比较次数相互处于常数因子范围之内。

对于标准的快速排序，随着数组规模的增大其运行时间会趋于平均运行时间，大幅偏离的情况非常罕见，因此可以肯定三向切分的快速排序的运行时间和输入的信息量的 N 倍是成正比的。在实际应用中这个性质很重要，因为对于包含大量重复元素的数组，它将排序时间从线性对数级降低到了线性级别。这和元素的排列顺序没有关系，因为算法会在排序之前将其打乱以避免最坏情况。元素的概率分布决定了信息量的大小，没有基于比较的排序算法能够用少于信息量决定的比较次数完成排序。这种对重复元素的适应性使得三向切分的快速排序成为排序库函数的最佳算法选择——需要将包含大量重复元素的数组排序的用例很常见。

经过精心调优的快速排序在绝大多数计算机上的绝大多数应用中都会比其他基于比较的排序算法更快。快速排序在今天的计算机业界中的广泛应用正是因为我们讨论过的数学模型说明了它在实际应用中比其他方法的性能更好，而近几十年的大量实验和经验也证明了这个结论。

在第 5 章中我们会发现，这些并不是快速排序发展的终点，因为有人研究出了完全不需要比较的排序算法！但快速排序的另一个版本在那个环境下仍然是最棒的，和这里一样。 301

答疑

问 有没有将数组平分的办法，而不是根据切分元素的最后位置来切分数组？

答 这个问题困扰了专家们十多年。这和用数组的中位数切分的想法类似。我们在 2.5.3.4 节中讨论了寻找中位数的问题。在线性时间内找到是可能的，但用现有的算法（基于快速排序的切分），这么做的代价远远超过将数组平分而节省的 39%。

问 随机地将数组打乱似乎占了排序用时的一大部分，这么做值得吗？

答 值得。这能够防止出现最坏情况并使运行时间可以预计。Hoare 在 1960 年提出这个算法的时候就推荐了这种方法——它是一种（也是第一批）偏爱随机性的算法。

问 为什么都将注意力放在重复元素上？

答 这个问题直接影响到实际应用中的性能。它曾被忽略了数十年，结果是一些老的实现对含有大量重复元素的数组排序时用时超过平方级别，这在实际应用中肯定出现过。像算法 2.5 等较好的实现对于这种数组的复杂度是线性对数级别的，但在很多情况下，如本节最后将其改进为信息量最佳的线性级别是很值得的。 302

练习

2.3.1 按照 `partition()` 方法的轨迹的格式给出该方法是如何切分数组 `E A S Y Q U E S T I O N` 的。

2.3.2 按照本节中快速排序所示轨迹的格式给出快速排序是如何将数组 `E A S Y Q U E S T I O N` 排序的（出于练习的目的，可以忽略开头打乱数组的部分）。

2.3.3 对于长度为 N 的数组，在 `Quick.sort()` 执行时，其最大的元素最多会被交换多少次？

2.3.4 假如跳过开头打乱数组的操作，给出六个含有 10 个元素的数组，使得 `Quick.sort()` 所需的比较次数达到最坏情况。

2.3.5 给出一段代码将已知只有两种主键值的数组排序。

2.3.6 编写一段代码来计算 C_N 的准确值，在 N=100、1000 和 10 000 的情况下比较准确值和估计值 $2N\ln N$ 的差距。

2.3.7 在使用快速排序将 N 个不重复的元素排序时，计算大小为 0、1 和 2 的子数组的数量。如果你喜欢数学，请推导；如果你不喜欢，请做一些实验并提出猜想。

2.3.8 `Quick.sort()` 在处理 N 个全部重复的元素时大约需要多少次比较？

2.3.9 请说明 `Quick.sort()` 在处理只有两种主键值的数组时的行为，以及在处理只有三种主键值的数组时的行为。

2.3.10 Chebyshev 不等式表明，一个随机变量的标准差距离均值大于 k 的概率小于 $1/k^2$。对于 N=100 万，用 Chebyshev 不等式计算快速排序所使用的比较次数大于 1000 亿次的概率（$0.1N^2$）。

2.3.11 假如在遇到和切分元素重复的元素时我们继续扫描数组而不是停下来，证明使用这种方法的快速排序在处理只有若干种元素值的数组时的运行时间是平方级别的。

303

2.3.12 按照代码所示轨迹的格式给出信息量最佳的快速排序第一次是如何切分数组 `B A B A B A B A C A D A B R A` 的。

2.3.13 在最佳、平均和最坏情况下，快速排序的递归深度分别是多少？这决定了系统为了追踪递归调用所需的栈的大小。在最坏情况下保证递归深度为数组大小的对数级的方法请见练习 2.3.20。

2.3.14 证明在用快速排序处理大小为 N 的不重复数组时，比较第 i 大和第 j 大元素的概率为 $2/(j-i)$，并用该结论证明命题 K。

304

提高题

2.3.15 *螺丝和螺帽。*(G. J. E. Rawlins) 假设有 N 个螺丝和 N 个螺帽混在一堆，你需要快速将它们配对。一个螺丝只会匹配一个螺帽，一个螺帽也只会匹配一个螺丝。你可以试着把一个螺丝和一个螺帽拧在一起看看谁大了，但不能直接比较两个螺丝或者两个螺帽。给出一个解决这个问题的有效方法。

2.3.16 *最佳情况* 编写一段程序来生成使算法 2.5 中的 `sort()` 方法表现最佳的数组（无重复元素）：数组大小为 N 且不包含重复元素，每次切分后两个子数组的大小最多差 1（子数组的大小与含有 N 个相同元素的数组的切分情况相同）。（对于这道练习，我们不需要在排序开始时打乱数组。）*以下练习描述了快速排序的几个变体。它们每个都需要分别实现，但你也很自然地希望使用 `SortCompare` 进行实验来评估每种改动的效果。*

2.3.17 *哨兵。*修改算法 2.5，去掉内循环 `while` 中的边界检查。由于切分元素本身就是一个哨兵（`v` 不可能小于 `a[lo]`），左侧边界的检查是多余的。要去掉另一个检查，可以在打乱数组后将数组的最大元素放在 `a[length-1]` 中。该元素永远不会移动（除非和相等的元素交换），可以在所有包含它的子数组中成为哨兵。*注意：在处理内部子数组时，右子数组中最左侧的元素可以作为左子数组右边界的哨兵。*

2.3.18 *三取样切分。*为快速排序实现正文所述的三取样切分（参见 2.3.3.2 节）。运行双倍测试来确认这项改动的效果。

2.3.19 *五取样切分。*实现一种基于随机抽取子数组中 5 个元素并取中位数进行切分的快速排序。将取样元素放在数组的一侧以保证只有中位数元素参与了切分。运行双倍测试来确定这项改动的效果，并和标准的快速排序以及三取样切分的快速排序（请见上一道练习）进行比较。*附加题：*找到一种对于任意输入都只需要少于 7 次比较的五取样算法。

305

2.3.20 *非递归的快速排序。*实现一个非递归的快速排序，使用一个循环来将弹出栈的子数组切分并将结果子数组重新压入栈。*注意：*先将较大的子数组压入栈，这样就可以保证栈最多只会有 $\lg N$ 个元素。

2.3.21 *将重复元素排序的比较次数的下界。*完成命题 M 的证明的第一部分。参考命题 I 的证明并注意当有 k 个主键值时所有元素存在 $N!/f_1!f_2!\cdots f_k!$ 种不同的排列，其中第 i 个主键值出现的频率为 f_i（即 Np_i，按照命题 M 的记法），且 $f_1+\cdots+f_k=N$。

2.3.22 *快速三向切分。*（J. Bently，D. McIlroy）用将重复元素放置于子数组两端的方式实现一个信息量最优的排序算法。使用两个索引 p 和 q，使得 a[lo..p-1] 和 a[q+1..hi] 的元素都和 a[lo] 相等。使用另外两个索引 i 和 j，使得 a[p..i-1] 小于 a[lo]，a[j+i..q] 大于 a[lo]。在内循环中加入代码，在 a[i] 和 v 相当时将其与 a[p] 交换（并将 p 加 1），在 a[j] 和 v 相等且 a[i] 和 a[j] 尚未和 v 进行比较之前将其与 a[q] 交换。

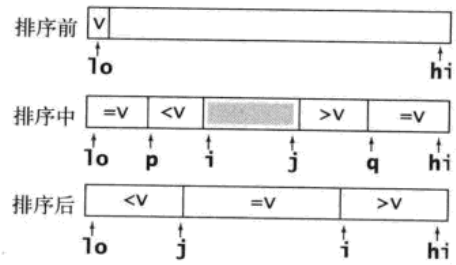

图 2.3.6 Bently-McIlroy 三向切分

添加在切分循环结束后将和 v 相等的元素交换到正确位置的代码，如图 2.3.6 所示。*请注意：*这里实现的代码和正文中给出的代码是等价的，因为这里额外的交换用于和切分元素相等的元素，而正文中的代码将额外的交换用于和切分元素不等的元素。

2.3.23 *Java 的排序库函数。*在练习 2.3.22 的代码中使用 Tukey's ninther 方法来找出切分元素——选择三组，每组三个元素，分别取三组元素的中位数，然后取三个中位数的中位数作为切分元素，且在排序小数组时切换到插入排序。

2.3.24 *取样排序。*（W. Frazer，A. McKellar）实现一个快速排序，取样大小为 2^k-1。首先将取样得到的元素排序，然后在递归函数中使用样品的中位数切分。分为两部分的其余样品元素无需再次排序并可以分别应用于原数组的两个子数组。这种算法被称为*取样排序*。 306

实验题

2.3.25 *切换到插入排序。*实现一个快速排序，在子数组元素少于 M 时切换到插入排序。用快速排序处理大小 N 分别为 10^3、10^4、10^5 和 10^6 的随机数组，根据经验给出使其在你的计算环境中运行速度最快的 M 值。将 M 从 0 变化到 30 的每个值所得到的平均运行时间绘成曲线。*注意：*你需要为算法 2.2 添加一个需要三个参数的 sort() 方法以使 Insertion.sort(a, lo, hi) 将子数组 a[lo..hi] 排序。

2.3.26 *子数组的大小。*编写一个程序，在快速排序处理大小为 N 的数组的过程中，当子数组的大小小于 M 时，排序方法需要切换为插入排序。将子数组的大小绘制成直方图。用 $N=10^5$，M=10、20 和 50 测试你的程序。

2.3.27 *忽略小数组。*用实验对比以下处理小数组的方法和练习 2.3.25 的处理方法的效果：在快速排序中直接忽略小数组，仅在快速排序结束后运行一次插入排序。*注意：*可以通过这些实验估计出电脑的缓存大小，因为当数组大小超出缓存时这种方法的性能可能会下降。

2.3.28 *递归深度。*用经验性的研究估计切换阈值为 M 的快速排序在将大小为 N 的不重复数组排序时的平均递归深度，其中 M=10、20 和 50，$N=10^3$、10^4、10^5 和 10^6。

2.3.29 *随机化。*用经验性的研究对比随机选择切分元素和正文所述的一开始就将数组随机化这两种策略的效果。在子数组大小为 M 时进行切换，将大小为 N 的不重复数组排序，其中 M=10、20 和 50，$N=10^3$、10^4、10^5 和 10^6。

2.3.30 *极端情况*。用初始随机化和非初始随机化的快速排序测试练习 2.1.35 和练习 2.1.36 中描述的大型非随机数组。在将这些大数组排序时，乱序对快速排序的性能有何影响？

2.3.31 *运行时间直方图*。编写一个程序，接受命令行参数 N 和 T，用快速排序对大小为 N 的随机浮点数数组进行 T 次排序，并将所有运行时间绘制成直方图。令 $N=10^3$、10^4、10^5 和 106，为了使曲线更平滑，T 值越大越好。这个练习最关键的地方在于找到适当的比例绘制出实验结果。

307