



计算机导论与程序设计——第 4 篇

算法基础

Computer Introduction and Programming

学习目标



- 掌握算法的基本概念和基本结构
- 学会用自然语言和流程图表示算法
- 掌握常用的基本算法

目录

CONTENTS

- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

目录

CONTENTS

- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

算法的概念

算 法

为解决一个问题而采取的方法和步骤。

对同一个问题，可以有不同的解题方法和步骤。

为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法。

算法——程序的灵魂

算法+数据结构=程序

数据结构

对数据的描述。在程序中要指定用到哪些数据，以及这些数据的类型和数据的组织形式。

算法

对操作的描述。即要求计算机进行操作的步骤

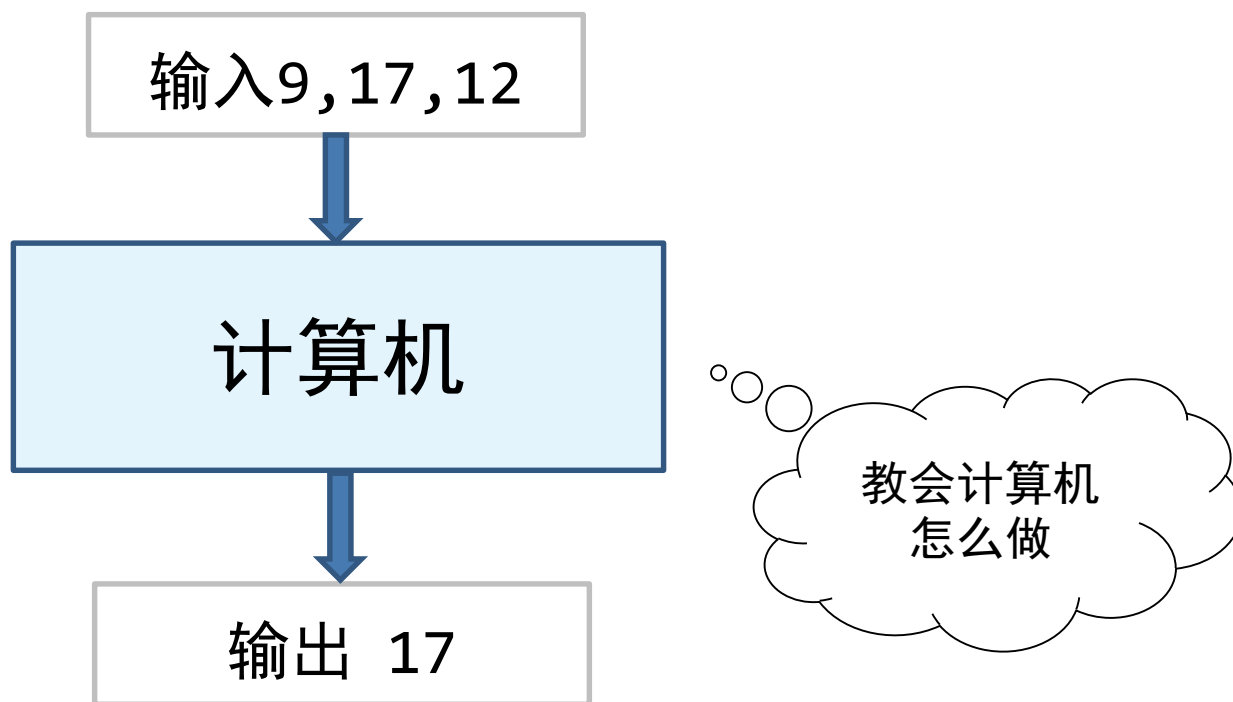


沃思

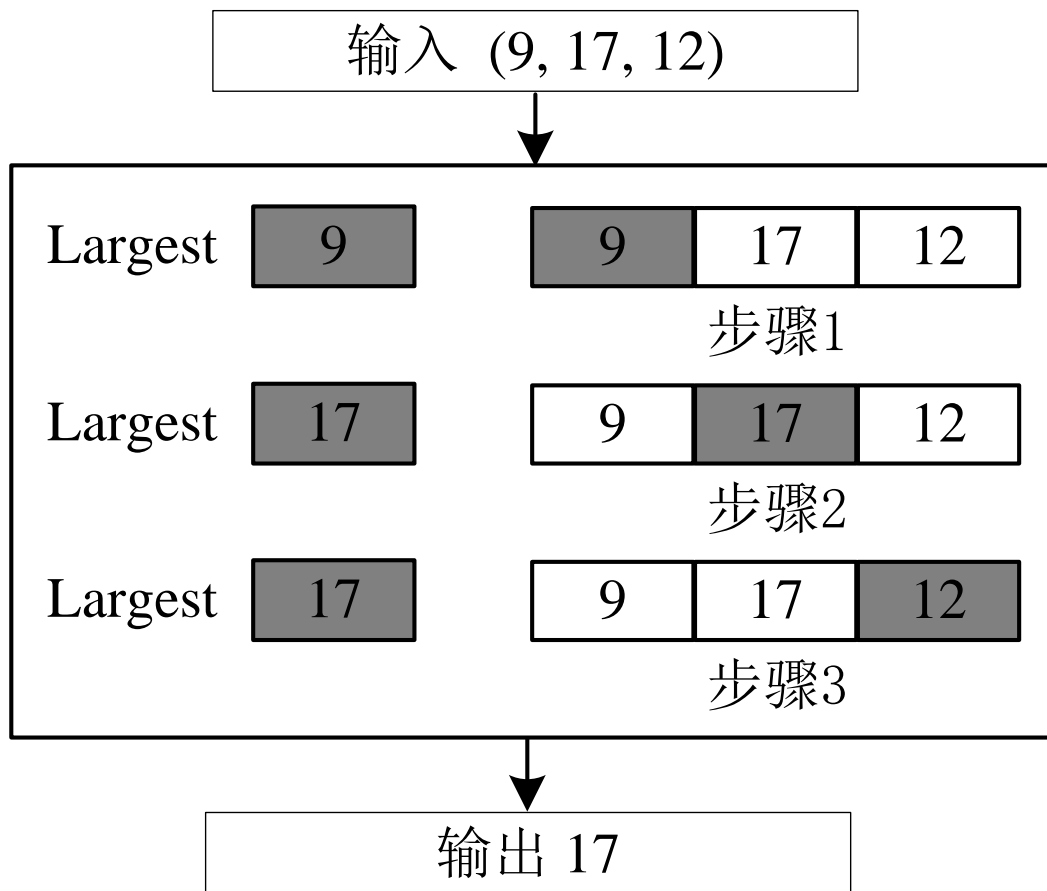
找最大的整数

问题：如何从3个整数中找到最大的一个整数？

假设给定的3个整数是9,17,12，怎么做？

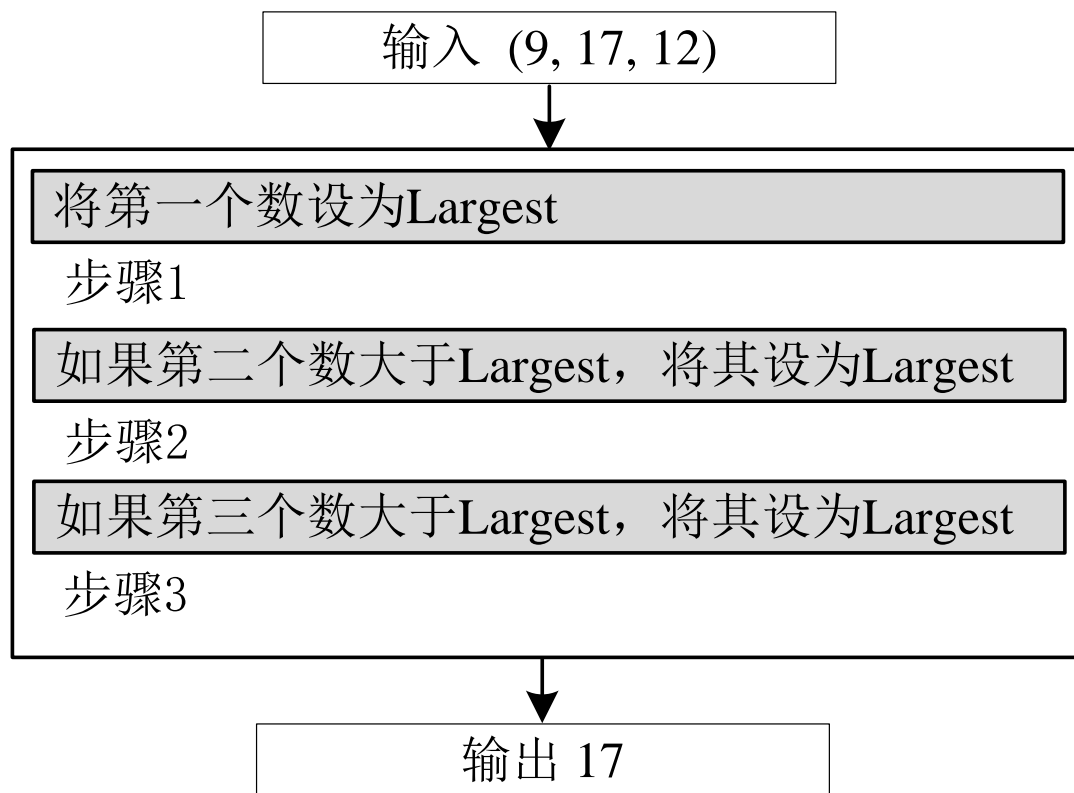


找最大的整数



找最大的整数

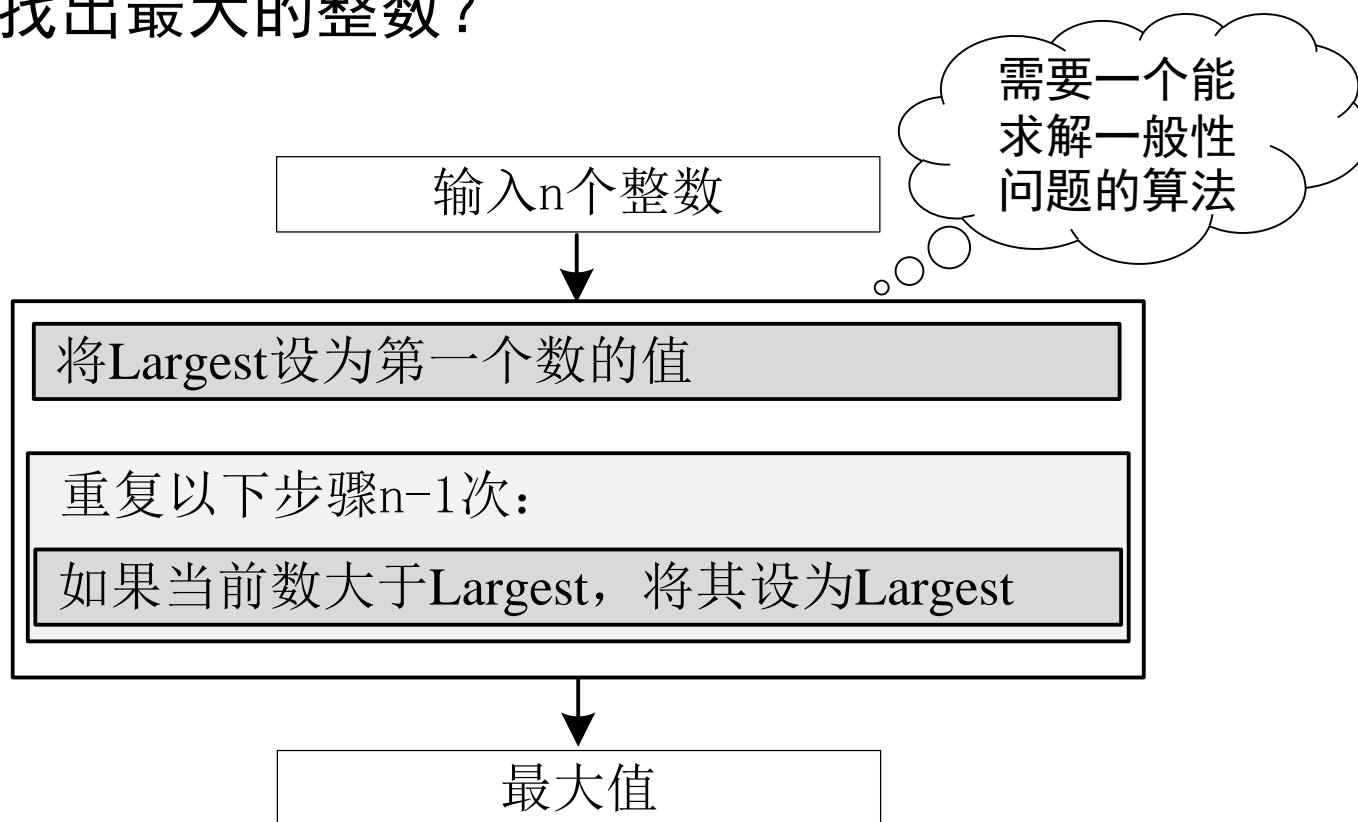
如何用自然语言描述从3个整数中求最大值算法？



找最大的整数

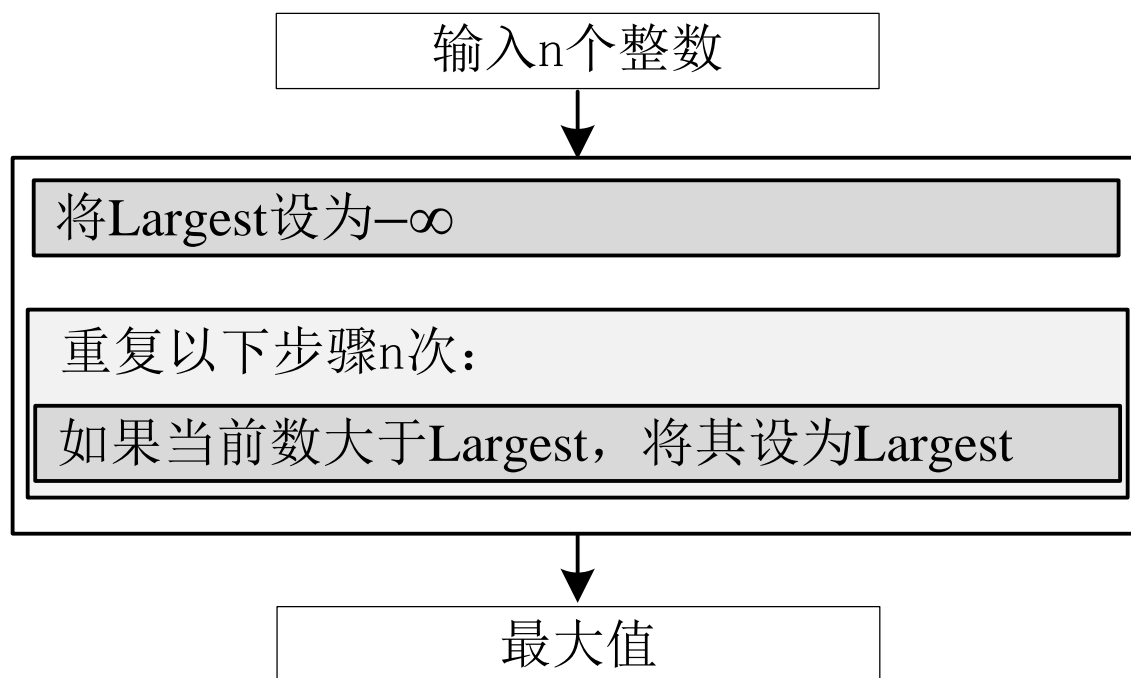
如果是5个数，10个数， \dots ， n 个数呢？

如何从中找出最大的整数？



找最大的整数

更好的一种求最大值算法的描述(带初始化步骤):



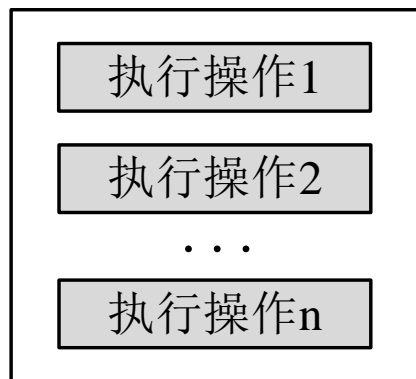
目录

CONTENTS

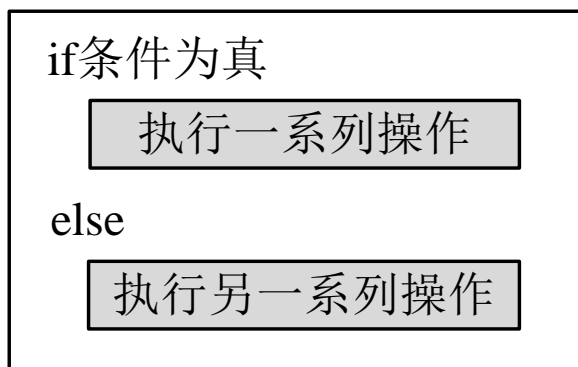
- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

算法的三种逻辑结构

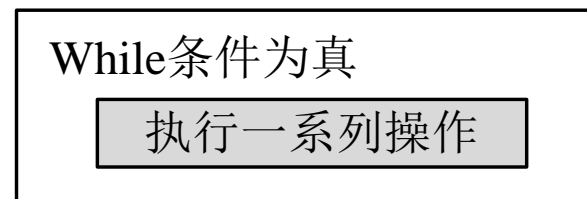
定义了三种基本结构来描述算法，任何功能的算法或者程序都可以通过这三种基本结构的组合实现。



a) 顺序结构



B) 选择结构



c) 循环结构

目录

CONTENTS

- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

4.3 算法的表示

4.3.1 流程图



4.3.2 伪代码

4.3 算法的表示

4.3.1 流程图



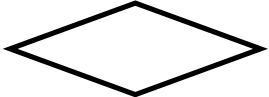





4.3.2 伪代码

流程图

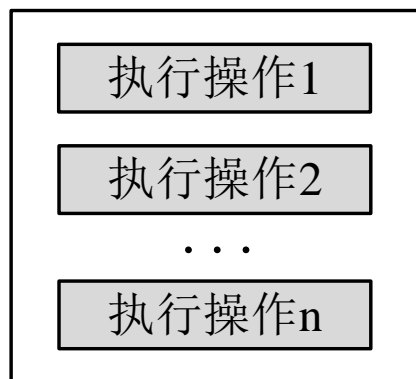
流程图

流程图是一种用图形符号表示算法的工具，直观形象，容易理解。

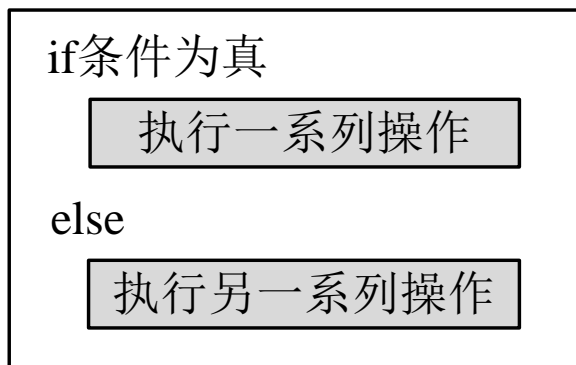
符号	含义
	起止框，表示算法的起始和终止。
	输入/输出框，用于算法中数据的输入和输出。
	判断框，表示算法中对条件的判断。
	处理框，表示对数据进行计算或操作。
	流程线，用于连接流程图中各个符号，表明算法中的步骤次序。
	连接点，连接其它流程图的入口或出口。

流程图

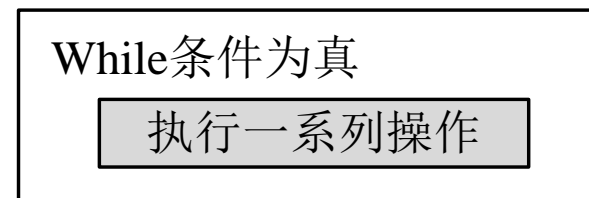
如何用流程图表示算法的三种逻辑结构？



a) 顺序结构



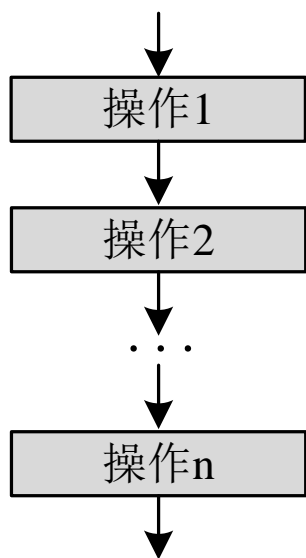
B) 选择结构



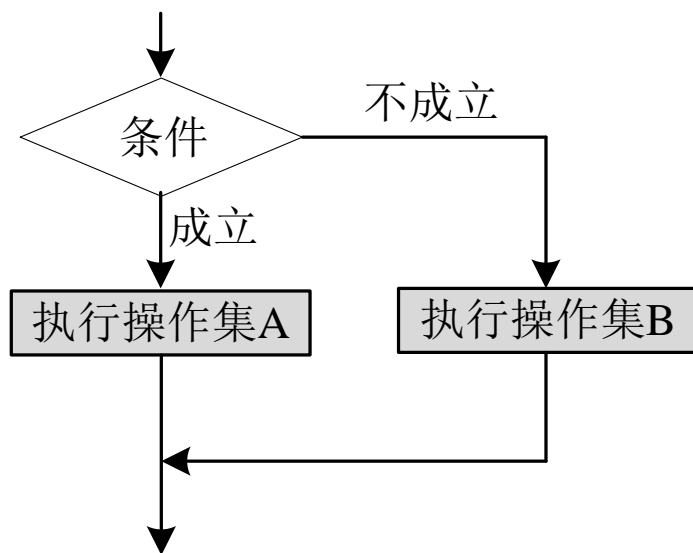
c) 循环结构

流程图

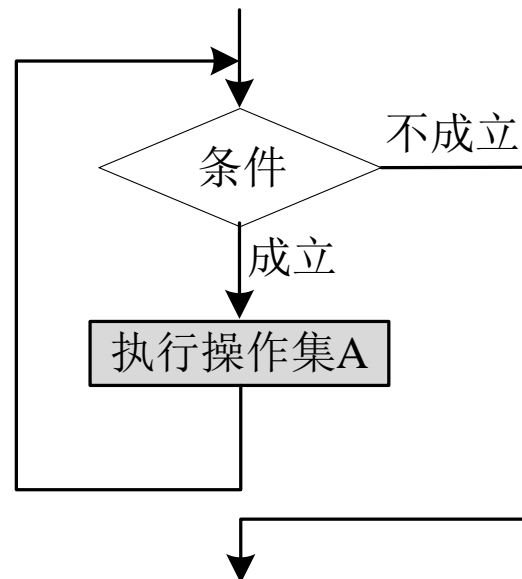
如何用流程图表示算法的三种逻辑结构？



a) 顺序结构



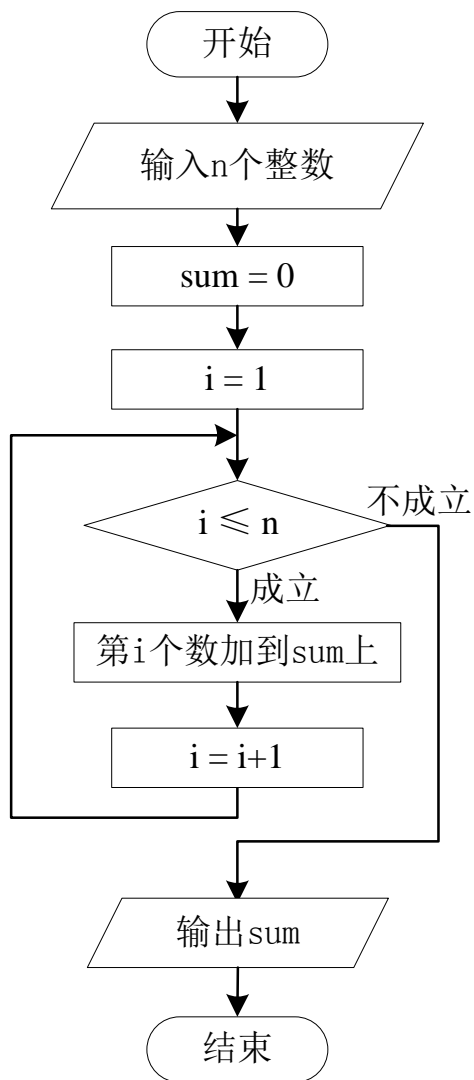
b) 选择结构



c) 循环结构

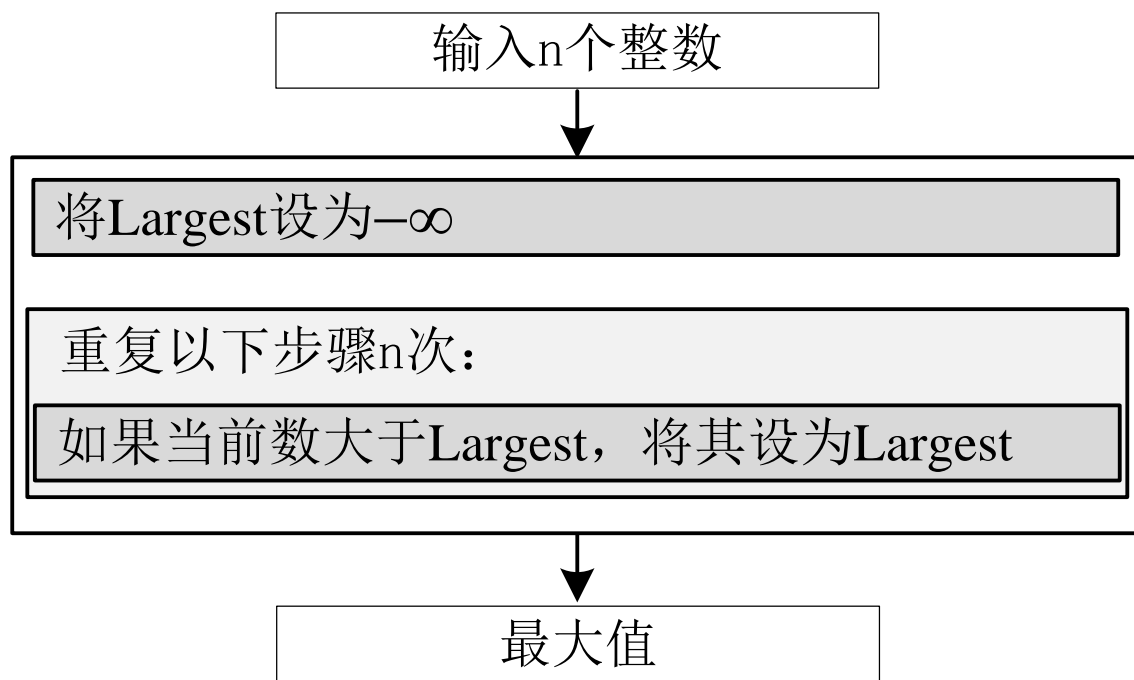
流程图

例4.1 用流程图的形式表示求若干个整数和的算法。



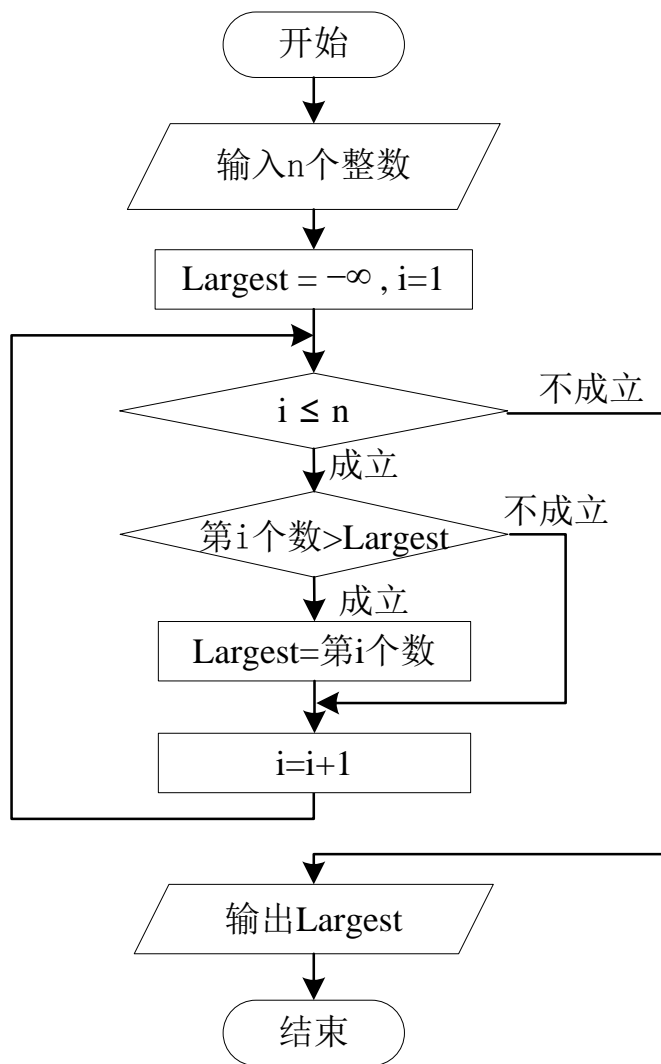
随堂练习

根据自然语言描述的求最大值算法，用流程图的形式表示。（在纸张上画出流程图拍照上传）



流程图

例4.2 用流程图的形式表示求最大值算法。



4.3 算法的表示

4.3.1 流程图



4.3.2 伪代码

伪代码

伪代码

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法。它如同一篇文章一样，自上而下地写下来。每一行(或几行)表示一个基本操作。它不用图形符号，因此书写方便，格式紧凑，修改方便，容易看懂，也便于向计算机语言算法(即程序)过渡。

伪代码

如何用伪代码表示算法的三种逻辑结构？

```
操作1  
操作2  
... ..  
操作n
```

顺序结构

```
If(条件)  
{  
    操作集A  
}  
else  
{  
    操作集B  
}
```

选择结构

```
While(条件)  
{  
    操作集A  
}
```

循环结构

伪代码

例4.3 用伪代码写出求两个整数之和的算法。

算法: **SumOfTwo**(first, second)

目的: 求两个整数之和

前提: 给定两个整数(first和second)

后续: 无

返回: 和的值

```
{  
    sum ← first + second  
    return sum  
}
```

伪代码

例4.4 编写将数字型成绩(整数)变为字母等级成绩的算法。

算法: **LetterGrade(score)**

目的: 给定成绩, 找到相应字母等级

前提: 给定成绩

后续: 无

返回: 字母等级

```
{  
    if (100 ≥ score ≥ 90)    grade ← 'A'  
    if (89 ≥ score ≥ 80)     grade ← 'B'  
    if (79 ≥ score ≥ 70)     grade ← 'C'  
    if (69 ≥ score ≥ 60)     grade ← 'D'  
    if (59 ≥ score ≥ 0)      grade ← 'E'  
    return grade  
}
```

课后练习

1. 用流程图形式表示求一组整数乘积的算法。
2. 用流程图形式表示求一组整数中最小数的算法。
3. 用伪代码表示求下面函数的算法(根据输入的x求相应的y值)

$$y = \begin{cases} x & x > 0 \\ 2x + 1 & x = 0 \\ 3x^2 + 1 & x < 0 \end{cases}$$

目录

CONTENTS

- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

4.4 基本算法

4.4.1 求和



4.4.2 乘积

4.4.3 最大和最小

4.4.4 排序

4.4.5 查找

4.4 基本算法

4.4.1 求和



4.4.2 乘积

4.4.3 最大和最小

4.4.4 排序

4.4.5 查找

基本算法——求和

求和算法可分为三个逻辑部分：

1. 将和(sum)初始化，即 $\text{sum}=0$ ；
2. 循环，在每次迭代中将一个新数加到和(sum)上；
3. 退出循环，返回结果。

4.4 基本算法

4.4.1 求和



4.4.2 乘积

4.4.3 最大和最小

4.4.4 排序

4.4.5 查找

基本算法——乘积

求一系列整数的乘积的方法和求和算法类似，也分为三个逻辑部分：

1. 将乘积(product)初始化，即 $\text{product}=1$ ；
2. 循环，在每次迭代中将一个新数与乘积(product)相乘；
3. 退出循环，返回结果。

4.4 基本算法

4.4.1 求和



4.4.2 乘积

4.4.3 最大和最小

4.4.4 排序

4.4.5 查找

基本算法——最大和最小

求最大和最小算法的共同点

算法包含一个循环结构来遍历所有输入的数，而在每次循环中，使用选择结构求两个数中的最大(或最小)值。

求最大和最小算法的不同点

最大值算法使用一个很小的数作为初始值，而最小值算法使用一个很大的数作为初始值。

4.4 基本算法

4.4.1 求和



4.4.2 乘积

4.4.3 最大和最小

4.4.4 排序

4.4.5 查找

排序

排序

排序根据数据的值对它们进行排列，把一组“无序”的数据变为“有序”的数据。

基本排序算法

- 选择排序、冒泡排序、插入排序。

术语和概念

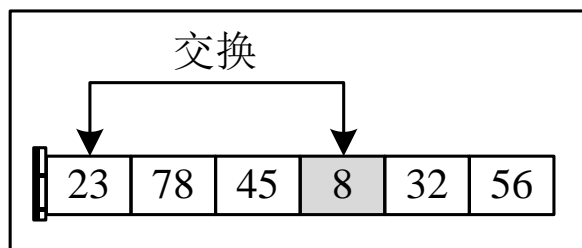
- 输入的是一列待排序的数，把这列数称为**数字列表**。
- 列表中的每个数称为列表的**元素**。
- 元素的**位置**在排序过程中可能改变。

排序——选择排序

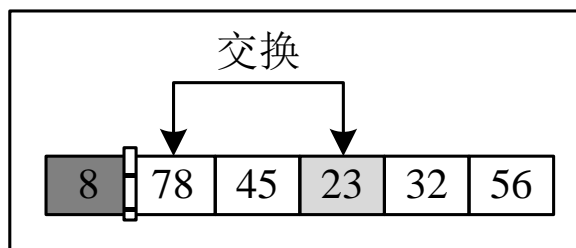
选择排序算法的思路

- 把数字列表分成两个子列表：**已排序的和未排序的**，它们之间通过假象的一堵**墙**分开。
- 从未排序的子列表中找到最小的元素，并与未排序的子列表的第一个元素交换，然后把假象的这堵墙向前移动一个元素，这样就完成了一轮**排序**。
- 一轮排序的结果是已排序子列表中增加了一个元素，而未排序的子列表中减少了一个元素。
- 重复上述过程，直到数字列表有序。

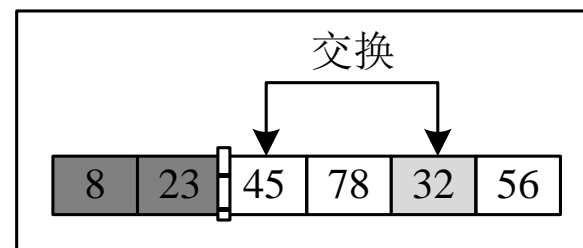
排序——选择排序



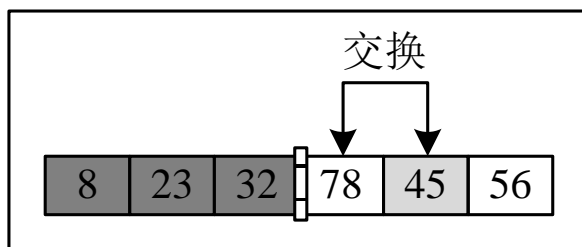
初始列表



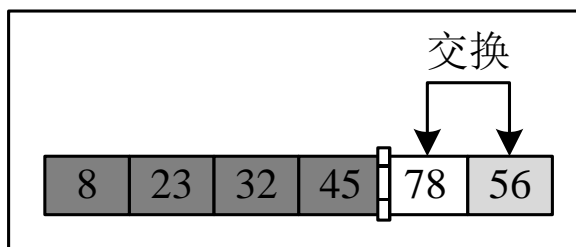
第1轮后



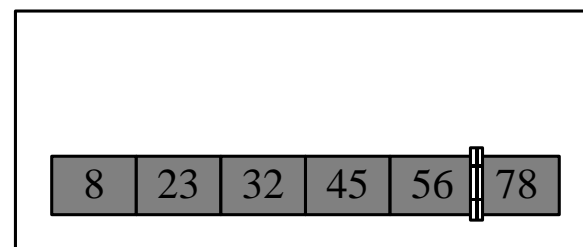
第2轮后



第3轮后



第4轮后



第5轮后

随堂练习

一个含有 n 个元素的数字列表，选择排序需要多少轮排序来完成数据的重新排列？

A $n-2$

B $n-1$

C n

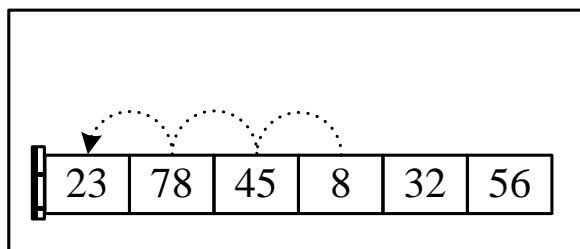
D $n+1$

排序——冒泡排序

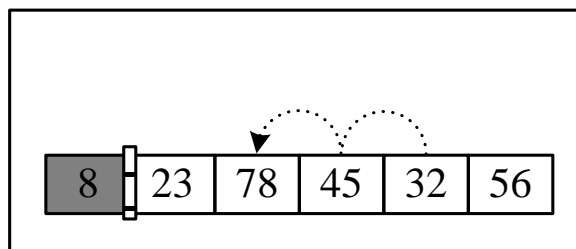
冒泡排序算法的思路

- 把数字列表分成两个子列表：**已排序的和未排序的**，它们之间通过假象的一堵**墙**分开。
- 在未排序子列表中，最小的元素通过**冒泡**的方法选出。
- 墙向前移动一个元素，把最小的元素移到已排序列表，这样完成一轮排序。
- 重复上述过程，直到数字列表有序。

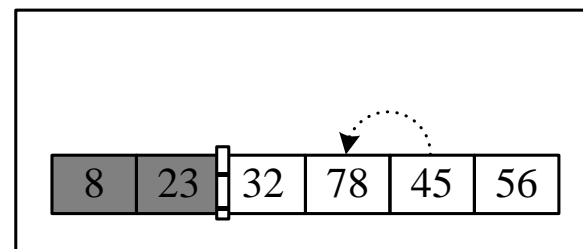
排序—冒泡排序



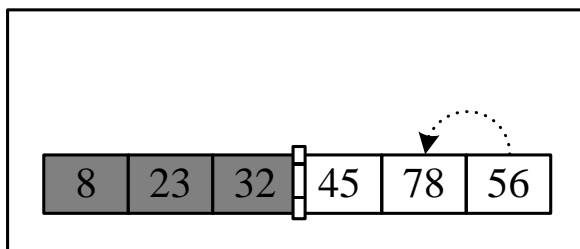
初始列表



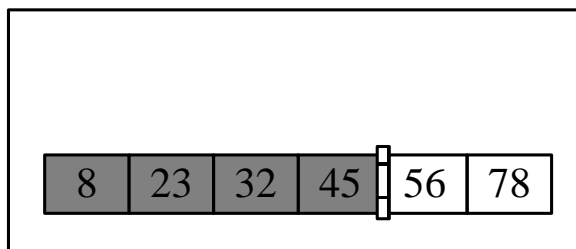
第1轮后



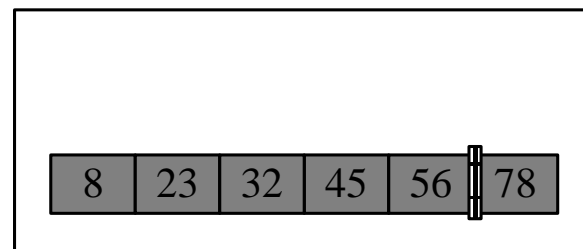
第2轮后



第3轮后



第4轮后



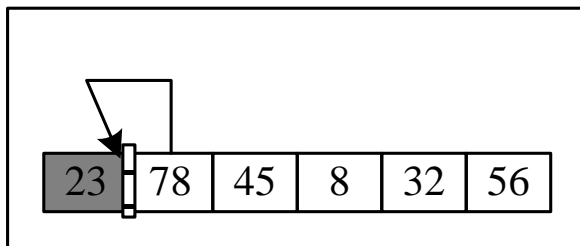
第5轮后

排序——插入排序

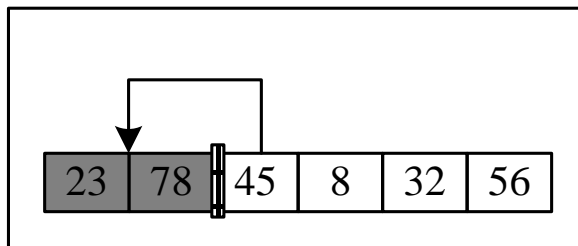
插入排序算法的思路

- 把数字列表分成两个子列表：**已排序的和未排序的**，它们之间通过假象的一堵**墙**分开。
- 在每轮排序中，把未排序子列表中的第一个元素移到已排序子列表中，并且**插入**到合适的位置。
- 重复上述过程，直到数字列表有序。

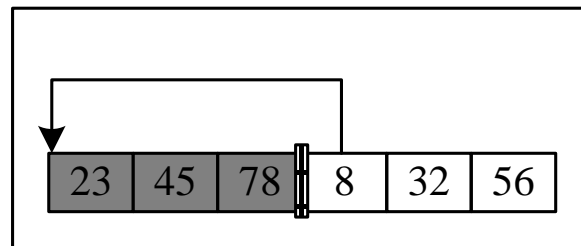
排序——插入排序



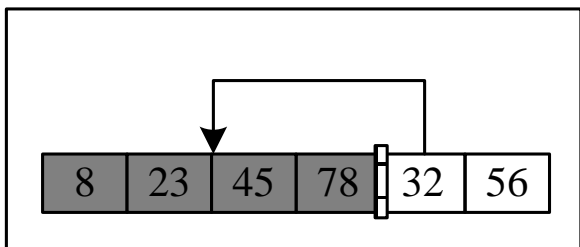
初始列表



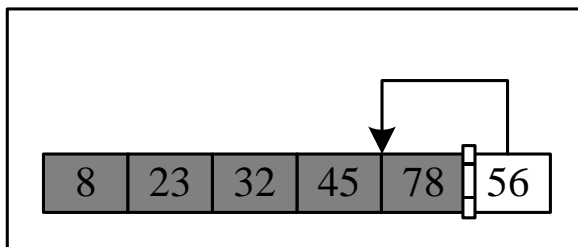
第1轮后



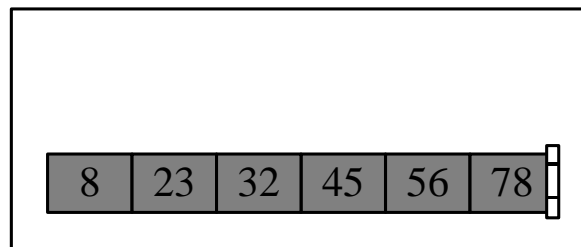
第2轮后



第3轮后



第4轮后



第5轮后

排序—思考和练习

思考

分析三种排序算法的异同点。

相同点：

- 把数字列表分成已排序和未排序子列表；
- 每轮排序后已排序子列表增加一个元素，未排序子列表减少一个元素；
- n 个元素排序共需 $n-1$ 轮。

不同点：

- 插入排序和其它两种排序算法的初始化不同；
- 选择和冒泡排序每一轮后，未排序子列表中的元素比已排序子列表中的元素大，而插入排序则不一定。

排序—小结

排序小结

- 选择排序、冒泡排序、插入排序是最简单、最基础的算法。
- 算法的**运行时间**较长，即**效率**较低，尤其是在排序的列表中有大量元素的情况下。
- 这三种算法是其它更高效算法，如快速排序、堆排序、合并排序等算法的基础。

4.4 基本算法

4.4.1 求和



4.4.2 乘积

4.4.3 最大和最小

4.4.4 排序

4.4.5 查找

查找

查找

查找是一种在列表中确定目标所在位置的算法。

查找算法

- 顺序查找和折半查找。

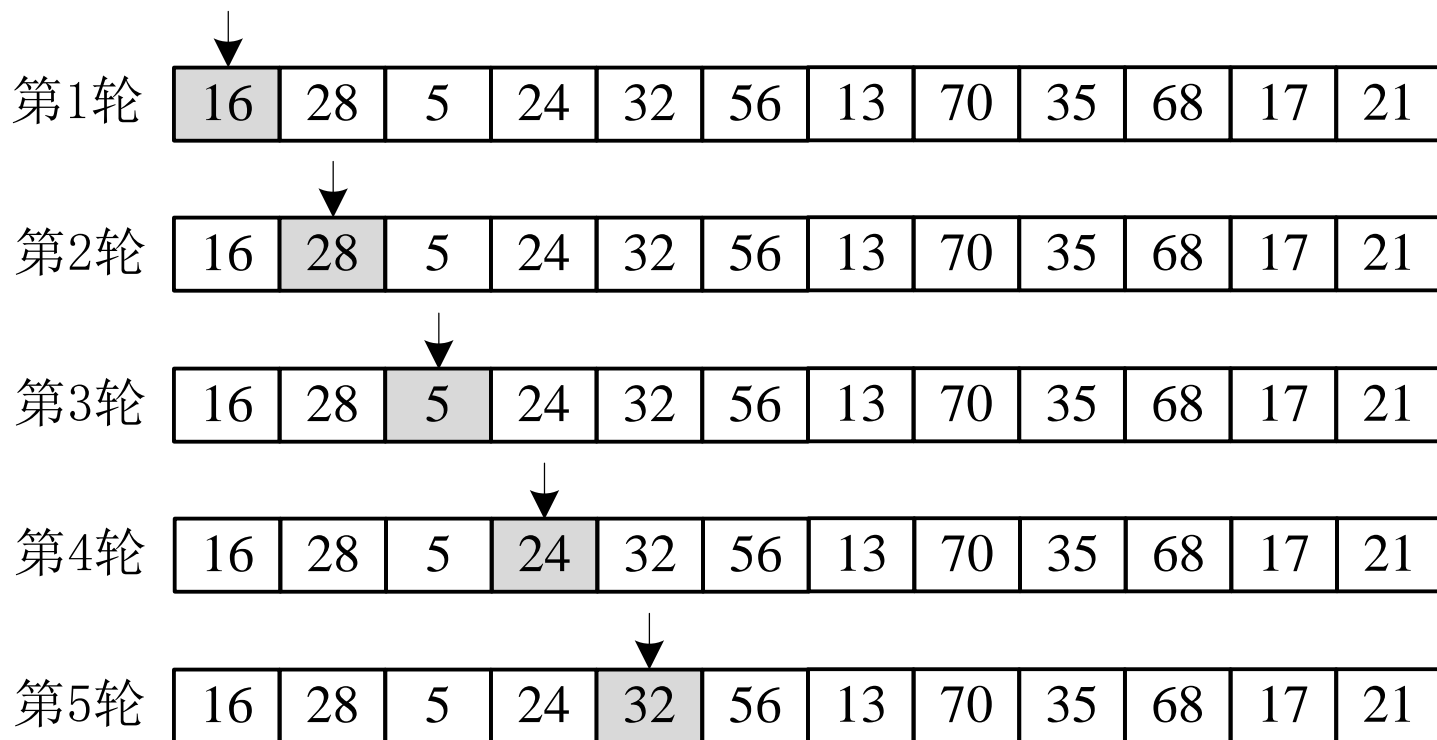
查找—顺序查找

顺序查找

顾名思义，顺序查找就是依次检查列表中的每个元素，和查找的目标值进行比对，直到找到目标值；或者检查完所有的元素，没有发现目标值。

查找—顺序查找

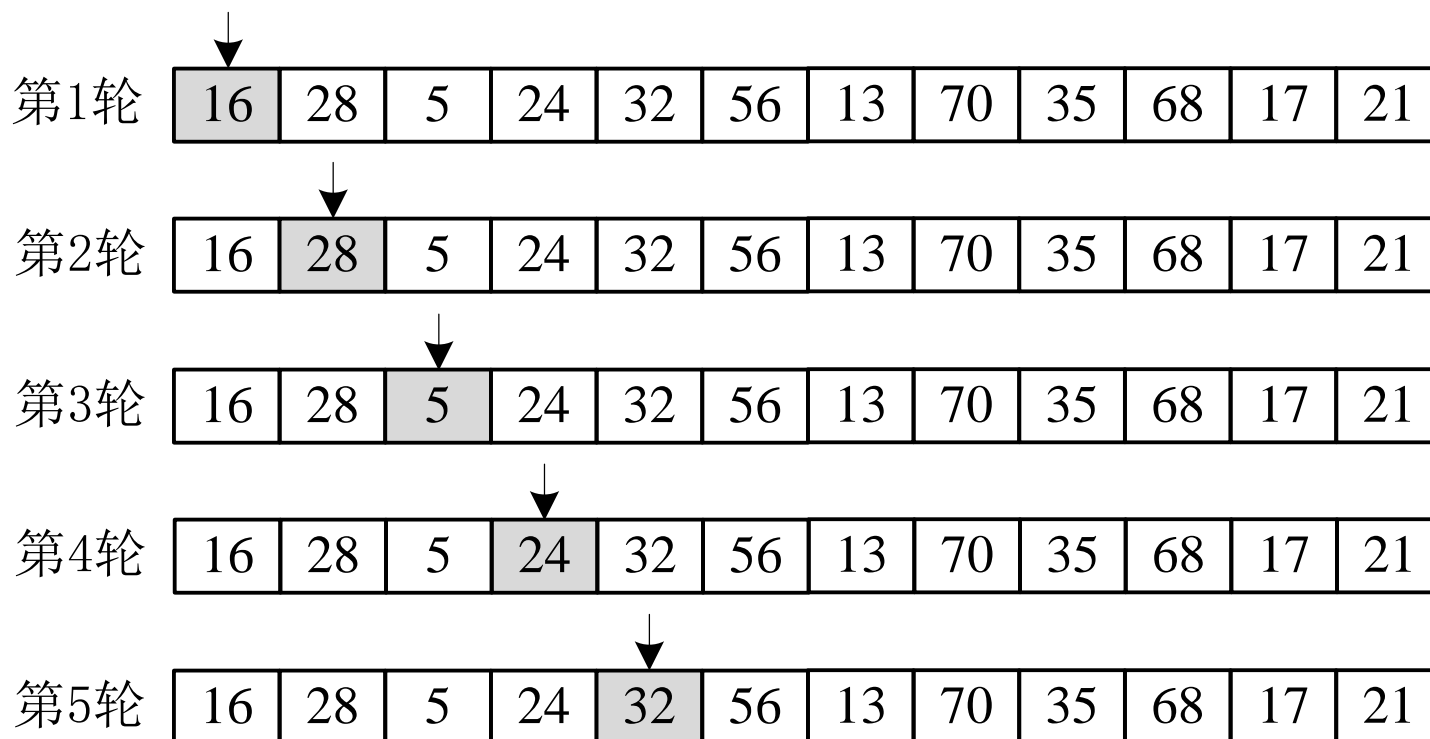
如何在16, 28, 5, 24, 32, 56, 13, 70, 35, 68, 17, 21中找到32?



找到目标32用了几次比较?

查找—顺序查找

如何查找的目标是19呢？



查找目标19用了几次比较？

顺序查找—思考和练习

思考

- 如果在一个具有 n 个整数的列表中查找一个数，最坏情况下需要进行多少次比较？

有没有更快的查找办法？

3	5	7	8	9	12	15
---	---	---	---	---	----	----

如何快速找到9？

查找—折半查找

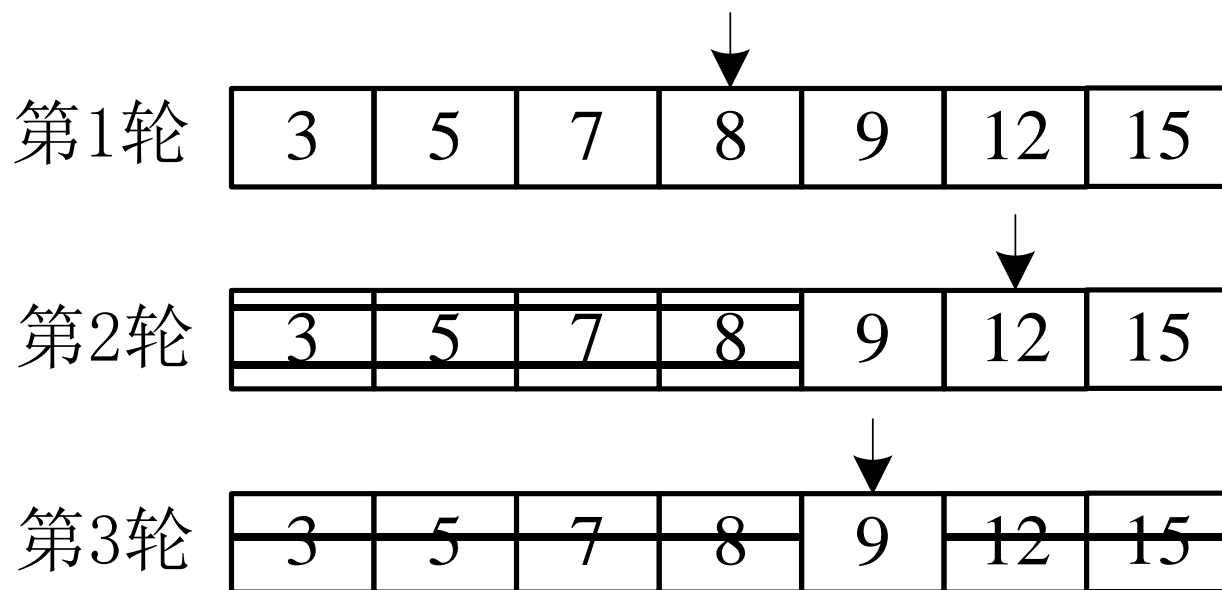
折半查找的前提

- 列表有序

折半查找的思路

- 先和列表的中间元素比较，如果目标值比中间元素小，则只要在列表的前半部分查找；
- 如果目标值比中间元素大，则只要在列表的后半部分查找；
- 如果目标值等于中间元素，则返回目标值的位置，查找结束。

查找—折半查找



随堂练习

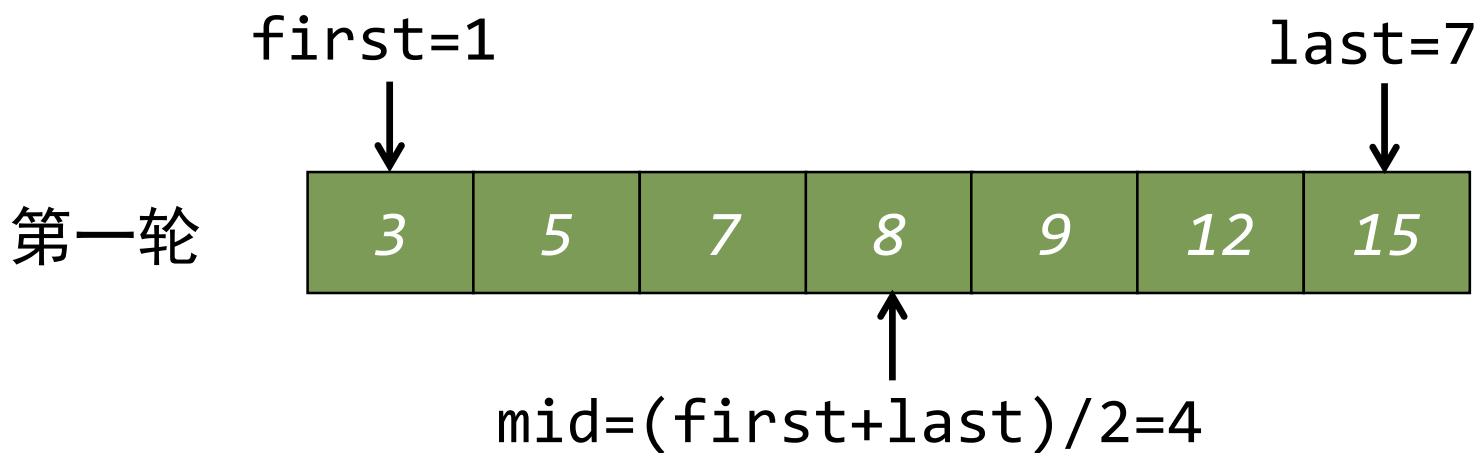
在折半查找过程中，搜索范围是怎么变化的？

- ☐ *A* 比原来减少了1个
- ☐ *B* 比原来减少了2个
- ☐ *C* 比原来减少了二分之一
- ☐ *D* 比原来减少了四分之一

折半查找—思考和练习

思考和练习

➤如果用first,last指示搜索的范围，用mid指示中间元素位置，那么这三个变量在每轮查找后该如何变化？

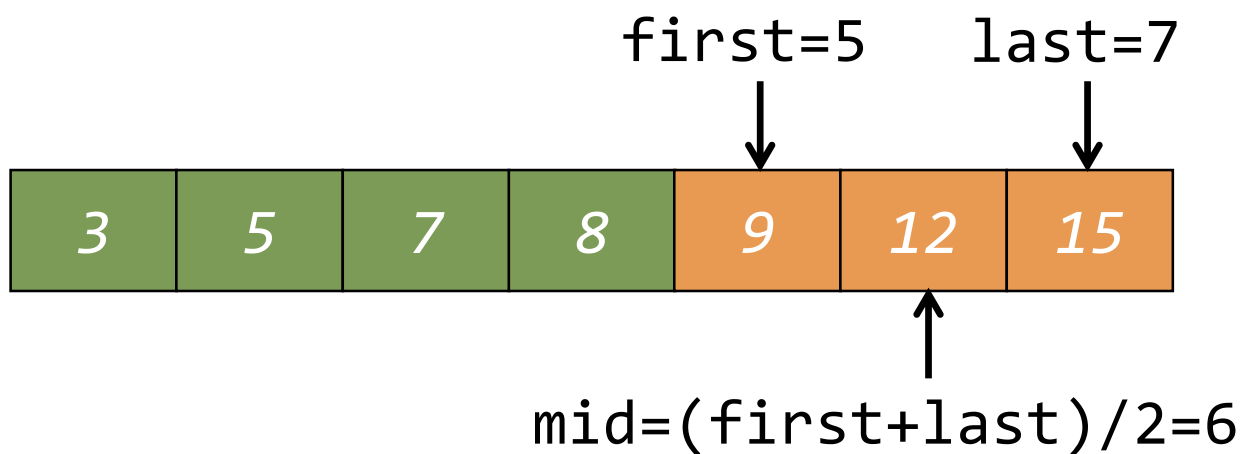


折半查找—思考和练习

思考和练习

➤如果用first, last指示搜索的范围，用mid指示中间元素位置，那么这三个变量在每轮查找后该如何变化？

第二轮

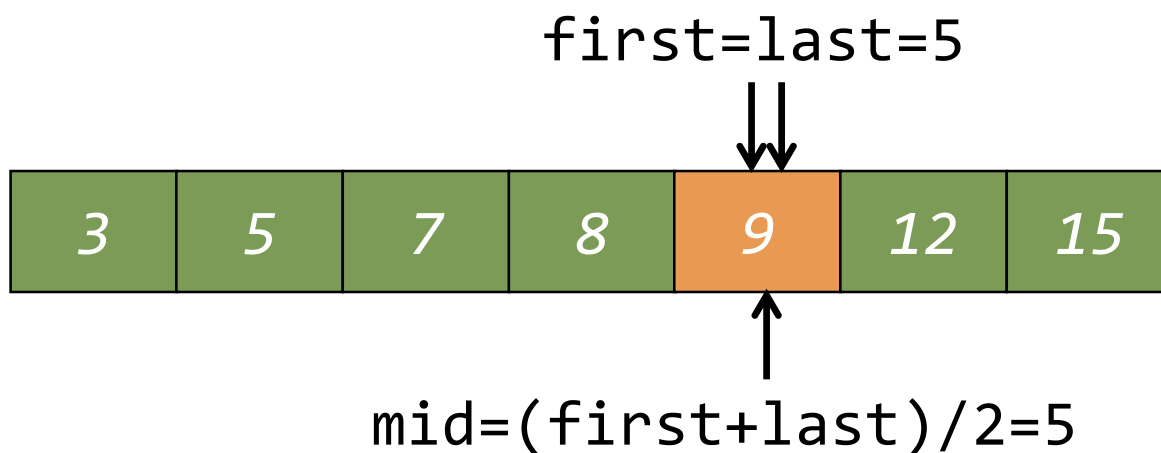


折半查找—思考和练习

思考和练习

➤如果用first, last指示搜索的范围，用mid指示中间元素位置，那么这三个变量在每轮查找后该如何变化？

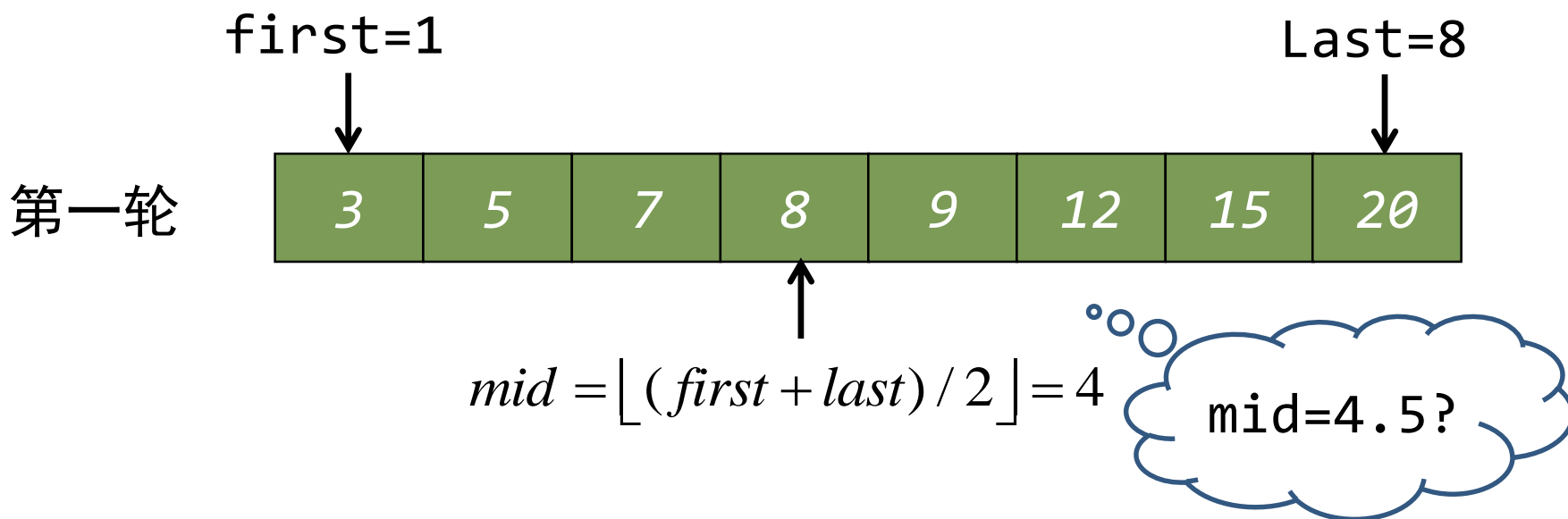
第三轮



折半查找—思考和练习

思考和练习

➤如果列表中的元素数是偶数呢？



目录

CONTENTS

- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

算法与子算法

子算法的引入

- 为了解决一个复杂问题，通常将它分解成若干个子问题求解；
- 针对每个子问题设计算法，将这些算法组合以完成最终的设计要求。
- 每个子问题所对应的算法，对整个算法来说，就是子算法。

问题：如何分解问题？或者，分解问题的基本原则是什么？

- 基本原则就是**功能相对独立**，即将复杂问题中一个独立的任务划分为一个子问题。

算法与子算法

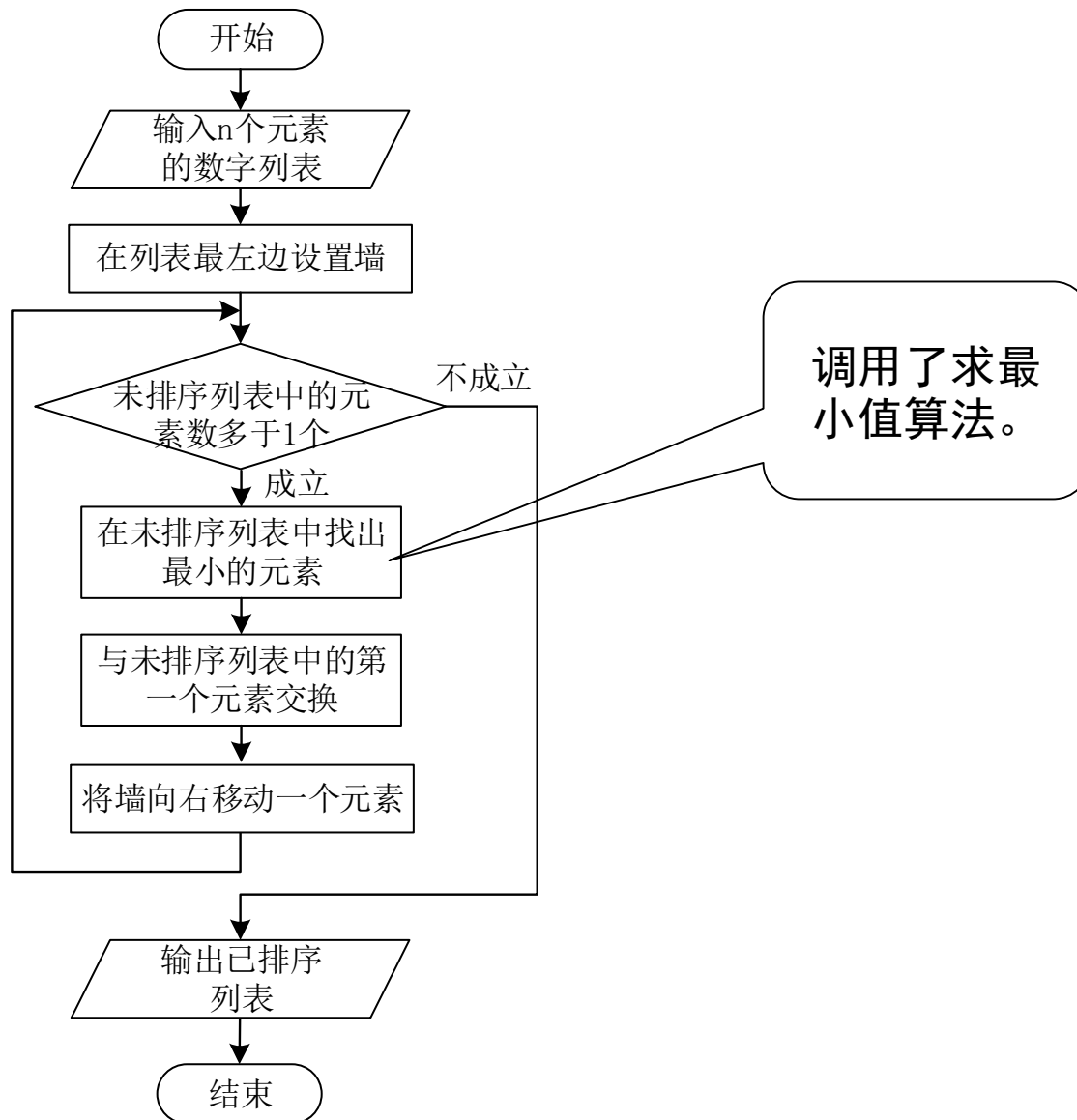
子算法

把这种完成独立任务的算法称为子算法。

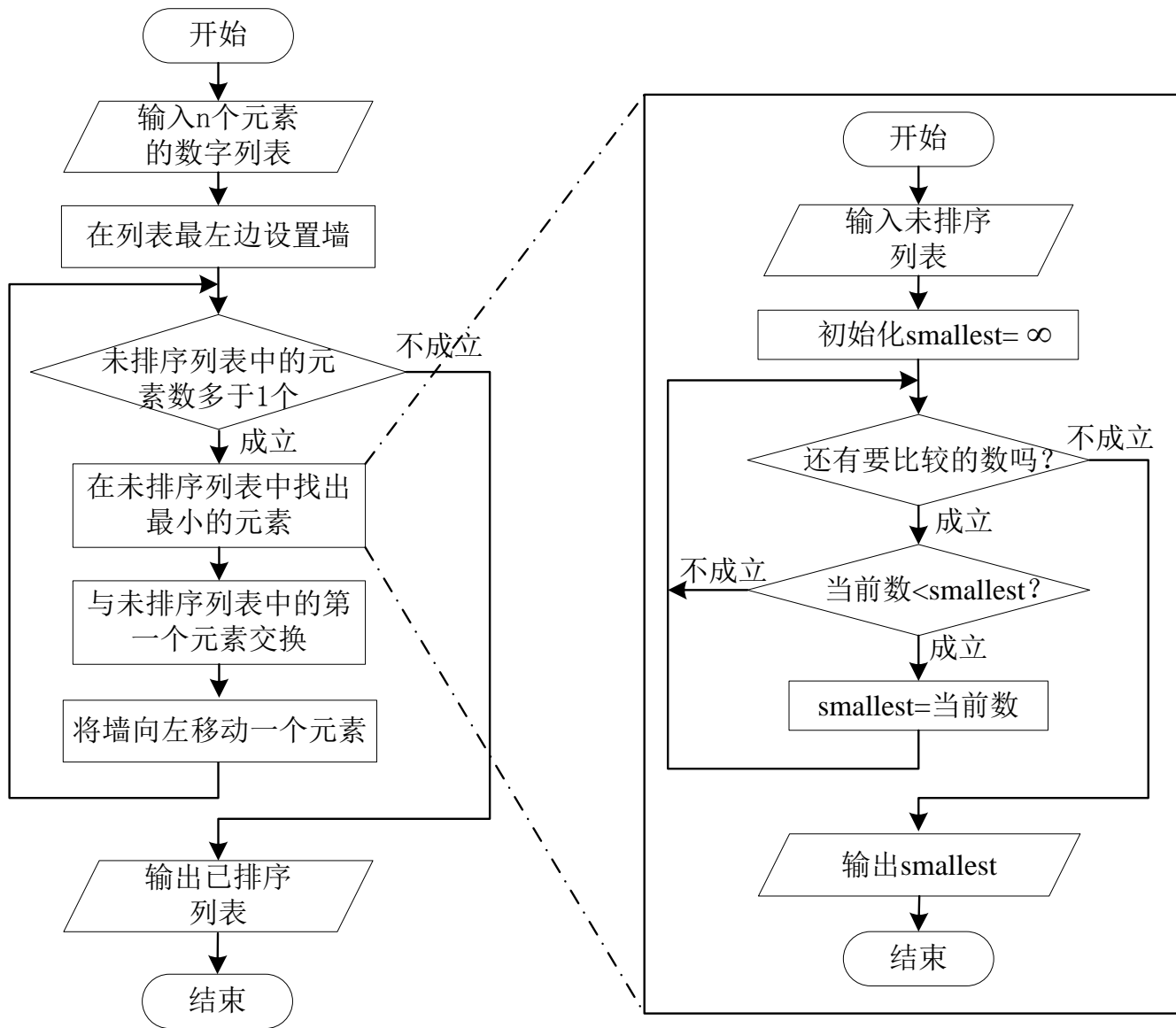
子算法的优点

- 使用子算法的优点包括使算法(或程序)更容易理解。
- 子算法可以在主算法的不同地方调用，而无需重复。

算法与子算法



算法与子算法



目录

CONTENTS

- 4.1 - 算法的概念
- 4.2 - 算法的三种逻辑结构
- 4.3 - 算法的表示
- 4.4 - 基本算法
- 4.5 - 算法与子算法
- 4.6 - 迭代与递归

迭代与递归

迭代和递归是两种在计算机科学中广泛使用的基本设计方法，虽然两者都用于重复性操作的处理，但是两种方法有各自的特点。

迭代利用计算机运算速度快、适合做重复性操作的特点，让计算机对一组操作进行重复执行，在每次执行这组操作时，都将从原值推出它的一个新值。

迭代与递归

例如，迭代求 $1+2+3\dots+n$ 的过程，设 $n=5$ ， $sum=0$

$$0 + 1 = 1$$

$$1 + 2 = 3$$

$$3 + 3 = 6$$

$$6 + 4 = 10$$

$$10 + 5 = 15$$

迭代与递归

递归则是一种算法自我调用的过程。当一个算法在执行过程中，又直接或间接调用了自身算法，这个过程就是递归。

使用递归需具备以下条件：

1. 原问题可以通过转化为较小的子问题来解决，而子问题的求解方法与原问题相同，被处理的数据有规律地减少。
2. 当子问题小至一定程度时，调用自身算法的过程会终止。

迭代与递归

例如，用递归求 $1+2+3\dots+n$ 的过程，设 $n=5$ ， $\text{sum}=0$

$\text{sum}(5)$

$5+\text{sum}(4)$

$5+4+\text{sum}(3)$

$5+4+3+\text{sum}(2)$

$5+4+3+2+\text{sum}(1)$

$5+4+3+2+1+\text{sum}(0)$

$5+4+3+2+1+0$

$5+4+3+2+1$

$5+4+3+3$

$5+4+6$

$5+10$

15

迭代与递归

问题：如何计算一个数的阶乘？

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \times (n-1) \times (n-2) \cdots 3 \times 2 \times 1 & \text{if } n>0 \end{cases}$$

迭代求阶乘算法

算法: **Factorial**(n)

目的: 使用循环求一个整数的阶乘

前提: 给定 n

后续: 无

返回: $n!$

```
{  
     $F \leftarrow 1$   
     $i \leftarrow 1$   
    while( $i \leq n$ )  
    {  
         $F \leftarrow F \times i$   
         $i \leftarrow i + 1$   
    }  
    return  $F$   
}
```

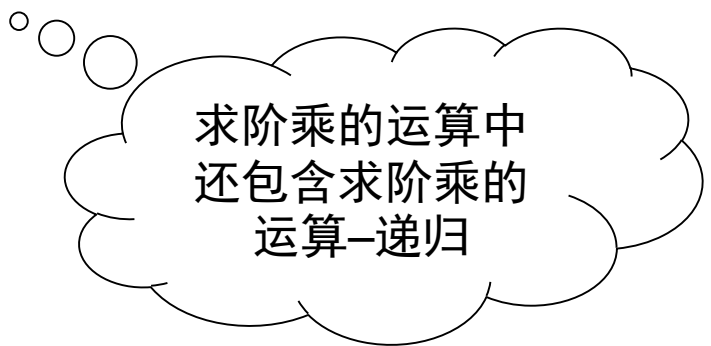
迭代与递归

按照定义式求阶乘：

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \times (n-1) \times (n-2) \cdots 3 \times 2 \times 1 & \text{if } n>0 \end{cases}$$

另外一种求阶乘的方法：

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \times (n-1)! & \text{if } n>0 \end{cases}$$



求阶乘的运算中
还包含求阶乘的
运算-递归

递归求阶乘算法

算法: **Factorial(n)**

目的: 使用递归求一个整数的阶乘

前提: 给定n

后续: 无

返回: $n!$

```
{  
    if(n=0)  
        return 1  
    else  
        return  $n \times \text{Factorial}(n-1)$ ○○  
}
```



调用了自身

迭代与递归—思考和练习

迭代和递归小结

- 递归算法的优点是算法(或程序)变得更简单和漂亮，也便于分析。
- 缺点是理解起来比较困难；效率不如迭代算法高。

课后练习

1. 若列表包含以下元素：8 23 12 6 100 -7 10。分别用选择排序、冒泡排序和插入排序对该列表排序，并给出排序过程。
2. 若列表包含以下元素：4 12 17 23 45 67 88。用折半查找法分别查找元素17和73，请给出查找过程(用图示，并给出每一轮first,mid,last三个量的值)。
3. 分别用迭代和递归算法求5!，给出计算过程。