

# 前言

- 计算机用途
  - 教育, 商业, 研究, 娱乐, 制造业, 健康管理, 战争
- 两个推力
  - 微电子学和芯片设计技术带来更快的硬件
  - 高效的算法

# 前言

## 1、重要性

**程序=算法+数据结构**

**算法的设计与分析是计算机科学的核心问题之一，是重要的专业基础课**

## 2、但是，很多算法是现成的

**数据库**

**网络**

# 前言： 算法概念

- 什么是算法？
  - 解决给定问题的一个计算序列
  - 输入
  - 输出
  - 映射或者转换过程
  - 结束
- 例子： 排序
  - 输入： 一个序列
  - 输出： 该序列的非降序排列
  - 输入： 31, 41, 59, 26, 41, 58

算法： 若干条指令构成的有穷序列，满足4个特征

- 1) 输入
- 2) 输出
- 3) 确定性
- 4) 有限性

# 前言 算法概念：问题及问题实例

- 问题
  - 规定了输入与输出之间的关系，可以用通用语言来描述
- 问题实例
  - 某一个问题的实例包含了求解该问题所需的输入
- 如：将一系列数按非降顺序排序（排序问题）
  - 输入：由 $n$ 数组成的一个序列
  - 输出：对输入系列的一个排列(重排)



- 输入实例
- 问题的具体计算例子。
- 如，排序问题的3个输入实例：
- ① 13,5,6,37,8,92,12
- ② 43,5,23,76,25
- ③ 53,67,32,42,22,33,4,39,56
- 问题规模:算法的输入实例大小。
  - 如上面排序问题的3个输入实例的规模大小分别为7,5,9

# 前言

- 算法的正确性
  - 如果一个算法对问题每一个输入实例，都能输出正确的结果并停止，则称它为正确的。
- 不正确的算法
  - 不停止
  - 给出的不是预期的结果
  - 可能是有用的

# 前言

- 哪些问题需要算法解决
  - 人类基因组计划
  - 互联网
  - 电子商务
  - 制造业
  - 矩阵序列乘积
- 两个特点
  - 有很多候选方案，从中找出所需要的
  - 有实际的应用

# 前言

- 如果计算机无限快、存储器都是免费的,算法研究是否还需要?
  - ① Yes! 证明方案是正确的, 可以给出正确结果
  - ② Yes! 希望自己的实现符合良好的软件工程实践要求, 采用最容易的实现方法。
- 算法对于当代计算机非常重要! 对比硬件与软件的不同, 算法的迥异带来的意义可能更明显
- 例如如, 对100万个数字进行排序:



# 前言

- 例子  $10^6$  个数进行排序
- 插入排序( $c_1 n^2$ ) vs. 合并排序( $c_2 n \log_2 n$ )
- A: 1.0G 插入排序  $2n^2$
- B: 10M 合并排序  $50n \log_2 n$
- 计算时间
- A:  $2 * (10^6)^2 / (1 * 10^9) = 2000(s)$
- B:  $50 * (10^6) * \log_2(10^6) / 10^7 \approx 100(s)$
- $10^7$  个数进行排序
- 2.3天 20分钟

# 前言

- 有没有计算机无法解决的问题
  - NP完全问题
- 是什么使某些问题很难计算,又使另一些问题容易计算?
  - 在给定的计算模型下研究问题的复杂性
  - 按照难度给问题分类
  - 抽象复杂性研究

# 前言

计算机的基本能力和限制是什么？

- 可计算理论:
- 计算模型
- 可计算问题/不可计算问题

# 前言

- 1930s 计算理论
- 图灵 停机问题
- 计算复杂性
  - 难问题 vs 易问题
  - NP-complete
  - 0/1背包问题
  - 旅行商（货郎担）问题
  - 分割问题
  - 陈列馆警卫问题

# 前言

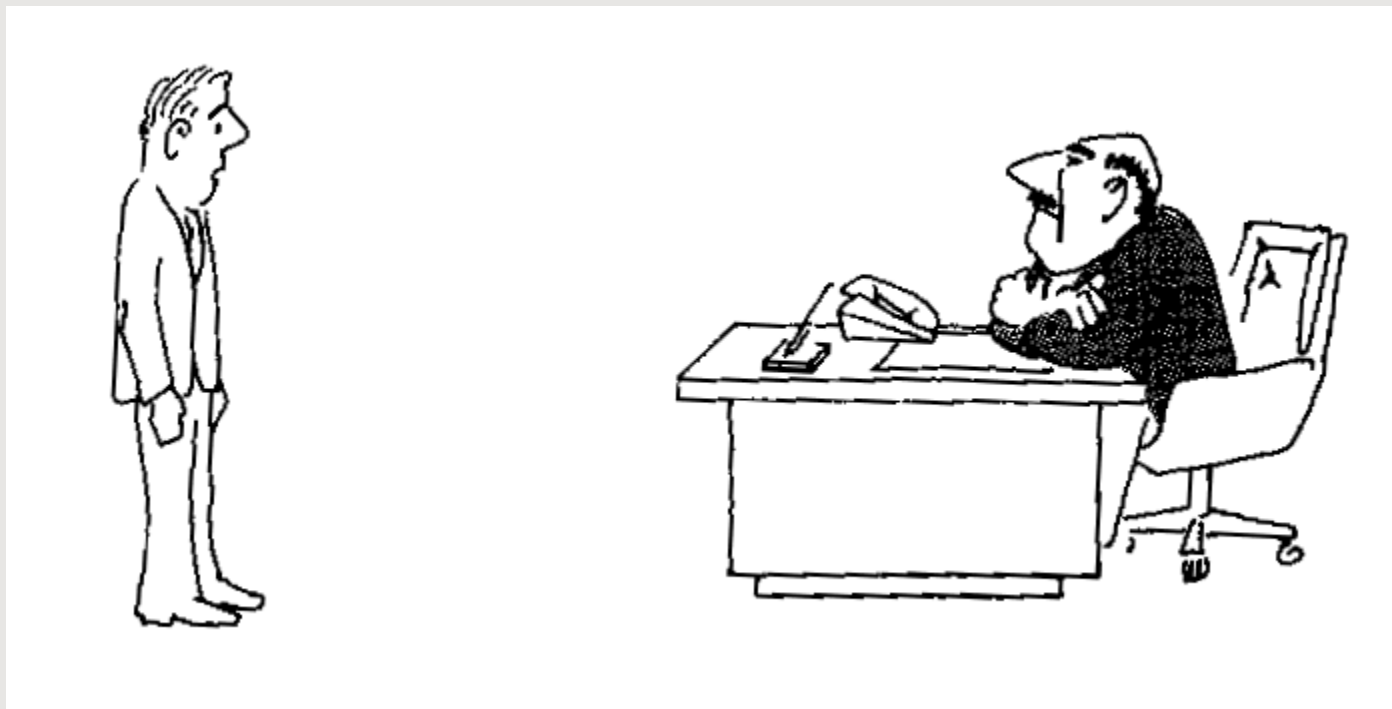
- 易解决的问题
  - 算法效率
    - 策略 VS 技巧
  - 规模 VS 时间

# 前言：



- 学习算法目的
  - 选择算法
  - 设计算法
- 案例
  - 小区预报 30分钟
  - 海洋气象预报 30倍
- 如何优化算法

# 前言：学习算法课的意义 Story-1



"I can't find an efficient algorithm, I guess I'm just too dumb."

Serious damage to your  
position within the company !!!



# 前言：学习算法课的意义 Story-2



"I can't find an efficient algorithm, because no such algorithm is possible!"

Unfortunately, proving intractability can be just as hard as finding efficient algorithms !!!

$\therefore$  No hope !!!

$P \neq NP$



# 前言：学习算法课的意义 Story-3



"I can't find an efficient algorithm, but neither can all these famous people"



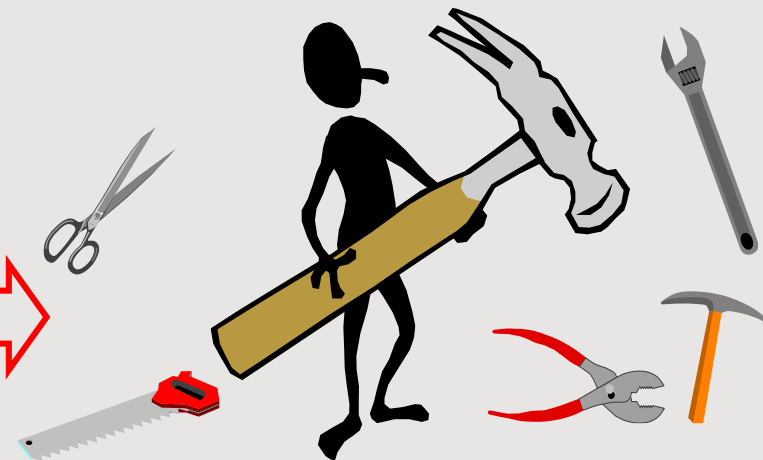
# 前言：学习算法课的意义 Story-4



"Anyway, we have to solve this problem.  
Can we satisfy with a good solution ? "

# 前言

- A survey of algorithmic design techniques.
- Abstract thinking.
- How to develop new algorithms for any problem that may arise.
- How to analysis a given algorithm and optimize it if possible
- Be a great thinker and designer.
- Not: A list of algorithms
  - - Learn their code
  - - Trace them until work
  - - Implement them
  - - be a mundane programmer

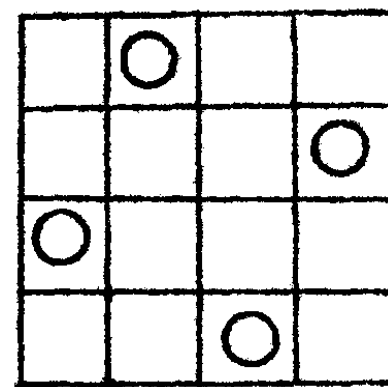


# 一些有趣的问题

- (1) 皇后问题:这是高斯1850年提出的一个著名问题:国际象棋中的“皇后”在横向、直向、和斜向都能走步和吃子,问在 $n \times n$  格的棋盘上如何能摆上 $n$ 个皇后而使她们都不能互相攻击。

当 $n$ 很大时, 问题很难。  
对于 $n=8$ , 现已知此问题共有92种解, 但只有12种是独立的, 其余的都可以由这12种利用对称或旋转而得到。

设 $n=4$ , 试一试。



(2) 背包问题1：有一旅行者要从 $n$ 种物品中选取不超过 $b$ 千克重的行李随身携带，要求总价值最大。

例：设背包的容量为50千克。物品1重10千克，价值60元；物品2重20千克，价值100元；物品3重30千克，价值120元。求总价值最大。

(3) 背包问题2：有一商人要从 $n$ 种货物中选取不超过 $b$ 千克重的行李随身携带，要求总价值最大。

例：设背包的容量为50千克。物品1有60千克，每千克价值60元；物品2有20千克，每千克价值100元；物品3有40千克，每千克价值120元。求总价值最大。

# 前言

**(4) 装箱问题：**设有体积分别为  $v_1, v_2, \dots, v_n$  的  $n$  种物品  $u_1, u_2, \dots, u_n$  装到容量为  $L$  的箱子里。不同的装箱方案所需的箱子数目可能不同，问如何装箱能装完这  $n$  种物品且使用的箱子数目最少。

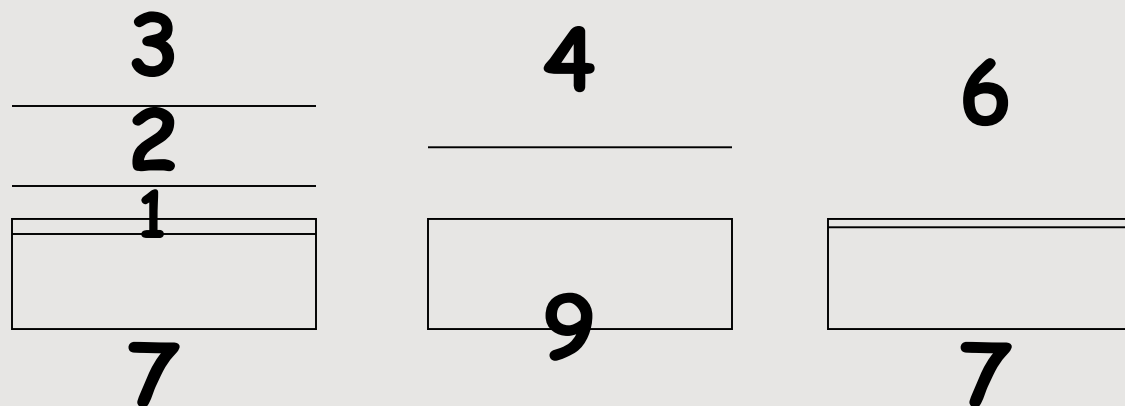


# 装箱问题

## 1) 首次适宜的算法firstfit (FF算法)

物品的顺序任意，每个物品依次放在它第一次放得进去的箱子中。

例： $L=(7,9,7,1,6,2,4,3)$  箱子尺寸为13





## 2)首次适宜降序算法**firstfitdesc** (FFD算法)

将**FF**算法中物品顺序按尺寸递减排序

## 3)最适宜算法**bsetfit**(BF算法)

物品的顺序任意，下一个物品要放入到能造成最小剩余空间的那个箱子中去。

## 4)最适宜降序算法**bsetfitdesc**(BFD算法)

将**BF**算法中物品顺序按尺寸递减排序





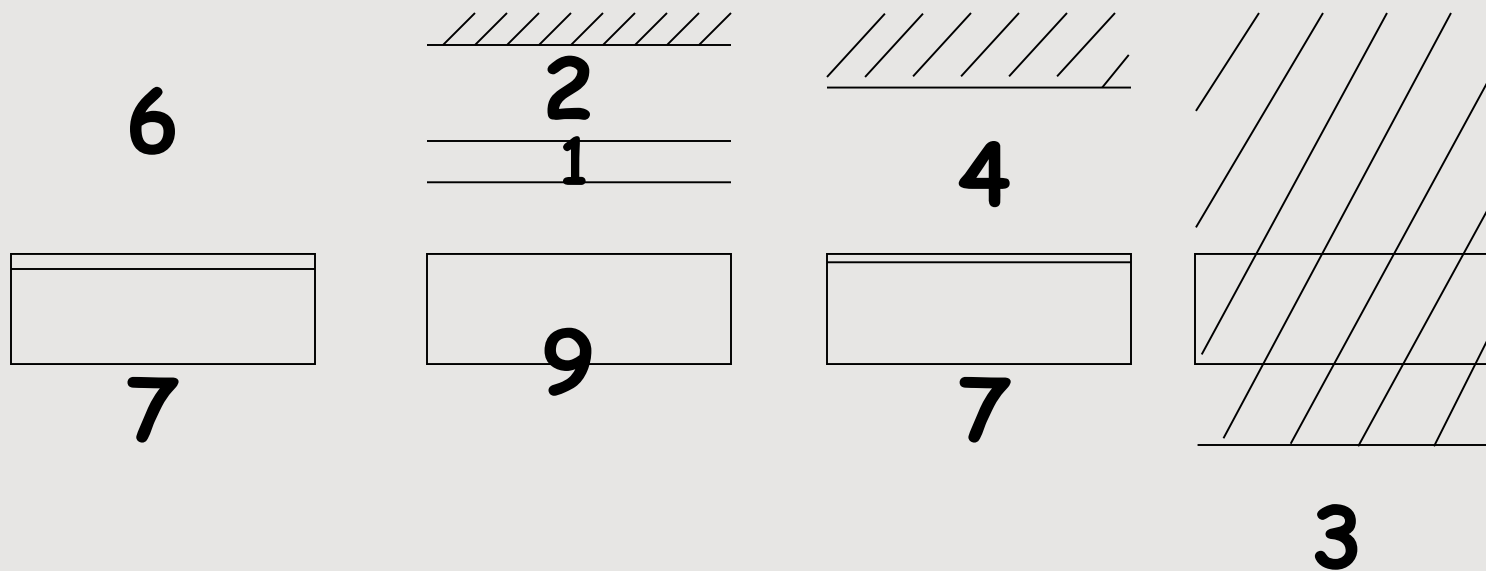
例：  $L=(7,9,7,1,6,2,4,3)$  箱子尺寸为13

练习：画出BF,FFD,BFD的图示，  
并写出近似最优解成立的条件



例:  $L=(7,9,7,1,6,2,4,3)$  箱子尺寸为13

BF:



# 过桥问题

**(5)** 有4个人打算过桥，他们都在桥的某一端。时间是晚上，他们只有一只手电筒。最多只能有两个人同时过桥，而且必须携带手电筒。必须步行将手电筒带来带去。每个人走路的速度不同：甲过桥需要1分钟，乙2分钟，丙5分钟，丁10分钟。两个人一起走的速度等于其中较慢的人的速度。要求过桥总时间最短。

**17分钟**



- 小明一家过一座桥，过桥时是黑夜，所以必须有灯。现在小明过桥要1秒，小明的弟弟要3秒，小明的爸爸要6秒，小明的妈妈要8秒，小明的爷爷要12秒。每次此桥最多可过两人，而过桥的速度依过桥最慢者而定，而且灯在点燃后30秒就会熄灭。问：小明一家如何过桥？

- 小明和弟弟过桥 3                      小明回            4
- 妈妈和爷爷过桥 16                    弟弟回            19
- 小明和爸爸过桥 25                    小明回            26
- 小明和弟弟过桥 29

- n个人呢？

- 在最初的4步里用模式一或模式二把最慢的两个旅行者移动到彼岸，于是问题被约化成N-2个旅行者的形式。问题在于应该选择哪一种模式。继续假设A、B为走得最快和次快的旅行者，过桥所需时间分别为a、b；而Z、Y为走得最慢和次慢的旅行者，过桥所需时间分别为z、y。

使用模式一移动Z和Y到彼岸所需的时间为：

$$z + a + y + a$$

使用模式二移动Z和Y到彼岸所需的时间为：

$$b + a + z + b$$

所以，当 $2b > a+y$ 时，应该使用模式一；

当 $2b < a+y$ 时，应该使用模式二；

当 $2b = a+y$ 时，使用模式一或二都可以。

- 搜索算法
- 最短路径

# 考试安排

## (6) 已知研究生选课情况，安排课程考试的日程

### 研究生选课情况表

姓名	选修课1	选修课2	选修课3
杨一	算法分析 (A)	形式语言 (B)	计算机网络 (E)
石磊	计算机图形学 (C)	模式识别 (D)	
魏涛	计算机图形学 (C)	计算机网络 (E)	人工智能 (F)
马耀先	模式识别 (D)	人工智能 (F)	算法分析 (A)
齐砚生	形式语言 (B)	人工智能 (F)	

时间最短?

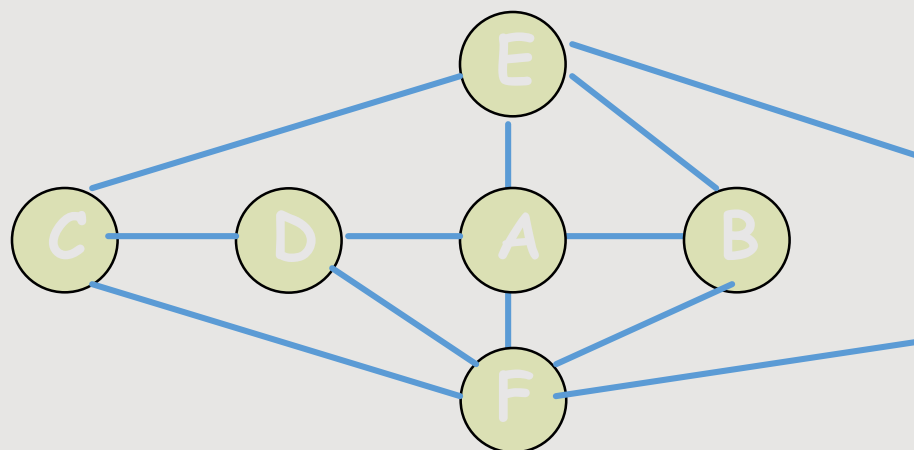
分析：

◆ 问题涉及的对象：课程；

◆ 课程之间的关系：同一个研究生选修的不能安排在同一时间内考试；

课程及课程之间的关系可用如下所示的图表示：

课程关系图



顶点：表示课程；

边：同一研究生选修的课程用边连接——有边连接的课程不能安排在同一时间考试；

## ◆ 课程考试安排问题转化为图的着色问题

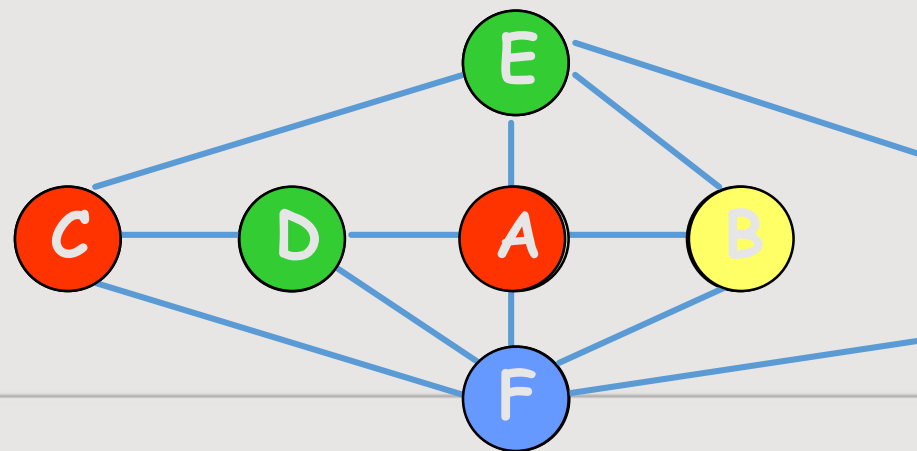
--用尽可能少的颜色给图的每个顶点着色，使相邻的顶点着上不同的颜色；

---每一种颜色代表一个考试时间，着上相同颜色的顶点是可以安排在同一时间考试的课程；

按顶点度数从大到小排列：**F A E C B D**

**F**: 蓝色； **A,C**: 红色； **E,D**: 绿色； **B**: 黄色；

即 **A,C** 可安排在同一时间考试，**E,D**可安排在同一时间考试；

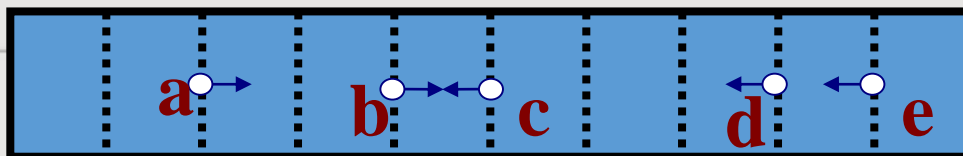




# 蚂蚁问题

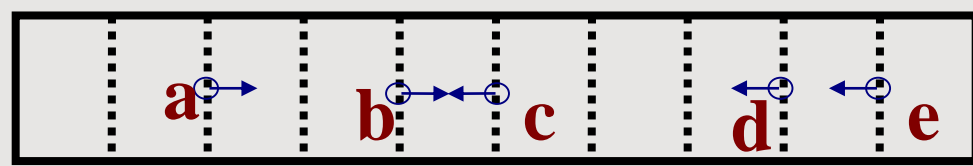
(7) 有一根**10厘米**的细木杆，在第**2厘米**、第**4厘米**、第**5厘米**、第**8厘米**、第**9厘米**这五个位置上各有一只蚂蚁。木杆很细，只能同时通过一只蚂蚁。开始时，蚂蚁的头朝左还是朝右是任意的，它们只能朝前走或调头，但不会后退。所有蚂蚁的速度都相同，均为**1cm/s**。当任意两只蚂蚁碰头时，两只蚂蚁会同时掉头并且仍然按**1cm/s**朝相反方向走。求蚂蚁掉下木杆的最小和最大时间。

1 2 3 4 5 6 7 8 9

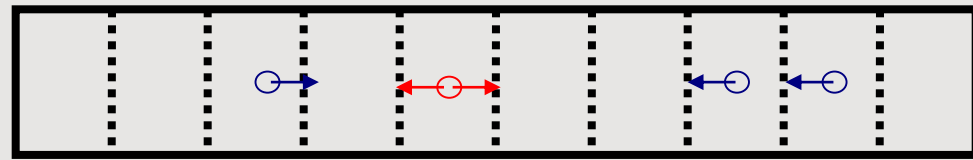




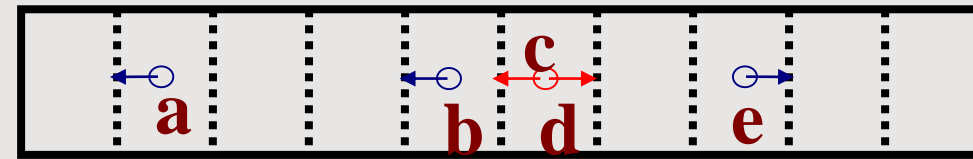
1 2 3 4 5 6 7 8 9



0.5s



3.5s



5秒，a掉；6秒，e掉；8秒，b和d掉；9秒，c掉。



- 方法1:穷举法
- 方法2:求蚂蚁掉下木杆的最小和最大时间?



- 在太平洋的一个小岛上生活着土人，他们不愿意被外人打扰，一天，一个探险家到了岛上，被土人抓住，土人的祭司告诉他，临死前还可以有一个机会留下一句话，如果这句话是真的，你将被烧死，是假的，你将被五马分尸，可怜的探险家如何才能活下来？
- 探险家说他将被五马分尸

- 在一个平面上画1999条直线最多能将这一平面划分成多少个部分？
- 增加一条直线,若该直线与其他直线有 $n$ 个交点, 就再能把平面多分出 $n+1$ 个部分, 因此若想把平面划分的部分最多, 新添入的直线必须与前 $k$ 条直线交 $k$ 个点,
- 第二条直线要与第一条直线交1个点, 多划分出 $1+1=2$ 个部分
- 第三条要与前两条交2个点, 多划分出 $1+2=3$ 个部分
- .....
- 第1999条与前1998条交1998个 点, 多划分出 $1+1998=1999$ 个部分。



- 第一条直线把平面划分出2个部分，因此1999条直线能划分平面的块数为：

$$\begin{aligned} & 2+2+3+4+5+\dots+1998+1999 \\ &= 1+(1+2+3+4+5+\dots+1998+1999) \\ &= 1+(1+1999)*1999/2 \\ &= 1999001 \end{aligned}$$

# 前言

## 问题1:

**旅行商问题** Traveling Salesman Problem (TSP):

设有 $n$ 个城市，已知任意两城市之间距离，现有一推销员想从某一城市出发巡回经过每一城市（且每城市只经过一次），最后又回到出发点，问如何找一条最短路径。

穷举法的时间复杂性为 $O(n!)$

$n = 20$ 时， $20! = 2.43 \times 10^{18}$ ，设每条路径需CPU时间为 $10^{-7}s$ ，则总共要7千多年才能完成。

## 问题2：排序

# 前言

- 问题3
  - 围棋
- 问题4
  - 象棋
- 停机问题
- 定理证明



# 前言

- 学习目标
  - 一般问题的经典算法
  - 一些基本的算法设计技术、工具
  - 算法（问题）分析的原理、技术
  - 正确性证明

# 前言

- 解决一个计算问题的过程
  - ① 可计算否
  - ② 能行可计算否
  - ③ 算法设计与分析
  - ④ 用计算机语言实现算法
  - ⑤ 软件系统

# 第1章 算法概述

## 学习要点:

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。
- 掌握用C++语言描述算法的方法。



## 一、算法设计与分析的研究对象

非数值问题

## 二、什么是算法

例1 已知两正整数 $m$ 和 $n$ ，求二者的最大公因子

算法1.1 欧几里德(Euclid)算法

输入 正整数 $m, n$

输出  $m$ 和 $n$ 的最大公因子

- 定理

对任意正整数 $m$ 和任意非负整数 $n$ , 并且 $m > n \geq 0$

有  $\gcd(m,n)=\gcd(n,m \bmod n)$

如  $\gcd(24,18)$

$= \gcd(18,6)$

$= \gcd(6,0) = 6$

### 算法1.1 欧几里德算法

S1 求余数: $m$ 除以 $n$ ,令 $r$ 是所得的余数, 转S2

S2 判断余数:若 $r=0$ ,则输出 $n$ 的当前值, 算法结束, 否则转S3

S3 代替: $m \leftarrow n, n \leftarrow r$ ,转S1

## • 算法1.1 欧几里德算法

• 输入：正整数 $m, n$

• 输出： $m, n$ 的最大公因子

• 1. `int euclid(int m, int n)`

• 2. {

• 3.     `int       r;`

• 4.     `do {`

• 5.         `r = m % n;`

• 6.         `m = n;`

• 7.         `n = r;`

• 8.     `} while(r)`

• 9.     `return m;`

• 10. }





1:

算法是一个有穷规则的集合

规则：解某问题所用到的各种运算的序列

注意：算法 $\neq$ 程序 算法设计 $\neq$ 程序设计

2、评价算法的标准

- 1) 算法执行的时间短（时间复杂性）
- 2) 算法需要的存储空间小（空间复杂性）
- 3) 正确性    4) 可读性
- 5) 最优性    6) 精确性

## 程序(Program)

- 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。





- 10进制计数 AD600
- 9世纪 Al Khwarizmi  
加, 减, 乘, 除, 平方根,  $\Pi$
- Fibonacci
- Fibonacci数列
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...



$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 \end{cases}$$



- 算法1:
  - fib1(int n)
  - {
    - If( $n==0$ )return 0;
    - If( $n==1$ )return 1;
    - Return fib( $n-1$ )+fib( $n-2$ );
  - }

## 问题

- 1: 正确性 ?
- 2: 时间花费 ?
- 3: 能不能改进 ?

- $T(n)$ : 计算  $F(n)$  所需要的  
计算次数

$$T(n) \leq 2 \quad n \leq 1$$

$$T(n) = T(n-1) + T(n-2) + 3 \quad n > 1$$

$$T(200) \geq F(200) \geq 2^{138}$$

$$F(n) = 2^{0.694n} \approx 1.6^n$$

地球模拟器：太阳变成红巨星以后

IBM Roadrunner (2008年7月) 快30倍，可以多  
算7个数

■ 摩尔定律：每年计算速度增加1.6倍

□ 如果  $F(n)$  可以在一年内计算出来，下一年可以计算  
 $F(n+1)$

为什么？

太多重复计算

$$F(n) = \frac{1}{\sqrt{5}} \left( \left( \frac{1 + \sqrt{5}}{2} \right)^{n+1} - \left( \frac{1 - \sqrt{5}}{2} \right)^{n+1} \right)$$



- 算法2:
  - `int fib2(int n)`
  - `{`
  - `int val,*array= (int*)malloc ((n+1)*sizeof(int));`
  - `array[0]=0;`
  - `array[1]=1;`
  - `for(i=2;i<=n;i++)array[i]=array[i-1]+array[i-2];`
  - `val= array[n];`
  - `free(array);`
  - `return val;`
  - `}`



- 所花时间 $c_1n$ ,代价空间 $c_2n$
- 空间换时间或者时间换空间 非常常见的策略
- 可以轻易地计算 $F_{200}, F_{200,000}$
- 有没有更好的算法？

$$F_1 = F_0 \quad F_2 = F_1 + F_0 \quad \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

- 算法2: 所花时间  $c_1 n$ ,
- 代价空间  $c_2 n$

2个  $2 \times 2$  矩阵相乘: 4次加法; 8次乘法

$x^n$  可以通过  $\log n$  次矩阵相乘得到

$O(\log n)$ ?

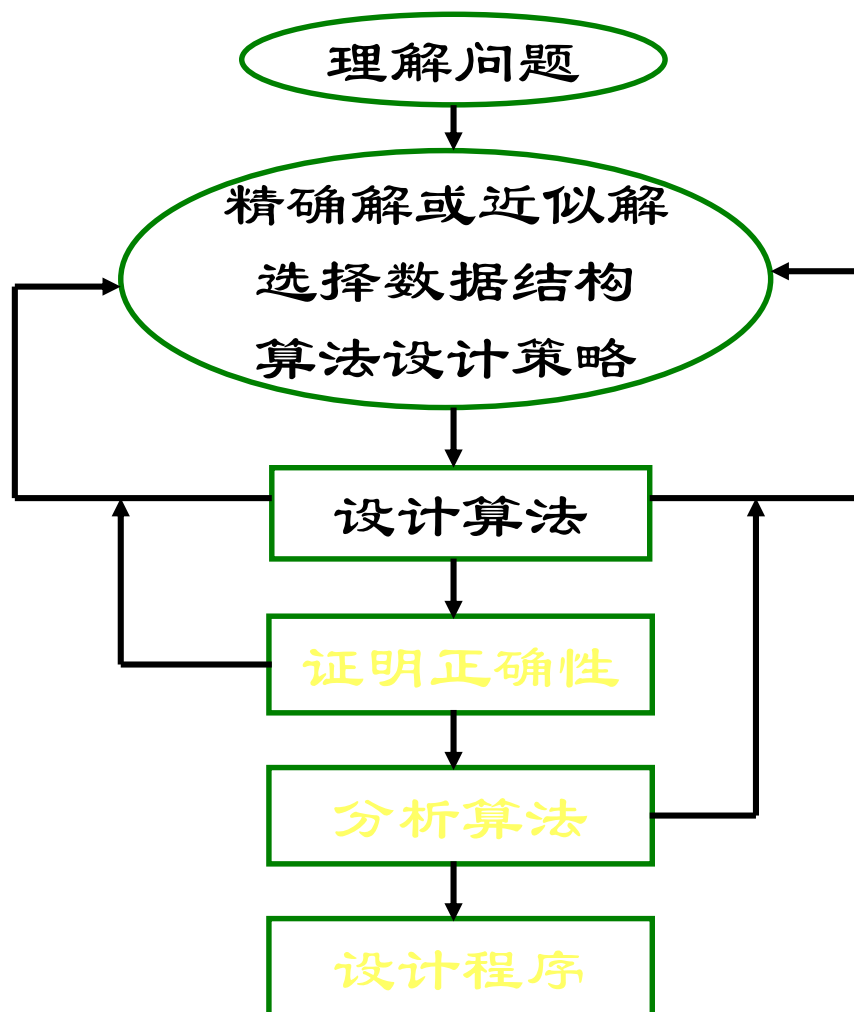
6.If your attack is going really well, it's an ambush.



- 简化, 所有的操作均花费一个固定的时间单位时间
- $F_{200} + F_{201}$  也可以在一个固定时间 ?
- $F(n) \geq 2^{0.694n}$
- $n$ 位数加法乘法所花时间?
- 加法: 线性时间; 乘法: 平方
- 算法1:  $c_1 F_n$  ??, 算法2:  $c_2 n^2$ , 空间  $c_3 n^2$
- 乘法3时间复杂度?  $c_4 (\log n) n^2$
- 乘法时间复杂度  $M(n)$  可以通过  $O(M(n) \log n)$



# 问题求解(Problem Solving)



## 算法复杂性分析

- 当问题规模变大以后, 所需要的资源?
- 衡量算法优劣的标准: 时空复杂性
- 算法复杂性 = 算法所需要的计算机资源
- 算法的时间复杂性  $T(n)$
- 算法的空间复杂性  $S(n)$
- 其中  $n$  是问题的规模 (输入大小)

- 1、问题规模：与问题相关的整数量，它可以衡量问题的规模并表示输入数据量的尺度。也称为问题尺度
- 2、算法的时间复杂性：处理一个尺度为 $n$ 的输入，算法所需要的时间，记为 $T(n)$
- 3、算法的时间复杂性：处理一个尺度为 $n$ 的输入，算法所需要的执行次数（或执行的步数），记为 $T(n)$
- 4、算法的空间复杂性：  
处理一个尺度为 $n$ 的输入，算法所需要的空间数，记为 $S(n)$

## 算法的时间复杂性

- (1) **最坏情况**下的时间复杂性
- $T_{\max}(n) = \max\{ T(I) \mid \text{size}(I)=n \}$
- (2) **最好情况**下的时间复杂性
- $T_{\min}(n) = \min\{ T(I) \mid \text{size}(I)=n \}$
- (3) **平均情况**下的时间复杂性
- $T_{\text{avg}}(n) = \sum_{\text{size}(I)=n} p(I)T(I)$
- 其中 $I$ 是问题的规模为 $n$ 的实例， $p(I)$ 是实例 $I$ 出现的概率。

## 算法渐近复杂性

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

## 渐近分析的记号

- 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 $O$
- $O(g(n)) = \{ f(n) \mid \text{存在正常数} c \text{和} n_0 \text{使得对所有} n \geq n_0 \text{有: } 0 \leq f(n) \leq cg(n) \}$
- (2) 渐近下界记号 $\Omega$
- $\Omega(g(n)) = \{ f(n) \mid \text{存在正常数} c \text{和} n_0 \text{使得对所有} n \geq n_0 \text{有: } 0 \leq cg(n) \leq f(n) \}$

- (3) 非紧上界记号  $o$

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

- $0 \leq f(n) < cg(n) \}$

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

- (4) 非紧下界记号  $\omega$

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

- $0 \leq cg(n) < f(n) \}$

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

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



- (5) 紧渐近界记号 $\Theta$
- $\Theta(g(n)) = \{ f(n) \mid \text{存在正常数 } c_1, c_2 \text{ 和 } n_0 \text{ 使得对所有 } n \geq n_0 \text{ 有: } c_1 g(n) \leq f(n) \leq c_2 g(n) \}$
- **定理1:**  $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$



## 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  $O, o, \Omega$  和  $\omega$  的意义是类似的。

## 渐近分析中函数比较

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

## 渐近分析记号的若干性质

### • (1) 传递性:

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



- (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:

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

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



- (5) 算术运算:

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



- 规则  $O(f(n)) + O(g(n)) = O(\max\{f(n), g(n)\})$
- **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  $c_3 = \max\{c_1, c_2\}$ ,  $n_3 = \max\{n_1, n_2\}$ ,  $h(n) = \max\{f(n), g(n)\}$ 。



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

- 证明:

- 则对所有的  $n \geq n_3$ , 有

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

## ● 思考题

- $f(n) + g(n) = \Theta(\min(f(n), g(n)))$ .....?
  - F  $f=n^4$   $g=n$
- $f(n)+g(n) = \Theta(\max\{f(n), g(n)\})$ .....?
  - T





- 等式  $0.75n^2 = O(n)$  何时成立?
- 永不成立
- 等式  $3n^{1/2} = O(n)$  在  $n=9$  时成立吗?
- $n=9$  时, 虽然两边值相等, 但等式不成立

## 算法渐近复杂性分析中常用函数

- **(1) 单调函数**

- 单调递增:  $m \leq n \Rightarrow f(m) \leq f(n)$  ;
- 单调递减:  $m \leq n \Rightarrow f(m) \geq f(n)$ ;
- 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

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

## • (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;

- $a > 1 \Rightarrow \quad \quad \quad \Rightarrow n^b = o(a^n)$

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \Lambda = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

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

$$\lim_{n \rightarrow \infty} \left( 1 + \frac{x}{n} \right)^n = e^x$$

## • (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  $a > 0, b > 0, c > 0$

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



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

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

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

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

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

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



# 符号

•

$$\log n = \log_2 n, \quad \lg n = \log_{10} n \quad \lg 2 = 0.301$$

$$\log^k n = (\log n)^k$$

$$\log \log n = \log(\log n)$$

性质：

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

$$\log_k n = c \log_l n$$

# 符号

• 证明:

$$\begin{aligned}a^{\log_b n} &= b^{\log_b a^{\log_b n}} \\&= b^{\log_b n \cdot \log_b a} \\&= (b^{\log_b n})^{\log_b a} \\&= n^{\log_b a}\end{aligned}$$

- $|x| \leq 1 \Rightarrow \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \Lambda$ .

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

- for any  $a > 0$ ,  $\lim_{n \rightarrow \infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n \rightarrow \infty} \frac{\log^b n}{n^a} = 0$   
 $= o(n^a)$


- (6) 阶乘函数

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

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

- Stirling's approximation

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


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

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

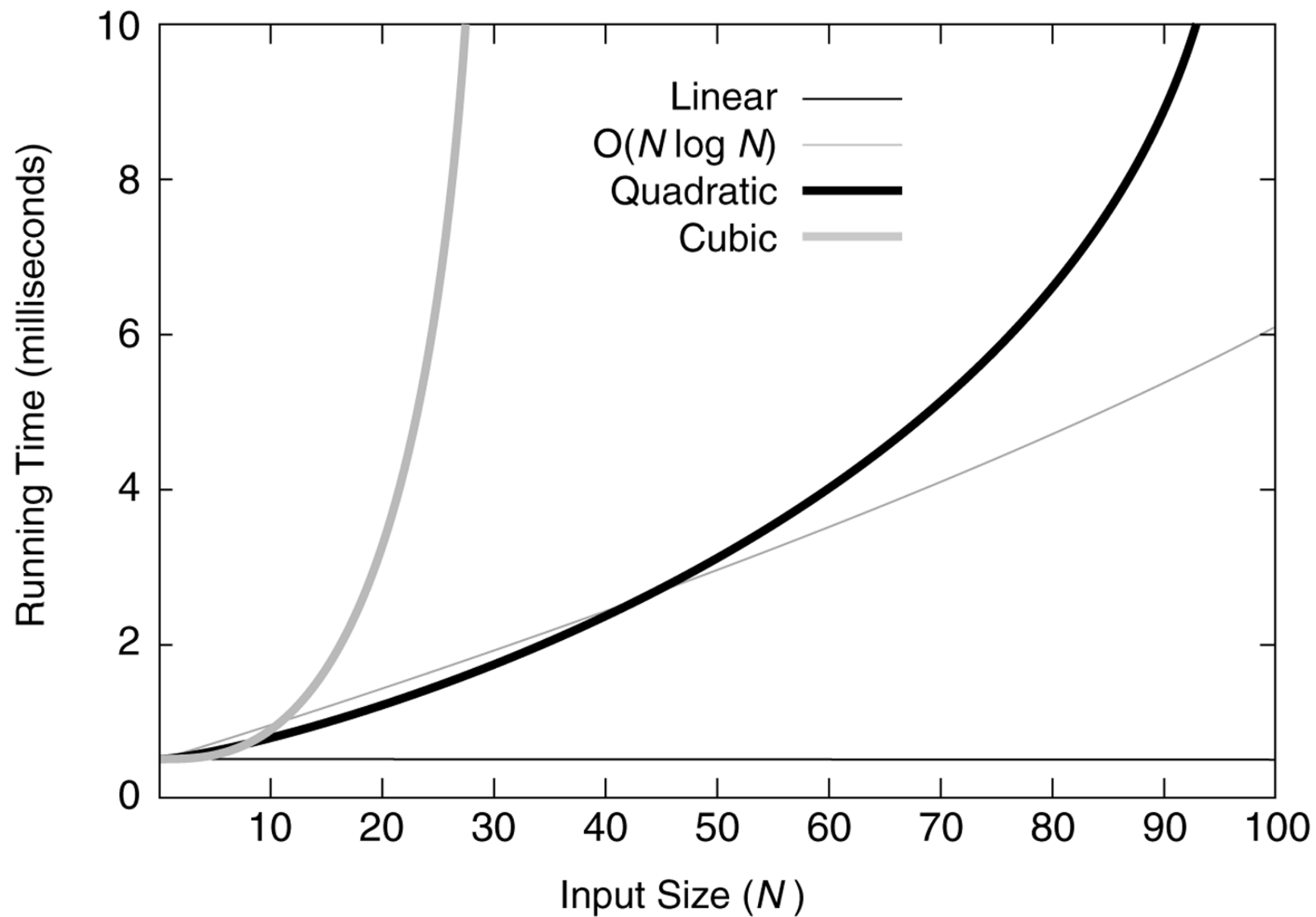
$$n! = \omega(2^n)$$

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

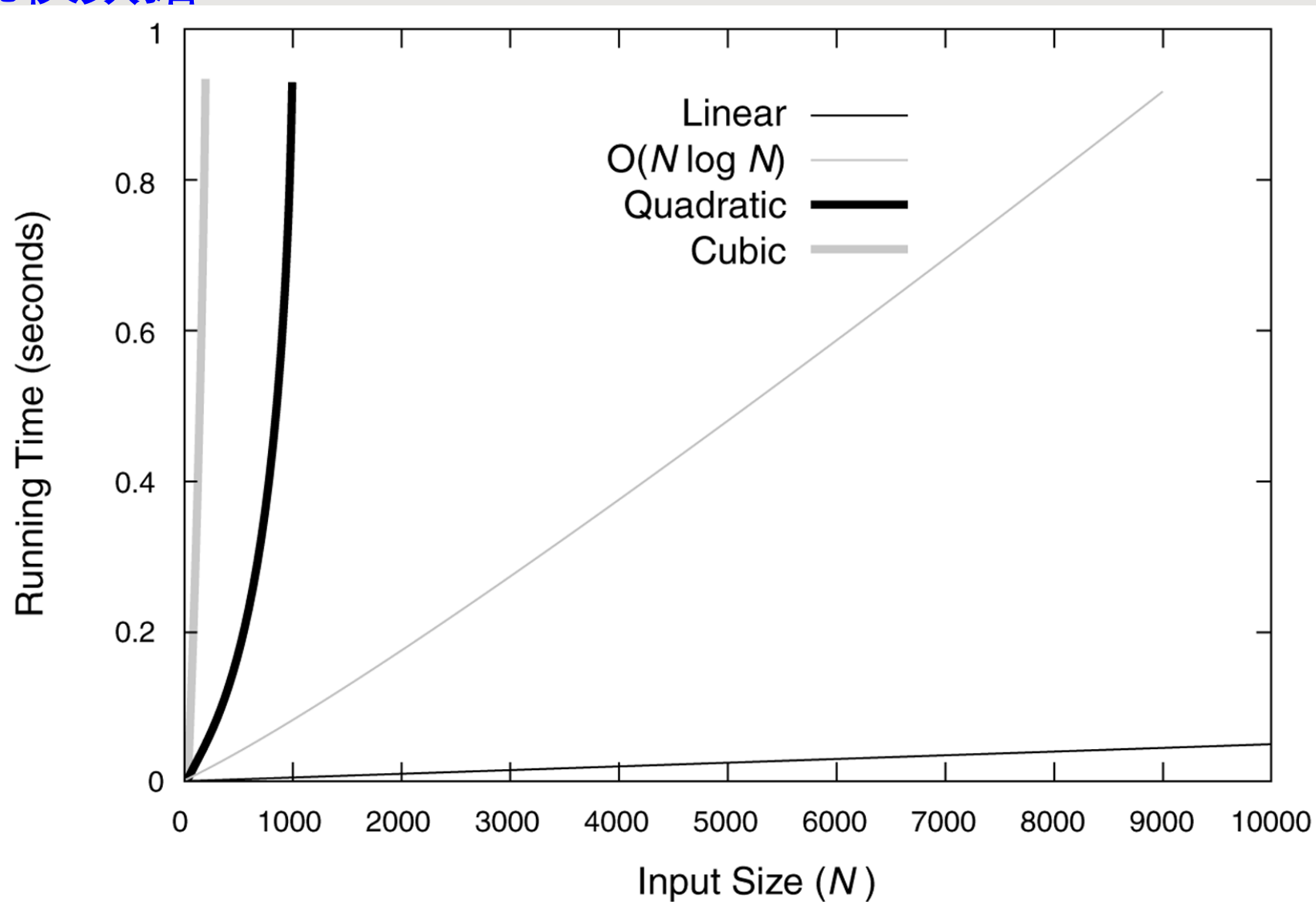
# 算法分析中常见的时间复杂度

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



## 中等规模数据





# 0、o、Ω、ω、Θ 第一种理解方法

设 $f$ 和 $g$ 是定义域为自然数 $N$ 上的函数 $f(n)=O(g(n))$ .

若存在正数 $c$ 和 $n_0$ 使得对一切 $n \geq n_0$ 有  $0 \leq f(n) \leq cg(n)$

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

若存在正数 $c$ 和 $n_0$ 使得对一切 $n \geq n_0$ 有  $0 \leq cg(n) \leq f(n)$

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

若对任意正数 $c$ 存在 $n_0$ 使得对一切 $n \geq n_0$ 有  $0 \leq f(n) < cg(n)$

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

若对任意正数 $c$ 存在 $n_0$ 使得对一切 $n \geq n_0$ 有  $0 \leq cg(n) < f(n)$

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

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



# O、o、Ω、ω、Θ 第二种理解方法

- 求复杂性函数阶的极限方法

$$\text{若 } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = s$$

当  $s \neq 0$  时, 说明  $f(n)$  和  $g(n)$  同阶, 则  $f = \Theta(g)$ ;

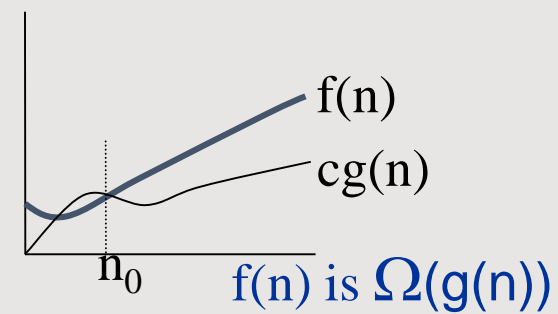
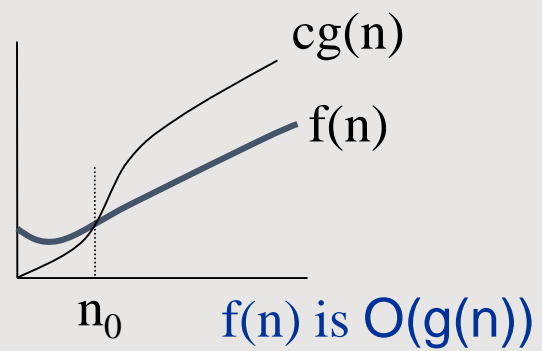
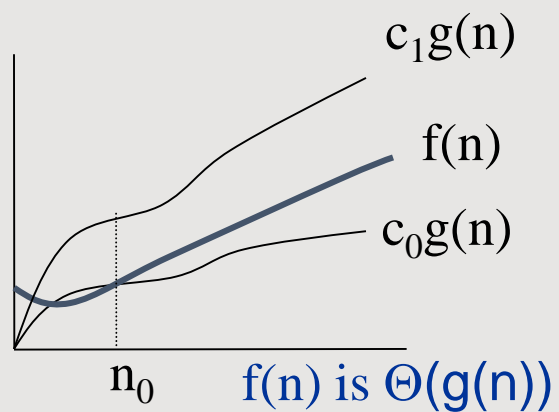
当  $s = 0$  时, 说明  $f(n)$  比  $g(n)$  的阶低, 则  $f = o(g)$ ;

当  $s = \infty$  时, 说明  $f(n)$  比  $g(n)$  的阶高, 则  $f = \omega(g)$ ;

例如,

$$f(n)=n^2/4, g(n)=n^2, \text{ 则 } n^2/4 = \Theta(n^2)$$

$$f(n)=\log n, g(n)=n, \text{ 则 } \log n = O(n)$$



- 标准复杂性函数的比较

$$\underbrace{O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)}_{\text{多项式时间阶}} < \underbrace{O(2^n) < O(n!) < O(n^n)}_{\text{指数时间阶}}$$

多项式时间阶

指数时间阶

注意：1) 不能划等号 2) 以下若无特殊声明，log  
是以2为底的对数

3) 上式只有在 $n$ 较大的时候成立

$O(1)$ 的含义？

计算时间由一个常数（零次多项式）来限界



## 指数时间

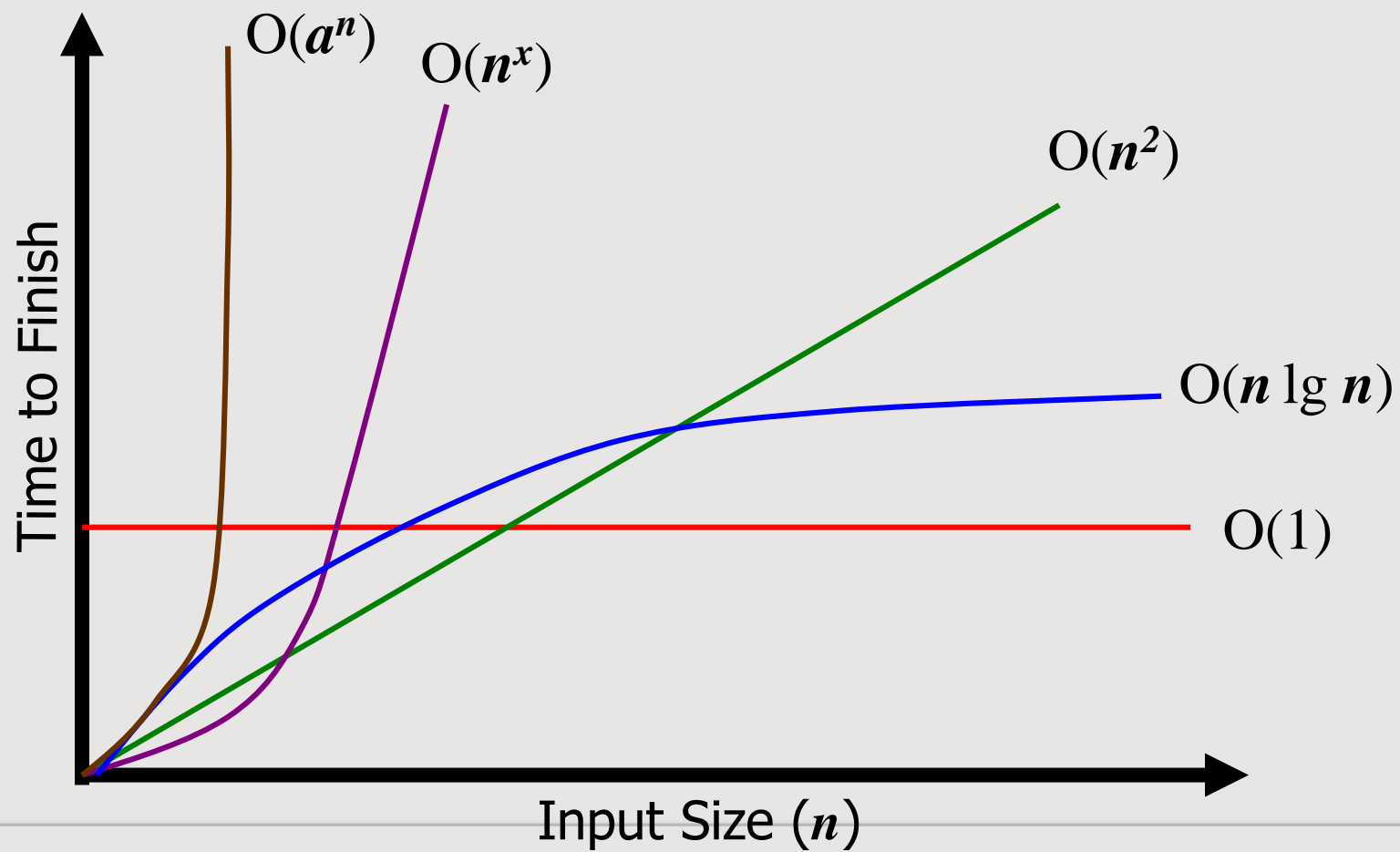
一个算法的时间复杂性如果是 $\Omega(2^n)$ ，则称此算法需要指数时间。

## 多项式时间

一个算法的时间复杂性如果是 $O(n^k)$ ( $k$ 为有理数)，则称此算法需要多项式时间。

## 有效算法

以多项式时间为限界的算法称为有效算法。



时间复杂性函数	问题规模					
	10	20	30	40	50	60
n	$10^{-5}$	$2 \times 10^{-5}$	$3 \times 10^{-5}$	$4 \times 10^{-5}$	$5 \times 10^{-5}$	$6 \times 10^{-5}$
$n^2$	$10^{-4}$	$4 \times 10^{-4}$	$9 \times 10^{-4}$	$16 \times 10^{-4}$	$25 \times 10^{-4}$	$36 \times 10^{-4}$
$n^3$	$10^{-3}$	$8 \times 10^{-3}$	$27 \times 10^{-3}$	$64 \times 10^{-3}$	$125 \times 10^{-3}$	$216 \times 10^{-3}$
$n^5$	$10^{-1}$	3.2	24.3	1.7分	5.2分	13.0分
$2^n$	.001秒	1.0秒	17.9分	12.7天	35.7年	366世纪
$3^n$	.059秒	58分	6.5年	3855世纪	$2 \times 10^8$ 世纪	$1.3 \times 10^{13}$ 世纪

时间复杂性 函数	1小时可解的问题实例的最大规模		
	计算机	快100倍的计算机	快1000倍的计算机
$n$	$N_1$	$100 N_1$	$1000 N_1$
$n^2$	$N_2$	$10 N_2$	$31.6 N_2$
$n^3$	$N_3$	$4.64 N_3$	$10 N_3$
$n^5$	$N_4$	$2.5 N_4$	$3.98 N_4$
$2^n$	$N_5$	$N_5 + 6.64$	$N_5 + 9.97$
$3^n$	$N_6$	$N_6 + 4.19$	$N_6 + 6.29$



# 例

- 例：算法 $A_1, A_2$ 的时间复杂性分别是 $n, 2^n$ , 设 $100\mu s$ 是一个单位时间, 求 $A_1, A_2$ 在 $1s$ 内能处理的问题规模。
- 已知 $\lg 2 = 0.301$

$$T(n) = n$$

$$T(n) * 10^{-4} = 1$$

$$\text{即 } n * 10^{-4} = 1$$

$$\text{所以 } n = 10^4$$

$$\text{即 } 2^n * 10^{-4} = 1$$

$$2^n = 10^4$$

$$\text{所以 } n = 13.29$$

# 复杂性的基本概念

- 有效算法：

以多项式时间为限界的算法。

- 按算法的时间复杂性，可以把问题分为2类：

## 1) p类问题

能以多项式时间为界的算法解决的问题

## 2) NP类问题：已知的最好算法是以非多项式时间为界的问题。

实际上可计算的问题：

多项式时间可解的问题

## 用c++描述算法

CATEGORY	EXAMPLES	ASSOCIATIVITY
Operations on References	. []	Left to right
Unary	++ -- ! - (type)	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift (bitwise)	<< >>	Left to right
Relational	< <= > >= instanceof	Left to right
Equality	== !=	Left to right
Boolean (or bitwise) AND	&	Left to right
Boolean (or bitwise) XOR	^	Left to right
Boolean (or bitwise) OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= *= /= %= += -=	Right to left



- (1) 选择语句:
- (1.1) if 语句:

```
if (expression) statement;  
else statement;
```

- (1.2) ? 语句:

```
exp1?exp2:exp3  
y= x>9 ? 100:200; 等价于:  
if (x>9) y=100;  
else y=200;
```

### (1.3) switch语句:

```
switch (expression) {  
    case 1:  
        statement sequence;  
        break;  
    case 2:  
        statement sequence;  
        break;  
    ...  
    default:  
        statement sequence;  
}
```



## (2) 迭代语句:

- **(2.1) for 循环:**
  - for (init;condition;inc) statement;
- **(2.2) while 循环:**
  - while (condition) statement;
- **(2.3) do-while 循环:**
  - do{
  - statement;
  - } while (condition);

### (3) 跳转语句:

- (3.1) return语句:
- return expression;
- (3.2) goto语句:
- goto label;
- ...
- label:

#### (4) 函数:

```
return-type function name(para-list)
{
    body of the function
}
```

- 例:

```
int max(int x,int y)
{
    return x>y?x:y;
}
```



## (5) 模板template :

```
template <class Type>  
Type max(Type x,Type y)  
{  
    return x>y?x:y;  
}
```

```
int i=max(1,2);  
double x=max(1.0,2.0);
```

## (6) 动态存储分配:

### • (6.1) 运算符new :

- 运算符new用于动态存储分配。
- new返回一个指向所分配空间的指针。
- 例: `int *x; y=new int; *y=10;`
- 也可将上述各语句作适当合并如下:
- `int *y=new int; *y=10;`
- 或 `int *y=new int(10);`
- 或 `int *y; y=new int(10);`

## (6.2) 一维数组：

- 为了在运行时创建一个大小可动态变化的一维浮点数组x，可先将x声明为一个float类型的指针。然后用new为数组动态地分配存储空间。
- 例：
  - `float *x=new float[n];`
  - 创建一个大小为n的一维浮点数组。运算符new分配n个浮点数所需的空间，并返回指向第一个浮点数的指针。
  - 然后可用x[0]， x[1]， ...， x[n-1]来访问每个数组元素。

### (6.3) 运算符delete :

- 当动态分配的存储空间已不再需要时应及时释放所占用的空间。
- 用运算符delete来释放由new分配的空间。
- **例:**
- delete y;
- delete [ ]x;
- 分别释放分配给\*y的空间和分配给一维数组x的空间。

#### (6.4) 动态二维数组：

- 创建类型为Type的动态工作数组，这个数组有

```
template <class Type>
void Make2DArray(Type** &x,int rows, int cols)
{
    x=new Type*[rows];
    for (int i=0;i<rows;i++)
        x[i]=new Type[cols];
}
```

- 当不再需要一个动态分配的二维数组时，可按以下步骤释放它所占用的空间。首先释放在for循环中为每一行所分配的空间。然后释放为行指针分配的空间。

```
template <class Type>
void Delete2DArray(Type** &x,int rows){
    for (int i=0;i<rows;i++)
        delete []x[i];
    delete []x;
    x=0;
}
```

- 释放空间后将x置为0，以防继续访问已被释放的空间。

• 例：顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
        if (a[i]==k) return i;
    return -1;
}
```

- (1)  $T_{\max}(n) = \max\{ T(I) \mid \text{size}(I)=n \} = O(n)$
- (2)  $T_{\min}(n) = \min\{ T(I) \mid \text{size}(I)=n \} = O(1)$
- (3) 在平均情况下, 假设:
  - (a) 搜索成功的概率为  $p$  ( $0 \leq p \leq 1$ );
  - (b) 在数组的每个位置  $i$  ( $0 \leq i < n$ ) 搜索成功的概率相同, 均为  $p/n$ 。

$$\begin{aligned}
 T_{\text{avg}}(n) &= \sum_{\text{size}(I)=n} p(I)T(I) \\
 &= \left( 1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \Lambda + n \cdot \frac{p}{n} \right) + n \cdot (1 - p) \\
 &= \frac{p}{n} \sum_{i=1}^n i + n(1 - p) = \frac{p(n+1)}{2} + n(1 - p)
 \end{aligned}$$



## 算法分析的基本法则

### • 非递归算法：

- (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。



```

template<class Type>
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 of (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  $1 \leq i < n$ ;

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

$$+ c_4(n-1) + c_7(n-1)$$

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

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

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

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

$$\begin{aligned} T_{\max}(n) &\leq c_1 n + c_2(n-1) + c_3(n-1) + \\ &c_4 \left( \frac{n(n+1)}{2} - 1 \right) + c_5 \left( \frac{n(n-1)}{2} \right) + c_6 \left( \frac{n(n-1)}{2} \right) + c_7(n-1) \\ &= \frac{c_4 + c_5 + c_6}{2} n^2 + \left( c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7) \\ &= O(n^2) \end{aligned}$$

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

$$\begin{aligned} T_{\max}(n) &\geq \frac{c_4 + c_5 + c_6}{2} n^2 \\ &\quad + \left( c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7 \right) n \\ &\quad - (c_2 + c_3 + c_4 + c_7) \\ &= \Omega(n^2) \end{aligned}$$

- 由此可见,  $T_{\max}(n) = \Theta(n^2)$

## 最优算法

- 问题的计算时间下界为 $\Omega(f(n))$ ，则计算时间复杂度为 $O(f(n))$ 的算法是最优算法。
- 例如，排序问题的计算时间下界为 $\Omega(n \log n)$ ，计算时间复杂度为 $O(n \log n)$ 的排序算法是最优算法。
- 堆排序算法是最优算法。

## 课后作业

- 习题 1-1, 1-4, 1-9。