算法设计与分析

宋洪涛

Email：songhongtao@hrbeu.edu.cn

平时成绩和考试

- 48学时（授课32、上机16）
- 2.5学分

成绩分布：

- 平时成绩：10%
- 实验成绩：30%
- 闭卷考试：60%

课程安排

- 算法概述
- 递归和分治
- 动态规划
- 贪心算法
- 回溯法
- 分支限界法

第1章　算法概述

- 算法的概念
- 算法的地位
- 算法实例
- 算法分析基础

算法的概念

算法

- 算法是指解决问题的一种方法或一个过程。
- 算法是若干指令的有穷序列，满足性质：
  - 输入：有外部提供的量作为算法的输入。
  - 输出：算法产生至少一个量作为输出。
  - 确定性：组成算法的每条指令是清晰，无歧义的。（相同的输入得到相同的输出）
  - 有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序

- 程序是算法用某种程序设计语言的具体实现。
- 程序可以不满足算法的性质(4)。
  - 例如操作系统，是一个无限循环执行的程序，因而不是一个算法。
  - 操作系统的任务是针对一些单独的问题，每个问题由操作系统中的一个子程序实现，程序得到结果后终止。

问题和问题的实例

- 问题
  - 对一个给定数组进行排序
- 问题的实例
  - 对数组[5, 2, 4, 6, 1, 3]进行排序
- 注意
  - 一个算法面向一个问题，而不是仅求解一个问题的一个或几个实例。

算法的地位

算法是计算机科学基础的重要主题

- 70 年代前
  - 计算机科学基础的主题没有被清楚地认清。
- 70 年代
  - Knuth 出版了《The Art of Computer Programming》以算法研究为主线
  - 确立了算法为计算机科学基础的重要主题 Knuth 于1974 年获得图灵奖。
- 70 年代后
  - 算法作为计算机科学核心推动了计算机科学技术飞速发展
  - 和生活密切相关，地铁收费、交通摄像、网络购物等等

算法的地位

❑ 盖楼的例子

能盖楼吗

↓

适合盖什么样的楼　　　　建筑专家论证

↓

大楼的设计方案　　　　建筑师设计

↓

找施工队按设计盖楼　　　　农民工垒砖

算法的地位

- 解决一个计算问题的

可计算否

↓

适合计算否

↓

算法设计与分析

↓

用计算机语言实现算法

一个例子

- 排序问题
  - 输入：n个数$a_1, a_2, \ldots, a_n$.
  - 输出：一个排列$a'_1, a'_2, \ldots, a'_n$满足$a'_1 \le a'_2 \le \ldots \le a'_n$

- 实例
  - A[1 ,..., n]=5, 2, 4, 6, 1, 3

一个例子

- 插入排序（抓扑克牌）
  - A[1 ,..., n]=5, 2, 4, 6, 1, 3
  - A[1 ,..., n]=5
  - A[1 ,..., n]=2, 5
  - A[1 ,..., n]=2, 4, 5
  - A[1 ,..., n]=2, 4, 5, 6
  - A[1 ,..., n]= 1, 2, 4, 5, 6
  - A[1 ,..., n]= 1, 2, 3, 4, 5, 6

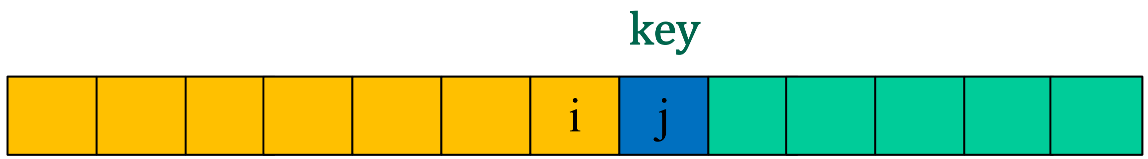
一个例子

❑ Insertion-sort(A)

Input: A[1,.....,n]=n个数
output: A[1,.....,n]=n个sorted数

```
for j=2  to n  do
    key←A[j];
    i←j-1;
    while i>0 and A[i]>key  do
        A[i+1]←A[i];
        i←i-1;
    A[i+1]←key;
```

1.共1个for 和 1个while循环
2.循环内，key不变，最终找到key的位置
3.While循环每比较一次，为key串出待插入的位置
4.判断结束，插入key



算法分析基础

？什么更重要

算法的正确性分析

- 一个算法是正确的，它对于每一个输入都最终停止,而且产生正确的输出
- 例：证明插入排序算法是正确的

```
Insertion-sort(A)
Input:  A[1,.....,n]=n个数
output: A[1,.....,n]=n个sorted数
1. for j=2  to n  do
2.     key←A[j];
3.     i←j-1;
4.     while i>0 and A[i]>key  do
5.         A[i+1]←A[i];
6.         i←i-1;
7.     A[i+1]←key;
```

正确性证明

- 只需证明循环不变性：
  - 在每次for循环开始前，子数组A[1…j-1]恰好是原始数组中A[1…j-1]各元素排好序的形式
- 证明
  - 初始化：j=2
  - 归纳：A[1…j-1]排好序，A[j]正确插入
  - 终止：算法终止时A[1…n]排好序

算法复杂性分析

算法的复杂性：运行算法所需的计算机资源，反映算法的效率。

- 分析算法运行的时间
  - 时间复杂性
- 分析算法运行的空间（存储器）
  - 空间复杂性

算法的时间复杂性

- 以插入排序算法为例
  - 更长的数组——更多的时间
- 把算法运行的时间定义为**输入大小**的函数
- 输入大小
  - 排序问题的输入大小=数组的长度
  - 矩阵问题的输入大小=矩阵的行数/列数
  - 图论问题的输入大小=图的边数/结点数

算法的时间复杂性

- 以插入排序算法为例
  - 相同长度的数组—— 运行时间不一定相同
- 分析三种情况下的时间复杂性
  - 最好情况下时间复杂性$T_{\min}(n)$
    - 在长度为n的输入上运行的最短时间
  - 最坏情况下时间复杂性$T_{\max}(n)$
    - 在长度为n的输入上运行的最长时间
  - 平均时间复杂性$T_{avg}(n)$
    - 在长度为n的输入上运行的平均时间

算法的时间复杂性

- 用$t(I)$表示算法在某个实例 I 上的运行时间，$size(I)$表示实例 I 的大小，D是所有输入I的集合

$$T_{\max}(n) = \max_{size(I \in D)=n} t(I)$$ 最有实际价值！

$$T_{\min}(n) = \min_{size(I \in D)=n} t(I)$$

$$T_{avg}(n) = \sum_{size(I \in D)=n} P(I) * t(I)$$

- $P(I)$为实例I出现的概率

插入排序的时间复杂性(最坏情况)

Insertion-sort(A)

		cost	times
1.	**for** j=2 **to** n **do**		
2.	key←A[j];	------------ c1	n-1
3.	i←j-1;	----------- c2	n-1
4.	**while** i>0 **and** A[i]>key **do**	-------- c3	$\sum_{j=2}^{n} j$
5.	A[i+1]←A[i];	----------- c4	$\sum_{j=2}^{n} j-1$
6.	i←i-1;	----------- c5	$\sum_{j=2}^{n} j-1$
7.	A[i+1]←key;	------------- c6	n-1

$$T_{\max}(n) = c_1(n-1) + c_2(n-1) + c_3\left(\frac{n(n+1)}{2} - 1\right)$$

$$+ c_4\left(\frac{n(n-1)}{2}\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6(n-1)$$

$$= \frac{c_3 + c_4 + c_5}{2} n^2 + \left(c_1 + c_2 + c_6 + \frac{c_3 - c_4 - c_5}{2}\right) n - (c_1 + c_2 + c_6 - c_3)$$

插入排序的时间复杂性(最好情况)

Insertion-sort(A)	cost	times
for j=2 to n do		
key←A[j];	------------ c1	n-1
i←j-1;	----------- c2	n-1
while i>0 and A[i]>key do	-------- c3	n-1
A[i+1]←A[i];	----------- c4	0
i←i-1;	----------- c5	0
A[i+1]←key;	------------- c6	n-1

$$T_{\min}(n) = c_1(n-1) + c_2(n-1) + c_3(n-1) + c_6(n-1)$$
$$= (c_1 + c_2 + c_3 + c_6)n - (c_1 + c_2 + c_3 + c_6)$$

渐近时间复杂性

$$T_{\min}(n) = an + b \qquad T_{\max}(n) = \alpha n^2 + \beta n + \gamma$$

- $T_{\min}(n) \sim n$ $\qquad T_{\max}(n) \sim n^2$

- 一般来说，$n \to \infty$, $T(n) \to \infty$。如果存在 $\tilde{T}(n)$使得当$n \to \infty$时，$(T(n)\text{-}\tilde{T}(n))/T(n) \to 0$, 则称 $\tilde{T}(n)$为 $T(n)$当 $n \to \infty$ 时的渐进性态，或称$\tilde{T}(n)$为渐进复杂性。
- 直观的讲，$\tilde{T}(n)$是$T(n)$去掉低阶后的主项。

渐近复杂性 O （符号）

- 对于正值函数$f(n) \ge 0$和$g(n) \ge 0$，如果存在正常数c和n_0使得对所有$n \ge n_0$有：$f(n) \le cg(n)$，则称$f(n)$是$g(n)$的低阶函数或$g(n)$是$f(n)$的渐近上界，记为$f(n)=O(g(n))$

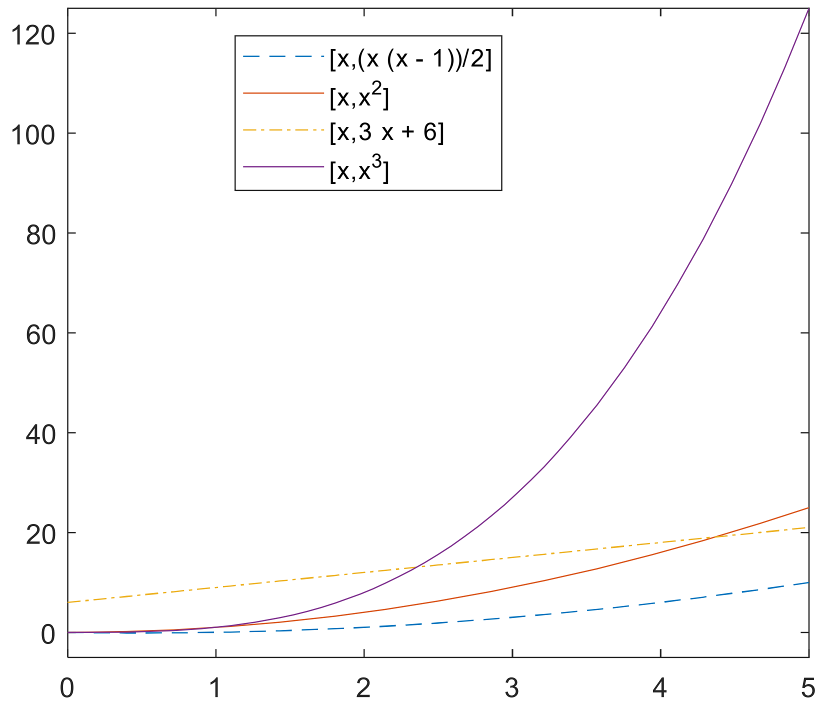
$$\frac{n(n-1)}{2} = O(n^2)$$

$$3n+6=O(n)$$

$$n+1024=O(n)$$

$$3n+6=O(n^2)$$

$$n^3 \ne O(n^2)$$



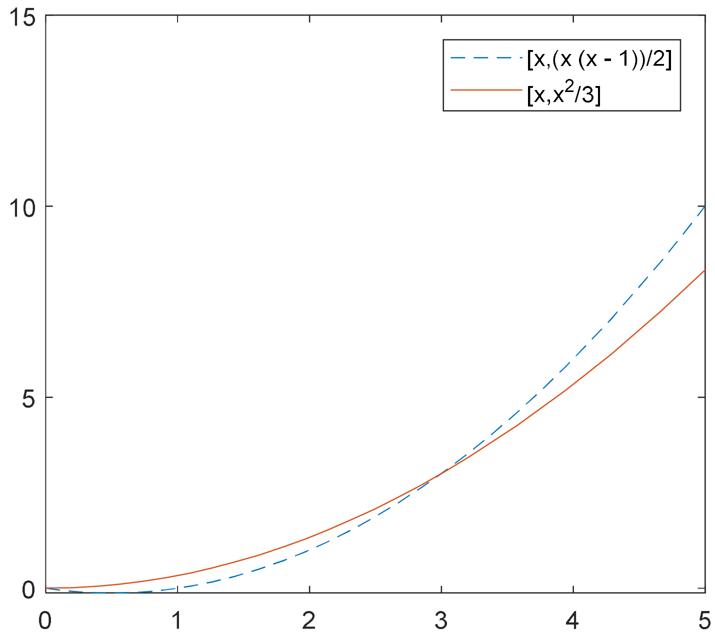
渐近复杂性 Ω

- 对于正值函数$f(n)$ 和$g(n)$，如果存在正常数c和n_0使得对所有$n \geq n_0$有：$f(n) \geq cg(n)$，则称$f(n)$是$g(n)$的高阶函数或$g(n)$是$f(n)$的渐近下界，记为$f(n)=\Omega(g(n))$

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

$$\frac{n(n-1)}{2} = \Omega(n^2)$$

$$\frac{n(n-1)}{2} = \Omega(n)$$



渐近复杂性 θ

- 对于正值函数$f(n)$ 和$g(n)$，如果存在正常数c_1，c_2和n_0使得对所有$n \geq n_0$有： $c_1 g(n) \leq f(n) \leq c_2 g(n)$，则称$f(n)$是$g(n)$的同阶函数，记为$f(n)=\theta(g(n))$

  - $f(n)=\theta(g(n))$ if $f(n)=O(g(n)) \wedge f(n)=\Omega(g(n))$

$$\frac{n(n-1)}{2} = \theta(n^2)$$

$$3n \neq \theta(n^2)$$

$$3n+6=\theta(n)$$

$$n+1024=\theta(n)$$

渐近复杂性 o

- 正值函数$f(n)$ 和$g(n)$，如果对于任意正常数c，存在n_0使得对所有$n \geq n_0$有：$f(n) < cg(n)$，则称$f(n)$是$g(n)$的严格低阶函数或$g(n)$是$f(n)$的严格渐近上界，记为

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

$$3n+6=o(n^2)$$

渐近复杂性 ω

- 正值函数$f(n)$ 和$g(n)$，如果对于任意正常数c，存在n_0使得对所有$n \geq n_0$有：$f(n) > cg(n)$，则称$f(n)$是$g(n)$的严格高阶函数或$g(n)$是$f(n)$的严格渐近下界，记为

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

$$\frac{n(n-1)}{2} = \omega(n)$$

渐近分析中函数比较

- $f(n) = O(g(n)) \approx a \le b$;
- $f(n) = \Omega(g(n)) \approx a \ge b$;
- $f(n) = \Theta(g(n)) \approx a = b$;
- $f(n) = o(g(n)) \approx a < b$;
- $f(n) = \omega(g(n)) \approx a > b$;

渐近分析记号的若干性质

- （1）传递性：
- $f(n)=\theta(g(n))$， $g(n)=\theta(h(n)) \Rightarrow f(n)=\theta(h(n))$；
- $f(n)=O(g(n))$， $g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$；
- $f(n)=\Omega(g(n))$， $g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$；

渐近分析记号的若干性质

- （2）反身性：

  $f(n)=\theta(f(n))$；

  $f(n)=O(f(n))$；

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

- （3）对称性：

  $f(n)=\theta(g(n)) \Leftrightarrow g(n)=\theta(f(n))$.

- （4）互对称性：

  $f(n)=O(g(n)) \Leftrightarrow g(n)=\Omega(f(n))$ ；

渐近分析记号的若干性质

- （5）算术运算：
- $O(f(n))+O(g(n)) = O(\max\{f(\mathrm{n}), g(n)\})$ ；
- $O(f(n))+O(g(n)) = O(f(\mathrm{n})+g(n))$ ；
- $O(f(n))*O(g(n)) = O(f(\mathrm{n})*g(n))$ ；
- $O(cf(n)) = O(f(\mathrm{n}))$ ；
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(\mathrm{n}))$ 。

渐近分析记号的若干性质

- 证明$O(f(n))+O(g(n)) = O(f(n) + g(n))$
  - 对于任意$F(n) = O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n \geq n_1$，有$F(n) \leq c_1 f(n)$。
  - 类似地，对于任意$G(n) = O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n \geq n_2$，有$G(n) \leq c_2 g(n)$。
  - 令$c_3=\max\{c_1, c_2\}$，$n_3 =\max\{n_1, n_2\}$，$h(n)= f(n)+g(n)$。
  - 则对所有的 $n \geq n_3$，有
  - $F(n) +G(n) \leq c_1 f(n) + c_2 g(n)$

$$\leq c_3 f(n) + c_3 g(n)= c_3(f(n) + g(n))$$

$$= O(h(n)) .$$

渐近分析记号的若干性质

- 证明 $O(f(n))*O(g(n)) = O(f(\mathrm{n})*g(n))$；
  - 对于任意$F(n) = O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n \geq n_1$，有$F(n) \leq c_1 f(n)$。
  - 类似地，对于任意$G(n) = O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n \geq n_2$，有$G(n) \leq c_2 g(n)$。
  - 令$c_3 = c_1 * c_2$，$n_3 = n_1 * n_2$，$h(n) = f(\mathrm{n}) * g(n)$。
  - 则对所有的 $n \geq n_3$，有
  - $F(n) * G(n) \leq c_1 f(n) * c_2 g(n)$

$$\leq c_3 f(n) * g(n) = c_3(f(n) * g(n))$$

$$= O(h(n)) .$$

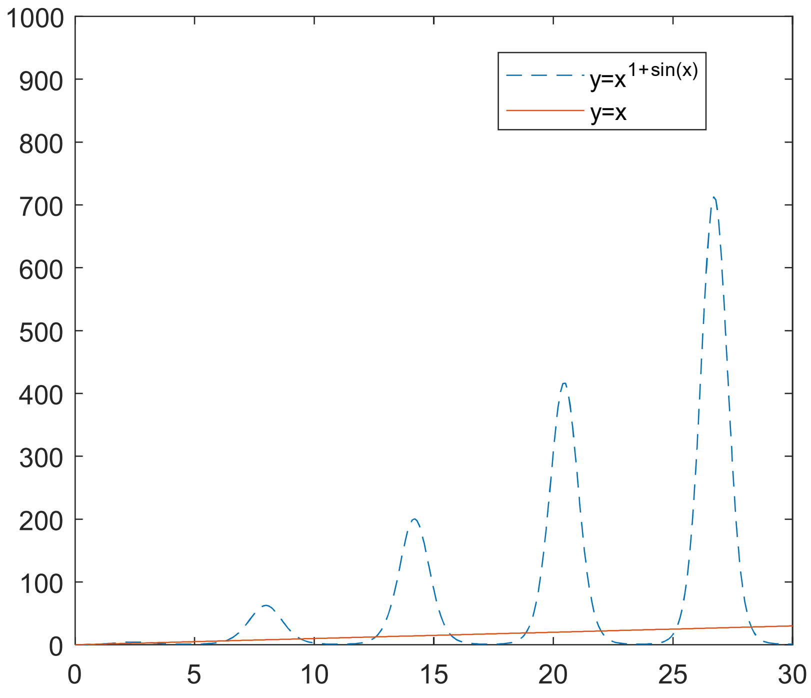
渐近分析记号的若干性质

- 证明 $O(cf(n)) = O(f(n))$；
  - 另$g(n)=cf(n)$,对于任意$G(n) =O(cf(n))= O(g(n))$，存在正常数c_1和自然数n_1，使得对所有$n\geq n_1$，有$G(n) \leq c_1 g(n) \leq c_1 c f(n)$。
  - 令$c_2=c_1 * c$，则对所有的 $n \geq n_1$，有$G(n) \leq c_2 f(n)$,
  - 所以， $G(n) =O(f(n))= O(cf(n))$.

渐近复杂性

- 并非所有函数都是可比的，即对于有的$f(n)$和$g(n)$，$f(n)\neq O(g(n))$, $f(n)\neq \Omega(g(n))$

- 例如，n 和 $n^{1+\sin(n)}$



算法渐近复杂性分析中常用函数

- **（1）单调函数**
- 单调递增：$m \le n \Rightarrow f(m) \le f(n)$;
- 单调递减：$m \le n \Rightarrow f(m) \ge f(n)$;
- 严格单调递增：$m < n \Rightarrow f(m) < f(n)$;
- 严格单调递减：$m < n \Rightarrow f(m) > f(n)$.
- **（2）取整函数**
- $\lfloor x \rfloor$：不大于x的最大整数；↓
- $\lceil x \rceil$：不小于x的最小整数。↑

算法渐近复杂性分析中常用函数

- $x\text{-}1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$；　x=2.5

  $x = 2.5$

  $\lfloor x \rfloor = 2$，$\lceil x \rceil = 3$

- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;　　n=2.5，n=-3

  $n=2.5$

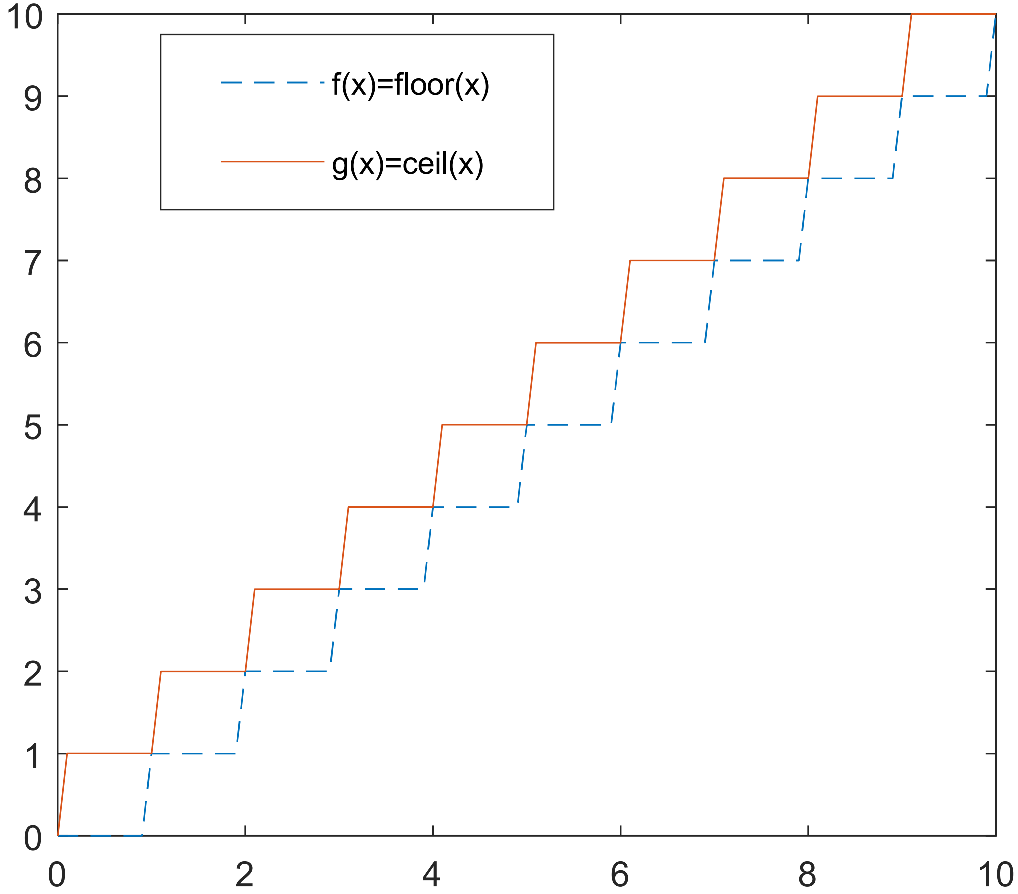
  $\lfloor n/2 \rfloor + \lceil n/2 \rceil = 1+2 = 3 \ne n$　*（需规定n是整数）*

  $n=-3$

  $\lfloor n/2 \rfloor + \lceil n/2 \rceil = -2 + (-1) = -3 = n$

算法渐近复杂性分析中常用函数

- $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。



算法渐近复杂性分析中常用函数

- $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
- $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;

表达式中的多个同向取整可以合并！

算法渐近复杂性分析中常用函数

- （3）多项式函数
- $p(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$；
  - $p(n) = \theta(n^d)$;
  - $k \geq d \Rightarrow p(n) = O(n^k)$;
  - $k \leq d \Rightarrow p(n) = \Omega(n^k)$;
- $f(n) = O(n^k) \Leftrightarrow f(n)$ 多项式有界；
- $f(n) = O(1) \Leftrightarrow f(n) \leq c$; 有界

算法渐近复杂性分析中常用函数

- （4）指数函数
- 对于正整数m,n和实数$a>0$:

$a^0=1$;

$a^1=a$;

$a^{-1}=1/a$;

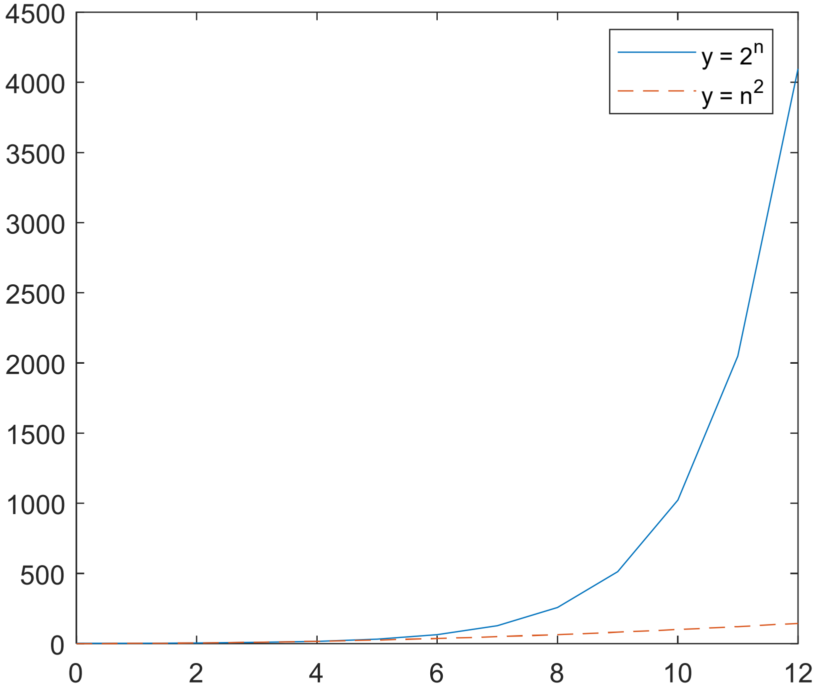
$(a^m)^n = a^{mn}$;

$(a^m)^n = (a^n)^m$;

$a^m a^n = a^{m+n}$;

$a>1 \Rightarrow a^n$为单调递增函数;

$a>1 \Rightarrow \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$



算法渐近复杂性分析中常用函数

- （4）指数函数
- 对于正整数*m*,*n*和*a*>1: $\lim\limits_{n\to\infty}\frac{n^b}{a^n}$

因为$n \to \infty$时$n^b \to \infty, a^n \to \infty$

求b阶导：

$$\lim_{n\to\infty}\frac{n^b}{a^n}=\lim_{n\to\infty}\frac{bn^{b-1}}{na^{n-1}}=\lim_{n\to\infty}\frac{b(b-1)n^{b-2}}{n(n-1)a^{n-2}}$$

$$=\lim_{n\to\infty}\frac{\mathrm{b}\,!}{n(n-1)\cdots(n-b)a^{n-\mathrm{b}}}=0$$

算法渐近复杂性分析中常用函数

- **（4）指数函数**
- **对于正整数*m*,*n*和*a*>1:** $\lim_{n\to\infty}\frac{n^b}{a^n}$

$$\lim_{n\to\infty}\frac{n^b}{a^n}=\lim_{n\to\infty}\frac{b\log n}{\mathrm{n}\log a}=\lim_{n\to\infty}\frac{\log n}{\mathrm{n}}$$

因为$\mathrm{n}\to\infty$，$\log\mathrm{n}\to\infty$

$$=\lim_{n\to\infty}\frac{1}{\mathrm{n}}=0$$

算法渐近复杂性分析中常用函数

重要极限：$e=\lim\limits_{n\to\infty}\left(1+\frac{1}{n}\right)^n$

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n=\lim_{n\to\infty}\left[\left(1+\frac{x}{n}\right)^{n/x}\right]^x=e^x$$

算法渐近复杂性分析中常用函数

泰勒公式：是将一个在**$x=x_0$**处具有**n**阶导数的函数**f**（**x**）利用关于（**$x-x_0$**）的**n**次多项式来逼近函数的方法。

若函数f（x）在包含x_0的某个闭区间[a,b]上具有n阶导数，且在开区间（a,b）上具有（n+1）阶导数，则对闭区间[a,b]上任意一点x，成立下式：

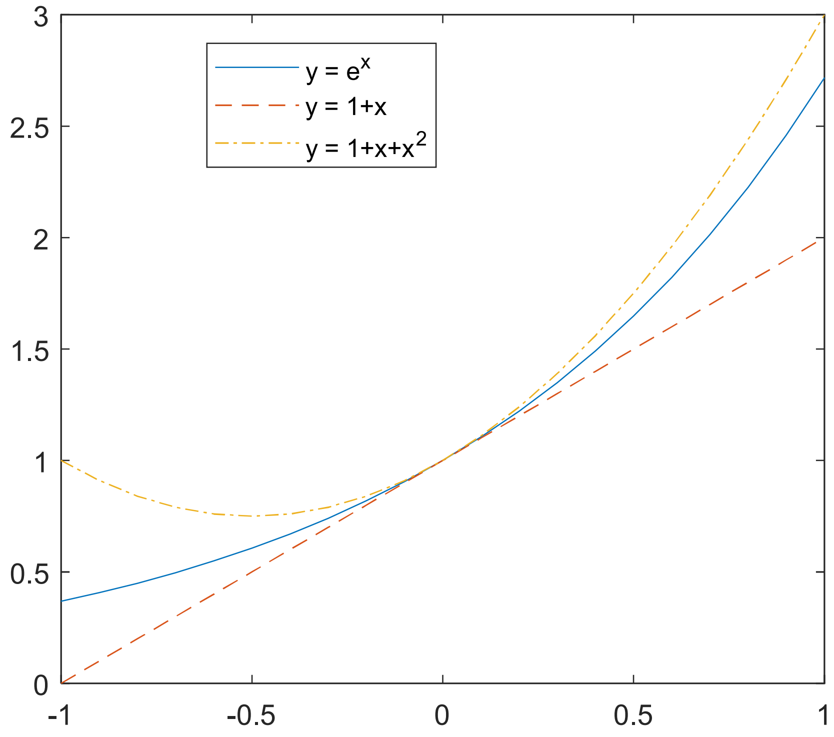
$$f(x)=\frac{f(x_0)}{0!}+\frac{f'(x_0)}{1!}(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\ldots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+R_n(x)$$

其中，表示f（x）的n阶导数，等号后的多项式称为函数f（x）在x_0处的泰勒展开式，剩余的R_n（x）是泰勒公式的余项，是（$x-x_0$）^{n}的高阶无穷小。

算法渐近复杂性分析中常用函数

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$ （x=0的泰勒展开式）

- $e^x \geq 1+x$;
- $|x| \leq 1 \Rightarrow 1+x \leq e^x \leq 1+x+x^2$;
- $e^x = 1+x+\Theta(x^2)$, as $x \to 0$;



算法渐近复杂性分析中常用函数

- **（5）对数函数**
- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$;
- $\log \log n = \log(\log n)$;
- 调和级数的n个部分和： $H_n = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$

算法渐近复杂性分析中常用函数

对数的性质：

当 a>0,b>0,c>0：

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

算法渐近复杂性分析中常用函数

- **-1<x≤1** $\Rightarrow$ $\ln(1+x)=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}-\cdots.$

$$f(x)=\ln(1+x),\ f^{(n)}(x)=(-1)^{n-1}(n-1)!(1+x)^{-n}$$

$$\ln(1+x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}(x-x_0)^n}{n(1+x_0)^n}$$

$$=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n}x^n \leftarrow x_0=0$$

$$=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}-\cdots.$$

算法渐近复杂性分析中常用函数

- **for $x > -1$,** $$\frac{x}{1+x} \le \ln(1+x) \le x$$

- **中值定理**

  如果函数f（x）满足在闭区间[a,b]上连续，开区间(a,b)上可导，那么在（a,b）上至少存在一点ε使得等式f(b)-f(a)=f'(ε)(b-a)成立。

- 对于f(x)=ln(x),存在c∈（1，1+x），使得：

  ln(1+x)-ln(1) = x/c

  即：x/(1+x)≤ln(1+x)≤x

算法渐近复杂性分析中常用函数

- 如果 $f(n) = O(\log^k n)$，则称 $f(n)$ 对数多项式有界

- for any $a > 0$, $\log^b n = o(n^a)$

$$\lim_{n\to\infty}\frac{\log^b n}{n^a} = \lim_{n\to\infty}\frac{\log^b n}{(2^a)^{\log n}}$$

因为$\mathrm{n} \to \infty, \log \mathrm{n} \to \infty$，另$\mathrm{m} = \log \mathrm{n}$

$$= \lim_{\mathrm{m}\to\infty}\frac{m^b}{(2^a)^{\mathrm{m}}} = \lim_{\mathrm{m}\to\infty}\frac{m^b}{\tilde{a}^m} = 0$$

算法渐近复杂性分析中常用函数

- （6）阶乘函数

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

- Stirling’s approximation(斯特林公式,阶乘近似值)

$$n! = \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

算法渐近复杂性分析中常用函数

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \qquad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n\log n)$$

算法渐近复杂性分析中常用函数

$$n! = o(n^n) \qquad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$\lim_{n\to\infty}\frac{n!}{n^n} = \frac{\sqrt{2pn}\left(\frac{n}{e}\right)^n e^{a_n}}{n^n} = \lim_{n\to\infty}\frac{\sqrt{2pn}\left(\frac{1}{e}\right)^n e^{a_n}}{1}$$

$$= \lim_{n\to\infty}\frac{\sqrt{2pn}}{e^{n-a_n}} = 0$$

算法渐近复杂性分析中常用函数

$$\log(n!) = \Theta(n\log n)$$

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$\lim_{n\to\infty}\frac{\log(n!)}{n\log n} = \frac{\log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n}\right)}{n\log n}$$

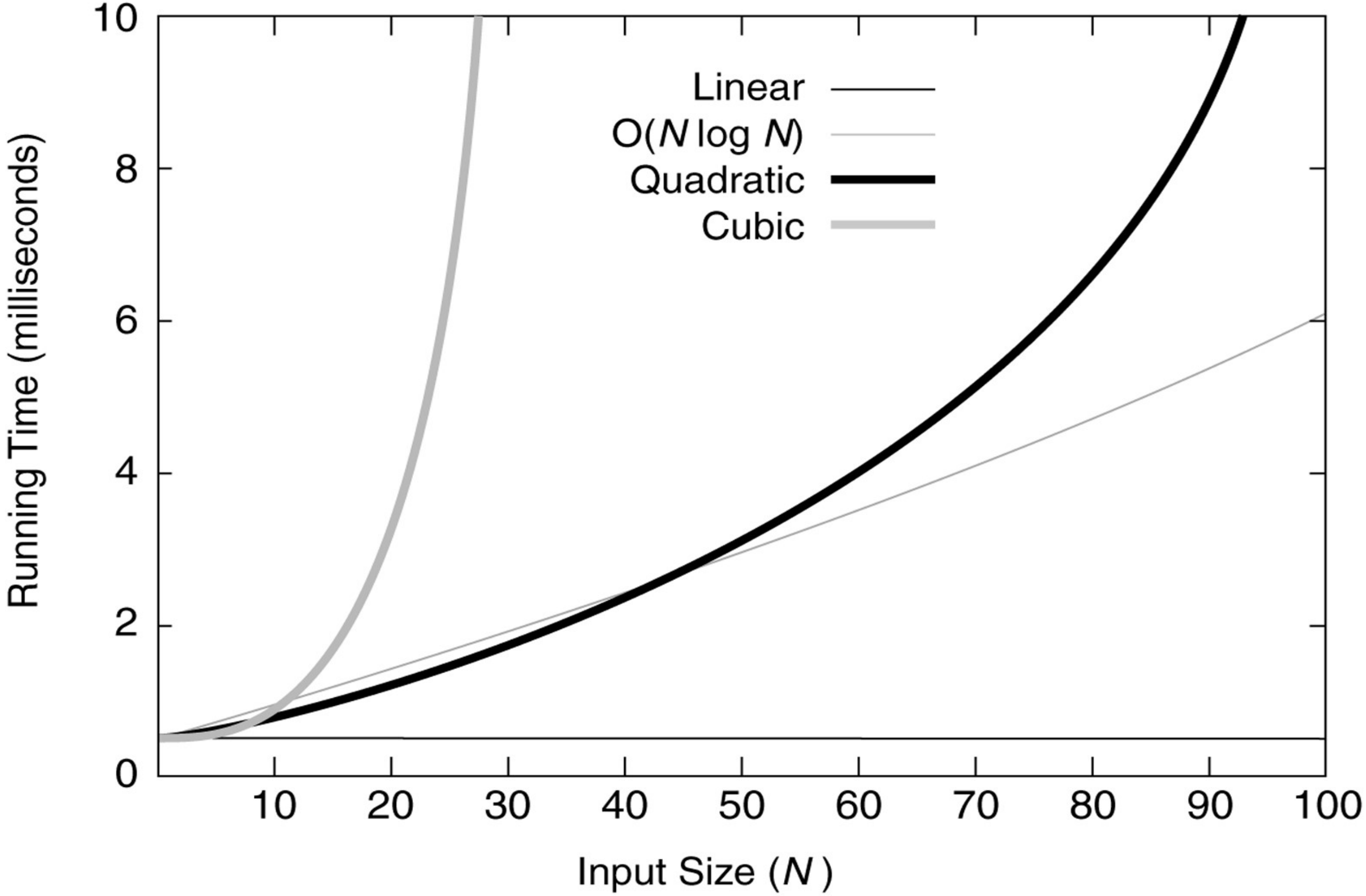
$$= \frac{\log(\sqrt{2\pi n}) + n\log\left(\frac{n}{e}\right) + \log(e^{\alpha_n})}{n\log n}$$

$$= \frac{n\log\left(\frac{n}{e}\right)}{n\log n} = 1$$

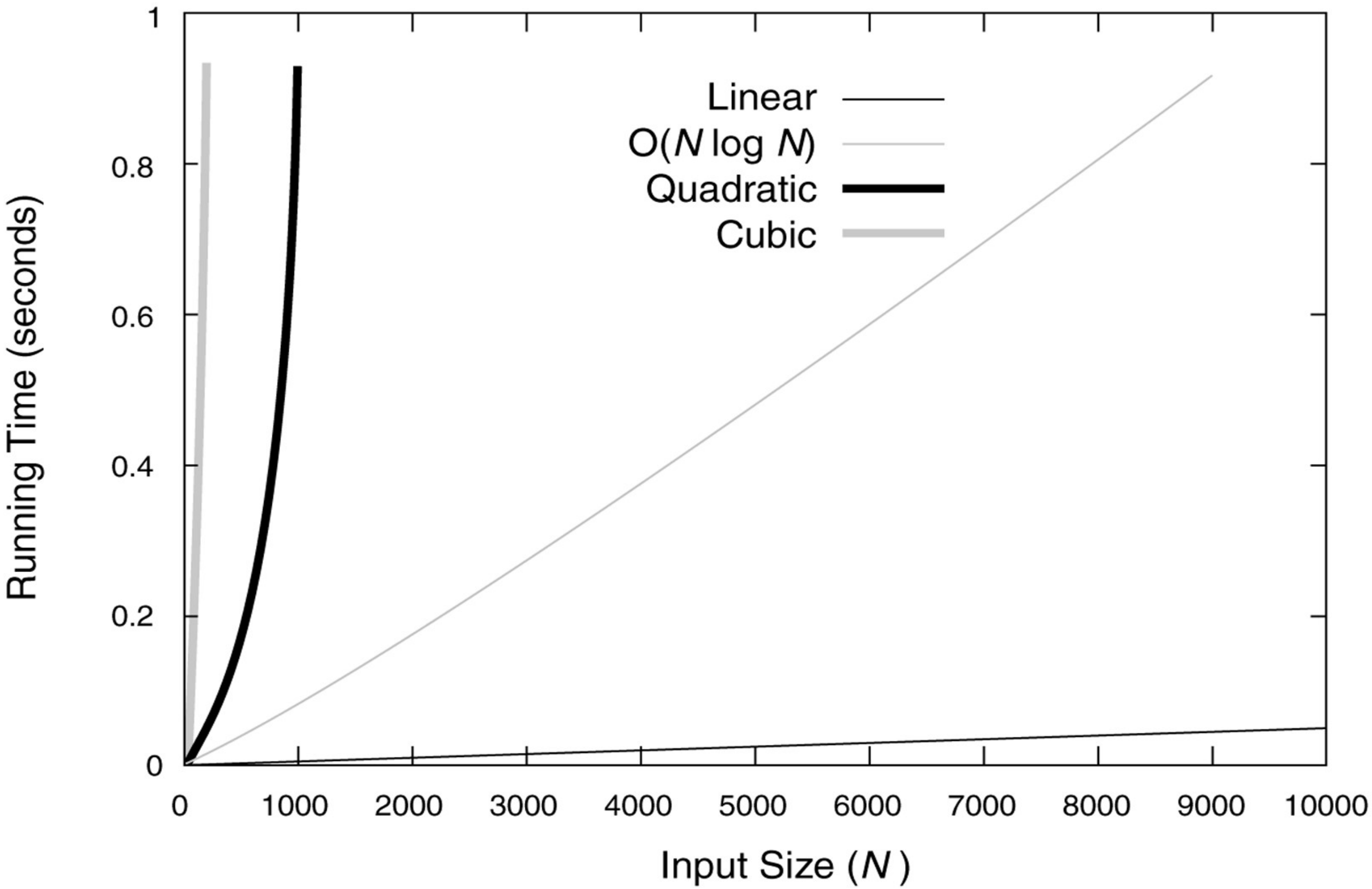
算法分析中常见的复杂性函数

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

小规模数据


Linear
O(N log N)
Quadratic
Cubic
Running Time (milliseconds)
Input Size (N)
0
2
4
6
8
10
10
20
30
40
50
60
70
80
90
100

中等规模数据



Linear
O(N log N)
Quadratic
Cubic
Running Time (seconds)
Input Size (N)
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

求渐进表达式

$3n^2 + 10n$　　$n^2/10 + 2^n$　　$21 + 1/n$

$\log n^3 + 9$　　$10\log 3^n$　　$5n^2 + 2n\log n$

$n + \log^2 n^2$　　$\log n + n^{2/3}$　　$n^4 + 2^{n/3}$

算法的输入规模与计算时间分析

算法的输入规模为n时计算时间为T（n）=3*2^n。在某台计算机上实现并完成该算法的时间为t秒。现有另一台计算机，其运行速度为第一台的64倍，那么在这台新机器上用同一算法在t秒内能解决多大规模的问题？

$$t = 3 * 2^n$$

$$64t = 3 * 2^{n'}$$ （速度快64倍等价于时间资源多了64倍）

$$n' = n + 6$$

算法的输入规模和计算时间

若上述算法计算时间改进为T(n)=n^2，其余条件不变，则在新机器上t时间可以解多大规模的问题？

$t=n^2$

$64t=n'^2$

$$n' = 8n$$

若上述算法计算时间改进为T(n)=8，其余条件不变，则在新机器上t时间可以解多大规模的问题？

函数渐进性态分析与证明

对于下列函数f(n)和g（n），确定g（n）是f(n)的上界或下界或同阶函数，并简述理由

$f(n) = \log n^2, g(n) = \log n + 5$

$f(n) = 2^n, g(n) = 3^n$

$f(n) = \log n^2, g(n) = \sqrt{n}$

$f(n) = n, g(n) = \log^2 n$

$f(n) = \log^2 n, g(n) = \log n$

$f(n) = 10, g(n) = \log 10$

函数渐进性态分析与证明

如何证明？

$$f(n) = \log n^2, g(n) = \log n + 5$$

存在c1=2，n1=1，使得对于所有n>=n1时，都有f(n)<=c1*g(n),所以f(n)=O(g(n)).

存在c2=1，n2=32，使得对于所有n>=n2时，都有f(n)>=c2*g(n),所以f(n)=Ω(g(n)).

所以，存在c1=2，c2=1,n0=32,使得对于所有n>=n0时都有c2*g(n)<=f(n)<=c1*g(n)，故f(n)=Θ(g(n)).

函数渐进性态分析与证明

如何证明？

$$f(n)=2^n, g(n)=3^n$$

因为：

$$\lim_{n\to\infty}\frac{2^n}{3^n}=\lim_{n\to\infty}(\frac{2}{3})^n=0$$

所以：

$$f(n)=O(g(n))$$

计算时间复杂性的表示

证明：如果一个算法平均计算时间复杂性是Θ (f(n)),则该算法在最坏的情况下所需的计算时间是Ω(f(n)).

$$T_{avg}(N) = \sum_{I \in D_N} P(I)T(N,I) \le \sum_{I \in D_N} P(I) \max_{I' \in D_N} T(N,I')$$

$$= T(N,I^*) \sum_{I \in D_N} P(I) = T(N,I^*) = T_{\max}(N)$$

$$T_{\max}(N) = \Omega(T_{avg}(N)) = \Omega(\theta(f(n))) = \Omega(f(n))$$