華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

算法设计与分析

刘渝
Liu_yu@hust.edu.cn

2025秋季-华科-计算机
24级CS

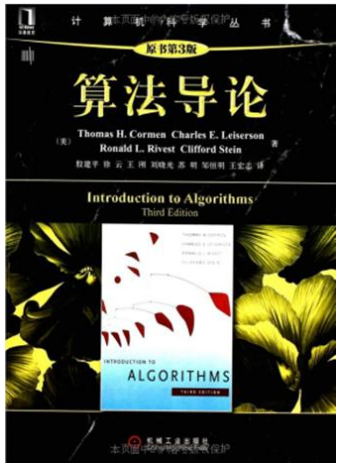
课程导言

教材及教辅
课程基础
学习目标
成绩评定

加入群

群名称：2023春季-算法设计与分析-1
群　号：539344936

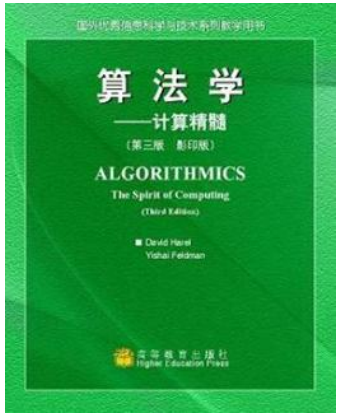
教材及教辅



- **Introduction to algorithms**

  Thomas H. Cormen,etc.,

  third edition, The MIT Press.



- **Algorithmics: The Spirit of Computing**

  David Harel.

基础课程

- 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

学习目标

- 理论上，掌握算法设计与分析的基本理论和方法，具备算法设计和算法分析的能力
- 实践上，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应用与分析
- 培养独立思考和创新能力

授课形式

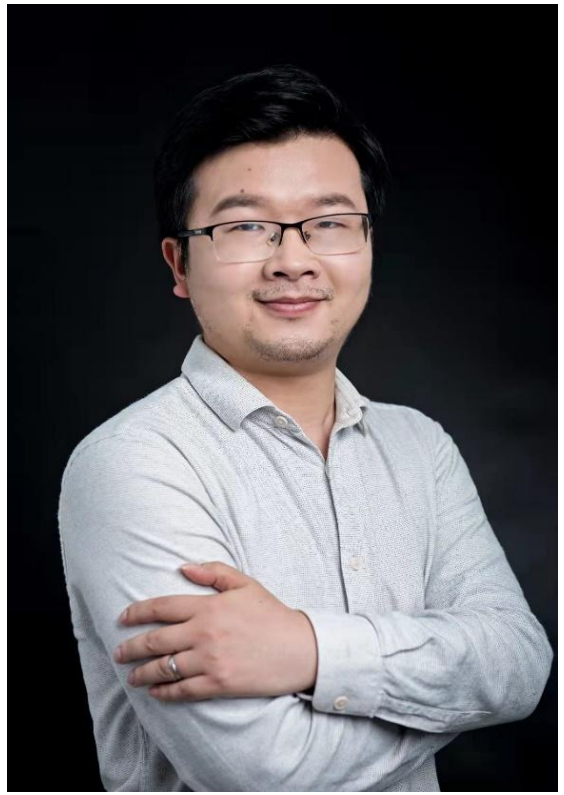
- 课堂教学
- 课堂讨论：课堂作业
- 上机实践：实验，交实验报告

成绩评定

- 考试：闭卷，70%
- 平时成绩：作业(10%) + 上机(15%) + 考勤(5%)

30%

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST



刘渝，华中科技大学、计算机学院副研究员，CCF会员，国家光电研究中心IDSM实验室成员，并任腾讯实习营负责人。

主要从事认知存储、AI for Storage、AIOps。擅长认知存储中的基于相似性hash的大数据研究。获得湖北省技术发明一等奖（2024）；中国电子协会技术发明奖二等奖（2022）。

在SIGMOD, VLDB, ICDE, IJCAI, DAC, MM, TPDS, TIP等多媒体、人工智能、数据库、数据挖掘和存储领域的顶级期刊和权威会议上发表论文50余篇，在APWEB-WAIN会议上获得best paper runner up。作为第一指导老师获得互联网+大学生创新创业国赛金奖一次，铜奖一次；获得“中国软件杯”一等奖一次，三等奖两次；获得2023腾讯开悟人工智能全国公开赛博弈算法高级赛道总决赛亚军，2024腾讯开悟人工智能全球公开赛AI芯片算子开发赛道一等奖。

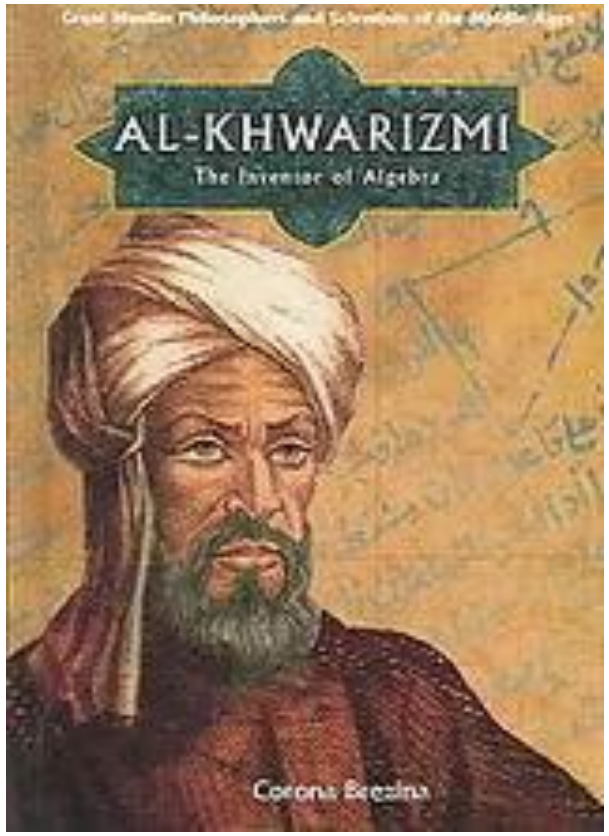
邮箱：liu_yu@hust.edu.cn　　地址：武汉光电国家研究中心大楼C536


华中科技大学计算机科学与技术学院
School of Computer Science &Technology,HUST

算法分析与设计
引言

目　录

历史由来



伟大的波斯数学家阿尔•花瑞子模

阿拉伯数学家Khwarizmi 代指算法
著有 《波斯教科书》
algorithm 是当时拉丁语对该数学家名字的拙劣翻译
也有部分受 arithmetic 的影响。

历史由来

数学大全词典

Algorithmus（算法)一词的如下定义：“在这个名称之下，组合了四种类型的算术计算的概念，即加法、乘法、减法、除法”

几何原本

Algorithm一词经常地同欧几里德算法(Euclid’s algorithm)联系在一起。是求两个数的最大公约数的过程(即辗转相除法)

历史由来

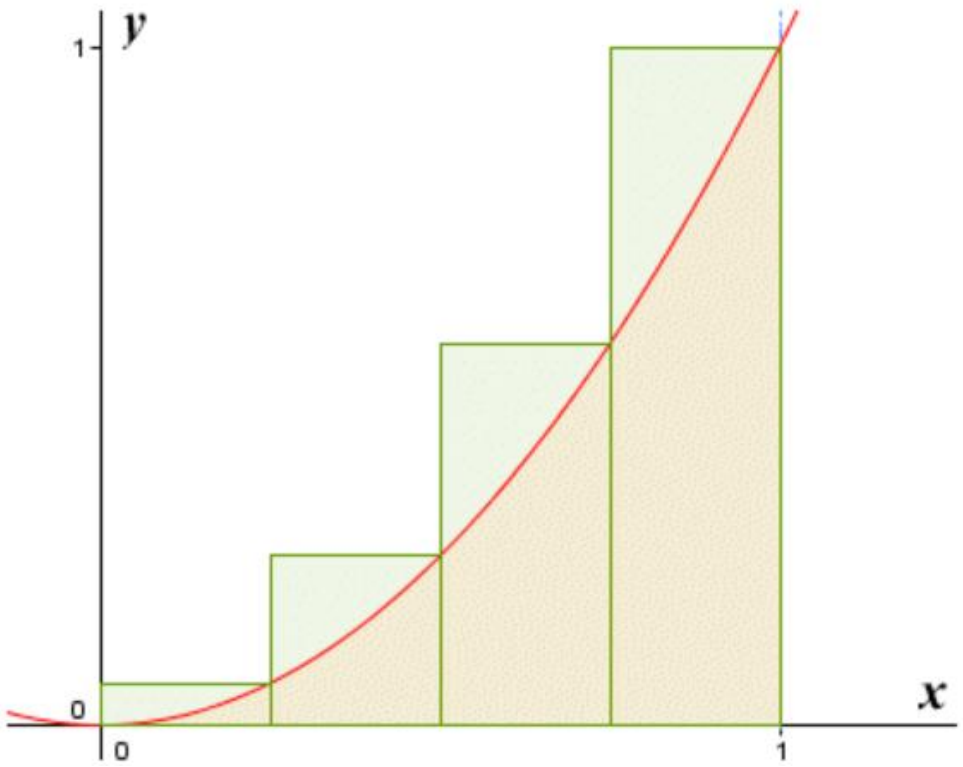
九章算术

中国古代第一部数学专著

百度百科

算法（Algorithm）是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令——“方法”

算与算法



$y=x^2$

用矩形来逼近原图形

$$
\begin{aligned}
S &= \frac{1}{n}\left(\frac{1}{n}\right)^2+\frac{1}{n}\left(\frac{2}{n}\right)^2+\frac{1}{n}\left(\frac{3}{n}\right)^2+\cdots\frac{1}{n}\left(\frac{n}{n}\right)^2 \\
&= \frac{1}{n^3}\left(1+2^2+3^2+\cdots+n^2\right) \\
&= \frac{1}{n^3}\left(\frac{2n^3+3n^2+n}{6}\right) \\
&= \frac{1}{3}+\frac{1}{2n}+\frac{1}{6n^2}
\end{aligned}
$$

```
S=0
for(i=1;i<N;i++){
    S = S + 1/N * pow(i/N,2)
}
```

冰雹猜想

1976年的一天，《华盛顿邮报》于头版头条报道了一条数学新闻。文中记叙了这样一个故事：

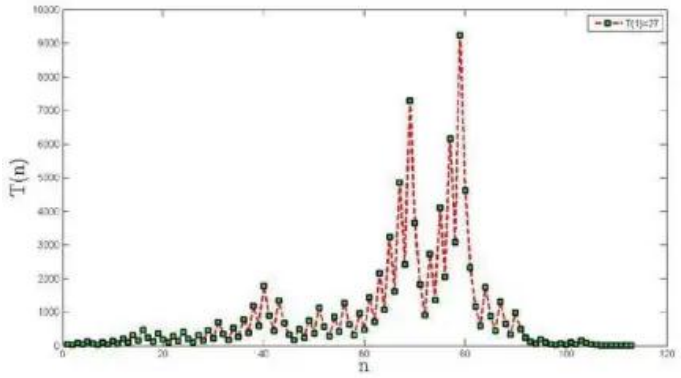
70年代中期，美国各所名牌大学校园内，人们都废寝忘食地玩弄一种数学游戏。这个游戏十分简单：任意写出一个正整数N，并且按照以下的规律进行变换：

如果是个奇数，则下一步变成3N+1。

如果是个偶数，则下一步变成N/2。

冰雹猜想

英国剑桥大学教授John Conway找到了一个自然数27。，27要经过77步骤的变换到达顶峰值9232，然后又经过34步骤到达谷底值1。全部的变换过程（称作“雹程”）需要111步，其顶峰值9232，达到了原有数字27的342倍多，如果以瀑布般的直线下落（2的N次方）来比较，则具有同样雹程的数字N要达到2的111次方。其对比何其惊人！



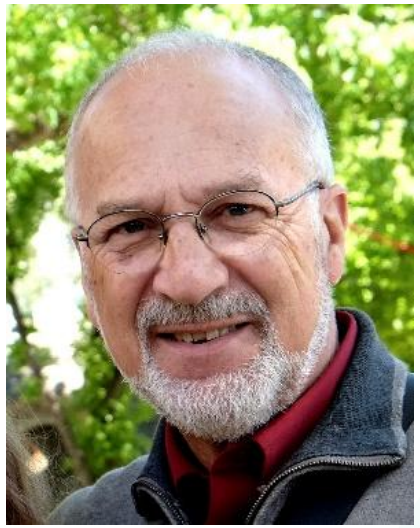
基石论

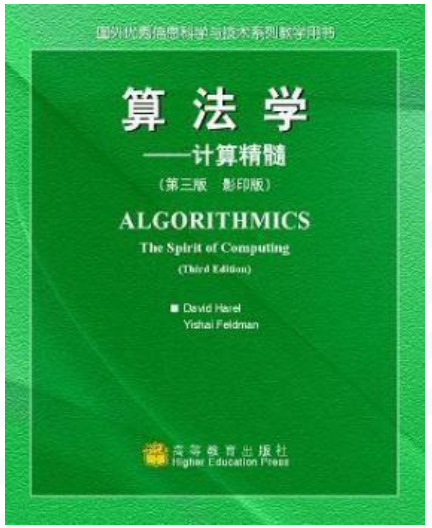
算法被公认为是计算机科学的基石

- 当今计算机技术的进步，其中关键的一部分体现在算法进步上，如大数据、AI技术的发展。
- 算法的应用无所不在：
  - 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,SS（Shotgun Sequencing）
  - 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等
  - 硬件设计
  - 制造业和其他应用领域

算法学

David Harel 在他的《Algorithmics: The Spirit of Computing》（《算法学:计算精髓》）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”



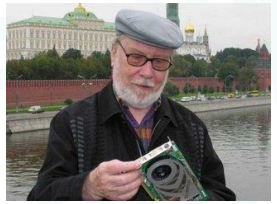


基石论

软件是计算机的灵魂

算法是软件的灵魂

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）

国家科技综合实力

算法分析与设计
第一章
算法在计算中的作用

目　录

什么是算法

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。

—— 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。

什么是算法

算法

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（算法是由运算组成的）

运算

包括算术、逻辑、关系、赋值、过程调用等运算形式

算法性质

- **特殊性**：不同问题有不同的算法，没有求解所有问题的万能的算法

- **有穷性**：不仅书写上规则/运算/语句数量有穷（静态），算法的执行过程时间上也是有限的（动态）

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**：形式化定义如下

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、归并排序等完成上述排序的过程

解决同一个问题的算法，目标是一致的，但具体方法可能是不同的。

什么是正确的算法

若对每个输入实例，算法都以正确的输出终止，则称该算法是正确的，并称正确的算法正确地解决了给定的问题。

不正确的算法对某些输入实例可能根本无法停止，也可能给出一个回答然后终止，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。*

思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

- *在数据结构等课程里我们学会了解决一些具体问题的算法，如排序算法、路径算法。而在这些算法的背后有没有一些共性的东西?*
- *能不能总结出来一些规律和方法，来指导新问题的求解、新算法的设计呢?*

总结：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

较高的算法能力不仅在于简单使用一些具体的算法，更在于对算法设计方法的掌握上。只有深入理解并掌握了算法设计的一般规律和方法，才能在面对新问题时创造出新的算法。

什么是“好”算法

为求解一个问题而设计的不同算法在资源利用和性能方面常常具有显著的差别，这种差别比由于硬件和软件造成的差别还要重要。

资源

时间 空间

性能

效率 精确性

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?

Example：排序问题

* *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*:

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

慢近20倍

* 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。

总结与思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

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。

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?

总结：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

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。

算法分析与设计
第二章
算法基础

目　录

问题与伪代码描述

用伪代码描述问题，书写程序。

伪代码

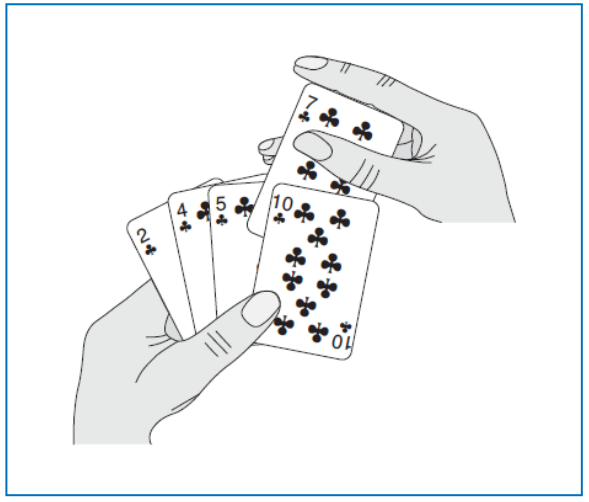
非真实代码，书写类似于 C/C++, Java等实际语言，
主要用于描述算法的计算步骤而非放在实际的计算机上运行，没有固定的标准，本书的规范见P11，“Pseudocode conventions”

Example：排序问题描述

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 $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

插入排序的基本思想：



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的存储空间中重排这些数，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。

Show me your codes!

伪代码描述插入排序算法

```
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

- 插入排序是一种“**原址排序**”：算法在数组A的存储空间中重排这些数，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。

插入排序算法的过程

设A=<5,2,4,6,1,3>， 插入排序在A上的排序过程如图所示：

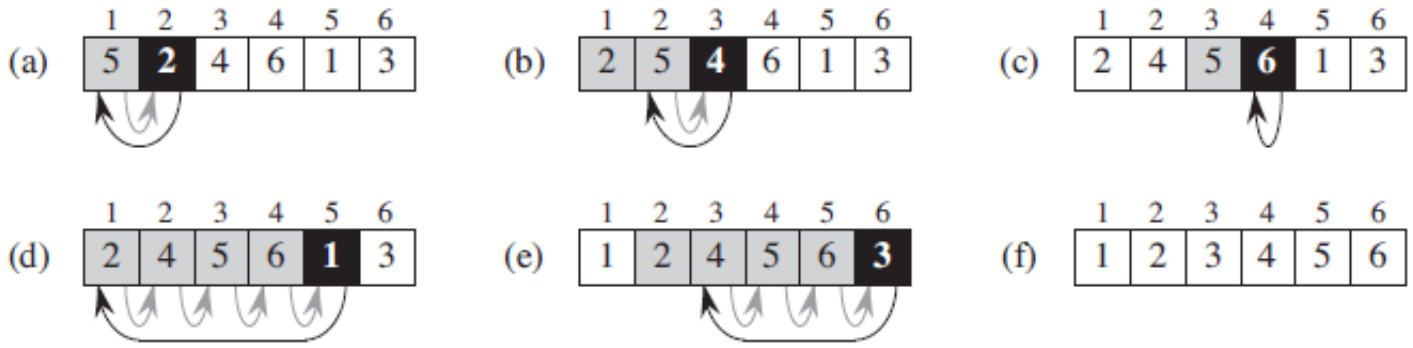


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle 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.

正确性证明

使用循环不变式证明循环/算法的正确性

循环不变式

以插入排序为例，在for循环中，循环变量为j，即当前正在被插入到序列中的元素下标，循环过程具有以下性质：

子数组A[1~j-1]是已经被排好序的子序列。

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”。

插入排序算法正确性

```
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
```

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。

证明三步走

（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迭代之后终止。

（1）初始化

证明循环不变式在循环开始之前为真

```
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
```

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

```
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
```

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]而言适当的位置，之后在第8行将A[j]插入该位置。

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

```
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
```

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

（1）初始化（2）保持（3）终止

```
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
```

故此，上述三条性质都成立，那么根据证明策略可得：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。

算法分析

分析算法的目的主要是预测算法需要资源的程度。

资源

时间　　空间（内存）　　其他（带宽，电力）

从算法的角度，主要关心算法的时间复杂度和空间复杂度

分析算法目的

算法选择

通过对算法的分析，可以了解算法性能，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。因为不同的算法对时间和空间需求是不同的。

算法优化

通过对算法分析，可以对算法作深入了解，发现其中可以改进的地方，从而可以进一步优化算法，让其更好地工作。

约定

区分计算机的理论模型与实际的计算机

形式理论模型

有限自动机（FA）、Turing机

RAM模型

指令一条一条被取出和执行，没有并发操作

顺序计算机模型

单CPU、有足够的“内存”，并能在固定的时间内存取数据单元

思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

日常生活中，时间是以小时、分钟、秒、毫秒为计量单位。

算法/程序的执行时间是什么？

算法的执行时间

算法的执行时间是算法中所有运算执行时间的总和

$$算法的执行时间=\sum f_i t_i$$

其中，f_i：是算法中运算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程序设计语言和计算机硬件有关。

思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

如何确定算法使用了哪些运算，每种运算的f_i和t_i又是多少？

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

- 基本算术运算，如整数、浮点数的加、减、乘、除
- 字符运算、赋值运算、过程调用等

特点：一次执行的时间是固定量，与操作数无关。

时间囿界于常数的运算(一般性质)

设有n种运算c_1，c_2，…，c_n，它们的实际执行时间分别是t_1，t_2，…，t_n。

令t_0 = max(t_1，t_2，…，t_n)，则每种运算执行一次的时间都可以用t_0进行抽象，即可用t_0进行限界(上界)。

进一步，视t_0为一个单位时间，则这些运算“理论”上都可视为仅花一个固定的单位时间t_0即可完成的运算。

时间非囿界于常数的运算

- 字符串操作：与字符串中字符的数量成正比，

  如： 比较运算strcmp

- 记录操作：与记录的属性数、属性类型等有关

特点：执行时间与操作数相关，每次执行的时间是一个不定的量。

时间非囿界于常数的运算（分析）

运算通常是由更基本的运算组成的。而这些基本运算是时间囿界于常数的运算。

分析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。

如：字符串比较时间t_{string} = Length（String）* t_{char}

思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

为什么要引入“时间囿界于常数的运算”这一概念？

结论：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

引入“单位时间”的概念，将“单位时间”视为“1”，从而将算法的理论执行时间转换成“单位时间”意义下的执行次数，从而实现了“执行时间”与时间复杂度概念的统一。

$$\sum f_i t_i$$

工作数据集

- 一般规模越大，执行时间越长
- 同一算法的不同执行情况，可分为最好、最坏和平均等情况讨论

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

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)

算法分析的目的是求取算法时间/空间复杂度的限界函数

特征函数是通过分析算法的控制逻辑得来的，是对算法所用运算执行次数的统计结果。与使用的程序设计语言和计算机硬件无关。

时间分析

统计算法中各类运算的执行次数

统计对象

运算

1）**基本运算**：时间囿界于常数的运算
2）**复合运算**：具有固定执行时间的程序块，如一条语句、复合语句，甚至一个函数，只要是具有固定执行时间即可

统计内容

频率计数
(执行次数)

1）**顺序结构**中的运算：语句执行次数计为1
2）**嵌套结构**中的运算：语句执行次数等于被循环执行的次数

Example

Case 1

```
x ← x+y
```

x ← x+y

执行了1次

Case 2

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

x ← x+y

执行了n次

Case 3

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

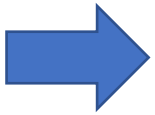
x ← x+y

执行了n^2次

思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

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



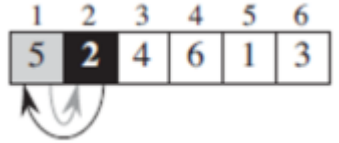
```
for i ← 1 to n do
    for j ← i to n do
        x ← x+y
    repeat
repeat
```

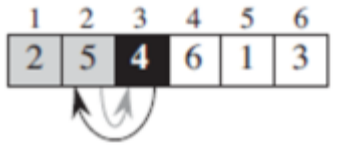
x ← x+y 执行了 次?

哪个算法的执行是如此?

Example：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**：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**：每一行语句被执行的总次数。t_j表示对当前的j，第5行执行while循环测试的次数。

Example：插入排序时间分析

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个值时插入排序的运行时间，则有

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

cost	*times*
c_1	n
c_2	$n-1$
0	$n-1$
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	$n-1$

Example：插入排序时间分析

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while循环的循环体都不会执行。

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	$n-1$

Example：插入排序时间分析

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while循环的循环体都不会执行。所以最好情况的运行时间为：

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

$$c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) \quad \Rightarrow \quad c_5(n-1)$$

执行时间为n的**线性函数**，即具有an+b的形式

Example：插入排序时间分析

最坏情况：初始数组反向排序好了。此时对每一次for循环，内部while循环的循环体都执行最多次数的测试（j-1）。

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	$n-1$

Example：插入排序时间分析

最坏情况：初始数组反向排序好了。此时对每一次for循环，内部while循环的循环体都执行最多次数的测试（j-1）。最坏情况的运行时间为：

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

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

Example：插入排序时间分析

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**）

思考：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

我们更关心算法的那种情况？

为什么？

结论：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

我们更关心算法的最坏执行时间

1）最坏情况执行时间代表任何输入的运行时间上界。知道了这个界，就能确保算法绝不需要更长的时间。

2）对某些算法，最坏情况经常出现。

3）对很多算法，平均情况往往与最坏情况大致一样。

时间复杂度的限界函数

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)

函数表达式中的最高次项的次数称为函数表达式的**数量级**（或阶）

- 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。

时间复杂度的限界函数（特例）

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)

- 对于足够大的n，低阶项和常系数都不如最高次项中的因子重要，因此g(n)忽略了函数表达式中次数较低的“次要”项，体现了算法复杂性的最本质特征，且忽略常系数。

具体情况与n相关　　如：$T(n) = n^4+100000$

算法五特性

确定性、可行性、输入、输出、有穷性

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

可行性

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

算法五特性

确定性、可行性、输入、输出、有穷性

输入

每个算法都有0个或多个输入，取自特定的对象集合—定义域

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

算法五特性

确定性、可行性、输入、输出、有穷性

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、可行性、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”

计算过程

满足确定性、可行性、输入、输出，但不一定满足有穷性的一组规则称为计算过程

实效性

只有在要求的时间内解决问题才是有意义的

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

---通过“算法分析”


华中科技大学计算机科学与技术学院
School of Computer Science &Technology,HUST

分析结论的证明

目　录

数学归纳法

数学归纳法是用来证明某些与自然数有关的数学命题的一种推理方法

Maurolico

Maurolico, Francesco,1494-1575
意大利数学家、物理学家、天文学家

已知最早的使用数学归纳法的证明出现于 Francesco Maurolico 的 Arithmeticorum libri duo (算术 1575年)。Maurolico 证明了前 n 个奇数的总和是 n^2。

数学归纳法二步走

数学归纳法采用递推策略进行命题证明

基准情形

递归基础、初始情形、平凡情形

证明定理对于某个（某些）小的值是正确的，一般证明命题在n=1（n_0）时命题成立。这是递推的基础

归纳证明

归纳假设和推论的证明

首先假设定理对于直到某个有限数k（或k以内）的所有情况都成立，然后使用这个假设证明定理对于下一个值（通常就是k+1）也成立。如果证明了k+1正确，则完成整个证明

Example：斐波那契数 $F_i<(5/3)^i$

证明：

初始情形

容易验证$F_1 = 1 <5/3$、$F_2=2<(5/3)^2$，

所以，初始情形成立。

归纳假设

设定理对于 i = 1,2,……k 都成立，现证明定理对于i=k+1时也成立，
即$F_{k+1}<(5/3)^{k+1}$

Example：斐波那契数 $F_i<(5/3)^i$

证明：

归纳假设

设定理对于 i = 1,2,.....k 都成立，现证明 $F_{k+1}<(5/3)^{k+1}$

根据斐波那契数的定义有：$F_{k+1} = F_k + F_{k-1}$

将归纳假设用于等号右边，有：$F_{k+1} < (5/3)^k + (5/3)^{k-1} <(5/3)^{k+1}$

∴ n=k+1时结论成立

∴ 定理得证

结论：

華中科技大學
计算机科学与技术学院
School of Computer Science &Technology，HUST

数学归纳法的两步密切相关，缺一不可。而且证明的关键是n=k+1时命题成立的推证，这是无限递推下去的理论依据，它将判定命题的正确性由特殊推广到一般，使命题的正确性突破了有限而达到无限。

可以参考算子 n和k ， 再思考奇数与偶数的例子n=2k；n=2k+1.

反证法

反证法通过假设定理不成立，然后证明该假设将导致某个已知的性质不成立，从而证明假设是错误的而原始命题是正确的。

双重否定律 $\neg\neg A \Leftrightarrow A$

Example：证明存在无穷多个素数

反证法：

假设定理不成立。则存在最大的素数P_k。令$P_1,P_2,\ldots\ldots P_k$是依次排列的所有素数

令： $N= P_1P_2\ldots\ldots P_k+1$　　显然N是比P_k大的数

因为每个整数，或者是素数，或者是素数的乘积，而N均不能$P_1,P_2,\ldots\ldots P_k$整除

因为用$P_1,P_2,\ldots P_k$中的任何一个除N总有余数1，所以N是素数。

而$N>P_k$，故与P_k是最大的素数的假设相矛盾。

∴ 假设不成立，原定理得证。

反例法

反例法举一组具体的数据(算例)，带入定理给出的表达式，证明该数据计算出来的结果与预想结果不符，从而证明定理的结论有错。

特别的应用：证明定理不成立最直接的方法就是举出一个反例。

Example：斐波那契数$F_k \le k^2$是否成立

举反例：

令 k=11

则：$F_{11} = 144 > 11^2$,

所以原命题不成立