算法设计与分析

Computer Algorithm Design & Analysis

吕志鹏

zhipeng.lv@hust.edu.cn

班级名称：算法设计与分析2025

QQ：419123077

微助教： QE854

Do you know Algorithm 一词的由来

Algorithm(算法)一词的由来本身就十分有趣。

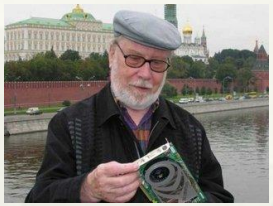
- 这个词一直到1957年之前在《韦氏新世界词典》(Webster ‘s New World Dictionary)中还未出现，只能找到带有古代涵义的相近形式的一个词“Algorism”(算术)，指的是用阿拉伯数字进行算术运算的过程。
- Algorism一词在历史中也经历了漫长的演变，据数学史学家发现研究，诞生于约公元前800多年，但后来“algorism”的形式和意义就变得面目全非了。
- 后来，如牛津英语字典所说明的，这个词是由于同arithmetic(算术)相混淆而形成的错拼词，由algorism变成algorithm。

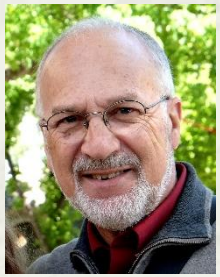
- 软件是计算机的灵魂
- 算法是计算机软件的灵魂
  - 软件=数据结构+算法

  （Niklaus Wirth，瑞士，计算机科学家，图灵奖获得者，Pascal语言的发明者）
  - 算法是计算机科学的核心

  the core of computer science

  （David Harel）
- 算法是国家科技综合实力的体现
- 学习算法具有重要的现实意义





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，培养算法设计和算法分析的能力。
- 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。
- 培养独立思考和创新能力。

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

授课形式:

- 课堂教学：（√）
- 课堂讨论：
- 上机实践：在poj上注册账号，完成定期布置的题目（实践课内容）

成绩评定:

考试：闭卷，70%

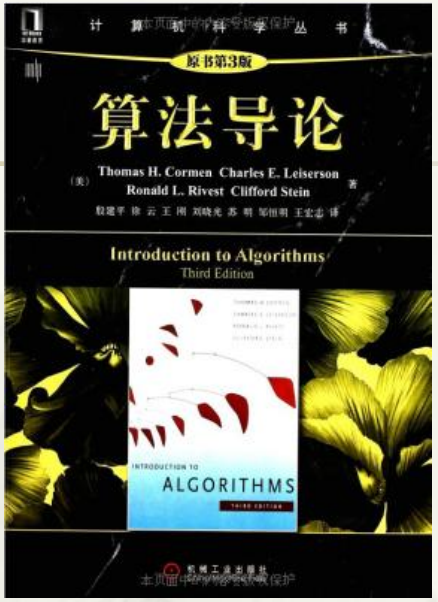
平时成绩：作业+考勤，30%

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

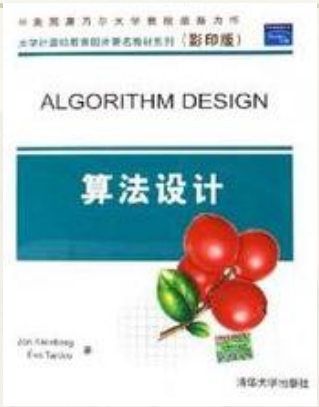
读一本好书很重要

推荐阅读

- **Algorithm Design**

  Jon Kleinberg，Eva Tardos etc.

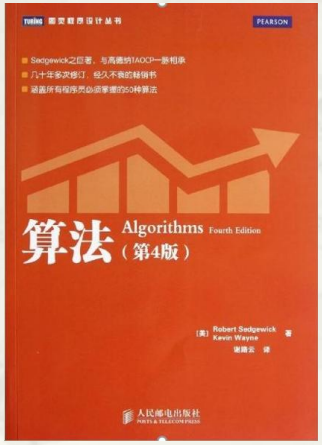
  Cornell University



- **Algorithms**

  Robert Sedgewick / Kevin

  Princeton University



Chapter 1

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

本章尝试回答以下问题：

- 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）。

—— 算法就是把输入转换成输出的计算步骤的一个序列。

—— 在计算机科学中，算法是使用计算机解一类问题的精确、有效方法的代名词；

- 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。

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这种输入到输出的转换。

Example：排序问题

- **问题陈述：**排序问题（*sorting problem*）的形式定义如下

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

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

- **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。

目标一致，但方法各异

什么叫做“正确的算法”？

- 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（相对于最优而言），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（如NP问题和近似算法）。
  - 但通常，我们还只是关心正确的算法（P问题）。

1.2 作为一种技术的算法

- 对于算法，不仅强调其正确性，还有其性能的好坏。
  - 现实应用中，时间和空间都是有限的资源，我们应选择时间和空间方面有效的算法来求解问题。
  - 效率（Efficiency）对于算法的有效性有非常重要的影响。

为求解一个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。

For example：排序问题

insertion sort: time roughly equal to c_1n^2

merge sort: time roughly equal to c_2nlgn

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（10^7）个数进行排序。并设插入排序在每秒执行百亿（10^{10}）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，计算机速度相差1000倍。程序的实际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、时间效率高的算法，即使采用较差的编译器、运行速度较慢的计算机，在数据规模足够大的时候，也比时间效率低的算法快。

- 上述的例子表明，我们应该像对待计算机硬件一样把算法看成是一种技术，研究并选择有效的算法来解决现实问题。
  - 思考：如果计算机速度无限快、内存无限大，那么我们还有研究算法的必要吗？
- 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
  - 算法分析基础等

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：非真实代码，书写类似于 C/C++, Java等实际语言，主要用于描述算法的计算步骤而非放在实际的计算机上运行，没有固定的标准，本书的规范见**P11，“Pseudocode conventions”**。
- 以下用一个例子：插入排序算法，说明算法的伪代码描述和算法证明等相关问题。

2.1 插入排序（Insertion sort）

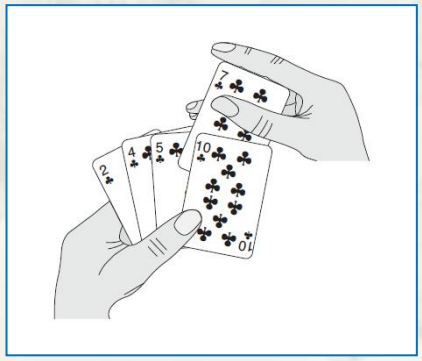
- 排序问题的描述：

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

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

1. 插入排序的基本思想

- 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

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
```

变量可以没有类型说明

语法结构类似现实语言

- INSERTION-SORT是一种“**原址排序**”：输入的原始数据在数组A中， 算法在数组A空间中重排这些数，并且在任何时候，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=<5,2,4,6,1,3>， INSERTION-SORT在A上的排序过程如图所示：

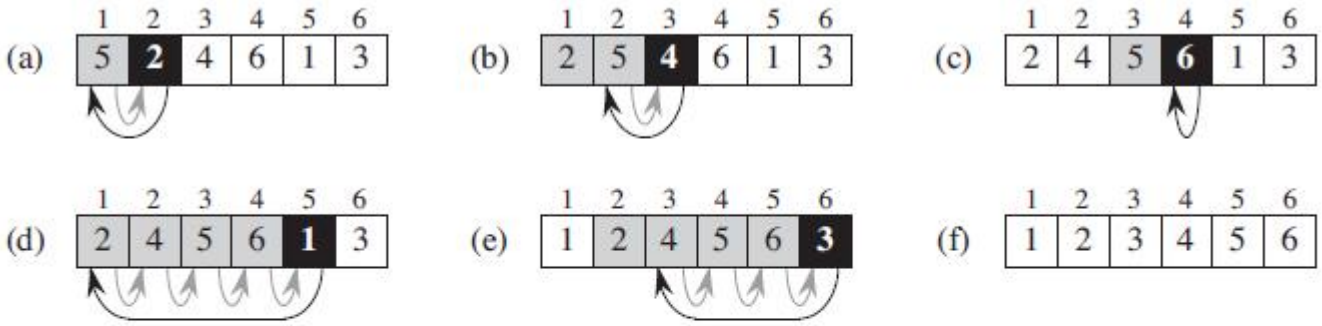


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle 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. 使用循环不变式证明循环/算法的正确性

1）循环不变式（Loop invariants）

以INSERTION-SORT为例，其for循环中，循环变量为j，循环过程具有以下性质（j是当前正要被插入到序列中的元素所在的位置下标）。 ：

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、以后每次循环之后还成立的关系称为“循环不变关系”或“循环不变式”、“循环不变性质”。

- 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，且已按序排列。

- 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：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：证明循环可以在有限次循环之后终止。

证明循环正确性的思路：

- 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

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，执行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，插入排序算法是正确的 —— 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

- 分析算法目的主要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
- 算法优化的需要

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模型（random-access machine，随机访问机）
- 通用的顺序计算机模型：
  - 单CPU——串行算法（部分内容可能涉及多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

2）计算的约定

算法/程序的执行时间是什么?

$O(n^2)$、$O(n\log n)$...和执行时间有什么关系呢?

- 算法的执行时间是算法中所有运算执行时间的总和

可以表示为:

$$\text{算法的执行时间}=\sum f_i t_i$$

其中，f_i：是运算i的执行次数，称为该运算的频率计数

t_i：是运算i在实际的计算机上每执行一次所用的时间

- f_i仅与算法的控制流程有关，与实际使用的计算机硬件和编制程序的语言无关。
- t_i与程序设计语言和计算机硬件有关。

运算的分类

依照运算的时间特性，将运算分为时间囿界于常数的运算和时间非囿界于常数的运算。

- 时间囿界于常数的运算：

特点：执行时间是固定量，与操作数无关。

例：　1+1 = 2　vs 10000+10000 = 20000

100*100 = 10000 vs 10000*10000 = 100000000

CALL INSERTIONSORT

- 时间非囿界于常数的运算

特点：运算的执行时间与操作数相关，每次执行的时间是一个不定的量。

如：

- 字符串操作：与字符串中字符的数量成正比，如字符串的比较运算strcmp。
- 记录操作：与记录的属性数、属性类型等有关

时间非囿界于常数的运算时，将其分解成若干时间囿界于常数的运算即可。

如：字符串比较时间t_{string}

$$t_{string} = Length(String) * t_{char}$$

3）工作数据集的选择

- 算法的执行情况与输入的数据有什么样的关系呢？
  - 算法的执行时间与输入数据的规模相关，一般规模越大，执行时间越长。
  - 编制不同的数据配置，分析算法的最好、最坏、平均工作情况是算法分析的一项重要工作

3. 算法分析

- 算法分析的目的是求取算法时间/空间复杂度的限界函数。
- 限界函数通常是关于问题规模n的特征函数，被表示成O、Ω或Θ的形式。

  如：归并排序的时间复杂度是Θ(nlogn)。

- **怎么获取算法复杂度的特征函数?**

如何进行时间分析?

- 统计算法中各类运算的执行次数
  - 频率计数，即算法中语句或运算的执行次数。
    - 顺序结构中的运算/语句执行次数计为1
    - 嵌套结构中的运算/语句执行次数等于被循环执行的次数

例：执行次数的统计

```
x←x+y
```

(a)

```
for i ←1 to n do
    x ← x + y
repeat
```

(b)

```
for i ←1 to n do
    for j ←1 to n do
        x ← x +y
    repeat
repeat
```

(c)

分析：

(a)： x←x+y执行了1次

(b)： x←x+y执行了n次

(c)： x←x+y执行了n^2次

思考：

```
for i ←1 to n do
    for j ←i to n do
        x ← x +y
    repeat
repeat
```

(d)

插入排序时间分析：

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ 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。
- times列：给出了每一行语句被执行的总次数。

t_j>=1表示考查第j个元素时while循环执行的次数

- 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_i的时间，则该语句的总执行时间是：$c_i n$.
  - 令 T (n)是输入n个值时INSERTION-SORT的运行时间，则有：

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

即使规模相同，一个算法的执行时间也可能依赖于给定的输入。

如：INSERTION-SORT

- 最好情况：

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ 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	$n-1$

- **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（ t_j>=1 ）。

所以最好情况的运行时间为：

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

- **最坏情况**：初始数组是反向排序的。

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ 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	$n-1$

- 最坏情况的运行时间为：

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

While循环判断行：t_j从2，3，……，n
While循环内部：则从1,2,……,n-1

- 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。

- 限界函数：取自频率计数函数表达式中的最高次项，并忽略常系数，记为：g(n)。
  - g(n)通常是关于n的形式简单的单项式函数，如：logn，nlogn等
  - g(n)通常是对算法中最复杂的计算部分分析而来的。

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

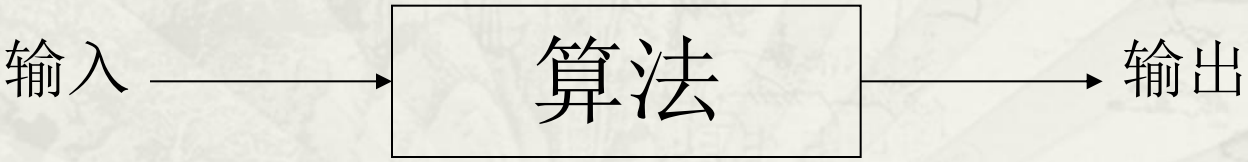
2）能行性：算法中有待实现的运算都是基本的运算，原理上每种运算都能由人用纸和笔在“有限”的时间内完成。

- 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

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。

课外阅读：

算法导论（3rd）：第1章、2.1插入排序、2.2分析算法

其他章节自行阅读

作业1：抄写　通过抄写，熟悉伪代码的书写方法

P10，INSERTIONSORT

P19，MERGESORT

P17，MERGE

作业2：

1.2-2

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

1.2-3

What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

思考题：2.2-2 选择算法