21世纪

21世纪高等学校计算机学科系列教材

计算机算法设计与分析

王晓东 编著

全国高等学校计算机教育研究会
课程与教材建设委员会推荐出版

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

URL: http://www.phei.com.cn

21世纪高等学校计算机学科系列教材

计算机算法设计与分析

王晓东　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书是21世纪计算机学科系列教材之一。它以算法设计策略为知识单元系统地介绍计算机算法的设计方法和分析技巧。其主要内容包括:算法及算法复杂性基本概念,算法描述,有效算法最常用的设计策略——递归和分治法,动态规划法的设计要点与适用性,贪心算法,回溯法和分支限界法,许多难解问题的高效算法——概率算法,以及NP完全理论和NP难问题的近似解法。书中既涉及传统算法的实例分析,更有算法领域热点研究课题追踪,具有很高的实用价值。

本书可作为高等院校计算机科学与工程专业本科生和研究生学习计算机算法设计的教材,也适合广大工程技术人员和自学者学习参考。

图书在版编目(CIP)数据

计算机算法设计与分析/王晓东编著.-北京:电子工业出版社,2001.1

21世纪高等学校计算机学科系列教材

ISBN 7-5053-6391-3

Ⅰ.计… Ⅱ.王… Ⅲ.①电子计算机-算法设计-高等学校-教材②电子计算机-算法分析-高等学校-教材 Ⅳ.TP301.6

中国版本图书馆CIP数据核字(2000)第80435号

丛 书 名:21世纪高等学校计算机学科系列教材

书　　名:**计算机算法设计与分析**

编 著 者:王晓东

责任编辑:张荣琴　童占梅

排版制作:电子工业出版社计算机排版室

印 刷 者:北京四季青印刷厂

装 订 者:河北省涿州桃园装订厂

出版发行:电子工业出版社　URL:http://www.phei.com.cn

北京市海淀区万寿路173信箱　邮编100036

经　　销:各地新华书店

开　　本:787×1092　1/16　印张:19　字数:486.4千字

版　　次:2001年1月第1版　2001年1月第1次印刷

书　　号:ISBN 7-5053-6391-3/TP·3467

印　　数:5 000册　　定价:24.00元

序　言

这套教材是21世纪计算机学科系列教材。为什么要组织这套教材？根据什么组织这套教材？这些都是在这篇序言中要回答的问题。

计算机学科是一个飞速发展的学科，尤其是近十年来，计算机向高度集成化、网络化和多媒体化发展的速度一日千里。但是，从另一方面来看，目前高等学校的计算机教育，特别是教材建设，远远落后于现实的需要。现在的教材主要是根据《教学计划1993》的要求组织编写的。这个教学计划，在制订过程中主要参照了美国IEEE和ACM《教学计划1991》。

10年来，计算机学科已有了长足发展，这就要求高等学校计算机教育必须跟上形势发展的需要，在课程设置和教材建设上作出相应调整，以适应面向21世纪计算机教育的要求。这是组织这套教材的初衷。

为了组织好这套教材，全国高等学校计算机教育研究会课程与教材建设委员会在天津召开了“全国高等学校计算机学科课程与教材建设研讨会”；在北京召开了“教材编写大纲研讨会”。在这两次会议上，代表们深入地研讨了全国高校计算机专业教学指导委员会和中国计算机学会教育委员会制订的《计算机学科教学计划2000》和美国IEEE和ACM《计算机学科教学计划2001》。这是这套教材参照的主要依据。

IEEE和ACM《计算机学科教学计划2001》是在总结了从《计算机学科教学计划1991》到现在，计算机学科十年来发展的主要成果的基础上诞生的。计划中提出面向21世纪计算机学科应包括14个主科目，其中12个主科目为核心主科。它们是：算法与分析(AL)、体系结构(AR)、离散结构(DS)、计算科学(CN)、图形学、可视化、多媒体(GR)、网络计算(NC)、人机交互(HC)、信息管理(IM)、智能系统(IS)、操作系统(OS)、程序设计基础(PF)、程序设计语言(PL)、软件工程(SE)、社会、道德、法律和专业问题(SP)。其中除CN和GR为非核心主科目外，其他12项均为核心主科目。

将2001教学计划与1991教学计划比较可看出：

在1991计划中，离散结构只作为数学基础提出，而在2001计划中，则作为核心主科目提出，显然，提高了它在计算机学科中的地位；

在1991计划中，未提及网络计算，而在2001计划中，则作为核心主科目提出，以适应网络技术飞速发展的需求；

图形学、可视化与多媒体也是为适应发展要求新增加的内容。

除此之外，2001计划在下述5个方面作了调整：

程序设计语言引论调整为程序设计基础；将人—机通信调整为人机交互；将人工智能与机器人学调整为智能系统；将数据库与信息检索调整为信息管理；将数值与符号计算调整为计算科学。

显然，这些变化使2001计划更具有科学性，也更好地适应了学科发展的需要。

在组织这套教材的过程中，充分考虑了这些变化和调整，在软件和硬件的课程体系、界面划分均做了相应的调整，使整套教材更具有科学性和实用性。

另外，还要说明一点，教材建设既要满足必修课的要求，又要满足限选课和任选课的要求。

因此，教材应按系列组织，反映整个计算机学科的要求，采用大拼盘结构，以适应各校不同的具体教学计划的要求，各校可根据自己的需求进行选拼使用。

这套教材包括：《微机应用基础》、《离散数学》、《电路与电子技术》、《电路与电子技术习题与实验指南》、《数字逻辑与数字系统》、《计算机组成原理》、《微机接口技术》、《计算机体系结构》、《计算机网络》、《计算机网络实验教程》、《通信原理》、《计算机网络管理》、《网络信息系统集成》、《多媒体技术》、《计算机图形学》、《计算机维护技术》、《数据结构》、《计算机算法设计与分析》、《计算机数值分析》、《汇编语言程序设计》、《PASCAL 语言程序设计》、《VB 程序设计》、《C 语言程序设计》、《C++语言程序设计》、《JAVA 语言程序设计》、《操作系统原理》、《UNIX 操作系统原理与应用》、《LINUX 操作系统》、《软件工程》、《数据库系统原理》、《编译原理》、《编译方法》、《人工智能》、《计算机信息安全》、《计算机图形学》、《人机交互》、《计算机伦理学》等。对于《IEEE 和 ACM 教学计划 2001》中提出的 14 个主科目这套系列教材均涵盖，能够满足不同层次院校、不同教学计划的要求。

这套系列教材由全国高等学校计算机教育研究会课程与教材建设委员会主任李大友教授精心策划和组织。编者均为具有丰富教学实践经验的专家和教授。所编教材体系结构严谨、层次清晰、概念准确、论理充分、理论联系实际、深入浅出、通俗易懂。

教材组织过程中，得到了哈尔滨工业大学蒋宗礼教授、西安交通大学董渭清副教授、武汉大学张焕国教授、吉林大学张长海教授、福州大学王晓东教授、太原理工大学余雪丽教授等的大力支持和帮助，在此一并表示衷心感谢。

李大友

2000 年 6 月

前　言

计算机的普及极大地改变了人们的生活。目前，各行业、各领域都广泛采用了计算机信息技术，并由此产生出开发各种应用软件的需求。为了以最少的成本、最快的速度、最好的质量开发出适合各种应用需求的软件，必须遵循软件工程的原则。设计一个高效的程序不仅需要编程小技巧，更需要合理的数据组织和清晰高效的算法，这正是计算机科学领域里数据结构与算法设计所研究的主要内容。一些著名的计算机科学家在有关计算机科学教育的论述中认为，计算机科学是一种创造性思维活动，其教育必须面向设计。计算机算法设计与分析正是一门面向设计，且处于计算机学科核心地位的教育课程。通过对计算机算法系统的学习与研究，掌握算法设计的主要方法，培养对算法的计算复杂性正确分析的能力，为独立设计算法和对算法进行复杂性分析奠定坚实的理论基础，对每一位从事计算机系统结构、系统软件和应用软件研究与开发的科技工作者都是非常重要和必不可少的。为了适应21世纪我国培养计算机各类人才的需要，本课程结合我国高等学校教育工作的现状，追踪国际计算机科学技术的发展水平，更新了教学内容和教学方法，以算法设计策略为知识单元系统地介绍计算机算法的设计方法与分析技巧，以期为计算机学科的学生提供一个广泛扎实的计算机算法知识基础。

全书共分9章，第1章介绍算法的基本概念，并对算法的计算复杂性和算法的描述作了简要阐述。然后围绕算法设计常用的基本设计策略组织了第2章至第9章的内容。

第2章介绍递归与分治策略，它是设计有效算法最常用的策略，也是必须掌握的方法。

第3章是动态规划算法，以具体实例详述动态规划算法的设计思想、适用性以及算法的设计要点。

第4章介绍贪心算法，它也是一种重要的算法设计策略，它与动态规划算法的设计思想有一定的联系，但其效率更高。按贪心算法设计出的许多算法能导致最优解。其中有许多典型问题和典型算法可供学习和使用。

第5章和第6章分别介绍了回溯法和分支限界法。这两章所介绍的算法适合于处理难解问题。其解题的思想各具特色，值得学习和掌握。

第7章介绍了概率算法，对许多难解问题提供了高效的解决途径，是有很高实用价值的算法设计策略。

第8章介绍NP完全性理论。首先介绍了计算模型，确定性和非确定性图灵机，然后进一步深入介绍NP完全性理论。这一章是全书理论性最强的一章，难度较大，适合于高年级本科生或研究生。

第9章介绍了解NP难问题的近似算法，这是当前计算机算法领域的热门研究课题，具有很高的实用价值。

在本书各章的论述中，首先介绍一种算法设计策略的基本思想，然后从解决计算机科学和应用中的实际问题入手，由简到繁地描述几个精典的精巧算法。同时对每个算法所需的时间和空间进行分析，使读者既能学到一些常用的精巧算法，又能通过对算法设计策略的反复应用，牢固掌握这些算法设计的基本策略，以期收到融会贯通之效。在为各种算法设计策略选择用于展示其设计思想与技巧的具体应用问题时，本书有意重复选择某些精典问题，使读者能深

刻地体会到一个问题可以用多种设计策略求解。同时通过对解同一问题的不同算法的比较，使读者更容易体会到每一种具体算法的设计要点。随着本书内容的逐步展开，读者也将进一步感受到综合应用多种设计策略可以更有效地解决问题。

本书采用面向对象的C++语言为算法描述手段，在保持C++优点的同时，尽量使算法的描述简明、清晰。

为了帮助读者加深对知识的理解，各章配有难易适当的习题，以满足不同程度读者练习的需要。

由于作者的知识和写作水平有限，书稿虽几经修改，仍难免有缺点和错误。热忱欢迎同行专家和读者批评指正，使本书在使用中不断得到改进，日臻完善。

在本书的编写过程中，得到了全国高等学校计算机专业教学指导委员会的关心和支持。福州大学"211工程"计算机与信息工程重点学科实验室为本书的写作提供了优良的设备和工作环境。电子工业出版社负责本书编辑出版工作的全体同仁为本书的出版付出了大量辛勤劳动，他们认真细致、一丝不苟的工作精神保证了本书的出版质量。傅清祥教授在百忙之中认真审阅了全书，提出了许多宝贵的改进意见。在此，谨向每一位曾经关心和支持本书编写工作的各方面人士表示衷心的谢意！

作　者

2000年6月

目　录

第1章 算法概述

1.1 算法与程序

对于计算机科学来说，算法(Algorithm)的概念是至关重要的。例如在一个大型软件系统的开发中，设计出有效的算法将起决定性的作用。通俗地讲，算法是指解决问题的一种方法或一个过程。更严格地讲，算法是由若干条指令组成的有穷序列，且满足下述几条性质：

(1) 输入：有零个或多个由外部提供的量作为算法的输入。

(2) 输出：算法产生至少一个量作为输出。

(3) 确定性：组成算法的每条指令是清晰的，无歧义的。

(4) 有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序(Program)与算法不同。程序是算法用某种程序设计语言的具体实现。程序可以不满足算法的性质(4)。例如操作系统，它是一个在无限循环中执行的程序，因而不是一个算法。然而我们可把操作系统的各种任务看成是一些单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

描述算法可以有多种方式，如自然语言方式、表格方式等。在本书中，我们采用C++语言来描述算法。C++语言的优点是类型丰富、语句精炼，具有面向过程和面向对象的双重特点。用C++来描述算法可使整个算法结构紧凑，可读性强。在本书中，有时为了更好地阐明算法的思路，我们还采用C++与自然语言相结合的方式来描述算法。

1.2 算法复杂性分析

一个算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上，所需资源越多，我们就说该算法的复杂性越高；反之，所需资源越少，我们就说该算法的复杂性越低。最重要的计算机资源是时间和空间(即存储器)资源。因此，算法的复杂性有时间复杂性和空间复杂性之分。

不言而喻，对于任意给定的问题，设计出复杂性尽可能低的算法是我们在设计算法时追求的一个重要目标。另一方面，当给定的问题已有多种算法时，选择其中复杂性最低者，是我们在选用算法时遵循的一个重要准则。因此，算法的复杂性分析对算法的设计或选用有着重要的指导意义和实用价值。

算法的复杂性是算法运行所需要的计算机资源的量，需要时间资源的量称为时间复杂性，需要的空间资源的量称为空间复杂性。这个量应该集中反映算法的效率，并从运行该算法的实际计算机中抽象出来。换句话说，这个量应该是只依赖于要解的问题的规模、算法的输入和算法本身的函数。如果分别用 N、I 和 A 表示算法要解的问题的规模、算法的输入和算法本身，而且用 C 表示复杂性，那么，应该有 $C = F(N, I, A)$，其中 $F(N, I, A)$ 是一个由 N、I 和 A 确定的三元函数。如果把时间复杂性和空间复杂性分开，并分别用 T 和 S 来表示，应该有：$T = T(N, I, A)$ 和 $S = S(N, I, A)$。通常，我们让 A 隐含在复杂性函数名当中，因而将 T 和 S 分

别简写为 $T = T(N,I)$ 和 $S = S(N,I)$。

由于时间复杂性与空间复杂性概念雷同，计量方法相似，且空间复杂性分析相对简单些，所以本书将主要讨论时间复杂性。现在的问题是如何将复杂性函数具体化，即对于给定的 N、I 和 A，如何导出 $T(N,I)$ 和 $S(N,I)$ 的数学表达式，来给出计算 $T(N,I)$ 和 $S(N,I)$ 的法则。下面以 $T(N,I)$ 为例，将复杂性函数具体化。

根据 $T(N,I)$ 的概念，它应该是算法在一台抽象的计算机上运行所需要的时间。设此抽象的计算机所提供的元运算有 k 种，它们分别记为 $O_1,O_2,\cdots,O_k$。又设每执行一次这些元运算所需要时间分别为 $t_1,t_2,\cdots,t_k$。对于给定的算法 A，设经统计，用到元运算 O_i 的次数为 e_i，$i = 1,2,\cdots,k$。很清楚，对于每一个 i，$1 \leqslant i \leqslant k$，$e_i$ 是 N 和 I 的函数，即 $e_i = e_i(N,I)$。那么有

$$T(N,I) = \sum_{i=1}^{k} t_i e_i(N,I)$$

其中，$t_i(i = 1,2,\cdots,k)$ 是与 N 和 I 无关的常数。

显然，我们不可能对规模 N 的每一种合法的输入 I 都去统计 $e_i(N,I)$，$i = 1,2,\cdots,k$。因此 $T(N,I)$ 的表达式还要进一步简化，或者说，我们只能在规模为 N 的某些或某类有代表性的合法输入中统计相应的 e_i，$i = 1,2,\cdots,k$，评价其时间复杂性。

本书只考虑三种情况下的时间复杂性，即最坏情况、最好情况和平均情况下的时间复杂性，并分别记为 $T_{\max}(N)$、$T_{\min}(N)$ 和 $T_{\text{avg}}(N)$。在数学上有

$$T_{\max}(N) = \max_{I \in D_N} T(N,I) = \max_{I \in D_N} \sum_{i=1}^{k} t_i e_i(N,I) = \sum_{i=1}^{k} t_i e_i(N,I^*) = T(N,I^*)$$

$$T_{\min}(N) = \min_{I \in D_N} T(N,I) = \min_{I \in D_N} \sum_{i=1}^{k} t_i e_i(N,I) = \sum_{i=1}^{k} t_i e_i(N,\widetilde{I}) = T(N,\widetilde{I})$$

$$T_{\text{avg}}(N) = \sum_{I \in D_N} P(I) T(N,I) = \sum_{I \in D_N} P(I) \sum_{i=1}^{k} t_i e_i(N,I)$$

式中，D_N 是规模为 N 的合法输入的集合；I^* 是 D_N 中一个使 $T(N,I^*)$ 达到 $T_{\max}(N)$ 的合法输入；$\widetilde{I}$ 是 D_N 中一个使 $T(N,\widetilde{I})$ 达到 $T_{\min}(N)$ 的合法输入；而 $P(I)$ 是在算法的应用中出现输入 I 的概率。

以上三种情况下的时间复杂性从不同角度来反映算法的效率，各有其局限性，也各有各的用处。实践表明可操作性最好且最有实际价值的是最坏情况下的时间复杂性。本书对算法的时间复杂性分析的重点将放在这种情形上。

随着经济的发展、社会的进步和科学研究的深入，要求用计算机解决的问题越来越复杂，规模越来越大。对求解这类问题的算法作复杂性分析具有特别重要的意义，因而要特别关注。在此，我们引入复杂性渐近性态的概念。

设 $T(N)$ 是前面所定义的关于算法 A 的复杂性函数。一般说来，当 N 单调增加且趋于 ∞ 时，$T(N)$ 也将单调增加趋于 ∞。对于 $T(N)$，如果存在 $\widetilde{T}(N)$，使得当 $N \to \infty$ 时有 $(T(N) - \widetilde{T}(N))/T(N) \to 0$，那么，我们就说 $\widetilde{T}(N)$ 是 $T(N)$ 当 $N \to \infty$ 时的渐近性态，或叫 $\widetilde{T}(N)$ 为算法 A 当 $N \to \infty$ 的渐近复杂性而与 $T(N)$ 相区别。因为在数学上，$\widetilde{T}(N)$ 是 $T(N)$ 当 $N \to \infty$ 时的渐近表达式，直观上，$\widetilde{T}(N)$ 是 $T(N)$ 中略去低阶项所留下的主项，所以它无疑比 $T(N)$ 来得简单。比如当 $T(N) = 3N^2 + 4N\log N + 7$ 时，$\widetilde{T}(N)$ 的一个答案是 $3N^2$，因为这时有

$$(T(N)-\widetilde{T}(N))/T(N)=\frac{4N\log N+7}{3N^2+4N\log N+7}\to 0,\text{当 } N\to\infty \text{ 时}$$

显然,$3N^2$ 比 $3N^2+4N\log N+7$ 简单得多。

由于当 $N\to\infty$ 时,$T(N)$ 渐近于 $\widetilde{T}(N)$,我们有理由用 $\widetilde{T}N)$ 来替代 $T(N)$ 作为算法 A 在 $N\to\infty$ 时的复杂性的度量。而且由于 $\widetilde{T}(N)$ 明显地比 $T(N)$ 简单,这种替代是对复杂性分析的一种简化。进一步考虑到分析算法的复杂性的目的在于比较求解同一问题的两个不同算法的效率。而当要比较的两个算法的渐近复杂性的阶不相同时,只要能确定出各自的阶,就可以判定哪一个算法的效率高。换句话说,这时的渐近复杂性分析只要关心 $\widetilde{T}(N)$ 的阶就够了,不必关心包含在 $\widetilde{T}(N)$ 中的常数因子。所以,我们常常又对 $\widetilde{T}(N)$ 的分析进一步简化,即假设算法中用到的所有不同的元运算各执行一次所需要的时间都是一个单位时间。

综上所述,我们已经给出了简化算法复杂性分析的方法和步骤,即只要考察当问题的规模充分大时,算法复杂性在渐近意义下的阶。本书的算法分析都将这么做。为此引入以下渐近意义下的记号:O、Ω、θ 和 o。

以下设 $f(N)$ 和 $g(N)$ 是定义在正数集上的正函数。

如果存在正的常数 C 和自然数 N_0,使得当 $N\geqslant N_0$ 时有 $f(N)\leqslant Cg(N)$,则称函数 $f(N)$ 当 N 充分大时上有界,且 $g(N)$ 是它的一个上界,记为 $f(N)=O(g(N))$。这时我们还说 $f(N)$ 的阶不高于 $g(N)$ 的阶。

举几个例子:

(1) 因为对所有的 $N\geqslant 1$ 有 $3N\leqslant 4N$,我们有 $3N=O(N)$;

(2) 因为当 $N\geqslant 1$ 时有 $N+1024\leqslant 1025N$,我们有 $N+1024=O(N)$;

(3) 因为当 $N\geqslant 10$ 时有 $2N^2+11N-10\leqslant 3N^2$,我们有

$$2N^2+11N-10=O(N^2)$$

(4) 因为对所有 $N\geqslant 1$ 有 $N^2\leqslant N^3$,我们有 $N^2=O(N^3)$;

(5) 作为一个反例 $N^3\neq O(N^2)$。因为若不然,则存在正的常数 C 和自然数 N_0,使得当 $N\geqslant N_0$ 有 $N^3\leqslant CN^2$,即 $N\leqslant C$。显然,当取 $N=\max\{N_0,\lfloor C\rfloor+1\}$ 时这个不等式不成立,所以 $N^3\neq O(N^2)$。

按照符号 O 的定义,容易证明它有如下运算性质:

(1) $O(f)+O(g)=O(\max(f,g))$;

(2) $O(f)+O(g)=O(f+g)$;

(3) $O(f)O(g)=O(fg)$;

(4) 如果 $g(N)=O(f(N))$,则 $O(f)+O(g)=O(f)$;

(5) $O(Cf(N))=O(f(N))$,其中 C 是一个正的常数;

(6) $f=O(f)$。

性质(1)的证明:设 $F(N)=O(f)$。根据符号 O 的定义,存在正常数 C_1 和自然数 N_1,使得对所有的 $N\geqslant N_1$,有 $F(N)\leqslant C_1f(N)$。

类似地,设 $G(N)=O(g)$,则存在正的常数 C_2 和自然数 N_2,使得对所有的 $N\geqslant N_2$ 有 $G(N)\leqslant C_2g(N)$。

令 $C_3=\max\{C_1,C_2\}$,$N_3=\max\{N_1,N_2\}$,$h(N)=\max\{f,g\}$。则对所有的 $N\geqslant N_3$,有

$$F(N) \leqslant C_1 f(N) \leqslant C_1 h(N) \leqslant C_3 h(N)$$

类似地,有

$$G(N) \leqslant C_2 f(N) \leqslant C_2 h(N) \leqslant C_3 h(N)$$

因而

$$\begin{aligned} O(f) + O(g) &= F(N) + G(N) \\ &\leqslant C_3 h(N) + C_3 h(N) \\ &= 2C_3 h(N) \\ &= O(h) \\ &= O(\max(f,g)) \end{aligned}$$

其余性质的证明类似,留给读者作为练习。

应该指出,根据符号 O 的定义,用它评估算法的复杂性,得到的只是当规模充分大时的一个上界。这个上界的阶越低则评估就越精确,结果就越有价值。

关于符号 Ω,文献里有两种不同的定义。本书只采用其中的一种,定义如下:如果存在正的常数 C 和自然数 N_0,使得当 $N \geqslant N_0$ 时有 $f(N) \geqslant Cg(N)$,则称函数 $f(N)$ 当 N 充分大时下有界;且 $g(N)$ 是它的一个下界,记为 $f(N) = \Omega(g(N))$。这时我们还说 $f(N)$ 的阶不低于 $g(N)$ 的阶。Ω 的这个定义的优点是与 O 的定义对称,缺点是当 $f(N)$ 对自然数的不同无穷子集有不同的表达式,且有不同的阶时,不能很好地刻画出 $f(N)$ 的下界。比如当

$$f(N) = \begin{cases} 100 & N \text{ 为正偶数} \\ 6N^2 & N \text{ 为正奇数} \end{cases}$$

时,按上述定义,得到 $f(N) = \Omega(1)$,这是一个平凡的下界,对算法分析没有什么价值。然而,考虑到上述定义有与符号 O 定义的对称性,本书还是选用它。

同样地,用 Ω 评估算法的复杂性,得到的只是该复杂性的一个下界。这个下界的阶越高,则评估就越精确,结果就越有价值。再则,这里的 Ω 只对问题的一个算法而言。如果它是对一个问题的所有算法或某类算法而言,即对于一个问题和任意给定的充分大的规模 N,下界在该问题的所有算法或某类算法的复杂性中取,那么它将更有意义。这时得到的相应下界,我们称之为问题的下界或某类算法的下界。它常常与符号 O 配合以证明某问题的一个特定算法是该问题的最优算法或该问题的某算法类中的最优算法。

我们定义 $f(N) = \theta(g,(N))$ 当且仅当 $f(N) = O(g(N))$ 且 $f(N) = \Omega(g(N))$。这时,我们说 $f(N)$ 与 $g(N)$ 同阶。

最后,我们来看符号 o 的定义。如果对于任意给定的 $\varepsilon > 0$,都存在正整数 N_0,使得当 $N \geqslant N_0$ 时有 $f(N)/g(N) < \varepsilon$,则称函数 $f(N)$ 当 N 充分大时的阶比 $g(N)$ 低,记为 $f(N) = o(g(N))$。

例如:$4N\log N + 7 = o(3N^2 + 4N\log N + 7)$。

习题一

1-1 求下列函数的渐近表达式:

$3n^2 + 10n$; $n^2/10 + 2^n$; $21 + 1/n$; $\log n^3$; $10\log 3^n$。

1-2 试论 $O(1)$ 和 $O(2)$ 的区别。

1-3 画出下列表达式的函数图像，并说明各表达式当 n 在什么范围内取值时效率最高。

$$4n^2,\ \log n,\ 3^n,\ 20n,\ 2,\ n^{2/3}$$

1-4 按照渐近阶从低到高的顺序排列以下表达式：$4n^2, \log n, 3^n, 20n, 2, n^{2/3}$。又 $n!$ 应该排在哪一位？

1-5 (1) 假设某算法在输入规模为 n 时的计算时间为 $T(n) = 3 \times 2^n$。在某台计算机上实现并完成该算法的时间为 t 秒。现有另一台计算机，其运行速度为第一台的 64 倍，那么在这台新机器上用同一算法在 t 秒内能解输入规模为多大的问题？

(2) 若上述算法的计算时间改进为 $T(n) = n^2$，其余条件不变，则在新机器上用 t 秒时间能解输入规模为多大的问题？

(3) 若上述算法的计算时间进一步改进为 $T(n) = 8$，其余条件不变，那么我们在新机器上用 t 秒时间能解输入规模为多大的问题？

1-6 硬件厂商 XYZ 公司宣称他们最新研制的微处理器运行速度为其竞争对手 ABC 公司同类产品的 100 倍。对于计算复杂性分别为 n, n^2, n^3 和 $n!$ 的各算法，若用 ABC 公司的计算机在 1 小时内能解输入规模为 n 的问题，那么用 XYZ 公司的计算机在 1 小时内分别能解输入规模为多大的问题？

1-7 对于下列各组函数 $f(n)$ 和 $g(n)$，确定 $f(n) = O(g(n))$ 或 $f(n) = \Omega(g(n))$ 或 $f(n) = \theta(g(n))$，并简述理由。

(1) $f(n) = \log n^2$; $g(n) = \log n + 5$

(2) $f(n) = \log n^2$; $g(n) = \sqrt{n}$

(3) $f(n) = n$; $g(n) = \log^2 n$

(4) $f(n) = n\log n + n$; $g(n) = \log n$

(5) $f(n) = 10$; $g(n) = \log 10$

(6) $f(n) = \log^2 n$; $g(n) = \log n$

(7) $f(n) = 2^n$; $g(n) = 100n^2$

(8) $f(n) = 2^n$; $g(n) = 3^n$

1-8 证明：$n! = \theta(n^n)$。

1-9 下面的算法段用于确定 n 的初始值。试分析该算法段所需计算时间的上界和下界。

```
while(n > 1)
  if(odd(n))
n = 3 * n + 1;
  else
    n = n/2;
```

1-10 证明：如果一个算法在平均情况下的计算时间复杂性为 $\theta(f(n))$，则该算法在最坏情况下所需的计算时间为 $\Omega(f(n))$。

第2章　递归与分治策略

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，解题所需的计算时间往往也越少，从而也较容易处理。例如，对于 n 个元素的排序问题，当 $n=1$ 时，不需任何计算。$n=2$ 时，只要作一次比较即可排好序。$n=3$ 时只要作两次比较即可……而当 n 较大时，问题就不那么容易处理了。要想直接解决一个较大的问题，有时是相当困难的。分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。如果原问题可分割成 k 个子问题，$1<k\leqslant n$，且这些子问题都可解，并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易求出其解。这样，就自然导致递归算法的产生。分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

2.1　递归的概念

一个直接或间接地调用自身的算法称为递归算法。一个使用函数自身给出定义的函数称为递归函数。在计算机算法设计与分析中，使用递归技术往往使函数的定义和算法的描述简洁且易于理解。有些数据结构如二叉树等，由于其本身固有的递归特性，特别适合用递归的形式来描述。还有一些问题，虽然其本身并没有明显的递归结构，但用递归技术来求解使设计出的算法简洁易懂且易于分析。下面我们来看几个实例。

例2-1　阶乘函数

阶乘函数可递归地定义为：

$$n!=\begin{cases}1 & n=0\\ n(n-1)! & n>0\end{cases}$$

阶乘函数的自变量 n 的定义域是非负整数。递归式的第一式给出了这个函数的一个初始值，是非递归定义的。每个递归函数都必须有非递归定义的初始值，否则，递归函数就无法计算。递归式的第二式是用较小自变量的函数值来表示较大自变量的函数值的方式来定义 n 的阶乘。定义式的左右两边都引用了阶乘记号，是一个递归定义式，可递归地计算如下：

```
int Factorial(int n)
  {
    if (n = = 0) return 1;
    return n * Factorial(n - 1);
  }
```

例2-2　Fibonacci 数列

无穷数列 1,1,2,3,5,8,13,21,34,55,…，称为 Fibonacci 数列。它可以递归地定义为：