第1章

绪　论

教学要求

① 了解：数据结构这门学科的发展历史，以及在计算机科学中所处的地位。

② 掌握：与数据结构有关的概念和术语。

③ 了解：算法的概念及特征。

④ 掌握：如何描述算法，以及算法和程序的关系。

⑤ 掌握：如何评价一个算法的好坏。

1.1 引言

在 20 世纪 40 年代第一台 ENIAC 产生的时候，电子计算机的应用范围几乎仅局限于科学和工程的计算，其处理的对象是纯数值性的信息，通常，人们把这类问题称为数值计算。

电子计算机的发展异常迅猛，这不仅表现在计算机本身运算速度不断提高、信息存储量日益扩大、价格逐步下降，更重要的是计算机广泛地应用于情报检索、企业管理、系统工程等方面，已远远超出了科技计算的范围，而渗透到人类社会活动的一切领域。与此相应，计算机的处理对象也从简单的纯数值性信息发展到非数值性的和具有一定结构的信息（或数据），而非数值性数据是具有一定的结构的，数据与数据之间的关系也较为复杂。如何选择合适的数据表示（即结构），如何有效地组织计算机存储，如何在此基础上有效地实现对象之间的“运算”关系？传统的解决数值计算的许多理论、方法和技术已不能满足解决非数值计算问题的需要，必须进行新的探索。数据结构就是研究和解决这些问题的重要基础理论。

1.2 数据结构的发展简史及其在计算机科学中所处的地位

1. 发展史

“数据结构”作为一门独立的课程在国外是从 1968 年开始设立的。在这之前，它涉及的内容分布在其他课程中。1968 年在美国一些大学的计算机系的教学中，将“数据结构”规定为一门课程，美国人唐·欧·克努特教授开创了数据结构的最初体系，他所著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。

2. 地位

（1）“数据结构”在计算机科学中是一门综合性的专业基础课。

（2）数据结构是介于数学、计算机硬件（特别是编码理论、存储装置和存取方法等）和计算机软件三者之间的一门核心课程。

（3）数据结构这门课的内容不仅是一般程序设计（特别是非数值性程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。

1.3 什么是数据结构

一般来说，用计算机解决一个实际问题时，通常要经过以下几个步骤：首先从具体问题抽象出一个适当的数据模型（或数学公式）；然后设计一个用来描述此模型的算法；最后编写程序并上机调试，直到最终解决实际问题。若计算机处理的是数值计数问题，求解模型可

用数学方程描述，涉及的运算对象一般是整型、实型或字符型等一些简单类型的数据。此时，程序设计者主要关注程序设计的技巧，而对数据的组织和存储就不那么关心。随着计算机应用领域的不断扩大，计算机处理的对象更多是非数值计算问题，如资料的查询、组织机构的管理、交通道路的规划等问题，它们的数学模型无法用数学方程进行描述，此时就必须建立相应的数据结构进行描述，分析问题中所用的数据是如何组织的，研究数据之间的关系，进而为解决这些问题设计出合适的数据结构。

下面从简单的案例入手，先直观地了解数据结构研究的是什么，然后给出具体的定义。

1. 线性表示例

例 1-1　一个具有 8 条记录的反映学生信息的数据文件如图 1-1 所示。

学号	姓名	性别	籍贯	电　话	通 信 地 址
01	张三	男	长沙	8639000	麓山南路 327 号
02	李四	男	北京	23456789	学院路 435 号
03	王五	女	广州	30472589	天河路 478 号
04	赵六	男	上海	41237568	南京路 1563 号
05	钱七	女	南京	5013472	南京大学
06	刘八	女	武汉	61543726	武汉大学
07	朱九	男	昆明	4089651	云南大学
08	孙十	女	杭州	6154372	西湖路 635 号

图 1-1　学生数据表

该学生数据表（即花名册）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）。整个花名册就是一个数据结构。针对该学生数据表，研究的问题主要有以下三方面。

① 元素之间的关系：具体表现为表中有且仅有一条起始记录和终止记录；除了起始记录外，表中其他记录有且仅有一条记录是与之相邻的，且位于它前面的记录（称为直接前驱）；除了终止记录外，表中其他记录有且仅有一条记录是与之相邻的，且位于它后面的记录（称为直接后继）。这就是该花名册表的逻辑结构。

② 存储：主要描述表中的记录如何存放在计算机中，这是存储结构。

③ 操作：对表中涉及的数据如何进行操作，如查找、插入、删除、修改和排序等，这就涉及数据的运算问题。

只有弄清以上三个方面的问题，才能有效地使用花名册这个数据结构，有效地解决这些学生基本信息的计算机管理问题。

2. 树形结构示例

例 1-2　一个单位的组织结构（或表示人类血缘关系的祖谱图）通常如图 1-2 所示。最顶层的结点代表某个机构的最高权力部分，它下面有若干分支，分别代表不同的机构或部门。

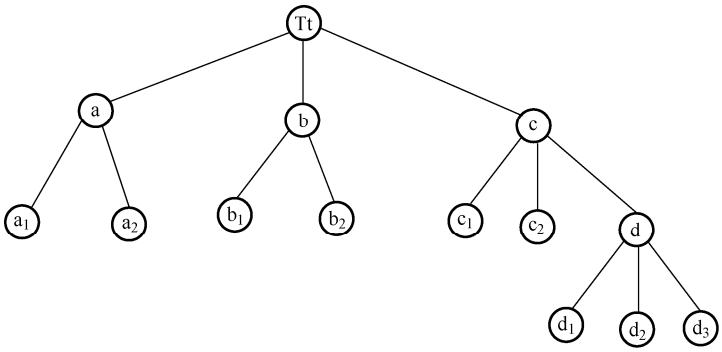

图 1-2　树形结构示意图

从图 1-2 中可看出呈现这种结构的记录与记录之间的关系（即逻辑结构）不同于例 1-1 的线性表，它像一棵倒立的树，具体表现如下：除起始记录（也称树根结点）外，其他记录有且仅有一个直接前驱记录，但直接后继记录可以有多个。这也是一个数据结构。

3. 图状（形）结构示例

例 1-3　城市之间的交通联系如图 1-3 所示，图中的顶点（结点）代表各城市，图中的边表示城市之间的道路。这幅交通网络图也表示一个数据结构，图中记录与记录之间的关系不同于以上两种：在这个数据结构中，结点之间的关系可以是任意的，任意两个结点之间都可以相关。即图中任意一个结点的直接前驱和直接后续都可以有多个。

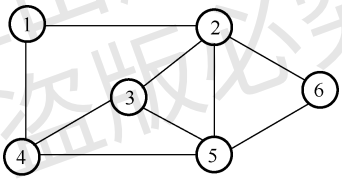

图 1-3　图形结构示意图

1.4　基本概念和术语

下面介绍数据结构的相关术语。

1．数据（Data）

数据是指能够输入计算机中，并被计算机识别和处理的符号的集合。例如，数字、字母、汉字、图形、图像、声音都可以称为数据。

2．数据元素（Data Element）

数据的基本单位是数据元素。在计算机中通常作为一个整体进行考虑和处理，例如，图 1-1 中学号为 01 的学生记录如下：

学号	姓名	性别	籍贯	电话	通信地址
01	张三	男	长沙	8639000	麓山南路 327 号

一个数据元素可由若干个数据项（域或称字段）组成，如上图中学号为 01 的这个数据元素，是由学号、姓名等 6 个数据项组成的。数据项是数据的最小单位。数据元素也称为结点、元素、记录等。

3. 数据类型（Data Type）

数据类型是一组性质相同的值的集合，以及定义于这个值集合上的一组操作的总称。数据类型是指程序设计语言中各变量可取的数据种类。数据类型是高级程序设计语言中的一个基本概念，它和数据结构的概念密切相关，主要体现在以下两方面：

① 在程序设计语言中，每一个数据都属于某种数据类型。数据类型显式或隐含地规定了数据的取值范围、存储方式及允许进行的运算。可以认为，数据类型是在程序设计中已经实现的数据结构。例如，C 语言中用到的基本整数类型（int），它的取值范围是−32 767~+32 768，可进行的运算有加、减、乘、除、取模（即+、−、*、/、%）。

② 在程序设计过程中，当需要引入某种新的数据结构时，总是借助编程语言所提供的数据类型来描述数据的存储结构。

4．抽象数据类型（Abstract Data Type，ADT）

ADT 是指一个数学模型及定义在该数学模型上的一组操作，用户进行软件系统设计时，从问题的数学模型中抽象出来的逻辑数据结构和逻辑数据结构上的运算，而不考虑计算机的具体存储结构和运算的具体实现算法。

ADT 并不是全新的概念，是大家熟悉的基本数据类型概念的引伸和发展。具体关系如下。

① 与数据类型本质相同：基本数据类型已隐含着数据模型和定义在该模型上的运算。如各计算机均有的“整数”类型是一个抽象数据类型，不同处理器上实现的方法不同，但其定义的数学特征相同，在用户看来都是相同的。故“抽象”的意义在于数据类型的数学抽象特性。

② 不同在于抽象数据类型的范畴更广（即引申和发展）：数据类型指的是高级语言支持的基本数据类型；抽象数据类型指的是在基本数据类型支持下用户新设计的数据类型。比如本书主要讨论的表、堆栈、队列、串、树、图等典型的常用数据结构。这些数据结构就是一个个不同的抽象数据类型。

抽象数据类型的特征是使用与实现相分离，实行封装和信息隐蔽。也就是说，在抽象数据类型设计时，把类型的定义与其实现分离开来。

通常软件设计采用模块化方法，抽象数据类型是构造大型软件的最基本模块。

描述抽象数据类型通常采用如下书写格式：

```
ADT <抽象数据类型名> {
   Data < 数据描述>
   Operations <操作声明>
} ADT 抽象数据类型名
```

5. 数据结构（Data Structure）

数据结构是一门研究数据是如何组织、存储、数据之间的相互关系及运算操作的学科。

具体来讲，数据结构主要包含 3 个方面的内容，即数据的逻辑结构、数据的存储结构和对数据所施加的运算（或操作）。

① 元素之间的相互关系又称为数据的逻辑结构。数据的逻辑结构独立于计算机，是数据本身所固有的。

② 数据元素在计算机内存中的表示，称为数据的物理结构（存储结构），必须依赖于计算机。

③ 对数据需要施加的操作主要包括查找、插入、删除、修改和排序等。运算的定义直接依赖于逻辑结构，但运算的实现必依赖于存储结构。

6. 数据结构的分类

（1）从逻辑结构的角度出发划分数据结构

① 线性结构：主要特点如下。

- 有且仅有一个开始结点（表头结点）a_1。
- 有且仅有一个终端结点（表尾结点）a_n。
- 除表头结点外，其余的结点有且仅有一个直接前驱结点。
- 除表尾结点外，其余的结点有且仅有一个直接后继结点。

总之，元素之间为一对一的线性关系。

② 非线性结构：元素之间为一对多或多对多的非线性关系，即每个元素有多个直接前驱或多个直接后继。其中，元素之间为一对多关系的是树形结构，呈现多对多关系的是图状（形）结构。

（2）从存储结构角度出发划分数据结构

① 顺序存储（向量存储）：所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存仍然相邻。

② 链式存储：所有元素存放在可以不连续的存储单元中，但元素之间的关系可以通过地址确定，逻辑上相邻的元素存放到计算机内存后不一定是相邻的。

③ 索引存储：使用该方法存放元素的同时，还建立附加的索引表，索引表中的每一项称为索引项，索引项的一般形式是（关键字，地址），其中的关键字是能唯一标识一个结点的某个数据项，如例 1-4 所示。

例 1-4 用索引存储方法来表示各城市的基本信息，具体包括城市名、区号及简单说明。索引存储结构如表 1-1（a）、（b）所示。

表 1-1（a）　索引表

地址	关键字	指针
500	0531	320
510	021	230
520	010	200
530	0371	300
540	0755	280
550	022	250

表 1-1（b）　结点表

地址	城市名	区号	说明
200	北京	010	中国首都
230	上海	021	金融中心
250	天津	022	直辖市
280	深圳	0755	开放特区
300	郑州	0371	中原中心
320	济南	0531	山东省会

④ 散列存储：散列存储结构是根据结点的值确定结点的存储地址。通过构造散列函数，以结点中某个字段的值为自变量，通过散列函数计算对应的函数值，用该函数值来确定结点（即元素）存放的地址。详细可见第 7 章。

一般来说，这 4 种基本存储方法既可单独使用，也可组合起来，同一种逻辑结构可采用不同的存储方法，得到不同的存储结构，选择何种结构要视具体问题的要求而定。

1.5　算法

1.5.1　算法的概念

软件的最终成果都是以程序的形式表现的，数据结构的各种操作都是以算法的形式描述的。数据结构、算法和程序是密不可分的。三者的关系正如瑞士苏黎世联邦工业大学著名计算机科学家沃思（Wirth）提出的一个公式：数据结构+算法=程序。

其中，算法是灵魂，它解决“做什么”和“怎么做”的问题。

在给出算法的概念之前，先举两个例子来说明什么是算法。

例 1-5　求 1+2+3+4+…+100=?

方法 1：直接相加，得出结果。

方法 2：=100+(1+99)+(2+98)+…+(49+51)+50

=50*100+50

=5050

例 1-6　求 1*2*3*4*5=?

最原始的方法：S1：先求 1*2，得 2。

S2：将 2*3，得 6。

S3：将 6*4，得 24。

S4：将 24*5=120，便得到最后的结果。

若求 1*2*3*…*1000=？，再用此方法就不可行。因为需要 999 个步骤，故要找一种通用的方法。可设两个变量，一个变量代表被乘数（p），一个变量代表乘数（i）。

S1：使 p=1。

S2：使 i=2。

S3：使 p*i,将 p*i=>p。

S4：i+1=>i。

S5：若 i 不大于 5，返回重新执行 S3、S4 和 S5；否则，算法结束。

可见，算法（Algorithm）是对特定问题求解步骤的一种描述，是指令的有限序列。其中每一条指令表示一个或多个操作。

1.5.2 算法的特征

一个算法应该具有下列特性。

① 有穷性：一个算法应包含有限的操作步骤，而不能是无限的。

② 确定性：每一步应是确定的，而不应是含糊的、模棱两可的，即无二义性。

③ 有输入：有零个或多个输入。

④ 有输出：有一个或多个输出，算法的目的是为了“解”，求结果。所以，没有输出的算法是没有意义的。

⑤ 有效性（可行性）。算法中的每一步都应能有效地执行，并得到确定的结果，如 b=0，则表达式 a/b 是不能有效执行的。

1.5.3 算法和程序

算法的含义与程序十分相似，但二者是有区别的。一个程序不一定满足有穷性（如操作系统这个系统软件，只要整个系统不遭破坏，它将永远不会停止，即使没有作业需要处理，它仍处于动态等待中。因此，操作系统不是一个算法）。另外，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。算法代表了对问题的解，而程序则是算法在计算机上的特定的实现。一个算法若用计算机语言来书写，则它就是一个程序。

算法与数据结构是相辅相成的。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率。反之，一种数据结构的优劣由各种算法的执行来体现。设计一个好的算法应考虑以下 4 条准则。

① 正确性：算法应当满足具体问题的需求。

② 可读性：算法主要为了人的阅读与交流，其次才是机器执行。可读性好有助于人对算法的理解，这样程序易于调试和修改。

③ 健壮性（坚固性）：当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫明其妙的输出结果。

④ 效率与低存储量需求：即高时间效率和高空间效率，其中效率指的是算法执行时间。

1.5.4 算法的描述

算法可以使用各种不同的方法来描述，主要有以下几种。

1. 自然语言

自然语言是人们日常使用的语言，如汉语、英文或其他语言。用自然语言来描述算法的优点是简单且便于人们对算法的阅读；缺点是不够严谨。

2. 流程图表示算法

流程图表示算法是一种用图形表示算法的方法。ANSI 规定了一些常用的流程图符号，如表 1-2 所示。

表 1-2　常用的流程图符号

符号名称	图形表示
起止框	
输入/输出框	
判断框	
处理框	
流程线	

例 1-7　用流程图表示例 1-6 的算法，如图 1-4 所示。

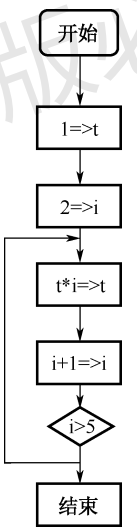

图 1-4　算法的流程图表示

当算法比较复杂时，画流程图既费时又不方便，因为对流程线的使用没有严格限制，使用者可以不受限制地使流程随意地转来转去，使流程图变得毫无规律。这种乱麻一样的算法称为 BS 算法（a Bowl of Spaghetti)。

为了限制这种无规律的任意转向，人们规定了 3 种基本结构：顺序结构、选择机构和循环结构。由这些基本结构像建房子一样搭成各种算法结构，这样就能保证算法的质量。

① 顺序结构如图 1-5 所示。

② 选择结构如图 1-6 所示。

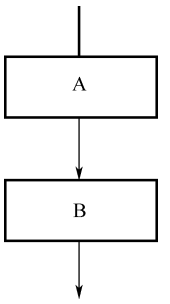

图 1-5　顺序结构

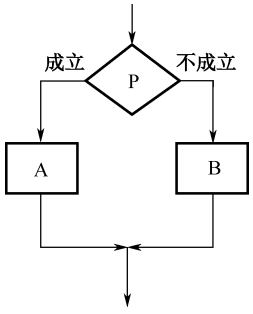

图 1-6　选择结构

③ 循环结构根据不同的执行情况，分为当型循环和直到型循环两种，如图 1-7 和图 1-8 所示。

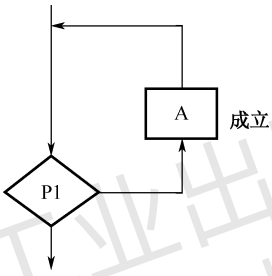

图 1-7　当型循环

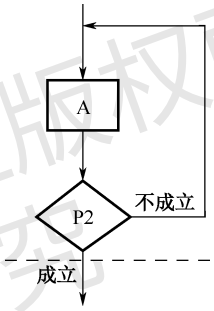

图 1-8　直到型循环

由这些基本结构构成的算法属于“结构化”算法。其实，基本结构不一定只限于上面 3 种，只要具备下面 4 个特点都可作为基本结构：

- 只有一个入口；
- 只有一个出口；
- 结构内的每一部分都有机会被执行到（即对每一个框来说，有一条从入口到出口的路径通过它）；
- 结构内不存在“死循环”（无终止的循环）。

3. 用 N-S 图表示算法

1973 年美国学者提出一种新的用图形表示算法的形式，即 N-S 图（盒图）。N-S 图完全去掉了带箭头的流程线。这种流程图适用于结构化程序设计。同样有三种结构：顺序结构、选择结构和循环结构。N-S 图的每种基本结构都是一个矩形框，整个算法可像堆积木一样堆成。

① 顺序结构如图 1-9 所示。

② 选择结构如图 1-10 所示。

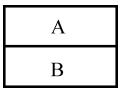

图 1-9　顺序结构

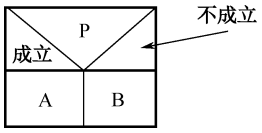

图 1-10　选择结构

③ 循环结构如图 1-11 所示。

例 1-8　用流程图表示例 1-6 的算法，如图 1-12 所示。

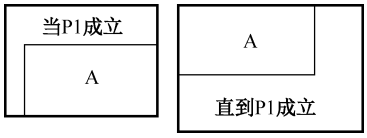

图 1-11　循环结构的当两种形式

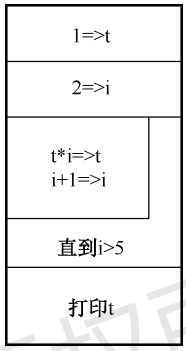

图 1-12　算法的 N-S 图表示

无论是流程图还是 N-S 图，均是算法的图形表示方法，优点是直观形象，易于理解；缺点是不能直接在计算机上执行，若要将它转换成可执行的程序，还有一个编程的问题。

4．伪代码表示算法

为解决理解与执行这两者之间的矛盾，人们常常使用一种称为伪码语言的描述方法来进行算法描述。伪码语言介于高级程序设计语言和自然语言之间，它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此，它比程序设计语言更容易描述和被人理解，而比自然语言更接近程序设计语言。伪码语言虽然不能直接执行，但很容易被转换成高级语言，所以，伪代码表示具有结构清晰、代码简单、可读性好的特点。

例 1-9　用伪代码表示任意 3 个数中求最大数的算法。代码如下：

```
Begin（算法开始）
输入 A，B，C
IF A>B 则 A→Max
否则 B→Max
IF C>Max 则 C→Max
Print Max
End （算法结束）
```

5．高级语言表示

直接用某种高级编程语言来表示算法，要求按照严格的语法规则来描述，该方法描述的算法可直接用于程序之中。

注：本书的算法全部用 C 语言来描述。

1.5.5 算法分析

衡量算法优劣最重要的一点是效率与低存储量要求。算法效率（算法运行的时间）由时间复杂度来度量。一般，鉴于空间（内存）较为充足，故把算法的时间复杂度作为分析的重点。在求时间复杂度之前，先介绍频度统计法。

频度：一条语句的频度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基本语句的频度之和。

例 1-10 求下列算法段的语句频度。

```
for(i=1; i<=n; i++)
for(j =1; j <=n ; j++)
x=x+1;
```

分析：该算法为一个二重循环，执行次数为内、外循环次数相乘，因此频度=n^2。

1．时间复杂度

在刚才提到的算法频度中，n 称为问题的规模（通常用正整数 n 表示）。当 n 不断变化时，语句频度也会不断变化。但有时我们想知道它变化时呈现什么规律。为此，我们引入时间复杂度概念。或者说时间复杂度是问题规模的函数。但算法的时间复杂度考虑的只是对问题规模 n 的增长率，难以精确计算出语句频度，只需求出它关于 n 的增长率或阶（数量级）即可。

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 $f(n)$，记作 $T(n)=O(f(n))$。

其中，$T(n)$表示时间复杂度，大写字母 O 为英文 Order（即数量级）单词的第一个字母。

例如，

```
for (i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    x=x+1;
```

语句 $x=x+1$ 执行次数关于 n 的增长率为 n^2，故它的时间复杂度 $T(n)=O(n^2)$。

例 1-11 求出下列算法段的时间复杂度。

```
(1) x=x+1
(2)for(i=1; i<=n; i++)
    x=x+1;
 (3)for(k=1;k<n;k++)
   { y=y+1;   ①
   for(j=0;j<=2n;j++)
   x++;      ②
     }
(4)for (i=1;i<=n;i++)
   for  (j=1;j<=i;j++)
   for ( k=1;k<=j;k++)
      x=i+j-k;
(5) k=1;          ①
    while(k<=n)
    k=k*2         ②
```

分析可知：（1）$T(n)=O(1)$（2）$T(n)=O(n)$，第 3 段代码，语句①的频度是 $n-1$，语句②的频度是$(n-1)(2n+1)=2n^2-n-1$，则该程序段的时间复杂度 $T(n)=n-1+2n^2-n-1=O(n^2)$，第 4 段代码是三层 for 循环，故时间复杂度为 $O(n^3)$。第 5 段代码中的语句①的频度是 1，语句②

的频度设为 $f(n)$，则有 $2^{f(n)} \leqslant n$，即 $f(n) \leqslant \log_2 n$，取最大值 $f(n)=\log_2 n$。故该程序段的时间复杂度 $T(n)=1+\log_2 n=O(\log_2 n)$。

思考：以下代码段的时间复杂度是多少？

```
k=1;
while(k<=n)
k=k*3;
```

总结：在各种不同算法中，若程序的运行时间不随着问题规模 n 的增加而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数，此类算法的时间复杂度仍是 O(1)。另外，在频度不相同时，时间复杂度有可能相同，如 $T(n)=n^2+3n+4$ 与 $T(n)=4n^2+2n+1$ 的频度不同，但时间复杂度相同，都为 $O(n^2)$。

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 $O(n^3)\cdots k$ 次方阶 $O(n^k)$、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，算法的执行效率降低。

2．空间复杂度

一个程序需要的空间即存储量，包括输入数据所占空间、程序本身所占空间和辅助变量所占空间。我们一般是讨论算法占用的辅助存储空间，即空间复杂度是对一个算法在运行过程中临时占用的存储空间大小的量度。一般也作为问题规模 n 的函数，以数量级形式给出，记作：

```
S(n)=O(g(n))
```

讨论方法与时间复杂度类似，在此不再赘述。

习题 1

一、填空题

① 数据的逻辑结构包括（　　）和（　　）两种类型，后者主要包括（　　）和（　　）两种结构。

② 数据的存储结构包括（　　）、（　　）、（　　）和（　　）四种基本类型。

③ 数据的最小单位是（　　）。

二、按要求完成以下各题

① 数据结构的研究内容是什么？

② 求下面程序段的时间复杂度。

```
    for(k=1;k<=n;k++)
{ for(j=1;j<=n;j++)
  {c[k][j]=0;
    for(m=1;m<=n;m++)
      c[k][j]=c[k][j]+a[k][m]*b[m][j]
    }
}
```