

数据结构

深圳技术大学
大数据与互联网学院

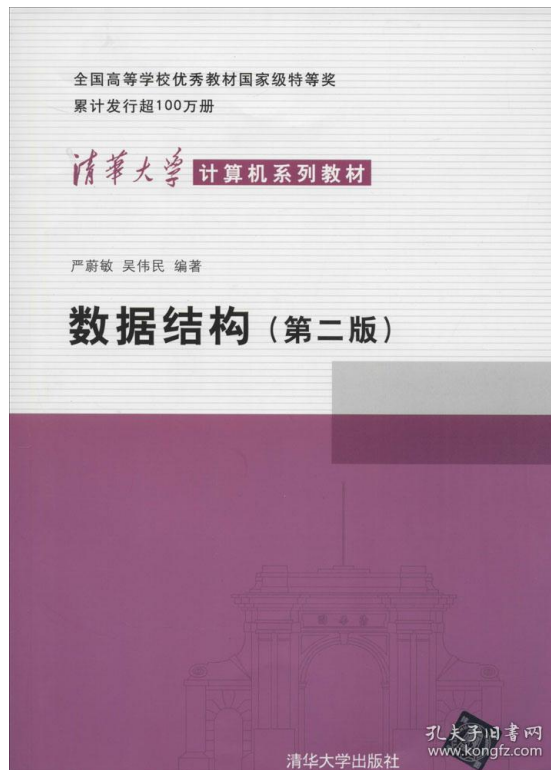
数据结构

⑩ 课程介绍

- ✧ 专业基础课、必修课、核心课程 学分**3+3+1**
- ✧ 计算机专业研究生考试的必考的四门专业课之一
- ✧ 介于数学、计算机硬件、计算机软件三者之间
- ✧ 不仅是一般程序设计的基础，也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
- ✧ 先修课程：**C/C++** 计算机导论
- ✧ 理论课+实验课 考勤+实验室签到
- ✧ 每周实验+上机测试+闭卷考试

数据结构

10 使用教材



10 课程使用程序语言：C/C++

一些其他的教学资源

10 浙大慕课:

✎ <http://mooc.study.163.com/course/ZJU-1000033001#/info>

10 清华慕课:

✎ <http://www.xuetangx.com/courses/course-v1:TsinghuaX+30240184X+sp/about>

10 Coursera:

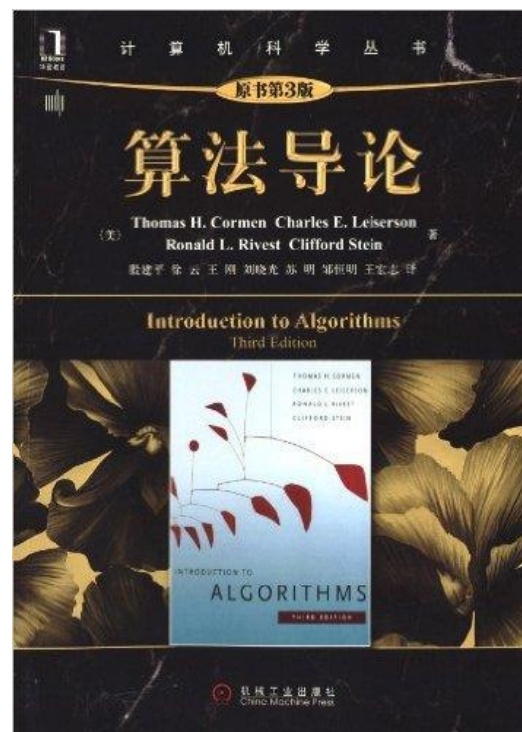
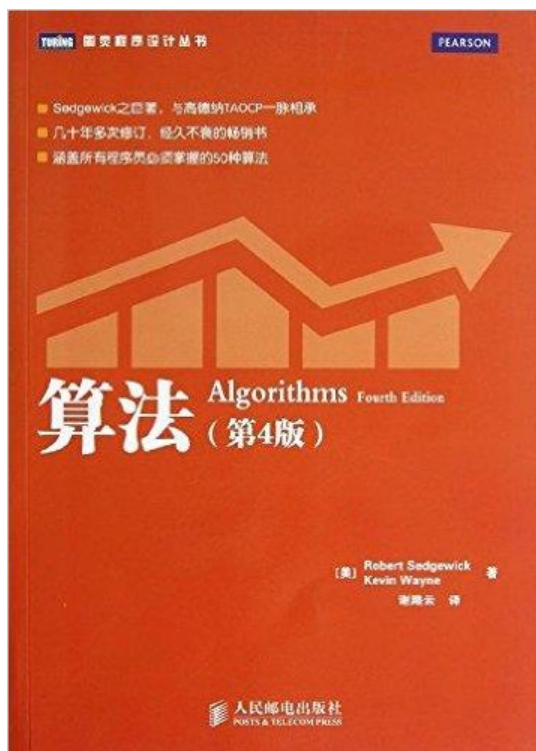
✎ 算法1: <https://www.coursera.org/learn/introduction-to-algorithms>

✎ 算法2: <https://www.coursera.org/learn/java-data-structures-algorithms>

10 知乎: 如何学习数据结构

✎ <http://www.zhihu.com/question/21318658>

- ⑩ 算法可视化网站
- ⑩ <http://visualgo.net>
- ⑩ <http://zh.visualgo.net>



数据结构

10 课程要求

- 上课不迟到，不缺勤，不定期点名，多练习
- 实验必须在实验室在限定时间完成，IP签到，考勤，否则计0分
- 所有上机实验独立完成，一旦发现抄袭，均计0分

10 教师联系方式

- 姓名 龙梓
- 手机 17688956461
- 邮箱 longzi@sztu.edu.cn
- 微信群（答疑+ppt分享）



第一章 绪论

- 1.1 什么是数据结构**
- 1.2 基本概念和术语**
- 1.3 抽象数据类型的表现和实现**
- 1.4 算法和算法分析**

[例1.1] 该如何摆放书，才能让读者很方便地找到你手里这本《数据结构》？



【分析】

[方法1] 随便放——任何时候有新书进来，哪里有空就把书插到哪里。
放书方便，但查找效率极低！

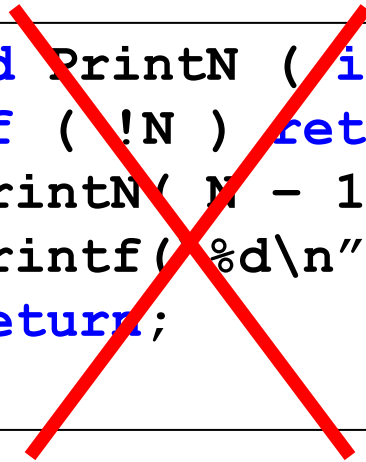
[方法2] 按照书名的**拼音字母顺序**排放。
查找方便，但插入新书很困难！

[方法3] 把书架**划分成几块区域**，每块区域指定摆放某种类别的图书；
在每种类别内，按照书名的拼音字母顺序排放。

查找、插入方便，但每种类型的书不知道有多少本，有可能造成空间的浪费！

[例1.2] 写程序实现一个函数**PrintN**，使得传入一个正整数为**N**的参数后，能顺序打印从**1**到**N**的全部正整数。

```
void PrintN ( int N )
{ int i;
  for ( i=1; i<=N; i++ )
    printf("%d\n", i );
  return;
}
```



```
void PrintN ( int N )
{ if ( !N ) return;
  PrintN( N - 1 );
  printf("%d\n", N );
  return;
}
```

考虑当**N**足够大时，会出现什么问题！

[例1.3] 多项式的标准表达式可以写为:

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

现给定一个多项式的阶数 n ，并将全体系数存放在数组 $a[]$ 里。请写程序计算这个多项式在**给定点 x 处的值**。

[方法2] 秦九韶法

$$f(x) = a_0 + x (a_1 + x (a_2 + \dots + x (a_n) \dots))$$

double f(int n, double a[], double x)

{ /* 计算阶数为n，系数为a[0]...a[n]的多项式在x点的值 */

int i;

double p = a[n];

for (i=n; i>0; i--)

p = a[i-1] + x*p;

return p;

}

数据结构

- 即使解决一个非常简单的问题，往往也有多种方法，且不同方法之间的效率可能相差甚远
- 解决问题方法的效率
 - 跟数据的组织方式有关（如例1.1）
 - 跟空间的利用效率有关（如例1.2）
 - 跟算法的巧妙程度有关（如例1.3）

“数据结构是计算机中存储、组织数据的方式。精心选择的数据结构可以带来最优效率的算法。”

1.1 数据结构

- 数据结构是一门研究非数值计算的程序设计问题中, 计算机的操作对象以及它们之间的关系和操作等的学科。
- 数据结构是研究数据元素之间抽象化的相互关系和这种关系在计算机中的存储表示（即所谓数据的逻辑结构和物理结构），并对这种结构定义相适应的运算，设计出相应的算法，而且确保经过这些运算后得到的新结构仍然是原来的结构类型。
- 数据结构与相应算法是为了在解决具体某个问题时，让计算机的处理过程实现空间与时间上的最优

1.2 基本概念和术语

一. 数据

- 是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中，被计算机程序识别和处理的符号的集合。

- ☐ 数值性数据
- ☐ 非数值性数据

1.2 基本概念和术语

二. 数据元素 (**Data Element**)

- 数据的基本单位。在计算机程序中常作为一个整体进行考虑和处理。
- 有时一个数据元素可以由若干数据项(**Data Item**)组成（此时数据元素被称为记录）
- 数据元素又称为元素、结点、记录

1.2 基本概念和术语

三. 数据项 (**Data Item**)

- 具有独立含义的最小标识单位

学号	姓名	学院	专业
----	----	----	----

1.2 基本概念和术语

四、数据对象（**Data Object**）

- 具有相同性质的数据元素的集合。

整数数据对象

$$N = \{0, \pm 1, \pm 2, \dots\}$$

字母字符数据对象

$$C = \{ 'A', 'B', 'C', \dots, 'F' \}。$$

1.2 基本概念和术语

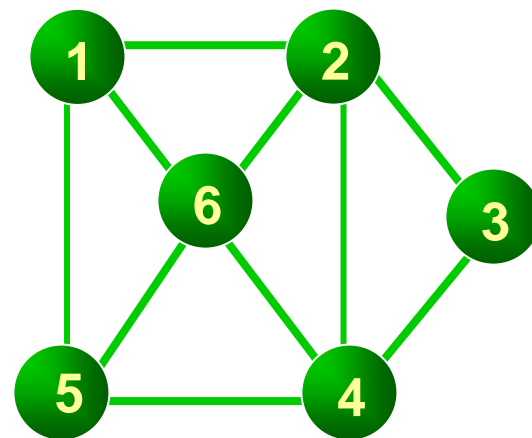
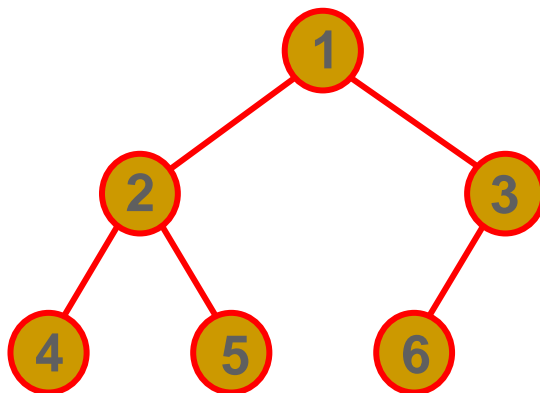
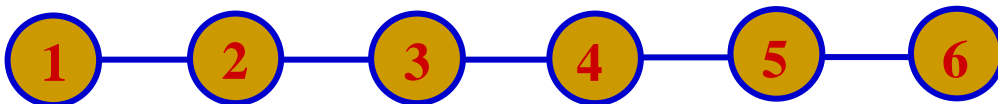
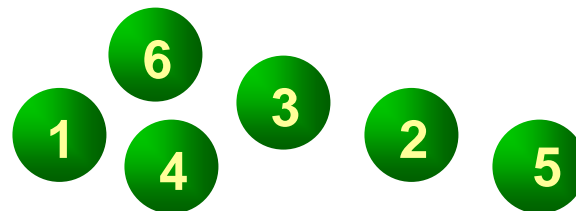
五. 结构 (Structure)

■ 结构是元素之间的关系

- ☐ 空间位置关系
- ☐ 相互作用和依赖关系

■ 四种结构

- ☐ 集合结构
- ☐ 线性结构
- ☐ 树形结构
- ☐ 图形结构



1.2 基本概念和术语

六. 数据结构（Data Structure）

- 形式定义：数据结构是一个二元组，记为：

$$\text{Data_Structure} = \{D, S\}$$

D是某一数据对象，也记作K

S是该对象中所有数据成员之间的关系有限集合，也可记作R。

1.2 基本概念和术语

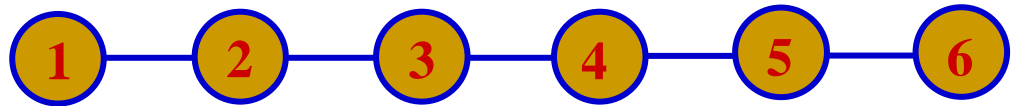
六. 数据结构 (Data Structure)

■ 线性数据结构举例

$L = \{K, R\}$

$K = \{1, 2, 3, 4, 5, 6\}$

$R = \{\langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 4 \rangle, \langle 4, 5 \rangle, \langle 5, 6 \rangle\}$



1.2 基本概念和术语

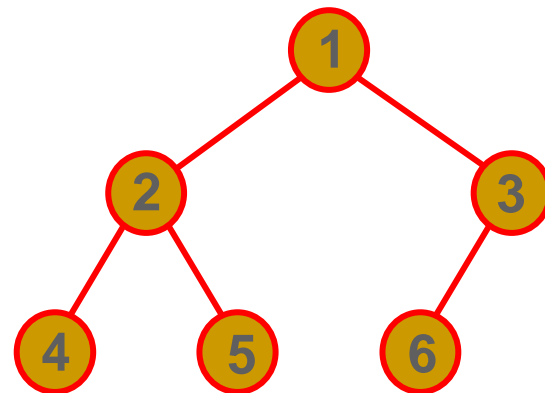
六. 数据结构 (Data Structure)

■ 树形数据结构举例

$$T = \{K, R\}$$

$$K = \{1, 2, 3, 4, 5, 6\}$$

$$R = \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 4 \rangle, \langle 2, 5 \rangle, \langle 3, 6 \rangle\}$$



1.2 基本概念和术语

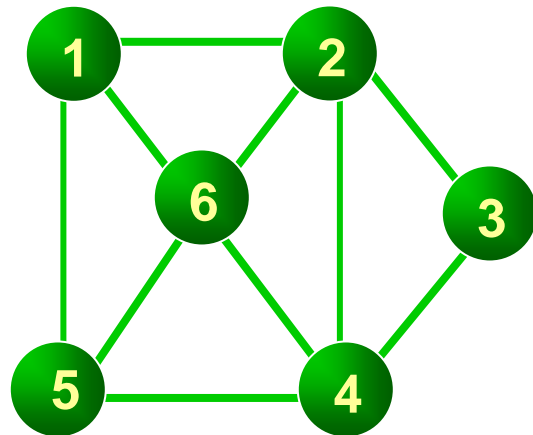
六. 数据结构 (Data Structure)

■ 图形数据结构举例

$$G = \{K, R\}$$

$$K = \{1, 2, 3, 4, 5, 6\}$$

$$R = \{(1, 2), (1, 5), (1, 6), (2, 3), (2, 4), \\ (2, 6), (3, 4), (4, 5), (4, 6), (5, 6)\}$$



1.2 基本概念和术语

七. 数据结构应用举例

■ 线性数据结构

数据结构学生选课名单（部分）

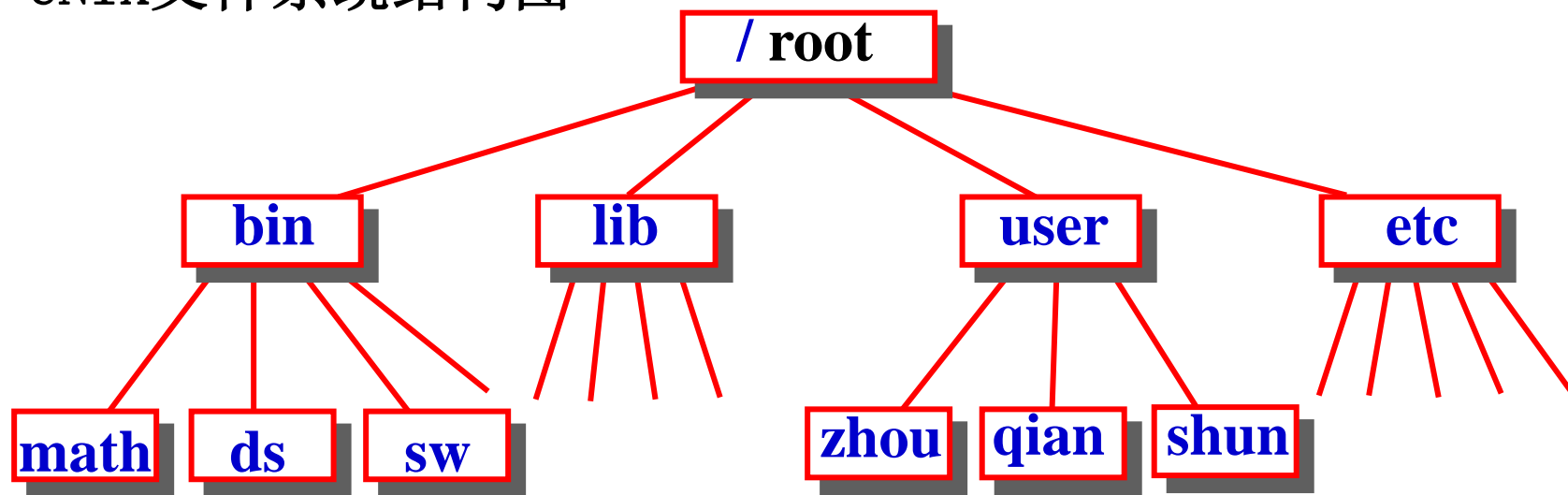
姓名	学号	学院	专业
刘泰源	2006044060	计算机与软件学院	软件工程
温国乾	2006131006	计算机与软件学院	软件工程
强健	2006131043	计算机与软件学院	软件工程
杨海波	2006131047	计算机与软件学院	软件工程
李耀东	2006131051	计算机与软件学院	软件工程

1.2 基本概念和术语

七. 数据结构应用举例

■ 树形数据结构

UNIX文件系统结构图

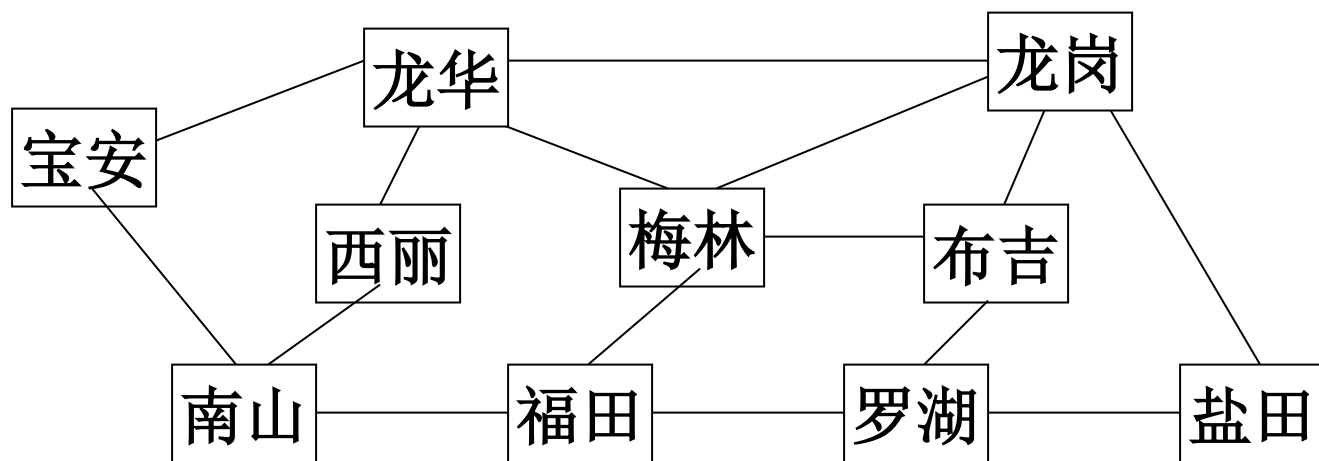


1.2 基本概念和术语

七. 数据结构应用举例

■ 图形数据结构举例

深圳城市交通示意图



1.2 基本概念和术语

七. 逻辑结构与物理结构

- 逻辑结构描述数据元素之间的关系
 - 线性结构，线性表（表，栈，队列，串等）
 - 非线性结构，树、图等
- 物理结构是数据结构在计算机中的表示(或映象)
 - 顺序存储表示(例如C语言中一维数组表示)
 - 链接存储表示(例如C语言中的指针表示)
 - 索引存储表示
 - 散列存储表示

1.3 数据类型

一. 数据类型的概念

- 数据类型是一个值的集合和定义在这个值集上的一组操作的总称

如C/C++语言中的整型变量(int)，其值集为某个区间上的整数，定义在其上的操作为 $+$ ， $-$ ， \times ， $/$ 等

- 抽象数据类型 (**Abstract Data Type**，简称**ADT**)，是指一个数学模型以及定义在该模型上的一组操作
- 抽象数据类型和数据类型实质上是一个概念

1.3 数据类型

二. 三种数据类型

■ 原子数据类型

- 不可分解的数据类型，例如C语言中的整型(int)，实型(float)，字符型(char)，指针类型(*)和空类型(void)等数据类型

■ 固定聚合类型

- 由确定数目的成分按某种结构组成，例如复数由两个实数按确定次序组成

■ 可变聚合类型

- 组成的成分数目不确定

■ 固定聚合和可变聚合又统称为结构类型

1.3 数据类型

二. 抽象数据类型

■ 描述

抽象数据类型 = 数据结构 + 定义在此数据结构上的一组操作

■ 抽象数据类型可用 (D, S, P) 三元组表示

- D是数据对象
- S是D上的关系集
- P是对D的基本操作集

1.3 数据类型

三. 抽象数据类型的定义

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(函数)的定义〉

} ADT 抽象数据类型名

1.3 数据类型

三. ADT的定义举例

ADT Triplet {

数据对象: $D = \{e1, e2, e3 \mid e1, e2, e3 \in \text{ElemSet}\}$

数据关系: $R = \{\langle e1, e2 \rangle, \langle e2, e3 \rangle\}$

基本操作: $\text{Max}(T, \&e)$

初始条件: 三元组T已存在。

操作结果: 用e返回T的3个元素中的最大值。

$\text{Min}(T, \&e)$

初始条件: 三元组T已存在。

操作结果: 用e返回T的3个元素中的最小值。

} ADT Triplet

[例1.4] “矩阵”的抽象数据类型定义

类型名称: Matrix

数据对象集: 一个 $M \times N$ 的矩阵A。

操作集: 对于任意矩阵 $A, B, C \in \text{Matrix}$, 以及正整数 i, j, M, N , 以下仅列出几项有代表性的操作。

- 1) **Matrix Create(int M, int N):** 返回一个 $M \times N$ 的空矩阵;
- 2) **int GetMaxRow(Matrix A):** 返回矩阵A的总行数;
- 3) **int GetMaxCol(Matrix A):** 返回矩阵A的总列数;
- 4) **ElementType GetEntry(Matrix A, int i, int j):** 返回矩阵A的第 i 行、第 j 列的元素;
- 5) **Matrix Add(Matrix A, Matrix B):** 如果A和B的行、列数一致, 则返回矩阵 $C=A+B$, 否则返回错误标志;
- 6) **Matrix Multiply(Matrix A, Matrix B):** 如果A的列数等于B的行数, 则返回矩阵 $C=AB$, 否则返回错误标志;
- 7)

1.4 算法和算法分析

一. 算法 (Algorithm)

■ 算法是对特定问题求解步骤的一种描述，是一有限长的操作序列

■ 算法特性

- 有穷性：算法在执行有穷步后能结束
- 确定性：每步定义都是确切的、无歧义的
- 可行性：算法描述的操作都是可以通过已实现的基本运算经过有限次来实现的
- 输入：有0个或多个输入
- 输出：有一个或多个输出

1.4 算法和算法分析

一. 算法 (Algorithm)

■ 算法举例

□ 问题：递增排序

□ 解决方案：逐个选择最小数据，即选择排序

□ 算法框架：

```
for ( int i = 0; i < n-1; i++ ) {      //n-1次  
    从a[i]检查到a[n-1]，找到最小数；  
    若最小整数在a[k]，交换a[i]与a[k]；  
}
```

1.4 算法和算法分析

二. 算法要求

- 正确性：满足具体问题的需求
- 可读性：便于理解和修改
- 健壮性：当输入数据非法时，也能适当反应
- 效率高：执行时间少
- 空间省：执行中需要的最大存储空间

1.4 算法和算法分析

三. 时间复杂度

- 衡量算法的效率，主要依据算法执行所需要的时间，即时间复杂度
- 事后统计法：计算算法开始时间与完成时间差值
- 事前统计法：依据算法选用何种策略及问题的规模 n ，是常用的方法

❖ 我们经常关注下面两种复杂度：

➤ 最坏情况复杂度： $T_{\text{worst}}(n)$

➤ 平均复杂度： $T_{\text{avg}}(n)$

➤ 显然： $T_{\text{avg}}(n) \leq T_{\text{worst}}(n)$ 。

对 $T_{\text{worst}}(n)$ 的分析往往比对 $T_{\text{avg}}(n)$ 的分析容易。

❖ 如果：

程序A执行了 $(3n+4)$ 步，

程序B执行了 $(2n+2)$ 步，

A一定比B慢吗？

❖ No!

❖ Why?

❖ 如何来“度量”一个算法的时间复杂度呢？

- 首先，它应该与运行该算法的**机器和编译器无关**；
- 其次，它应该与要解决的问题的**规模 n 有关**；
(有时，描述一个问题的规模需要多个参数)
- 再次，它应该与算法的“**1步**”执行需要的**工作量无关**！
- 所以，在描述算法的时间性能时，人们只考虑**宏观渐近性质**，即当输入**问题规模 n** “充分大”时，观察算法复杂度随着 n 的“**增长趋势**”：
当变量 n 不断增加时，解决问题所需要的时间的增长关系。

❖ 比如：线性增长 $T(n) = c \cdot n$

即问题规模 n 增长到**2倍、3倍.....**时，解决问题所需要的时间 $T(n)$ 也是增长到**2倍、3倍.....**（与 c 无关）

❖ 平方增长： $T(n) = c \cdot n^2$

即问题规模 n 增长到**2倍、3倍.....**时，解决问题所需要的时间 $T(n)$ 增长到**4倍、9倍.....**（与 c 无关）

❖ 引入下面几种数学符号：

[定义1.1] $T(n) = O(f(n))$ 表示存在常数 $c > 0, n_0 > 0$,
使得当 $n \geq n_0$ 时有 $T(n) \leq c f(n)$

➤ 例1.3 中秦九韶算法的时间复杂度是 $O(n)$,
而简单直接法的时间复杂度是 $O(n^2)$ 。

[定义1.2] $T(n) = \Omega(g(n))$ 表示存在常数 $c > 0, n_0 > 0$,
使得当 $n \geq n_0$ 时有 $T(n) \geq c g(n)$

[定义1.3] $T(n) = \Theta(h(n))$ 表示
 $T(n) = O(h(n))$ 同时 $T(n) = \Omega(h(n))$

1.4 算法和算法分析

三. 时间复杂度

- 时间复杂度是问题规模 n 的函数 $f(n)$ ，即：

$$T(n) = O(f(n))$$

- 一般地，时间复杂度用算法中最深层循环内的语句中的原操作的重复执行次数表示

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

❖ 对给定的算法做渐进分析时，有几个小窍门：

(1) 若干层嵌套循环的时间复杂度等于各层循环次数的乘积再乘以循环体代码的复杂度。

例如下列2层嵌套循环的复杂度是 $O(N^2)$ ：

```
for ( i=0; i<N; i++ )
    for ( j=0; j<N; j++ )
        { x = y*x + z; k++; }
```

(2) if-else 结构的复杂度取决于if的条件判断复杂度和两个分支部分的复杂度，总体复杂度取三者中最大。即对结构：

```
if (P1) /* P1的复杂度为  $O(f_1)$  */
```

```
    P2; /* P2的复杂度为  $O(f_2)$  */
```

```
else
```

```
    P3; /* P3的复杂度为  $O(f_3)$  */
```

总复杂度为 $\max(O(f_1), O(f_2), O(f_3))$ 。

1.4 算法和算法分析

三. 时间复杂度

- a, b, c三个算法中, “ $y *= x;$ ” 程序段的时间复杂度分别为 $O(1)$, $O(n)$, $O(n^2)$, 分别称为常量阶, 线性阶, 平方阶

a. $\{y *= x;\}$

b. **for** ($i=1$; $i \leq n$; $i++$) $\{y *= x;\}$

c. **for** ($j=1$; $j \leq n$; $j++$)
 for ($i=1$; $i \leq n$; $i++$) $\{ y *= x; \}$

1.4 算法和算法分析

三. 时间复杂度

- d, e算法中, “ $y *= x;$ ” 程序段的时间复杂度都是 $O(\log_2 n)$ 称为对数阶

d. **for** ($i=1$; $i \leq n$; $i *= 2$) { $y *= x$; }

e. **for** ($i=n$; $i \geq 1$; $i /= 2$) { $y *= x$; }

估算下列程序段所代表的算法的时间复杂性的上界和下界:

(1) **for** ($i=1$; $i \leq n$; $i++$) {
 $x=x+y$; $y=x+y$;
}

(2) **while** ($n > 1$) {
 if ($n \% 2$) $n=n-1$;
 else $n=n/2$;

1.4 算法和算法分析

我们希望随着问题规模时间的增长复杂度是趋于稳定地上升,但上升幅度不能太大

三. 时间复杂度

■ 时间复杂度:

- 常量阶 [$O(1)$]
- 线性阶 [$O(n)$]
- 平方阶 [$O(n^2)$]
- 对数阶 [$O(\log n)$]
- 排列阶 [$O(n!)$]
- 指数阶 [$O(2^n)$] 等, 是相对于问题规模 n 的增长率的表示方法

■ 排列阶、指数阶随问题规模增长过快, 一般不宜使用

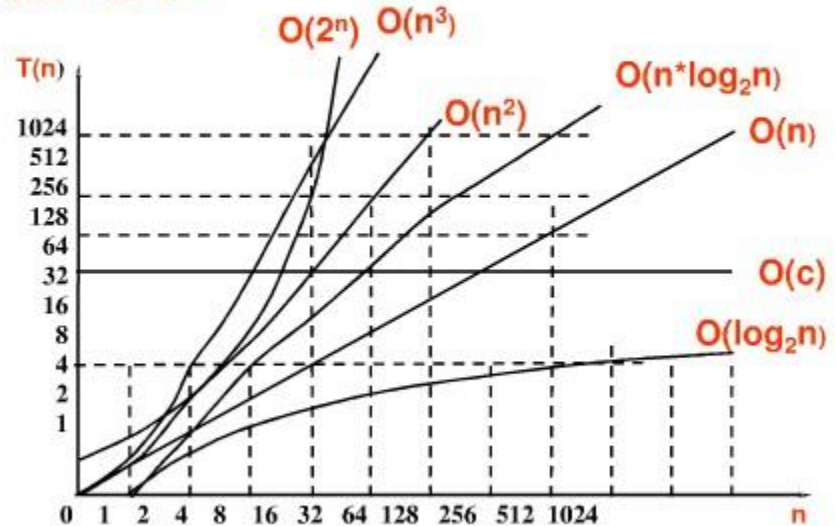
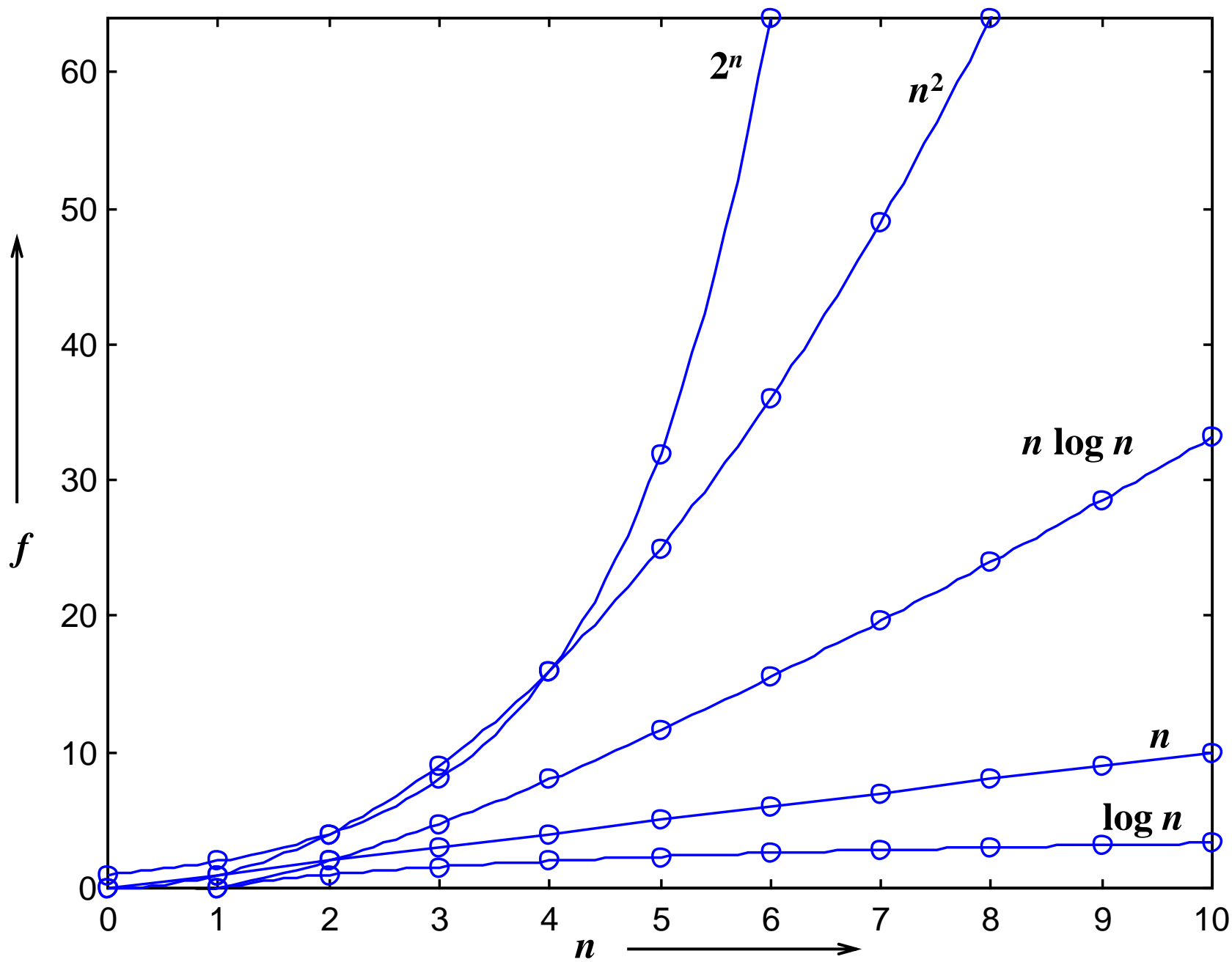


图1-7 常见的 $T(n)$ 随 n 变化的增长率

❖ 常用函数增长表

	输入规模 n					
函数	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log_2 n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log_2 n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40320	2092278988800	26313×10^{33}



每秒10亿指令计算机的运行时间表

n	$f(n)=n$	$n \log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01 μ s	.03 μ s	.1 μ s	1 μ s	10 μ s	10sec	1 μ s
20	.02 μ s	.09 μ s	.4 μ s	8 μ s	160 μ s	2.84hr	1ms
30	.03 μ s	.15 μ s	.9 μ s	27 μ s	810 μ s	6.83d	1sec
40	.04 μ s	.21 μ s	1.6 μ s	64 μ s	2.56ms	121.36d	18.3min
50	.05 μ s	.28 μ s	2.5 μ s	125 μ s	6.25ms	3.1yr	13d
100	.10 μ s	.66 μ s	10 μ s	1ms	100ms	3171yr	4×10^{13} yr
1,000	1.00 μ s	9.96 μ s	1ms	1sec	16.67min	3.17×10^{13} yr	32×10^{283} yr
10,000	10 μ s	130.03 μ s	100ms	16.67min	115.7d	3.17×10^{23} yr	
100,000	100 μ s	1.66ms	10sec	11.57d	3171yr	3.17×10^{33} yr	
1,000,000	1.0ms	19.92ms	16.67min	31.71yr	3.17×10^7 yr	3.17×10^{43} yr	

 μ s = 微秒 = 10^{-6} 秒ms = 毫秒 = 10^{-3} 秒 sec = 秒

min = 分

hr = 小时

yr = 年

d = 天

1.4 算法和算法分析

三. 时间复杂度

- 如果算法的执行有多种可能的操作顺序，则求其平均时间复杂度
- 如果无法求取平均时间复杂度，则采用最坏情况下的时间复杂度
- 时间复杂度是衡量算法好坏的一个最重要的标准

数组长度 length=6

数组元素为：17, 3, 25, 14, 20, 9

执行过程：（每次将最小元素推至最前）

初始状态：	17	3	25	14	20	9
第一趟排序：	3	17	9	25	14	20
第二趟排序：	3	9	17	14	25	20
第三趟排序：	3	9	14	17	20	25
第四趟排序：	3	9	14	17	20	25
第五趟排序：	3	9	14	17	20	25

1.4 算法和算法分析

四. 空间复杂度

- 空间复杂度指算法执行时，所需要存储空间的量度，它也是问题规模的函数，即：

$$S(n) = O(f(n))$$

Take Home Message

- ⑩ 数据结构是为了在解决问题时让处理过程实现空间与时间上的最优
 - ⑩ 抽象数据类型可用 (D, S, P) 三元组表示
 - ⑩ 时间复杂度
 - ∞ 上界 O ，下界 Ω ，渐进 Θ
 - ⑩ 空间复杂度
-