算法分析与设计

西安交通大学
计算机学院

课程简介

- 授课教师：朱晓燕
  - 计算机学院，副教授
  - 邮箱： xjtuwill@163.com
  - 研究方向：机器学习，数据挖掘
  - 讲授课程
    - 本科生：算法分析与设计
    - 研究生：机器学习与数据挖掘
- QQ群： 1031157914
- 思源学堂：bb.xjtu.edu.cn
- 助教：赵明宽、张桢权

课程简介

- 学时48：其中理论课32学时，课内实验16学时
- 考核：平时成绩30%(包括作业和实验)，期末考试70%
- **教材：**王晓东，算法设计与分析（第四版），清华大学出版社

参考教材：

- [1] 黄宇. 算法设计与分析（第1版）。机械工业出版社，2017。
- [2] H.Ellis, S.Sartaj, R.Sanguthevar著，冯博琴等译。 计算机算法. 机械工业出版社，2006。
- [3] B.Gilles, B.Paul著，邱仲潘等译. 算法基础。清华大学出版社，2005

课程简介

- 课程内容：
  - 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

课程目标

- 1. 掌握算法及其复杂性分析的基本理论、基础知识和基本方法，能够对具体算法的性能进行分析和评估；理解问题计算复杂性的概念，能够对问题的计算复杂性进行分析和判定。具有对实际计算问题的分析能力。（毕业要求B2）；
- 2. 掌握各种典型算法的设计策略，能够熟练运用各种算法设计策略，根据实际问题的特征选用合适的算法设计策略，对具体问题能够进行算法设计。具有解决实际问题的能力，能够针对复杂性较高的问题设计实用的快速求解算法，并能进行测试和实验验证。（毕业要求A2，C2）。

备注：毕业要求

- A2：能针对具体的对象建立数学模型并求解;
- B2：能基于相关科学原理和数学模型方法正确表达复杂工程问题
- C2：能够针对特定需求，完成单元（部件）的设计

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高效的算法

低效的算法可能无法解决大型复杂的问题

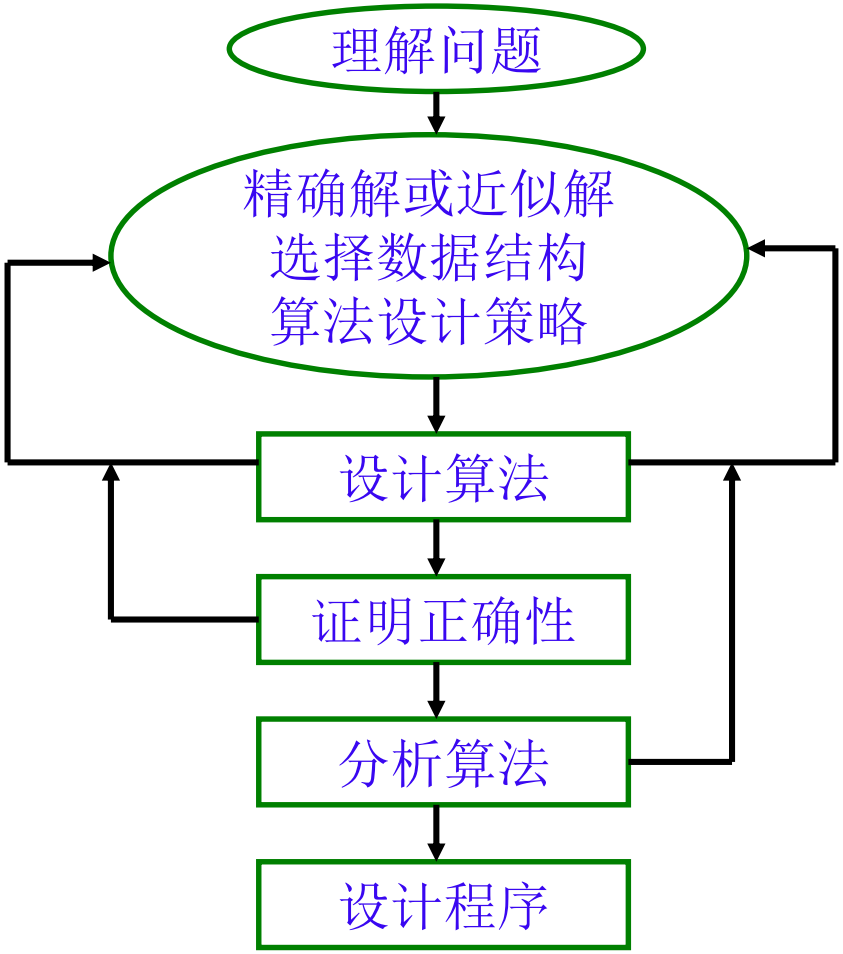
算法简介

算法是完成特定任务的有限指令集合，
满足下述性质：

1. 输　入：有零个或多个外部量作为算法的输入。
2. 输　出：算法产生至少一个量作为输出。
3. 确定性：组成算法的每条指令清晰、无歧义。
4. 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。
5. 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。

算法简介

计算机求解问题的基本过程



算法简介

- 算法的研究可以分成四个不同的领域：

1）怎样设计算法： 算法设计的策略，技巧
2）怎样验证算法： 验证算法可以正确运行。
①程序验证； ②算法证明
3）怎样分析算法： 算法分析，分析算法的性能（时间/空间）
4）怎样测试算法： 包括调试、评测

本课程主要集中于算法的设计与分析。

算法举例

3、判定问题

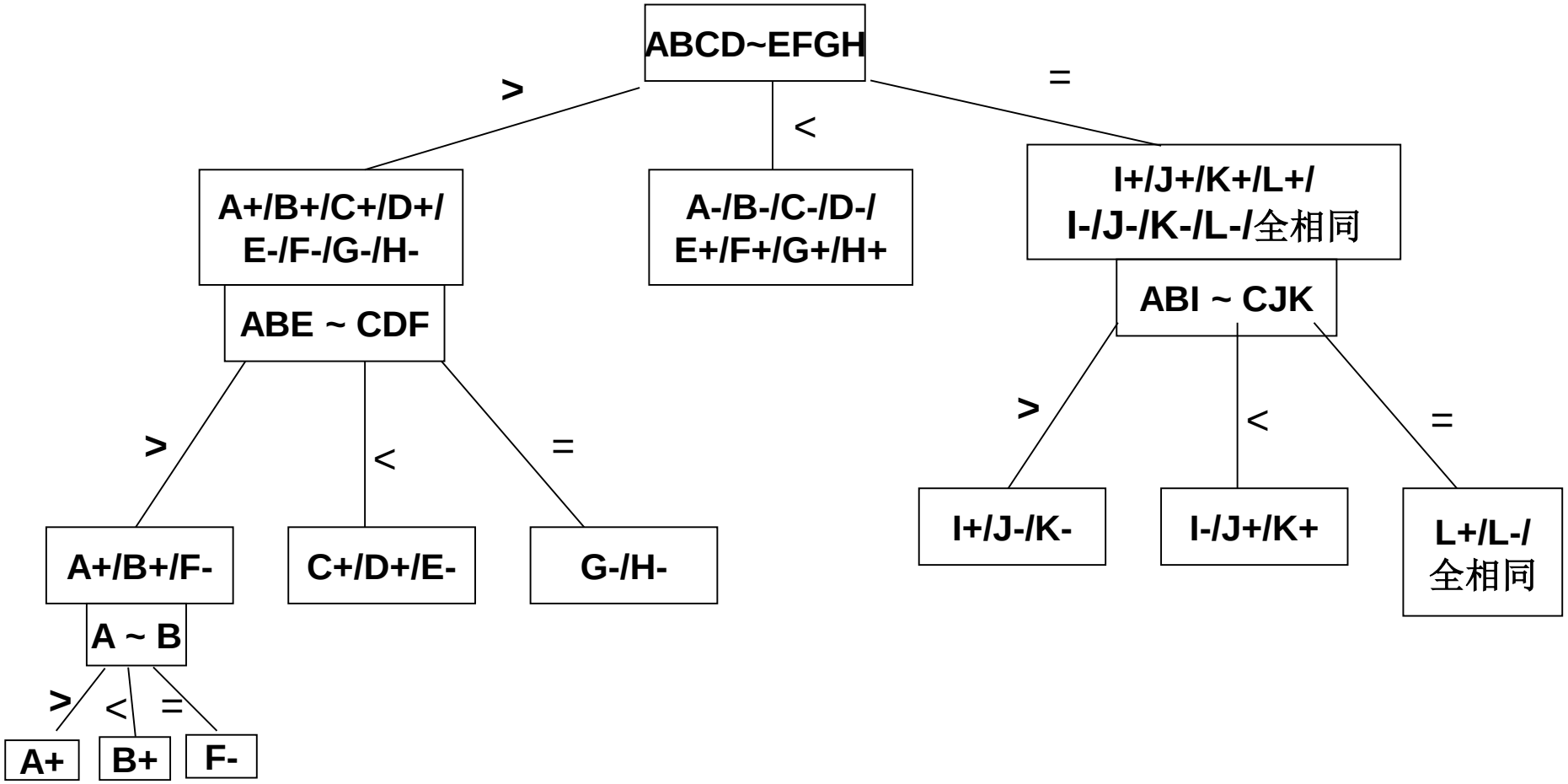
给定12枚硬币，它们重量或者全部相等，或者其中一枚与其它11枚重量不等。能否用天平在三次内判断这些硬币重量是否相同？如果不相同，找出那个不相同的硬币，并判定它比其他硬币重或轻.

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。

算法举例（续）



注：图中A+表示A硬币重，A-表示A硬币轻，其它类同。

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，需要的时间资源的量称为**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和算法的输入的函数。

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n=size(I)，若用C表示复杂性，那么，应该有： C=F(n, 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(n, I)和S=S(n, I) 。简化为：T=T(n)和S=S(n) 。

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

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

（2）**最好情况**下的时间复杂性

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

（3）**平均情况**下的时间复杂性

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

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

可操作性最好且最有实际价值的是最坏情况下的时间复杂度

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 $n \to \infty$，则$t(n)$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。

- 渐进复杂性表示记号：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，$f(n) \geq 0$，$g(n) \geq 0$。

表示的是$f(n)$的复杂性。

（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。

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）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：

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

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

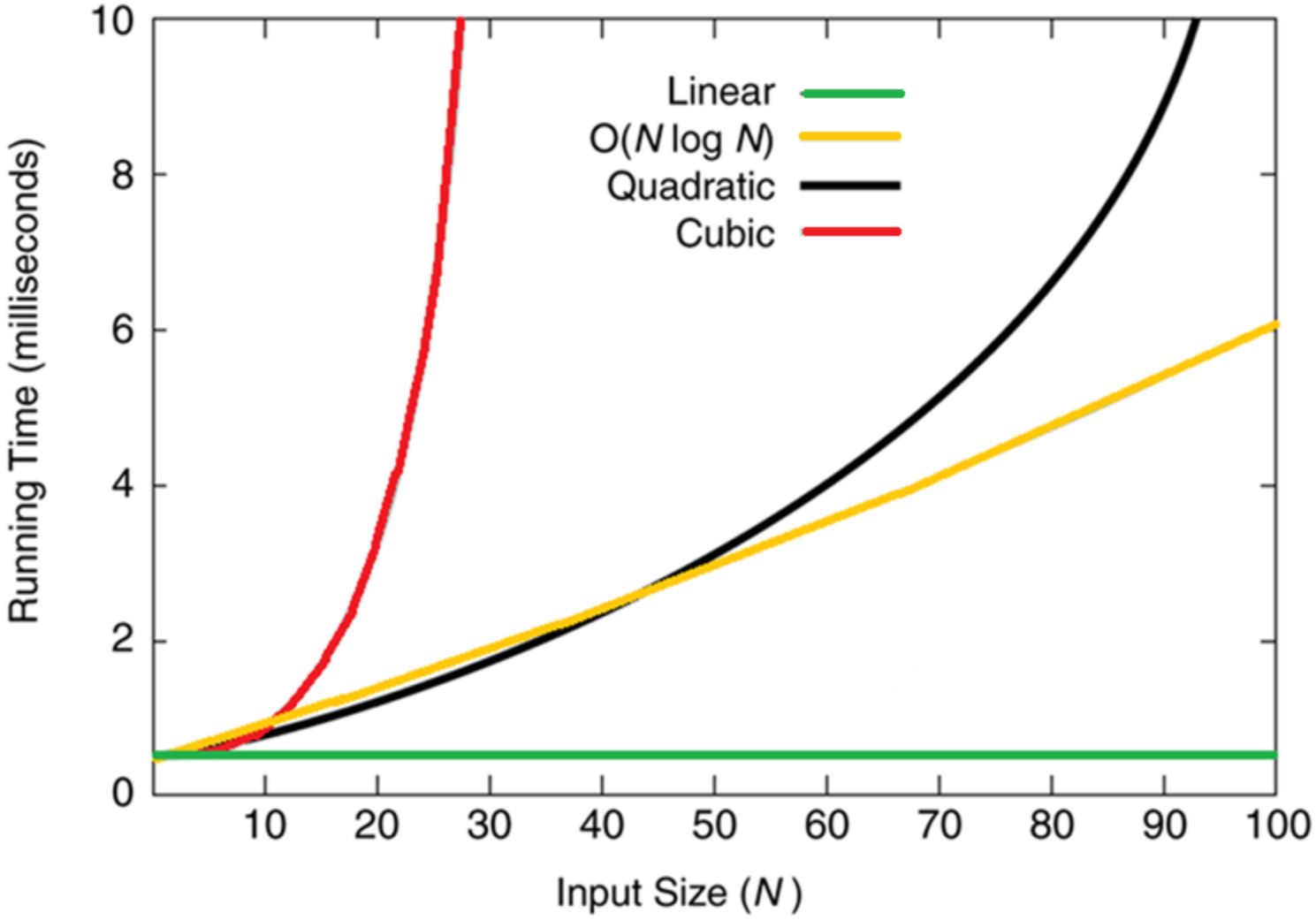
$$
\begin{aligned}
f_1(n)+g_1(n) &\leq c_1 f(n) + c_2 g(n) \\
&\leq c_3 f(n) + c_3 g(n) = c_3(f(n) + g(n)) \\
&\leq c_3 2 \max\{f(n), g(n)\} \\
&= 2c_3 h(n)
\end{aligned}
$$

故$O(f(n))+O(g(n)) = O(max\{f(n), g(n)\})$

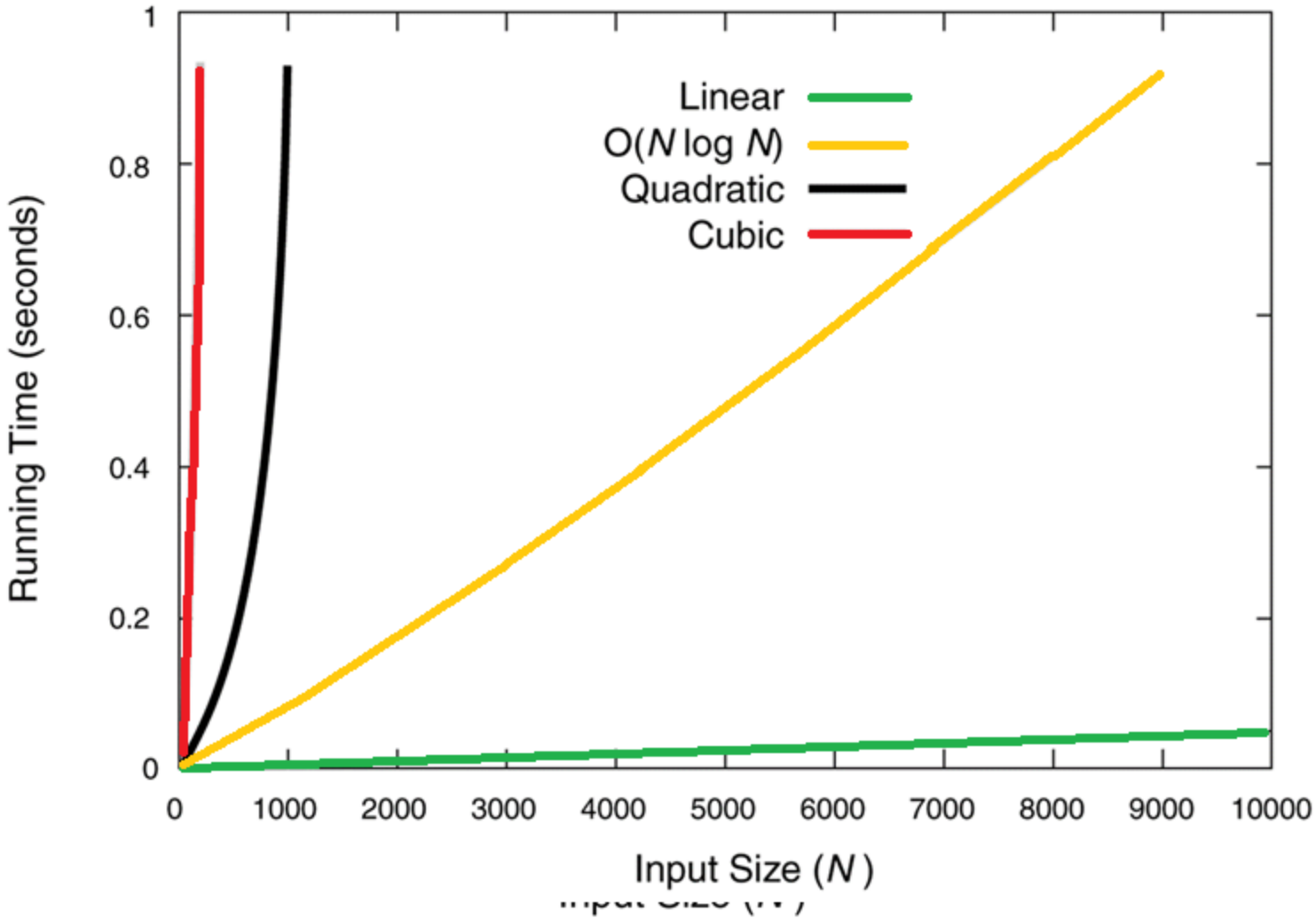
算法分析中常见的复杂性函数

Function	Name
c	Constant
logN	Logarithmic
$\log^2 N$	Log-square
N	Linear
NlogN	NlogN
N^2	Quadratic
N^3	Cubic
2^N	Exponential
N!	Factorial
N^N	-

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

大O比率定理

- 【大O比率定理】对于函数f (n)和g (n)，若

  $\lim_{n\to\infty} f(n)/g(n)$ 存在，则f (n)= O(g (n))当且仅当存在确定的正常数c，有 $\lim_{n\to\infty} f(n)/g(n)\leq c$

证明：如果f (n)= O(g (n))，则存在c> 0及某个n0，使得对于所有的n≥n0，有f (n) ≤cg(n) ,从而f (n) /g(n)≤c，因此

$\lim_{n\to\infty} f(n)/g(n)\leq c$。

接下来假定 $\lim_{n\to\infty} f(n)/g(n)\leq c$，它表明存在一个n0，使得对于所有的n≥n0，有 f(n)≤max{1, c} *g (n)。

大Ω比率定理

- 【大Ω比率定理】对于函数$f(n)$和$g(n)$，若$\lim_{n\to\infty} f(n)/g(n)$存在，则$f(n)=\Omega(g(n))$当且仅当存在确定的正常数c，有$\lim_{n\to\infty} f(n)/g(n)\geq c$。

证明：如果$f(n)=\Omega(g(n))$，则存在$c>0$及某个n_0，使得对于所有的$n\geq n_0$，有$f(n)/g(n)\geq cg(n)$, 从而$f(n)/g(n)\geq c$，因此$\lim_{n\to\infty} f(n)/g(n)\geq c$。

接下来假定$\lim_{n\to\infty} f(n)/g(n)\geq c$，它表明存在一个$n_0$，使得对于所有的$n\geq n_0$，有$f(n)\geq c*g(n)$。

例题

多项式函数

已知：$f(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$；　$a_d>0$;

证明：$f(n) = \Theta(n^d)$;

对于所有的n≥1有：$f(n) \le \sum_{i=0}^{d} |a_i| n^i$

$$= n^d \sum_{i=0}^{d} |a_i| n^{i-d}$$

$$\le n^d \sum_{i=0}^{d} |a_i|$$

故 $f(n) = O(n^d)$

例题

多项式函数

已知：$f(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$；　$a_d>0$;

证明：$f(n) = \Theta(n^d)$;

对于所有的n≥1有：$f(n) \geq -\sum_{i=0}^{d-1}|a_i|n^i + a_dn^d$

$$\geq -\frac{\sum_{i=0}^{d-1}|a_i|}{n} * n^d + a_dn^d$$

取n_0，使得$n \geq n_0$时，$a_d \geq 2\frac{\sum_{i=0}^{d-1}|a_i|}{n}$

则$f(n) \geq \frac{\sum_{i=0}^{d-1}|a_i|}{n} * n^d$

故 $f(n) = \Omega(n^d)$

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 t_i
            a[j+1]=a[j];               //  c5        sum of (t_i-1)
            j--;                       //  c6        sum of (t_i-1)
        a[j+1]=key;                    //  c7        n-1
}
```

注：若无特别说明，为显示整洁，本课所有ppt代码中省略大括号，以缩进表示。

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{i=1}^{n-1}t_i+c_5\sum_{i=1}^{n-1}(t_i-1)+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7(n-1)$$

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{i=1}^{n-1}t_i+c_5\sum_{i=1}^{n-1}(t_i-1)+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7(n-1)$$

在最好情况下，$t_i=1$, for $1 \le i < n$;

$$T_{\min}(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4(n-1)+c_7(n-1)$$

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

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{i=1}^{n-1}t_i+c_5\sum_{i=1}^{n-1}(t_i-1)+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7(n-1)$$

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

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

$$T_{\max}(n)=c_1n+c_2(n-1)+c_3(n-1)+$$

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

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

$$=O(n^2)$$

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。

【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，程序步执行时间与机器无关，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 t_i
            a[j+1]=a[j];                   //   1       sum of (t_i-1)
            j--;                           //   1       sum of (t_i-1)
        a[j+1]=key;                        //   1       n-1
}
```

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

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

- 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，$t_i = i+1$, for $1 \le i < n$;

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

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

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

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

1.3.2 渐近复杂性分析的数学基础

（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。

取整函数的若干性质

- $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$;
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
- 对于$n \geq 0$，$a,b>0$，有：
  - $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
  - $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
- $\lceil a/b \rceil \leq (a+(b-1))/b$;
- $\lfloor a/b \rfloor \geq (a-(b-1))/b$;
- $f(x) = \lfloor x \rfloor$, $g(x) = \lceil x \rceil$为单调递增函数。

（3）多项式函数

- $p(n) = a_0+a_1n+a_2n^2+\ldots+a_dn^d;\quad a_d>0;$
  - $p(n) = \Theta(n^d);$
  - $k \geq d \Rightarrow p(n) = O(n^k)$;
  - $k \leq d \Rightarrow p(n) = \Omega(n^k)$;
  - $k > d \Rightarrow p(n) = o(n^k)$;
  - $k < d \Rightarrow p(n) = \omega(n^k)$;
- $f(n) = O(n^k) \Leftrightarrow f(n)$多项式有界;
- $f(n) = O(1) \Leftrightarrow f(n) \leq c;$

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

（4）指数函数

- $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$
  - $e^x \geq 1+x;$
  - $|x| \leq 1 \Rightarrow 1+x \leq e^x \leq 1+x+x^2$;
  - $e^x = 1+x+ \Theta(x^2),$ as $x \to 0;$
  - $\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$

（5）对数函数

- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$;
- $\log \log n = \log(\log n)$;
- for a>0,b>0,c>0

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

（5）对数函数

- $\log_c(ab) = \log_c a + \log_c b$
- $\log_b a^n = n \log_b a$
- $\log_b a = \frac{\log_c a}{\log_c b}$
- $\log_b(1/a) = -\log_b a$
- $\log_b a = \frac{1}{\log_a b}$
- $a^{\log_b c} = c^{\log_b a}$

（5）对数函数

- $|x| \le 1 \Rightarrow \quad \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots.$
- for $x > -1$, $\frac{x}{1+x} \le \ln(1+x) \le x$
- for any $a > 0$, $\lim_{n\to\infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n\to\infty} \frac{\log^b n}{n^a} = 0,$

$$\Rightarrow \log^b n = o(n^a)$$

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

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

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

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

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

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

（7）求和公式

- $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$
- $\sum_{i=1}^{n} i^2 = \frac{1}{6}(2n^3 + 3n^2 + n)$
- $\sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a} \quad for \quad a \neq 1$
- $\sum_{i=1}^{n} ia^i = \frac{a}{(1-a)^2}(1 - a^n - na^n + na^{n+1}) \quad for \quad a \neq 1$

证明如下：

(7) 求和公式 $\sum_{i=1}^{n} ia^i = \frac{a}{(1-a)^2}(1 - a^n - na^n + na^{n+1}) \quad for \quad a \neq 1$

$$\because \quad \sum_{i=1}^{n} ia^i = \sum_{i=1}^{n} a^i + \sum_{i=1}^{n} (i-1)a^i = \sum_{i=1}^{n} a^i + a\sum_{i=0}^{n-1} ia^i$$

$$= \frac{a - a^{n+1}}{1-a} + a\sum_{i=0}^{n} ia^i - na^{n+1}$$

右侧的$a\sum_{i=0}^{n} a^i$移到左侧合并

$$\therefore \quad (1-a)\sum_{i=1}^{n} ia^i = \frac{a - a^{n+1}}{1-a} - na^{n+1}$$

$$= \frac{a}{1-a}\left(1 - a^n - na^n + na^{n+1}\right)$$

即：$$\sum_{i=1}^{n} ia^i = \frac{a}{(1-a)^2}\left(1 - a^n - na^n + na^{n+1}\right)$$

（7）求和公式

$$\sum_{i=1}^{n} ia^i = \frac{a}{(1-a)^2}\left(1-a^n-na^n+na^{n+1}\right)$$

特别地：

当$a=1/2$时，$$\sum_{i=1}^{n}\frac{i}{2^i}=2-\frac{3n+2}{2^n}$$

当$a=2$时，$$\sum_{i=1}^{n} i2^i = 2+(n-1)2^{n+1}$$

（7）求和公式

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \ln n + r$$ (欧拉常数 r ≈ 0.5772156649)

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \int_1^{n+1} \frac{1}{\lfloor x \rfloor}$$

当$x \geq 1$时 $\quad \frac{1}{x} \leq \frac{1}{\lfloor x \rfloor} \quad \therefore \quad \int_1^{n+1} \frac{1}{x} \leq \int_1^{n+1} \frac{1}{\lfloor x \rfloor} \Rightarrow \ln(n+1) \leq H_n$

当$x \geq 2$时 $\quad \frac{1}{\lfloor x \rfloor} \leq \frac{1}{x-1} \quad \therefore \quad \int_2^{n+1} \frac{1}{\lfloor x \rfloor} \leq \int_2^{n+1} \frac{1}{x-1} \Rightarrow H_n - 1 \leq \ln(n)$

$$\therefore \quad \ln(n+1) \leq H_n \leq 1 + \ln n$$

即 $\quad H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n)$

例：堆构建的时间复杂度分析

假定对于一个有n个顶点的完全二叉树，当该树为满二叉树时需要筛选调整的次数最多，此时有$n=2^d-1$，其中$d=\lceil \log n \rceil$。

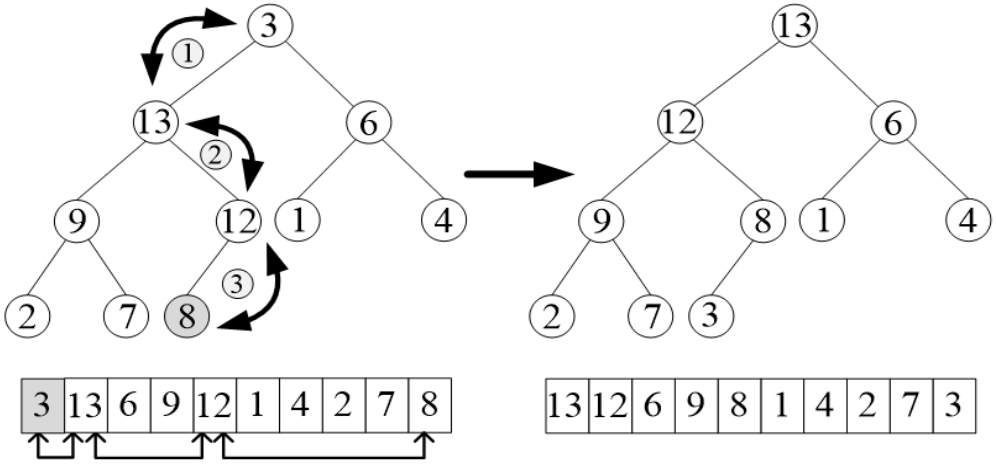


图 5-47 筛选法调整堆

对于一个深度为d的满二叉树，第$k(k \geq 0)$层中有2^k个结点，每个结点最多需要向下调整d - k - 1层，即从该结点所在层移动至叶子结点层。在最坏情况下，建堆需要比较的次数（每个节点与左右子节点各比较一次）为：

$$2\sum_{k=0}^{d-1} 2^k(d-k-1) = 2\left[(d-1)\sum_{k=0}^{d-1} 2^k - \sum_{k=0}^{d-1} k2^k\right] = 2[2^d - d - 1] = 2[n - \lceil \log n \rceil]$$