“十二五”普通高等教育规划教材

数据结构

SHUJU JIEGOU

主　编　刘遵仁

“互联网+”立体化教材

广益教育

北京邮电大学出版社
www.buptpress.com

“十三五”普通高等教育规划教材

数据结构

主　编　刘遵仁
副主编　汤　雷

广益教育“九斗”APP 操作说明

北京邮电大学出版社
·北京·

内容简介

本书是为“数据结构”课程编写的教材，也可作为学习数据结构及其算法的C语言程序设计的参考教材。本书是为计算机类本科生编写的教材，对于计算机大专类或非计算机专业的学生而言，本书也特别适合，因为对“数据结构”课程中涉及的主要内容，本书均给出了可运行的源代码，有利于加深理解。

全书共分10章，分别介绍了各种数据结构的基本概念、逻辑结构和存储结构，讨论了在各种数据结构上的基本操作。书中内容包括线性表、堆栈和队列、数组、字符串、广义表、树与二叉树、图、查找、排序。算法用ANSI C语言给出，简明易懂，具有较好的可读性。

图书在版编目(CIP)数据

数据结构/刘遵仁主编. -- 北京：北京邮电大学出版社，2018.8
ISBN 978-7-5635-5436-2

Ⅰ.①数… Ⅱ.①刘… Ⅲ.①数据结构 Ⅳ.①TP311.12

中国版本图书馆CIP数据核字(2018)第087687号

书　　名	数据结构
主　　编	刘遵仁
责任编辑	向　蕾
出版发行	北京邮电大学出版社
社　　址	北京市海淀区西土城路10号(100876)
电话传真	010-82333010　62282185(发行部)　010-82333009　62283578(传真)
网　　址	www.buptpress3.com
电子信箱	ctrd@buptpress.com
经　　销	各地新华书店
印　　刷	中煤(北京)印务有限公司
开　　本	787 mm×1 092 mm　1/16
印　　张	15.5
字　　数	385千字
版　　次	2018年8月第1版　2018年8月第1次印刷

ISBN 978-7-5635-5436-2　　定价：42.00元

preface 前言

随着计算机科学的迅速发展，“数据结构”课程已经越来越受到人们的重视，被认为是计算机领域的一门十分重要的基础学科。通过对该课程的学习，能使学生学会如何分析和研究计算机加工的数据对象的特性，学会数据的组织方法，以便选择合适的数据逻辑结构和存储结构，以及相应的操作，把实际中的问题转化为在计算机内部的表示和处理。以上就是“数据结构”课程所要研究并加以解决的问题。

本书较详细地介绍了数据的逻辑结构和存储结构。从逻辑结构上看，数据有三种基本结构：线性结构、树结构和图结构；从存储结构上看，数据有两种基本结构：顺序结构和链式结构。同一种逻辑结构可以采用不同的存储结构。程序设计语言与数据结构之间存在着密切的联系：程序设计语言为数据结构的描述提供了很好的手段；数据结构为程序设计语言类型系统的发展与完善奠定了基础。因此，本书着重强调了数据结构的概念及其在程序设计中的应用，以便提高学生的编程能力。

本书内容取材适中，概念明确，由浅入深，前后衔接，在算法分析上力求描述简洁。全书共分 10 章。第 1 章阐述了数据结构的基本概念；第 2 章到第 6 章主要讨论了线性结构，其中包括线性表、堆栈和队列、数组、字符串及广义表；第 7 章和第 8 章讨论了非线性结构，重点讨论了树和二叉树、图的基本概念及其应用；第 9 章和第 10 章分别为查找和排序，重点讨论了几种查找方法和排序方法。

本书中的算法都是以 ANSI C 编写的，对于学习本教材的学生而言，最好具有 C 语言的基础。C 语言的编译器，建议使用开源的 Dev-C++版本，因为目前国内很多计算机软件大赛都采用该编译器。

本书由刘遵仁担任主编，汤雷担任副主编，其他参与编写的还有梁咏梅。本书由许曰滨教授、沈金虎教授共同审定，在审定过程中，他们认真审阅了全部书稿，提出了宝贵意见，在此表示衷心的感谢。

由于编者学识浅薄，加之时间仓促，错误和不足之处在所难免，敬请专家和读者批评指正。

编　者

2018 年 3 月

contents 目 录

第1章 绪 论

在电子计算机发展的初期阶段，人们使用其的主要目的是处理数值性问题，解决人们用手工或机械计算机难以胜任的数值计算。当时所设计的操作对象都比较简单，不外乎是整型、实型和布尔型数据。以此为对象的程序设计称为数值型程序设计，对应的软件或程序称为数学软件。随着计算机使用领域的扩大和深入，解决非数值性问题越来越引起人们的关注。例如在银行业、电信业、工商企业等领域的管理信息系统，以及支持多媒体的资料查询、模式识别，网络与通讯，图形化用户界面技术等。解决诸如此类问题使用的数学工具已经不再是分析数学及其计算方法，而是更多地用到离散数学和计算机的有关知识，所涉及的数据也越来越复杂。

非数值计算问题的突出特点是：数据元素之间所具有的特定联系已不能用分析数学的方程式来简单地描述。这些就是数据结构形成和发展的背景。

本章重点介绍：

- 基本术语
- 算法
- 时间复杂度
- 算法的表述

1.1 基本术语

数据(data)：描述客观事物的数字、字符以及一切能够输入到计算机中，并且能够被计算机程序处理的符号的集合。即数据是计算机能够加工处理的对象或信息。数据的含义十分广泛，在不同的场合下具有不同的含义。例如，一本书、一篇文章、一张图表、一段声音、一幅图像等是数据，一个单词、一个算术表达式、一个数值、一个字符等也都是数据。

数据元素(data element)：数据的基本单位，即数据集合中的一个个客体。在程序中数据元素通常是作为一个整体来处理的。例如，对于一个数据库文件来说，每条记录就是它的数据

元素；对于一个字符串来说，每个字符就是它的数据元素；对于一个数组来说，每个单元就是它的数据元素。

一个数据元素可以由若干个数据项（数据项是数据的最小单位）组成。例如，对于学生简历来说，每一个记录表示一个学生的信息，它是由若干个数据项构成的，如表 1-1 所示。

表 1-1 学生简历

学号	姓名	性别	爱好	身高	备注
20000301	王 刚	男	足球	1.80	
20000302	张 芳	女	音乐	1.65	学习委员
20000303	南晓飞	男	小提琴	1.78	
20000304	刘锴	男	表演	1.83	三好学生
…	…	…	…	…	…

由表可知，一个数据元素由 6 个数据项组成，而每个数据项是不可再分解的最小的数据，它们分别描述了客体（学生）某一方面的特征。

数据对象（data object）是指具有相同特性的数据元素的集合，是数据这个集合的一个子集。例如，自然数的数据对象是集合{1,2,3,…}，而由 26 个英文字母字符组成的数据对象则是集合{A,B,C,D,…,Z}。

数据处理（data processing）是指对数据进行查找、插入、删除、排序、计算、输入、输出等操作过程，处理的目的是获得有用的信息。

简单地说，数据结构（data structure）是指数据元素之间的联系。因为数据是客观世界事物及其活动的描述，而任何事物及其活动都不是孤立存在的，彼此之间必然存在着某种联系。由于这种联系是内在的，或根据人们的需要定义的，所以被看作“逻辑”上的联系，又把数据结构称作数据的逻辑结构或逻辑关系。数据结构在计算机存储器上的存储表示称为数据的物理结构或存储结构。由于存储表示的方法有顺序、链式两种，所以，一种数据结构可用一种或两种物理结构实现。

为了确切地描述数据结构，通常采用二元组表示：

$$\mathrm{DS}=(D,R)$$

DS 是一种数据结构，它由数据元素的有限集合 D 和 D 上二元关系的有限集合 R 组成。其中

$$D=\{d_i \mid 1\leqslant i\leqslant n, n\geqslant 0\}$$
$$R=\{r_j \mid 1\leqslant j\leqslant m, m\geqslant 1\}$$

d_i 表示第 i 个数据元素，n 为 DS 中数据元素的个数，特别地，若 $n=0$，则 D 是一个空集，故 DS 也就无结构而言，或者说它具有任何结构；r_j 表示第 j 个二元关系（简称关系），m 为 D 上关系的个数。

在本书所讨论的数据结构中，只讨论 $m=1$ 的情况，即 $R=\{r\}$。

D 上的一个关系 r 是有序偶的集合。对于 r 中的任何一个有序偶对$<x,y>(x,y\in D)$，把 x 叫作有序偶对的第一个元素，把 y 叫作有序偶对的第二个元素。有时，把有序偶对的第一个元素称为第二个元素的直接前驱，简称前驱，称第二个元素为第一个元素的直接后继，简

称后继。

由于同一种逻辑结构可以映射成不同的存储结构，如顺序存储结构和链式存储结构(或称为非顺序存储结构)，因此，数据的存储结构一定要正确地反映出数据元素之间的逻辑关系。

为了具体说明这一点，下面再举一个例子。

例如，一个具有20个记录的反映学生家庭地址情况的数据文件，其记录在文件中的先后顺序是按学生学号从小到大排列的，如表1-2所示。

表1-2 学生家庭地址表

学号	姓名	城市	街道	门牌号
19980801	邱卫国	北京	王府井大街	20
19980802	刘 锴	上海	淮海中路	31
19980803	南 健	青岛	香港中路	8
…	…	…	…	…
19980820	卞 军	广州	中山路	19

该数据文件中记录之间的逻辑关系是一个线性关系(因此也称该数据文件为一个线性表)。对应这个逻辑结构，在计算机内可以有两种物理结构，即顺序存储结构和链式存储结构。前者用内存中一块地址连续的存储空间依次存放该文件的20个记录，记录物理上的先后关系映射了记录逻辑上的先后关系。也就是说，逻辑上相邻的两个数据元素，其存储位置也相邻。后者则用20个称为链表节点的存储空间分别存放这20个记录，每个节点物理地址上可以不连续，但是通过链指针映射记录之间的逻辑关系。如图1-1所示。

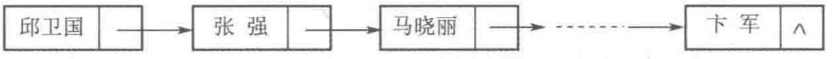

图1-1 线性链表示意图

由此不难想到，在实现某种操作之前，首先需要选择合适的存储结构，而同一种操作在不同的存储结构中实现的方法不同，有的则完全依赖于所选择的存储结构。可见，数据的逻辑结构与存储结构是紧密相连的两个方面。

综上所述，数据结构所要研究的主要内容，可以简单地归纳为以下三个方面。

①研究数据元素之间固有的客观联系(逻辑结构)。

②研究数据在计算机内部的存储方法(存储结构)。

③研究如何在数据的各种结构(逻辑上和物理上)实施有效的操作(算法)。

因此，应该说数据结构是一门抽象地研究数据之间结构关系的学科。

1.2 算法的概念

数据结构与算法之间存在着密切的联系。算法的结构设计和选择，在很大程度上依赖于作为其基础的数据结构，即数据结构为算法提供了工具，而算法是应用这些工具来具体解决问题的方案。凡是从事面向过程设计人都会有这样一个体会：程序设计过程中有相当多的时

间花费在构思算法上，一旦有了合适的算法，用具体的程序设计语言来实现（编写程序）并不是一件很困难的事情。瑞士苏黎世大学著名的计算机科学家、Pascal 程序设计语言之父、结构化程序设计首创者、1984 年图灵奖获得者沃斯（Niklaus Wirth）于 1976 年提出了著名的观点：**算法＋数据结构＝程序**。从这个角度上说，要设计出一个好的程序，在很大程度上取决于设计出一个好的算法。

算法（algorithm）是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。即算法是用来解决某个问题的一些指令的集合。由此可以说，程序就是用计算机语言表述的算法，流程图就是图形化的算法，甚至一个公式也可以叫算法。

在计算机领域，一个算法实质上是根据所处理问题的需要，在数据的逻辑结构和物理结构的基础上施加的一种操作。由于数据的逻辑结构与物理结构不是唯一的，在很大程度上可以由设计人员进行选择和设计，因而处理同一个问题的算法也不一定是唯一的。另外，即使具有相同的逻辑结构与物理结构，如果算法的设计思路和技巧不同，设计出来的算法也不会相同。显然，根据数据处理的需要，为处理的数据选择合适的逻辑结构与物理结构，进而设计出比较满意的算法，是学习数据结构这门课程的主要目的。

作为一个完整的算法，应该满足下面五个标准，通常称之为算法的基本性质。

- 确定性：算法中的每一条指令必须有确定的含义，不应该产生二义性。即不允许出现对于每个指令，不同的人有不同的理解。并且，在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。
- 有穷性：一个算法必须在执行有限的步骤之后结束，并且每一步骤也必须在有限的时间之内完成。
- 可行性：算法中描述的每一个操作，都可以通过已经实现了的基本运算执行有限次后完成。
- 输入：一个算法有零个或若干个输入，这些输入都来自于某一特定的对象的集合。
- 输出：一个算法有一个或多个输出。

算法还有一个重要的方面，就是构成这个算法所依据的方法（公式、方案、准则）。有许多问题只要对数据对象进行细致的分析，就能确定处理方法，有的问题则不然。不过，作为寻找设计思路的基本思想方法，对任何算法设计都是有用的。这些方法通常有枚举法、归纳法、递归法等。

算法独立于具体的计算机与具体的程序设计语言。在设计一个算法时，如何选择一种合适的方式来表达算法思想，或者说，有了解决问题的算法思想，如何选择一种合适的语言来描述算法的各个步骤。在计算机发展的初期，人们往往用自然语言来表达自己的算法思想。下面看一个简单例子。

判断正整数 M 和 N 哪个大。若采用自然语言描述解决该问题的各个步骤，可表达为

①M 减 N，将差值赋给临时变量 R。

②判断 R 是否为零？

若 R 等于零，则 M 和 N 一样大；若 R 大于零，则 M 大于 N；若 R 小于零，则 M 小于 N。

类似的简单问题用自然语言表达还是可以的，但很快就会发现，采用自然语言描述算法很不方便，也不直观，更谈不上有良好的可读性，稍微复杂一些的算法就难以表达，甚至无法表达。另外，由于自然语言本身的一些限制，用它描述的算法可能会出现二义性。如果采用流程

图的形式来描述算法(见图 1-2),它比采用自然语言表达直观了一些,但依然没有解决复杂算法的表达,而且移植性也不好。

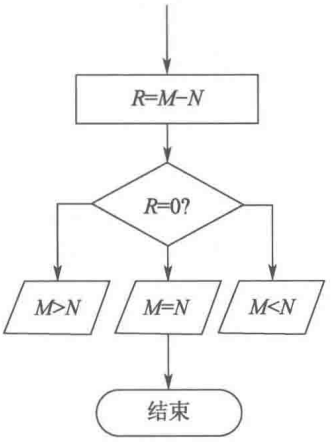

图 1-2 判断正整数 M 和 N 哪个大的流程图

一般而言,如果算法直接采用某种具体的程序设计语言来描述,则将受到具体语言语法细节的限制(如烦琐的变量说明,语句的书写规则等)。通常在进行算法描述时,采用"类 C 语言"作为工具。所谓类 C 语言就是在 ANSI C 语言的基础上所做的取舍,它忽略了 ANSI C 语言中语法规则的一些细节,这样使得描述出的算法清晰、直观便于阅读和分析。这样书写的算法,虽然不能直接在计算机上执行,但稍加修改和补充,就很容易变成计算机所能执行的程序了。

对于上面的问题,用类 C 语言描述的方法为

```
void comp(int M,int N)
{
  R = M-N;
  if(R == 0)
    printf("M = N");
  else if(R > 0)
    printf("M > N");
  else
    printf("M < N");
}
```

1.3 算法描述

为了读者能够验证算法的执行效果,本书采用了 ANSI C 语言作为算法的描述语言。以下作简要说明。

(1)预定义常量

```
#define TRUE 1
#define FALSE 0
```

(2)数据元素类型

约定为 ElemType,在使用该数据类型时进行自行定义。

(3)基本操作的算法

用以下形式的函数描述。

```
函数类型 函数名(函数参数表){
/*算法简单说明 */
语句体;
}
```

(4)赋值语句

```
变量名 = 表达式;
变量名[ ] = 表达式;
```

(5)条件语句

```
条件语句 1 if(表达式) 语句体;
条件语句 2 if(表达式) 语句体;
           else 语句体;
```

(6)循环语句

```
for(赋初值表达式序列; 条件; 修改表达式序列) 语句体;
while(条件) 语句体;
do {
    语句体;
} while(条件);
```

(7)结束语句

- 函数结束语句:

```
return 表达式;
```

- 异常结束语句:

```
exit(异常代码);
```

(8)输入和输出语句

- 输入语句:

```
scanf([格式串],变量 1,变量 2,…,变量 n);
```

- 输出语句:

```
printf([格式串],表达式 1,表达式 2,…,表达式 n);
```

(9)逻辑运算

- 逻辑“与”:对于 A&&B,当 A 的值为 0 时,不再对 B 求值。
- 逻辑“或”:对于 A||B,当 A 的值为非 0 时,不再对 B 求值。

(10)符号和函数

符号⌊x⌋表示不大于 x 的最大整数,如⌊2.7⌋ = 2;符号⌈x⌉表示不小于 x 的最小整数,如⌈3.1⌉ = 4;m % n 表示 m 对 n 取模。

1.4 算法分析

针对同样一个问题,不同的人能够写出不同的算法。对算法进行分析,可以从解决同一个问题的不同算法中,选择出较为合适的一个,也能知道如何对现有算法进行改进,从而设计出更好的算法来。

算法分析的目的在于改进算法的设计。对一个算法进行评价时，首先要看算法是否正确，这是前提条件。算法的正确性是指，当输入一组合法的数据时，算法能够在有限的时间内得出正确的结果；对于不合理的数据输入，也应该能够给出相应的错误提示信息。通常要验证一个算法是否正确，可以通过输入不同的数据进行测试，特别是输入一些极端情况下的数据来进行验证。然而，要从理论上证明一个算法是否正确却不是一件容易的事情。

对算法进行分析时，除了要考虑算法的正确性，还需从以下的三个方面来考察。

①根据该算法编写出的程序，对计算机中运行时间的度量，即所谓的时间复杂度。它是一个算法运行时间的相对度量。

②根据该算法编写出的程序，对占用计算机存储空间的度量，即所谓的空间复杂度。

③其他方面。诸如算法的可读性、可移植性、简单性以及容易测试等。

从理论上讲，一个好的算法既不占用很多的内存存储空间，运行时间又短，并且在其他方面性能也好。然而，时间与空间上的开销往往是一对矛盾，因此，十全十美的算法实际上极少甚至是不存在的。有时候，一个形式上看起来很简单的算法，其对应的程序运行时间要比一个形式上复杂得多的算法对应的程序慢得多；一个运行时间很少的程序占用的存储空间却很大。在具体设计一个算法时，要综合考虑以上诸方面的因素。

1.4.1 时间复杂度

一个程序在计算机中运行时，它所花费的时间与许多种因素都有关系，其中主要有下面几种。

①问题的规模 n。例如，是求 10 个数相加还是求 1 000 个数相加的运算。

②编译程序所产生的机器代码的质量。

③机器执行一条指令的时间长短。

④程序中语句的执行次数。

一般情况下，前三个因素与计算机系统的硬件、软件以及要解决的问题有关，也就是说，不同的计算机系统可能会产生不同的结果，只有第四个因素直接与算法有关。因此，通常的做法是，把算法中语句执行的次数作为算法时间的度量，因为知道了解决同一个问题的两个不同算法的语句执行次数，就可以比较出它们的时间复杂程度。这种把语句执行次数的多少作为算法时间度量的分析方法称为频度统计法。

一条语句的频度(frequency count)就是指该语句被执行的次数。而整个算法的频度是指算法中所有语句的频度之和。

【例 1-1】 对 n 个数累加求和的算法做时间复杂度分析。

算法 A：累加求和

```
int  sum(int A[],int n)
/* A 表示一维数组,n 表示数组的大小 */
{
    (1)s = 0;/*  给累加变量 s 赋初值 */
    (2)for(i = 1;i <=n;i++)
    s = s + A[i];  /* 进行累加求和 */
```

```
    (3)return  s;  /* 返回值 */
}
```

计算机执行这个算法时，第(1)步和第(3)步只需要一次赋值操作，为了分析第(2)步语句的频度，可改写为

```
    i = 1;
first:if i>n then goto last;
    s = s + A[i];
    i = i + 1;
    goto first;
last:return s;
```

把第(2)步分解后的每一条语句的执行次数加起来，就得到了算法中所有语句的频度之和，即为 $4n+2$。整个算法的语句的频度若用 $f(n)$表示，有

$$f(n)=4n+2$$

当 $n\to\infty$时，$f(n)$与 n 成正比。于是，引入一个符号 O(英语单词 Order(数量级)的第一个字母的大写，读“大欧”)，记为

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

表示当 n 足够大时，该程序运行时间的度量与 n 为同一个数量级。或者说，运行时间与 n 成正比。下面给出求 O 的一个严格数学定义。

若 n 足够大，有

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\text{常数}\neq 0$$

则称函数 $f(n)$与 $g(n)$同阶。或者说，$f(n)$与 $g(n)$为同一个数量级，记作

$$f(n)=O(g(n))$$

称上式为算法的时间复杂度(Time Complexity)。n 为问题的规模(大小)的度量。

常见的时间复杂度有常量阶 $O(1)$、对数阶 $O(\log_2 n)$、线性阶 $O(n)$、对数线性阶 $O(n\log_2 n)$、平方阶 $O(n^2)$、指数 $O(2^n)$等，不同数量级时间复杂度的性状如图 1-3 所示。


算法的时间复杂度

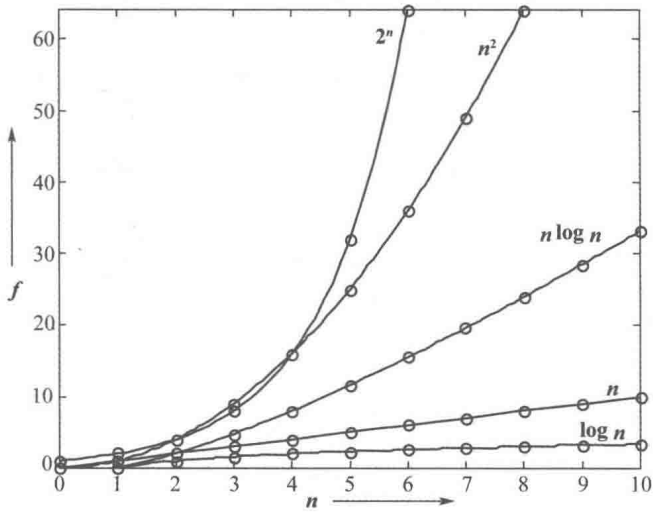

图 1-3 常见函数的增长率

算法的时间复杂度采用数量级的形式表示后，将给求一个算法的时间复杂度带来很大的方便，这时只需要分析影响一个算法运行时间的主要部分（如循环语句）即可，不必对每一步都进行详细的分析。同时，对最主要部分的分析也可简化，一般只要分析清楚循环体内“语句执行次数”即可。例如对于算法 A，只要根据第(2)步的循环次数 n，就可求出其时间复杂度为 $O(n)$。下面再给一个例子来说明如何求时间复杂度。

算法 B：矩阵相加。

```
void matrix(A,B,C,n)
/* A、B、C 分别为 n 阶矩阵，A、B 表示两个加数，C 表示和 */
{
    for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
        C[i][j] = A[i][j]+B[i][j];
}
```

对于上述算法 B，只要弄清楚双重循环内语句的频度值为 n^2，就可求出其时间复杂度为 $O(n^2)$。

表 1-3 给出了各种有代表性的 $g(n)$ 函数的算法在不同 n 值时的运行时间。表中凡是未注明时间的单位为微秒(10^{-6} s)。因算法的实际运行时间随机器而异，所以此表上的时间主要用于相互比较。

表 1-3 算法的运行时间与 $g(n)$ 函数的关系

$g(n)$	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	n^5	2^n	$n!$
$n=20$	4.3	20	86.4	400	8 ms	3.2 s	1.05 s	771 世纪
$n=40$	5.3	40	213	1 600	64 ms	1.7 min	12.7 d	2.59×10^{32} 世纪
$n=60$	5.9	60	354	3 600	216 ms	13 min	366 世纪	2.64×10^{66} 世纪

从表中可以得出两点结论。

①当 $g(n)$ 为对数函数、幂函数或它们的乘积时，算法的运行时间是可以接受的，我们称这些算法为有效的算法；当 $g(n)$ 为指数函数或阶乘函数时，算法的运行时间随着 n 而迅速增长，变得不可接受，我们称这种算法是“坏”的算法或无效的算法。

②随着 n 值的增大，各种 $g(n)$ 函数所对应的运行时间的增长速度大不相同，对数函数的增长速度最慢，线性函数稍快些，往后逐渐增快。因此，当 n 足够大时，各种不同数量级的 $g(n)$ 函数存在着下列关系。

$$O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<\cdots<O(2^n)<O(n!)$$

一个算法的时间复杂度除了与问题的规模 n 有关外，还与输入的具体数据以及数据的输入次序有关，当输入的具体数据、次序不同时，其算法的时间复杂度也可能不同。所以，当计算一个算法的时间复杂度时，还要考虑到具体数据输入时的各种可能情况。

【例 1-2】 从一维数组 A[n]中查找值等于给定值 K 的算法如下。

```
int  locate(A[n],K)
{ /*查找成功时，返回对应的下标；查不到时，返回值-1 */
    (1)i = 0;
```