答疑

问 排序看起来是个很简单的问题，我们用计算机不是可以做很多更有意思的事情吗？

答 也许吧，但快速的排序算法才使得那些更有意思的事情成为可能。在2.5节以及全书的其他章节你都可以找到很多这样的例子。排序算法今天仍然值得我们学习是因为它易于理解，你能从中领会到许多精妙之处。

问 为什么有这么多排序算法？

答 原因之一是许多排序算法的性能都和输入模型有很大的关系，因此不同的算法适用于不同应用场景中的不同输入。例如，对于部分有序和小规模的数组应该选择插入排序。其他限制条件，例如空间和重复的主键，也都是需要考虑的因素。我们将会在2.5节中再次讨论这个问题。

问 为什么要使用 `less()` 和 `exch()` 这些不起眼的辅助函数？

答 它们抽象了所有排序算法都会用到的共同操作，这种抽象使得代码更便于理解。而且它们增强了代码的可移植性。例如，算法2.1和算法2.2中的大部分代码在其他几种编程语言中也是可以执行的。即使是在Java中，只要将 `less()` 实现为 `v < w`，这些算法的代码就可以将不支持 `Comparable` 接口的基本数据类型排序了。

问 当我运行 `SortCompare` 时，每次的结果都不一样（而且和书上的也不相同），为什么？

答 对于初学者，你的计算机和我们的计算机不同，操作系统、Java运行时环境等都不一样。这些不同可能导致算法代码生成的机器码不同。每次运行所得结果不同的原因可能在于当时运行的其他程序或是很多其他原因。大量的重复实验可以淡化这种干扰，我们的经验是现如今算法性能的微小差异很难观察。这就是我们要关注较大差异的原因。

263

练习

2.1.1 按照算法2.1所示轨迹的格式给出选择排序是如何将数组 E A S Y Q U E S T I O N 排序的。

2.1.2 在选择排序中，一个元素最多可能会被交换多少次？平均可能会被交换多少次？

2.1.3 构造一个含有 N 个元素的数组，使选择排序（算法2.1）运行过程中 `a[j] < a[min]`（由此 `min` 会不断更新）成功的次数最大。

2.1.4 按照算法2.2所示轨迹的格式给出插入排序是如何将数组 E A S Y Q U E S T I O N 排序的。

2.1.5 构造一个含有 N 个元素的数组，使插入排序（算法2.2）运行过程中内循环（`for`）的两个判断结果总是假。

2.1.6 在所有的主键都相同时，选择排序和插入排序谁更快？

2.1.7 对于逆序数组，选择排序和插入排序谁更快？

2.1.8 假设元素只可能有三种值，使用插入排序处理这样一个随机数组的运行时间是线性的还是平方级别的？或是介于两者之间？

2.1.9 按照算法2.3所示轨迹的格式给出希尔排序是如何将数组 E A S Y S H E L L S O R T Q U E S T I O N 排序的。

2.1.10 在希尔排序中为什么在实现 `h` 有序时不使用选择排序？

2.1.11 将希尔排序中实时计算递增序列改为预先计算并存储在一个数组中。

2.1.12 令希尔排序打印出递增序列的每个元素所带来的比较次数和数组大小的比值。编写一个测试用例对随机 `Double` 数组进行希尔排序，验证该值是一个小常数，数组大小按照10的幂次递增，

不小于 100。 264

提高题

2.1.13 *纸牌排序。*说说你会如何将一副扑克牌按花色排序（花色顺序是黑桃、红桃、梅花和方片），限制条件是所有牌都是背面朝上排成一列，而你一次只能翻看两张牌或者交换两张牌（保持背面朝上）。

2.1.14 *出列排序。*说说你会如何将一副扑克牌排序，限制条件是只能查看最上面的两张牌，交换最上面的两张牌，或是将最上面的一张牌放到这摞牌的最下面。

2.1.15 *昂贵的交换。*一家货运公司的一位职员得到了一项任务，需要将若干大货箱按照发货时间摆放。比较发货时间很容易（对照标签即可），但将两个货箱交换位置则很困难（移动麻烦）。仓库已经快满了，只有一个空闲的仓位。这位职员应该使用哪种排序算法呢？

2.1.16 *验证。*编写一个 `check()` 方法，调用 `sort()` 对任意数组排序。如果排序成功而且数组中的所有对象均没有被修改则返回 `true`，否则返回 `false`。不要假设 `sort()` 只能通过 `exch()` 来移动数据，可以信任并使用 `Arrays.sort()`。

2.1.17 *动画。*修改插入排序和选择排序的代码，使之将数组内容绘制成正文中所示的棒状图。在每一轮排序后重绘图片来产生动画效果，并以一张“有序”的图片作为结束，即所有圆棒均已按照高度有序排列。*提示：*使用类似于正文中的用例来随机生成 `Double` 值，在排序代码的适当位置调用 `show()` 方法，并在 `show()` 方法中清理画布并绘制棒状图。

2.1.18 *可视轨迹。*修改你为上一题给出的解答，为插入排序和选择排序生成和正文中类似的可视轨迹。*提示：*使用 `setYscale()` 函数是一个明智的选择。*附加题：*添加必要的代码，与正文中的图片一样用红色和灰色强调不同角色的元素。

2.1.19 *希尔排序的最坏情况。*用 1 到 100 构造一个含有 100 个元素的数组并用希尔排序和递增序列 1 4 13 40 对其排序，使比较的次数尽可能多。

2.1.20 *希尔排序的最好情况。*最好情况是什么？证明你的结论。 265

2.1.21 *可比较的交易。*用我们的 `Date` 类（请见 2.1.1.4 节）作为模板扩展你的 `Transaction` 类（请见练习 1.2.13），实现 `Comparable` 接口，使交易能够按照金额排序。

解答：

```
public class Transaction implements Comparable<Transaction>
{
   ...
   private final double amount;
   ...
   public int compareTo(Transaction that)
   {
      if (this.amount > that.amount) return +1;
      if (this.amount < that.amount) return -1;
      return 0;
   }
   ...
}
```

2.1.22 *事务排序测试用例。*编写一个 `SortTransaction` 类，在静态方法 `main()` 中从标准输入读取一系列事务，将它们排序并在标准输出中打印结果（请见练习 1.3.17）。

解答：

```
public class SortTransactions
{
   public static Transaction[] readTransactions()
   {  // 请见练习 1.3.17  }
   public static void main(String[] args)
   {
      Transaction[] transactions = readTransactions();
      Shell.sort(transactions);
      for (Transaction t : transactions)
         StdOut.println(t);
   }
}
```

266

实验题

2.1.23 *纸牌排序。*请几位朋友分别将一副扑克牌排序（见练习 2.1.13）。仔细观察并记录他们所使用的方法。

2.1.24 *插入排序的哨兵。*在插入排序的实现中先找出最小的元素并将其置于数组的最左边，这样就能去掉内循环的判断条件 `j>0`。使用 `SortCompare` 来评估这种做法的效果。注意：这是一种常见的规避边界测试的方法，能够省略判断条件的元素通常被称为哨兵。

2.1.25 *不需要交换的插入排序。*在插入排序的实现中使较大元素右移一位只需要访问一次数组（而不用使用 `exch()`）。使用 `SortCompare` 来评估这种做法的效果。

2.1.26 *原始数据类型。*编写一个能够处理 `int` 值的插入排序的新版本，比较它和正文中所给出的实现（能够隐式地用自动装箱和拆箱转换 `Integer` 值并排序）的性能。

2.1.27 *希尔排序的用时是次平方级的。*在你的计算机上用 `SortCompare` 比较希尔排序和插入排序以及选择排序。测试数组的大小按照 2 的幂次递增，从 128 开始。

2.1.28 *相等的主键。*对于主键仅可能取两种值的数组，评估和验证插入排序和选择排序的性能，假设两种主键值出现的概率相同。

2.1.29 *希尔排序的递增序列。*通过实验比较算法 2.3 中所使用的递增序列和递增序列 1，5，19，41，109，209，505，929，2161，3905，8929，16 001，36 289，64 769，146 305，260 609（这是通过序列 $9\times4^k-9\times2^k+1$ 和 $4^t-3\times2^k+1$ 综合得到的）。可以参考练习 2.1.11。

2.1.30 *几何级数递增序列。*通过实验找到一个 t，使得对于大小为 $N=10^6$ 的任意随机数组，使用递增序列 1，$\lfloor t\rfloor$，$\lfloor t^2\rfloor$，$\lfloor t^3\rfloor$，$\lfloor t^4\rfloor$，…的希尔排序的运行时间最短。给出你能找到的三个最佳 t 值以及相应的递增序列。

267

以下练习描述的是各种用于评估排序算法的测试用例。它们的作用是用随机数据帮助你增进对性能特性的理解。随着命令行指定的实验次数的增大，可以和 `SortCompare` 一样在它们中使用 `time()` 函数来得到更精确的结果。在以后的几节中我们会使用这些练习来评估更加复杂的算法。

2.1.31 *双倍测试。*编写一个能够对排序算法进行双倍测试的用例。数组规模 `N` 的起始值为 1000，排序后打印 `N`、估计排序用时、实际排序用时以及在 `N` 增倍之后两次用时的比例。用这段程序验证在随机输入模型下插入排序和选择排序的运行时间都是平方级别的。对希尔排序的性能作出猜想并验证你的猜想。

2.1.32 *运行时间曲线图。*编写一个测试用例，使用 StdDraw 在各种不同规模的随机输入下将算法的平均运行时间绘制成一张曲线图。可能需要添加一两个命令行参数，请尽量设计一个实用的工具。

2.1.33 *分布图。*对于你为练习 2.1.33 给出的测试用例，在一个无穷循环中调用 `sort()` 方法将由第三个

命令行参数指定大小的数组排序，记录每次排序的用时并使用 StdDraw 在图上画出所有平均运行时间，应该能够得到一张运行时间的分布图。

2.1.34 *罕见情况。*编写一个测试用例，调用 sort() 方法对实际应用中可能出现困难或极端情况的数组进行排序。比如，数组可能已经是有序的，或是逆序的，数组的所有主键相同，数组的主键只有两种值，大小为 0 或是 1 的数组。

2.1.35 *不均匀的概率分布。*编写一个测试用例，使用非均匀分布的概率来生成随机排列的数据，包括：

- 高斯分布；
- 泊松分布；
- 几何分布；
- 离散分布（一种特殊情况请见练习 2.1.28）。

评估并验证这些输入数据对本节讨论的算法的性能的影响。 268

2.1.36 *不均匀的数据。*编写一个测试用例，生成不均匀的测试数据，包括：

- 一半数据是 0，一半是 1；
- 一半数据是 0，1/4 是 1，1/4 是 2，以此类推；
- 一半数据是 0，一半是随机 int 值。

评估并验证这些输入数据对本节讨论的算法的性能的影响。

2.1.37 *部分有序。*编写一个测试用例，生成部分有序的数组，包括：

- 95% 有序，其余部分为随机值；
- 所有的元素和它们的正确位置的距离都不超过 10；
- 5% 的元素随机分布在整个数组中，剩下的数据都是有序的。

评估并验证这些输入数据对本节讨论的算法的性能的影响。

2.1.38 *不同类型的元素。*编写一个测试用例，生成由多种数据类型元素组成的数组，元素的主键值随机，包括：

- 每个元素的主键均为 String 类型（至少长 10 个字符），并含有一个 double 值；
- 每个元素的主键均为 double 类型，并含有 10 个 String 值（每个都至少长 10 个字符）；
- 每个元素的主键均为 int 类型，并含有一个 int[20] 值

评估并验证这些输入数据对本节讨论的算法的性能的影响。 269