线性时间排序

到目前为止，我们已经介绍了几种能在$O(n\lg n)$时间内排序n个数的算法。归并排序和堆排序达到了最坏情况下的上界；快速排序在平均情况下达到该上界。而且，对于这些算法中的每一个，我们都能给出n个输入数值，使得该算法能在$\Omega(n\lg n)$时间内完成。

这些算法都有一个有趣的性质：*在排序的最终结果中，各元素的次序依赖于它们之间的比较*。我们把这类排序算法称为**比较排序**。到目前为止，我们介绍的所有排序算法都是比较排序。

8.1节将证明对包含n个元素的输入序列来说，任何比较排序在最坏情况下都要经过$\Omega(n\lg n)$次比较。因此，归并排序和堆排序是渐近最优的，并且任何已知的比较排序最多就是在常数因子上优于它们。

8.2节、8.3节和8.4节讨论三种线性时间复杂度的排序算法：计数排序、基数排序和桶排序。当然，这些算法是用运算而不是比较来确定排序顺序的。因此，下界$\Omega(n\lg n)$对它们是不适用的。

8.1 排序算法的下界

在一个比较排序算法中，我们只使用元素间的比较来获得输入序列$\langle a_1, a_2, \cdots, a_n\rangle$中的元素间次序的信息。也就是说，给定两个元素$a_i$和$a_j$，可以执行$a_i<a_j$、$a_i\leqslant a_j$、$a_i=a_j$、$a_i\geqslant a_j$或者$a_i>a_j$中的一个比较操作来确定它们之间的相对次序。我们不能用其他方法观察元素的值或者它们之间的次序信息。

不失一般性，在本节中，我们不妨假设所有的输入元素都是互异的。给定了这个假设后，$a_i=a_j$的比较就没有意义了。因此，我们可以假设不需要这种比较。同时，注意到$a_i\leqslant a_j$、$a_i\geqslant a_j$、$a_i>a_j$和$a_i<a_j$都是等价的，因为通过它们所得到的关于a_i和a_j的相对次序的信息是相同的。这样，又可以进一步假设所有比较采用的都是$a_i\leqslant a_j$形式。

决策树模型

比较排序可以被抽象为一棵决策树。**决策树**是一棵完全二叉树，它可以表示在给定输入规模情况下，某一特定排序算法对所有元素的比较操作。其中，控制、数据移动等其他操作都被忽略了。图8-1显示了2.1节中插入排序算法作用于包含三个元素的输入序列的决策树情况。

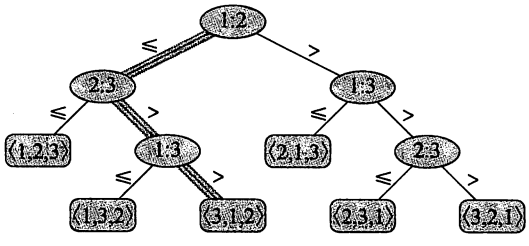

图8-1 作用于3个元素时的插入排序决策树。标记为$i:j$的内部结点表示a_i和a_j之间的比较。排列为$\langle\pi(1), \pi(2), \cdots, \pi(n)\rangle$的叶结点表示得到的顺序$a_{\pi(1)}\leqslant a_{\pi(2)}\leqslant\cdots\leqslant a_{\pi(n)}$。加了阴影的路径表示在对输入序列$\langle a_1=6, a_2=8, a_3=5\rangle$进行排序时所做的决策；叶结点上的排列$\langle 3, 1, 2\rangle$表示排序的结果是$a_3=5\leqslant a_1=6\leqslant a_2=8$。对于输入元素来说，共有$3!=6$种可能的排列，因此决策树至少包含6个叶结点

在决策树中，每个内部结点都以$i:j$标记，其中，i和j满足$1\leqslant i, j\leqslant n$，$n$是输入序列中的元素个数。每个叶结点上都标注一个序列$\langle\pi(1), \pi(2), \cdots, \pi(n)\rangle$（序列的相关背景知识参阅C.1节）。排序算法的执行对应于一条从树的根结点到叶结点的路径。每一个内部结点表示一次比较$a_i\leqslant a_j$。左子树表示一旦我们确定$a_i\leqslant a_j$之后的后续比较，右子树则表示在确定了$a_i>a_j$后的后续比较。当到达一个叶结点时，表示排序算法已

经确定了一个顺序 $a_{\pi(1)} \leqslant a_{\pi(2)} \leqslant \cdots \leqslant a_{\pi(n)}$。因为任何正确的排序算法都能够生成输入的每一个排列，所以对一个正确的比较排序算法来说，n 个元素的 $n!$ 种可能的排列都应该出现在决策树的
192 叶结点上。而且，每一个叶结点都必须是可以从根结点经由某条路径到达的，该路径对应于比较排序的一次实际执行过程(我们称这种叶结点为“可达的”)。因此，在后续内容中，我们将只考虑每一种排列都是一个可达的叶结点的决策树。

最坏情况的下界

在决策树中，从根结点到任意一个可达叶结点之间最长简单路径的长度，表示的是对应的排序算法中最坏情况下的比较次数。因此，一个比较排序算法中的最坏情况比较次数就等于其决策树的高度。同时，当决策树中每种排列都是以可达的叶结点的形式出现时，该决策树高度的下界也就是比较排序算法运行时间的下界。下面的定理给出这样的一个下界。

定理 8.1 *在最坏情况下，任何比较排序算法都需要做 $\Omega(n \lg n)$ 次比较。*

证明 根据前面的讨论，对于一棵每个排列都是一个可达的叶结点的决策树来说，树的高度完全可以被确定。考虑一棵高度为 h、具有 l 个可达叶结点的决策树，它对应一个对 n 个元素所做的比较排序。因为输入数据的 $n!$ 种可能的排列都是叶结点，所以有 $n! \leqslant l$。由于在一棵高为 h 的二叉树中，叶结点的数目不多于 2^h，我们得到：

$$n! \leqslant l \leqslant 2^h$$

对该式两边取对数，有

$$\begin{aligned} h &\geqslant \lg(n!) && \text{(因为 lg 函数是单调递增的)} \\ &= \Omega(n \lg n) && \text{(由公式(3.19))} \end{aligned}$$

■

推论 8.2 *堆排序和归并排序都是渐近最优的比较排序算法。*

证明 堆排序和归并排序的运行时间上界为 $O(n \lg n)$，这与定理 8.1 给出的最坏情况的下界 $\Omega(n \lg n)$ 是一致的。 ■

练习

193 **8.1-1** 在一棵比较排序算法的决策树中，一个叶结点可能的最小深度是多少？

8.1-2 不用斯特林近似公式，给出 $\lg(n!)$ 的渐近紧确界。利用 A.2 节中介绍的技术来求累加和 $\sum_{k=1}^{n} \lg k$。

8.1-3 证明：对 $n!$ 种长度为 n 的输入中的至少一半，不存在能达到线性运行时间的比较排序算法。如果只要求对 $1/n$ 的输入达到线性时间呢？$1/2^n$ 呢？

8.1-4 假设现有一个包含 n 个元素的待排序序列。该序列由 n/k 个子序列组成，每个子序列包含 k 个元素。一个给定子序列中的每个元素都小于其后继子序列中的所有元素，且大于其前驱子序列中的每个元素。因此，对于这个长度为 n 的序列的排序转化为对 n/k 个子序列中的 k 个元素的排序。试证明：这个排序问题中所需比较次数的下界是 $\Omega(n \lg k)$。(*提示*：简单地将每个子序列的下界进行合并是不严谨的。)

8.2 计数排序

计数排序假设 n 个输入元素中的每一个都是在 0 到 k 区间内的一个整数，其中 k 为某个整数。当 $k=O(n)$ 时，排序的运行时间为 $\Theta(n)$。

计数排序的基本思想是：对每一个输入元素 x，确定小于 x 的元素个数。利用这一信息，就可以直接把 x 放到它在输出数组中的位置上了。例如，如果有 17 个元素小于 x，则 x 就应该在第 18 个输出位置上。当有几个元素相同时，这一方案要略做修改。因为不能把它们放在同一个输出位置上。

在计数排序算法的代码中，假设输入是一个数组 $A[1..n]$，$A.length=n$。我们还需要两个数组：$B[1..n]$存放排序的输出，$C[0..k]$提供临时存储空间。 194

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]]=C[A[j]]+1
 6  //C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i]=C[i]+C[i−1]
 9  //C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]]=A[j]
12      C[A[j]]=C[A[j]]−1
```

图 8-2 图示了计数排序的运行过程。在第 2～3 行 **for** 循环的初始化操作之后，数组 C 的值全被置为 0；第 4～5 行的 **for** 循环遍历每一个输入元素。如果一个输入元素的值为 i，就将 $C[i]$值加 1。于是，在第 5 行执行完后，$C[i]$中保存的就是等于 i 的元素的个数，其中 $i=0, 1, \cdots, k$。第 7～8 行通过加总计算确定对每一个 $i=0, 1, \cdots, k$，有多少输入元素是小于或等于 i 的。

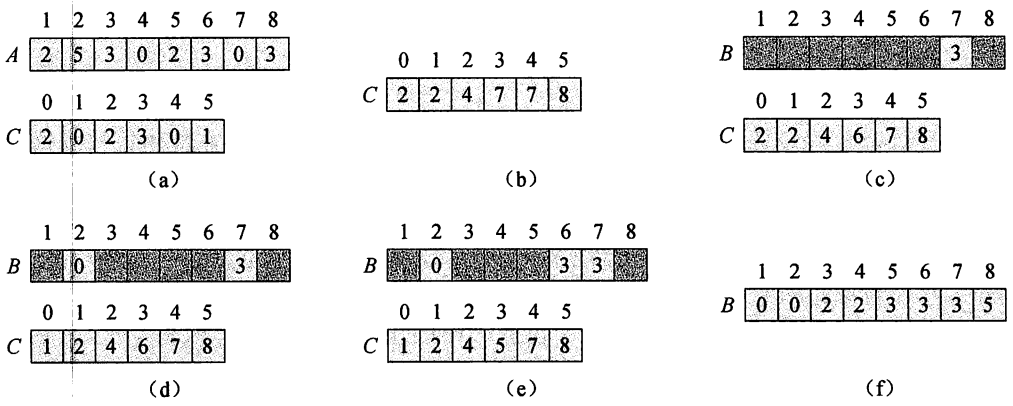

图 8-2　COUNTING-SORT 在输入数组 $A[1..8]$上的处理过程，其中 A 中的每一个元素都是不大于 $k=5$ 的非负整数。(a)第 5 行执行后的数组 A 和辅助数组 C 的情况。(b)第 8 行执行后，数组 C 的情况。(c)～(e)分别显示了第 10～12 行的循环体迭代了一次、两次和三次之后，输出数组 B 和辅助数组 C 的情况。其中，数组 B 中只有浅色阴影部分有元素值填充。(f)最终排好序的输出数组 B 195

最后，在第 10～12 行的 **for** 循环部分，把每个元素 $A[j]$放到它在输出数组 B 中的正确位置上。如果所有 n 个元素都是互异的，那么当第一次执行第 10 行时，对每一个 $A[j]$值来说，$C[A[j]]$就是 $A[j]$在输出数组中的最终正确位置。这是因为共有 $C[A[j]]$个元素小于或等于 $A[j]$。因为所有的元素可能并不都是互异的，所以，我们每将一个值 $A[j]$放入数组 B 中以后，都要将 $C[A[j]]$的值减 1。这样，当遇到下一个值等于 $A[j]$的输入元素(如果存在)时，该元素可以直接被放到输出数组中 $A[j]$的前一个位置上。

计数排序的时间代价是多少呢？第 2～3 行的 **for** 循环所花时间为 $\Theta(k)$，第 4～5 行的 **for** 循环所花时间为 $\Theta(n)$，第 7～8 行的 **for** 循环所花时间为 $\Theta(k)$，第 10～12 行的 **for** 循环所花时间为 $\Theta(n)$。这样，总的时间代价就是 $\Theta(k+n)$。在实际工作中，当 $k=O(n)$时，我们一般会采用计数排序，这时的运行时间为 $\Theta(n)$。

计数排序的下界优于我们在8.1节中所证明的$\Omega(n\lg n)$，因为它并不是一个比较排序算法。事实上，它的代码中完全没有输入元素之间的比较操作。相反，计数排序是使用输入元素的实际值来确定其在数组中的位置。当我们脱离了比较排序模型的时候，$\Omega(n\lg n)$这一下界就不再适用了。

计数排序的一个重要性质就是它是**稳定的**：具有相同值的元素在输出数组中的相对次序与它们在输入数组中的相对次序相同。也就是说，对两个相同的数来说，在输入数组中先出现的数，在输出数组中也位于前面。通常，这种稳定性只有当进行排序的数据还附带卫星数据时才比较重要。计数排序的稳定性很重要的另一个原因是：计数排序经常会被用作基数排序算法的一个子过程。我们将在下一节中看到，为了使基数排序正确运行，计数排序必须是稳定的。

练习

8.2-1 参照图8-2的方法，说明COUNTING-SORT在数组$A=\langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\rangle$上的操作过程。

8.2-2 试证明COUNTING-SORT是稳定的。

8.2-3 假设我们在COUNTING-SORT的第10行循环的开始部分，将代码改写为：

```
10  for  j = 1 to A.length
```

196 试证明该算法仍然是正确的。它还稳定吗？

8.2-4 设计一个算法，它能够对于任何给定的介于0到k之间的n个整数先进行预处理，然后在$O(1)$时间内回答输入的n个整数中有多少个落在区间$[a..b]$内。你设计的算法的预处理时间应为$\Theta(n+k)$。

8.3 基数排序

基数排序(radix sort)是一种用在卡片排序机上的算法，现在你只能在博物馆找到这种卡片排序机了。一张卡片有80列，在每一列上机器可以选择在12个位置中的任一处穿孔。通过机械操作，我们可以对排序机"编程"来检查每个卡片中的给定列，然后根据穿孔的位置将它们分别放入12个容器中。操作员就可以逐个容器地来收集卡片，其中第一个位置穿孔的卡片在最上面，其次是第二个位置穿孔的卡片，依此类推。

对十进制数字来说，每列只会用到10个位置(另两个位置用于编码非数值字符)。一个d位数将占用d列。因为卡片排序机一次只能查看一列，所以要对n张卡片上的d位数进行排序，就需要设计一个排序算法。

从直观上来看，你可能会觉得应该按最高有效位进行排序，然后对得到的每个容器递归地进行排序，最后再把所有结果合并起来。遗憾的是，为了排序一个容器中的卡片，10个容器中的9个都必须先放在一边。这一过程产生了许多要保存的临时卡片(见练习8.3-5)。

与人们直观感受相悖的是，基数排序是先按最低有效位进行排序来解决卡片排序问题的。然后算法将所有卡片合并成一叠，其中0号容器中的卡片都在1号容器中的卡片之前，而1号容器中的卡片又在2号容器中的卡片前面，依此类推。之后，用同样的方法按次低有效位对所有的卡片进行排序，并把排好的卡片再次合并成一叠。重复这一过程，直到对所有的d位数字都进行了排序。此时，所有卡片已按d位数完全排好序。所以，对这一叠卡片的排序仅需要进行d轮。图8-3说明了"一叠"7张3位数卡片的基数排序过程。

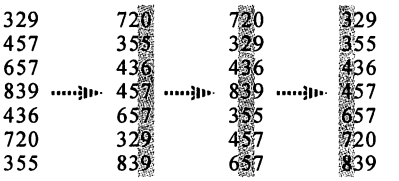

图8-3 一个由7个3位数组成的列表的基数排序过程。最左边的一列是输入数据，其余各列显示了由低位到高位连续进行排序后列表的情况。阴影指出了进行排序的位

为了确保基数排序的正确性，一位数排序算法必须是稳定的。卡片排序机所执行的排序是稳定

的，但操作员必须确保卡片从容器中被取出时不改变顺序，即使一个容器中的所有卡片在该位都是相同的数字也要确保这一点。

在一台典型的串行随机存取计算机上，我们有时会用基数排序来对具有多关键字域的记录进行排序。例如，我们希望用三个关键字(年、月和日)来对日期进行排序。对这个问题，我们可以使用基于特殊比较函数的排序算法：给定两个日期，先比较年，如果相同，再比较月，如果还是相同，就比较日。我们也可以采用另一种方法，用一种稳定排序算法对这些信息进行三次排序：先日，再月，最后是年。

基数排序的代码是非常直观的。在下面的代码中，我们假设 n 个 d 位的元素存放在数组 A 中，其中第 1 位是最低位，第 d 位是最高位。

```
RADIX-SORT(A, d)
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```

引理 8.3　*给定 n 个 d 位数，其中每一个数位有 k 个可能的取值。如果 RADIX-SORT 使用的稳定排序方法耗时 $\Theta(n+k)$，那么它就可以在 $\Theta(d(n+k))$ 时间内将这些数排好序。*

证明　基数排序的正确性可以通过对被排序的列进行归纳而加以证明(见练习 8.3-3)。对算法时间代价的分析依赖于所使用的稳定的排序算法。当每位数字都在 0 到 $k-1$ 区间内(这样它就有 k 个可能的取值)，且 k 的值不太大的时候，计数排序是一个好的选择。对 n 个 d 位数来说，每一轮排序耗时 $\Theta(n+k)$。共有 d 轮，因此基数排序的总时间为 $\Theta(d(n+k))$。 ■

当 d 为常数且 $k=O(n)$ 时，基数排序具有线性的时间代价。在更一般的情况中，我们可以灵活地决定如何将每个关键字分解成若干位。

引理 8.4　*给定一个 b 位数和任何正整数 $r \leqslant b$，如果 RADIX-SORT 使用的稳定排序算法对数据取值区间是 0 到 k 的输入进行排序耗时 $\Theta(n+k)$，那么它就可以在 $\Theta((b/r)(n+2^r))$ 时间内将这些数排好序。*

证明　对于一个值 $r \leqslant b$，每个关键字可以看做 $d=\lceil b/r \rceil$ 个 r 位数。每个数都是在 0 到 2^r-1 区间内的一个整数，这样就可以采用计数排序，其中 $k=2^r-1$。(例如，我们可以将一个 32 位的字看做是 4 个 8 位的数，于是有 $b=32$，$r=8$，$k=2^r-1=255$ 和 $d=b/r=4$)。每一轮排序花费时间为 $\Theta(n+k)=\Theta(n+2^r)$，计数排序花费的总时间代价为 $\Theta(d(n+2^r))=\Theta((b/r)(n+2^r))$。 ■

对于给定的 n 和 b，我们希望所选择的 $r(r \leqslant b)$ 值能够最小化表达式 $(b/r)(n+2^r)$。如果 $b < \lfloor \lg n \rfloor$，则对于任何满足 $r \leqslant b$ 的 r，都有 $(n+2^r)=\Theta(n)$。显然，选择 $r=b$ 得到的时间代价为 $(b/b)(n+2^b)=\Theta(n)$，这一结果是渐近意义上最优的。如果 $b \geqslant \lfloor \lg n \rfloor$，选择 $r=\lfloor \lg n \rfloor$ 可以得到偏差不超过常数系数范围内的最优时间代价。下面我们来详细说明这一点。选择 $r=\lfloor \lg n \rfloor$，得到的运行时间为 $\Theta(bn/\lg n)$。随着将 r 的值逐步增大到大于 $\lfloor \lg n \rfloor$ 后，分子中的 2^r 项比分母中的 r 项增加得快。因此，将 r 增大到大于 $\lfloor \lg n \rfloor$ 后，得到的时间代价为 $\Omega(bn/\lg n)$。反之，如果将 r 减小到 $\lfloor \lg n \rfloor$ 之下，则 b/r 项会变大，而 $n+2^r$ 项仍保持为 $\Theta(n)$。

基数排序是否比基于比较的排序算法(如快速排序)更好呢？通常情况，如果 $b=O(\lg n)$，而且我们选择 $r \approx \lg n$，则基数排序的运行时间为 $\Theta(n)$。这一结果看上去要比快速排序的期望运行时间代价 $\Theta(n \lg n)$ 更好一些。但是，在这两个表达式中，隐含在 Θ 符号背后的常数项因子是不同的。在处理的 n 个关键字时，尽管基数排序执行的循环轮数会比快速排序要少，但每一轮它所耗费的时间要长得多。哪一个排序算法更合适依赖于具体实现和底层硬件的特性(例如，快速排序通常可以比基数排序更有效地使用硬件的缓存)，以及输入数据的特征。此外，利用计数排序作为中间稳定排序的基数排序不是原址排序，而很多 $\Theta(n \lg n)$ 时间的比较排序是原址排序。因此，当主存的容量比较宝贵时，我们可能会更倾向于像快速排序这样的原址排序

算法。

练习

8.3-1 参照图 8-3 的方法，说明 RADIX-SORT 在下列英文单词上的操作过程：COW，DOG，SEA，RUG，ROW，MOB，BOX，TAB，BAR，EAR，TAR，DIG，BIG，TEA，NOW，FOX。

199

8.3-2 下面的排序算法中哪些是稳定的：插入排序、归并排序、堆排序和快速排序？给出一个能使任何排序算法都稳定的方法。你所给出的方法带来的额外时间和空间开销是多少？

8.3-3 利用归纳法来证明基数排序是正确的。在你所给出的证明中，在哪里需要假设所用的底层排序算法是稳定的？

8.3-4 说明如何在 $O(n)$时间内，对 0 到 n^3-1 区间内的 n 个整数进行排序。

***8.3-5** 在本节给出的第一个卡片排序算法中，为排序 d 位十进制数，在最坏情况下需要多少轮排序？在最坏情况下，操作员需要记录多少堆卡片？

8.4 桶排序

桶排序(bucket sort)假设输入数据服从均匀分布，平均情况下它的时间代价为 $O(n)$。与计数排序类似，因为对输入数据作了某种假设，桶排序的速度也很快。具体来说，计数排序假设输入数据都属于一个小区间内的整数，而桶排序则假设输入是由一个随机过程产生，该过程将元素均匀、独立地分布在[0，1)区间上(见 C.2 节中均匀分布的定义)。

桶排序将[0，1)区间划分为 n 个相同大小的子区间，或称为**桶**。然后，将 n 个输入数分别放到各个桶中。因为输入数据是均匀、独立地分布在[0，1)区间上，所以一般不会出现很多数落在同一个桶中的情况。为了得到输出结果，我们先对每个桶中的数进行排序，然后遍历每个桶，按照次序把各个桶中的元素列出来即可。

在桶排序的代码中，我们假设输入是一个包含 n 个元素的数组 A，且每个元素 $A[i]$满足 $0 \leqslant A[i] < 1$。此外，算法还需要一个临时数组 $B[0..n-1]$来存放链表(即桶)，并假设存在一种用于维护这些链表的机制(10.2 节将介绍如何实现链表的一些基本操作)。

200

```
BUCKET-SORT(A)
1  n = A.length
2  let B[0. . n−1] be a new array
3  for i = 0 to n−1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n−1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0],B[1],⋯,B[n−1] together in order
```

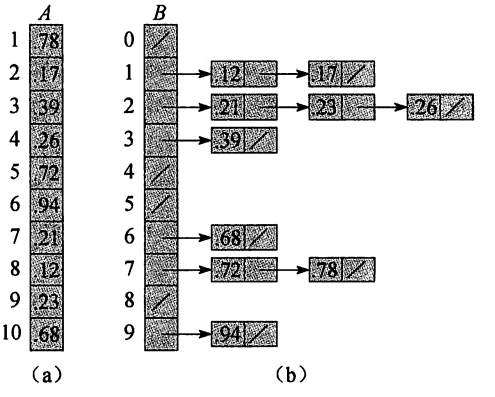

图 8-4 在 $n=10$ 时，BUCKET-SORT 的操作过程。(a)输入数组 $A[1..10]$。(b)在算法的第 8 行之后，$B[0..9]$中的已排序链表(桶)的情况。第 i 个桶中存放的是半开区间$[i/10, (i+1)/10]$中的值。排好序的输出是由链表 $B[0]$，$B[1]$，…，$B[9]$依次连接而成的

图 8-4 显示了在一个包含 10 个元素的输入数组上的桶排序过程。

为了验证算法的正确性，我们先来看看两个元素 $A[i]$和 $A[j]$。不失一般性，不妨假设 $A[i] \leqslant A[j]$。由于$\lfloor nA[i] \rfloor \leqslant \lfloor nA[j] \rfloor$，元素 $A[i]$或者与

$A[j]$被放入同一个桶中，或者被放入一个下标更小的桶中。如果 $A[i]$和 $A[j]$在同一个桶中，则第7～8行中的 **for** 循环会将它们按适当的顺序排列。如果 $A[i]$和 $A[j]$落入了不同的桶中，则第 9 行会将它们按适当的顺序排列。因此，桶排序算法是正确的。

现在来分析桶排序的运行时间。我们注意到，在最坏情况下，除第 8 行以外，所有其他各行时间代价都是 $O(n)$。我们还需要分析第 8 行中 n 次插入排序调用所花费的总时间。

现在来分析调用插入排序的时间代价。假设 n_i 是表示桶 $B[i]$中元素个数的随机变量，因为插入排序的时间代价是平方阶的(见 2.2 节)，所以桶排序的时间代价为：

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

我们现在来分析桶排序在平均情况下的运行时间。通过对输入数据取期望，我们可以计算出期望的运行时间。对上式两边取期望，并利用期望的线性性质，我们有：

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \qquad \text{(利用期望的线性性质)} \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) \qquad \text{(利用公式(C.22))} \end{aligned} \tag{8.1}$$

我们断言：

$$\mathrm{E}[n_i^2] = 2 - 1/n \tag{8.2}$$

对所有 $i=0, 1, \cdots, n-1$ 成立。这一点不足为奇：因为输入数组 A 的每一个元素是等概率地落入任意一个桶中，所以每一个桶 i 具有相同的期望值 $\mathrm{E}[n_i^2]$。为了证明公式(8.2)，我们定义指示器随机变量：对所有 $i=0, 1, \cdots, n-1$ 和 $j=1, 2, \cdots, n$，

$$X_{ij} = \mathrm{I}\{A[j] \text{ 落入桶 } i\}$$

因此，

$$n_i = \sum_{j=1}^{n} X_{ij}$$

为了计算 $\mathrm{E}[n_i^2]$，我们展开平方项，并重新组合各项：

$$\mathrm{E}[n_i^2] = \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] = \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] = \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\leqslant j\leqslant n}\ \sum_{\substack{1\leqslant k\leqslant n \\ k\neq j}} X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} \mathrm{E}[X_{ij}^2] + \sum_{1\leqslant j\leqslant n}\ \sum_{\substack{1\leqslant k\leqslant n \\ k\neq j}} \mathrm{E}[X_{ij}X_{ik}] \tag{8.3}$$

其中，最后一行是根据数学期望的线性性质得出的。我们分别计算这两项累加和，指示器随机变量 X_{ij} 为 1 的概率是 $1/n$，其他情况下是 0。于是有：

$$\mathrm{E}[X_{ij}^2] = 1^2 \cdot \frac{1}{n} + 0^2 \cdot \left(1 - \frac{1}{n}\right) = \frac{1}{n}$$

当 $k \neq j$ 时，随机变量 X_{ij} 和 X_{ik} 是独立的，因此有：

$$\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$

将这两个期望值带入公式(8.3)，我们得到：

$$\mathrm{E}[n_i^2] = \sum_{j=1}^{n} \frac{1}{n} + \sum_{1\leqslant j\leqslant n}\ \sum_{\substack{1\leqslant k\leqslant n \\ k\neq j}} \frac{1}{n^2} = n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n}$$

到此，公式(8.2)得证。

利用公式(8.1)中的期望值，我们可以得出结论：桶排序的期望运行时间为

$$\Theta(n)+n\cdot O(2-1/n)=\Theta(n)$$

即使输入数据不服从均匀分布，桶排序也仍然可以线性时间内完成。只要输入数据满足下列性质：所有桶的大小的平方和与总的元素数呈线性关系，那么通过公式(8.1)，我们就可以知道：桶排序仍然能在线性时间完成。

练习

8.4-1 参照图 8-4 的方法，说明 BUCKET-SORT 在数组 $A=\langle 0.79, 0.13, 0.16, 0.64, 0.39, 0.20, 0.89, 0.53, 0.71, 0.42\rangle$ 上的操作过程。

8.4-2 解释为什么桶排序在最坏情况下运行时间是 $\Theta(n^2)$？我们应该如何修改算法，使其在保持平均情况为线性时间代价的同时，最坏情况下时间代价为 $O(n\lg n)$？

8.4-3 设 X 是一个随机变量，用于表示在将一枚硬币抛掷两次时，正面朝上的次数。$E[X^2]$ 是多少呢？$E^2[X]$ 是多少呢？

***8.4-4** 在单位圆内给定 n 个点，$p_i=(x_i, y_i)$，对所有 $i=1, 2, \cdots, n$，有 $0<x_i^2+y_i^2\leqslant 1$。假设所有的点服从均匀分布，即在单位元的任一区域内找到给定点的概率与该区域的面积成正比。请设计一个在平均情况下有 $\Theta(n)$ 时间代价的算法，它能够按照点到原点之间的距离 $d_i=\sqrt{x_i^2+y_i^2}$ 对这 n 个点进行排序。(提示：在 BUCKET-SORT 中，设计适当的桶大小，用以反映各个点在单位圆中的均匀分布情况。)

***8.4-5** 定义随机变量 X 的**概率分布函数** $P(x)$ 为 $P(x)=\Pr\{X\leqslant x\}$。假设有 n 个随机变量 X_1，X_2，$\cdots$，X_n 服从一个连续概率分布函数 P，且它可以在 $O(1)$ 时间内被计算得到。设计一个算法，使其能够在平均情况下在线性时间内完成这些数的排序。

204

思考题

8-1 (比较排序的概率下界) 在这一问题中，我们将证明对于给定的 n 个互异的输入元素，任何确定或随机的比较排序算法，其概率运行时间都有下界 $\Omega(n\lg n)$。首先来分析一个确定的比较排序算法 A，其决策树为 T_A。假设 A 的输入的每一种排列情况都是等可能的。

a. 假设 T_A 的每个叶结点都标有在给定的随机输入情况下到达该结点的概率。证明：恰有 $n!$ 个叶结点标有 $1/n!$，其他的叶结点标记为 0。

b. 定义 $D(T)$ 表示一棵决策树 T 的外部路径长度，即 $D(T)$ 是 T 的所有叶结点深度的和。假设 T 为一棵有 $k>1$ 个叶结点的决策树，LT 和 RT 分别是 T 的左子树和右子树。证明：$D(T)=D(LT)+D(RT)+k$。

c. 定义 $d(k)$ 为所有具有 $k>1$ 个叶结点的决策树 T 的最小 $D(T)$ 值。证明：$d(k)=\min_{1\leqslant i\leqslant k-1}\{d(i)+d(k-i)+k\}$。(提示：考虑一棵能够取得该最小值的、有 k 个叶结点的决策树 T。设 i_0 是 LT 中的叶结点数，$k-i_0$ 是 RT 中的叶结点数。)

d. 证明：d 对于给定的 $k(k>1)$ 和 $i(1\leqslant i\leqslant k-1)$，函数 $i\lg i+(k-i)\lg(k-i)$ 在 $i=k/2$ 处取得最小值，并有结论 $d(k)=\Omega(k\lg k)$。

e. 证明：$D(T_A)=\Omega(n!\ \lg(n!))$，并得出在平均情况下，排序 n 个元素的时间代价为 $\Omega(n\lg n)$ 这一结论。

现在来考虑一个随机化的比较排序 B。通过引入两种结点，我们可以将决策树模型扩展来处理随机化的情况。这两种结点是：普通的比较结点和“随机化”结点。随机化结点刻画了算法 B 中所做的形如 RANDOM(1, r) 的随机选择情况。该类结点有 r 个子结点，在算法执行过程中，每一个子结点等概率地被选择。

f. 证明：对任何随机化比较排序算法 B，总存在一个确定的比较排序算法 A，其期望的比较次数不多于 B 的比较次数。 205

8-2 （线性时间原址排序）　假设有一个包含 n 个待排序数据记录的数组，且每条记录的关键字的值为 0 或 1。对这样一组记录进行排序的算法可能具备如下三种特性中的一部分：

1. 算法的时间代价是 $O(n)$。
2. 算法是稳定的。
3. 算法是原址排序，除了输入数组之外，算法只需要固定的额外存储空间。

a. 给出一个满足上述条件 1 和条件 2 的算法。

b. 给出一个满足上述条件 1 和条件 3 的算法。

c. 给出一个满足上述条件 2 和条件 3 的算法。

d. 你设计的算法(a)～(c)中的任一个是否可以用于 RADIX-SORT 的第 2 行作为基础排序方法，从而使 RADIX-SORT 在排序有 b 位关键字的 n 条记录时的时间代价是 $O(bn)$？如果可以，请解释应如何处理；如果不行，请说明原因。

e. 假设有 n 条记录，其中所有关键字的值都在 1 到 k 的区间内。你应该如何修改计数排序，使得它可以在 $O(n+k)$ 时间内完成对 n 条记录的原址排序。除输入数组外，你可以 $O(k)$ 使用大小的额外存储空间。你给出的算法是稳定的吗？（提示：当 $k=3$ 时，你应该如何做？）

8-3 （变长数据项的排序）

a. 给定一个整数数组，其中不同的整数所包含的数字的位数可能不同，但该数组中，所有整数中包含的总数字位数为 n。设计一个算法，使其可以在 $O(n)$ 时间内对该数组进行排序。

b. 给定一个字符串数组，其中不同的字符串所包含的字符数可能不同，但所有字符串中的总字符个数为 n。设计一个算法，使其可以在 $O(n)$ 时间内对该数组进行排序。（注意：此处的顺序是指标准的字典序，例如 $a<ab<b$。）

8-4 （水壶）　假设给了你 n 个红色的水壶和 n 个蓝色的水壶。它们的形状和尺寸都各不相同。所有的红色水壶中所盛的水都不一样多，蓝色水壶也是如此。而且，对于每一个红色水壶来说，都有一个对应的蓝色水壶，两者盛有一样多的水；反之亦然。 206

你的任务是找出所有的所盛水量一样多的红色水壶和蓝色水壶，并将它们配成一对。为此，可以执行如下操作：挑出一对水壶，其中一个是红色的，另一个是蓝色的，将红色水壶中倒满水，再将水倒入蓝色的水壶中。通过这一操作，可以判断出这个红色水壶是否比蓝色水壶盛的水更多，或者两者是一样多的。假设这样的比较需要花费一个单位时间。你的目标是找出一个算法，它能够用最少的比较次数来确定所有水壶的配对。注意，你不能直接比较两个红色或两个蓝色的水壶。

a. 设计一个确定性算法，它能够用 $\Theta(n^2)$ 次比较来完成所有水壶的配对。

b. 证明：解决该问题算法的比较次数下界为 $\Omega(n\lg n)$。

c. 设计一个随机算法，其期望的比较次数为 $O(n\lg n)$，并证明这个界是正确的。对你的算法来说，最坏情况下的比较次数是多少？

8-5 （平均排序）　假设我们不是要完全排序一个数组，而只是要求数组中的元素在平均情况下是升序的。更准确地说，如果对所有的 $i=1,2,\cdots,n-k$ 有下式成立，我们就称一个包含 n 个元素的数组 A 为 **k 排序的**（k-sorted）：

$$\frac{\sum_{j=i}^{i+k-1}A[j]}{k}\leqslant\frac{\sum_{j=i+1}^{i+k}A[j]}{k}$$

a. 一个数组是 1 排序的，表示什么含义？

b. 给出对数字 1，2，…，10 的一个排列，它是 2 排序的，但不是完全有序的。

c. 证明：一个包含 n 个元素的数组是 k 排序的，当且仅当对所有的 $i=1, 2, \cdots, n-k$，有 $A[i] \leqslant A[i+k]$。

d. 设计一个算法，它能在 $O(n\lg(n/k))$时间内对一个包含 n 个元素的数组进行 k 排序。当 k 是一个常数时，也可以给出 k 排序算法的下界。

e. 证明：我们可以在 $O(n\lg k)$时间内对一个长度为 n 的 k 排序数组进行全排序。（提示：可以利用练习 6.5-9 的结果。）

f. 证明：当 k 是一个常数时，对包含 n 个元素的数组进行 k 排序需要 $\Omega(n\lg n)$的时间。（提示：可以利用前面解决比较排序的下界的方法。）

207

8-6 （*合并有序列表的下界*） 合并两个有序列表是我们经常会遇到的问题。作为 MERGE-SORT 的一个子过程，我们在 2.3.1 节中已经遇到过这一问题。对这一问题，我们将证明在最坏情况下，合并两个都包含 n 个元素的有序列表所需的比较次数的下界是 $2n-1$。

首先，利用决策树来说明比较次数有一个下界 $2n-o(n)$。

a. 给定 $2n$ 个数，请算出共有多少种可能的方式将它们划分成两个有序的列表，其中每个列表都包含 n 个数。

b. 利用决策树和(a)的答案，证明：任何能够正确合并两个有序列表的算法都至少要进行 $2n-o(n)$次比较。

现在我们来给出一个更紧确界 $2n-1$。

c. 请说明：如果两个元素在有序序列中是连续的，且它们分别来自不同的列表，则它们必须进行比较。

d. 利用你对上一部分的回答，说明合并两个有序列表时的比较次数下界为 $2n-1$。

8-7 （*0-1 排序引理和列排序*） 针对两个数组元素 $A[i]$和 $A[j]$($i<j$)的**比较交换**操作的形式如下：

```
COMPARE-EXCHANGE(A, i, j)
1  if A[i] > A[j]
2      exchange A[i] with A[j]
```

经过比较交换操作之后，我们得到 $A[i] \leqslant A[j]$。

遗忘比较交换算法是指算法只按照事先定义好的操作执行，即需要比较的位置下标必须事先确定好。虽然算法可能依靠待排序元素个数，但它不能依赖待排序元素的值，也不能依赖任何之前的比较交换操作的结果。例如，下面是一个基于遗忘比较交换算法的插入排序：

```
INSERTION-SORT (A)
1  for j = 2 to A.length
2      for i = j - 1 downto 1
3          COMPARE-EXCHANGE(A, i, i + 1)
```

208

0-1 排序引理提供了有力的方法来证明一个遗忘比较交换算法可以产生正确的排序结果。该引理表明，如果一个遗忘比较交换算法能够对所有只包含 0 和 1 的输入序列排序，那么它也可以对包含任意值的输入序列排序。

你可以用反例来证明 0-1 排序引理：如果一个遗忘比较交换算法不能对一个包含任意值的序列进行排序，那么它也不能对某个 0-1 序列进行排序。不妨假设一个遗忘比较交换算法 X 未能对数组 $A[1..n]$排序。设 $A[p]$是算法 X 未能将其放到正确位置的最小的元素，而$A[q]$是被算法 X 放在 $A[p]$原本应该在的位置上的元素。定义一个只包含 0 和 1 的数组 $B[1..n]$如下：

$$B[i]=\begin{cases}0 & \text{若 } A[i]\leqslant A[p]\\ 1 & \text{若 } A[i]>A[p]\end{cases}$$

a. 讨论：当 $A[q]>A[p]$ 时，有 $B[p]=0$ 和 $B[q]=1$。

b. 为了完成 0-1 排序引理的证明，请先证明算法 X 不能对数组 B 正确地排序。

现在，需要用 0-1 排序引理来证明一个特别的排序算法的正确性。**列排序**算法是用于包含 n 个元素的矩形数组的排序。这一矩形数组有 r 行 s 列(因此 $n=rs$)，满足下列三个限制条件：

- r 必须是偶数
- s 必须是 r 的因子
- $r\geqslant 2s^2$

当列排序完成时，矩形数组是**列优先有序**的：按照列从上到下，从左到右，都是单调递增的。

如果不包括 n 的值的计算，列排序需要 8 步操作。所有奇数步都一样：对每一列单独进行排序。每一个偶数步是一个特定的排列。具体如下：

1. 对每一列进行排序。
2. 转置这个矩形数组，并重新规整化为 r 行 s 列的形式。也就是说，首先将最左边的一列放在前 r/s 行，然后将下一列放在第二个 r/s 行，依此类推。
3. 对每一列进行排序。
4. 执行第 2 步排列操作的逆操作。
5. 对每一列进行排序。
6. 将每一列的上半部分移到同一列的下半部分位置，将每一列的下半部分移到下一列的上半部分，并将最左边一列的上半部分置为空。此时，最后一列的下半部分成为新的最右列的上半部分，新的最右列的下半部分为空。
7. 对每一列进行排序。
8. 执行第 6 步排列操作的逆操作。

209

图 8-5 显示了一个在 $r=6$ 和 $s=3$ 时的列排序步骤(即使这个例子违背了 $r\geqslant 2s^2$ 的条件，列排序仍然有效)。

10	14	5	4	1	2	4	8	10	1	3	6	1	4	11
8	7	17	8	3	5	12	16	18	2	5	7	3	8	14
12	1	6	10	7	6	1	3	7	4	8	10	6	10	17
16	9	11	12	9	11	9	14	15	9	13	15	2	9	12
4	15	2	16	14	13	2	5	6	11	14	17	5	13	16
18	3	13	18	15	17	11	13	17	12	16	18	7	15	18
(a)			(b)			(c)			(d)			(e)		

1	4	11	5	10	16	4	10	16	1	7	13
2	8	12	6	13	17	5	11	17	2	8	14
3	9	14	7	15	18	6	12	18	3	9	15
5	10	16	1	4	11	1	7	13	4	10	16
6	13	17	2	8	12	2	8	14	5	11	17
7	15	18	3	9	14	3	9	15	6	12	18
(f)			(g)			(h)			(i)		

图 8-5　列排序的步骤。(a)6 行 3 列的输入数组。(b)第 1 步排序操作之后的情况。(c)第 2 步转置和规整化后的情况。(d)第 3 步排序操作之后的情况。(e)执行完第 4 步的情况，即反转第 2 步排列操作。(f)第5 步排序操作之后的情况。(g)第 6 步移动后的情况。(h)第 7 步排序操作之后的情况。(i)执行完第 8 步的情况，即反转第 6 步排列操作。现在数组已经是列优先有序了

c. 讨论：即使不知道奇数步采用了什么排序算法，我们也可以把列排序看做一种遗忘比较交换算法。

虽然似乎很难让人相信列排序也能实现排序，但是你可以利用 0-1 排序引理来证明这一点。因为列排序可以看做是一种遗忘比较交换算法，所以我们可以使用 0-1 排序引理。

[210] 下面一些定义有助于你使用这一引理。如果数组中的某个区域只包含全 0 或者全 1，我们定义这个区域是**干净的**。否则，如果这个区域包含的是 0 和 1 的混合，则称这个区域是**脏的**。这里，假设输入数据只包含 0 和 1，且输入数据能够被转换为 r 行 s 列。

d. 证明：经过第 1～3 步，数组由三部分组成：顶部一些由全 0 组成的干净行，底部一些由全 1 组成的干净行，以及中间最多 s 行脏的行。

e. 证明：经过第 4 步之后，如果按照列优先原则读取数组，先读到的是全 0 的干净区域，最后是全 1 的干净区域，中间是由最多 s^2 个元素组成的脏的区域。

f. 证明：第 5～8 步产生一个全排序的 0-1 输出，并得到结论：列排序可以正确地对任意输入值排序。

g. 现在假设 s 不能被 r 整除。证明：经过第 1～3 步，数组的顶部有一些全 0 的干净行，底部有一些全 1 的干净行，中间是最多 $2s-1$ 行脏行。那么与 s 相比，在 s 不能被 r 整除时，r 至少要有多大才能保证列排序的正确性？

h. 对第 1 步做一个简单修改，使得我们可以在 s 不能被 r 整除时，也保证 $r \geqslant 2s^2$，并且证明在这一修改后，列排序仍然正确。

本章注记

用于研究比较排序的决策树模型首先是由 Ford 和 Johnson[110]提出的。Knuth[211]有关排序的综述中涉及了排序问题的很多变形，包括本章所给出的排序问题复杂度的信息论下界。Ben-Or[39]利用泛化的决策树模型对排序的下界进行了全面分析。

根据 Knuth 所述，计数排序是由 H. H. Seward 于 1954 年提出的，而且他还提出了将计数排序与基数排序结合起来的思想。基数排序是从最低有效位开始的，这是一种机械式卡片排序机的操作员们所广泛采用的通用算法。根据 Knuth 所述，L. J. Comrie 于 1929 年首次在一篇描述卡片穿孔机文档中介绍了这一方法。自从 1956 年，桶排序就已经开始被使用了。当时这一基本思想是由 E. J. Isaac 和 R. C. Singleton[188]提出的。

Munro 和 Raman[263]给出一个稳定的排序算法，它在最坏情况下需要执行 $O(n^{1+\varepsilon})$ 次比较，

[211] 其中 $0<\varepsilon\leqslant 1$ 是任意的固定常数。尽管任一 $O(n \lg n)$ 时间算法所需比较次数更少，但 Munro 和 Raman 的算法仅需要将数据移动 $O(n)$ 次，而且它是原址排序。

许多研究人员都对如何在 $O(n \lg n)$ 时间内对 n 个 b 位整数进行排序做过研究，并已经获得了一些有益的成果，其中每一项成果都对计算模型做了略有不同的假设，对算法的限制也稍有差异。所有这些成果都假设计算机内存被划分成可寻址的 b 位字。Fredman 和 Willard[115]引入融合树(fusion tree)这一数据结构，它可以在 $O(n \lg n/\lg\lg n)$ 时间内对 n 个整数进行排序。Andersson[16]将这一界改善为 $O(n\sqrt{\lg n})$。这些算法要用到乘法和几个预先计算好的常量。Andersson、Hagerup、Nilsson 和 Raman[17]给出了一种不用乘法可以在 $O(n\lg\lg n)$ 时间内对 n 个整数进行排序的算法。但是，该算法所需要的存储空间以 n 来表示的话，可能是无界的。利用乘法散列技术，我们可以将所需的存储空间降至 $O(n)$，最坏情况运行时间的界 $O(n\lg\lg n)$ 成为期望运行时间的界。通过一般化 Andersson[16]提出的指数搜索树，Thorup[335]给出一个 $O(n(\lg\lg n)^2)$ 时间的排序算法，该算法不使用乘法和随机化，并且只需要线性存储空间。Han[158]把这些技术与一些新的想法结合起来，将排序算法的界改善至 $O(n\lg\lg n\lg\lg\lg n)$ 时间。尽管上述算法有着重要的理论突破，但都太复杂。就目前的情况来看，它们不太可能在实践中与现有的排序算法竞争。

[212] 思考题 8-7 中的列排序算法是由 Leighton[227]提出的。