

# 算法设计与分析

Computer Algorithm Design & Analysis

---

赵峰

[zhaof@hust.edu.cn](mailto:zhaof@hust.edu.cn)

---

在信息技术高速发展的今天，算法被认为是计算机科学的基石。

David Harel在他的《Algorithmics: The Spirit of Computing》一书中说到：“算法不仅是计算机科学的一个分支，它更是计算机科学的核心”。

当今计算机技术的进步：大数据、AI等，算法的进步是关键

。

# 关于算法教学

## □系统地学习算法，培养专业素养

- ◆ 掌握基本的概念、理论、方法和技术
- ◆ 奠定XXX基础

## □学习方法

- ◆ 勤于思考
  - 要思考每个算法背后的东西，是怎么被想出来的？设计原则是什么？并转化为自己的认知能力。
- ◆ 理论和实践要结合，设计新算法，求解新问题。

# 关于算法教学

## 课程核心：

介绍算法设计与分析的基本理论、方法和技术，训练程序思维，奠定算法设计的基础。

## 教学目的：

- 在理论学习上，掌握算法设计与分析的基本理论和方法，培养设计算法和分析算法的能力。
- 在实践教学上，掌握算法实现的技术、技巧，学习算法的正确性验证、效率分析、优化技术，以及算法在实际问题中的分析与应用。
- 培养独立研究和创新能力。

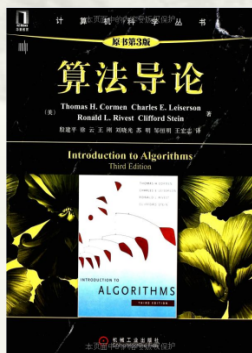
## 本课程需要的基础

---

- ◆ 数据结构 ✓
- ◆ 程序设计语言（C/C++）：结构化设计 ✓
- ◆ 一定的数学基础：高数、离散、概率 ✓
- ◆ 一些背景知识：操作系统、编译

# 主要参考书

- **Introduction to algorithms**, Thomas H. Cormen, etc., third edition, The MIT Press.
- **计算机算法基础**, 余祥宣等编著, 华中科技大学出版社
- **Algorithm Design**, Jon Kleinberg, Eva Tardos etc. Cornell University
- **Algorithms**, Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani



推荐阅读

# 其它参考书

- **The Art of Computer Programming**, Donald E.Knuth.  
Volume 1-3, Second Edition.
- Data Structures, Algorithms, and Applications in C++(Part 3) Sartaj Sahni, China Machine Press
- 算法设计与分析, 王晓东, 清华大学出版社
- etc.





---

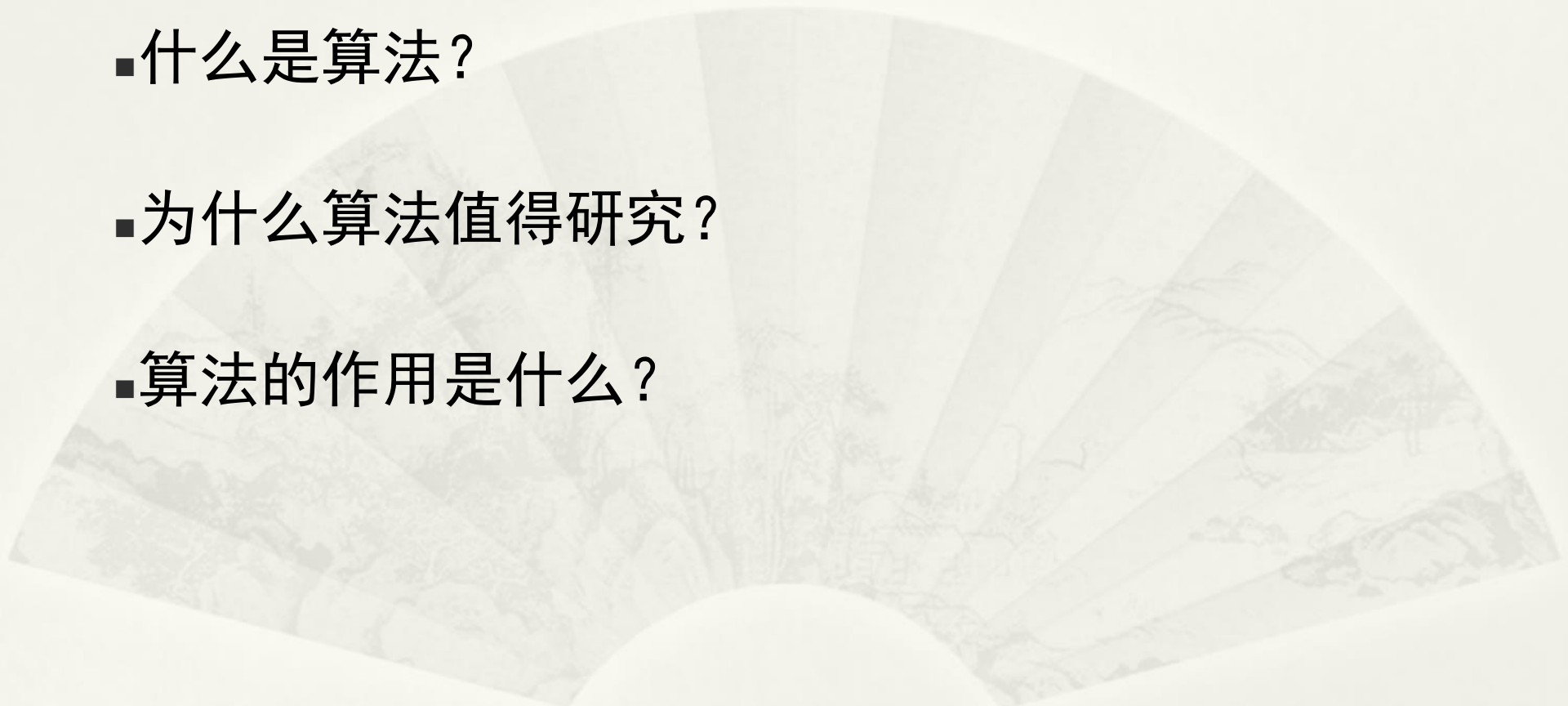
## Chapter 1

# The Role of Algorithms in Computing 算法在计算中的作用



本章回答尝试以下问题：

---

- 什么是算法？
  - 为什么算法值得研究？
  - 算法的作用是什么？
- 

# 1.1 What are algorithms?

---

非形式地说，算法就是任何良定义（well-defined）的计算过程，该过程取某个值或值的集合作为输入（input），并产生某个值或者值的集合作为输出（output）。

算法是一组有穷的规则，它规定了解决某一特定类型问题的一系列运算。（选自《计算机算法基础》）

一般来说，问题陈述说明了期望的输入/输出关系，而算法描述了一个特定的计算过程来实现输入到输出的转换。

Example: 排序问题 (*sorting problem*) 的形式定义如下

**Input:** A sequence of  $n$  numbers  $\langle a_1, a_2, \dots, a_n \rangle$ .

**Output:** A permutation (reordering)  $\langle a'_1, a'_2, \dots, a'_n \rangle$  of the input sequence such that  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

For example,

Input sequence: (31, 41, 59, 26, 41, 58)

Output sequence : (26, 31, 41, 41, 58, 59)

← instance

问题实例(instance): 由计算一个问题所必需的输入组成。

排序算法: 冒泡排序、插入排序、归并排序等，将完成上述的排序过程

# 算法无处不在

## ■从应用领域来看：

- 人类基因工程（The Human Genome Project）：会用到很多算法设计思想，如：LCS in DNA comparison.
- 互联网应用（The Internet Applications）：管理和处理大数据（Big Data）、网络路由、搜索引擎的设计都离不开算法
- 电子商务（Electronic commerce）：利用数值算法和数论进行个人信息的加密保护
- 制造业和其他应用领域

## ■从具体问题来看：

- 图计算
- 字符串处理
- 计算几何问题
- 最优化问题
- Etc. See page from 3 to 5

- ▶ 若对每个输入实例，算法都以正确的输出停机，则称该算法是**正确**的，并称正确的算法解决了给定的问题。
- ▶ **不正确的算法**对某些输入实例可能根本不停机，也可能给出一个回答然后停机，但结果与期望不符甚至相反。
  - ▶ 但不正确的算法也并非绝对无用，在不正确的算法错误率可控时，有可能也是有用和有效的。
  - ▶ 但通常，我们还只是关心正确的算法。

## 1.2 作为一种技术的算法

- **效率** ( Efficiency ) 对于算法的有效性有非常重要的影响。

现实应用中，**时间和空间都是有限的资源**，因此我们应选择时间和空间方面有效的算法来求解问题

为求解相同问题而设计的不同算法在效率方面常常具有显著的差别，这种差别可能比由于硬件和软件造成的差别还要重要的多。

For example : 排序问题

- *insertion sort*: time roughly equal to  $c_1 n^2$
- *merge sort*: time roughly equal to  $c_2 n \lg n$

Let  $c_1 = 2$ , and  $c_2 = 50$ ,  
假设对1000万个数进行排序

令  $c_1 = 2$ , and  $c_2 = 50$ , 对1000万个数进行排序。并设插入排序在每秒执行百亿条指令的计算机上运行, 归并排序在每秒仅执行1000万条指令的计算机上运行, 实际的执行时间有什么差别呢?

\* *insertion sort:*

$$\frac{2 \cdot (10^7)^2 \text{ instructions}}{10^{10} \text{ instructions/second}} = 20,000 \text{ seconds (more than 5.5 hours) ,}$$

\* *merge sort:*

$$\frac{50 \cdot 10^7 \lg 10^7 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 1163 \text{ seconds (less than 20 minutes) .}$$

- \* 可见使用一个运行时间“增长”较慢(时间复杂度低)的算法, 即使采用一个较差的编译器、运行速度较慢的计算机, 在数据规模足够大的时候, 时间效率高的算法也比时间效率低的算法快。



上述的例子表明，我们应该像计算机硬件一样把算法看成是**一种技术**——整个系统的性能不但依赖于选择快速的硬件，而且还依赖于选择有效的算法。

- 随着计算机能力的不断增强，我们使用计算机来求解的问题的规模会越来越大，算法之间效率的差异也变得越来越显著。
- 因此，是否具有算法知识与技术的坚实基础是区分真正熟练的程序员与初学者的一个基本特征。

使用现代计算技术，如果你对算法懂得不多，尽管你也可以完成一些任务，但如果有一个好的算法背景，你可以做更多的事情，也会做得更好一些。

---

## Chapter 2

### Getting Started 算法基础

\* 本章将介绍一个贯穿整个课程、进行算法设计与分析的框架。

\* 包括:

- 问题的描述
- 算法的伪代码描述
- 算法正确性证明
- 算法的分析等.

# 伪代码 ( Pseudocode )

- 在本课程中，我们将用一种“伪代码”书写程序。

伪代码类似于 C, C++, Java, Python, or Pascal , 但没有固定的标准，本书的规范见 **P11, “Pseudocode conventions”**。

- 伪代码和真实代码的区别在于:

- 在伪代码中，关注的是使用最清晰、最简洁的表示方法来说明给定的算法。可以不用关心软件工程的问题。
- 为了更简洁地表达算法的本质，用伪代码描述算法时，常常忽略数据抽象、模块性和错误处理等问题，从而使得算法设计的注意力放在算法最核心的部分。

## 2.1 插入排序 ( Insertion sort )

### ■排序问题的描述：

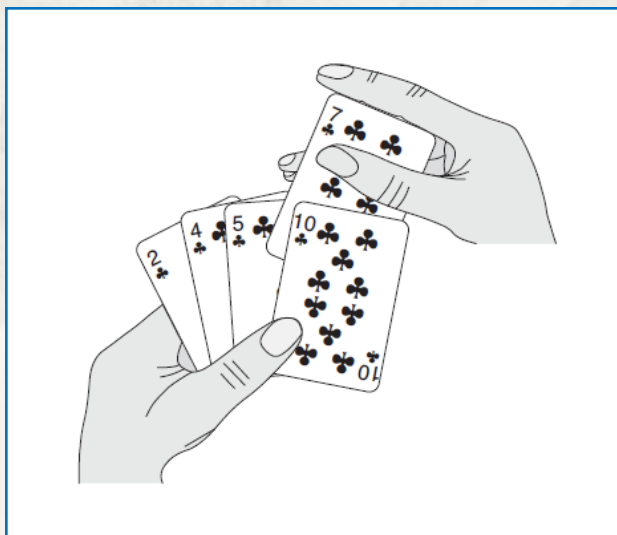
**Input:** A sequence of  $n$  numbers  $\langle a_1, a_2, \dots, a_n \rangle$ .

**Output:** A permutation (reordering)  $\langle a'_1, a'_2, \dots, a'_n \rangle$  of the input sequence such that  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

### 1. 插入排序的基本思想

➤插入排序是一个对少量元素排序比较有效的算法。

➤插入排序的基本思想：



## 2. 插入排序的伪代码描述：INSERTION-SORT

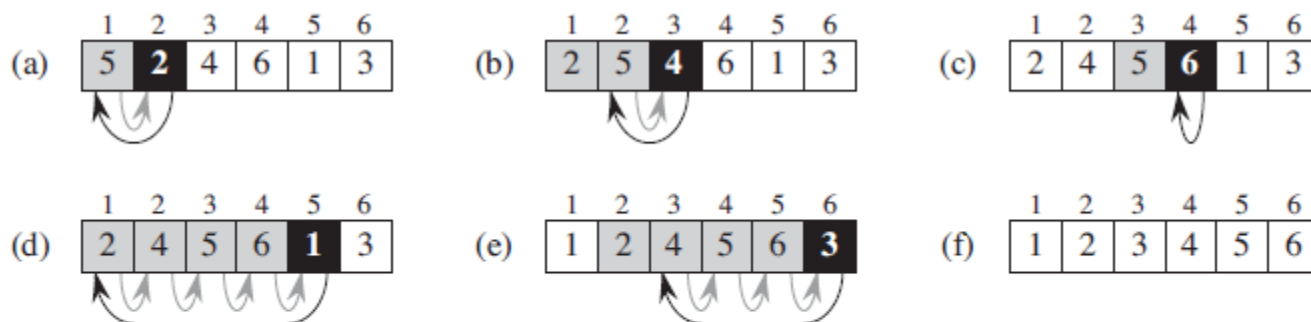
INSERTION-SORT(*A*)

```
1  for  $j = 2$  to  $A.length$ 
2       $key = A[j]$ 
3      // Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```

说明：

设输入的原始数据在数组A中，INSERTION-SORT将“**原址排序**”输入的数：算法在数组A中重排这些数，在任何时候，最多只有其中的常数个数字存储在数组之外。过程结束时，输入数组A包含排序好的输出序列。

设 $A = \langle 5, 2, 4, 6, 1, 3 \rangle$ ，INSERTION-SORT在A上的排序过程如图所示：



**Figure 2.2** The operation of INSERTION-SORT on the array  $A = \langle 5, 2, 4, 6, 1, 3 \rangle$ . Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. (a)–(e) The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from  $A[j]$ , which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. (f) The final sorted array.



### 3. 循环不变式与插入算法的正确性

#### 1) 循环不变式 ( Loop invariants )

分析：

- 在INSERTION-SORT 的for循环中，循环变量为j。j代表当前正要被插入到序列中的元素的下标。而子数组A[1~j-1]是已经被排好序的子序列。
- 这一性质，在j被赋予初值2，首次进入循环之前成立，而且每次循环之后（j加了1）、进入下一次循环之前也成立。

—— 把这种在第一次进入循环之前成立、每次循环之后还成立的关系称为 “循环不变式”。

■ 插入排序for循环的循环不变式可以描述为：

在第1~8行的for循环的每次迭代开始时，子数组A[1~j-1]由原来在A[1~j-1]中的元素组成，但已按序排列。

- 可以利用循环不变关系证明循环的正确性。

分三步走：

---

- 1) 初始化：证明初始状态时循环不变式成立，即证明循环不变式在循环开始之前为真；
- 2) 保持：即证明每次循环之后、下一次循环开始之前循环不变式仍为真；
- 3) 终止：即证明循环可以在有限次循环之后终止。

思路：

- 第1) 和2) 步类似于数学归纳法的证明策略
- 第3) 步保证算法可以终止
- 如果1) ~ 3) 都满足，则说明循环过程正确  
(为什么？有理论保证，自学：程序的逻辑)

- 这里利用循环不变关系，证明插入排序的正确性。

### 1) 初始化：证明循环不变式在循环开始之前为真

第一次循环之前， $j=2$ ，子数组 $A[1..j-1]$ 实际上只有一个元素，即 $A[1]$ ，且这个 $A[1]$ 是 $A$ 中原来的元素。所以表明第一次循环迭代之前循环不变式成立——初始状态成立。

### 2) 保持：证明每次循环之后循环不变式仍为真

观察for循环体，可以看到4~7行是将 $A[j-1]$ 、 $A[j-2]$ 、 $A[j-3]$ ...依次向右移动一个位置（while循环，这里不另行证明），直到找到对当前 $A[j]$  适当的位置，之后在第8行将 $A[j]$ 插入该位置。

这时子数组 $A[1..j]$ 显然是由原来在 $A[1..j]$ 中的元素组成，且已按序排列。再之后 $j+=1$ （下次循环开始之前的状态），此时原来的“ $A[1..j]$ ”变成了新的 $A[1..j-1]$ 。故循环不变式依然成立。

3) **终止**：循环可以有限次终止。

可以看到，每次循环后 $j$ 都加1，而循环终止条件是 $j > n$  (即 $A.length$ )，所以必有在 $n-1$ 次循环后 $j = n+1$ ，循环终止。

注：此时，循环不变式依然成立（即 $A[1..n]$ 是由原 $A$ 中的 $n$ 个元素组成且已排序）。

故此，**上述三条性质都成立**，根据证明策略，插入排序正确（确切地说是其中的for循环正确，但for循环是插入排序过程的主体，所以整个算法正确）

## 2.2 分析算法

- 分析算法的目的是预测算法需要资源的程度。
- 资源包括：
  - 时间
  - 空间：内存
  - 其他：通信带宽、硬件资源等

但从算法的角度，我们主要关心算法的时间复杂度和空间复杂度。

# 1. 分析算法的目的

---

- **算法选择**的需要：对同一个问题可以设计不同的算法，不同的算法对时间和空间需求是不同的。

因此，通过对算法的分析可以了解算法的性能，从而在解决同一问题的不同算法之间比较性能的好坏，选择好的算法，避免无益的人力和物力浪费。

- **算法优化**的需要：有没有可以改进的地方，以使算法工作的更好？通过对算法分析，可以对算法作深入了解，从而可以进一步优化算法，让其更好地工作。



## 2. 重要的假设和约定

### 1) 计算机模型的假设

- 计算机形式理论模型：有限自动机（FA）、Turing机
- **RAM**（random-access machine，随机访问机）模型
  - ◆ 在RAM模型中，指令一条一条被取出和执行，没有并发操作。
  - ◆ 关于RAM的讨论参考P13相关内容。
- **通用的顺序计算机模型**：
  - ◆ 单CPU——串行算法，部分内容涉及多CPU（核）和并行处理的概念
  - ◆ 有足够的“内存”，并能在固定的时间内存取数据单元

区分计算机的理论模型与实际的计算机



## 插入排序时间分析：

INSERTION-SORT( <i>A</i> )	<i>cost</i>	<i>times</i>
1 <b>for</b> <i>j</i> = 2 <b>to</b> <i>A.length</i>	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3     // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

- **cost列**：给出了每一行语句执行一次的时间。这里假定第*i*行语句的执行的时间是 $c_i$ ,且 $c_i$  是一个常量。
- **times列**：给出了每一行语句被执行的次数。其中， $t_j$ 表示对当前的*j*，第5行执行while循环测试的次数。

- 整个算法的执行时间是执行所有语句的时间之和。

- 如果语句*i*需要执行*n*次，每次需要*c<sub>i</sub>*的时间，则该语句的总执行时间是：*c<sub>i</sub>n*.
- 令 *T* (*n*)是输入*n*个值时INSERTION-SORT的运行时间，则有：

$$\begin{aligned} T(n) = & c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\ & + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1) . \end{aligned}$$

**注：**一个算法的执行时间可能依赖于给定的输入，  
即使规模相同。

如：INSERTION-SORT

- **最好情况：**初始数组已经排序好了。此时对每一次for循环，内部的while循环体都不会执行。

最好情况的运行时间为：

$$\begin{aligned} T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

- 第5行做了n-1次测试，但第6、7行没有被执行。
- 执行时间的表达式为n的**线性函数**，即具有an+b的形式。

- **最坏情况**：初始数组是反向排序的。

此时对每一次for循环，内部的while循环体都执行最多次数的测试（即j-1次）。

最坏情况的运行时间为：

$$\begin{aligned} T(n) &= c_1n + c_2(n-1) + c_4(n-1) + c_5 \left( \frac{n(n+1)}{2} - 1 \right) \\ &\quad + c_6 \left( \frac{n(n-1)}{2} \right) + c_7 \left( \frac{n(n-1)}{2} \right) + c_8(n-1) \\ &= \left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

- 执行时间的表达式为n的**二次函数**，即具有 $an^2+bn+c$ 的形式。

- 除了最坏情况、最好情况分析，还有平均情况分析。

平均情况是规模为 $n$ 的情况下，算法的平均执行时间。

- 通常情况下，平均运行时间是算法在各种情况下执行时间之和与输入情况数的比值（算术平均）。

但一般，我们更关心算法的最坏情况执行时间。

为什么？

- 一个算法的最坏情况执行时间给出了任何输入的运行时间的一个上界。知道了这个界，就能确保算法绝不需要更长的时间，我们就不需要再对算法做更坏的打算。
- 对某些算法，最坏情况经常出现。
- 对很多算法，平均情况往往与最坏情况大致一样。
  - 如插入排序，就一般情况而言，while循环中为确定当前 $A[j]$ 的插入位置，平均要检查一半的元素。那么导致的平均执行时间就数量级来说和最坏情况一样，依然是 $n$ 的二次函数，只是常系数小了一些。

## 2. 算法的五个重要特性

确定性、能行性、输入、输出、有穷性

- 1) 确定性：算法使用的每种运算必须要有确切的定义，不能有二义性。
  - 不符合确定性的运算如： $5/0$ ，将6或7与x相加
- 2) 能行性：算法中有待实现的运算都是基本的运算，原理上每种运算都能由人用纸和笔在“有限”的时间内完成。
  - 整数的算术运算是“能行”的，实数的算术运算可能是“不能行”的



# 如何认识算法的确定性和能行性？

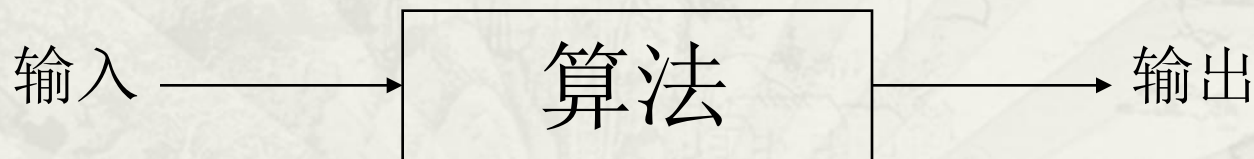
---

- ◆ 确定性和能行性是算法设计过程需要思考的问题。
- ◆ 一个实际的程序设计语言规定了该语言中可以使用的数据类型及其能够参与的运算，编译器可以对程序中的非法运算检错。非确定、非能行的“臆造”运算是不能存在的。
- ◆ 但算法本身的正确性不仅在于此，更在于逻辑的正确性。

3 ) 输入：每个算法都有0个或多个输入。

- 这些输入是在算法开始之前给出的量，取自于特定的对象集合——定义域

4 ) 输出：一个算法产生一个或多个输出，这些输出是同输入有某种特定关系的量。



## 5 ) 有穷性

一个算法总是在执行了有穷步的运算之后终止。

- 计算过程：满足确定性、能行性、输入、输出，但不一定满足有穷性的一组规则称为计算过程。

- 操作系统是计算过程的典型代表：（不终止的运行过程）
- 算法是“可以终止的计算过程”。

▣ **时效性**：实际问题往往都有时间要求，只有在要求的时间内解决问题才是有意义的。

例：国际象棋（启发）、数值天气预报

➤ 基于算法的时效性，只有把在**相当有穷步**内终止的算法投入到计算机上运行才是实际可行的。

——这就要通过“算法分析”，了解算法性质，给出算法计算时间的一个精确的描述，以衡量算法的执行速度，选择合适的算法**有效地**解决问题。