第一章　算法概述

第1章 算法概述

- 本章主要知识点：
  - 1.1 算法与程序
  - 1.2 算法与数据结构
  - 1.3 算法描述
  - 1.4 算法设计
  - 1.5 算法分析的基本原则
  - 1.6 算法复杂性分析

1.1 算法与程序

- 算法（algorithms）的研究是计算机科学的核心课题之一。早在计算机问世以前，就有人开创了算法的研究，并创建了许多有效的算法。
  - ·欧几里德算法（求两数的最大公因子）
  - ·孙子算法（求若干个数的最小公倍数）
  - ·快速富里叶变换（FFT）算法（60年代后半期）
  - ·并行算法（70年代）
- 据不完全统计，50年代以前约占文献总量10%；产生60年代的约占文献总量的30%；产生于70年代（76年前）约占文献总量60%。

1.1 算法与程序

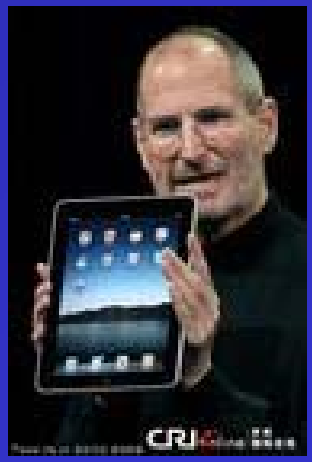
- 一年一度的计算机科学领域最高荣誉——计算图灵奖（Turing Awards），被誉为计算机科学领域的“诺贝尔奖”，从1966年至1999年，约有12人是由于在算法与数据结构、计算复杂性理论、程序设计等获图灵奖，比如1984年图灵设计得主瑞士的N·Wirth教授提出“算法+数据结构=程序”著名公式。1980年授予在程序设计和算法方面，因发明Quick Sort算法的英国Hoare教授。

1.1 算法与程序

- 阿兰·麦席森·图灵（Alan Mathison Turing，1912.6.23—1954.6.7），是英国著名的数学家和逻辑学家，被称为计算机科学之父、人工智能之父，是计算机逻辑的奠基者，提出了“图灵机”和“图灵测试”等重要概念。人们为纪念其在计算机领域的卓越贡献而设立“图灵奖”。

1.1 算法与程序

- 图灵享年42岁，科学家为了纪念他，1966年美国计算机协会设立了“图灵奖”成为计算机科学家的最高奖项。后来一位加利福尼亚的小伙子为了纪念图灵，开办了一家公司，而公司的Logo就是图灵死时手里拿着的被咬过一口的苹果，这家公司就是现在很出名的苹果公司，而那个小伙子则是苹果的第一任CEO乔布斯.



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。
- 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 算法与数据结构

- 描述算法可以有多种方式
  - 自然语言方式、表格方式、图示形式等
  - 本书将采用C++与自然语言相结合的方式
- 算法与数据结构的关系
  - 不了解施加于数据上的算法就无法决定如何构造数据，可以说算法是数据结构的灵魂；
  - 反之算法的结构和选择又常常在很大程度上依赖于数据结构，数据结构则是算法的基础。
- 算法＋数据结构＝程序

1.3 算法描述

- 算法是要通过程序才能加以实现的。常用的算法描述方式：

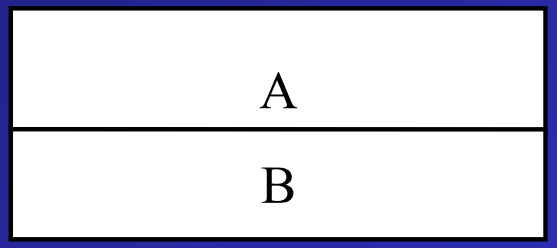
1. 自然语言

- 自然语言就是人们日常使用的语言，可以是中文、英文等。
- 例如，求3个数中最大者的问题，可以描述为：
- ① 比较前两个数。
- ② 将①中较大的数与第三个数进行比较。
- ③ 步骤②中较大的数即为所求。

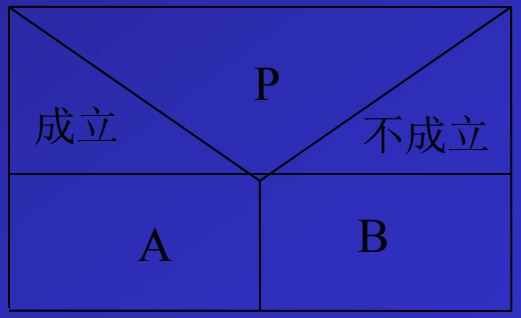
1.3 算法描述

2. 流程图

- 流程图是用规定的一组图形符号、流程线和文字说明来描述算法的一种表示方法。
- (1) 顺序结构。程序执行完A语句后接着执行B语句，如图所示。

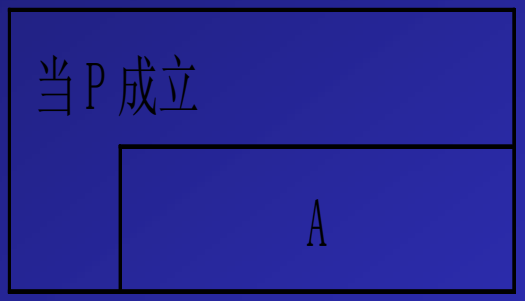


- (2) 选择结构。当条件P成立时，则执行A语句，否则执行B语句，如图所示。

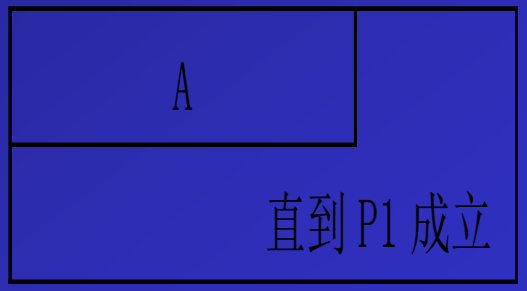


1.3 算法描述

- (3) 当型循环结构。当条件P成立时，则循环执行A语句，如图所示



- (4) 直到型循环结构。循环执行A语句，直到条件P1成立为止，如图所示。



1.3 算法描述

3. 伪代码

- 伪代码是用一种介于自然语言与计算机语言之间的文字和符号来描述算法，它比计算机语言形式灵活、格式紧凑，没有严格的语法。

例如，求两个数的较大者，用伪代码描述算法如下：

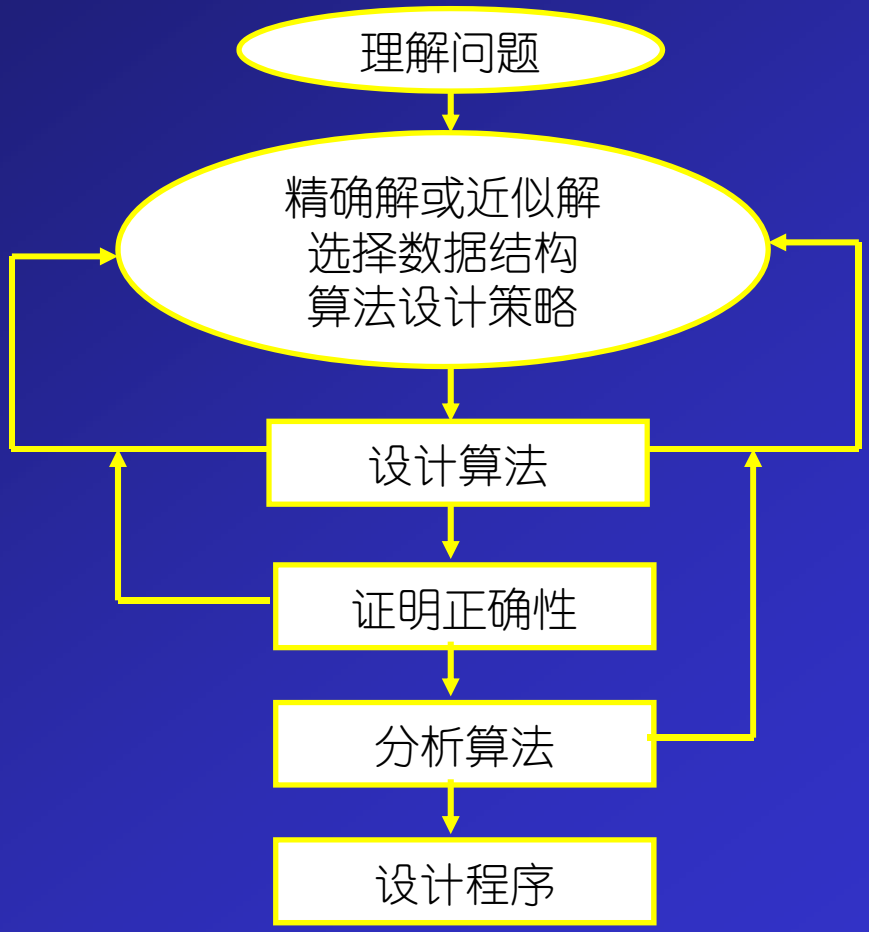
```
Find the bigger
Input: two number s:a,b
1.  if (the first number a is greater than or equal to the  second number b)
      then
      1.1  return a
else
      1.2  return b
end if
      end
```

1.3 描述算法与算法设计

- 采用C++语言来描述算法。
- C++语言的优点是类型丰富、语句精炼，具有面向对象和面向过程的双重优点。
- 用C++来描述算法可使整个算法结构紧凑、可读性强。
- 在课程中，有时为了更好地阐明算法的思路，我们还采用C++与自然语言相结合的方式来描述算法。
- 算法设计方法主要有：分治策略、动态规划、贪心算法、回溯法、分支限界、概率算法等，我们将在后面的章节中陆续介绍。

1.4 算法设计

- 问题求解(Problem Solving)



1.4 算法设计

- 1、问题的陈述

准确地陈述问题是解决问题的第一步。为了设计求解某一问题的算法，首先必须了解问题的实质。即已知条件是什么？要求回答是什么？一个问题的正确描述应当使用科学的语言把所有已知条件和需要的答案陈述清楚。

1.4 算法设计

➢2、模型的选择或拟制

当问题陈述清楚后，选择或拟制描述问题的数学模型是非常重要的工作。模型适当与否影响算法设计的速度和算法的效率。模型选择无公式可循，取决于设计者的知识结构、工作经验等。设计者应当十分重视模型的选择，宁可多下功夫来选择或拟制一个适合当前问题的数学模型，才能为后续工作的顺利开展铺平道路。

1.4 算法设计

- 3、算法设计和正确性证明

数学模型一经选定，就可以进行算法设计。算法的选择与模型的选择是密切相关的。对同一模型仍有不同的算法，这些算法的有效性可能相差很大。算法的设计是指设计求解某个具体问题类的一般步骤，并且这些步骤通过计算机的各种操作来实现。算法设计是一种复杂的、艰苦的创造性劳动，要求设计者充分发挥主观能动性，充分运用已知知识和抽象思维，形成算法思想，写出算法的具体步骤。

1.4 算法设计

- 4、算法的程序实现

将一个算法正确地编写成一个机器程序，叫做算法的程序实现。将给定的算法正确地转换成一个程序，并不是一个简单的工作，要求设计者掌握多种程序设计方法和技巧。

1.4 算法设计

- 5、算法分析

算法分析与算法设计有必然联系，算法设计的中心任务对于现实生活中提出的某些问题是如何设计出一个求解的算法。算法分析研究各种算法的特征和优劣。主要关心如何判定算法的优劣，标准是什么？

1.4 算法设计

- 6、程序的测试、文档编制

编制完实现算法的程序后，对程序进行测试，程序测试可以被认为是对程序应完成的任务的实验验证，同时确定程序的使用范围。编制文档应该和算法设计及实现阶段交织在一起。编制文档的主要目的是让人了解别人编写的算法和程序。

1.5 算法分析的基本原则

1. 正确性

- 定义：在给定有效输入后，算法经过有限时间的计算并产生正确的答案，就称算法是正确的。
- 正确性证明的内容：
  - 方法的正确性证明——算法思路的正确性。证明一系列与算法的工作对象有关的引理、定理以及公式。
  - 程序的正确性证明——证明所给出的一系列指令确实做了所要求的工作。
- 程序正确性证明的方法：
  - 大型程序的正确性证明——可以将它分解为小的相互独立的互不相交的模块，分别验证。
  - 小模块程序可以使用以下方法验证：数学归纳法、软件形式方法等。

1.5 算法分析的基本原则

2. 占用时间——时间复杂性分析

- 计量工作量的标准: 对于给定问题，该算法所执行的基本运算的次数。
- 基本运算的选择：根据问题选择适当的基本运算。

问题	基本运算
在顺序表中查找x	比较
实矩阵相乘	实数乘法
排序	比较
遍历二叉树	置指针

1.5 算法分析的基本原则

3. 占用空间——空间复杂性分析

- 两种占用：
  - 存储程序和输入数据的空间
  - 存储中间结果或操作单元所占用空间——额外空间
- 影响空间的主要因素：
  - 存储程序的空间一般是常数(和输入规模无关)
  - 输入数据空间为输入规模O(n)
- 空间复杂性考虑的是额外空间的大小。
- 如果额外空间相对于输入规模是常数，称为原地工作的算法。

1.5 算法分析的基本原则

4. 简单性
   - 含义：算法简单，程序结构简单。
   - 好处：
     - 容易验证正确性
     - 便于程序调试
   - 简单的算法效率不一定高。要在保证一定效率的前提下力求得到简单的算法。

1.5 算法分析的基本原则

5. 最优性
   - 含义：指求解某类问题中效率最高的算法
   - 两种最优性
     - 最坏情况下最优：设A是解某个问题的算法，如果在解这个问题的算法类中没有其它算法在最坏情况下的时间复杂性比A在最坏情况下的时间复杂性低，则称A是解这个问题在最坏情况下的最优算法。
     - 平均情况下最有：设A是解某个问题的算法，如果在解这个问题的算法类中没有其它算法在平均情况下的时间复杂性比A在平均情况下的时间复杂性低，则称A是解这个问题在平均情况下的最优算法。

1.5 算法分析的基本原则

- 例：在n个不同的数中找最大的数。
- 基本运算：比较
- 算法：Find Max
- 输入：数组L，项数 n =1 1
- 输出：L中的最大项MAX

1) MAX←L(1); i←2;
2) while i≤n do
3) if MAX<L(i) then MAX←L(i);
4) i←i+1;
5) end。

- 行3的比较进行n－1次， 故W(n) = n－1。
- 定理：在n个数的数组中找最大的数，并以比较作为基本运算的算法类中的任何算法最坏情况下至少要做n－1次比较。
- 证：因为MAX是唯一的，其它的n－1个数必须在比较后被淘汰。一次比较至多淘汰一个数， 所以至少需要n－1次比较。
- 结论：Find Max算法是最优算法。

1.6 算法复杂性分析

- 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。这个量应该只依赖于算法要解的问题的规模、算法的输入和算法本身的函数。如果分别用N、I和A表示算法要解问题的规模、算法的输入和算法本身，而且用C表示复杂性，那么，应该有$C=F(N,I,A)$。
- 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$S=S(N,I)$。（通常，让A隐含在复杂性函数名当中）

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）。

1.6 算法复杂性分析

- T(n)= T(N,I,A) 具体可以表示成什么形式呢?
- 可以用以下的分析得到:
- 例1:
- 设一台抽象的计算机可以提供K种元运算,记为:$O_1,O_2,\ldots O_K$每种原运算所需要的时间分别为t_i (i=1, 2, . . k)对于某个算法A,经过统计,用到原运算O_i (i=1, 2, . . k)的次数为e_i次(i=1, 2, ...k),那么该算法的时间复杂度T(n)可表示为
-
- $$T(n)= \sum_{i=1}^{k} e_i * t_i(N,I)$$

1.6 算法复杂性分析

- （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
- $T_{\mathrm{avg}}(n) = \sum_{size(I)=n} p(I)T(I)$
- 其中I是问题的规模为n的实例，$p(\mathrm{I})$是实 例I出现的概率。T(I)为该实例下的运算次数(或所需时间)

1.6 算法复杂性分析

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

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

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

- 以上分别是最坏情况下、最好情况下和平均情况下的时间复杂性。
- 其中D_N是规模为N的合法输入的集合；I^*是D_N中使$T(N, I^*)$达到$T_{Max}(N)$的合法输入；I^-是D_N中使$T(N, I^-)$达到$T_{Min}(N)$的合法输入；而$P(I)$是在算法的应用中出现输入I的概率。

1.6 算法复杂性分析

- 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）最坏情况 $T_{max}(n) = \max\{ T(I) \mid size(I)=n \} = n$

（2）最好情况 $T_{min}(n) = \min\{ T(I) \mid size(I)=n \} = 1$

1.6 算法复杂性分析

- （3）在平均情况下，假设：
- (a) 搜索成功的概率为p（ $0 \le p \le 1$ ）；
- (b) 在数组的每个位置i（ $0 \le i < n$ ）搜索成功的概率相同，均为 p/n。

$$ASL_{ss} = \sum_{i=1}^{n} P_i C_i = \frac{1}{n}\sum_{i=1}^{n}(n-i+1) = \frac{n+1}{2}$$

$$ASL'_{ss} = \frac{1}{2n}\sum_{i=1}^{n}(n-i+1) + \frac{1}{2}(n+1) = \frac{3}{4}(n+1)$$

该结果表明,如果要查找的元素一定在表中的话,即p=1时,那么平均情况下需要查找(n+1)/2个线性表的元素.如果p=1/2,那么平均情况下需要查找3(n+1)/4个线性表中的元素。

应用: 比较以下两算法好坏

- 例3:对同一个问题P,假设一个算法在最坏情况下的复杂度为: $T_{max1}(n)=3n^2$,而另一个算法的时间复杂度为$T_{max2}(n)=25n$,比较那个算法效率更高.
- 当n=8时, $T_{max1}(8)=192$, $T_{max2}(8)=200$
- 当n=9时, $T_{max1}(9)=273$, $T_{max2}(9)=225$
- 显然,仅通过上述的三种情况,还不能完全比较算法的有效性,因此,我们提出算法的渐进性态的概念.

1.6 算法复杂性分析

- 函数的渐进性态与渐进表达式：一般来说，当N单调增加且趋于∞时，$T(N)$也将单调增加趋于∞。
- 对于$T(N)$，如果存在函数$T'(N)$，使得当$N\rightarrow\infty$使有$(T(N)-T'(N))/T(N)\rightarrow 0$，那么我们就说$T'(N)$是$T(N)$当$N\rightarrow\infty$时的渐进性态。
- 在数学上，$T'(N)$是$T(N)$当$N\rightarrow\infty$时的渐进表达式。
- 例如：$3N^2+4N\log N+7$与$3N^2$。

1.6 算法复杂性分析

- 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

$$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

1.6 算法复杂性分析

- X = X + 1;　　复杂度：O（1）
- for (I = 1; I<=n; I++)
  X = X + 1;　复杂度：O（n）
- for (I = 1; I<=n; I++)
  for(J=1;J<=I; J++)
  X = X + 1;　复杂度：O（n^2）
- 折半查找　　复杂度：O（logn）
- 快速排序、堆排序　　复杂度：O（nlogn）

1.6 算法复杂性分析

- 函数的增长
- 1 大多数程序的大部分指令执行一次，或者至多只执行几次。如果一个程序的所有指令均具有这个性质，我们说程序的运行时间为常量。
- log N 当程序的运行时间为对数时，程序随着N的增长稍微变慢。一般在求解一个大规模问题的程序中，若把问题变成一些小问题（规模缩小几分之几）的时候，会出现这样的运行时间。当N加倍时，log N 只增加常量，只有N增加到N^2时， log N 才会加倍。

1.6 算法复杂性分析

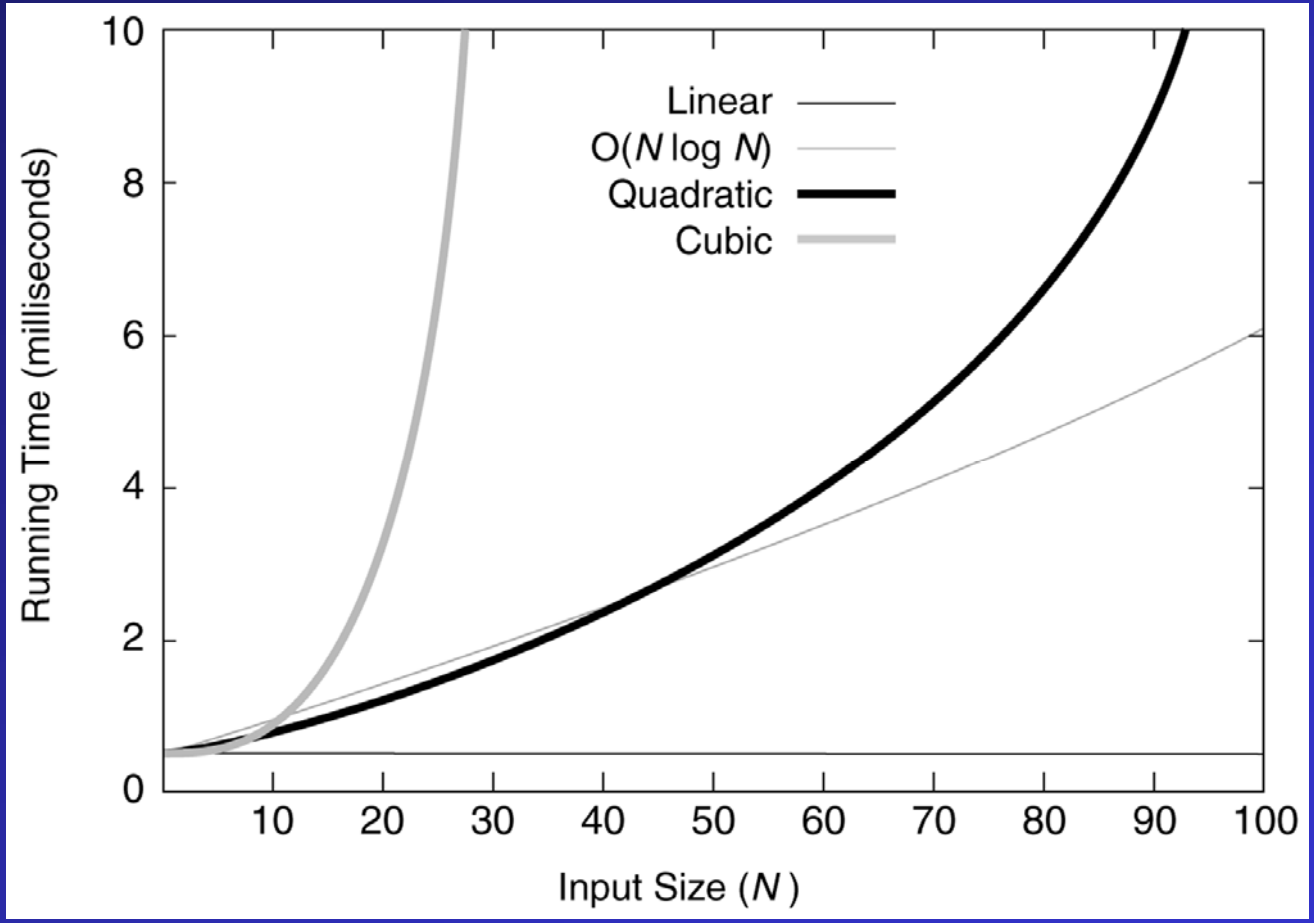
- N 当程序的运行时间为线性时，通常对每个输入元素只作了少量处理工作。当N = 1000 000时，运行时间也为1000 000.当N加倍时，运行时间也随之加倍。这种情况对于一个必须处理N个输入（或者产生N个输出）的算法是最优的。
- NlogN 当把问题分解成小的子问题，且独立求解子问题，然后把这些子问题的解组合成原问题的解时，就会出现NlogN 的运行时间。当N = 1000 000时， NlogN 约为20 000 000，当N加倍时，运行时间略多于两倍。

1.6 算法复杂性分析

- N^2 运行时间为N^2时，算法只适用于规模相对小的问题。二次运行时间一般出现在需要处理所有数据项对的（也许是双层嵌套循环）的算法中。当N=1000时，运行时间为1000 000.当N加倍时，运行时间增加四倍。
- N^3 类似的，处理三个数据项的算法（或许是三层嵌套循环），算法只适用于小规模的问题。当N=100时，运行时间为1 000 000。当N加倍时，运行时间增加八倍。
- 2^N 一个指数运行时间的算法很难在实际中使用，当N=20时，他的运行时间为1 000 000。当N加倍时，运行时间是原来的平方！

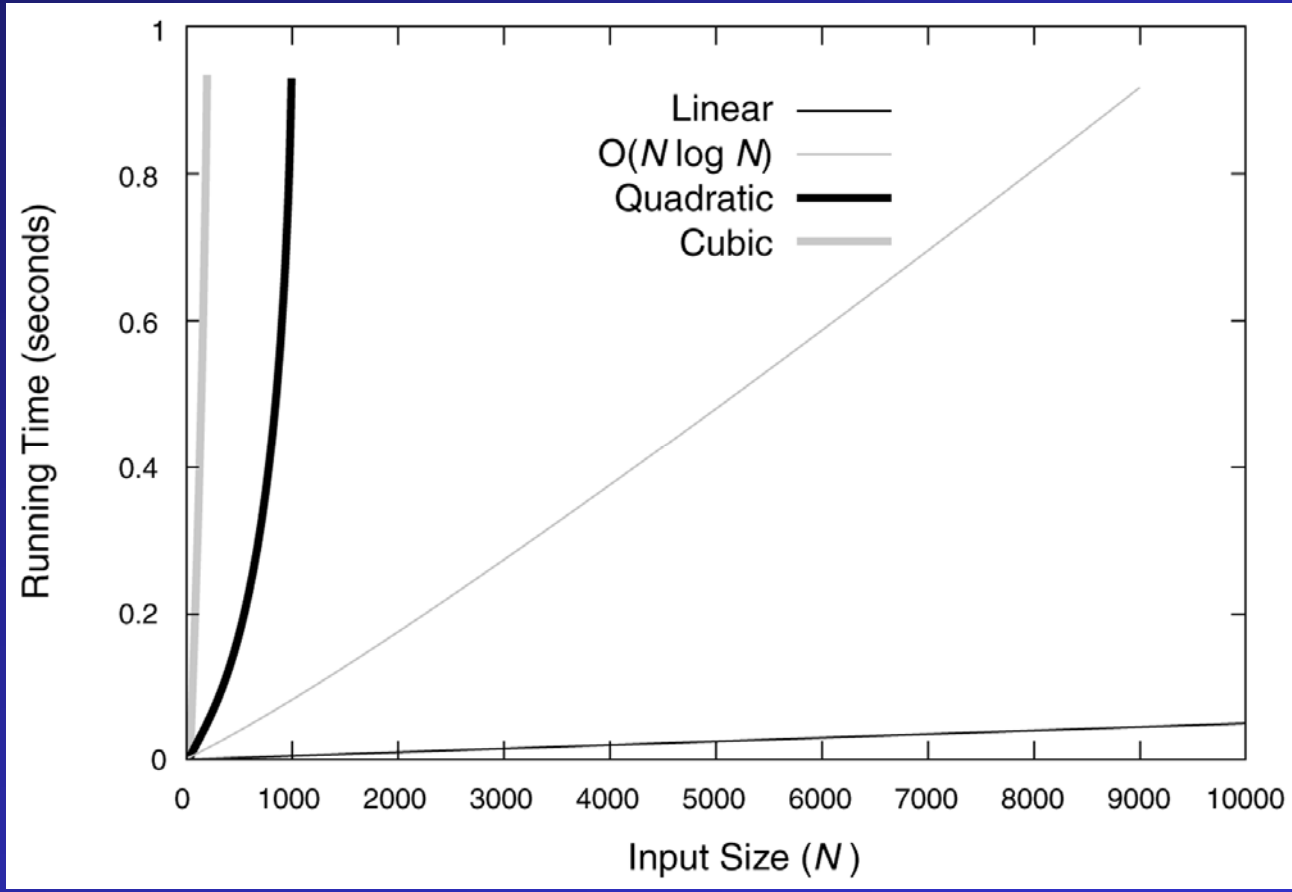
1.6 算法复杂性分析

- 小规模数据复杂性增长图



1.6 算法复杂性分析

- 中等规模数据复杂性增长图



1.6 算法复杂性分析

- 为了降低程序的总运行时间，我们把注意力放在**使内层循环中的指令数最少上**。
- 对于小规模的问题，我们采取哪种方法差别不是很大，在一台快速的现代计算机上很快就能完成任务。但随着规模不断增大，即使是对于最快的计算机，执行这些指令所需要的时间也变得难以接受。实验表明：**快速算法比快速计算机在我们面临极度运行时间的问题时更能帮助我们解决问题。**

摸底测试

➢1、打印九九乘法表

1×1=1								
1×2=2	2×2=4							
1×3=3	2×3=6	3×3=9						
1×4=4	2×4=8	3×4=12	4×4=16					
1×5=5	2×5=10	3×5=15	4×5=20	5×5=25				
1×6=6	2×6=12	3×6=18	4×6=24	5×6=30	6×6=36			
1×7=7	2×7=14	3×7=21	4×7=28	5×7=35	6×7=42	7×7=49		
1×8=8	2×8=16	3×8=24	4×8=32	5×8=40	6×8=48	7×8=56	8×8=64	
1×9=9	2×9=18	3×9=27	4×9=36	5×9=45	6×9=54	7×9=63	8×9=72	9×9=81

➢2、气球射击问题（字节跳动面试真题）

射击气球的游戏中，如果在连续T枪中打爆了所有颜色的气球，将得到奖励。（每种颜色的球至少被打爆一只）。

假设游戏中有m种不同颜色的气球，编号1到m；共有n发子弹，然后连续开了n枪。求：在这n枪中，打爆所有颜色的气球最少用了连续几枪？