数据结构

Data Structure

- 课程安排及要求
- 课程概览
- 基本概念和术语
- 抽象数据类型（ADT）
- 算法和算法分析

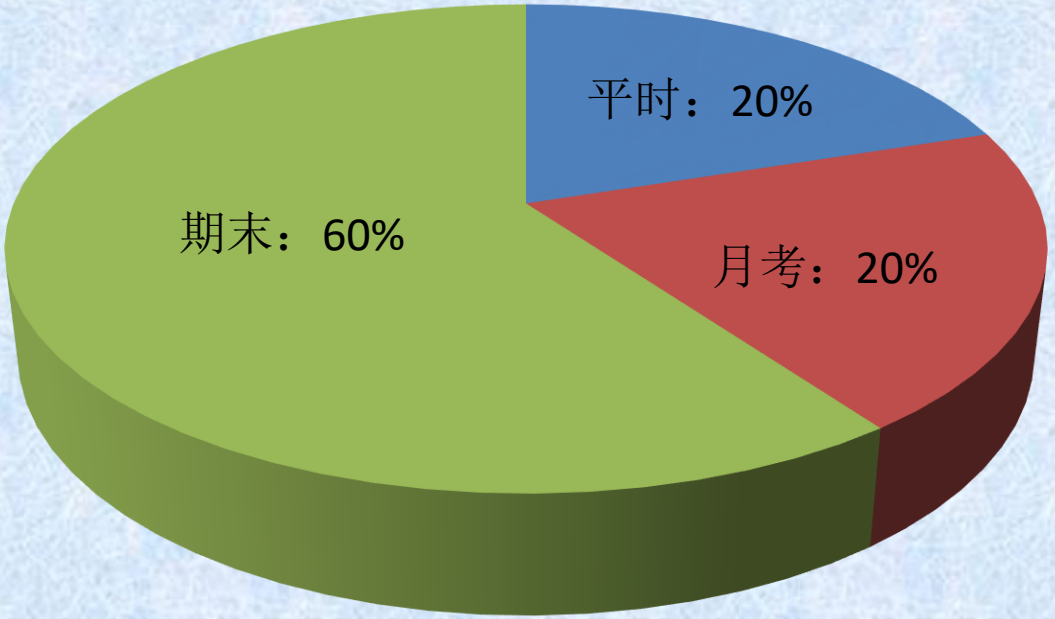
- 课程安排及要求
- 课程概览
- 基本概念和术语
- 抽象数据类型（ADT）
- 算法和算法分析

课程安排及要求

性质　专业必修

学分　3学分

学时　72学时



课程安排及要求（续）

- 学习本课程的意义
- 学习本课程的条件
- 学习本课程的要求

- 课程安排及要求
- 课程概览
- 基本概念和术语
- 抽象数据类型（ADT）
- 算法和算法分析

计算机应用

数值计算：加工处理的对象——纯粹的数值。

非数值计算

工业检测
过程控制
管理系统
数据处理
……

占据了当今计算机应用的绝大多数。

加工处理的对象

字符
表格
图象
声音
……

具有一定的结构

研究对象的特性及其相互之间的关系

逻辑结构

有效地组织计算机存贮

存储结构

有效地实现对象之间的“运算”关系

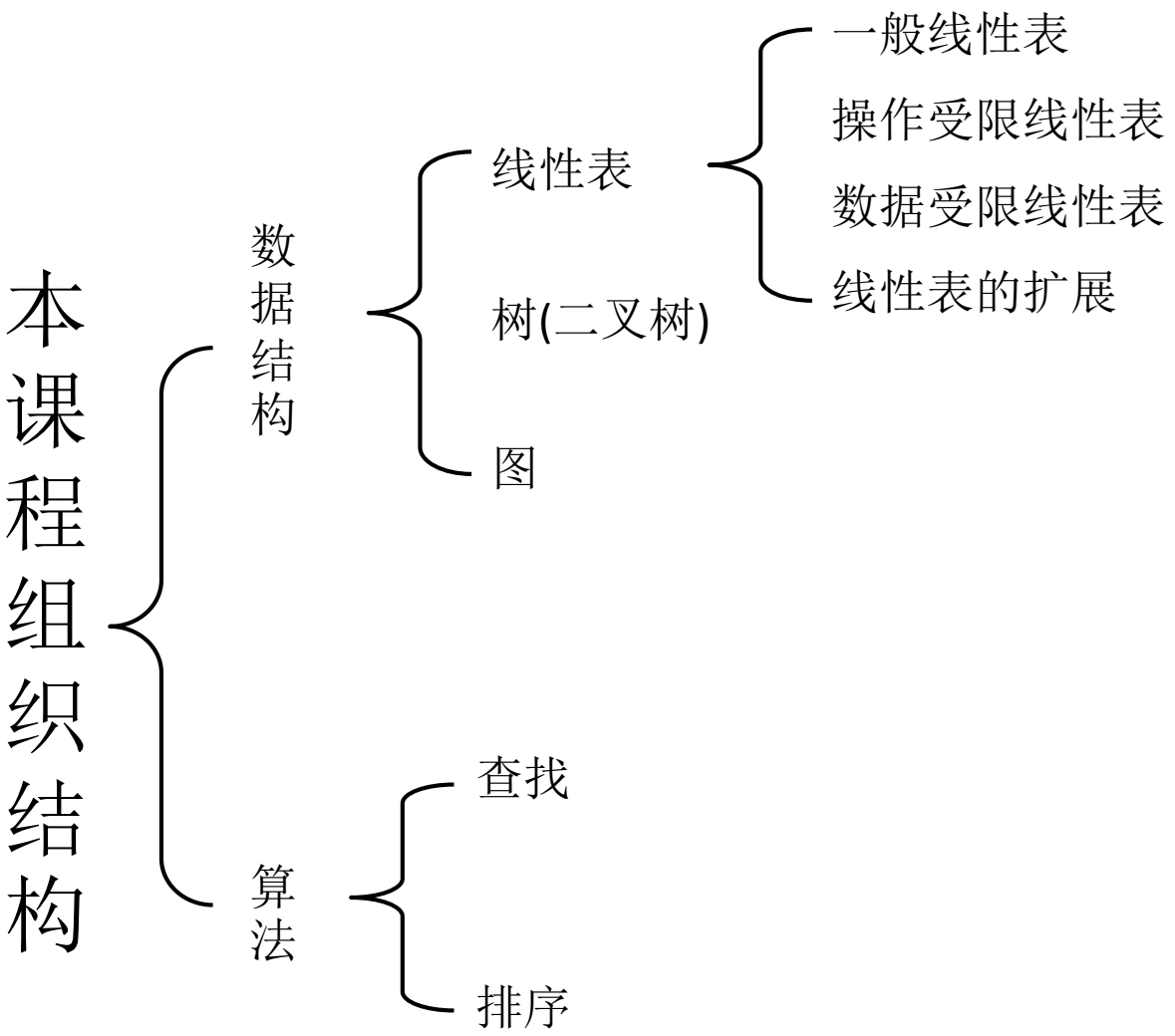
算法

《数据结构》的研究内容

计算机解题步骤

- 抽象数学模型
  - 分析问题
  - 提取操作对象
  - 找出操作对象之间的关系
  - 用数学语言描述 ⇒ 数据结构
- 设计算法
- 编程、调试、运行

《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**的一门学科。


本课程组织结构
数据结构
线性表
一般线性表
操作受限线性表
数据受限线性表
线性表的扩展
树(二叉树)
图
算法
查找
排序

- 课程安排及要求
- 课程概览
- 基本概念和术语
- 抽象数据类型（ADT）
- 算法和算法分析

基本概念和术语

- **数据（Data）**

  是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。

- **数据对象 （data object）**

  性质相同的数据元素的集合，是数据的一个子集。

- **数据元素 （data element）**

  是数据的基本单位，在计算机程序中通常作为一个整体而考虑和处理。

基本概念和术语

- **数据项 （data item）**

  一个数据元素可由若干个数据项组成，数据项是数据不可分割的最小单位。

- **结构（Structure）**

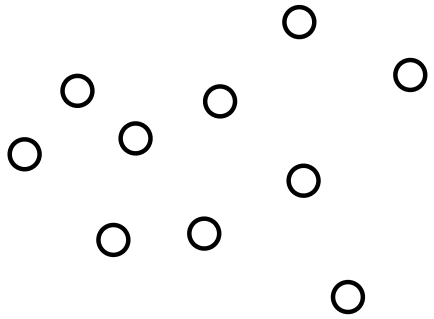
  是组成整体的各部分的关系和关联。

- **数据结构（ Data Structure）**

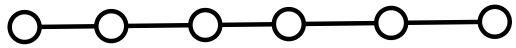
  数据结构是相互之间存在一种或多种特定关系的数据元素的集合，也可称其为逻辑结构。

四类基本数据结构

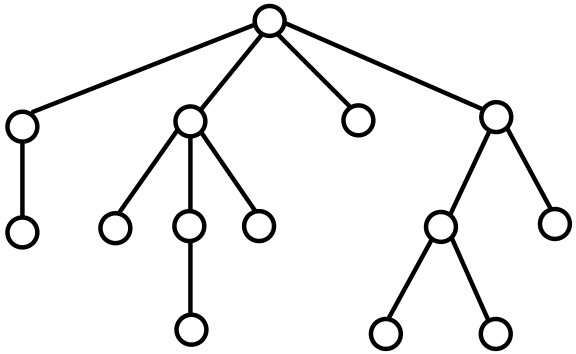
集合



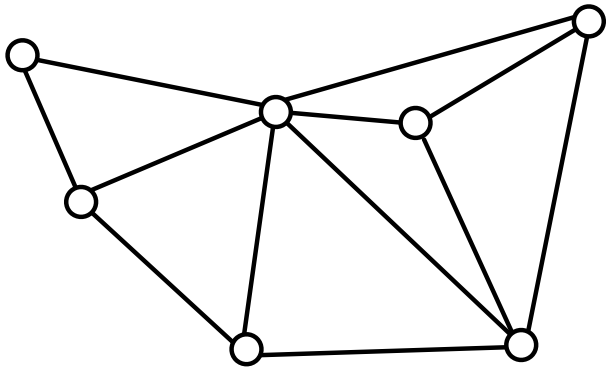
线性



树



图



关系、关联的表示--用序偶表示

- 数据结构的形式定义：
  Data_Structure = (D, S)

- 例如：复数数据结构Complex = (C, R)
  C = {c1,c2|c1,c2属于任意实数}
  R = {<c1,c2>|c1表示实部,c2表示虚部}

- 三种基本的关系
  1:1　　1:n　　n:m

基本概念和术语(续)

- 物理结构(存储结构)

  数据结构在计算机中的表示(映像)称为数据的物理结构。

  – 顺序映像

  顺序映像的特点是借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系。

  – 非顺序映像

  非顺序映像的特点是借助指示元素存储地址的指针表示数据元素之间的逻辑关系。

基本概念和术语(续)

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。 值的集合+值集合上的一组操作

数据类型的作用
- 约束变量的内存空间，
- 约束变量或常量的取值范围，
- 约束变量或常量的操作。

例如，*C* 语言中的 int 型变量，是指一定范围中的值构成的集合及一组操作（加、减、乘、除、乘方 等）的总称。

基本概念和术语(续)

- **抽象数据类型（abstract data type ADT）**

  是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。

- **原子类型、固定聚合类型、可变聚合类型、多形数据类型**（p8-9）

- 课程安排及要求
- 课程概览
- 基本概念和术语
- 抽象数据类型（ADT）
- 算法和算法分析

抽象数据类型 Abstract Data Type, ADT

- 含义

  一种数据类型，其数据对象和对象操作的规格说明独立于对象的存储表示和对象上操作的实现。

书号	书名	作者
1	数据结构	严蔚敏
2	信息论	周萌清
……		

学号	姓名	分数
1	张三	80
2	李四	90
……		

货号	品名	型号	价格
1	洗衣机	XM-1	1500
2	电视机	HDV-2	5000
……			

序号	系数	指数
1	2	500
2	-5	1500
……		

ADT的定义

- 定义：

和数据结构的形式定义相对应，抽象数据类型可以用三元组来刻画:(D,S,P)。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。

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码描述，基本操作的定义格式为：

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>

一个例子---二元组的定义

ADT Compare{

数据对象：D={e1,e2| e1,e2为可比较的同类型的元素}

数据关系：R={< e1,e2 >}

基本操作：

InitCom(&C,ee1,ee2)

操作结果：构造一个二元组c，元素e1,e2分别被赋成ee1,ee2。

FirElemBig(C)

初始条件：二元组已经存在。

操作结果：如果首元素大，则返回1，否则返回0。

…

} ADT Compare

类C语言简介

- 类C就是一种新的语言，跟C语言很类似，但是现在没有一个编译器支持它。
- &操作符
- 预定义常量和类型等
- P10.

抽象数据类型的表示

- 抽象数据类型的表示就是要将该类型映射到计算机中，也就是确定抽象数据类型的存储结构以及给出基于该结构之上的基本操作的函数原型。
- 例子

```
typedef  int   ElemType; //整形元组
typedef  ElemType  * Compare; //动态顺序存储结构
//初始化二元组
InitCom(Compare  &C, ElemType  ee1, ElemType  ee2)

//判断二元组的首元素是否比次元素大
FirElemBig(Compare  C)
……
```

抽象数据类型的实现

- 抽象数据类型的实现就是基于特定存储结构之上的基本操作的实现。

- 例子

```
//初始化二元组
Status InitCom(Compare  &C, ElemType  ee1, ElemType  ee2)
 {
      C = (ElemType *)malloc(2*sizeof(ElemType));
      if(!C) exit (OVERFLOW);
       C[0] = ee1;  C[1] = ee2;
       return OK;
 }

 //判断二元组的首元素是否大
Status FirElemBig(Compare  C)
{
       if(C[0] > C[1])  return  TRUE;
       else  return  FALSE;
}
```

思考

- 矩形抽象数据类型的定义？
- 矩形抽象数据类型的表示？
- 矩形抽象数据类型的实现？

抽象数据类型矩形的定义

ADT Rectangle {
    数据对象： length//非负实数，表示矩形的长；
                width//非负实数，表示矩形的宽；
    数据关系： 无
    基本操作：
        Init(&R, length， width)
            初始条件：无
            操作结果：将矩形R的长和宽初始化成length和width
        Area(R)
            初始条件：矩形R存在
            操作结果：返回矩形的面积
        Circumference(R)
            初始条件：矩形R存在
            操作结果：返回矩形的周长
} ADT Rectangle

矩形抽象数据类型的表示

```
// 定义矩形的存储结构
typedef struct
    {   float length;                    // 矩形的长
        float width；                 // 矩形的宽
    } Rectangle;
// 操作目的：对矩形R初始化
// 初始条件：
// 操作结果：将矩形R的长初始化成 l，宽初始化为w
bool Init(Rectangle &R, float l, float w);
// 操作目的：求矩形R的面积
// 初始条件：矩形R存在
// 操作结果：返回矩形的面积
float Area(Rectangle R);
// 操作目的：求矩形R的周长
// 初始条件：矩形R存在
// 操作结果：返回矩形的周长
float Circumference(Rectangle R);
```

矩形ADT的实现

```
bool Init(Rectangle &R, float l, float w)
{
    if(l>0&&w>0){
        R.length=l;
        R.width=w;
        return true;}
    else
        return false;
}
float Area(Rectangle R)
{
    return R.length*R.width;
}
float Circumference(Rectangle R)
{
  return 2*(R.length+R.width);
}
```

ADT小结

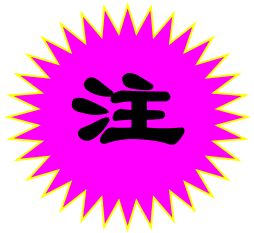
- 抽象数据类型主要指用户在设计软件系统时自己定义的数据类型。
- 在构造软件系统的各个相对独立的模块时，定义**一组数据**和施与这些数据之上的**一组操作**，并在模块**内部**给出它们的**表示和实现细节**，在模块**外部**使用的只是**抽象的数据和抽象的操作**。

- 课程安排及要求
- 课程概览
- 基本概念和术语
- 抽象数据类型（ADT）
- 算法和算法分析

算法的基本概念

算法---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。

- 算法的五个特性
  - **有穷性**
  - **确定性**
  - **可行性**
  - **0个或多个输入**
  - **1个或多个输出**
- 算法的设计要求
  - **正确性**
  - **可读性**
  - **健壮性**
  - **高效率**
  - **低存储**



算法的含义与程序十分相似，但二者是有区别的。

1、一个程序不一定满足有穷性（如一个操作系统在用户未使用前一直处于“等待” 的循环中，直到出现新的用户事件为止。这样的系统可以无休止地运行，直到系统停工。）；

2、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。算法若用计算机语言来书写，则它就可以是程序。

一个算法可以用自然语言、数学语言或约定符号来描述，也可以用流程图、计算机高级程序语言（如 C 语言）或伪代码等来描述。

算法的表现形式

```
void   bubble_sort(int   a[] , int  n)
{   //将a中整数序列按从小到大的顺序排序
     for(i = n-1, change = TURE;i>=1 && change; i--)
     {
         change = FALSE;//交换标识
          for(j = 0; j<i;j++)
               if(a[j] > a[j+1])
               {
                    a[j]<--->a[j+1];
                      change = TRUE;
               }
}
```

算法效率的度量

- 事后统计法
- 事前分析法

--只考虑问题规模（基本操作）

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))，它表示随着问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同。

--频度的概念

基本操作的执行次数。

例：{++*x*; *s*=0;}

包含“*x* 增 1”基本操作的语句的频度为 1，即时间复杂度为*O*(1)。*O*(1) 表示算法的运行时间为常量。即：**常量阶**。

例：for(*i* =1; *i* <=*n*; ++*i*)

{++*x*; *s* += *x*;}

包含“*x* 增 1”基本操作的语句的频度为：*n*，其时间复杂度为：*O*(*n*)，即：**线性阶**。

例：for(*i* =1; *i* <= *n*; ++*i*)

for(*j* =1; *j* <= *n*; ++*j*)

{++*x*; *s* += *x*;}

包含 “*x* 增 1” 基本操作的语句的频度为：n^2，其时间复杂度为：$O(n^2)$，即：平方阶。

例：for(i =2; i <= n; ++i)

for(j =2; j <= i - 1; ++j)

{++x; a[i, j]=x;}

包含 “*x* 增 1” 基本操作的语句的频度为：

$1+2+3+\ldots+n-2 = (1+n-2)\times(n-2)/2 = (n-1)(n-2)/2 = \frac{1}{2}n^2 - \frac{3}{2}n + 1$

其时间复杂度为：$O(n^2)$，即：平方阶。

f(n)的求法

- f(n)一般用频度表达式中增长最快的项表示，并将其常数去掉。

- 例如： 假设某元操作的频度：$100*2^n+8n^2$

  则：$T(n)=O(2^n)$

算法的时间复杂度常见的有:

常数阶 $O(1)$，对数阶 $O(\log n)$，线性阶 $O(n)$，

线性对数阶 $O(n\log n)$，平方阶 $O(n^2)$，立方阶 $O(n^3)$，…，

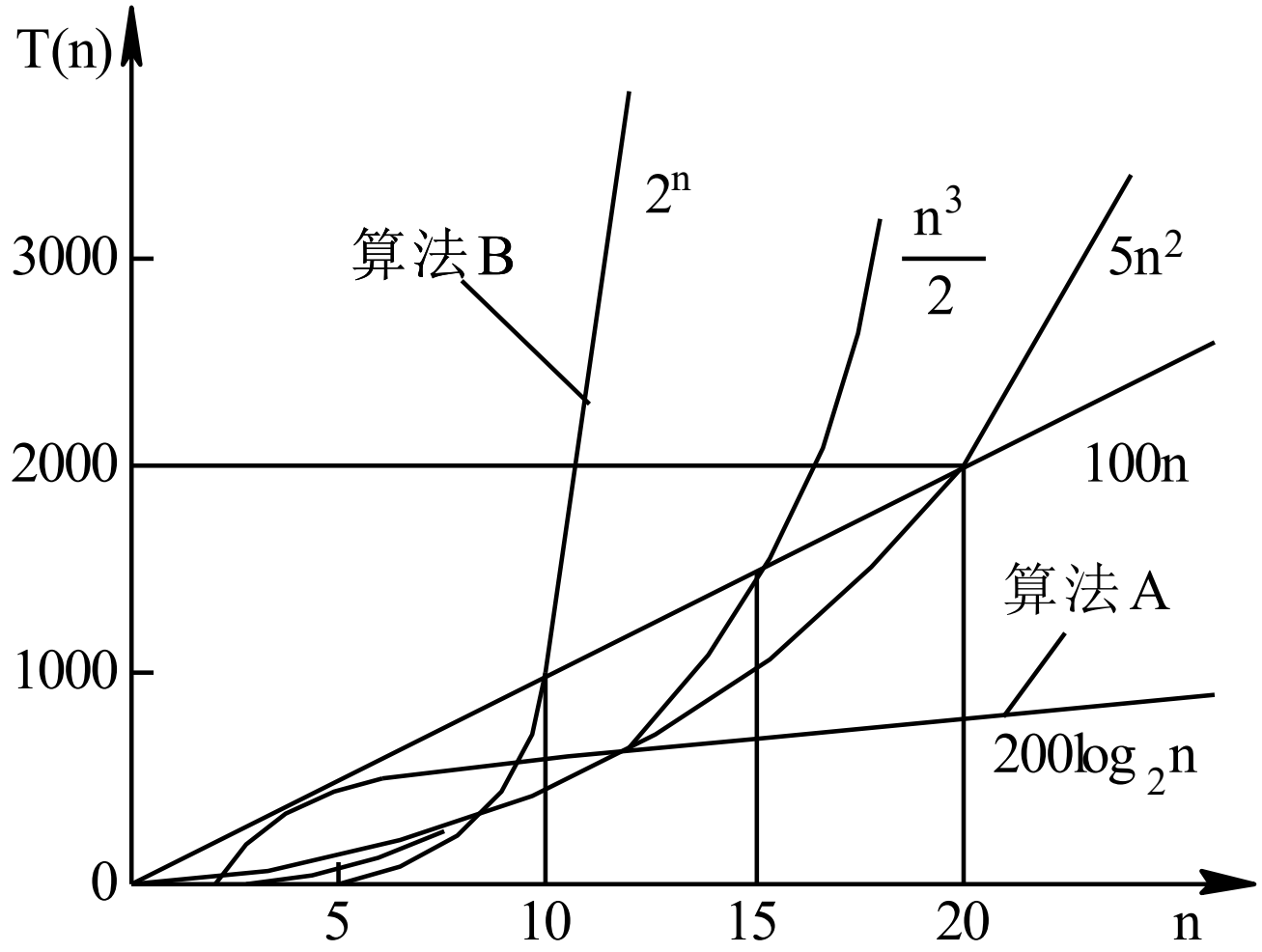
k 次方阶$O(n^k)$，指数阶 $O(2^n)$，阶乘阶 $O(n!)$。

常见的算法的时间复杂度之间的关系为:

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(2^n)<O(n!)<O(n^n)$$

当 n 很大时，指数阶算法和多项式阶算法在所需时间上非常悬殊。因此，只要有人能将现有指数阶算法中的任何一个算法化简为多项式阶算法，那就取得了一个伟大的成就。

常见函数的增长率


T(n)
3000
2000
1000
0
5
10
15
20
n
2^n
算法B
$\frac{n^3}{2}$
$5n^2$
100n
算法A
$200\log_2 n$

时间复杂度的三种具体情况

```
Void bubble-sort(int a[]，int n)
{  //将 a 中整数序列重新排列成自小至大有序的整数序列。
    for(i = n-1, change = TURE; i > 1 && change; --i)
        change = false;
        for ( j = 0;  j < i; ++j)
            if (a[ j] > a[ j +1]) {a[ j]←→a[ j +1];  change = TURE}
}// bubble-sort
```

最好情况：0次

最坏情况：1+2+3+…+n-1=n(n-1)/2

平均时间复杂度为：$O(n^2)$

在本课程中讨论的时间复杂度，均指最坏的时间复杂度。

程序代码本身所占空间对不同算法通常不会有数量级之差别，因此在比较算法时可以不加考虑；算法的输入数据量和问题规模有关，若输入数据所占空间只取决于问题本身，和算法无关，则在比较算法时也可以不加考虑；由此只需要分析除输入和程序之外的额外空间。

一个算法所需存储空间
- 算法本身的存储空间
- 输入数据的存储空间
- 算法在运行过程中临时占用的存储空间

若所需临时空间不随问题规模的大小而改变，则称此算法为**原地工作**。

若所需存储量依赖于特定的输入，则通常按最坏情况考虑。

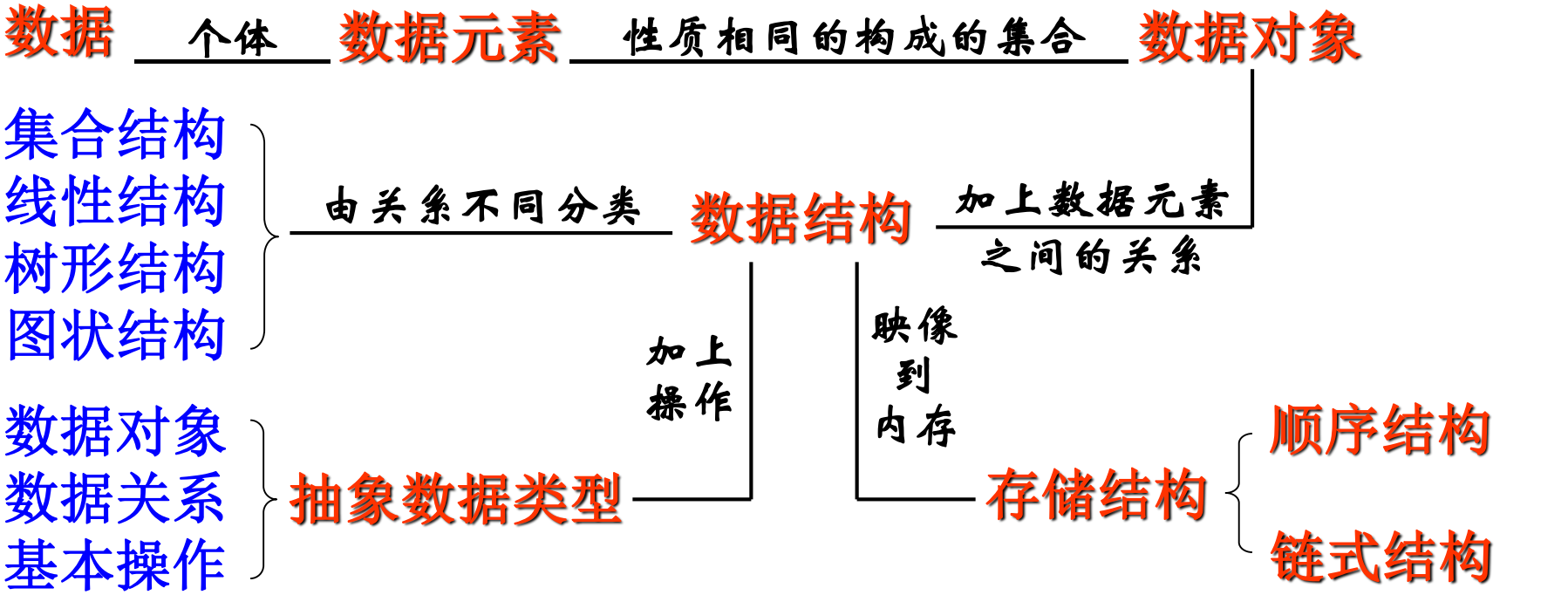
```
float abc ( float a, float b, float c )
{
        return a + b + b * c;
}


float sum ( float list [ ], int n )
{
        float tempsum = 0;
        for( int i = 0; i < n; i++ )        tempsum += list[ i ];
        return tempsum;
}
```

```
float rsum ( float list [ ], int n )
{
        if(n == 0) return 0;

         return rsum( list, n-1 ) + list[ n ];
}
```

小结：本章是为以后各章讨论的内容作基本知识的准备。



算法
- 特性：有穷性、确定性、可行性、输入、输出
- 衡量标准：正确性、可读性、健壮性、效率、存储量需求
  - 效率 → 时间复杂度
  - 存储量需求 → 空间复杂度

小结

- 什么是数据、数据结构?
- 本书主要研究哪几种数据结构？
- 什么是数据结构、逻辑结构、物理结构？
- 什么是数据对象、数据元素、数据项，及其之间的关系?
- 什么是数据类型、抽象数据类型?
- 抽象数据类型的结构、表示、实现。
- 什么是算法(基本结构)、算法有哪几个重要特性、算法设计的要求是什么、算法的时间复杂度如何度量?

Thank You !