0，使得对所有$n \geqslant n_0$，都有：$f(n) \geqslant c \times g(n)$，则称$f(n) = \Omega(g(n))$



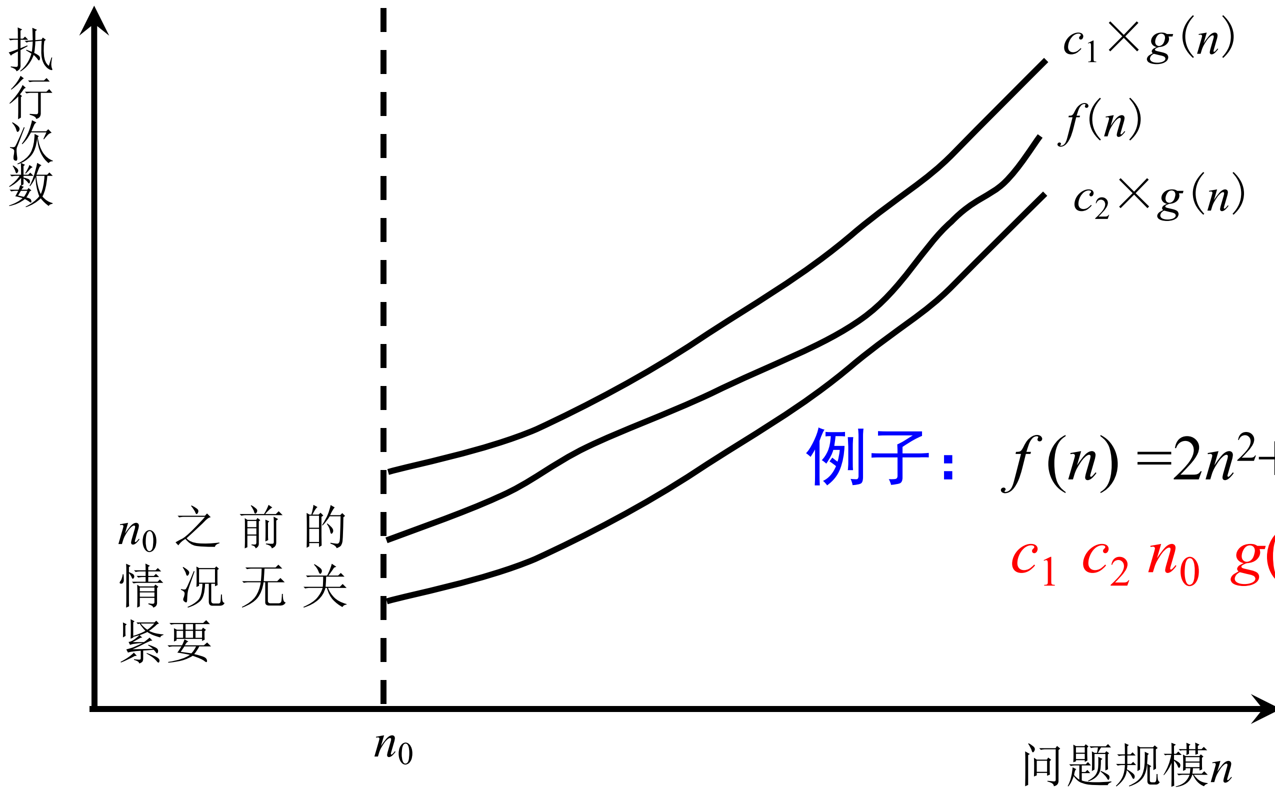
渐近分析的记号



$f(n)=\Omega(g''(n))=\Omega(n\log(n))$
$f(n)=\Omega(g'(n))=\Omega(n^2)$
$g'(n)=n^2$
$f(n)=2n^2+n+10$
$g''(n)=n\log(n)$
$f(n)$
n
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

紧渐近界记号Θ

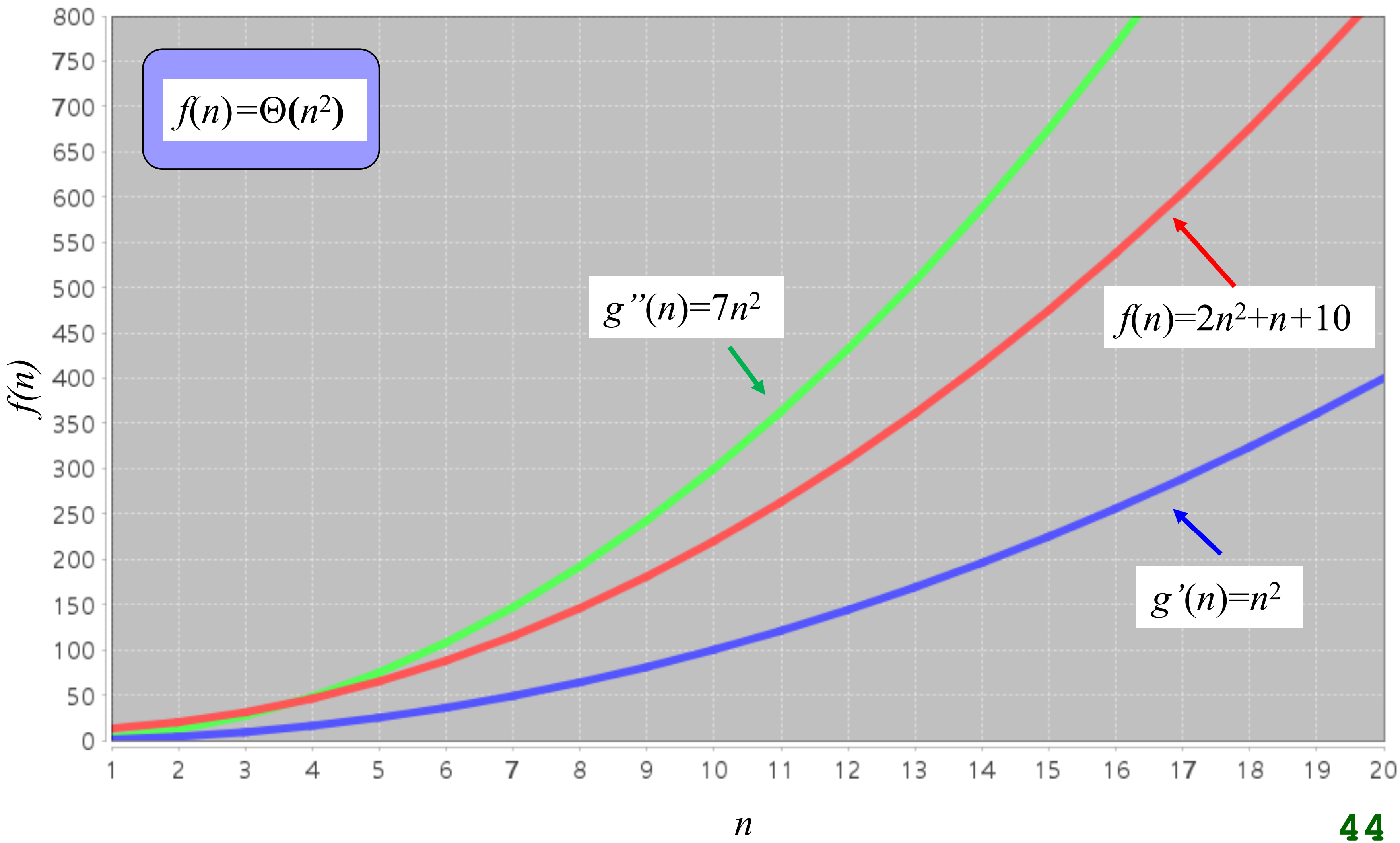
- 紧渐近界记号Θ
  - 若存在三个正的常数c_1、c_2和n_0，使得对所有$n \geqslant n_0$，都有：$c_1 \times g(n) \geqslant f(n) \geqslant c_2 g(n)$，则称 $f(n) = \Theta(g(n))$



例子：$f(n) = 2n^2+n+10$

c_1 c_2 n_0 $g(n)$ =?

紧渐近界记号Θ



f(n)=Θ(n2)
g''(n)=7n2
f(n)=2n2+n+10
g'(n)=n2
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
f(n)
n

非紧上/下界记号

- 非紧上界记号o
  - $o(g(n))=\{f(n)$ | 对于任何正常数$c>0$，存在正数$n_0>0$使得对所有$n \geqslant n_0$有：$0 \leqslant f(n)<cg(n)\}$
  - 等价于 $f(n)/g(n) \to 0$，as $n \to \infty$。
- 非紧下界记号ω
  - $\omega(g(n))=\{f(n)$ | 对于任何正常数$c>0$，存在正数$n_0>0$使得对所有$n \geqslant n_0$有：$0 \leqslant cg(n)<f(n)\}$
  - 等价于 $f(n)/g(n) \to \infty$，as $n \to \infty$。
  - $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

多项式时间与指数时间

- 设每秒可做某基本运算10^9次，n=60

	算法1	算法2	算法3	算法4	算法5	算法6
复杂度	n	n^2	n^3	n^5	2^n	3^n
运算时	$6*10^{-8}$s	$3.6*10^{-6}$s	$2.16*10^{-4}$s	0.013min	3.66世纪	$1.3*10^{13}$世纪

- 两个结论
  - 多项式时间的算法互相之间虽有差距，一般可接受
  - 指数量级时间的算法对于较大的n无实用价值

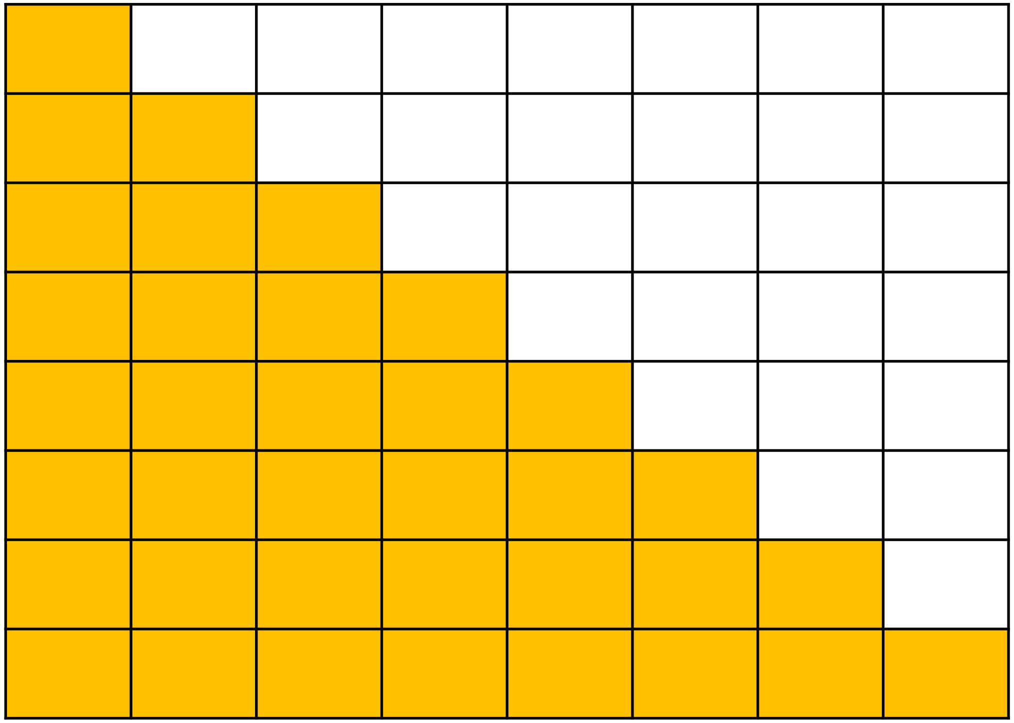
一些记号

- $\lfloor x \rfloor$表示小于等于x的最大整数
- $\lceil x \rceil$表示大于等于x的最小整数
  - $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
- $\log n = \log_2 n$， $\lg n = \log_2 n$
- $\ln n = \log_e n$

如何估计增长率？

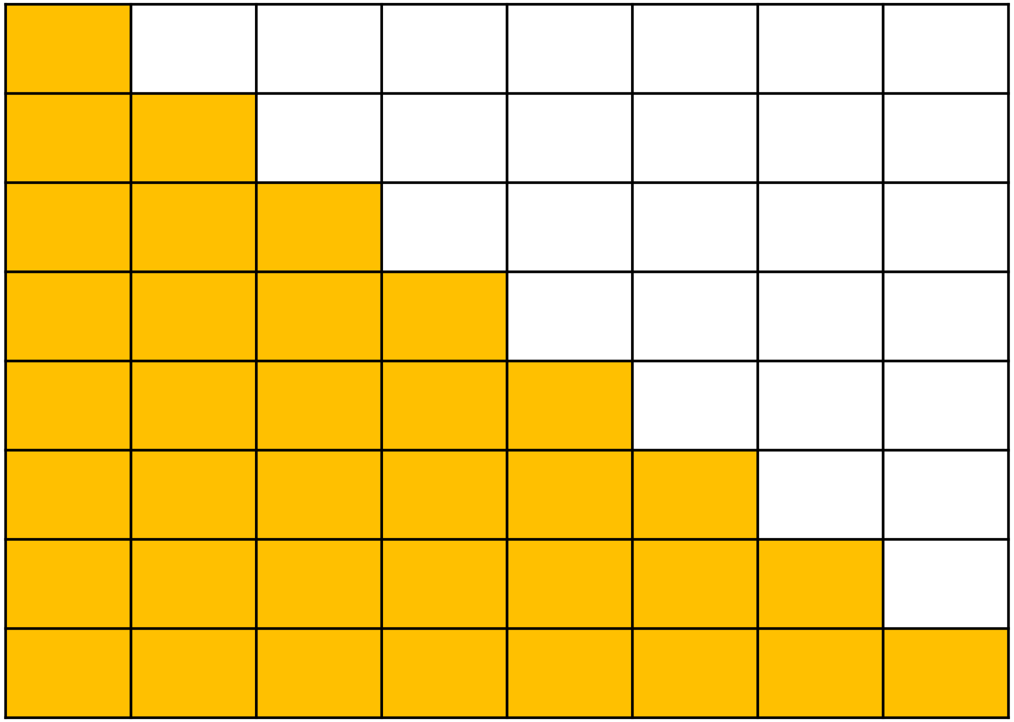
和的估计与界限

- 直接求和的界限
  - $\sum_{k=1}^{n} k$



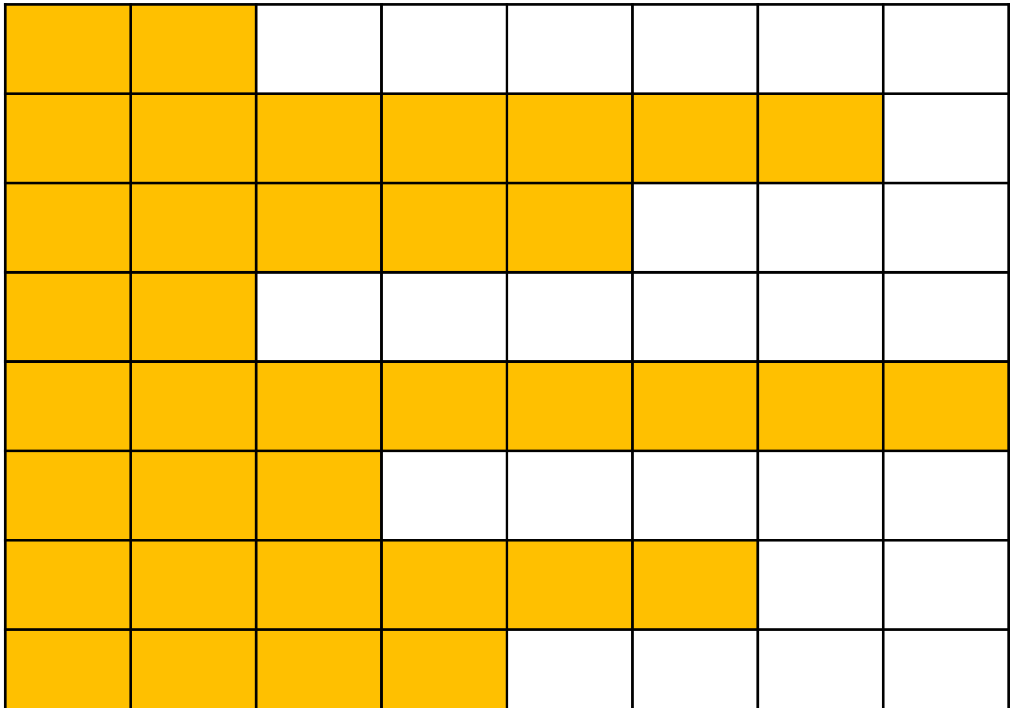
和的估计与界限

- 直接求和的界限
  - $\sum_{k=1}^{n} k \leq \sum_{k=1}^{n} n \leq n^2$



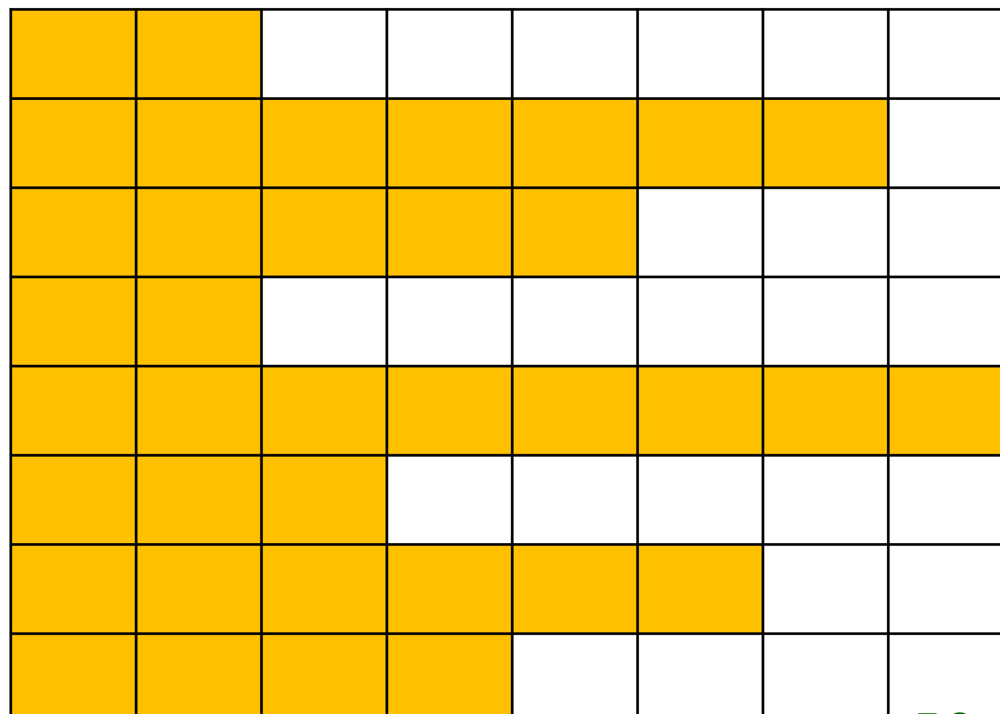
和的估计与界限

- **直接求和的界限**
  - $\sum_{k=1}^{n} a_k$



和的估计与界限

- 直接求和的界限
  - $\sum_{k=1}^{n} a_k \le n \max\limits_{1\le k\le n}\{a_k\}$



和的估计与界限

- 直接求和的界限
  - 对于所有$k \geq 0$，有$\frac{a_{k+1}}{a_k} \leq r < 1$，求$\sum_{k=1}^{n} a_k$上界

和的估计与界限

- 直接求和的界限
  - 对于所有$k \geq 0$，有$\frac{a_{k+1}}{a_k} \leq r < 1$，求$\sum_{k=1}^{n} a_k$上界
    - $\frac{a_1}{a_0} \leq r \to a_1 \leq a_0 r$
    - $\frac{a_2}{a_1} \leq r \to a_2 \leq a_1 r \leq a_0 r^2$
    - …
    - $\frac{a_{k+1}}{a_k} \leq r \to a_{k+1} \leq a_k r \leq a_0 r^k$
    - $\sum_{k=1}^{n} a_k \leq \sum_{k=1}^{n} a_0 r^k = a_0 \sum_{k=1}^{n} r^k = a_0 \frac{1-r^n}{1-r} \leq \frac{a_0}{1-r}$

和的估计与界限

- 直接求和的界限
  - 对于所有$k \geq 0$，有$\frac{a_{k+1}}{a_k} \leq r < 1$，求$\sum_{k=1}^{n} a_k$上界
  - 求$\sum_{k=1}^{\infty} k/3^k$上界
    - $\frac{\frac{k+1}{3^{k+1}}}{\frac{k}{3^k}} = \frac{1}{3}\frac{k+1}{k} \leq \frac{2}{3}$
    - $\sum_{k=1}^{\infty} k/3^k \leq \sum_{k=1}^{\infty} a_1 r^k = \sum_{k=1}^{\infty} \frac{1}{3}\left(\frac{2}{3}\right)^k = \frac{1}{3} \cdot \frac{1}{1-\frac{2}{3}} = 1$

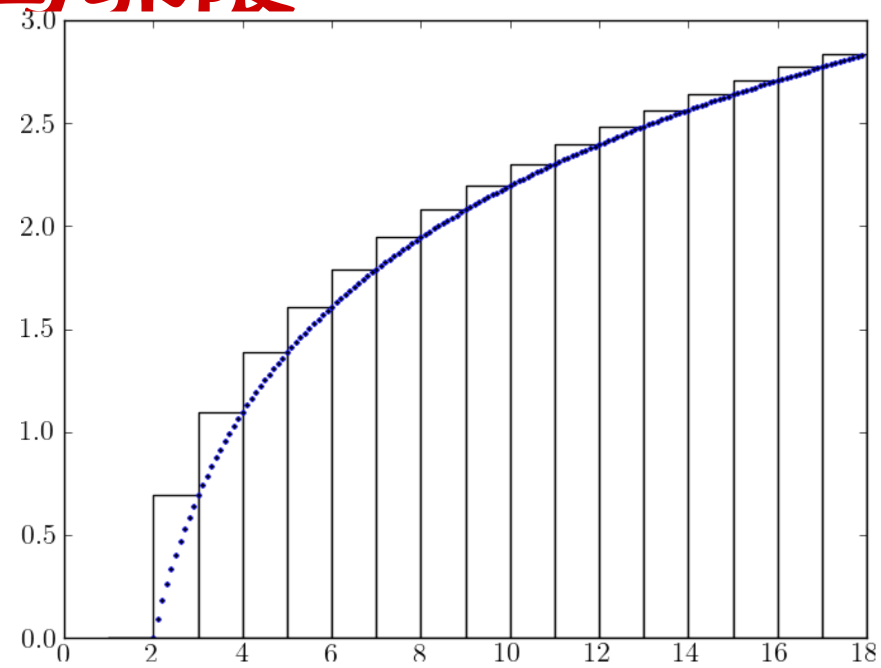
和的估计与界限

- 直接求和的界限
  - 对于所有$k \geq 0$，有$\frac{a_{k+1}}{a_k} \leq r < 1$，求$\sum_{k=1}^{n} a_k$上界
  - 求$\sum_{k=0}^{\infty} k^2/2^k$上界
    - 当$k \geq 3$时，有$\frac{\frac{(k+1)^2}{2^{k+1}}}{\frac{k^2}{2^k}} = \frac{1}{2}\frac{(k+1)^2}{k^2} \leq \frac{8}{9}$
    - $\sum_{k=0}^{\infty} k^2/2^k \leq \sum_{k=0}^{2} \frac{k^2}{2^k} + \sum_{k=3}^{\infty} \frac{k^2}{2^k} \leq \sum_{k=0}^{2} \frac{k^2}{2^k} + \sum_{k=3}^{\infty} \frac{9}{8}\left(\frac{8}{9}\right)^k = O(1)$

和的估计与界限

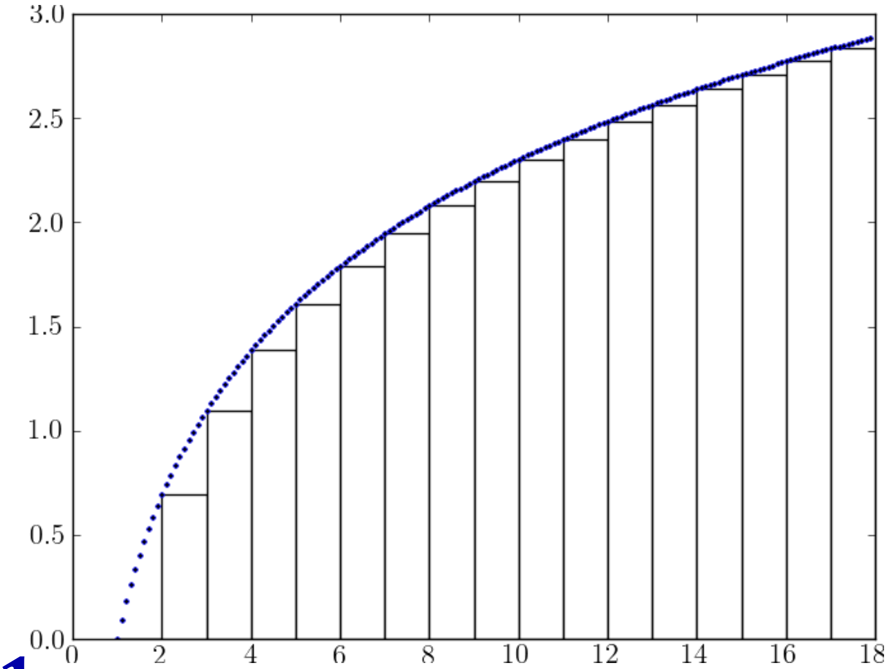
- 求和转换为求积分
  - $\log n! = \Omega(?)$
  - $\log n! = O(?)$

和的估计与界限



- 求和转换为求积分
  - $\log n! = \sum_{i=1}^{n} \log i$
  - 曲线之下面积
  - $\therefore \log n! > \int_1^n \log x \, dx$
  - $\because \log x = \log e \ln x$
  - $\int_1^n \ln x \, dx = n \ln n - n + 1$
  - $\therefore \log n! > (n \ln n - n + 1) \log e$
  - $\log n! = \Omega(n \log n)$

和的估计与界限



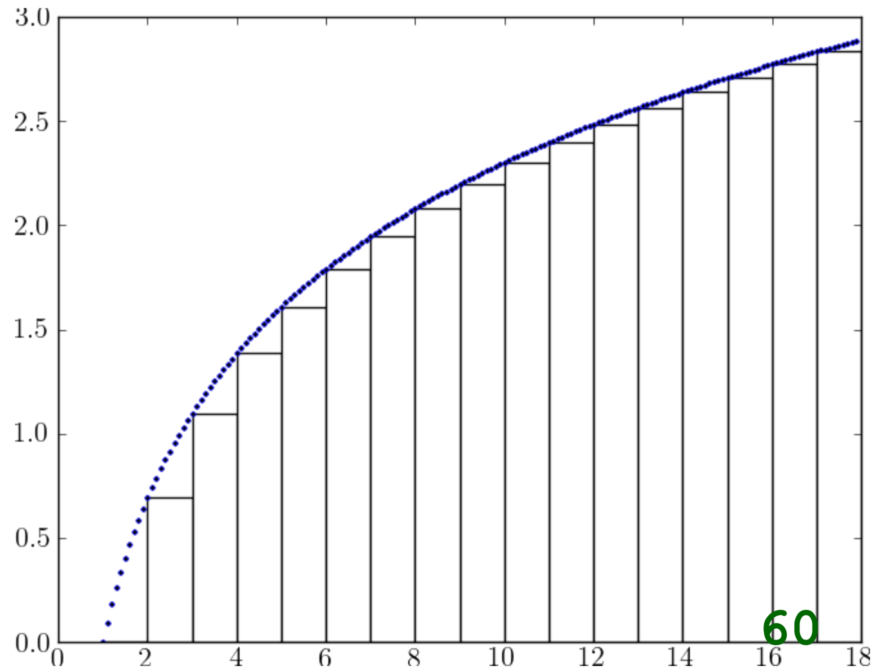
- 求和转换为求积分
  - $\log \boldsymbol{n}! = \sum_{i=1}^{n} \log \boldsymbol{i}$
  - **曲线之下面积**
  - $\log \boldsymbol{n}! < \int_{1}^{n+1} \log x \, \boldsymbol{dx}$
  - $\because \log \boldsymbol{x} = \log \boldsymbol{e} \ln \boldsymbol{x}$
  - $\int_{1}^{n} \ln \boldsymbol{x} \, \boldsymbol{dx} = \boldsymbol{n} \ln \boldsymbol{n} - \boldsymbol{n} + \mathbf{1}$
  - $\therefore \log \boldsymbol{n}! < [(\boldsymbol{n}+\mathbf{1}) \ln(\boldsymbol{n}+\mathbf{1}) - (\boldsymbol{n}+\mathbf{1}) + \mathbf{1}] \log \boldsymbol{e}$
  - $\log \boldsymbol{n}! = \mathbf{O}(\boldsymbol{n} \log \boldsymbol{n})$

$$\log \boldsymbol{n}! = \boldsymbol{\Theta}(\boldsymbol{n} \log \boldsymbol{n})$$

和的估计与界限

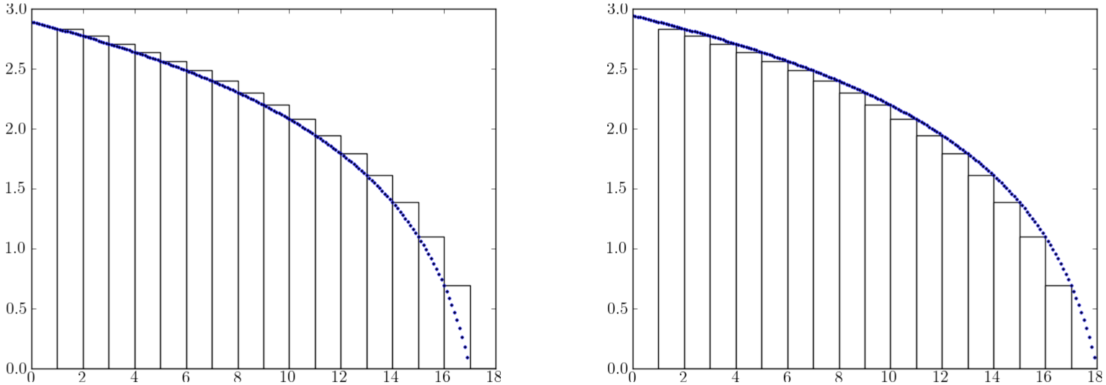
- 求和转换为求积分
  - 当$f(x)$单调递增时，有
  - $\int_{m-1}^{n} f(x)\,dx \le \sum_{m}^{n} f(x) \le \int_{m}^{n+1} f(x)\,dx$

- 求和转换为求积分
  - 当$f(x)$单调递增时，有
  - $\int_{m-1}^{n} f(x)\,dx \le \sum_{m}^{n} f(x) \le \int_{m}^{n+1} f(x)\,dx$



和的估计与界限

- 求和转换为求积分
  - 类似地，当$f(x)$单调递减时，有
  - $\int_{m}^{n+1} f(x)\,dx \le \sum_{m}^{n} f(x) \le \int_{m-1}^{n} f(x)\,dx$
  - 例如：
    - $\sum_{i=1}^{n} \frac{1}{i} = 1 + \sum_{i=2}^{n} \frac{1}{i} < 1 + \int_{1}^{n} \frac{dx}{x} = 1 + \ln n$
    - $\sum_{i=1}^{n} \frac{1}{i} > \int_{1}^{n+1} \frac{dx}{x} = \ln(n+1)$



递归方程

- 例： Merge-sort排序算法的复杂性方程
  - $T(n)=\begin{cases}\Theta(1), & n=1\\ 2T\left(\frac{n}{2}\right)+\Theta(n), & n>1\end{cases}$
  - 解：$T(n)=\Theta(n\log n)$

递归方程

- 递归逐层展开求解
  - $T(n) = n + 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right)$
    $= n + 3(\left\lfloor \frac{n}{4} \right\rfloor + 3T\left(\left\lfloor \frac{n}{16} \right\rfloor\right))$
    $= n + 3(\left\lfloor \frac{n}{4} \right\rfloor + 3\left(\left\lfloor \frac{n}{16} \right\rfloor + 3T(\left\lfloor \frac{n}{64} \right\rfloor)\right))$
    $= n + 3\left\lfloor \frac{n}{4} \right\rfloor + 3^2\left\lfloor \frac{n}{4^2} \right\rfloor + 3^3\left\lfloor \frac{n}{4^3} \right\rfloor + \cdots + 3^k T(\left\lfloor \frac{n}{4^k} \right\rfloor)$
  - 深度 $k = \log_4 n$，最底层有$3^k = 3^{\log_4 n} = n^{\log_4 3}$个
  - $T(n) = \sum_{i=0}^{(\log_4 n)-1} 3^i \frac{n}{4^i} + \Theta(n^{\log_4 3})$
  - $\leq 4n + \Theta(n^{\log_4 3}) = O(n)$

递归方程

- 变量替换法求解
  - $T(n) = 2T(\sqrt{n}) + \log n$
  - 令 $m = \log n$，则$n = 2^m$， $T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$
  - 令 $S(m) = T(2^m)$，则 $S\left(\frac{m}{2}\right) = T\left(2^{\frac{m}{2}}\right)$
  - 于是 $S(m) = 2S\left(\frac{m}{2}\right) + m$
  - 显然 $S(m) = \Theta(m \log m)$
  - 则 $T(n) = \Theta(\log n \log(\log n))$

递归方程

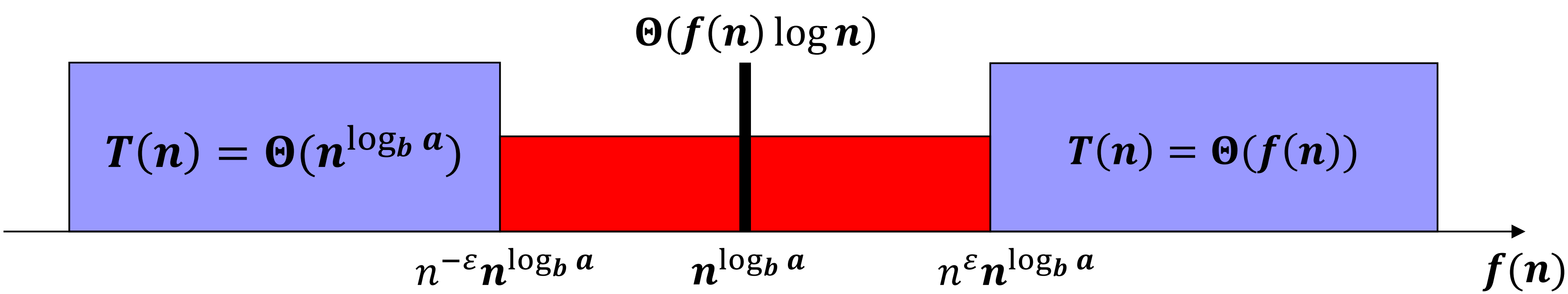
- **Master定理求解**
  - 求解 $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ 型递归方程，其中 $a \geq 1$，$b > 1$ 是常数，$f(n)$ 是正函数
    - 记住三种情况，可快速求解

递归方程

- **Master定理求解**
  - 求解 $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ 型递归方程，其中 $a \geq 1$，$b > 1$ 是常数，$f(n)$ 是正函数
    - 若 $f(n) = O(n^{(\log_b a)-\varepsilon})$，$\varepsilon > 0$ 是常数，则有$T(n) = \Theta(n^{\log_b a})$
    - 若 $f(n) = \Theta(n^{\log_b a})$，$\varepsilon > 0$ 是常数， 则有$T(n) = \Theta(n^{\log_b a} \log n)$
    - 若 $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$，$\varepsilon > 0$ 是常数，且对所有充分大的 n 有 $af(\frac{n}{b}) \leq cf(n)$，$c < 1$ 是常数，则有 $T(n) = \Theta(f(n))$

递归方程

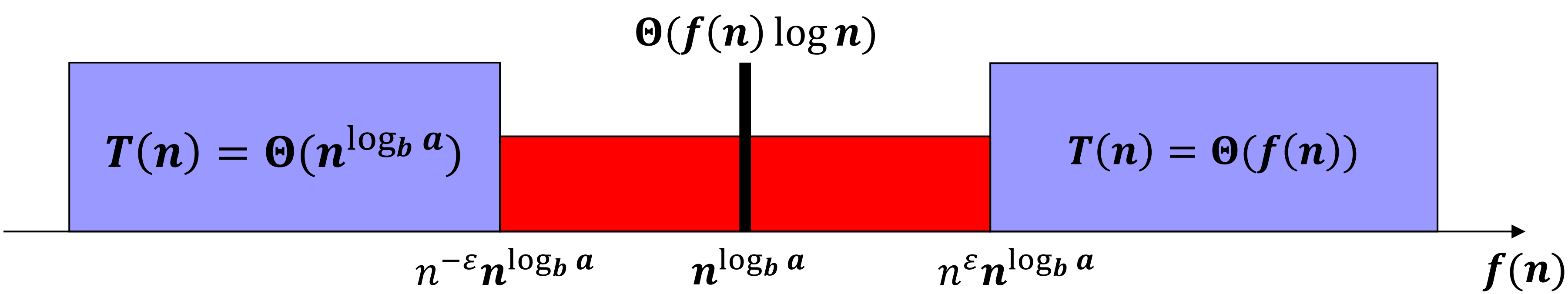
- Master定理（直观理解，一般情况）
  - 用 $f(n)$ 与 $n^{\log_b a}$ 的阶比较，
    - 若 $n^{\log_b a}$ 更大，则 $T(n)=\Theta(n^{\log_b a})$
    - 若 $f(n)=\Theta(n^{\log_b a})$，即 $f(n)$ 与 $n^{\log_b a}$ 同阶， 则有 $T(n)=\Theta\left(n^{\log_b a}\log n\right)=\Theta(f(n)\log n)$
    - 若 $f(n)$ 更大，则 $T(n)=\Theta(f(n))$



对于红色部分，Master定理无能为力

递归方程

- **Master定理（更进一步理解）**
  - 用 $f(n)$ 与 $n^{\log_b a}$ 的阶比较，
    - 第一种情况，$f(n)$不仅小于 $n^{\log_b a}$，而且要小于 $n^{\log_b a}/n^{\varepsilon}$，即$f(n) = O(n^{(\log_b a)-\varepsilon})$
    - 第三种情况，$f(n)$不仅大于 $n^{\log_b a}$，而且要大于 $n^{\log_b a} * n^{\varepsilon}$，即$f(n) = \Omega(n^{(\log_b a)+\varepsilon})$



对于红色部分，Master定理无能为力

递归方程

- Master定理（例子）
  - 求解 $T(n) = 9T\left(\frac{n}{3}\right) + n$
  - $a = 9,\ b = 3,\ f(n) = n,\ n^{\log_b a} = n^2$
  - $\because f(n) = n = O\left(n^{(\log_b a) - \varepsilon}\right)$, 即 $\varepsilon = 1$
  - $\therefore T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^2\right)$

递归方程

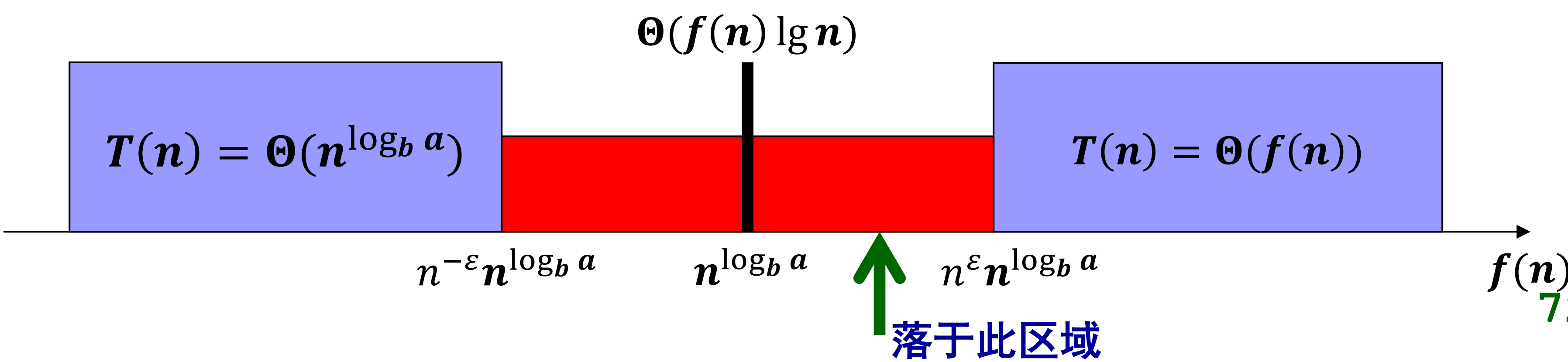
- **Master定理（例子）**
  - 求解 $T(n) = T\left(\frac{2n}{3}\right) + 1$
  - $a = 1, b = \frac{3}{2}, f(n) = 1, n^{\log_b a} = n^{\log_{3/2} 1} = 1$
  - $\because f(n) = 1 = \Theta(n^{\log_b a}),$
  - $\therefore T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta(\lg n)$

递归方程

- Master定理（例子）
  - 求解 $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$
  - $a = 3, b = 4,$
  - $f(n) = n\log n,\quad n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
  - $\because f(n) = n\log n \geq n = \Omega(n^{\log_b a+\varepsilon}), \varepsilon \approx 0.2$
  - 对所有n有 $af\left(\frac{n}{b}\right) = 3\frac{n}{b}\log\frac{n}{b} \leq \frac{3}{4}n\log n = cf(n)$
  - $\therefore T(n) = \Theta(f(n)) = \Theta(n\log n)$

递归方程

- **Master定理（例子）**
  - 求解 $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$
  - $a = 2, b = 2,$
  - $f(n) = n\log n,\quad n^{\log_b a} = n^{\log_2 2} = O(n)$
  - 虽然 $f(n) = n\log n \geq n^{\log_b a} = n$，但是$\frac{f(n)}{n^{\log_b a}} = \log n$渐近小于 n^{ε}



Master定理证明

- 证明思路
  - $a \geq 1$，$b > 1$ 是常数，$f(n)$ 是正函数
  - 对 $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ 展开可得
  - $T(n) = \Theta\left(n^{\log_b a}\right) + \sum_{i=0}^{k-1} a^i f(\frac{n}{b^i})$,
  - $n = b^k, k = \log_b n\, , a^k = a^{\log_b n} = n^{\log_b a}$
  - 令 $g(n) = \sum_{i=0}^{k-1} a^i f(\frac{n}{b^i})$
  - $T(n) = \Theta\left(n^{\log_b a}\right) + g(n)$

Master定理证明

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

- **证明思路** $n=b^k, k=\log_b n, a^k=a^{\log_b n}=n^{\log_b a}$
  - 若 $f(n)=O\left(n^{(\log_b a)-\varepsilon}\right)$，$\varepsilon>0$ 是常数，则有
  - $g(n)=O\left(\sum_{i=0}^{k-1}a^i\left(\frac{n}{b^i}\right)^{(\log_b a)-\varepsilon}\right)$
  - $=O\left(n^{(\log_b a)-\varepsilon}\sum_{i=0}^{k-1}\left(\frac{ab^{\varepsilon}}{b^{\log_b a}}\right)^i\right)$
  - $=O\left(n^{(\log_b a)-\varepsilon}\sum_{i=0}^{k-1}(b^{\varepsilon})^i\right)$
  - $=O\left(n^{(\log_b a)-\varepsilon}\sum_{i=0}^{k-1}(b^{\varepsilon})^i\right)$
  - $=O\left(n^{(\log_b a)-\varepsilon}\frac{n^{\varepsilon}-1}{b^{\varepsilon}-1}\right)$
  - $=O\left(n^{\log_b a}\right)$
  - $\therefore T(n)=\Theta\left(n^{\log_b a}\right)+g(n)=\Theta\left(n^{\log_b a}\right)$

Master定理证明

$$g(n)=\sum_{i=0}^{k-1}a^i f(\frac{n}{b^i})$$

- 证明思路

$$n=b^k, k=\log_b n\,, a^k=a^{\log_b n}=n^{\log_b a}$$

  - 若 $f(n)=\Theta(n^{\log_b a})$，$\varepsilon>0$ 是常数， 则有
  - $g(n)=\Theta(\sum_{i=0}^{k-1}a^i\left(\frac{n}{b^i}\right)^{\log_b a})$
  - $=\Theta(n^{\log_b a}\sum_{i=0}^{k-1}\left(\frac{a}{b^{\log_b a}}\right)^i)$
  - $=\Theta(n^{\log_b a}\sum_{i=0}^{k-1}1)$
  - $=\Theta(n^{\log_b a}k)$
  - $=\Theta(n^{\log_b a}\log_b n)$
  - $=\Theta\left(n^{\log_b a}\log n\right)$
  - $\therefore T(n)=\Theta\left(n^{\log_b a}\right)+g(n)=\Theta(n^{\log_b a}\log n)$

Master定理证明

$$g(n)=\sum_{i=0}^{k-1}a^i f(\frac{n}{b^i})$$

- 证明思路

$$n=b^k, k=\log_b n, a^k=a^{\log_b n}=n^{\log_b a}$$

  - 若 $f(n)=\Omega(n^{(\log_b a)+\varepsilon})$，$\varepsilon>0$ 是常数，且对所有充分大的 n 有 $af(\frac{n}{b})\le cf(n)$，$c<1$ 是常数，则有
  - $af\left(\frac{n}{b^2}\right)\le cf\left(\frac{n}{b}\right)$
  - $\vdots$
  - $af\left(\frac{n}{b^i}\right)\le cf\left(\frac{n}{b^{i-1}}\right)$
  - 两边分别相乘，可得 $a^i f(\frac{n}{b^i})\le c^i f(n)$
  - $g(n)=\sum_{i=0}^{k-1}a^i f(\frac{n}{b^i})\le\sum_{i=0}^{k-1}c^i f(n)=f(n)\sum_{i=0}^{k-1}c^i$
  - $\le f(n)\frac{1}{1-c}=\Theta(f(n))$
  - $\therefore T(n)=\Theta\left(n^{\log_b a}\right)+g(n)=\Theta(f(n))$