深入浅出 系列规划教材

国家级精品课程配套教材

深入浅出

ShenRu QianChu ShuJu JieGou

数据结构

翁惠玉 俞 勇 编著

Weng Huiyu Yu Yong

清华大学出版社

数据结构

翁惠玉　俞　勇　编著

清华大学出版社
北京

内 容 简 介

数据结构是计算机专业最基础、最重要的课程之一，也是全国硕士研究生入学考试计算机专业的必考科目。

本书严格按照计算机考研大纲的内容和次序来组织。每一章都包括相关知识点的介绍和总结，以及大量的习题解答。习题中包含2009—2014年的所有考研习题，并给出了深入浅出的解答。除了考研题目之外，本书还选用了大量的习题，包含的题型有概念题、选择题和程序设计题。这些习题可以很好地帮助读者理解数据结构的基本知识以及灵活应用数据结构。

本书可作为参加计算机及相关专业硕士研究生考前复习的参考教材，也可以作为各高等院校计算机专业或其他相关专业“数据结构”课程的教材。

图书在版编目(CIP)数据

深入浅出数据结构/翁惠玉，俞勇编著. —北京：清华大学出版社，2014
深入浅出系列规划教材
ISBN 978-7-302-36891-5

Ⅰ. ①深…　Ⅱ. ①翁…　②俞…　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(2014)第131362号

责任编辑：白立军　薛　阳
封面设计：傅瑞学
责任校对：李建庄
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954

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：185mm×260mm　　**印　　张**：17　　**字　　数**：426千字
版　　次：2014年11月第1版　　**印　　次**：2014年11月第1次印刷
印　　数：1～2000
定　　价：29.50元

产品编号：053847-01

《深入浅出数据结构》前言

数据结构是计算机专业最基础、最重要的课程之一，也是全国硕士研究生入学考试计算机专业的必考科目。

本书以作者多年的数据结构课程教学经验为基础，在分析研究了2009—2014年的考题的基础之上编写而成。在内容的选取和安排上严格依据计算机专业考研大纲，对大纲要求的内容进行了深入浅出的阐述；同时提供了丰富的例题，其中包含历年计算机专业考研的真题，并对这些真题进行了分析和解答。

根据考研大纲，本书共分为7章。第1章绪论，对数据结构进行了全面的介绍，包括数据结构的研究内容、算法的时间复杂度等概念，帮助读者对数据结构有一个整体的了解。第2章线性表，第3章栈、队列和数组，第4章树与二叉树，第5章图，第6章查找，第7章排序，每一章都先给出了核心内容的介绍，然后是习题与解答。书中的习题包括三大类型：简答题、选择题和程序设计题。由于本书强调的是数据结构的原理及实现思想，因而采用类C++语言描述数据结构的存储及算法实现，没有按照面向对象的思想把数据结构封装成类。每章后面的程序设计题中的所有程序都用C++语言实现，并在Visual C++ 6.0平台下调试通过。

本书可作为参加计算机及相关专业硕士研究生考前复习的参考教材，也可以作为各高等院校计算机专业或其他相关专业“数据结构”课程的教材，或作为从事计算机工程的科技人员的参考资料。

由于作者水平有限，本书难免存在不足之处，敬请读者批评指正。

FOREWORD

《深入浅出数据结构》目 录

第1章 绪 论

学习数据结构首先要了解数据结构研究的对象和内容。数据结构研究的对象是非数值计算，它在抽象的层面上研究数据元素之间的关系及关系的存储和处理。本章的重点是时间复杂度和空间复杂度的概念和计算方法。计算某个程序段的时间复杂度是考研选择题中的一个重要考点。

1.1 算法与数据结构

数据结构是计算机专业最重要的专业课之一，也是考研必考的科目。

在学习数据结构之前，各位读者想必已学过了程序设计。在程序设计的学习中，我们知道设计一个程序的难点在于设计如何存储待处理的数据以及数据处理的算法。数据结构针对非数值计算的特点，从抽象的层面上研究了非数值计算中数据元素之间的关系、关系的存储及关系的操作。一个数据结构就是解决具有某种关系的数据元素的存储和处理方法。

掌握了数据结构以后，当要解决一个问题时，首先要分析被处理的数据元素之间是什么关系，它需要完成哪些操作，然后选择一个合适的数据结构来处理数据。而不需要自己想办法解决数据的存储和处理方法了。

1.1.1 数据的逻辑结构

一个数据结构是由一组同类的数据元素依据某种联系组织起来的。数据元素间的逻辑关系的描述称为数据的**逻辑结构**。不管应用如何变化，从抽象层面上看，数据的逻辑结构有下列4种。

1. 集合结构

数据元素间的次序是任意的。元素之间除了“属于同一集合”的联系外没有其他的关系。例如，某次聚会中的所有的人员，公共汽车上的所有乘客，存放在仓库中的产品。

2. 线性结构

数据元素之间构成一个有序序列。其中，第一个元素只有后继没有前驱，最后一个元素只有前驱没有后继。除了第一个和最后一个元素外，其余元素都有一个前驱和一个后继。例如，水泊梁山上的一百零八条好汉，他们形成了一个集合，但他们之间有一个次序，宋江排第一，卢俊义排第二……

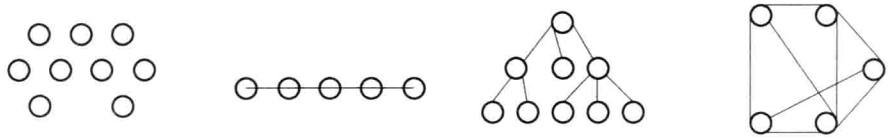
3. 树状结构

除了一个特殊的根元素外，每个元素有且仅有一个前驱，后继数目不限。根元素没有前驱。树状结构表示的是一种层次关系。例如，一个家族就可以表示为树状结构。这家的老祖宗就是树根，老祖宗的儿子就是老祖宗的后继。每个人可以有多个儿子，因此后继数目不限。但每个人只能有一个父亲，因此只有一个前驱。老祖宗的儿子们又可以有他们的儿子，这样就形成了一棵树。老祖宗的父亲无历史记载，因此老祖宗就没有前驱。

4. 图形结构

图是最一般的逻辑结构，图中的每个元素的前驱和后继数目都不限。例如，在一个计算机网络中，各个网络设备之间是由线路连接起来。如果把连接看成是网络设备之间的关系，那么一个计算机网络的拓扑结构就形成了一个图状结构，因为每台网络设备都可以和多台其他设备相连接，向多台设备发送信息，也可以接收多台设备发来的信息。

这4种结构如图1-1所示。有时，也把线性结构以外的其他三种结构称为非线性结构。



(a) 集合结构　(b) 线性结构　(c) 树状结构　(d) 图状结构

图1-1　数据的逻辑结构

一些表面上很不相同的数据可以有相同的逻辑结构。如一个计算机网络可以用一个图形结构来表示，每个数据元素是一个网络设备，数据元素之间的关系是设备间的物理介质。如果两台设备间有一条物理媒体连接起来，则两台设备间有关系。如果用先导课程和后继课程作为课程之间的关系，那么一个课程管理系统中的课程也是一个图状的结构。一门课程可以有若干门先导课程，也可以有若干门后继课程。因此，**逻辑结构是数据组织的本质。只要解决了这个本质问题，就可以把解决方案用到各种应用中去，而不必分别对每个问题进行研究。**

1.1.2　数据结构的运算

每种逻辑结构都有一定的处理要求，这些处理要求称为数据结构的**操作**或**运算**。数据结构最常见的运算有以下几种。

(1) 初始化运算(init)：创建一个空的数据结构。

(2) 清除运算(clear)：删除数据结构中的所有数据元素。

(3) 插入运算(insert)：在数据结构指定的位置上插入一个新数据元素。

(4) 删除运算(remove)：将数据结构中的某个数据元素删去。

(5) 搜索运算(search)：在数据结构中搜索满足特定条件的数据元素。

(6) 更新运算(update)：修改数据结构中的某个数据元素的值。

(7) 访问运算(visit)：访问数据结构中的某个数据元素。

(8) 遍历运算(traverse)：按照某种次序访问数据结构中的每一数据元素，使每个数据元素恰好被访问一次。

除了这些运算之外，每种数据结构还可以包含一些特定的运算，如树状结构中，需要找某个数据元素的儿子，或找某个数据元素的父亲的操作；线性结构中，需要找某个元素的直接前驱或直接后继的操作。

数据元素之间的逻辑关系和数据结构的运算是数据结构不可分割的两个方面。一个数据结构就是针对某一个逻辑结构讨论数据的存储以及运算的实现，通常被称为**存储实现**和**运算实现**。

1.2 存储实现

存储实现的基本目标是建立数据的机内表示，它包括两个部分：数据元素的存储和数据元素之间的关系的存储。有时为了方便运算的实现，还可能会增加一些辅助信息的存储。数据的机内表示称为数据的**物理结构**。按照上述思路，一个物理结构由以下三个部分组成。

(1) 存储结点，每个存储结点存放一个数据元素。

(2) 数据元素之间的关系的存储，也就是逻辑结构的机内表示。

(3) 附加信息，便于运算实现而设置的一些“哑结点”，如链表中的头结点。

其中，前两个部分是每个存储实现都必须具备的，第三部分则是根据运算实现的需要而决定是否设置。由于数据元素本身比较简单，因此存储结点的组织也比较简单，可能是一个程序设计语言的基本数据类型，也可能是一个记录类型或用户定义的类类型。因此，物理结构主要讨论的是数据元素之间的关系的存储。由于每个数据元素被表示为一个存储结点，所以逻辑结构就由存储结点之间的关联方式间接地表示。通常存储结点之间的关联有以下 4 种实现方式。

1. 顺序实现

所有的存储结点存放在一块连续的存储区域中，结点之间的逻辑关系可以通过结点的存储位置来体现。在高级语言中，一块连续的存储空间通常可以用一个数组来表示。因此，顺序实现通常用一个数据元素类型的数组来存储。

2. 链接实现

存储结点可以分散地存放在存储器的不同位置，结点之间的关系通过一个指针显式地指出。因此，在链接存储中，每个存储结点包含两个部分：数据元素部分和指针部分。数据元素部分保存数据元素的值，指针部分保存一组指针，每个指针指向一个与本结点有逻辑关系的结点。

3. 哈希存储方式

是专用于集合结构的数据存放方式。在哈希存储中，各个结点均匀地分布在一块连续的存储区域中，用一个哈希函数将数据元素和存储位置关联起来。

4. 索引存储方式

所有的存储结点按照生成的次序连续存放在存储器中。另外设置一个索引区域存储结点之间的关系。

这些基本的存储方式以及它们的组合可以实现数据的灵活存储。

1.3 算法分析

一般而言，对同一个问题可以设计出不同的解决方法。因此，一个运算也可以有不同的实现，即可以设计出实现该运算的不同的算法。这样就产生了如何评价这些算法的问题。通过这种评价，设计人员可以区分不同实现的相对的优劣，从而选择一个较好的算法。通常算法的评价从两个角度展开：时间性能和空间性能。时空性能分别称为**时间复杂度**和**空间复杂度**。

1.3.1 时间复杂度

衡量一个算法的好坏不能看它在某一台计算机上运行时间的长短，运行时间与机器性能、处理的数据量以及被处理数据的分布有关。时间性能分析摒弃了不同机器之间的差异，仅分析运行时间函数，即运算量与问题规模之间的关系函数。

算法的运行时间函数可能是一个很复杂的函数，解决同一问题的不同算法会有不同的运行时间函数，如何比较这些函数并从中选取出一个好的算法呢？这是一项很困难的工作。例如，对于两个算法 f_1 和 f_2，当处理的问题规模在 100 以下时，f_1 的运行时间函数值比 f_2 小，而在问题规模大于 100 以后，f_2 的运行时间函数值比 f_1 小，那么到底是 f_1 好还是 f_2 好呢？

事实上，由于计算机运行速度很快，当问题规模很小的时候，无论运行时间是什么函数，运行时间都可忍受，时间性能主要考虑的是问题规模很大的时候运行时间随问题规模的变化规律。也就是对问题规模比较小的情况忽略不计，只考虑当问题规模大于一定值以后的情况。例如，对于上述的 f_1 和 f_2 算法，认为 f_2 的时间性能优于 f_1 的时间性能。

同理，在问题规模很大时的运行时间函数也可以有各种各样的形式，那么如何判断一个函数的值总比另一个函数小呢？这是一些很复杂的数学问题。在算法分析中，将这个问题进行了简化，不考虑具体的运行时间函数，只考虑运行时间函数的数量级，然后比较这些函数的数量级。这种方法称为**渐进表示法**。最常用的渐进表示法是大 O 表示法。

定义 1.1 如果存在两个正常数 c 和 N_0，使得当 $N \geqslant N_0$ 时有 $T(N) \leqslant cF(N)$，则记为 $T(N)=O(F(N))$。

例 1.1 设 $T(n)=(n+1)^2$。那么，取 $n_0=1$ 及 $c=4$ 时，$T(n)\leqslant cn^2$ 成立。所以，$T(n)=O(n^2)$。

例 1.2 设 $T(n)=3n^3+2n^2$。选 $n_0=0$ 及 $c=5$；$T(n)\leqslant cn^3$。所以，$T(n)=O(n^3)$。

大 O 表示法给出了算法在问题规模 n 达到一定程度后运行时间增长率的上界，因此被称为**渐进时间复杂度**，简称为**时间复杂度**。大 O 表示法的 O 是单词 Order 的首字母，表示“数量级”。因此，大 O 表示法并不需要给出运行时间的精确值，而只需要给出一个数量级，表示当问题规模很大时算法运行时间的增长是受限于哪一个数量级的函数，所以在选择 $F(N)$时，通常选择的是比较简单的函数形式，并忽略低次项和系数。表 1-1 给出了常用函数及其名称。

表 1-1 常用的时间复杂度函数

函 数	名 称	函 数	名 称
1	常量	N^3	立方
$\log N$	对数	2^N	指数
N	线性	$N!$	指数
$N\log N$	$N\log N$	N^N	指数
N^2	平方		

在表 1-1 中，常量的时间复杂度表示算法的运行时间与问题规模无关，总是执行有限个标准操作。

时间复杂度为多项式的算法称为**多项式时间算法**，时间复杂度为指数函数的算法称为**指数时间算法**。最常见的多项式时间算法的时间复杂度之间的关系为

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

即 $O(1)$是最好的算法，其次是 $O(\log N)$。指数时间算法的时间复杂度之间的关系为

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

而多项式时间的算法要比指数时间的算法好。这样就能为算法的时间性能评价给出依据。例如，一个时间复杂度为 $O(N)$的算法要比一个时间复杂度为 $O(N^3)$的算法好。

由于算法的运算量除了与问题规模有关以外，还与被处理的数据的分布情况有关。不同的数据分布有不同的运行时间函数。在算法分析中，通常用**最好情况的时间复杂度**、**最坏情况的时间复杂度**以及**平均情况的时间复杂度**来描述。

例如，在一个有 n 个元素的数组中查找某个元素是否出现，采用的算法是从第一个元素开始，依次往后查找，直到找到了该元素或找遍了整个数组都没有找到这个元素。如果被查找的元素出现在第一个位置，则比较一次就找到了该元素。在这种情况下，算法所需的时间最短，这就是最好情况。如果被查找的元素出现在最后或根本没有出现，则需要比较所有的元素，即比较 n 次。在这种情况下，算法所需的运行时间最长，这就是最坏情况。如果每个元素被查找的概率是相同的，对于多次查找平均起来需要检查 $n/2$ 个元素，这就是平均情况。

在程序运行时，不可能如此好运，每次都遇到最好情况，因此最好情况下的时间性能

不足以说明问题。平均的时间性能比较难算，有时甚至定义平均本身也已经够难的。因此在衡量时间性能时，通常采用比较容易分析和计算，并且也最具有实际价值的最坏情况的时间性能。

要计算时间复杂度必须先得到时间性能函数。要得到时间性能函数需要先定义能代表算法主要操作的标准操作，再计算标准操作的次数，得到一个标准操作次数和问题规模的函数。然后取出函数的主项，就是它的时间复杂度的大 O 表示。但这种计算法过于复杂。事实上，可以有一些简化的方法。时间复杂度的计算主要依据以下两个定理。

定理 1.1　求和定理：假定 $T_1(n)$、$T_2(n)$是程序段 P_1、P_2 的运行时间，并且 $T_1(n)$是 $O(f(n))$的，而 $T_2(n)$是 $O(g(n))$的。那么，先运行 P_1、再运行 P_2的总的运行时间是：$T_1(n)+T_2(n)=O(\mathrm{MAX}(f(n),g(n))$。

证明：根据定义，对于某些常数 c_1、n_1，及 c_2、n_2，由已知可得：

在 $n\geqslant n_1$ 时，$T_1(n)\leqslant c_1 f(n)$成立。

在 $n\geqslant n_2$ 时，$T_2(n)\leqslant c_2 g(n)$成立。

设 n_1 和 n_2 之间的最大值为 n_0，即 $n_0=\mathrm{MAX}(n_1,n_2)$。

那么，在 $n\geqslant n_0$时，$T_1(n)+T_2(n)\leqslant c_1 f(n)+c_2 g(n)$成立。所以，$T_1(n)+T_2(n)\leqslant (c_1+c_2)\mathrm{MAX}(f(n),g(n))$。于是，命题得证。

定理 1.2　求积定理：如果 $T_1(n)$和 $T_2(n)$分别是 $O(f(n))$和 $O(g(n))$的，那么 $T_1(n)\times T_2(n)$是 $O(f(n)\times g(n))$。

证明：根据已知条件，可得在 $n\geqslant n_1$ 时，$T_1(n)\leqslant c_1 f(n)$成立。在 $n\geqslant n_2$ 时，$T_2(n)\leqslant c_2 g(n)$成立。其中 c_1、n_1 及 c_2、n_2 都是常数。所以，在 $n\geqslant \mathrm{MAX}(n_1,n_2)$时，$T_1(n)\times T_2(n)\leqslant c_1 c_2 f(n)g(n)$，所以：$T_1(n)\times T_2(n)$是 $O(f(n)\times g(n))$的。

这两个定理为时间复杂度的计算带来了很大的便利。从这两个定理可以得到如下的计算规则。

规则 1：每个简单语句，如赋值语句、输入输出语句，它们的运行时间与问题规模无关，在每个计算机系统中运行时间都是一个常量，因此时间复杂度为 $O(1)$。

规则 2：条件语句，if＜条件＞then＜语句＞else＜语句＞的运行时间为执行条件判断的代价，一般为 $O(1)$，再加上执行 then 后面的语句的代价（若条件为真），或执行 else 后面的语句代价（若条件为假）之和，即 max(O(then 子句)，O(else 子句))。

规则 3：与问题规模有关的循环语句是分析的重点。循环语句的执行时间是循环控制行和循环体执行时间的总和。循环控制行一般是一个简单的条件判断，因此循环语句的执行时间是循环体的运行时间乘循环次数。

规则 4：嵌套循环语句，对外层循环的每个循环周期，内层循环都要执行它的所有循环周期，因此，可用求积定理计算整个循环的时间复杂度，即最内层循环体的运行时间乘所有循环的循环次数。例如语句：

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

的时间复杂度为 $O(n^2)$。

连续语句：利用求和定理把这些语句的时间复杂度相加。例如程序段：

```
for (i=0; i<n; i++) a[i]=0;
for (i=0; i<n; i++)
    for (j=0; j<n; j++) a[i]=i+j;
```

由两个连续的循环组成。第一个循环的时间复杂度为 $O(n)$，第二个循环的时间复杂度为 $O(n^2)$。根据求和定理，整段程序的时间复杂性为 $O(n^2)$。

有了这些简化方法后，求算法的时间复杂度就简单多了。没有必要统计所有标准操作的次数，而只要找出算法中最复杂、运行时间最长的程序段，计算这个程序段的运算量，从而得到整个算法的时间复杂度。

1.3.2 空间复杂度

当求解问题的规模增大时，算法所占用的空间也会随之增加。一个算法所使用的空间，除了程序本身所占用的空间之外，还有在运行过程中系统为存储数据分配的空间，如数组、结构、类占用的空间以及动态分配的空间。有时算法还使用一些额外的存储空间。算法所需的这些空间和问题规模之间的关系函数 $S(n)$，当 $n\to+\infty$ 时的数量级，被称为**渐进空间复杂度**。它的分析方法与渐进时间复杂度类似，所以就不再重复讨论了。

在一般的算法和数据结构的教科书中，通常都把重点放在时间复杂度分析上，对空间复杂度仅分析除了存储被处理的数据元素及关系以外算法所使用的额外空间，并把它们和具有相同功能的算法所使用的额外空间互相对照比较。在以后的分析中，也将遵循这一惯例。

习题与解答

一、简答题

1. 把以下函数按照等价的大 O 分组。

$$x^2, x, x^2+x, x^2-x \text{ 和 } x^3/(x-1)$$

【解】 大 O 分析法只关心运行时间函数的主项，不关心系数和低次项，因此，上面这些函数可以分成两大类。一类是 $O(x)$，一类是 $O(x^2)$。

$O(x)$的函数有：x

$O(x^2)$的函数有：$x^2, x^2+x, x^2-x, x^3/(x-1)$

2. 给出下列代码的大 O 分析。

(1)
```
Sum=0;
For (i=0; i<n; i++)
    Sum++;
```

(2)
```
Sum=0;
For (i=0; i<n; i++)
```

```
        For (i=0; j<n; j++)
            Sum++;
```

(3)
```
Sum=0;
For (i=0; i<n; i++)
    For (i=0; j<n*n; j++)
        Sum++;
```

(4)
```
Sum=0;
For (i=0; i<n; i++)
    For (i=0; j<i; j++)
        Sum++;
```

【解】 (1) 语句段由一个赋值语句和一个重复 n 次的循环组成。赋值语句的时间复杂度是 $O(1)$。根据求和定理,循环语句的时间复杂度就是整个语句段的时间复杂度。该循环的循环体是一个自增语句,其时间复杂度是 $O(1)$。循环的循环次数是 n,因此整个循环的时间复杂度是 $O(n)$。

(2) 语句段由一个赋值语句和一个嵌套循环组成。根据求和定理,嵌套循环语句的时间复杂度就是整个语句段的时间复杂度。该嵌套循环的外层循环和里层循环的循环次数都是 n,最内层的循环体的时间复杂度是 $O(1)$。根据求积定理,该循环的时间复杂度是 $O(n^2)$。

(3) 语句段由一个赋值语句和一个嵌套循环组成。根据求和定理,嵌套循环语句的时间复杂度就是整个语句段的时间复杂度。该循环的外层循环 n 次,内层循环 $n\times n$ 次,最内层循环体的时间复杂度是 $O(1)$。根据求积定理,该循环的时间复杂度是 $O(n^3)$。

(4) 语句段由一个赋值语句和一个嵌套循环组成。根据求和定理,嵌套循环语句的时间复杂度就是整个语句段的时间复杂度。该循环的外层循环的循环次数是 n,内层循环的可能的最大循环次数也是 n,最内层的循环体的时间复杂度是 $O(1)$。根据求积定理,该循环的时间复杂度是 $O(n^2)$。

二、选择题

1. (2011 年考研真题)设 n 是描述问题规模的非负整数,下面的程序片段的时间复杂度是(　　)。

```
x=2;
while (x<n/2)
  x=2*x;
```

A. $O(\log_2 n)$　　B. $O(n)$　　C. $O(n\log_2 n)$　　D. $O(n^2)$

【解】 正确答案为 A。

这个程序片段由两个语句组成。第一个是赋值语句,运行时间是常量,即 $O(1)$的时间复杂度。第二个语句是一个循环语句,循环体是一个赋值语句,因此这个语句的时间复杂度就是循环执行的次数。循环从 $x=2$ 开始,每执行一次循环体,x 的值增大一倍,直到 x 大于等于 $n/2$,因此循环一共执行 $\log_2 n/2$ 次。忽略系数,循环语句的时间复杂度是

$O(\log_2 n)$。根据加法定理，这段程序的时间复杂度是 $O(\log_2 n)$。

2.（2012 年考研真题）求整数 $n(n\geqslant 0)$阶乘的算法如下，其时间复杂度是(　　)。

```
int fact(int n)
{   if (n<=1) return 1;
    return n * fact(n-1);
}
```

A. $O(\log_2 n)$　　B. $O(n)$　　C. $O(n\log_2 n)$　　D. $O(n^2)$

【解】 正确答案为 B。

函数 fact 是一个递归函数，该函数只有一个 if 语句。设函数的运行时间函数是 $T(n)$，则当 $n\leqslant 1$ 时，执行 return 语句，即 $T(n)$是常量，记为 $T(n)=c$。当 $n\geqslant 1$ 时，先调用 fact$(n-1)$，然后执行一次乘法后返回，即 $T(n)=T(n-1)+c$。由此可得

$$T(n) = T(n-1) + c$$
$$T(n-1) = T(n-2) + c$$
$$T(n-2) = T(n-3) + c$$
$$\cdots$$
$$T(1) = c$$

将上述所有公式相加，可得

$$T(n) = nc$$

忽略系数，可得 $O(n)=n$。

3. 下列程序段的时间复杂度是(　　)。

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

A. $O(\log_2 n)$　　B. $O(n)$　　C. $O(n\log_2 n)$　　D. $O(n^2)$

【解】 正确答案为 C。

程序由一个嵌套的循环组成，外层循环的循环变量 k 从 1 开始，每个循环周期 k 都乘 2，直到 $k>n$，因此外层循环执行的次数是 $\log_2 n$。里层循环的循环变量 j 从 1 开始，每个循环周期增加 1，直到 n，里层循环的执行次数是 n。在外层循环的每个循环周期中，里层循环必须执行它的所有循环周期，所以总的时间复杂度是 $O(n\log_2 n)$。

三、程序设计题

1. 素数是除了 1 和它本身之外没有其他因子的数。试设计一个时间复杂度为$O(\sqrt{n})$的判断一个正整数 N 是不是素数的算法 isPrime。

【解】 判断一个数是否为素数的最直接的方法是从素数的定义出发。检查所有 1～n 之间的所有数，找出所有 n 的因子，检查因子个数是否为两个。如果正好是两个因子，则为素数，否则为非素数。显然，该算法的时间复杂度是 $O(n)$的。

要得到 $O(\sqrt{n})$的算法，需要对上述的蛮力算法进行修改。观察发现，可以有以下几个