数据结构

第 5 周 – 排序 I：冒泡排序、选择排序和插入排序

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **二分查找：** O(log n) – 但只适用于**有序**数组！
- 在数组中移动元素代价很高：插入、删除 **O(n)**
- 字符串比较：**字典序**

今天的问题

- 怎样把一个数组排好序?
- 需要多少次比较和移动?
- 什么时候该选哪一种简单排序?

排序问题

已知： **n** 个可以比较（≤ 关系）的元素组成的序列。

任务： 重新排列这些元素，使 A[1] ≤ A[2] ≤ … ≤ A[n]。

为什么重要?

- **快速查找：** 在有序数组上二分查找 – O(log n)
- **分组：** 相等的元素排在一起（重复元素、频率）
- **展示：** 按字母排序的名单、排行榜、时间顺序
- 据估计，计算机全部运行时间中有相当一部分花在排序上

基本概念

- **比较：**考察两个元素的关系（`A[i] < A[j]`） – 排序的主要代价
- **移动 / 交换：**改变元素的位置 – 一次交换是 3 次赋值
- **原地排序：**只需要 O(1) 的额外内存
- **稳定排序：**键相等的元素**保持**原来的相对顺序
- **比较排序：**只通过比较来决定元素的顺序

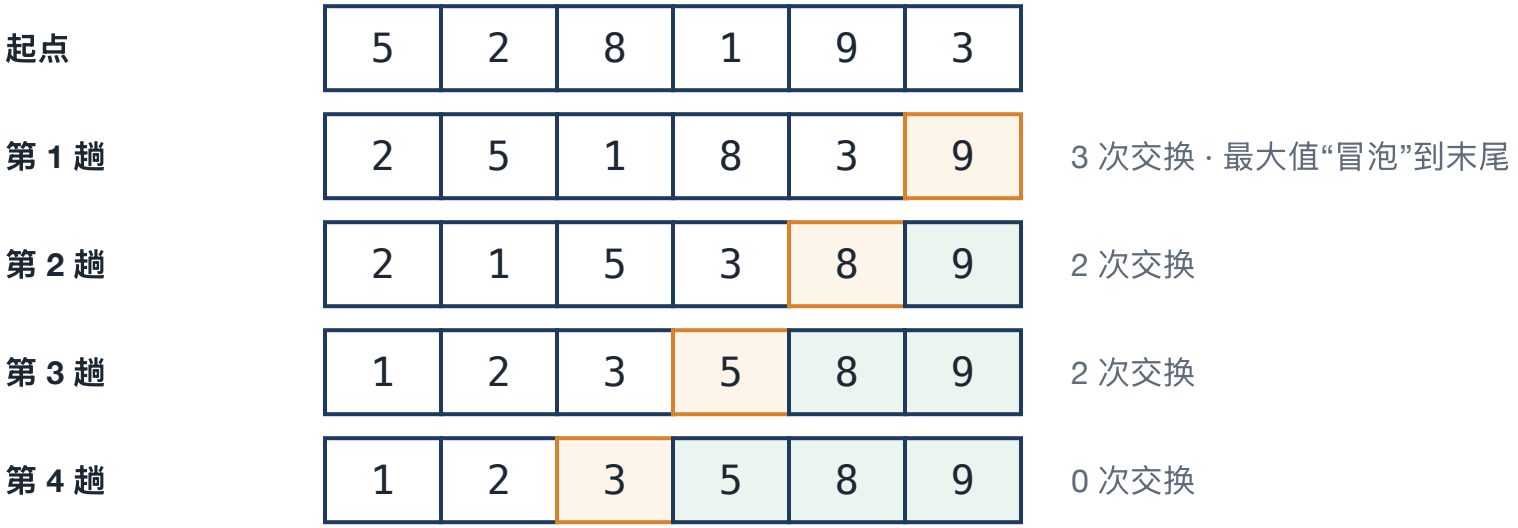
> 今天我们学习三种 $O(n^2)$ 步的**简单**排序算法。更快的 O(n log n) 方法在第 13 周讲解。

冒泡排序

冒泡排序 – 思路

- 遍历数组，比较**相邻**的元素
- 如果顺序不对，就**交换**它们
- 一趟之后**最大**的元素到达数组末尾（它“冒泡”上去）
- 在未排序的部分上重复

冒泡排序：每一趟交换相邻的逆序对



冒泡排序 – 算法

伪代码

```
Procedure BubbleSort(A, n)
  For i ← n down to 2
    swapped ← false
    For j ← 1 to i − 1
      If A[j] > A[j + 1] then
        swap(A[j], A[j + 1])
        swapped ← true
      End if
    End for
    If not swapped then
      exit the loop          // 退出循环
  End for
End procedure
```

Java

```java
for (int i = n - 1; i >= 1; i--) {
    boolean swapped = false;
    for (int j = 0; j < i; j++)
        if (a[j] > a[j + 1]) {
            int t = a[j];
            a[j] = a[j + 1];
            a[j + 1] = t;
            swapped = true;
        }
    if (!swapped) break;
}
```

Go

```go
if a[j] > a[j+1] {
    a[j], a[j+1] = a[j+1], a[j]
    swapped = true
}
```

冒泡排序 – 分析

情况	比较次数	交换次数
最好（已经有序）	n – 1	0
最坏（逆序）	n(n – 1) / 2	n(n – 1) / 2
平均	~ n^2 / 2	~ n^2 / 4

- 步数：**O(n^2)** · 在有序数组上，有 `swapped` 标志时：**O(n)**
- **原地**排序，**稳定**（从不交换相等的元素）
- 交换次数 = **逆序对**（顺序错误的元素对）的个数

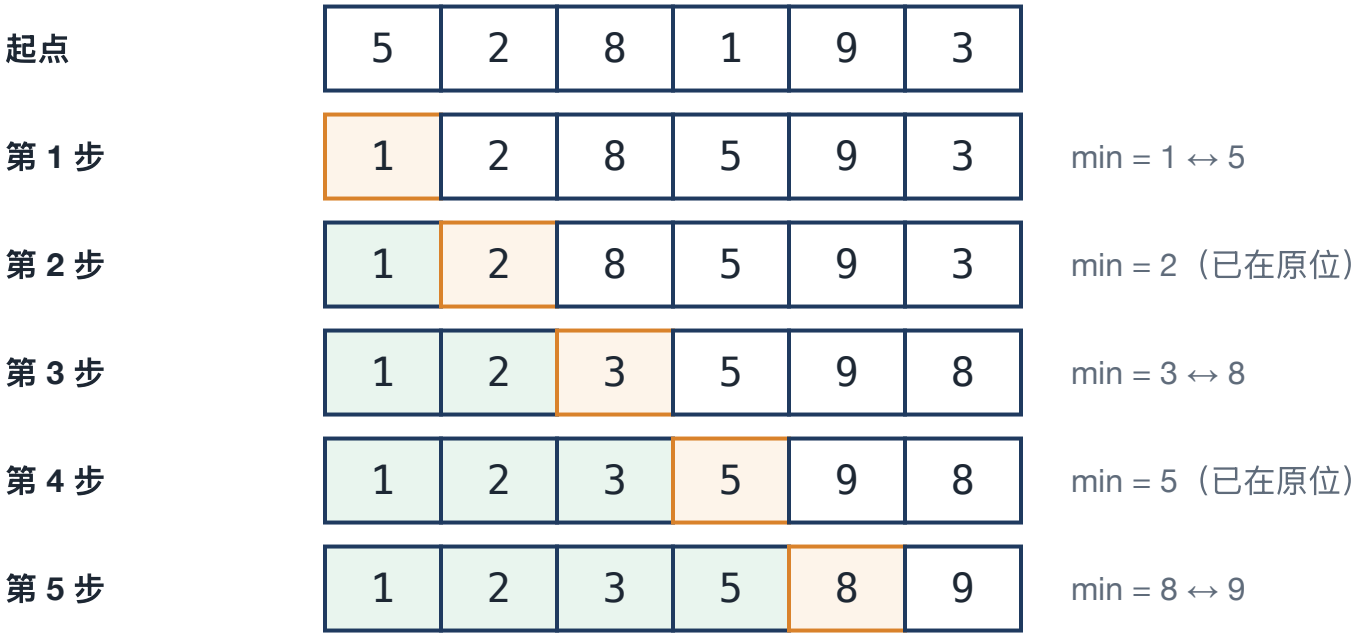
> 实际上它是简单排序中最慢的 – 交换次数太多。

选择排序

选择排序 – 思路

- 找出**最小**的元素，把它交换到**第一个**位置
- 在剩下的（未排序的）部分中找出最小的，放到**第二个**位置
- … 以此类推

选择排序：最小值放到未排序部分的最前面



选择排序 – 算法

伪代码

```
Procedure SelectionSort(A, n)
  For i ← 1 to n − 1
    min ← i
    For j ← i + 1 to n
      If A[j] < A[min] then
        min ← j
      End if
    End for
    swap(A[i], A[min])
  End for
End procedure
```

内层循环就是**求最小值**（第 2 周：求最大值）。

Java

```java
for (int i = 0; i < n − 1; i++) {
    int min = i;
    for (int j = i + 1; j < n; j++)
        if (a[j] < a[min]) min = j;
    int t = a[i];
    a[i] = a[min];
    a[min] = t;
}
```

Go

```go
for i := 0; i < len(a)-1; i++ {
    min := i
    for j := i + 1; j < len(a); j++ {
        if a[j] < a[min] {
            min = j
        }
    }
    a[i], a[min] = a[min], a[i]
}
```

选择排序 – 分析

情况	比较次数	交换次数
所有情况	n(n – 1) / 2	n – 1

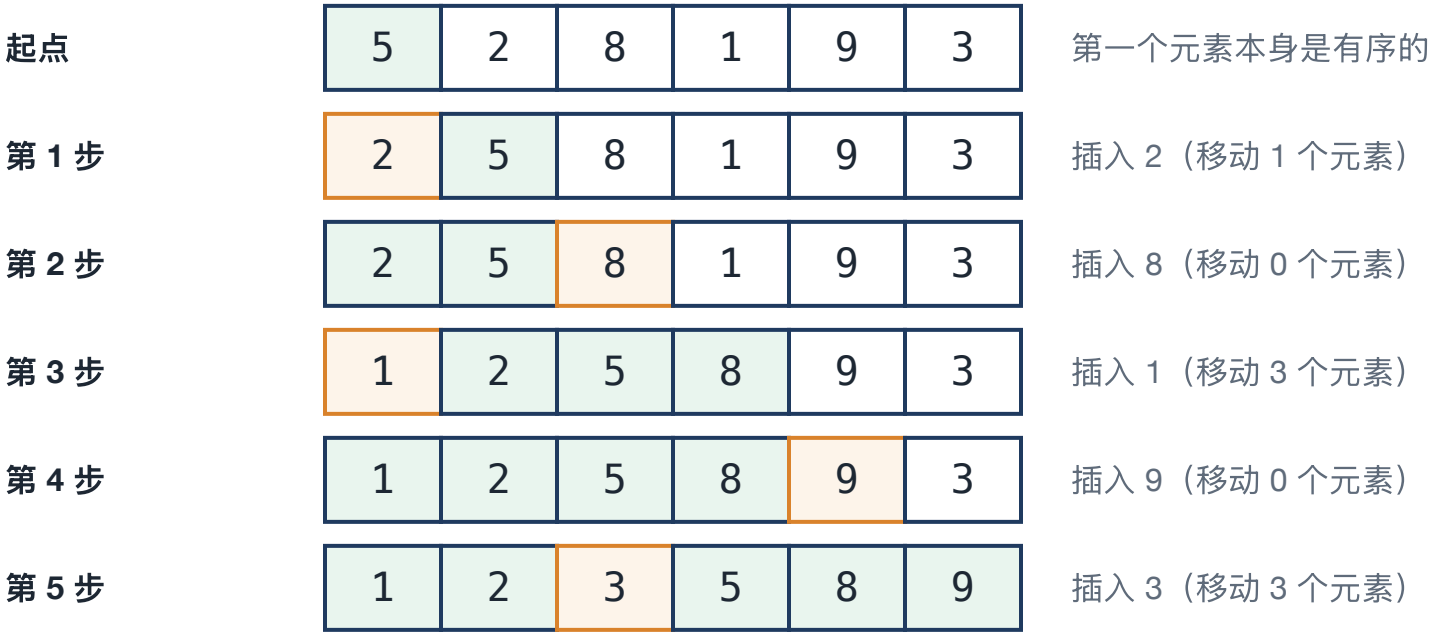
- 步数：**总是 $O(n^2)$** – 即使数组有序，也要扫描剩余部分
- 只有 **n – 1 次交换** – 移动代价很高时（大记录）很有用
- **原地**排序
- **不稳定**：远距离交换可能越过相等的元素
  `[5a, 5b, 2]` → `[2, 5b, 5a]`

插入排序

插入排序 – 思路

- 就像整理手中的扑克牌：把下一张牌**插入**到已经排好的牌中的正确位置
- 数组的前一部分是有序的；下一个元素向左移动，直到到达它的位置

插入排序：把下一个元素插入有序部分



插入排序 – 算法

伪代码

```
Procedure InsertionSort(A, n)
  For i ← 2 to n
    x ← A[i]
    j ← i − 1
    While j ≥ 1 and A[j] > x
      A[j + 1] ← A[j]
      j ← j − 1
    End while
    A[j + 1] ← x
  End for
End procedure
```

它不交换而是**移动**：一步就是一次赋值。

Java

```java
for (int i = 1; i < n; i++) {
    int x = a[i];
    int j = i - 1;
    while (j >= 0 && a[j] > x) {
        a[j + 1] = a[j];
        j--;
    }
    a[j + 1] = x;
}
```

Go

```go
for i := 1; i < len(a); i++ {
    x, j := a[i], i-1
    for j >= 0 && a[j] > x {
        a[j+1] = a[j]
        j--
    }
    a[j+1] = x
}
```

插入排序 – 分析

情况	比较次数	移动次数
最好（已经有序）	$n - 1$	$2(n - 1)$
最坏（逆序）	$n(n - 1) / 2$	$\sim n^2 / 2$
平均	$\sim n^2 / 4$	$\sim n^2 / 4$

- 步数：**$O(n^2)$**，在有序或**基本有序**的数组上 **$O(n)$**
- **原地**排序，**稳定**（用 `A[j] > x`，不能用 `≥`!）
- **在线算法：**可以按元素到达的顺序边接收边排序
- 在小数组上（$n < 20$）最快 – 专业的排序实现（Java、Go 类库）对小区间就使用它

比较

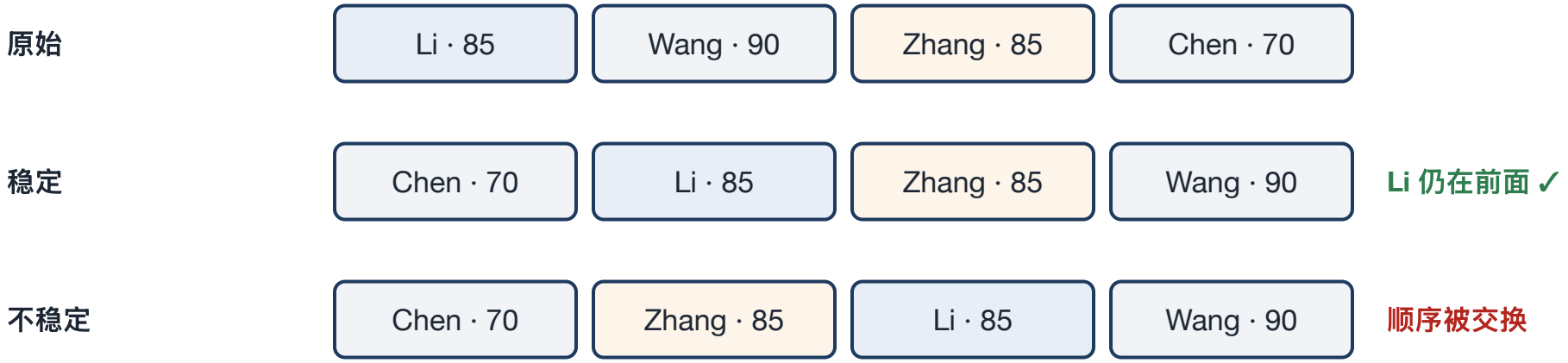
实测结果（n = 1000）

比较次数 / 移动次数（`SortCompare` 程序）：

	随机	有序	逆序
冒泡	499 004 / 750 423	**999 / 0**	499 500 / 1 498 500
选择	499 500 / **2 997**	499 500 / 2 997	499 500 / 2 997
插入	**251 585** / 252 589	**999 / 1 998**	499 500 / 501 498

> 移动 = 赋值（一次交换是 3 次赋值）。注意：选择排序的步数总是**相同**的，插入排序在有序数据上是**线性**的。

稳定性



- **为什么重要?** 如果先按姓名排序，再用**稳定**排序按分数排序，分数相同的学生仍然保持字母顺序
- 稳定：冒泡、插入 · 不稳定：选择

应该选哪一种?

	冒泡	选择	插入
最坏情况	$O(n^2)$	$O(n^2)$	$O(n^2)$
最好情况	$O(n)$	$O(n^2)$	$O(n)$
交换 / 移动	多	**少（n – 1）**	中等
稳定	是	否	是
原地	是	是	是

- 一般情况：**插入排序** – 简单，在基本有序的数据上很快
- 移动代价高时：**选择排序**
- 冒泡排序：主要用于教学

> 它们都不适合很大的 **n**:$n = 10^6$ 时约需 10^{12} 步。解决办法：**归并排序**和**快速排序**（第 13 周），$O(n \log n)$。

内置排序

Java

```java
int[] a = {5, 2, 8, 1};
Arrays.sort(a);   // 基本类型:双轴快速排序

String[] s = {"Wuhan", "Eger"};
Arrays.sort(s);   // 对象:TimSort(稳定)

list.sort(Comparator
    .comparing(Student::score));
```

Go

```go
a := []int{5, 2, 8, 1}
slices.Sort(a)     // pdqsort

sort.Strings(s)

slices.SortStableFunc(st,
    func(x, y Student) int {
        return x.Score - y.Score
    })
```

实际中我们使用类库的排序 – 但**必须了解它的工作原理**，以及它在什么情况下有多快。

小结

- **冒泡**：交换相邻元素，每一趟最大的元素到达末尾 · $O(n^2)$，稳定
- **选择**：最小值放到前面 · 总是 $O(n^2)$，只有 $n - 1$ 次交换，不稳定
- **插入**：把下一个元素插入有序部分 · $O(n^2)$，在有序数据上 $O(n)$，稳定
- 三者都是**原地**排序
- **稳定性**：键相等的元素保持原来的顺序

下周：集合数据结构 – 位向量、无序表示和有序表示、集合运算。

复习题

1. 在纸上用三种方法分别对数组 `[4, 1, 3, 2]` 排序！逐步写出各个状态！
2. 判断对错：选择排序在有序数组上只需 O(n) 步。
3. 哪种排序的交换次数最少？是多少？
4. 判断对错：冒泡排序是稳定的。
5. 对 10 个元素的逆序数组，插入排序进行多少次比较？
6. 如果插入排序用 `A[j] ≥ x` 代替 `A[j] > x`，会怎样？

实践课

✎ 示例程序

程序	Java	Go	演示内容
简单排序	`SimpleSorts.java`	`w05/simplesorts`	三种排序，逐步打印
比较	`SortCompare.java`	`w05/sortcompare`	不同输入下的比较次数、移动次数、时间

运行 `SortCompare` 程序！哪种排序依赖于输入，哪种不依赖？**n** 加倍时，运行时间怎样变化？

✎ 练习

1. **是否有序？** 用 O(n) 步判断数组是否按递增顺序排列！
2. **降序：** 把插入排序改写成降序排序！
3. **字符串：** 用选择排序对城市名数组排序！
4. **记录：** 按分数对学生（姓名、分数）进行**稳定**排序！
5. **最小的 k 个：** 用选择排序的前 **k** 步找出数组中最小的 **k** 个元素！需要多少步？
6. **计数排序：** 不用比较，用 6 个元素的计数数组对考试成绩（1–5）排序！需要多少步？
7. **逆序对：** 统计数组中顺序错误的元素对！它和冒泡排序有什么关系？

参考答案：`Exercises05.java`、`w05/exercises`

✎ 课后作业

1. **鸡尾酒排序：**冒泡排序的一种变体，交替地从左往右和从右往左遍历。实现它，并测量在数组 `[2, 3, 4, 5, 1]` 上能节省多少次比较！
2. **二分插入排序：**用二分查找（第 2 周）确定插入位置！比较次数和移动次数怎样变化？
3. 举一个例子证明选择排序**不稳定**！
4. 把学生**首先按分数**降序排列，**分数相同时按姓名**排列！（提示：先后进行两次稳定排序，或者使用组合比较。）