傅灵丽 刘恩海 刘金河 主编
吕华 师硕 丁建军 副主编

实用数据结构教程

清华大学出版社

实用数据结构教程

傅灵丽 刘恩海 刘金河 主编
吕华 师硕 丁建军 副主编

清华大学出版社
北京

内 容 简 介

数据结构是计算机以及其他与计算机技术关系密切专业必修的核心课程。本书系统地介绍了各种基本类型的数据结构及其算法实现。本书中对典型算法有详尽的实例描述和算法分析。全书采用C语言作为数据结构和算法的描述语言。

本书是数据结构的入门书籍，结构严谨，条理清晰，按照线性数据结构、层次数据结构和网状数据结构的顺序，由易到难地介绍主要抽象数据类型及其应用，最后介绍各种查找和排序方法。抽象的数据结构原理与算法实现紧密结合的写作特点使读者能够快速而卓有成效地掌握数据结构原理和经典算法，以加深读者对数据结构和算法的理解，从而提高编程能力。

本书可作为计算机类专业或信息类相关专业的高等学校教材，也可作为从事计算机工程与应用工作的技术人员的自学参考书。

图书在版编目（CIP）数据

实用数据结构教程/傅灵丽等主编．—北京：清华大学出版社，2011.10

ISBN 978-7-302-23764-8

Ⅰ．①实…　Ⅱ．①傅…　Ⅲ．①数据结构－高等学校－教材　Ⅳ．①TP311.12

中国版本图书馆CIP数据核字(2010)第168140号

责任编辑：汪汉友

责任校对：梁　毅

责任印制：李红英

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦A座

http://www.tup.com.cn　　**邮　　编**：100084

社　总　机：010-62770175　　**邮　购**：010-62786544

投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn

质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 刷 者：北京富博印刷有限公司

装 订 者：北京市密云县京文制本装订厂

经　　销：全国新华书店

开　　本：185×260　　**印　张**：13.75　　**字　数**：340千字

版　　次：2011年10月第1版　　**印　次**：2011年10月第1次印刷

印　　数：1～4000

定　　价：24.00元

产品编号：039882-01

前　言

数据结构是计算机专业教学的核心课程之一，数据结构课程不但为计算机程序设计提供了方法性的理论指导，还是其后续课程学习的重要基础。本书是为“数据结构”课程编写的教材，其内容选取符合教学大纲要求，并兼顾学科的广度和深度，适用面广。

数据结构课程对学生编程能力的训练包括两方面的内容：一个方面是训练学生理解各种基本数据结构的概念，并能理解和编写出各种典型的算法；另一个方面是训练学生应用各种典型算法进行具体应用问题的程序设计，这包括程序中变量设计、函数中参数设计、程序书写格式等方面的训练。全书用C语言描述算法，完全能兼顾两个方面的训练，这完全符合高等工科院校强调对学生实际动手能力的训练的要求。

本书共分10章，覆盖了数据结构课程的各部分内容。第1章综述数据、数据结构和抽象数据类型等基本概念；第2章至第7章从抽象数据类型的角度，分别讨论线性表、栈、队列、串、数组、广义表、树和二叉树以及图等基本类型的数据结构及其应用；第8章和第9章分别讨论查找和排序，除了介绍各种实现方法之外，并着重从时间上进行定性或定量的分析和比较；第10章介绍常用的文件结构。

本书在内容组织上力求丰富充实，结合实际；在语言描述上力求深入浅出、简洁明了。为便于学习和理解，书中列举了大量例题和综合应用题，每章都配有适量习题，并且每个章节中有知识点重难点提示和总结。

全书由傅灵丽、刘恩海、刘金河担任主编，并负责全书的总体策划与统稿、定稿工作。参加编写的人员及分工如下：第1章～第4章由傅灵丽、刘金河编写，第6章、第7章由师硕、刘恩海、李娟编写，第5章、第8章由吕华、丁建军编写，第9章、第10章由晏俊秋、冯志鸣、陈德生编写。张霞、代俊秋、张平、马长生、刘金旺等老师参加了本书大纲的讨论及部分编写工作，并对书中的程序进行了调试。

本书可作为计算机类专业的高等学校教材，也可以作为信息类相关专业的选修教材。本书文字通俗，简明易懂，便于自学，也可供从事计算机应用等工作的科技人员参考。

清华大学出版社的编校人员为本书的出版付出了辛勤的劳动并提出了宝贵的建议，在此表示诚挚的谢意。

在本书编写过程中，参考了大量文献资料，在此向这些文献资料的作者深表感谢。由于时间仓促，书中难免有不当和欠妥之处，敬请各位专家、读者不吝批评指正。

编　者

2010年7月

目　录

第1章 概 论

计算机中如何有效地组织和处理数据是计算机科学的基本研究内容，也是继续深入学习后续课程的基础。本章对数据结构课程中的基本概念进行了概括性的介绍，主要内容有数据的逻辑结构和存储结构、抽象数据类型的表示与实现、算法和算法设计的要求、算法的效率的度量。这些内容将贯穿于整个数据结构课程学习过程中，难点是算法的效率分析方法。

1.1 数据结构基本概念

1. 数据(data)

数据是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被程序处理的符号的总称。数据是计算机程序加工的“原料”。对计算机科学而言，数据的含义极为广泛，如图像、声音等都可以通过编码而归之于数据的范畴。

2. 数据元素(data element)

数据元素是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一个数据元素可由若干个数据项组成。数据项是数据的不可分割的最小单位。

例如，在表1.1所示的学生信息登记表中，学号、姓名、性别、年龄、籍贯是数据项，这些数据项构成了学生信息的一个数据元素。

表1.1 学生信息登记表

学 号	姓名	性别	年龄	籍贯	学 号	姓名	性别	年龄	籍贯
2009251001	李芳	女	19	山东	2009251004	吕华	女	18	天津
2009251002	张磊	男	19	河北	2009251005	陈强	男	19	湖北
2009251003	王海滨	男	18	北京					

在数据结构中，数据元素和数据项的描述都采用某种程序设计语言来表示，本书采用C语言。表1.1中学生信息的数据元素用C语言结构体描述定义如下：

```
struct Student
{
    long number;
    char name[10];
    char sex[2];
    int age;
    char add[6];
}
```

3. 数据对象(data object)

数据对象是由类型相同的数据元素组成的集合。数据对象是数据的一个子集。例如，有正整数组成的数据对象 $D_1=\{1,2,3,\cdots\}$，由 26 个英文字母组成的数据对象 $D_2=\{A,B,C,\cdots,Z\}$，其中 D_1 是无穷集，D_2 是有穷集。

4. 数据结构(data structure)

数据结构是指相互之间存在的一种或多种特定关系的数据元素的集合。从表 1.1 可以看到，数据元素不是孤立的，它们之间存在着某种关系，这种数据元素间的关系称为结构。

数据结构由某一数据对象及该对象中所有数据元素间的关系组成，记为：

```
Data_Structure= (D,S)
```

其中，D 是数据对象的有限集，S 是 D 上关系的有限集。

例如，在计算机科学中，复数是一种数据结构，可以做如下的定义：

```
Complex= (C,R)
```

其中，C 是含两个实数的集合$\{c_1,c_2\}$；$R=\{P\}$，P 是定义在集合 C 上的一种关系$\{<c_1,c_2>\}$，有序偶$<c_1,c_2>$表示 c_1 是复数的实部，c_2 是复数的虚部。

根据数据元素之间关系的不同特性，可以将数据结构划分为 4 种类型，如图 1.1 所示。

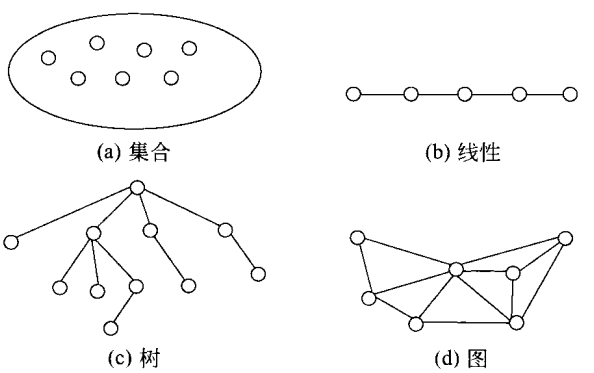

图 1.1　数据结构类型

(1) 集合：结构中的数据元素之间除了“同属于一个集合”的关系外，别无其他关系。

(2) 线性：结构中的数据元素之间存在一对一的关系，即相邻数据元素之间具有“前驱”和“后继”的关系。

(3) 树：结构中数据元素之间存在一对多的关系。

(4) 图(网)：结构中数据元素间存在多对多的关系。

由于“集合”是数据元素之间关系极为松散的一种结构，因此也可以用其他结构来表示。

5. 数据逻辑结构(logic structure)

上述数据结构的定义仅是对操作对象的一种数学描述，换句话说，是从操作对象抽象出来的模型。结构定义中关系描述的是数据元素之间的逻辑关系，因此又称为数据的逻辑结构。

数据的逻辑结构分为线性结构和非线性结构，图 1.1(b)是典型的线性结构，图 1.1(c)和图 1.1(d)是典型的非线性结构。

数据的逻辑结构不能描述计算机是如何操作数据的。研究数据结构的目的是为了在计算机中实现对它的操作，因此必须研究如何在计算机中表示数据结构，即数据的存储结构。

6. 数据存储结构(storage structure)

数据结构在计算机中的表示(或称映像)称为数据的存储结构，又称为物理结构。

数据结构在计算机中有两种不同的表示方法：顺序表示和非顺序表示，由此得出两种不同的存储结构：顺序存储结构和链式存储结构。

1) 顺序存储结构

顺序存储结构是把数据元素存储在一块连续地址空间的内存中，其特点是逻辑上相邻的数据元素在物理上也相邻，数据间的逻辑关系表现在元素的存储位置关系上。当采用高级语言表示时，实现顺序存储结构的方法是用数组。图 1.2 是包含数据元素 $a_0,a_1,\cdots,a_{n-1}$ 的线性结构的顺序存储结构示意图。其中 $0,1,\cdots,n-1$ 为存储数据元素数组 $a_0,a_1,\cdots,a_{n-1}$ 的下标，显然数据的逻辑位序与其在数组中物理位序是一致的。

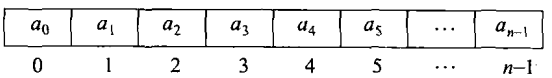

图 1.2 顺序存储结构

2) 链式存储结构

链式存储结构需要在每一个数据元素中增加一个存放地址的指针，用此指针来表示数据元素之间的逻辑关系。因此链式存储结构是使用指针把相互直接关联的结点(即直接前驱结点或直接后继结点)链接起来，其特点是逻辑上相邻的数据元素在物理上(即内存存储位置上)不一定相邻，数据间的逻辑关系表现在结点的链接关系上。图 1.3 是包含数据元素 $a_0,\cdots,a_{n-1}$ 的线性结构的链式存储结构示意图。其中，上一个结点到下一个结点的箭头表示上一个结点的指针域中保存的下一个结点在内存中的存储地址。head 是指向第一个结点的指针。

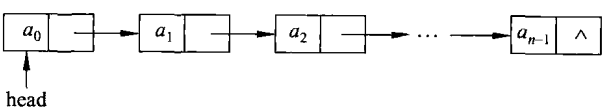

图 1.3 链式存储结构

在 C 语言中，顺序存储结构用一维数组来描述，而链式存储结构利用指针来描述。而在实际程序设计过程中，针对一种数据的逻辑结构，到底选择哪种存储结构会影响到具体算法的设计。也就是说，在设计具体算法之前，必须先确定存储结构。

1.2 抽象数据类型

数据之间的关系，在现实生活中千变万化，使用抽象化的方式成为计算机科学的基础处理方法。以抽象化的概念而言，比如，一节节车厢、排队买饭的学生都可以抽象成线性结构，

企业的组织结构、家族的家谱、字典的目录都可以抽象成树状结构，城市中错综复杂的交通网络、计算机网络都可以抽象成网状结构。数据结构就是使用这种抽象概念来描述各种程序设计时使用的结构。而数据类型是一个与数据结构密切相关的概念。

数据类型(data type)是在一种程序设计语言中，变量所具有的数据种类。是指一个值的集合和定义在这个值集上的一组操作的总称。如整数类型，不仅指所能表示的整数数值的集合，而且指能对这个整数类型进行的加(+)、减(-)、乘(*)除(\)和求模(%)操作。

抽象数据类型(abstract data type，ADT)是指一个数学模型以及定义在此数学模型上的一组操作。抽象数据类型可用以下三元组表示：(D,S,P)。其中，D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作。ADT 的定义为：

```
ADT 抽象数据类型名{
数据对象：(数据对象的定义)
数据关系：(数据关系的定义)
基本操作：(基本操作的定义)
}ADT 抽象数据类型名
```

一旦定义了一个抽象数据类型及具体实现，程序设计中就可以像使用基本数据类型那样，十分方便地使用抽象数据类型。抽象数据类型的设计者根据这些描述给出操作的具体实现，抽象数据类型的使用者依据这些描述使用抽象数据类型。

例如线性表这样的抽象数据类型，其数学模型是数据元素的集合，该集合内的元素有这样的关系：除第一个和最后一个外，每个元素有唯一的前驱和唯一的后继。可以有这样一些操作：插入一个元素、删除一个元素等。

抽象数据类型分为以下几类：

(1) 原子类型，值不可分解，如 int；

(2) 固定聚合类型，值由确定数目的成分按某种结构组成，如复数；

(3) 可变聚合类型，值的成分数目不确定如学生基本情况。

抽象数据类型可以更容易描述现实世界，例如用线性表描述学生成绩表，用树或图描述遗传关系。使用它的人可以只关心它的逻辑特征，不需要了解它的存储方式。定义它的人同样不必关心它如何存储。

软件设计采用模块化方法，抽象数据类型(如表、堆栈、队列、串、数组、树、二叉树、图等)就是构造大型软件的最基本模块。用已设计好的抽象数据类型就可以安全、快速、方便地设计功能复杂的大型软件。数据结构课程主要讨论表、堆栈、队列、串、数组、树、二叉树、图等抽象数据类型的基本功能和设计方法，通过数据结构课程对常用的抽象数据类型设计方法的讨论，学生可以掌握软件的模块化设计的基本方法。

1.3 算法和算法分析

1.3.1 算法

生活或工程中的问题(problem)是一个需要完成的任务。算法(algorithm)是描述求解问题方法的操作步骤的集合。

算法要用某种语言来描述。描述算法的语言主要有3种形式：文字、伪码和程序设计语言。文字形式是用中文或英文来描述算法。伪码形式是用一种仿程设计语言的语言(因这样的描述语言不是真正的程序设计语言，所以称作伪码)来描述算法。用程序设计语言描述算法最直接，可直接调试运行。本书采用C语言描述算法。

算法具有以下5个特性。

(1) 有穷性。一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。

(2) 确定性。算法中每一条指令必须有确切的含义，不存在二义性，且算法只有一个入口和一个出口。

(3) 可行性。一个算法是可行的。即算法描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

(4) 输入。一个算法有零个或多个输入，这些输入取自于某个特定的对象集合。

(5) 输出。一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

注意：*根据实际问题的需要算法中可以没有输入，但一定有输出。要想设计出好的算法，设计者必须熟悉程序设计语言并且严格按照算法设计规范去设计。*

1.3.2 算法的设计目标

算法设计应满足以下要求。

(1) 正确性。算法应确切地满足具体问题的需求，这是算法设计的基本目标。

(2) 可读性。算法的可读性有利于人们对算法的理解，这既有利于程序的调试和维护，也有利于算法的交流和移植。算法的可读性主要体现在两方面：一是变量名、常量名、函数名等命名要见名知意；二是要有足够的注释。

算法注释是一个完整算法不可缺少的部分。设计者应该在算法开始之处以注释的形式写明如下内容：算法的功能、函数参数表中各参数的含义和输入输出的属性、算法中引用了哪些全局变量或外部定义的变量以及它们的作用和入口初值、参数和引用的全局或外部变量的条件限制等。

另外在无论多难懂的语句和关键的语句(段)，只要在其之后加以注释可以大大提高程序的可读性。

(3) 健壮性。通常情况下，我们无法清楚地确定全部的合法输入。但却可以很容易界定非法输入和操作的异常情况(比如，内存不足等)。因此，算法中对于非法输入及异常情况必须提供适当的处理。

除非特别需要，尽量不要在底层算法中使用exit函数。最好使用函数返回值返回算法的执行状态(正确、错误及错误代码等)，这样便于调用者根据函数的执行状态在更高的调用层上对错误进行处理。

(4) 高时间效率。算法的时间效率指算法的执行时间。对于同一个问题如果有多个算法可选择，应尽可能选择执行时间短的算法。执行时间短的算法称作高时间效率的算法。

(5) 高空间效率。算法在执行时一般要求额外的内存空间。对于同一个问题如果有多个算法可供选择，应尽可能选择内存要求低的算法。内存要求低的算法称作高空间效率的算法。

算法的高时间效率和高空间效率通常是矛盾的。比如有些问题，若算法采用了较多的内存空间，而只需较少次循环就能实现，因而时间效率会提高；若算法采用了较少的内存空间，算法需要较多次循环才能实现，因而时间效率会降低。在目前计算机内存价格快速下降的趋势下，当算法设计的时间效率目标和空间效率目标发生矛盾时，算法的时间效率目标应首先被考虑。

1.3.3 算法效率的度量

同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。算法分析的目的在于选择合适算法和改进算法。一个算法的评价主要从时间复杂度和空间复杂度来考虑。

1. 时间复杂度

算法的执行时间需通过根据该算法编制的程序在计算机上运行所消耗的时间来度量。度量一个程序在计算机上的执行时间通常有如下两种方法。

(1) 事后统计方法。计算机内部均设有计时功能，有的甚至可精确到毫秒级，不同算法的程序可通过一组或若干组相同的统计数据以分辨优劣。但这种方法也有缺陷：一是必须运行依据算法编制的程序，而这通常是比较麻烦的；二是所得时间的统计量依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。因此人们常常采用另一种事前分析估算的方法。

(2) 事前分析方法。用数学方法直接对算法的效率进行分析。因为这种分析方法是在计算机实际运行依据该算法编制的程序前进行的，所以称为事前分析方法。

根据算法编制的程序在计算机上运行时所消耗的时间与下列因素有关：

① 书写算法的程序设计语言；

② 编译产生的机器语言代码质量；

③ 机器执行指令的速度；

④ 问题的规模，例如求 100 以内还是 1000 以内的素数；

⑤ 依据的算法选用何种策略。

显然，同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行时，效率均不相同。这表明使用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为一个特定算法“运行工作量”的大小，只依赖于问题的规模(通常用整数量 n 表示)，或者说，它是问题规模的函数。

一个算法是由控制结构(顺序、分支和循环)和基本操作(指固有数据类型的操作)构成的，则算法时间取决于两者的综合效果。为了便于比较同一问题的不同算法，通常的做法是，从算法中选取一种对于所研究的问题(或算法类型)来说是基本操作的原操作，以该基本操作重复执行的次数作为算法的时间量度。例如，变量赋值、两个整数进行四则运算、比较两个整数的大小等都是基本操作；而 n 个数累加就不是基本操作，因为它和输入规模有关。

设 n 为求解问题的规模，基本操作执行次数总和称为语句频度，记作 $f(n)$，时间复杂度记作 $T(n)$。一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数，算法的时间量度记作：

$$T(n) = O(f(n))$$

它表示随问题规模 n 的增大，算法执行时间的增长率和 $f(n)$ 的增长率相同，称 $O(f(n))$ 为算法的渐进时间复杂度，简称时间复杂度。

渐近时间复杂度的大 O 表示法有如下性质，可以用于求解算法时间复杂度函数的最简形：

(1) $O(c\times f(n))=O(f(n))$，其中 c 为常数，即可以忽略 $O()$ 表达式中的常数因子。

(2) $O(f(n))+O(g(n))=O(\max(f(n),g(n)))$，即顺序执行的两段程序的总的代价是它们之中开销较大的部分。

综合考虑以上性质，可以看出在计算算法时间复杂度的渐近增长率时，可以忽略所有的常数因子和低次项。

(3) $O(f(n))\times O(g(n))=O(f(n)\times g(n))$，此性质可用于分析程序中的嵌套循环，即如果外层循环的代价是 $f(n)$，内层循环的代价是 $g(n)$，则总代价是二者的乘积。

当 $f(n)$ 的表达式不同时，按数量级递增排列，常见的时间复杂度有常数阶 $O(1)$、对数阶 $O(\log 2^n)$、线性阶 $O(n)$、线性对数阶 $O(n\log 2^n)$、平方阶 $O(n^2)$、立方阶 $O(n^3)$，指数阶 $O(2^n)$。随着问题规模 n 的不断增大，上述时间复杂度不断增大，算法的执行效率越低。

例 1.1 分析下列程序段的时间复杂度。

```
void main()
{
    int n;
    scanf("%d",&n);              //1次
    n++;                         //1次
    printf("%d",n);              //1次
}
```

解：语句频度为 $f(n)=f(1)=3$。时间复杂度为 $T(n)=O(f(n))=O(3)=O(1)$。

说明该算法的时间复杂度为输入规模 n 的常量阶，即无论输入规模是多大，这个算法的时间代价是不变的。

例 1.2 分析下列程序段的时间复杂度。

```
void sum(int n)
{
    int i,m;
    m=0;                         //1次
    for(i=1;i<=n;i++)            //n次
        m+=i*i*i;                //4n次
    printf("%d",m);              //1次
}
```

解：语句频度为 $f(n)=1+n+4n+1=5n+2$。时间复杂度为 $T(n)=O(f(n))=O(5n+2)=O(n)$。

说明该算法的时间复杂度为输入规模 n 的线性阶。

例 1.3 已知数组 a 和 b，分析如下程序段实现两个 n 阶矩阵相乘算法的时间复杂度。

```
for(i=0;i<n;i++)
```

```
for(j=0;j<n;j++)
{
    c[i][j]=0;                                    //n² 次
    for(k=0;k<n;k++)
        c[i][j]=c[i][j]+a[i][k]*b[k][j];          //n³ 次
}
```

解：设基本语句的执行次数为 $f(n)$，有

$$f(n) = c_1 n^2 + c_2 n^3$$

因为 $f(n)=c_1 n^2+c_2 n^3 \leqslant cn^3$，其中 c_1、c_2、c 可为任意常数，忽略所有的常数因子和低次项，所以该算法的时间复杂度为 $T(n)=O(n^3)$。

例 1.4 分析冒泡排序算法的时间复杂度。注：数组 a 中的 n 个整数类型的数据元素 $a[0]\sim a[n-1]$从小到大进行排序。

```
void BubbleSort(int a[],int n)
{
    int i,j,flag=1;
    int temp;
    for(i=1;i<n && flag==1;i++)
    {
        flag=0;
        for(j=0;j<n-i;j++)
        {
            if(a[j]>a[j+1]
            {
                flag=1;
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
```

解：这个算法的时间复杂度随待排序数据的不同而不同。当某次排序过程中没有任何两个数组元素交换位置时，则表明数组元素已排序完毕，此时算法将因标记 flag=0 不满足循环条件而结束。但是，在最坏的情况下，每次排序过程中都至少有两个数组元素交换位置，因此，最坏情况下该算法的时间复杂度分析如下：

设基本语句的执行次数为 $f(n)$，最坏情况下有

$$f(n)n \approx n + 4 * n^2/2$$

因 $f(n)\approx n+2n^2 \leqslant cn^2$，其中 c 为常数，所以该算法的最坏时间复杂度为 $T(n)=O(n^2)$。

2. 最佳、最差和平均时间复杂度

对于某些算法而言，即使输入规模相同，如果输入的数据不同，其时间复杂度也不同。下面通过实例来说明算法时间复杂度的最佳、最差和平均情况。