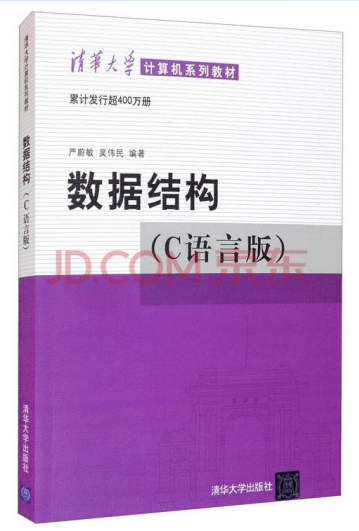
数据结构

智慧树课程号：k4236737

群名称：数据结构2022-2023

群　号：570120161

课程介绍



- 64学时，其中包括24学时上机实验
- 授课教师

  1-8周：李坤　　9-16周：金弟
- 办公地点：55B422（李坤）
- E-mail：lik@tju.edu.cn（李坤）

  jindi@tju.edu.cn （金弟）
- 考核方式：

平时出勤与互动10% + 平时作业10% + 上机30% + 闭卷考试40%

实验安排

- **实验时间：周五13:30-15:00 47教学楼第7机房(如遇疫情、考试等，转为线上)**

周五上机所在周的周二课取消（相当于调换）：

3月03日（2月28日课取消）	**4月28日**（4月25日课取消）
3月17日（3月14日课取消）	**5月05日**（5月02日课取消）
3月24日（3月21日课取消）	**5月19日**（5月16日课取消）
3月31日（3月28日课取消）	**5月26日**（5月23日课取消）
4月07日（4月04日课取消）	**6月02日**（5月30日课取消）
4月14日（4月11日课取消）	**6月09日**（6月06日课取消）

标蓝为重点关注日期；

3月17日大家有考试的话，改成线上上机（商量个时间）

数据结构的重要性

- 数据结构是世界观，程序语言的学习是具体方法
- 不仅仅是会编程，而是学习编程的原理
- 编程的上层设计，基本功

✓ 总之好好学数据结构就对了。数据结构就相当于：我塞牙了，就要用到牙签这“数据结构”，当然你用指甲也行，只不过“性能”没那么好；我要拧螺母，肯定用扳手这个“数据结构”，当然你用钳子也行，只不过也没那么好用。学习数据结构，就是为了了解以后在IT行业里搬砖需要用到什么工具，这些工具有什么利弊，应用于什么场景。以后用的过程中，你会发现这些基础的“工具”也存在着一些缺陷，你不满足于此工具，此时，你就开始自己在这些数据结构的基础上加以改造，这就叫做自定义数据结构。而且，你以后还会造出很多其他应用于实际场景的数据结构。你用这些数据结构去造轮子，不知不觉，你成了又一个轮子哥

数据结构是计算机科学的最核心课程

第一章 绪论

- 什么是数据结构
- 基本概念和术语
- 抽象数据类型
- 算法分析
- 性能分析与度量

基本概念

数据结构?

计算机中的“人际社会关系”

数据结构：相互之间

存在一种或多种特定关系的数据元素的集合

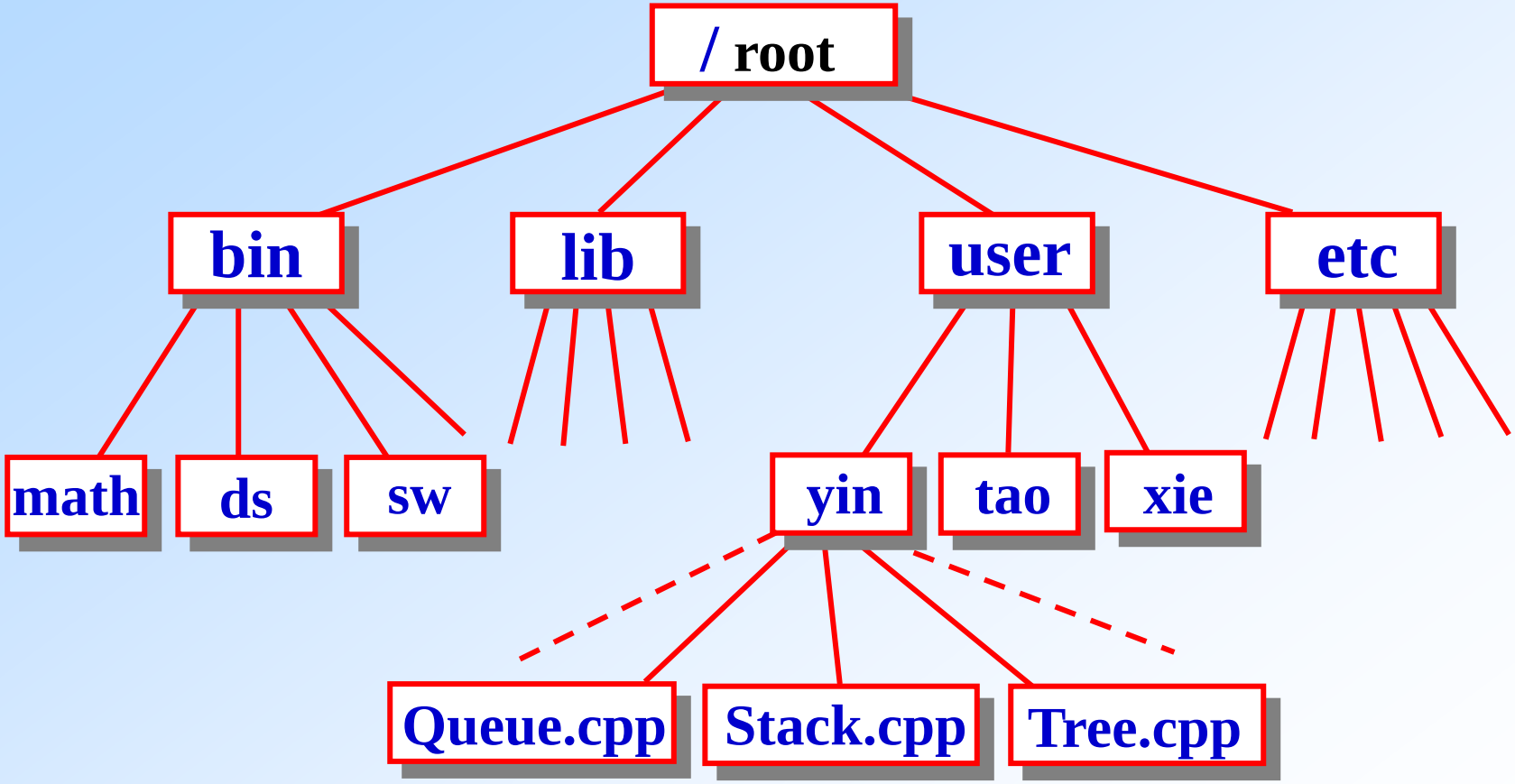
学生成绩表格

	学 号	姓 名	数据结构	系统结构	数学
1	20001	刘扬	89	69	67
2	20002	李平	70	83	89
3	20003	王方	86	81	78
4	20004	张策	69	69	78
5	20005	董立	79	89	68
6	20006	谢平	80	88	79
7	20007	高月	81	81	80
8	20008	刘平	89	85	87
9	20009	好园	86	80	84

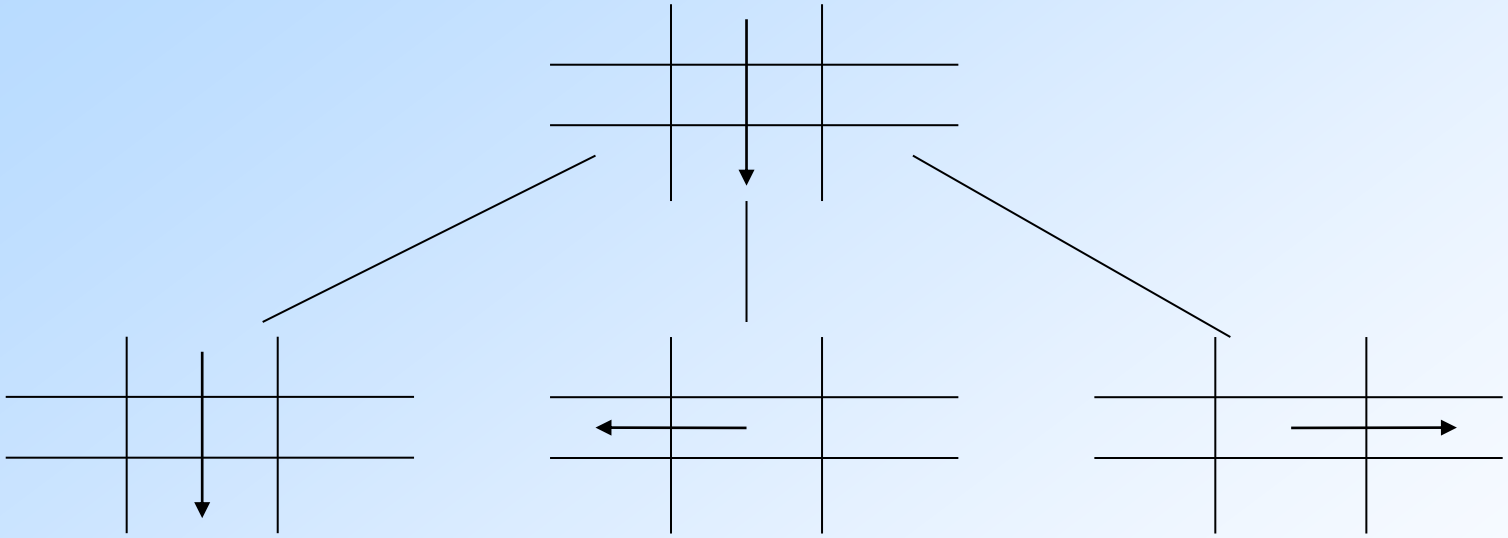
选课单

学号	课程号	时间	成绩
20001	DS2000	2001,2	78
	SX2000	2000,9	87
20002	ART2000	2002,2	68
	DS2000	2001,2	90
20003	SX2000	2000,9	87
	DS2000	2001,2	78
20004	SX2000	2000,9	89
	ART2000	2002,2	76

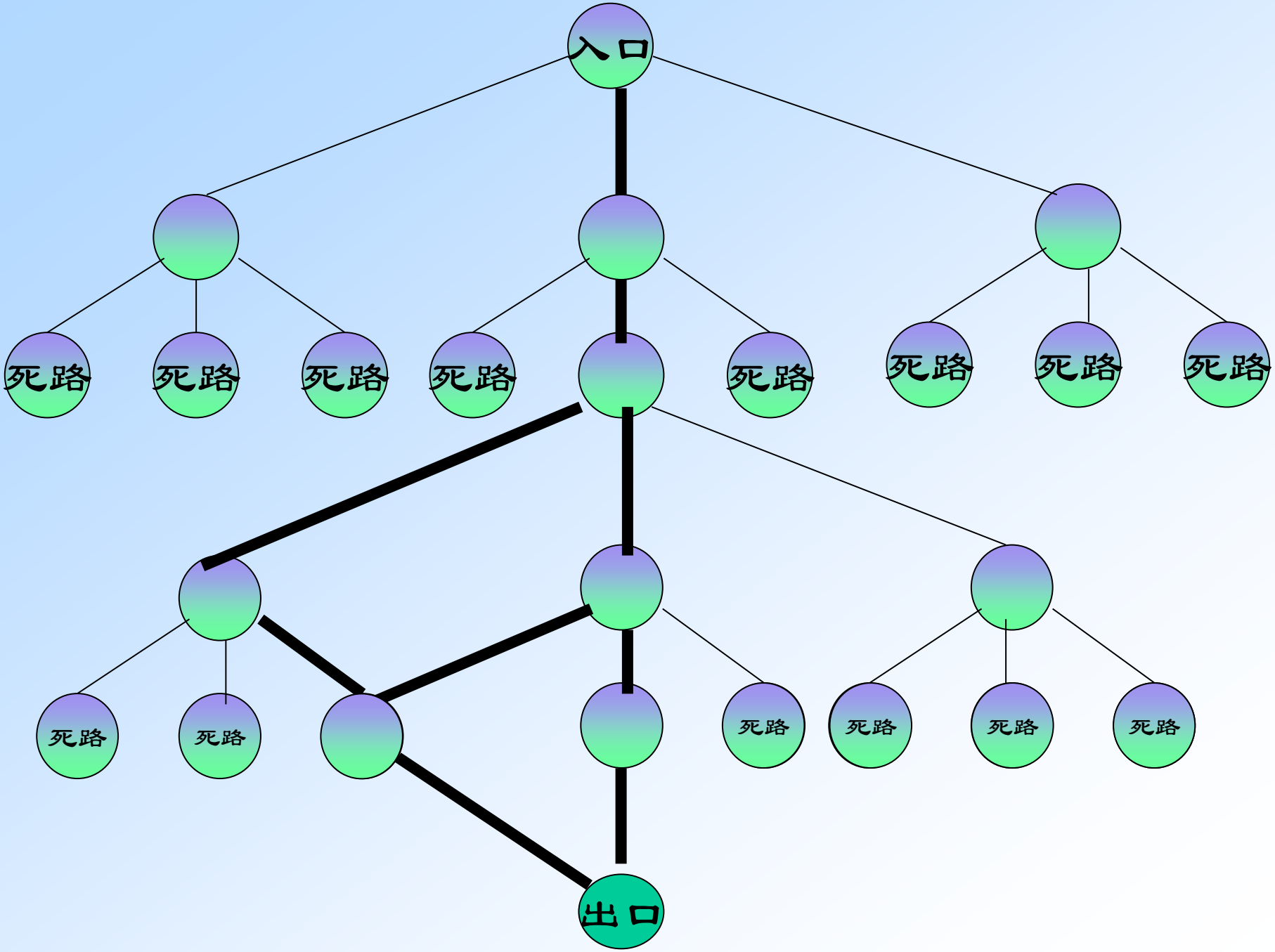
UNIX文件系统结构图



- 例如：在迷宫问题中，计算机之所以能够找到迷宫的出口，是因为人们已将搜索出口的策略事先存入计算机中．在迷宫中，每走到一处，接下来可走的通路有三条，如图：



计算机处理的这类对象之间通常不存在线性关系，若将从迷宫入口处到出口的过程中所有可能的通路都画出来，则可以得到一棵倒长的“树”．“树根”是迷宫入口，“树叶”是出口或死路．“树”可以是某些非数值计算问题的数学模型．


入口
死路
死路
死路
死路
死路
死路
死路
死路
死路
死路
死路
死路
死路
出口

- 在应用程序中涉及到各种各样的数据，如何在计算机中组织、存储、传递数据，需要讨论它们的归类及它们之间的关系，从而建立相应的数据结构，依此实现软件功能。
- 综上，描述这类非数值计算问题的数学模型不是数学方程,而是树、表和图之类的数据结构。
- 因此从广义上讲，数据结构描述现实世界实体的数学模型及其上的操作在计算机中的表示和实现。

基本概念和术语

- 数据结构：相互之间存在一种或多种特定关系的数据元素的集合

- **数据**：对信息的一种**符号**表示，指所有能输入到计算机中进行程序处理的符号的总称
  - ✓ 可以输入计算机，能被计算机处理，包括整型、实型等数值类型，声音、图像、视频等作为符号类型……
- **数据元素**：数据的基本单位，通常作为整体处理，也被称为记录。一个数据元素可由若干数据项组成
- **数据项**：组成数据元素的具有独立含义的最小单位

数据元素		
数据项1	数据项2	数据项3

人					
眼睛	手	脚	嘴巴	耳	…

基本概念和术语

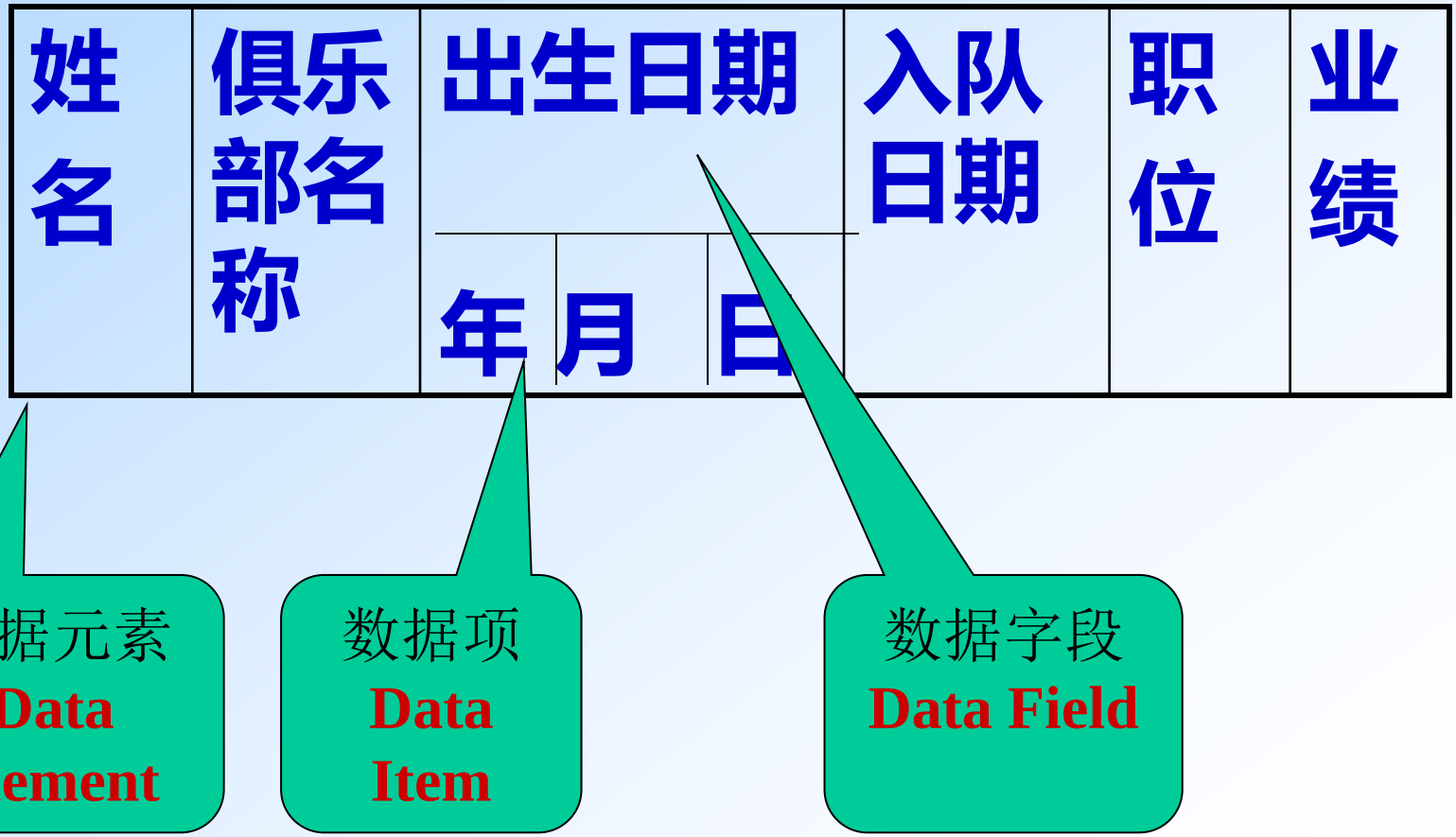
数据(Data)

- 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（整数、定点数、浮点数）
  - 非数值性数据（文字数据）

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。数据项是具有独立含义的最小标识单位。
- 数据元素又称为元素、结点、记录

数据项(Data Item)



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

    $N = \{ 0, \pm1, \pm2, \ldots \}$
  - 字母字符数据对象

    $C = \{ \text{‘A’}, \text{‘B’}, \text{‘C’}, \ldots \text{‘Z’} \}$

数据结构（Data Structure）

- **形式定义：**

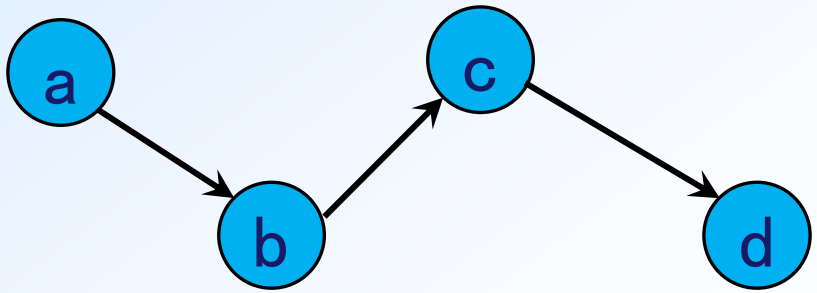
  某一数据对象的所有数据成员之间的关系。

  记为：

  Data_Structure = {D, S}

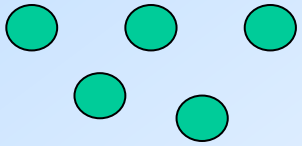
  其中，D 是某一数据对象， S 是该对象中所有数据成员之间的关系的有限集合。

- 例：A=(D,R), D={a,b,c,d}; R={<a,b>,<b,c>,<c,d>}

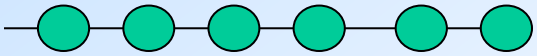


四个基本结构

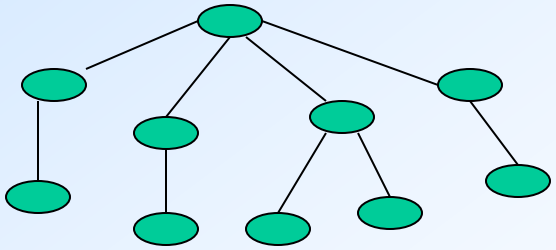
- 集合



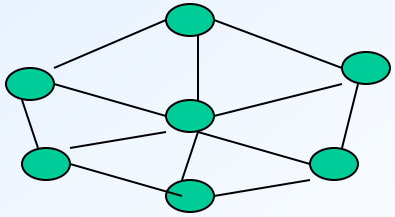
- 线性结构



- 树形结构



- 网状结构



数据的逻辑结构

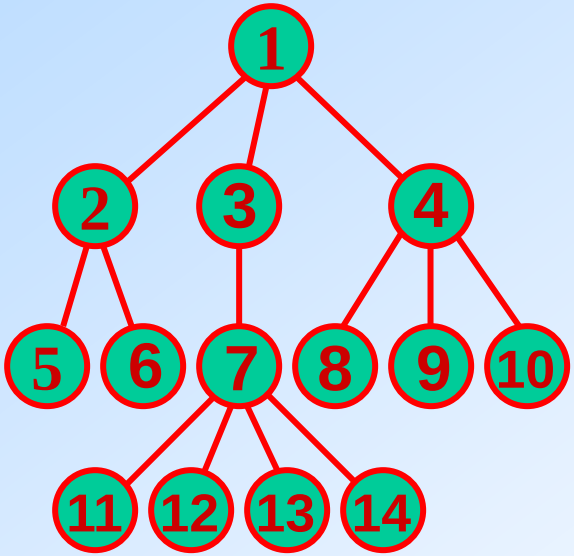
- 从逻辑关系上描述数据，与数据的存储无关；
- 从具体问题抽象出来的数据模型；
- 与数据元素本身的形式、内容无关；
- 与数据元素的相对位置无关。

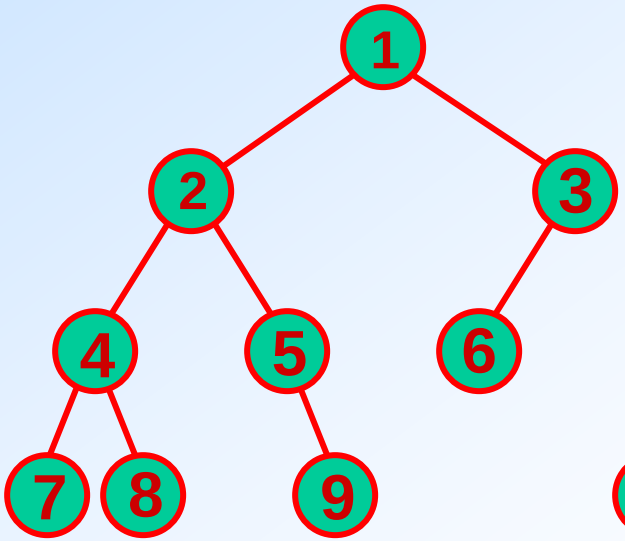
数据的逻辑结构分类

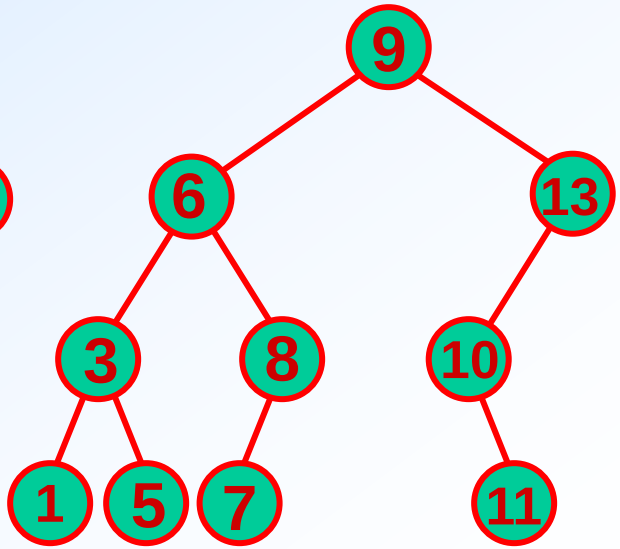
- 线性结构
  - 线性表
- 非线性结构
  - 树
  - 图（或网络）

线性结构

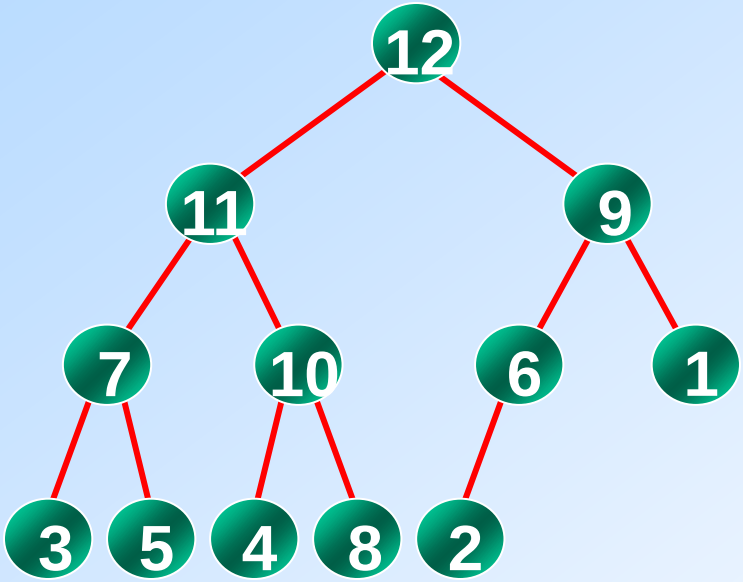
树形结构树　　二叉树　　二叉排序树



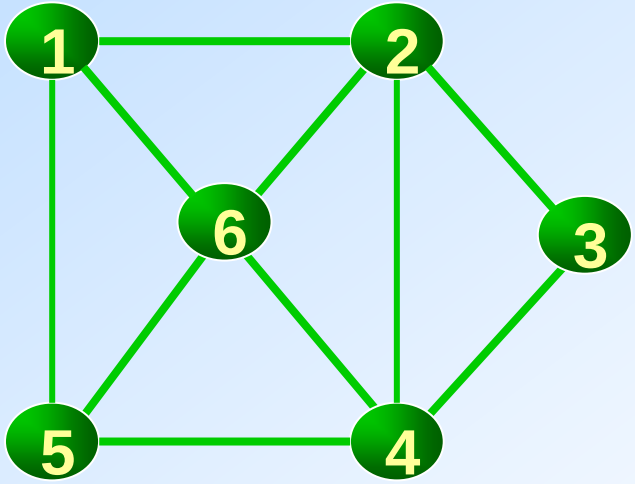




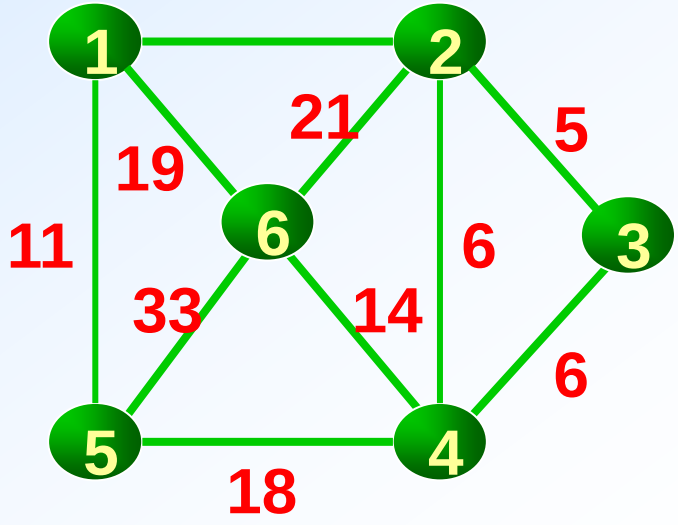
堆结构



图结构

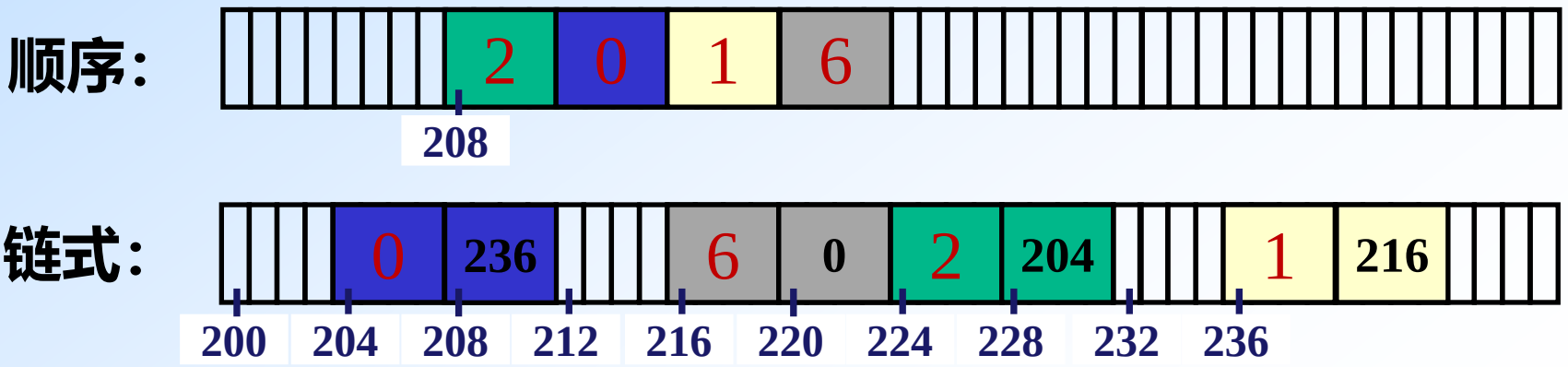


网络结构



数据的存储结构（物理结构）

- 数据结构在计算机中的表示。
- 数据的存储结构依赖于计算机语言。
  - 顺序存储表示
  - 链接存储表示
  - 索引存储表示
  - 散列存储表示



数据处理

- 将数据通过人力或机器，将收集到的数据加以系统的处理，归纳出有价值的信息。

1. 编辑（edit）：将存在某种媒体上的数据经过计算机复制到另一媒体时，对输入的数据逐一检查，其目的在于改变数据的存储形式和效率，以便后面的处理。
2. 排序（sort）：将数据根据某一键值，以某种顺序排序后输出，其目的在于方便其他方面的数据处理。

3. 归并（merge）：将两种以上相同性质的文件数据归并在一起。
4. 分配（distribute）：将一个文件的数据按照某一基准分配在两个以上的存储体，其目的在于方便各个分配的文件能独自处理。

5. 建档（generate）：根据某些条件规格，配合某些已存在的文件，再产生一个新的且有利用价值的文件。
6. 更新（update）：根据数据的变动来更新主档案，以保持主档案的正确与完整性。

7. 计算（compute）：将读取的文件数据，依据规定方法计算处理。
8. 链表（list）：是一种数据的集合，也就是一系列的数据存储于内存，以某种关系来连接这些相关联的数据。

9. 查找（search）：输入一个键值到数据表中进行对照，找出具有相同键值的数据。
10. 查询（inquiry）：根据数据项的键值或条件，到主档案中找出符合该条件或键值相同的数据，依照用户指定的方法输出。

11.其它处理：分类（classifying）、摘要（summarizing）、变换(transmission)。

数据类型

- 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**
  - ✓ 在程序设计语言中，一个变量的数据类型不仅规定了变量的取值范围，而且定义了该变量可用的操作

    例：C语言中的基本数据类型

char	int	float	double	void
字符型	整型	浮点型	双精度型	无值

  - ✓ 基本数据类型可以看作是程序设计语言中已实现的数据结构

抽象数据类型

- 是指一个数学模型以及定义在此数学模型上的一组操作
- 数据结构+定义在此数据结构上的一组操作 = **抽象数据类型**
- 例如：矩阵 + （求转置、加、乘、求逆、求特征值）

  构成一个矩阵的抽象数据类型

抽象数据类型的描述

- 抽象数据类型可用（D，S，P）三元组表示

  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。

  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、数据关系用伪码描述；基本操作定义格式为

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〉

- 基本操作有两种参数：赋值参数只为操作提供输入值；引用参数以&打头， 除可提供输入值外，还将返回操作结果。
- “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。若初始条件为空，则省略之。

抽象数据类型的表示和实现

- 抽象数据类型可以通过固有数据类型(高级编程语言中已实现的数据类型)来实现
  - 抽象数据类型　类 class
  - 数据对象　数据成员
  - 基本操作　成员函数(方法)
- 在C++中，类的成分(数据成员和成员函数)可以有三种访问级别
  - Private 私有成分（只允许类的成员函数进行访问）
  - Protected 保护成分（只允许类的成员函数及其子孙类进行访问）
  - Public 公有成分（允许类的成员函数、类的实例及其子孙类、子孙类的实例进行访问）

自然数的抽象数据类型定义

ADT *NaturalNumber* **is**

Objects: 一个整数的有序子集合,它开始于0,
结束于机器能表示的最大整数(*MaxInt*)。

Function: 对于所有的 $x, y \in NaturalNumber$;
False, *True* ∈ *Boolean*, +、 -、 <、 ==、 =等都是
可用的服务。

Zero( ) : *NaturalNumber*　返回自然数0

IsZero(x) :	if (*x*==0) 返回*True*
Boolean	else 返回*False*
Add (x, y) :	if (*x+y<=MaxInt*)返回 *x+y*
NaturalNumber	else 返回*MaxInt*
Subtract (x, y) :	if (*x* < *y*) 返回 0
NaturalNumber	else 返回 *x - y*
Equal (x, y) :	if (*x==y*) 返回*True*
Boolean	else 返回 *False*
Successor (x) :	if (*x==MaxInt*) 返回 *x*
NaturalNumber	else 返回 *x*+1

end *NaturalNumber*

程序的产生

- 五个阶段：

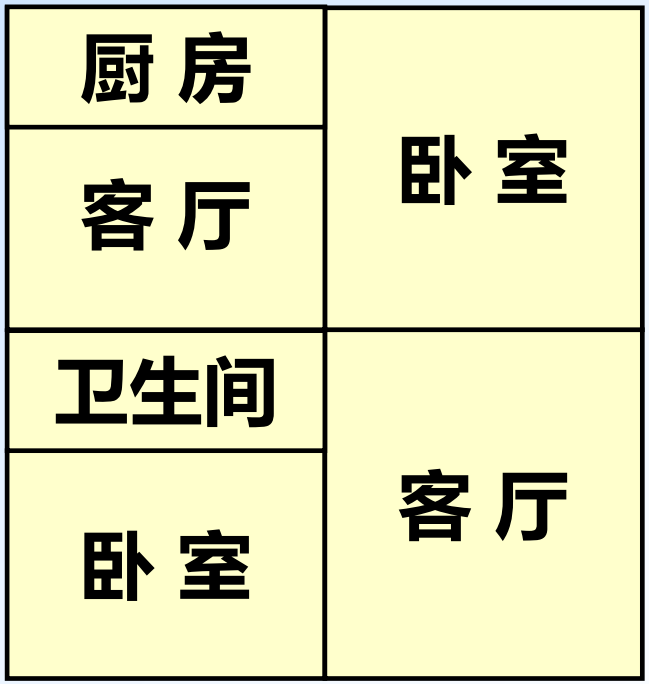
1. 需求（输入、输出）
2. 设计（编写算法）
3. 分析（选择最佳算法）
4. 细化与编码（编写程序）
5. 验证（程序验证、测试、调试）

数据结构与算法设计

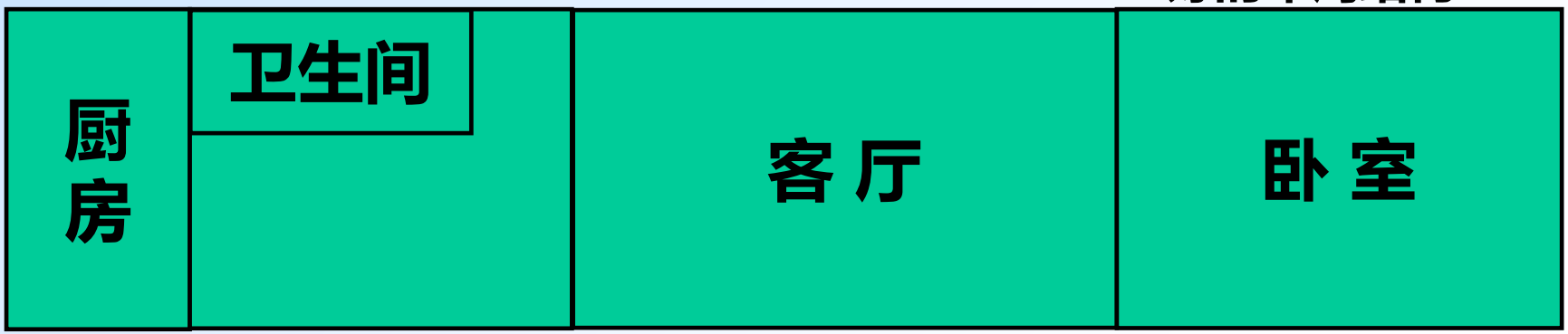
数据结构与算法的关系

- ✓ 算法基于数据结构
- ✓ 二者相辅相成

- ✓ 数据结构如同住房的布局
- ✓ 算法如同住房的装修
- ✓ 没有装修无法体现布局的好坏
- ✓ 再好的装修没有好的布局都是白搭



好的布局结构



不好的布局结构

算法分析

- **算法定义：为了解决某类问题而规定的一个有限长的操作序列 。**
- **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、无歧义**
  - **可行性 每一条运算应足够基本可行**
  - **输入 有0个或多个输入**
  - **输出 有一个或多个输出**

算法设计

- 例子： 选择排序
- 问题：递增排序
- 解决方案：逐个选择最小数据
- 算法框架：

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

- 细化：Select Sort

```
void selectSort ( int a[ ], int n ) {
 //对n个整数a[0],a[1],…,a[n-1]按递增顺序排序
      for ( int i = 0; i < n−1; i++ ) {
           int k = i;
           //从a[i]查到a[n-1], 找最小整数, 在a[k]
           for ( int j = i+1; j < n; j++ )
              if ( a[j] < a[k] ) k = j;
    int temp = a[i];  a[i] = a[k];   a[k] = temp;
     }
}
```

性能分析与度量

算法的性能标准

- 正确性：包括　不含语法错误
  对几组数据运行正确
  对典型、苛刻的数据运行正确；
  对所有数据运行正确
- 可读性
- 效率：高效、低存储需要。（算法执行时间短，同时所占用的存储空间小。
- 健壮性（鲁棒性）：当输入非法数据时，算法也能作出适当反应，而不会出现莫名其妙的输出结果。

算法的后期测试

在算法中的某些部位插装时间函数time ( ),
测定算法完成某一功能所花费时间

```
double start, stop;
time (&start);
int k = seqsearch (a, n, x);
time (&stop);
double runTime = stop - start;
printf (" %d%d\n " , n, runTime );
```

```
int seqsearch ( int a[ ], int n, int x ) {
//在a[0],…,a[n-1]中搜索x
    int i = 0;
    while ( i < n && a[i] != x )
        i++;
    if ( i == n ) return −1;
    return i;
}
```

算法的事前估计

- 空间复杂度度量

- **存储空间的固定部分**
  程序指令代码的空间，常数、简单变量、定长成分(如数组元素、结构成分、对象的数据成员等)变量所占空间
- **可变部分**
  尺寸与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所用空间、通过new和delete命令动态使用空间

时间复杂度度量

- 运行时间 = 算法中每条语句执行时间之和。
- 每条语句执行时间 = 该语句的执行次数（频度）* 语句执行一次所需时间。
- 语句执行一次所需时间取决于机器的指令性能和速度和编译所产生的代码质量，很难确定。
- 设每条语句执行一次所需时间为单位时间，则一个算法的运行时间就是该算法中所有语句的频度之和。

时间复杂度

一、时间复杂度

即算法中语句重复执行次数的数量级就是时间复杂度。

二、表示方法：

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

f(n)表示基本操作重复执行的次数，是n的某个函数，随问题规模n的增大，算法执行时间的增长率和f(n)的增长率属于同一数量级；

O表示f(n)和T(n)只相差一个常数倍。

T(n)称做渐进时间复杂度，简称时间复杂度。

几种时间复杂度

1. $O(1)$:常数时间
2. $O(\log_2 n)$：对数时间
3. $O(n)$：线性时间
4. $O(n\log_2 n)$：线性对数时间
5. $O(n^2)$:平方时间
6. $O(n^3)$:立方时间
7. $O(2^n)$:指数时间

上述的时间复杂度的优劣次序如下（n>=16):

$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)$

例子：排序

输入

数字序列

$a_1, a_2, a_3, \ldots\ldots, a_n$

1, 8, 3, 6, 7, 2

排序问题

输出

数字序列的元素置换序列

$b_1, b_2, b_3, \ldots\ldots, b_n$

1, 2, 3, 6, 7, 8

正确性

- ✓ 对于任意输入，算法输出需满足 $b_1 \le b_2 \le b_3 \le \ldots \le b_n$
- ✓ $b_1, b_2, b_3, \ldots, b_n$为$a_1, a_2, a_3, \ldots, a_n$的置换序列

运行时间

依赖于：

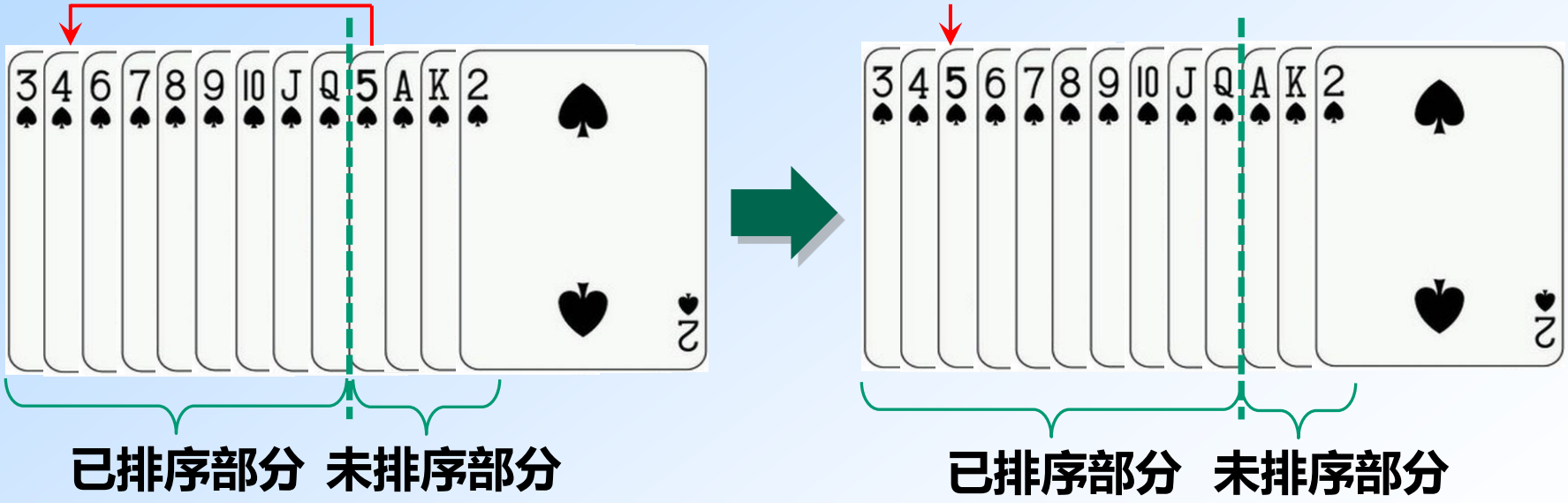
- ✓算法
- ✓元素个数
- ✓输入序列的有序性

最好/最坏/平均情况

- 最坏情况：最常使用，为复杂度上界，在许多应用中知道最坏情况非常重要
- 最坏情况经常发生
- 平均复杂度通常与最坏复杂度处于相同水平
- 计算平均复杂度一般非常困难

例子：插入排序

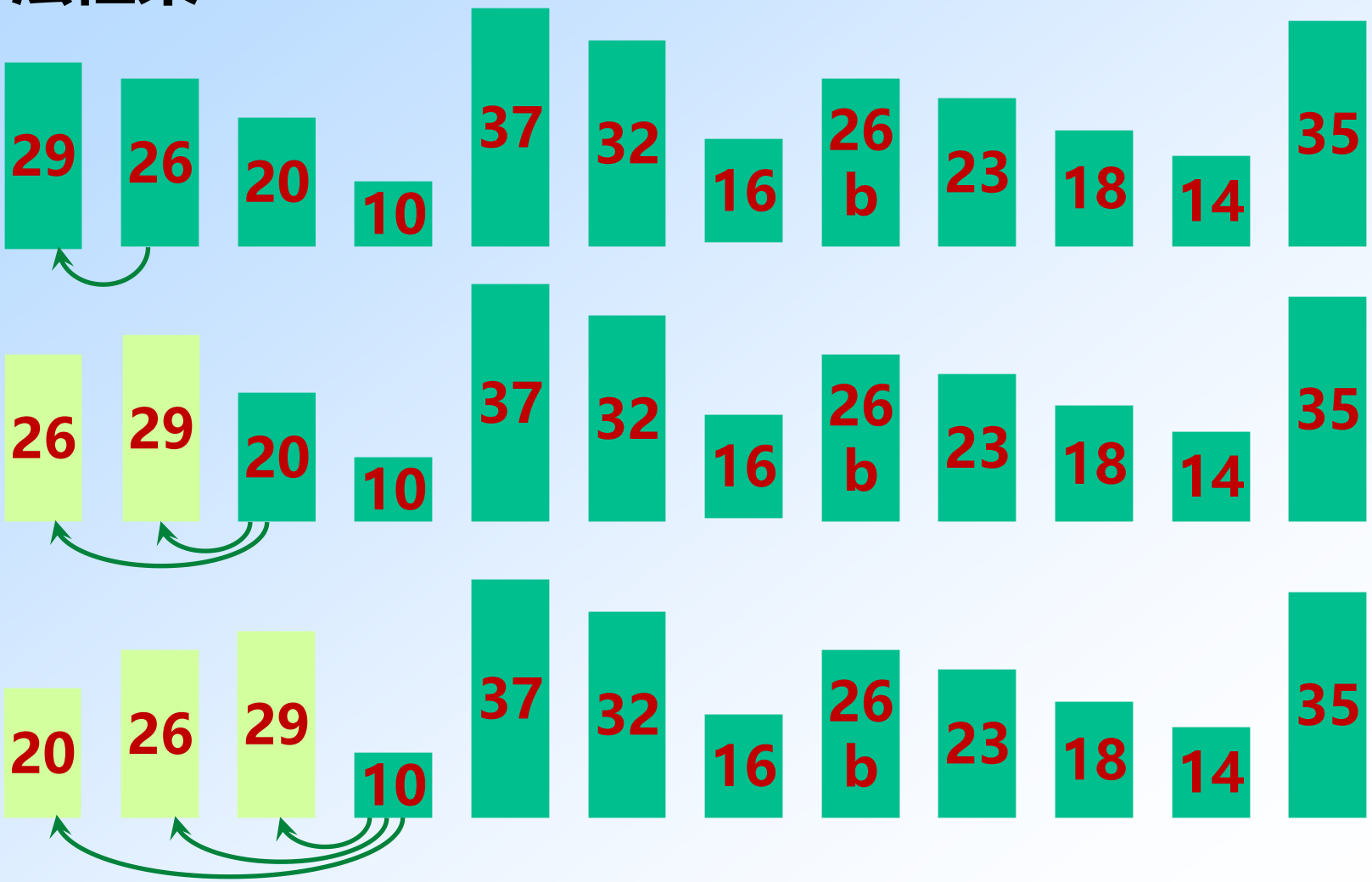
- 基本方法：每步将一个待排序的元素，按其排序码大小，插入到前面已经排好序的一组元素的适当位置上，直到元素全部插入为止



- 当插入第 i (i ≥ 1) 个元素时，前面的V[0], V[1], …, V[i-1]已经排好。用V[i]的排序码与V[i-1], V[i-2], …的排序码顺序进行比较，找到合适的插入位置将V[i]插入，原来位置上的元素向后顺移

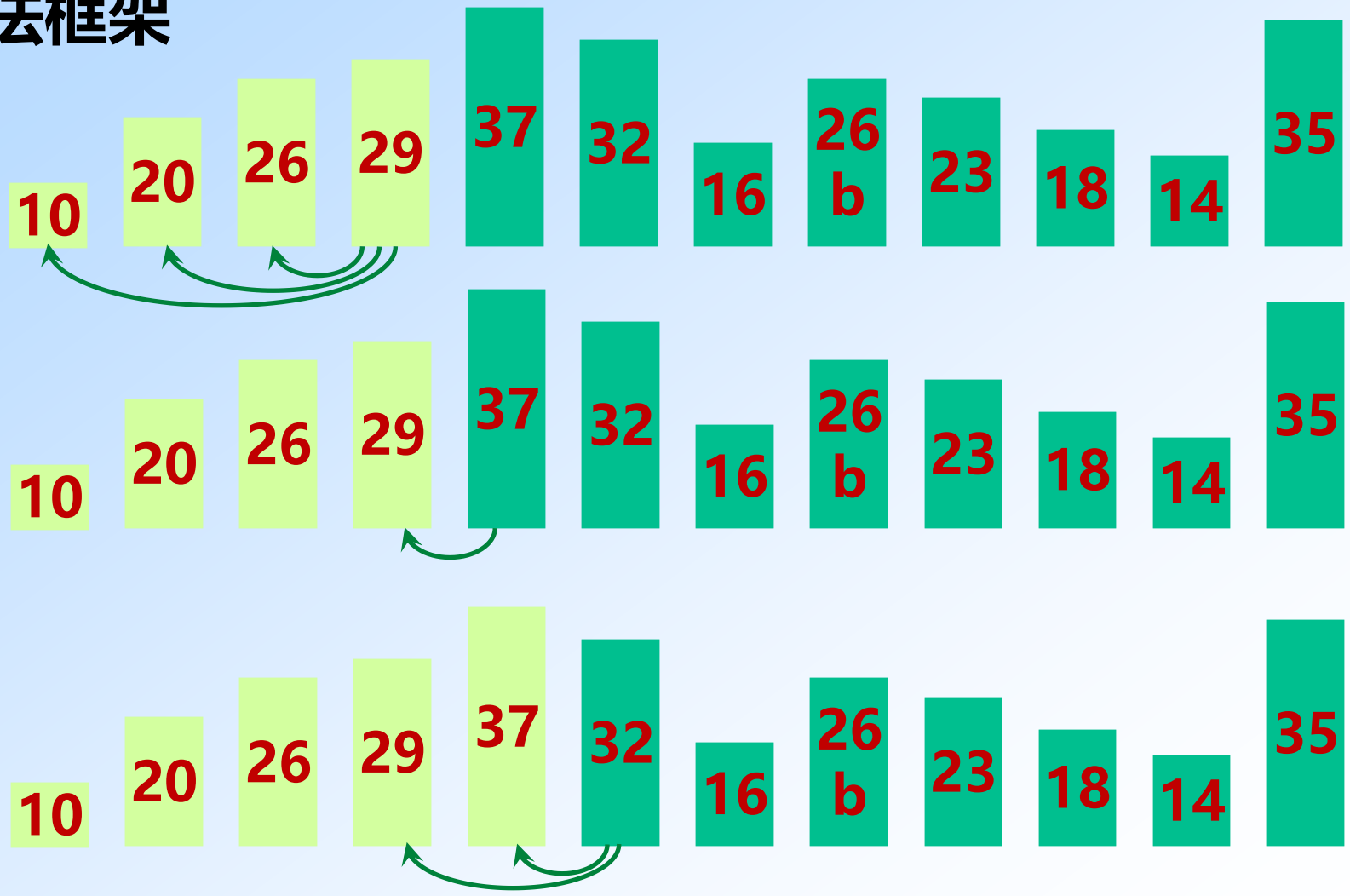
例子：插入排序

- 算法框架


29
26
20
10
37
32
16
26
b
23
18
14
35
26
29
20
10
37
32
16
26
b
23
18
14
35
20
26
29
10
37
32
16
26
b
23
18
14
35

例子：插入排序

- 算法框架


10
20
26
29
37
32
16
26
b
23
18
14
35
10
20
26
29
37
32
16
26
b
23
18
14
35
10
20
26
29
37
32
16
26
b
23
18
14
35

例子：插入排序

代码实现

```
void insertSort(int data[], int n) {
    int buffer = 0;
    for (int i = 1; i < n; i++) {  // 从第二个元素起进行插入操作
        buffer = data[i];          // 缓存该元素
        int j = i - 1;             // 初始化比较元素位置为前一个
        while(j >= 0 && buffer < data[j]){ //该次比较继续进行条件
            data[j + 1] = data[j]; //后移元素
            j -= 1;                //继续下一个位置比较
        }
        data[j + 1] = buffer;      //比较停止，将插入值插入对应位置
    }
}
void main(){
    int array[] = { 14, 10, 18, 16, 20, 26, 23, 29, 26, 35, 32, 37 };
    insertSort(array,12);
    for (int i = 0; i < 12; ++i) cout << array[i] << endl;
}
```

例子：插入排序复杂度分析

```
void insertSort(int data[], int n) {
    int buffer = 0;
    for (int i = 1; i < n; i++) {   // 比较次数n-1, 赋值次数n-1
        buffer = data[i];           // 赋值次数1
        int j = i - 1;              // 赋值次数1
        while(j >= 0 && buffer < data[j]){ // 最多(i+1)x2次比较
                                           // 最少2次比较
            data[j + 1] = data[j]; // 赋值次数1
            j -= 1;                // 赋值次数1
        }
        data[j + 1] = buffer;      // 赋值次数1
    }
}
```

总体最多操作次数 = (n-1)(2+1+1+1) + $\sum_{i=1}^{n-1}(i+1)(2+1+1)$ **多项式（平方）复杂度**

总体最少操作次数 = (n-1)(2+1+1+2) = 6n-6　线性复杂度

平均情况下操作次数为平方复杂度

习题

1. 顺序存储表示中数据元素之间的逻辑关系是由（ ）表示的？
链接存储表示中数据元素之间的逻辑关系是由（ ）表示的？

A. 指针　B. 逻辑顺序　C. 存储位置　D. 问题上下文

2. 在下面的程序段中，对i的赋值语句的频度为（ ）

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

A. O(3n)　B. O(n)　C. $O(n^3)$　D. $O(\log_3 n)$

习题

1. 顺序存储表示中数据元素之间的逻辑关系是由（C ）表示的？链接存储表示中数据元素之间的逻辑关系是由（A ）表示的？

A. 指针　B. 逻辑顺序　C. 存储位置　D. 问题上下文

2. 在下面的程序段中，对i的赋值语句的频度为（D ）

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

A. O(3n)　B. O(n)　C. $O(n^3)$　D. $O(\log_3 n)$

谢谢！

模板 (template) (C++)

- 定义
- 适合多种数据类型的类定义或算法，在特定环境下通过简单地代换，变成针对具体某种数据类型的类定义或算法

例：求最大值max(a,b)

- 宏定义：
- # define max(a,b) ((a)>(b) ? (a):(b))
- 缺点不能检查数据类型，损害类型安全性

例：求最大值max(a,b)

- 函数重载：

```
int max(int a, int b){return a>b?a:b;}
float max(float a,float b){return a>b?a:b;}
void main(){
    cout <<“Max(3,5) is”<<max(3,5)<<endl;
    cout<<“Max(‘3’,’5’)is”<<max(‘3’,’5’)<<endl;
}
```

对于输入max(3,5)和max(‘3’,’5’)的结果？

例：求最大值max(a,b)

- 函数模板：

```
#include <iostream.h>
template<class T>T max(T a,T b)
{ return a>b?a:b;}
void main(){
    cout <<“Max(3,5) is”<<max(3,5)<<endl;
    cout<<“Max(‘3’,’5’)is”<<max(‘3’,’5’)<<endl;
}
```