并时会留下一个未归并的分段。

当 numberOfSegments 为 1 时，结束递归的 merge 方法。在这种情况下，f1 中包含已排好序的数据。文件 f1 被重命名为 targetfile（第 45 行）。

23.8.4 外部排序复杂度

在外部排序中，主要开销是在 I/O 上。假设 n 是文件中要排序的元素个数。在阶段 I，从原始文件中读取元素个数 n，然后将它输出给一个临时文件。因此，阶段 I 的 I/O 复杂度为 $O(n)$。

对于阶段 II，在第一个合并步骤之前，排好序的分段的个数为 $\frac{n}{c}$，其中 c 是 MAX_ARRAY_SIZE。每一个合并步骤都会使分段的个数减半。因此，在第一次合并步骤之后，分段个数为 $\frac{n}{2c}$。在第二次合并步骤之后，分段个数为 $\frac{n}{2^2c}$。在第三次合并步骤之后，分段个数为 $\frac{n}{2^3c}$。在第 $\log\left(\frac{n}{c}\right)$ 次合并步骤之后，分段个数减到 1。因此，合并步骤的总数为 $\log\left(\frac{n}{c}\right)$。

在每次合并步骤中，从文件 f1 读取一半数量的分段，然后将它们写入一个临时文件 f2。合并 f1 中剩余的分段和 f2 中的分段。每一个合并步骤中 I/O 的次数为 $O(n)$。因为合并步骤的总数是 $\log\left(\frac{n}{c}\right)$，I/O 的总数是

$$O(n)\times\log\left(\frac{n}{c}\right)=O(n\log n)$$

因此，外部排序的复杂度是 $O(n\log n)$。

复习题

23.23 描述外部排序是如何工作的。外部排序算法的复杂度是多少？

23.24 10 个数字 {2,3,4,0,5,6,7,9,8,1} 保存在外部文件 largedata.dat 中。设 MAX_ARRAY_SIZE 为 2，手工跟踪 SortLargeFile 程序。

关键术语

bubble sort（冒泡排序）
bucket sort（桶排序）
complete binary tree（完全二叉树）
external sort（外部排序）
heap（堆）
heap sort（堆排序）
height of a heap（堆的高度）
merge sort（归并排序）
quick sort（快速排序）
radix sort（基数排序）

本章小结

1. 选择排序、插入排序、冒泡排序和快速排序的最差时间复杂度为 $O(n^2)$。
2. 归并排序的平均情况和最差情况的复杂度为 $O(n\log n)$。快速排序的平均时间也是 $O(n\log n)$。
3. 对于设计排序这样的高效算法，堆是一个很有用的数据结构。本章介绍了如何定义和实现一个堆类，以及如何向 / 从堆中插入和删除元素。
4. 堆排序的时间复杂度为 $O(n\log n)$。
5. 桶排序和基数排序都是针对整数键值的特定排序算法。这些算法不是通过比较键值而是使用桶来对键值排序的，它们会比一般的排序算法效率更高。

6. 可以使用归并排序的一种变体——称为外部排序——对外部文件中的大型数据进行排序。

测试题

回答位于网址 www.cs.armstrong.edu/liang/intro10e/test.html 的本章测试题。

编程练习题

23.3 ～ 23.5 节

23.1 （泛型冒泡排序）使用冒泡排序编写下面两个泛型方法。第一个方法使用 Comparable 接口对元素排序，第二个方法使用 Comparator 接口对元素排序。

```
public static <E extends Comparable<E>>
  void bubbleSort(E[] list)
public static <E> void bubbleSort(E[] list,
  Comparator<? super E> comparator)
```

23.2 （泛型归并排序）使用归并排序编写下面两个泛型方法。第一个方法使用 Comparable 接口对元素排序，第二个方法使用 Comparator 接口对元素排序。

```
public static <E extends Comparable<E>>
  void mergeSort(E[] list)
public static <E> void mergeSort(E[] list,
  Comparator<? super E> comparator)
```

23.3 （泛型快速排序）使用快速排序编写下面两个泛型方法。第一个方法使用 Comparable 接口对元素排序，第二个方法使用 Comparator 接口对元素排序。

```
public static <E extends Comparable<E>>
  void quickSort(E[] list)
public static <E> void quickSort(E[] list,
  Comparator<? super E> comparator)
```

23.4 （改进快速排序）本书提供的快速排序算法选择线性表中的第一个元素作为主元。修改该算法，在线性表中的第一个元素、中间元素和最后一个元素中选择一个中位数作为主元。

*23.5 （泛型堆排序）使用堆排序编写下面两个泛型方法。第一个方法使用 Comparable 接口对元素排序，第二个方法使用 Comparator 接口对元素排序。

```
public static <E extends Comparable<E>>
  void heapSort(E[] list)
public static <E> void heapSort(E[] list,
  Comparator<? super E> comparator)
```

23.6 （检查顺序）编写下面的重载方法，用于检查数组是按升序还是降序排列的。默认情况下，该方法是检查升序的。为检查降序，则将 false 传递给方法中的升序参数。

```
public static boolean ordered(int[] list)
public static boolean ordered(int[] list, boolean ascending)
public static boolean ordered(double[] list)
public static boolean ordered
  (double[] list, boolean ascending)
public static <E extends Comparable<E>>
  boolean ordered(E[] list)
public static <E extends Comparable<E>> boolean ordered
  (E[] list, boolean ascending)
public static <E> boolean ordered(E[] list,
  Comparator<? super E> comparator)
public static <E> boolean ordered(E[] list,
  Comparator<? super E> comparator, boolean ascending)
```

23.6 节

23.7 （最小堆）本书中介绍的堆也称为最大堆（max-heap），其中的每个结点都大于或等于它的任何一

个子结点。最小堆（min-heap）是指每个结点都小于或等于它的任何一个子结点的堆。修改程序清单 23-9 中的 Heap 类以实现最小堆。

*23.8 （使用堆排序）使用堆实现下面的 sort 方法。

```
public static <E extends Comparable<E>> void sort(E[] list)
```

*23.9 （使用 Comparator 的泛型堆）修改程序清单 23-9 中的 Heap，使用泛型参数和一个 Comparator 来比较对象。定义一个新的构造方法，以 Comparator 作为它的参数，如下所示：

```
Heap(Comparator<? super E> comparator)
```

**23.10 （堆的可视化）编写一个程序，图形化显示一个堆，如图 23-10 所示。该程序允许用户向堆中插入和从堆中删除元素。

23.11 （堆的 clone 和 equals 方法）实现 Heap 类中的 clone 和 equals 方法。

23.7 节

*23.12 （基数排序）编写程序，随机创建 1 000 000 个整数，然后使用基数排序对它们排序。

*23.13 （排序的执行时间）编写程序，获取输入规模为 50 000、100 000、150 000、200 000、250 000 和 300 000 时的选择排序、冒泡排序、归并排序、快速排序、堆排序以及基数排序的执行时间。该程序应随机地创建数据，然后打印如下所示的一个表格：

数组大小	选择排序	冒泡排序	归并排序	快速排序	堆排序	基数排序
50 000						
100 000						
150 000						
200 000						
250 000						
300 000						

（提示：可以使用下面的代码模板来获取执行时间。）

```
long startTime = System.currentTimeMillis();
perform the task;
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;
```

本书给出了一个递归的快速排序，在此编写一个非递归版本。

23.8 节

*23.14 （外部排序的执行时间）编写程序，获取输入规模为 5 000 000、10 000 000、15 00 0000、20 000 000、25 000 000 和 30 000 000 时外部排序的执行时间。该程序应该打印出如下所示的一个表格：

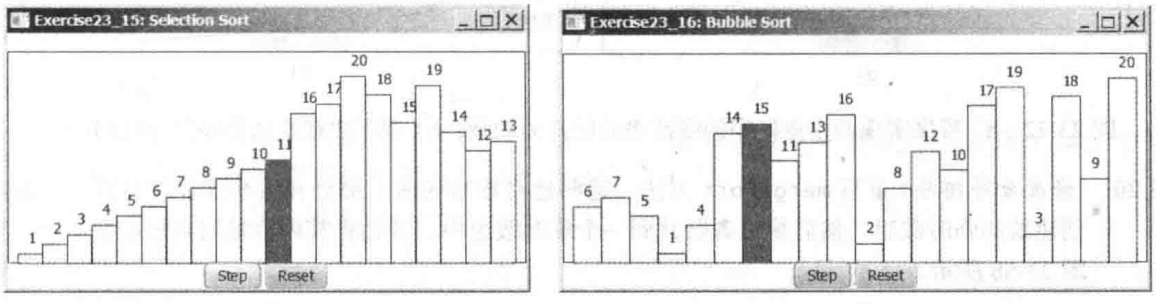
文件尺寸	5 000 000	10 000 000	15 000 000	20 000 000	25 000 000	30 000 000
时间						

综合

*23.15 （选择排序动画）编写一个程序，实现选择排序算法的动画。创建一个数组，以随机顺序包含从 1 到 20 的 20 个不同数字。数组元素在一个直方图中显示，如图 23-20a 所示。单击 Step 按钮使程序执行算法中外部循环的一次迭代，然后为新的数组重画直方图。将排好序的子数组标上颜色。当算法结束时，显示一条信息通知用户。单击 Reset 按钮为一次新的开始创建一个新的随机数组。（可以很容易地修改程序，来制作插入排序算法的动画。）

*23.16 （冒泡排序动画）编写一个程序，实现冒泡排序算法的动画。创建一个数组，以随机顺序包含从

1 到 20 的 20 个不同数字。数组元素在一个直方图中显示，如图 23-20b 所示。单击 Step 按钮使程序执行算法中的一次比较，然后为新的数组重画直方图。将表示考虑交换的数值条标上颜色。当算法结束时，显示一条信息通知用户。单击 Reset 按钮为一次新的开始创建一个新的随机数组。



a)　　b)

图 23-20　a) 程序实现选择排序的动画；b) 程序实现冒泡排序的动画

*23.17 （基数排序动画）编写一个程序，实现基数排序算法的动画。创建一个数组，以随机顺序包含从 1 到 1000 的 20 个不同数字。数组元素在一个直方图中显示，如图 23-21 所示。单击 Step 按钮使程序放置一个数字在一个桶中。刚放入的数字以红色显示。一旦所有的数字都放在桶中后，单击 Step 按钮从桶中收集所有的数字，将它们移回到数组中。当算法结束时，单击 Step 按钮显示一条信息通知用户。单击 Reset 按钮为一次新的开始创建一个新的随机数组。

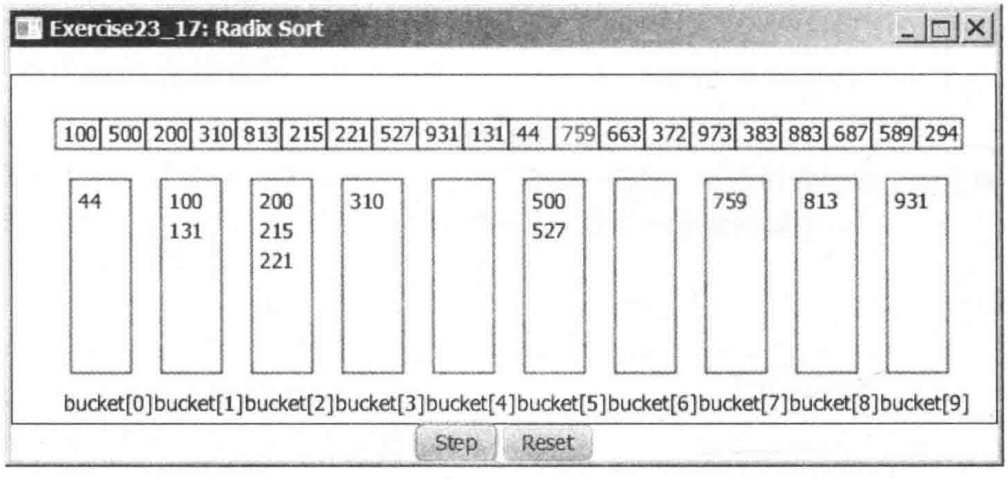
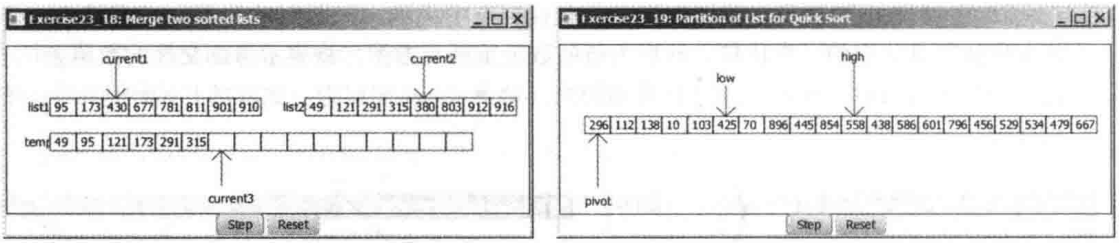


图 23-21　程序实现基数排序的动画

*23.18 （归并排序动画）编写一个程序，实现两个排好序的线性表的归并的动画。创建两个数组，`list1` 和 `list2`，每个包含从 1 到 999 的 8 个随机数字。数组元素如图 23-22a 所示。单击 Step 按钮使程序将 `list1` 或者 `list2` 中的一个元素移到 `temp` 中。单击 Reset 按钮为一个新的开始创建两个新的随机数组。当算法结束时，单击 Step 按钮显示一条信息通知用户。

*23.19 （快速排序分区动画）编写一个程序，实现快速排序的分区动画。程序创建一个包含从 1 到 999 的 20 个随机数字的线性表。线性表如图 23-22b 所示。单击 Step 按钮使程序将 `low` 移动到右边，或者 `high` 移动到左边，或者交换 `low` 和 `high` 位置的元素。单击 Reset 按钮为一个新的开始创建两个新的随机数组。当算法结束时，单击 Step 按钮显示一条信息通知用户。



a) b)

图 23-22 a）程序实现两个排好序的线性表的归并的动画；b）程序实现快速排序的分区的动画

*23.20 （修改合并排序）重写 `mergeSort` 方法，递归地对数组的前半部分和后半部分进行排序，而不创建新的临时数组。然后将二者归并到一个临时数组中，并且将其内容复制到原始数组中，如图 23-6b 所示。