工业和信息化普通高等教育“十三五”规划教材
21世纪高等教育计算机规划教材

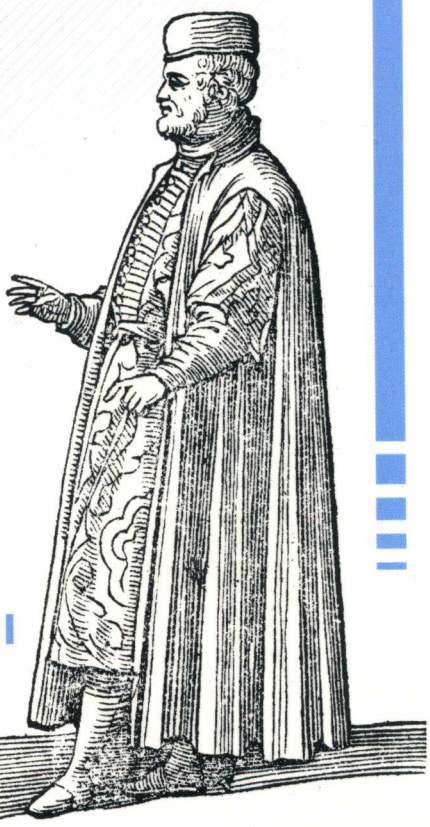
COMPUTER

计算机常用算法与程序设计教程（第2版）

Computer Common Algorithms and Programming Course

杨克昌 主编

- 注重常用算法的选取与组织
- 注重典型案例的精选与提炼
- 注重算法与程序的改进优化



中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

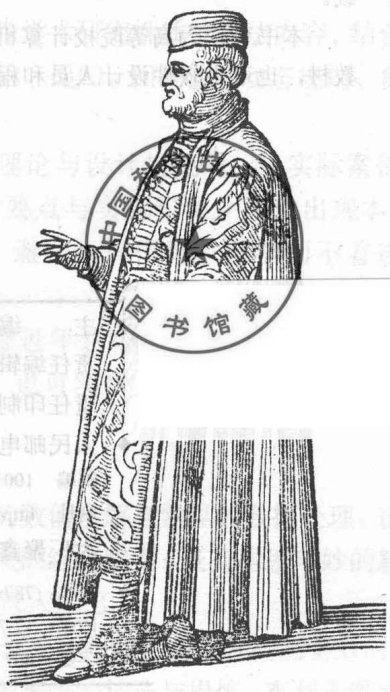
工业和信息化普通高等教育“十三五”规划教材
21世纪高等教育计算机规划教材

COMPUTER

计算机常用算法与程序设计教程（第2版）

Computer Common Algorithms and Programming Course

杨克昌 主编



人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

计算机常用算法与程序设计教程 / 杨克昌主编. -- 2版. -- 北京 : 人民邮电出版社, 2017.8
21世纪高等教育计算机规划教材
ISBN 978-7-115-45591-8

Ⅰ. ①计… Ⅱ. ①杨… Ⅲ. ①电子计算机－算法理论－高等学校－教材②程序设计－高等学校－教材 Ⅳ. ①TP301.6②TP311.1

中国版本图书馆CIP数据核字(2017)第092553号

内 容 提 要

本书遵循“精选案例，深入浅出，面向设计，注重能力培养”的要求，系统讲述枚举、递推、递归、回溯法、动态规划、贪心算法、分支限界法与模拟等常用算法及其应用。精选各算法设计求解的典型案例，从案例提出到算法设计、从程序实现到复杂度分析，环环相扣，融为一体，力求算法理论与实践应用相结合、算法与程序相统一，突出算法在程序设计中的核心地位与引导作用。

书中所有案例给出算法设计要点与完整的C程序代码，并给出程序运行示例（均在Visual C++ 6.0编译通过）与算法分析。为方便教学，每章都附有习题，同时推出与本书配套的课件供教学选用。书中所有源代码、部分习题解答提示与配套课件均可从人邮教育社区（http://www.ryjiaoyu.com）下载。

本书可作为高等院校计算机相关专业“算法设计与分析”和“程序设计基础与应用”等课程的教材，也可供软件设计人员和程序设计爱好者学习参考。

◆ 主　　编　杨克昌
责任编辑　张孟玮
责任印制　陈　犇

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷

◆ 开本：787×1092　1/16
印张：17.75　　2017 年 8 月第 2 版
字数：467 千字　　2017 年 8 月河北第 1 次印刷

定价：49.80 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

第 2 版前言

“计算机算法与程序设计”是计算机科学与技术的核心，是高等院校计算机相关专业的一门重要的专业基础课，课程教学目的是提高学生的算法设计水平，培养通过程序设计解决实际问题的能力。

现有诸多“算法设计与分析”教材通常在算法选取上贪多求全、贪广求深，在内容组织上普遍存在两个问题：算法与数据结构结合多，算法与程序设计脱节；罗列算法多，算法理论与实际应用脱节。由此直接造成学生对算法不感兴趣，不利于学生应用算法与程序设计解决实际问题能力的培养与提高。

为此，我们在编著《计算机常用算法与程序设计教程》（普通高等教育“十一五”国家级规划教材）的基础上进行全面调整与充实，对应用案例进行进一步提炼与优化。

本书遵循“精选案例，深入浅出，面向设计，注重能力培养”的宗旨，在常用算法应用案例的选取与深度的把握上，在算法理论与实际应用的结合上进行精心设计，力图适合高校计算机课程教学目标与知识结构要求。

本书体现以下 4 个特色。

（1）注重常用算法的选取与组织

在算法的选取上去除一些难度大、应用少的带学术研究性质的算法内容，结合高校计算机教学实际与应用需求，选取枚举、递推、递归、回溯法、动态规划、贪心算法、分支限界法与模拟等常用算法。

对精选的各种常用算法，在介绍算法基本理论与设计规范后，从实际案例的解决入手，讲述算法中要求学生掌握的设计要点与实施步骤，避免出现本、专科阶段与研究生阶段的教学内容混杂不分，避免面面俱到、空洞而不着边际的局面。

特别推介第 9 章的“竖式乘除模拟”，它是近年总结推广用于数论高精计算的创新成果，既可以处理整数高精度计算，也可实现一些无理数的指定精度计算。

（2）注重典型案例的精选与提炼

精选适合各常用算法的典型案例，包括经典的数值求解与常见的数据处理。这些案例中既有引导入门的基础问题，也有难度较大的综合案例；既有构思巧妙的新创趣题，也有历史悠久的经典名题，难度适宜，深入浅出。

培养学生的学习兴趣，激发学生的学习热情，不是一两句空洞说教所能奏效的，必须通过一系列有吸引力的实际案例来引导。应用案例的精选与提炼，有利于激发学生学习算法与程序设计的兴趣，有利于学生在计算机实际应用上开阔视野，使之在算法思路的开拓与设计技能的运用上有一个深层次的锻炼与提高。通过实际案例来引导算法设计的逐步深入，实现以典型案例支撑算法，以算法设计指导案例求解的良性循环。

（3）注重算法与程序的有机融合

算法与程序是一个有机的统一体，不应该也不可能将它们对立与分割。本书在材料的组织上克服了罗列算法多、应用算法设计解决实际问题少，算法与程序设计脱节、算法理论与实际应用脱节的问题。在讲述每一种常用算法的基本思路与设计规范基础上，落实到每一个案例求解，从问题背景到算法设计、从程序实现到运行示例、从复杂度分析到变通改进，环环相扣，融为一体。学生看得见、摸得着、学得会、用得上，力求算法理论与实际应用相结合、算法与程序设计相统一，突出算法在解决实际案例中的核心地位与引导作用。

本书采用功能丰富、应用面广、高校学生使用率最高的 C 语言描述算法与编写程序。为使读者使用方便，程序均可在 Visual C++6.0 编译通过。

（4）注重算法与程序的改进优化

本书对一些典型案例应用多种不同的算法设计，编写表现形式与设计风格不同的程序，体现了算法与程序设计的灵活性和多样性。

问题求解的算法与程序设计都不是一成不变的，可以根据问题的具体实际进行多方位多层次的变通。变通出成果，变通长能力。把对案例实施算法改进与程序优化的过程，体现为降低复杂度、提高求解效率的过程，转化为算法设计能力培养与提高的过程，提升为优化意识与创新能力增强的过程。

为方便读者程序设计练习与查阅，附录简介了在 Visual C++6.0 环境下运行 C 程序的方法，并列出 C 语言的常用库函数。同时，与本书配套的课件、书中的源代码与部分习题解答提示均可在人邮教育社区（http://www.ryjiaoyu.com）下载。

本书适合各高校计算机相关专业“算法设计与分析”与“程序设计基础与应用”等课程教学。参考学时为 60～72 学时，建议采用理论教学与上机实践相结合的教学模式，第 1～9 章分别为 6～8 学时（含上机实践，具体由任课教师根据教学实际确定），第 10 章作为自学提高及课程设计的参考素材。

本书由杨克昌教授主编，王岳斌教授、严权峰教授、周小强副教授与甘靖、陶跃进老师参加教材编写与试用。其中，王岳斌编写第 6 章，严权峰编写第 7 章，周小强编写第 3 章，甘靖编写第 4 章，陶跃进编写第 5 章，杨克昌编写其余各章并负责统稿。本书的编写得到湖南理工学院教务处和计算机学院的大力支持，编者在此表示衷心感谢。

编　者

2017 年 2 月于岳阳南湖

目 录

第 1 章 算法与程序设计概述

从计算机存储程序工作原理可知，程序设计在计算机科学与技术中非常重要，而程序设计的关键在于有一个合适的算法。本章介绍算法与程序设计基础知识。

1.1 算法概念与描述

算法（Algorithm）是计算机科学与技术中一个常用的基本概念。本节简要论述算法的定义与算法的描述。

1.1.1 算法概念

在计算机科学与技术中，算法是用于描述一个可用计算机实现的问题求解方法。算法是程序设计的基础，是计算机科学的核心。计算机科学家哈雷尔在《算法学——计算的灵魂》一书中指出："算法不仅是计算机学科的一个分支，它更是计算机科学的核心，而且可以毫不夸张地说，它和绝大多数科学、商业和技术都是相关的。"

在计算机应用的各个领域，技术人员都在使用计算机求解他们各自专业领域的课题，他们需要设计算法，编写程序，开发应用软件，所以学习算法对于越来越多的人来说变得十分必要。我们学习算法的重点就是把人类找到的求解问题的方法、步骤，以过程化、形式化、机械化的形式表示出来，以便让计算机执行，从而解决更多的实际问题。

1. 算法定义

我们首先给出算法的定义。

算法是解决问题的方法或过程，是解决某一问题的运算序列，或者说算法是问题求解过程的运算描述。

在数学和计算机科学之中，算法为一个计算的具体步骤，常用于计算、数据处理和自动推理。

当面临某一问题时，需要找到用计算机解决这个问题的方法与步骤，算法就是解决这个问题的方法与步骤的描述。

2. 算法的三要素

算法由操作、控制结构与数据结构三要素组成。

（1）操作

算术运算：加、减、乘、除等。

关系运算：大于、小于、等于、大于等于、小于等于、不等于等。

逻辑运算：与、或、非等。

其他操作：输入、输出、赋值等。

（2）控制结构

顺序结构：各操作按排列顺序依次执行。

选择结构：由条件是否成立来选择相应操作执行。

循环结构：重复执行某些操作，直到满足指定条件为止。

模块调用：一个模块调用另一个模块（包括自身直接或间接调用的递归结构）。

（3）数据结构

算法的处理对象是数据，数据之间的逻辑关系、数据的存储方式与处理方式就是数据结构。

常见的数据结构有：线性结构，如线性表、数组、堆栈和队列；树形结构，如树、堆；图形结构。

3. 算法的基本特征

一个算法由有限条可完全机械地执行的、有确定结果的指令组成。指令正确地描述了要完成的任务和它们被执行的顺序。

计算机按算法所描述的顺序执行算法的指令能在有限的步骤内终止：或终止于给出问题的解，或终止于指出问题对此输入数据无解。

算法是满足下列特性的指令序列。

（1）确定性

组成算法的每条指令是清晰的、无歧义的。

在算法中不允许有诸如“$x/0$”之类的运算，因为其结果不能确定；也不允许有“x 与 1 或 2 相加”之类的运算，因这两种可能的运算应执行哪一个，并不确定。

（2）可行性

算法中的运算是能够实现的基本运算，每一种运算可在有限的时间内完成。

在算法中两个实数相加是可行的；两个实数相除，例如求 2/3 的值，在没有指明位数时需由无穷个十进制位表示，并不可行。

（3）有穷性

算法中每一条指令的执行次数有限，执行每条指令的时间有限。

如果算法中的循环步长为零，运算进入无限循环，这是不允许的。

（4）算法有零个或多个输入

算法所能接受的数据输入。有些输入数据需要在算法执行过程中输入，有些算法看起来没有输入，实际上输入已被嵌入算法之中。

（5）算法有一个或多个输出

输出一个或多个与输入数据有确定关系的量，是算法对数据进行运算处理的结果。

通常求解一个问题可能会有多种算法可供选择，选择的主要标准是算法的正确性和可靠性，其次是算法所需要的存储空间少和执行时间短等。

4. 算法的重要意义

有人也许会认为：今天计算机运算速度这么快，算法还重要吗？

诚然，计算机的计算能力每年都在飞速增长，价格在不断下降。可日益先进的记录和存储手段使我们需处理的信息量也在快速增长，互联网的信息流量更在爆炸式地增长。在科学研究方面，随着研究手段的进步，数据量更是达到了前所未有的程度。例如在高能物理研究方面，很多实验

每秒都能产生若干个 TB 的数据量，但因为处理能力和存储能力的不足，科学家不得不把绝大部分未经处理的数据舍弃。无论是三维图形、海量数据处理、机器学习、语音识别等，都需要极大的计算量。

算法并不局限于计算机和网络。在网络时代，越来越多的挑战需要靠卓越的算法来解决。如果你把计算机的发展放到数据飞速增长的大环境下考虑，你一定会发现，算法的重要性不是在日益减小，而是在日益增强。

在实际工程中我们遇到许多的高难度计算问题，有的问题在巨型计算机上采用一个劣质的算法来求解可能要几个月的时间，而且很难找到精确解。但采用一个优秀的算法，即使在普通的个人计算机上，可能只需数秒就可以解答。计算机求解一个工程问题的计算速度不仅仅与计算机的设备水平有关，更取决于求解该问题的算法设计水平的高低。世界上许多国家，从大学到研究机关都高度重视对计算机算法的研究，已将提高算法设计水平看作是一个提升国家技术竞争力的战略问题。

对同一个计算问题，不同的人会有不同的计算方法，而不同算法的计算效率、求解精度和对计算资源的需求有很大的差别。

本书具体介绍枚举、递推、递归、回溯、动态规划、贪心算法、分支限界与模拟等常用算法，介绍这些常用算法在实际案例处理与求解中的应用。最后，介绍几个算法综合应用的案例。

1.1.2　算法描述

要使计算机能完成人们预定的工作，首先必须为如何完成这些工作设计一个算法，然后再根据算法编写程序。

一个问题可以设计不同的算法来求解，同一个算法可以采用不同的形式来描述。

算法是问题的程序化解决方案。描述算法可以有多种方式，如自然语言方式、流程图方式、伪代码方式、计算机语言表示方式与表格方式等。

当一个算法使用计算机程序设计语言描述时，就是程序。

本书采用 C 语言与自然语言相结合的方式来描述算法。之所以采用 C 语言来描述算法，是因为 C 语言功能丰富、表达能力强、使用灵活方便、应用面广，它既能描述算法所处理的数据结构，又能描述计算过程，是目前大学阶段学习计算机程序设计的首选语言。

为方便算法描述与程序设计，下面把 C 语言的语法要点作简要概括。

1. 标识符

标识符可由字母、数字和下划线组成，但是必须以字母或下划线开头。一个字母的大小写分别认为是两个不同的字符。

2. 常量

整型常量：十进制常数、八进制常数（以 0 开头的数字序列）、十六进制常数（以 0x 开头的数字序列）。

长整型常数（在数字后加字符 L 或 l）。

实型常量（浮点型常量）：小数形式与指数形式。

字符常量：用单引号（撇号）括起来的一个字符，可以使用转义字符。

字符串常量：用双引号括起来的字符序列。

3. 表达式

（1）算术表达式

整型表达式：参加运算的运算量是整型量，结果也是整型数。

实型表达式：参加运算的运算量是实型量，运算过程中先转换成 double 型，结果为 double 型。

（2）逻辑表达式

用逻辑运算符连接的整型量，结果为一个整数（0 或 1），逻辑表达式可以认为是整型表达式的一种特殊形式。

（3）字位表达式

用位运算符连接的整型量，结果为整数。字位表达式也可以认为是整型表达式的一种特殊形式。

（4）强制类型转换表达式

用“（类型）”运算符使表达式的类型进行强制转换，如（float）a。

（5）逗号表达式（顺序表达式）

形式为：表达式 1，表达式 2，…，表达式 *n*

顺序求出表达式 1，表达式 2，…，表达式 *n* 的值，结果为最后有表达式 *n* 的值。

（6）赋值表达式

将赋值号“=”右侧表达式的值赋给赋值号左边的变量，赋值表达式的值为执行赋值后被赋值的变量的值。注意，赋值号左边必须是变量，而不能是表达式。

（7）条件表达式

形式为：逻辑表达式 ？表达式 1 ：表达式 2

逻辑表达式的值若为非 0（真），则条件表达式的值等于表达式 1 的值；若逻辑表达式的值为 0（假），则条件表达式的值等于表达式 2 的值。

以上各种表达式可以包含有关的运算符，也可以是不包含任何运算符的初等量。例如，常数是算术表达式的最简单的形式。

表达式后加“;”，即为表达式语句。

4. 数据定义

对程序中用到的所有变量都需要进行定义，对数据要定义其数据类型，需要时要指定其存储类别。

（1）数据类型标识符有：

int（整型），short（短整型），long（长整型），unsigned（无符号型），char（字符型），float（单精度实型），double（双精度实型），struct（结构体名），union（共用体名）。

（2）存储类别有：

auto（自动的），static（静态的），register（寄存器的），extern（外部的）。

变量定义形式：存储类别　数据类型　变量表列

如：static float x,y

5. 函数定义

存储类别　数据类型　　<函数名> (形参表列)

　　{ 函数体}

6. 分支结构

（1）单分支：

if(表达式) <语句 1> [else <语句 2>]

功能：如果表达式的值为非 0（真），则执行语句 1；否则（为 0，即假），执行语句 2。所列语句可以是单个语句，也可以是用{}界定的若干个语句。

应用 if 嵌套可实现多分支结构。

（2）多分支：

```
switch(表达式)
   { case 常量表达式 1：<语句 1>
     case 常量表达式 2：<语句 2>
     …
     case 常量表达式 n：<语句 n>
     default：<语句 n+1>
   }
```

功能：取表达式 1 时，执行语句 1；取表达式 2 时，执行语句 2；…；其他所有情形，执行语句 n+1。

其中，case 常量表达式的值必须互不相同。

7. 循环结构

（1）while 循环：

```
while(表达式) <语句>
```

功能：表达式的值为非 0（条件为真），执行指定语句（可以是复合语句）。直至表达式的值为 0（假）时，脱离循环。

特点：先判断，后执行。

（2）Do-while 循环：

```
do  <语句>
while（表达式）;
```

功能：执行指定语句，判断表达式的值非 0（真），再执行语句；直到表达式的值为 0（假）时，脱离循环。

特点：先执行，后判断。

（3）for 循环：

```
for（表达式 1；表达式 2；表达式 3）<语句>
```

功能：解表达式 1；求表达 2 的值：若非 0（真），则执行语句；求表达式 3；再求表达式 2 的值；…；直至表达式 2 的值为 0（假）时，脱离循环。

以上三种循环，若执行到 break 语句，提前终止循环。若执行到 continue，结束本次循环，跳转下一次循环判定。

顺便指出，在不致引起误解的前提下，有时对描述的 C 语句进行适当简写或配合汉字标注，用以简化算法框架描述。

例如，从键盘输入整数 n，按 C 语言的键盘输入函数应写为：

```
scanf("%d",&n);
```

框架描述时可简写为：input(n);

或简写为：输入整数 n;

要输出整数量 $a(1),a(2),\cdots,a(n)$，按 C 语言的输出循环应写为：

```
for(k=1;k<=n;k++)
    printf("%d,",a[k]);
```

框架描述时可简写为：

```
print(a(1) ~ a(n));
```

或简写为：输出 a(1：n)；

例 1-1 对两个整数 $m,n(m>n)$最大公约数的欧几里德算法描述。

求两个整数最大公约数的欧几里德算法有着 2000 余年的历史。欧几里德算法依据的算法理论为如下定理：$\gcd(m,n)=\gcd(n,m \bmod n)$。

（1）m 除以 n 得余数 r；若 $r=0$，则 n 为所求的最大公约数。

（2）若 $r\neq 0$，以 n 为 m，r 为 n，继续（1）。

注意到任意两个整数总存在最大公约数，上述辗转相除过程中余数逐步变小，相除过程总会结束。

欧几里德算法又称为“辗转相除”法，应用 C 语言具体描述如下：

```
// 求最大公约数欧几里德算法描述
main()
{ long  m,n,c,r;
  printf("  请输入整数 m,n(m>n)：");
  scanf("%ld,%ld",&m,&n);                    //  输入两整数
  printf(" (%ld,%ld)=",m,n);
  r=m%n;
  while(r!=0)
    { m=n;n=r;                               //  通过循环实施辗转相除操作
      r=m%n;
    }
  printf("%ld",n);                           //  输出最大公约数
}
```

该算法中有输入，即输入整数 m,n；有操作处理，即通过条件循环实施“辗转相除”；最后有输出，即输出最大公约数 n。

以上案例的算法应用 C 语言（有时适当予以简化）描述，缩减了从算法写成完整 C 程序的距离，比应用其他方法描述更加方便。

1.2 算法复杂性分析

算法复杂性的高低体现运行该算法所需计算机资源的多少。算法的复杂性越高，所需的计算机资源越多；反之，算法的复杂性越低，所需的计算机资源越少。

计算机资源，最重要的是时间资源与空间资源。因此，算法的复杂性有时间复杂性与空间复杂性之分。需要计算机时间资源的量称为时间复杂度，需要计算机空间资源的量称为空间复杂度。时间复杂度与空间复杂度集中反映算法的效率。

算法分析是指对算法的执行时间与所需空间的估算，定量给出运行算法所需的时间数量级与空间数量级。

1.2.1　时间复杂度

算法作为计算机程序设计的基础，在计算机应用领域发挥着举足轻重的作用。一个优秀的算法可以运行在计算速度比较慢的计算机上求解问题，而一个劣质的算法在一台性能很强的计算机上也不一定能满足应用的需求。因此，在计算机程序设计中，算法设计往往处于核心地位。如何去设计一个适合特定应用的算法是众多技术开发人员所关注的焦点。

1. 算法分析的方法

要想充分理解算法并有效地应用算法求解实际案例，关键是对算法的分析。通常我们可以利用实验对比方法、数学方法来分析算法。

实验对比分析很简单，两个算法相互比较，它们都能解决同一问题，在相同环境下，哪个算法的速度更快我们一般就会认为这个算法性能更优。

数学方法能更为细致地分析算法，能在严密的逻辑推理基础上判断算法的优劣。但在完成实际项目过程中，我们很多时候都不能去做这种严密的论证与推断。因此，在算法分析中，我们往往采用能近似表达性能的方法来展示某个算法的性能指标。例如，当参数 n 比较大的时，计算机对 n^2 和 n^2+2n 的响应速度几乎没有什么区别，我们便可直接认为这两者的复杂度均为 n^2。

在分析算法时，隐藏细节的数学表示方法为大写字母“O”记法，它可以帮助我们简化算法复杂度计算的许多细节，提取主要成分，这和遥感图像处理中的主成分分析思想相近。

2. 运算执行频数

一个算法的时间复杂度是指算法运行所需的时间。一个算法的运行时间取决于算法所需执行的语句（运算）的多少。算法的时间复杂度通常用该算法执行的总语句（运算）的数量级决定。

就算法分析而言，一条语句的数量级即执行它的频数，一个算法的数量级是指它所有语句执行频数之和。

例 1-2　试计算下面三个程序段的执行频数。

```
（1）x=x+1;s=s+x;
（2）for(k=1;k<=n;k++)
       { x=x+y;
         y=x+y;
         s=x+y;
       }
（3）for(t=1,k=1;k<=n;k++)
       { t=t*2;
         for(j=1;j<=t;j++)
           s=s+j;
       }
```

解：如果把以上 3 个程序段看成 3 个相应算法的主体，我们来看 3 个算法的执行频数。

在（1）中，2 个语句各执行 1 次，共执行 2 次。

在（2）中，“k=1”执行 1 次；“k<=n”与“k++”各执行 n 次；3 个赋值语句，每个赋值语句各执行 n 次；共执行 $5n+1$ 次。

在（3）中，“t=1”与“k=1”各执行 1 次；“k<=n”与“k++”各执行 n 次；“t=t*2”执行 n 次；“j=1”执行 n 次；“j<=t”，“j++”与内循环的赋值语句“s=s+j”各执行频数为：

$$2+2^2+\cdots+2^n=2(2^n-1)$$

因而(3)总的执行频数为：$6\cdot 2^n+4n-4$。

3. 算法时间复杂度定义

算法的执行频数的数量级直接决定算法的时间复杂度。

定义 对于一个数量级为$f(n)$的算法，如果存在两个正常数c和m，对所有的$n\geqslant m$，有

$$|f(n)|\leqslant c|g(n)|$$

则记作$f(n)=O(g(n))$，称该算法具有$O(g(n))$的运行时间，是指当n足够大时，该算法的实际运行时间不会超过$g(n)$的某个常数倍时间。

显然，以上所列举的（1）与（2），其计算时间即时间复杂度分别为$O(1),O(n)$。

据以上定义，（3）的执行频数为：$6\cdot 2^n+4n-4$，取c=8，对任意正整数n，有

$$6\cdot 2^n+4n-4\leqslant 8\cdot 2^n$$

即得（3）的计算时间为$O(2^n)$，即（3）所代表的算法时间复杂度为$O(2^n)$。

可见前两个所代表的算法是多项式时间算法。最常见的多项式算法时间，其关系概括为(约定log$n$表示以2为底的对数):

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$$

算法（3）所代表的是指数时间算法。以下3种是最常见的指数时间算法，其关系为

$$O(2^n)<O(n!)<O(n^n)$$

随着n的增大，指数时间算法与多项式时间算法在所需的时间上相差非常大，表1-1具体列出了常用函数时间复杂度增长情况。

表1-1　　常用函数时间复杂度增长情况

函数$f(n)$	log$n$	n	$n$log$n$	n^2	n^3	2^n
$n=1$	0	1	0	1	1	2
$n=2$	1	2	2	4	8	4
$n=4$	2	4	8	16	64	16
$n=8$	3	8	24	64	512	256
$n=16$	4	16	64	256	4 096	65 536
$n=32$	5	32	160	1 024	32 768	429 967 296

一般地，当n取值充分大时，在计算机上实现指数算法是不可能的，就是比$O(n\log n)$时间复杂度高的多项式算法运行也很困难。

4. 符号O的运算规则

根据时间复杂度符号O的定义，有

定理1-1 关于时间复杂度符号O有以下运算规则：

$$O(f)+O(g)=O(\max(f,g)) \qquad (1.1)$$

$$O(f)O(g)=O(fg) \qquad (1.2)$$

证明 设$F(n)=O(f)$,根据O定义,存在常数$c1$和正整数$n1$,对所有的$n\geqslant n1$,有$F(n)\leqslant c1f(n)$。同样，设$G(n)=O(g)$，根据O定义，存在常数$c2$和正整数$n2$，对所有的$n\geqslant n2$，有$G(n)\leqslant c2g(n)$。令$c3=\max(c1,c2)$，$n3=\max(n1,n2)$，$h(n)=\max(f,g)$。对所有的$n\geqslant n3$，存在$c3$，有

$$F(n)\leqslant c1f(n)\leqslant c3f(n)\leqslant c3h(n)$$

$$G(n)\leqslant c2g(n)\leqslant c3g(n)\leqslant c3h(n)$$

则 $F(n)+G(n)\leqslant 2c3h(n)$

即　$O(f)+O(g)\leqslant 2c3h(n)=O(h)=O(\max(f,g))$

令 $t(n)=f(n)g(n)$，对所有的 $n\geqslant n3$，有

$$F(n)G(n)\leqslant c1c2t(n)$$

即　$O(f)O(g)\leqslant c1c2t(n)=O(fg)$。

式（1.1）、式（1.2）成立。

定理 1-2　如果 $f(n)=a_m n^m+a_{m-1}n^{m-1}+\cdots+a_1 n+a_0$ 是 n 的 m 次多项式，$a_m>0$，则

$$f(n)=O(n^m) \tag{1.3}$$

证明　当 $n\geqslant 1$ 时，根据符号 O 定义有

$$\begin{aligned} f(n)&=a_m n^m+a_{m-1}n^{m-1}+\cdots+a_1 n+a_0\\ &\leqslant |a_m|n^m+|a_{m-1}|n^{m-1}+\cdots+|a_1|n+|a_0|\\ &\leqslant (|a_m|+|a_{m-1}|+\cdots+|a_1|+|a_0|)n^m \end{aligned}$$

取常数 $c=|a_m|+|a_{m-1}|+\cdots+|a_1|+|a_0|$，根据定义，式（1.3）得证。

例 1-3　估算以下程序段所代表算法的时间复杂度。

```
for(k=1;k<=n;k++)
for(j=1;j<=k;j++)
    { x=k+j;
      s=s+x;
    }
```

解：在估算算法的时间复杂度时，为简单计，以后只考虑内循环语句的执行频数，而不细致计算各循环设计语句及其他语句的执行次数，这样简化处理不影响算法的时间复杂度。

每个赋值语句执行频率为 $1+2+\cdots+n=n(n+1)/2$，该算法的数量级为 $n(n+1)$；取 $c=2$，对任意正整数 n，有

$$n(n+1)\leqslant 2\cdot n^2 \quad\Leftrightarrow\quad n\leqslant n^2$$

即得该程序段的计算时间为 $O(n^2)$，即所代表算法的时间复杂度为 $O(n^2)$。

例 1-4　估算下列程序段所代表算法的时间复杂度。

（1）
```
t=1;m=0;
for(k=1;k<=n;k++)
  { t=t*2;
    for(j=t;j<=n;j++)
      m++;
  }
```

（2）
```
d=0;
for(k=1;k<=n;k++)
for(j=k*k;j<=n;j++)
    d++;
```

解：（1）设 $n=2^x$，则（1）中 m++语句的执行次数为：

$$\begin{aligned} S&=(n+1-2)+(n+1-2^2)+(n+1-2^3)+\cdots+(n+1-2^x)\\ &=x(n+1)-2(2^x-1)\\ &=(x-2)n+x+2 \end{aligned}$$

注意到 $x=\log n$，则当 $n\geqslant 2$ 时有