第 1 章　概论

1.1 引言

大千世界纷繁复杂，瞬息万变。

以日常生活中的城市交通为例，一座大城市会有数百万辆不同种类、不同品牌、不同年代的机动车，这些车辆又日日夜夜忙碌奔波在大街小巷中。交通管理部门需要记录每辆车的具体情况，包括车牌号码、车辆类型、品牌型号、所有权人、注册日期等基本属性；同时，交通管理部门更关注各个时段、各个路段的拥堵情况，并根据拥堵情况来规划和调整道路行驶策略，以减缓城市拥堵状况。另外，每位驾驶员则关心如何选择行驶路线从而更快到达目的地。

对于民用航空来说，也是如此。 每个航空公司拥有数百架不同年代、不同机型的飞机，每个机场也有各自的跑道、停机位等相关信息。在空管部门的协调下，每天成千上万的飞机在各地机场间来往穿梭，再考虑气象等影响因素，航班规划也是一件异常复杂的事情。

再举一个例子，大家生活中更为熟悉的朋友圈。朋友圈既记录每位朋友的基本信息，包括姓名、性别、出生日期、联系方式、与自己的关系等基本属性；更值得关注朋友圈中朋友状态的变化和朋友圈的动态发展。朋友们会在不同的时间、地点发布风格各异的消息。有时，朋友们也会同时发布共同关注的事情，形成“刷屏”的盛况。

如此复杂而又奇妙的世界，如何在计算机里表示的呢？又是如何通过强大的计算功能来改造和优化现实世界的呢？这将是本门课程所要探讨的两大问题。带着这两大问题开启数据结构课程之旅吧。

1.2 数据结构相关概念及术语

要把纷繁复杂的大千世界“装入”信息世界中，还要让信息世界与物理世界同步共舞。这离不开数据结构。为了便于后续学习，本节先讨论与数据结构相关的几个基本术语。

（1）数据

数据（data）是指描述物理世界的数值、字符、图形图像、语音等所有能输入到计算机中并被计算机处理的符号集合。也可以说，物理世界是通过数据的形式映射到信息世界中的。

数据可分为两类，一类是**数值型数据**，包括整数、浮点数、复数等，例如物理世界中的温度、湿度、长度等具体数量值在计算机中的表示；另一类是**非数值型数据**，包括字符和字符串、图形图像、语音等，例如多姿多彩的自然美景通过数字摄像机的光电变换、采样、压缩编码等一系列操作，最终以二进制流的形式存储在计算机中，并可通过解码等操作在显示器上重现。

（2）数据元素和数据项

数据元素 (data element) 是数据的基本单位，也是计算机处理或访问的基本单位，相当于物质的“分子”。例如学生成绩登记表，它由许多学生成绩记录组成，每条成绩记录则是构成登记表的元素。

数据元素可以是单个元素，也可以由许多数据项组成。 **数据项** (data item) 是数据的最小单位，相当于“原子”。例如，每个学生成绩记录中可以包括姓名、课程名、成绩等数据项。

（3）数据对象

数据对象 (data object) 是性质相同的数据元素的集合，是数据的一个子集。例如，所有整数构成了整数数据对象，26 个字母集合构成了字符数据对象。

（4）数据结构

通过引言部分的三个举例可以发现，每个数据元素并不是孤立存在的，它们之间存在着某种形式的联系，一般称之为**结构**。例如，朋友圈中朋友之间的联系形成了一张网，就是典型的图状结构。

为此，**数据结构**(data structure) 可以看成是由与特定问题相关的某一数据元素的集合和该集合中数据元素之间的关系组成的。值得注意的是，数据元素集合和数据元素之间的关系描述是静态的，而我们的世界是动态变化的，还需要在静态描述之上定义一组有意义的操作集合，从而可以让信息世界与物理世界同步共舞，并进一步实现对物理世界的优化和改造。

另外，数据结构还有两个相关的概念：一是数据结构的逻辑表示，二是数据结构的物理存储表示。

数据结构的**逻辑表示**是实现对物理世界的抽象和建模，是物理世界“装入”信息世界的第一步。例如物理世界中的人被抽象成图状结构中的一个顶点，人与人之间的关系被描述成顶点之间的边，从而在信息世界中的各种操作都围绕顶点和边这类抽象的逻辑表示来进行。可见，数据结构的逻辑表示是联系物理世界和信息世界的桥梁。数据的逻辑结构可归结为以下四类：线性结构、树形结构、图状结构和集合结构，如图 1.1 所示。

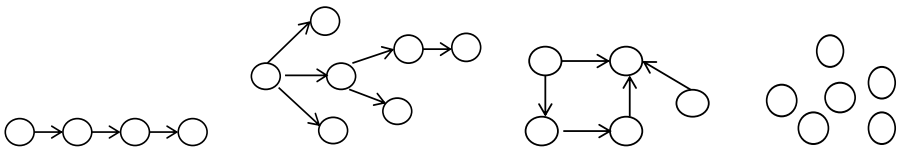

图 1.1 四类数据逻辑结构

数据结构的**物理存储表示**则是在计算机中的具体实现方式，包括数据元素的物理存储表示和数据元素之间关系的物理存储表示。对于数据关系的存储表示，一般又分为两种表现形式，顺序存储和链式存储。不同的物理存储方式将直接影响算法执行的效率。

（5）数据类型

数据类型 (data type) 是一个同类数据值的集合和定义在这个值集合上的一组操作的总称。例如，计算机中整数类型是由范围为 $-2^{15} \sim 2^{15}-1$ 的整数构成，并包括对这些整数的加、减、乘、除和取模运算等操作。

数据类型可分为基本数据类型和结构数据类型两种。基本数据类型中每个数据元素都是无法再分割的整体，如整数、浮点数、字符、指针、枚举量等，又称为原子类型。结构数据类型由基本数据类型或子结构类型按照一定的规则构造而成。例如，一个学生的基本情况是一个结构数据类型，除了包括姓名、性别、年龄等基本类型外，还包括如家庭成员等子结构类型。

（6）抽象数据类型

抽象数据类型 (abstract data type, ADT) 通常是指由用户定义，用以表示应用问题的数据模型以及定义在该模型上的一组操作，又称数据抽象。作为一个抽象数据类型，其构成的两要素为数据的结构和相应的操作集合。

抽象数据类型最重要的是其抽象性质，把使用和实现分离，实行封装和信息隐藏。换句话说，抽象数据类型有两个视图，外部视图和内部视图。外部视图包括抽象数据类型名称、数据对象的简要说明和一组可供用户使用的操作。内部视图包括数据对象的存储结构定义和基于这种存储表示的各种操作的实现细节。

抽象数据类型属于概念层次的模型，它的实现就是面向对象程序设计中的“类”。可见，类和对象是抽象数据类型的实现层次的表示。

1.3 抽象数据类型的表示与实现

抽象数据类型可以用以下三元组表示

$$(D,S,P) \qquad (1.1)$$

其中，D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作集。本书采用以下格式定义抽象数据类型：

```
ADT 抽象数据类型名
{
    数据对象：〈数据对象的定义〉
    数据关系：〈数据关系的定义〉
    基本操作：〈基本操作的定义〉
} // ADT 抽象数据类型名
```

其中，数据对象和数据关系的定义用伪代码描述，基本操作的定义格式如下：

```
基本操作名（参数表）
    初始条件：〈初始条件描述〉
    操作结果：〈操作结果描述〉
```

基本操作有两类参数，输入参数和输出参数。在程序设计时，可由赋值参数和引用参数的形式来实现。初始条件描述了操作执行之前数据结构和参数应满足的条件，若不满足，则操作失败，并返回相应出错信息。 操作结果说明了操作正常完成之后数据结构的变化状况和应返回的结果。

例 1-1 抽象数据类型复数 $z=\alpha+i\beta$ 的定义

```
ADT ComplexNumber
{
    数据对象：D = {α, β | α, β ∈ ElemSet }
    数据关系：R = { <α, β> }
    基本操作：
        //1.初始化、销毁操作
        InitComplex( &Z, a, b)
            操作结果：构造了复数 Z，元素实部α 和虚部 β 分别被赋以参数 a 和 b 的值。
        DestroyComplex( &Z )
            操作结果：复数 Z 被销毁
        //2.访问型操作
        GetComplex( Z, &a, &b )
            初始条件：复数 Z 已存在
            操作结果：将实部α 和虚部 β 赋给参数 a 和 b。
        PrintComplex( Z )
            初始条件：复数 Z 已存在
            操作结果：将复数 Z 打印输出。
        //3.加工型操作
        PutReal( &Z, a )
            初始条件：复数 Z 已存在
            操作结果：将复数实部α 值用参数 a 值替代。
        PutImaginary( &Z, b )
            初始条件：复数 Z 已存在
            操作结果：将复数虚部 β 值用参数 b 值替代。
        AddComplex( &Z, a, b )
            初始条件：复数 Z 已存在
            操作结果：将复数 Z 实部α 增加值 a，虚部 β 增加值 b。
        SubComplex( &Z, a, b )
```

初始条件：复数 Z 已存在
操作结果：将复数 Z 实部 α 减去值 a，虚部 β 减去值 b。

} // ADT ComplexNumber

例 1-1 可以看出，定义一个抽象数据类型时，首先定义数据对象及其取值的范围，然后定义针对具体应用的数据元素之间的关系，最后给出数据元素关系之上的基本操作集合。基本操作集合一般包括这三类操作：一是**初始化操作和销毁操作**，二是对数据元素进行查询的**访问型操作**，三是对数据元素进行修改的**加工型操作**。

另外，从例 1-1 也可看出，抽象数据类型的定义作为外部视图对用户而言已满足要求，但对编程实现而言，还缺少物理存储结构定义和操作细节的描述。

本书约定，文中涉及的代码实现均用类 C/C++的伪代码进行表示，便于突出重点。

例 1-2 抽象数据类型复数 $z = \alpha + i\beta$ 的表示和实现

```
//采用静态顺序存储结构定义复数的结构体
typdef  double  ElemType ;      // 本次用例中元素类型为实数
typedef struct
{
    ElemType real;              // 复数的实部 α
    ElemType imaginary;         // 复数的虚部 β
} ComplexNumber;

//基本操作的实现
typdef    int   Status ;        // 函数返回值类型，一般定义为整型
#define   OK              1     // 正常返回
#define   ERROR           0     // 错误返回
#define   OVERFLOW  -1    //存储空间溢出错误
//1. 初始化操作
Status InitComplex( ComplexNumber  &Z, ElemType a, ElemType b )
{
    //构造复数 Z，元素实部 α 和虚部 β 分别被赋以参数 a 和 b 的值。
    Z.real      = a;
    Z.imaginary = b;
    return OK;
}
//2. 销毁操作
Status DestroyComplex( ComplexNumber  &Z )
{
    //销毁复数 Z。注：静态存储方式下销毁操作无实际意义，动态存储方式下应释放相应存储空间。
    Z.real      = 0;
    Z.imaginary = 0;
    return OK;
}
//3. 取复数的实部和虚部
Status GetComplex( ComplexNumber  Z, ElemType &a, ElemType &b )
{
    //将实部 α 和虚部 β 赋给参数 a 和 b。
    a = Z.real;
    b = Z.imaginary;
    return OK;
}
//4. 打印查看复数
Status PrintComplex( ComplexNumber  Z)
{
    //将复数 Z 按照指定格式打印输出。
```

```
    cout<< Z.real<<"+"<< Z.imaginary<<"i"<<endl;
    return OK;
}
//5. 修改复数的实部
Status PutReal( ComplexNumber  &Z, ElemType a )
{
    //将复数实部α值用参数 a 值替代。
    Z.real = a;
    return OK;
}
//6. 修改复数的虚部
Status PutImaginary( ComplexNumber  &Z, ElemType b )
{
    //将复数虚部β值用参数 b 值替代。
    Z.imaginary = b;
    return OK;
}
//7. 复数的加运算
Status AddComplex( ComplexNumber  &Z, ElemType a, ElemType b )
{
    //将复数 Z 实部α增加值 a，虚部β增加值 b。
    Z.real      += a;
    Z.imaginary += b;
    return OK;
}
//8. 复数的减运算
Status SubComplex( ComplexNumber  &Z, ElemType a, ElemType b )
{
    //将复数 Z 实部α减去值 a，虚部β减去值 b。
    Z.real      -= a;
    Z.imaginary -= b;
    return OK;
 }
```

上述例 1-2 给出了例 1-1 抽象数据类型的详细表示与实现。基于此进行编程实现就非常容易了。说明一下，在本书中，函数中涉及的数据返回均通过引用型参数实现，函数本身的返回值 Status 用于对函数执行状态的判断，正常返回 OK 或者错误返回相应的错误代码。

1.4 算法与算法分析

1.4.1 算法

图灵奖获得者 Niklaus Wirth 有一句在计算机领域人尽皆知的名言：“ 算法+数据结构=程序”（ Algorithm + Data Structures = Programs ）。这个公式对计算机科学的影响程度足以类似物理学中爱因斯坦的 $E=mc^2$，一个公式展示了程序的本质。

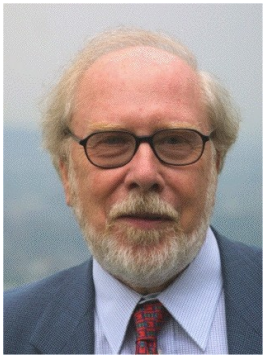

图 1.2 Niklaus Wirth

计算机程序设计基本上分为两个方面：一是数据组织，也称为数据的结构；二是求解问题的策略，也即算法。数据结构和算法是密切相关、不可分离的。一种数据结构的优劣是由实现其各种操作的算法体现的，对数据结构的分析实质上也就是对实现其各种操作的算法分析。

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。算法独立于具体的计算机语言，一般用伪代码、流程图等方式表示。

一个算法具有以下 5 个重要特性：

（1）有穷性。一个算法必须总是在执行有穷步之后结束，且每一步都可在有穷时间内完成。例如，循环的次数是有限的，不能陷入无限死循环。递归的执行有终止条件，不能无穷递归下去。

（2）确定性。算法中每条指令必须有确切的含义。在任何条件下，算法只有唯一的一条执行路径，也即对相同的输入只能得出相同的输出。

（3）可行性。 一个算法是可行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

（4）输入。一个算法有零个或多个的输入，这些输入取自于某个特定的对象的集合。 实际问题中，有些输入是通过人机交互

（5）输出。一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

同一问题，可由不同的算法来实现。例如在一个有序的序列中进行关键字查找，既可以用穷举法，从第一个元素开始，依次向后查找；也可以用二分法，每次和中间的关键字进行比较，每次舍去一半的查找空间。 可见，算法设计是个值得关注的事情。

通常，算法设计应考虑以下目标。

（1）正确性。算法应当满足具体问题的明确需求。对于正确的要求，有几个层次：i）程序不含语法错误，能正常运行；ii）程序对于常规的输入数据能够得到满足规格说明要求的结果；iii）程序对精心选择的，苛刻的数据输入能够得到满足要求的结果，特别是边界条件约束下的正确反馈；iv）程序对于一切合法的输入数据都能得到满足规格说明要求的结果。对于一个复杂工程问题，算法的正确性判断不是一件容易的事情，有专门的软件测试研究方向。本门课程中对算法的正确性要求达到第 iii 级即可。

（2）可读性。 算法描述和程序实现要便于人的阅读和交流。可读性好有助于人对算法的理解。特别是大型软件开发过程中，算法和程序的可读性和规范性有明确要求，便于软件开发和软件维护的团队合作。

（3）健壮性。当输入数据非法时，算法也能适当做出反应或进行处理，而不会产生莫名其妙的输出结果。对于出错的处理方法是返回一个表示错误或错误性质的值，而不是打印错误信息直接中止程序执行，以便在更高的抽象层次中处理。

（4）低执行时间与低存储量需求。 对于具体应用，算法应追求低执行时间与低存储量。在问题规模较小时，穷举法和二分法在执行时间上差别不大，但随着问题规模变大后，两者的差距将非常明显，下一小节将具体讨论。 作为计算机软件开发者，高效率的算法设计与实现是我们追求的目标，切忌因为算法设计时的“偷懒”而让计算机白白耗费运行时间、浪费计算资源。另外，执行时间和存储量两者往往难以同时取得最优，我们在算法设计时经常会“以空间换时间”，牺牲一定的存储空间来换取执行时间的大幅降低。

1.4.2 算法分析与度量

衡量算法优劣的主要指标有以下两个：

（1）**空间复杂度** $S(n)$（space complexity）。根据算法写成的程序在执行是占用存储单位的大小。一个程序出了需要存储空间来寄存本身所用指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。若输入数据所占空间只取决于问题本身和算法无关，则只需要分析除输入和程序之外的额外空间，否则应同时考虑输入

本身所需空间。 若额外空间相对于输入数据量来说是常数，则称此算法为**原地工作**。如果所占空间依赖于特定的输入，则应按最坏情况来分析。

（2）**时间复杂度**$T(n)$（time complexity）。根据算法写成的程序在执行时耗费时间的长度。时间复杂度主要与这些因素有关：算法的策略，问题的规模n，实现算法的编程语言，编译程序所产生的机器代码质量和机器执行指令的速度。算法的策略是决定时间复杂度最为关键的因素，例如前面提及的穷举法和二分法，不同的算法策略会导致程序执行时间的显著差异。在算法设计时，要充分考虑问题的规模n，时间复杂度是问题规模n的函数。另外三个因素则主要在算法实现时针对具体问题综合考虑。

如何来观察和计算算法的时间复杂度呢？

为估算算法的时间复杂度，需要统计算法中所有语句的执行频度。例 1-3 给出了两个n阶方阵乘积$C = A \times B$的算法及其语句执行频度的统计。

例 1-3 方阵乘法算法中语句执行频度统计

程序语句	语句执行次数
void MatrixMultiply(int A[ ][ ], int B[ ][ ] , int C[ ][ ], int n)	
{	
int i, j, k;	
for (i=0; i<n; i++)	$n+1$
{ for (j=0; j<n; j++)	$n(n+1)$
{	
C[i][j]=0;	n^2
for (k=0; k<n; k++)	$n^2(n+1)$
C[i][j] += A[i][k] * B[k][j];	n^3
}	
}	
}	
总计	$2n^3+3n^2+2n+1$

算法 1.1 方阵乘法

例 1-3 可见，执行频度最高的语句处于最深层循环内，也被称做问题的基本操作的原操作，它的重复执行次数和算法的执行时间成正比。

但对于一个大规模的系统而言，这样做涉及到很多细节，再加上算法描述中出现分支的情况，给出一个精确描述往往是困难的。 为此，退而求其次，设法估计算法复杂度的量级。对于算法时间复杂度，最重要的是其量级和趋势，这些是代价的主要部分，而代价函数的常量因子可以忽略不计。例如，可以认为$3n^2$和$100n^2$属于同一个量级，如果两个算法处理同样规模实例的代价分别为这两个函数，就认为它们的代价“差不多”。基于这些考虑，人们提出描述算法性质的“大O记法”。

定义 1.1 大O记法：对于单调的整数函数f,如果存在一个整数函数g和实常数$c>0$，使得对于足够大的n，总有$f(n) \le c \cdot g(n)$，就说函数g是f的一个渐近函数（忽略常量因子），记为$f(n)=O(g(n))$。可见，$f(n)=O(g(n))$说明在趋向无穷的极限意义下，函数f的增长速度受到函数g的约束。

将上述描述方式应用于算法的复杂度度量。假设存在函数g，使得算法 A 处理规模为n的问题实例所用的时间$T(n)=O(g(n))$，则称$O(g(n))$为算法 A 的渐近时间复杂度，简称时间复杂度。例 1-3 的时间复杂度则可表示为$O(n^3)$。算法的空间复杂度$S(n)$亦可类似定义。

在本门课程后续讨论中，常见的时间复杂度有以下几种说法：$O(1)$称为常量复杂度，$O(\log n)$称为对数复杂度，$O(n)$称为线性复杂度，$O(n^2)$称为平方复杂度，$O(2^n)$称为指数复杂度，等等。这些时间复杂度随问题规模n的变化趋势如图 1.3 所示。

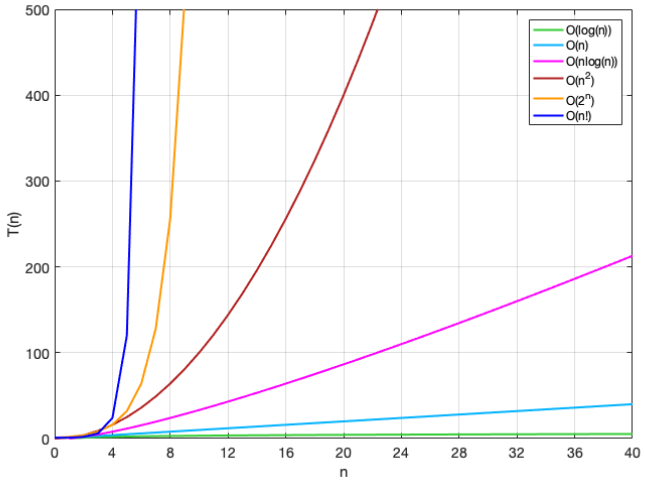

图 1.3 常见复杂度函数的增长情况对比

另外，算法的时间复杂度针对数据的不同表现形式，还可分为平均时间复杂度、最好情况下的时间复杂度和最坏情况下的时间复杂度。以起泡排序算法为例，如果初始输入数据的排列情况概率相等，则起泡排序算法的平均时间复杂度$T_{avg}(n)=O(n^2)$；如果输入数据是正序有序，则起泡排序算法会出现最好情况，时间复杂度为降为$O(n)$；如果输入数据是逆序有序，则起泡排序会出现最坏情况，时间复杂度也为$O(n^2)$。根据工程经验，我们对算法时间复杂度的估算一般是最坏情况下的时间复杂度。

1.5 如何学习数据结构课程

数据结构课程的学习关键是把握两条主线。

一条主线是如何理解和实现物理世界里各种联系在信息世界中的逻辑表示。物理世界中的万般联系都是形象而具体的，需要通过抽象建模，抓住这些联系的本质：线性、树形还是图状结构，从而能够变为信息世界中的逻辑表示。例如，生活中的各种排队可以抽象成线性结构中的队列，家族谱可以映射为树形结构，专业课程的先修后修关系又构成了有向的图结构。**抽象建模与表示**能力是工科类专业学生所必须具备的基本能力。不论是否是计算机类专业学生，在学习这门课程时，这条主线都必须牢牢把握。

另一条主线是如何在计算机中实现数据结构的存储和操作。存储的方式主要分为顺序存储和链式存储两大类。基本操作一般包括初始化、销毁操作，访问型操作和加工型操作三大类。这三类操作在两种不同的存储模式下算法的效率是有差异的，需要进行比较分析，总结各自的优势和不足。从而针对具体的现实问题，选择恰当的存储方式来保证操作执行的效率。**在比较分析基础上得到有效结论**是这条主线中的能力训练要求。

此外，算法设计与优化也是值得关注的问题，本书中的查找和排序部分，将展示算法设计与优化的思路。例如排序算法，如何从时间复杂度$O(n^2)$的经典算法开始，通过恰当结构的引入将算法时间复杂度降到$O(n\log n)$。再如查找算法，如何从时间复杂度$O(n)$的穷举算法开始，通过

对数据的排序约束和树形结构的引入，将时间复杂度降到$O(\log n)$，甚至在一些特殊规则的约束下能够接近$O(1)$。我们不仅要掌握经典算法本身，更要关注算法优化的策略和方向，以便于在解决实际问题时能够设计出高效的算法。

最后，从不同学习者的角度来讨论一下如何学习本门课程。

对于计算机类专业学生，本书所涉及的内容均需掌握。在此基础上，需要大量的训练，具备抽象建模能力，能够就实际问题，选择或设计恰当的数据结构；在编程实现过程中，能够基于问题的约束，在比较分析基础上选择恰当的存储方式和算法的实现方式，能够对算法的时间和空间复杂度有合理的分析，能够对算法的改进和优化有恰当的思路，从而为实际问题的解决提供高效的解决方案。

对于非计算机类专业学生，希望通过本门课程的学习达成两点目标：一是具备抽象建模能力，能够将物理世界中问题描述抽象建模从而在信息世界里进行合理表达；二是具备算法选择能力，能够根据输入数据的表现形式，数据结构的存储方式和基于此的算法效率特征，选择相应的数据结构和算法来高效解决实际问题。

1.6 小结

本章作为数据结构课程的概论，讨论了数据结构相关的概念及术语，特别具体阐述了抽象数据类型的表示与实现；给出了算法的定义、特性和算法设计的目标，算法的空间复杂度和时间复杂度，详细讨论了算法时间复杂度的计算方法；并对如何学习本门课程给出了相关的指引。

1.7 练习

（1）计算下列程序片段中 x=x+i 的执行频度。

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

（2）当 n 足够大时，对下列时间复杂度的增长趋势进行从小到大排序：

2^n, n^2, $\sqrt{n}$, $n!$, n, $n\log n$, $\log n$, $n^{\log n}$

（3）有一个顺序存储的整型数组，要从 n 个元素中查找特定元素 k。针对数组中数据为无序和有序的两种情况，试写出不同的算法，并分析各算法的时间复杂度。

（4）要统计全校大一年级 4000 人的高等数学考试成绩（百分制整数值）的最高分、最低分、平均分和中间值分（从高到低排名第 2000 名的分数），请设计一恰当的数据结构，并实现高效的算法。