算法设计与分析

西安交通大学
计算机科学与技术学院

课程简介

算法是完成特定任务的有限指令集合。

- 算法在计算机科学中的作用

1）算法是计算机科学研究的核心

计算机科学——研究算法的一门学科

2）算法是计算机求解问题的基础

问题→算法→程序

3）实际问题的求解需要有效的算法设计

无效的算法可能无法解决实际问题

课程简介

要注意区分：

计算机科学与技术——技术专家

应用计算机技术——领域专家

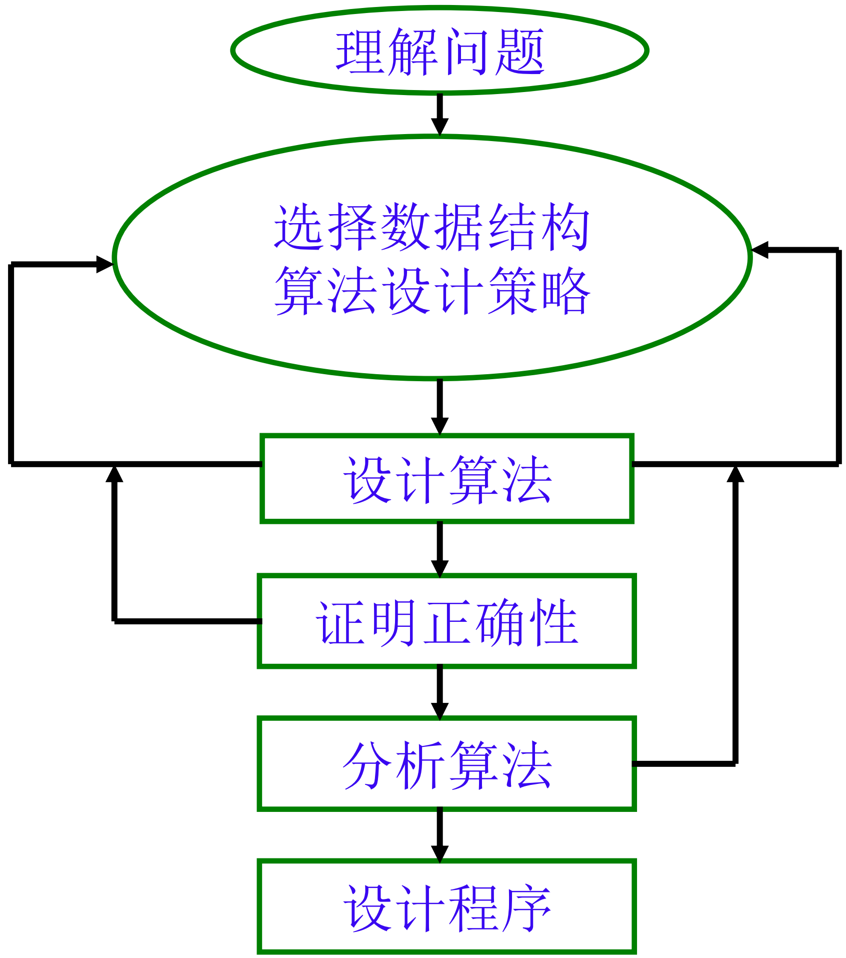
4）算法设计思想与算法分析：

领域与技术的桥梁

5）算法与模型的区别

课程简介

计算机求解问题的基本过程



课程简介

- **算法的研究可以分成四个不同的领域：**

1）怎样设计算法： 学习和实践算法设计的策略、技巧，设计符合问题要求的新算法

2）怎样验证算法： 证明算法可以正确的运行

①程序验证； ②算法证明

3）怎样分析算法： 分析算法的时间/空间复杂性

4）怎样测试算法： 软件调试与评估

本课程主要集中于算法的设计与分析。

课程简介

- 课程目的和意义：
  - 介绍算法及算法分析的基本知识，使大家了解并掌握计算机算法的基础理论和基本分析方法。
  - 介绍各种算法的设计策略，使大家理解和掌握各种算法设计策略的思想和技巧。
  - 对算法复杂性理论和NP-完全问题进行讲解，为独立设计算法和对算法进行复杂性分析奠定基础。

课程简介

- 学习本课程的方法：
  - **通过案例理解和思考算法设计思想。**
  - **切勿拘泥于案例。**
  - **动手写程序！不要止步于书上的程序。**
  - **对同一个问题用不同的算法设计思想设计算法并进行横向比较。**
  - **多看参考书和参考资料，从应用端深入理解算法。**

课程简介

- 本课程的考核方式：
  - **每章作业+大作业+闭卷考试。**
  - **考试成绩占60%，题型包括简答、算法分析解答、算法设计和实现。**

算法举例

1、大整数乘法

$$981 \times 1234 = 1210554$$

如果计算机只能计算两位数的乘法，利用分治法可以将上面的乘数和被乘数分割成两个两位数。

令：$w=09$，$x=81$，$y=12$，$z=34$，

则：$981 \times 1234 = (10^2 w + x) \times (10^2 y + z)$

$$= 10^4 wy + 10^2 (wz + xy) + xz$$

$$= 1080000 + 127800 + 2754$$

$$= 1210554$$

上述计算过程需要4次两位数乘法运算

算法举例（续）

2、大整数乘法改进

考虑到乘积：

$$r=(w+x)\times(y+z)=wy+(wz+xy)+xz$$

所以：$(wz+xy)=r-wy-xz$

因此，大整数乘法也可以用下列过程实现：

$$p=wy=09\times 12=108$$

$$q=xz=81\times 34=2754$$

$$r=(w+x)\times(y+z)=90\times 46=4140$$

$$\begin{aligned}981\times 1234&=10^4p+10^2(r-p-q)+q\\&=1080000+127800+2754\\&=1210554\end{aligned}$$

这个过程需要3次两位整数乘法运算。

算法举例（续）

3、判定问题

给定12枚硬币，它们重量或者全部相等，或者其中一枚与其它11枚重量不等。能否用天平在三次内判断这些硬币重量是否相同？如果不相同，找出那个不相同的硬币，并判定它比其他硬币重或轻.

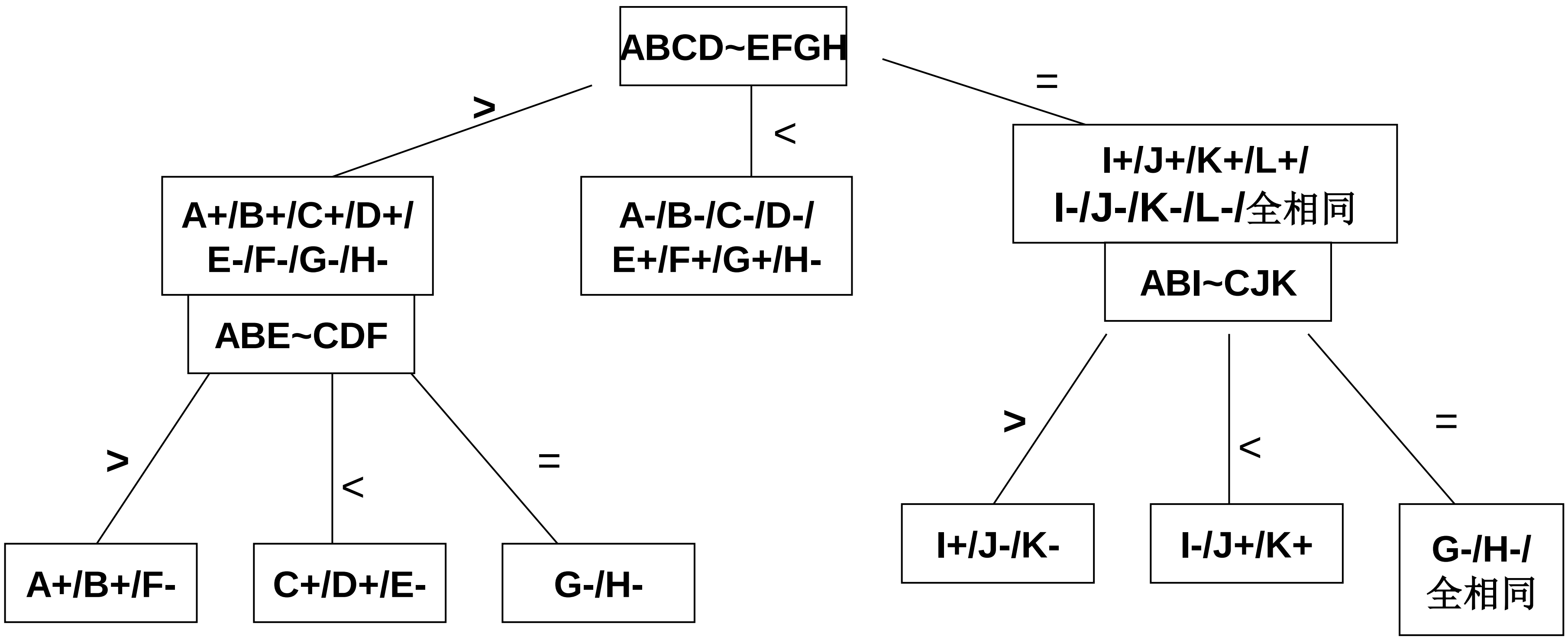
设硬币分别为：ABCDEFGHIJKL

可能的状态：全相等（1种）；其中一枚重（12种）；其中一枚轻（12种）。共有25种可能状态。

将12枚硬币分成三组：ABCD　EFGH　IJKL, 用天平称其中两组的重量，可以产生3种可能。按照此方法，三次称量可以产生27种状态，因此三次内是可以区分25种状态。

6、算法举例（续）

3、判定问题



注：图中A+表示A硬币重，图中A-表示A硬币轻，其它类同。

第1章 算法引论

本章主要知识点：

- 1.1 算法与程序
- 1.2 算法复杂性
- 1.3 复杂性渐进表示
- 1.4 算法复杂性分析方法

1.1 算法与程序

算法：是满足下述性质的指令序列。

- 输　入：有零个或多个外部量作为算法的输入。
- 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清晰、无歧义。
- 有限性：算法中每条指令的执行次数有限，执行每条指令的时间也有限。
- 可行性：每个指令原则上都能精确地用有限的运算即可完成。

1.1 算法与程序

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)即有限性。

- 例如操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
- 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

1.2 算法复杂性

算法复杂性是算法运行所需要的计算机资源的量，需要时间资源的量称为**时间复杂性**，需要的空间资源的量称为**空间复杂性**。

对于给定的算法，其复杂性是只依赖于算法要解的问题的规模、算法的输入的函数。

如果分别用n和I表示算法要解问题的规模和算法输入实例，其中I=size(I)，若用C表示复杂性，那么，应该有：

$$C=F(I)$$

一般把时间复杂性和空间复杂性分开，并分别用T和S来表示，则有： T=T(I)和S=S(I) 。

1.2 算法复杂性

设I是问题的规模为n的实例，则定义：

（1）**最坏情况**下的时间复杂性

$$T_{max}(n) = \max\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$$

（2）**最好情况**下的时间复杂性

$$T_{min}(n) = \min\{ T(\mathrm{I}) \mid \mathrm{size}(\mathrm{I})=n \}$$

（3）**平均情况**下的时间复杂性

$$T_{avg}(n) = \sum p(\mathrm{I})T(\mathrm{I})$$

其中， $\mathrm{size}(\mathrm{I})=n$，$p(\mathrm{I})$是实例I出现的概率。

1.3 复杂性的渐进表示

$T(n) \to \infty$， as $n \to \infty$；

$(T(n) - t(n))/ T(n) \to 0$， as $n \to \infty$；

$t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。

在数学上，$t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项。它比$T(n)$ 简单。

渐进复杂性表示：O、Ω、Θ 、o 、ω

1.3 复杂性的渐进表示

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

（1）**渐近上界记号*O***

$O(g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$ 有：$0 \leq f(n) \leq cg(n) \}$

（2）**渐近下界记号Ω**

$\Omega (g(n)) = \{ f(n) \mid$ 存在正常数c和n_0使得对所有$n \geq n_0$ 有：$0 \leq cg(n) \leq f(n) \}$

（3）**紧渐近界记号Θ**

$\Theta (g(n)) = \{ f(n) \mid$ 存在正常数c_1, c_2和n_0使得对所有 $n \geq n_0$有：$c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

定理1： $\Theta (g(n)) = O (g(n)) \cap \Omega (g(n))$

1.3 复杂性的渐进表示

（4） **非紧上界记号*o***

$o(g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，存在正数和$n_0>0$ 使得对所有$n \geq n_0$有：$0 \leq f(n)<cg(n) \}$

等价于 $f(n) / g(n) \to 0$ ， as $n \to \infty$。

（5） **非紧下界记号ω**

$\omega(g(n)) = \{ f(n) \mid$ 对于任何正常数$c>0$，存在正数和$n_0>0$ 使得对所有$n \geq n_0$有：$0 \leq cg(n) < f(n) \}$

等价于 $f(n) / g(n) \to \infty$ ， as $n \to \infty$。

$$f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$$

1.3 复杂性的渐进表示

渐近分析记号在等式和不等式中的意义

- $f(n)=\Theta(g(n))$的确切意义是：$f(n)\in\Theta(g(n))$。
- 一般情况下，等式和不等式中的渐近记号$\Theta(g(n))$表示$\Theta(g(n))$中的某个函数。
- 例如：$2n^2+3n+1=2n^2+\Theta(n)$ 表示
- $2n^2+3n+1=2n^2+f(n)$，其中$f(n)$ 是$\Theta(n)$中某个函数。
- 等式和不等式中渐近记号O, o, Ω和ω的意义是类似的。

1.3 复杂性的渐进表示

渐近表示记号的若干性质

（1）传递性：

- $f(n)=\Theta(g(n))$， $g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n))$；
- $f(n)=O(g(n))$， $g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$；
- $f(n)=\Omega(g(n))$， $g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$；
- $f(n)=o(g(n))$， $g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$；
- $f(n)=\omega(g(n))$， $g(n)=\omega(h(n)) \Rightarrow f(n)=\omega(h(n))$；

渐近表示记号的若干性质

（2）自反性：

- $f(n) = \Theta(f(n))$；
- $f(n) = O(f(n))$；
- $f(n) = \Omega(f(n))$.

（3）对称性：

- $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.

（4）互对称性：

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

渐近表示记号的若干性质

（5）算术运算：

- $O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$；
- $O(f(n))+O(g(n)) = O(f(n)+g(n))$；
- $O(f(n))*O(g(n)) = O(f(n)*g(n))$；
- $O(cf(n)) = O(f(n))$；
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(n))$。

渐近表示记号的若干性质

规则$O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$ 的证明：

对于任意$f_1(n) \in O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n \geq n_1$，有$f_1(n) \leq c_1 f(n)$。

类似地，对于任意$g_1(n) \in O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n \geq n_2$，有$g_1(n) \leq c_2 g(n)$。

令$c_3=\max\{c_1, c_2\}$，$n_3 =\max\{n_1, n_2\}$，$h(n)= \max\{f(n),g(n)\}$。

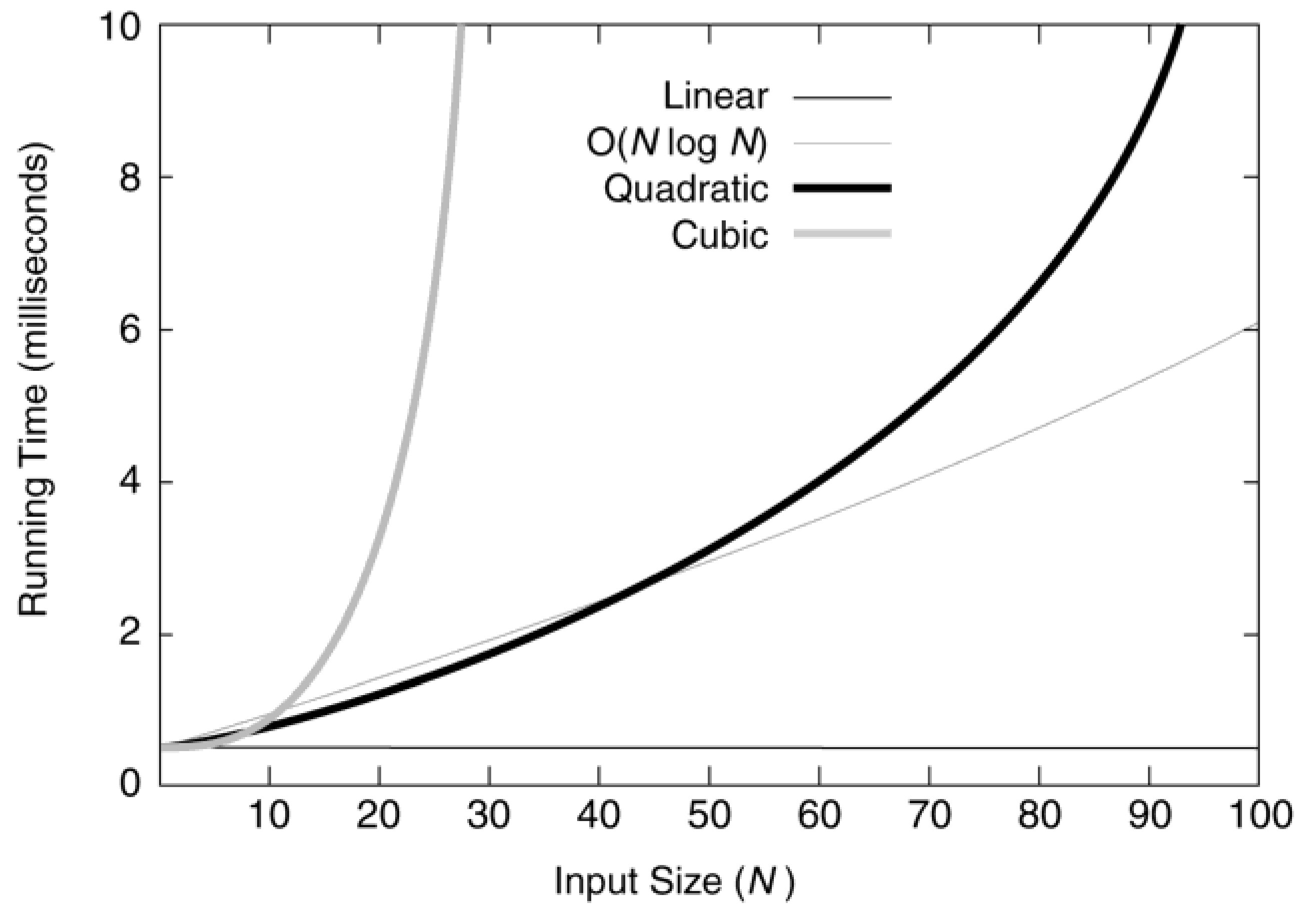
则对所有的 $n \geq n_3$，有

$$
\begin{aligned}
f_1(n) + g_1(n) &\leq c_1 f(n) + c_2 g(n) \\
&\leq c_3 f(n) + c_3 g(n) = c_3(f(n) + g(n)) \\
&\leq c_3 2 \max\{f(n),g(n)\} \\
&= 2c_3 h(n) = O(\max\{f(n),g(n)\}).
\end{aligned}
$$

算法分析中常见的复杂性函数

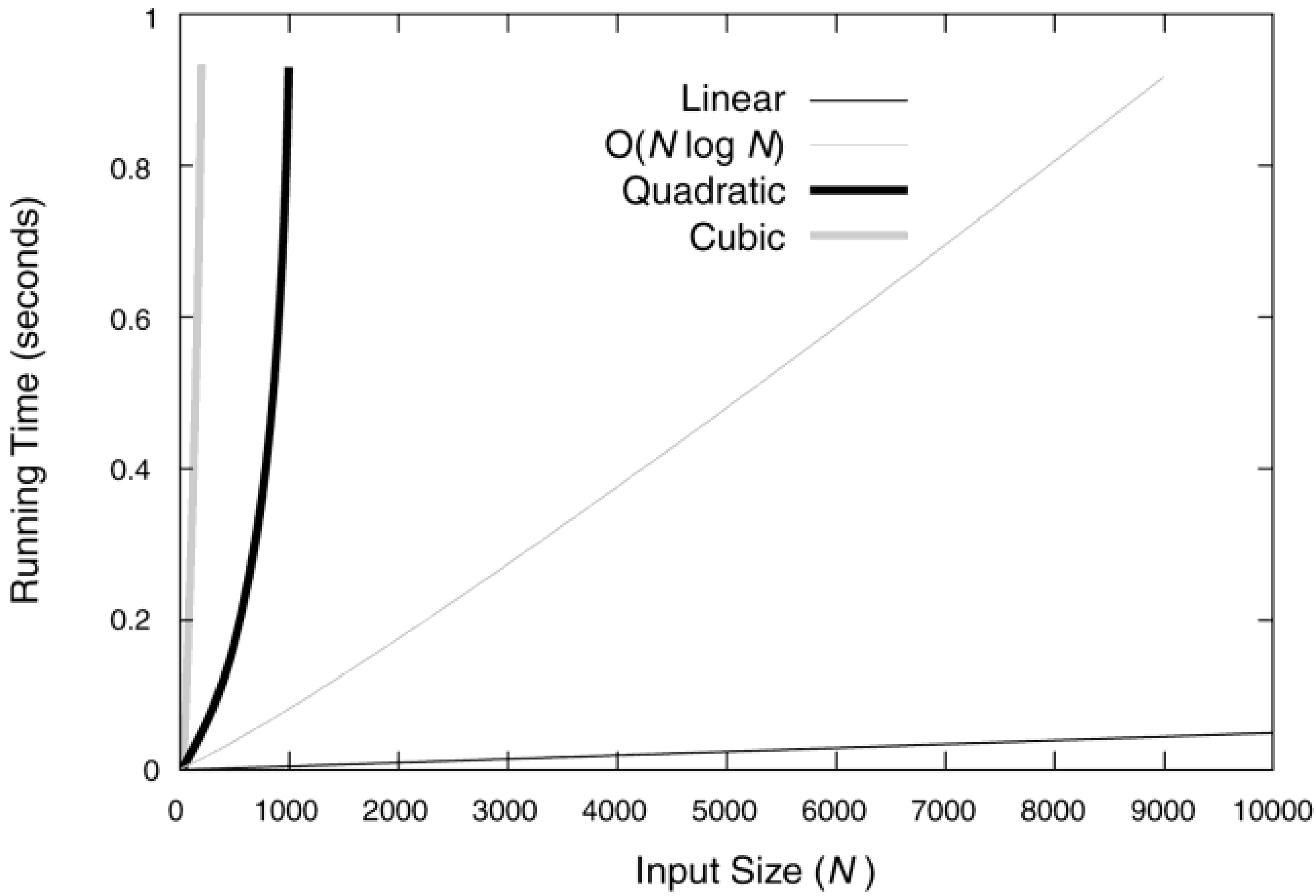
FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

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

1.4 算法复杂性分析方法

事后统计的方法

上机运行后，机器自动计时。

优点：不需要复杂的数学分析

缺点：耗时因程序处理的数据量、机器配置（甚至硬盘空间）的不同而不同；

必须运行程序后才能分析（对某些不可实际执行的程序无法用此方法）。

事前分析的方法

如果我们对于某个问题设计出了解题算法，或者已有若干解此问题的算法，如何对它进行分析呢？具体分析些什么呢？

算法分析的基本法则

- （1）for / while 循环

  循环体内计算时间*循环次数；
- （2）嵌套循环

  循环体内计算时间*所有循环次数；
- （3）顺序语句

  各语句计算时间相加；
- （4）if-else语句

  if语句计算时间和else语句计算时间的较大者。
- （5）过程调用

  过程体调用时间+过程体执行时间。

```
void insertion_sort(Type *a, int n)
{
    Type key;                              //  cost     times
    for (int i = 1; i < n; i++){           //  c1       n
        key=a[i];                          //  c2       n-1
        int j=i-1;                         //  c3       n-1
        while( j>=0 && a[j]>key ){         //  c4       sum of ti
            a[j+1]=a[j];                   //  c5       sum of (ti-1)
            j--;                           //  c6       sum og (ti-1)
        }
       a[j+1]=key;                         //  c7       n-1
      }
}
```

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{i=1}^{n-1} t_i + c_5\sum_{i=1}^{n-1}(t_i - 1) + c_6\sum_{i=1}^{n-1}(t_i - 1) + c_7(n-1)$$

- 在最好情况下，t_i=1, for 1 ≤ i <n;

$$T_{\min}(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = O(n)$$

- 在最坏情况下，$t_i \le i+1$, for 1 ≤ i <n;

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

$$T_{\max}(n) \le c_1 n + c_2(n-1) + c_3(n-1) +$$

$$c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$$

$$= O(n^2)$$

- 对于输入数据a[i]=n-i,i=0,1,…,n-1，算法insertion_sort 达到其最坏情形。因此，

$$T_{\max}(n) \geq \frac{c_4+c_5+c_6}{2}n^2+\left(c_1+c_2+c_3+\frac{c_4-c_5-c_6}{2}+c_7\right)n-(c_2+c_3+c_4+c_7)$$

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

- 由此可见，$T_{\max}(n)=\Theta(n^2)$

【程序步】指词法或语法上的可测度程序段, 其执行时间为常量，与问题规模无关。

- 为了用估计值代替精确值，对程序步执行次数计数，程序步执行次数与机器无关，每个程序步执行时间为O(1)。
- 一个程序执行工作量的统计可以是不同的。

例如: x = 2; 计为一次, y = 3 * x - 4;同样也可计为一次。

- 程序执行工作量的统计必须与常量无关。

例如: x =1000个数的和 计为一次, 而 x = n 个数的和 却不能计为一次。

```
void insertion_sort(Type *a, int n)
{
    Type key;                          //  cost    times
    for (int i = 1; i < n; i++){       //   1       n
        key=a[i];                      //   1       n-1
        int j=i-1;                     //   1       n-1
        while( j>=0 && a[j]>key ){     //   1       sum of ti
            a[j+1]=a[j];               //   1     sum of (ti-1)
            j--;                       //   1     sum og (ti-1)
        }
        a[j+1]=key;                    //   1       n-1
    }
}
```

$$T(n) = n + (n-1) + (n-1) + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} (t_i - 1) + \sum_{i=1}^{n-1} (t_i - 1) + (n-1)$$

- 在最好情况下，$t_i=1$, for $1 \le i < n$;

$$T_{\min}(n) = n + (n-1) + (n-1) + (n-1) + (n-1)$$

$$= 5n - 4 = O(n)$$

- 在最坏情况下，$t_i \le i+1$, for $1 \le i < n$;

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

$$T_{\max}(n) \le n + (n-1) + (n-1) +$$

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

$$= \frac{3}{2} n^2 + \frac{9}{2} n - 4 = O(n^2)$$