```
44              max = list[i];
45            return new Integer(max);
46          }
47          else {
48            int mid = (low + high) / 2;
49            RecursiveTask<Integer> left = new MaxTask(list, low, mid);
50            RecursiveTask<Integer> right = new MaxTask(list, mid, high);
51
52            right.fork();
53            left.fork();
54            return new Integer(Math.max(left.join().intValue(),
55              right.join().intValue()));
56          }
57        }
58      }
59    }
```

```
The maximal number is 8999999
The number of processors is 2
Time is 44 milliseconds
```

由于该算法返回一个整数，我们通过继承 RecursiveTask<Integer> 为分解合并操作定义一个任务类（第 26 ～ 58 行）。重写 compute 方法返回 list[low..high] 中的最大元素（第 39 ～ 57 行）。如果线性表较小，采用顺序方式解决更加高效（第 40 ～ 46 行）。对于一个大的线性表，将其分为两半（第 48 ～ 50 行），任务 left 和 right 分别找到左半边和右半边的最大元素。在任务上调用 fork() 将使得任务被执行（第 52 和 53 行）。join() 方法等待任务执行完，然后返回结果（第 54 和 55 行）。

复习题

30.37 如何定义一个 ForkJoinTask? RecursiveAction 和 RecursiveTask 的区别是什么?

30.38 如何告诉系统来执行一个任务?

30.39 可以使用什么方法来测试一个任务是否已经完成?

30.40 如何创建一个 ForkJoinPool？ 如何将一个任务放到一个 ForkJoinPool 中?

关键术语

condition（条件）
deadlock（死锁）
fail-fast（快速失效）
fairness policy（公平策略）
Fork/Join Framework（Fork/Join 框架）
lock（锁）
monitor（监视器）
multithreading（多线程）
race condition（竞争状态）
semaphore（信号量）
synchronization wrapper（同步包装类）
synchronized block（同步块）
thread（线程）
thread-safe（线程安全）

本章小结

1. 每个任务都是 Runnable 接口的实例。线程就是一个便于任务执行的对象。可以通过实现 Runnable 接口来定义任务类，通过使用 Thread 构造方法包住一个任务来创建线程。
2. 一个线程对象被创建之后，可以使用 start() 方法启动线程，可以使用 sleep(long) 方法将线程转入休眠状态，以便其他线程获得运行的机会。
3. 线程对象从来不会直接调用 run 方法。到了执行某个线程的时候，Java 虚拟机调用 run 方法。类必须覆盖 run 方法，告诉系统线程运行时将会做什么。

4. 为了避免线程破坏共享资源，可以使用同步的方法或块。同步方法在执行前需要获得一个锁。当同步方法是实例方法时，锁是在调用方法的对象上；当同步方法是静态（类）方法时，锁是在方法所在的类上。
5. 在执行方法中某个代码块时，可以使用同步语句获得任何对象上的锁，而不仅是 this 对象上的锁。这个代码块称为同步块。
6. 可以使用显式锁和条件，以及对象的内置监视器来便于进程之间的通信。
7. Java 合集框架提供的阻塞队列（ArrayBlockingQueue，LinkedBlockingQueue,PriorityBlockingQueue）自动地同步对队列的访问。
8. 可以使用信号量来限制访问共享资源的并行任务数量。
9. 如果两个或更多的线程获取多个对象上的锁时，每个线程都有一个对象上的锁并等待另一个对象上的锁，这时就有可能发生死锁现象。使用资源排序技术可以避免死锁。
10. JDK7 的 Fork/Join 框架被设计用于开发并行程序。可以定义一个继承自 RecursiveAction 或者 RecursiveTask 的任务类，在 ForkJoinPool 中并行执行任务类，并在所有任务执行完后得到整体的解答。

测试题

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

编程练习题

30.1 ～ 30.5 节

*30.1 （修改程序清单 30-1）改写程序清单 30-1，在文本域中显示输出结果，如图 30-30 所示。

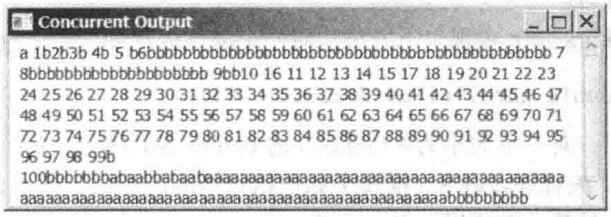


图 30-30　在一个文本域中显示三个线程的输出结果

30.2 （赛车）使用线程改写编程练习题 15.29，使之控制汽车赛跑。通过将两个程序中的延迟时间都设置为 10，比较此程序和编程练习题 15.29 中的程序，哪个运行动画快一些？

30.3 （升旗）使用线程模拟升旗动画来改写程序清单 15-13。通过将两个程序中的延迟时间都设置为 10，比较此程序和程序清单 15-13 中的程序，哪个运行动画快一些？

30.8 ～ 30.12 节

30.4 （同步线程）编写一个程序，启动 1000 个线程。每个线程给初始值为 0 的变量 sum 加 1。定义一个 Integer 包装对象来保存 sum。使用同步和不使用同步来运行这个程序，看一看它们的效果。

30.5 （显示转动的风扇）使用控制风扇动画的线程改写编程练习题 15.28。

30.6 （弹跳的球）使用控制球的弹跳动画的线程来改写程序清单 15-17。

30.7 （控制时钟）使用控制时钟动画的线程改写编程练习题 15.32。

30.8 （账户同步）使用对象的 wait() 和 notifyAll() 方法改写程序清单 30-6。

30.9 （演示 ConcurrentModificationException 异常）迭代器具有快速失效特性，编写一个程序来演示该特性，创建两个并发访问和修改集合的线程。第一个线程创建一个用数填充的散列集，并每秒钟向该合集内添加一个新的数。第二个线程获取上述合集的一个迭代器，并通过该迭代器每秒前后遍历一次合集。因为第二个线程遍历合集时，第一个线程正在修改合集，所以，会

得到一个 ConcurrentModificationException 异常。

*30.10 （使用同步合集）使用同步解决前一个练习题中的问题，使得第二个线程不抛出 ConcurrentModificationException 异常。

30.15 节

*30.11 （演示死锁）编写一个演示死锁的程序。

30.18 节

*30.12 （并行数组初始化器）使用 Fork/Join 框架实现下面的方法，可以设置随机值给线性表。

```
public static void parallelAssignValues(double[] list)
```

编写一个测试程序，创建一个具有 9 000 000 个元素的线性表，调用 parallelAssignValues 来赋随机值给线性表。另外实现一个顺序算法，并且比较两种方法执行的时间。注意，如果使用 Math.random()，并行代码的执行时间将比顺序代码的执行时间差，因为 Math.random() 是同步的，不能并行执行。为了解决这个问题，创建一个 Random 对象，用于赋随机值给一个小的线性表。

30.13 （通用的并行合并排序）修改程序清单 30-10，定义一个通用的并行合并算法方法，如下：

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

*30.14 （并行快速排序）实现下面方法，可以并行地使用快速排序对一个线性表进行排序（参见程序清单 23-7）。

```
public static void parallelQuickSort(int[] list)
```

编写一个测试程序，使用该并行方法和一个顺序方法，对一个大小为 9 000 000 的线性表的执行时间进行计时。

*30.15 （并行求和）使用 Fork/Join 实现以下方法，对一个线性表求和。

```
public static double parallelSum(double[] list)
```

编写一个测试程序，对一个大小为 9 000 000 的 double 值求和。

*30.16 （并行的矩阵加法）编程练习题 8.5 描述了如何执行矩阵的加法。假设你有一个多处理器，因此可以加速矩阵加法计算。实现以下并行方法：

```
public static double[][] parallelAddMatrix(
  double[][] a, double[][] b)
```

编写一个测试程序，分别对使用并行方法和顺序方法来实现两个 2000 × 2000 的矩阵加法计时。

*30.17 （并行的矩阵乘法）编程练习题 7.6 描述了如何执行矩阵的乘法。假设你有一个多处理器，因此可以加速矩阵乘法计算。实现以下并行方法：

```
public static double[][] parallelMultiplyMatrix(
  double[][] a, double[][] b)
```

编写一个测试程序，分别对使用并行方法和顺序方法来实现两个 2 000 × 2 000 的矩阵乘法计时。

*30.18 （并行计算八皇后问题）修改程序清单 22-11，开发一个并行算法，为八皇后问题找到所有的解决方案。（提示：运行 8 个子任务，每个子任务将皇后放在第一行的不同列中。）

综合

***30.19 （排序动画）为选择排序、插入排序和冒泡排序编写一个动画，如图 30-31 所示。创建一个由整数 1，2，…，50 构成的数组，然后随机地打乱它。创建一个面板来显示这个数组。需要在每个单独的线程中调用每个排序方法。每个算法使用两个嵌套的循环。当算法结束外层循环的一

次遍历，让线程休眠 0.5 秒，然后重新显示该柱状图中的数组。将排好序的子数组的最后一个柱条彩色显示。

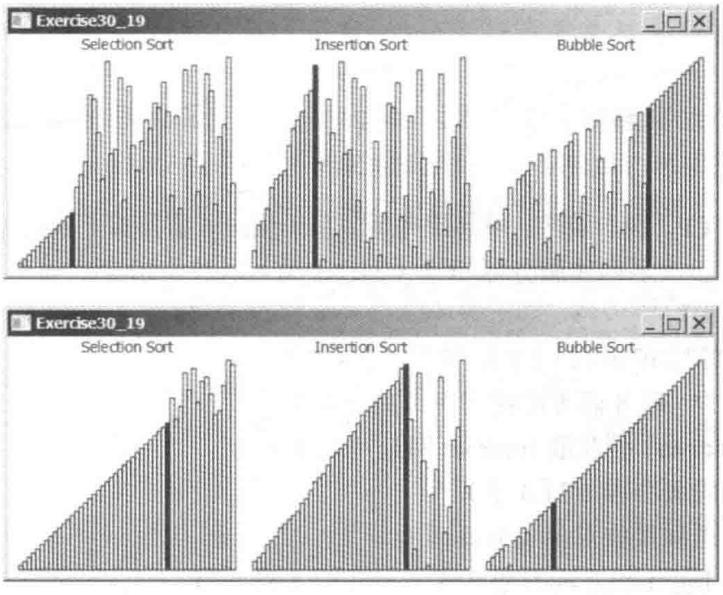
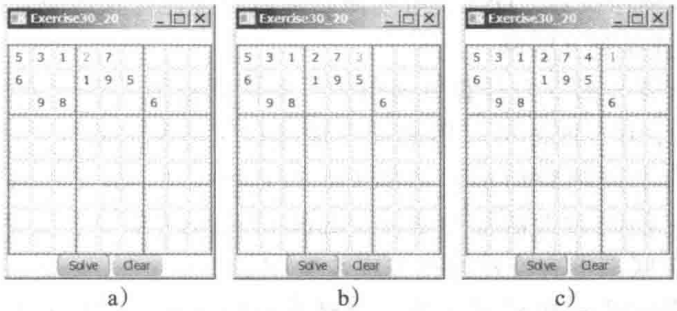


图 30-31　动画演示三种排序算法

***30.20 （数独搜索模拟）修改编程练习题 22.21，显示搜索的中间结果。图 30-32 给出了动画的一个截屏，数字 2 放在如图 30-32a 所示的单元中，数字 3 放在如图 30-32b 所示的单元中，数字 1 放在如图 30-32c 所示的单元中。模拟显示所有的搜索步骤。



a）　b）　c）

图 30-32　为数独问题动画显示中间搜索步骤

30.21 （结合碰撞的弹球）修改编程练习题 20.5，使用一个线程来动画模拟弹球的移动。

***30.22 （八皇后问题动画）修改程序清单 22-11，显示搜索的中间结果。如图 30-33 所示，高亮显示被搜索的当前行。每秒钟，棋盘的一个新状态被显示。

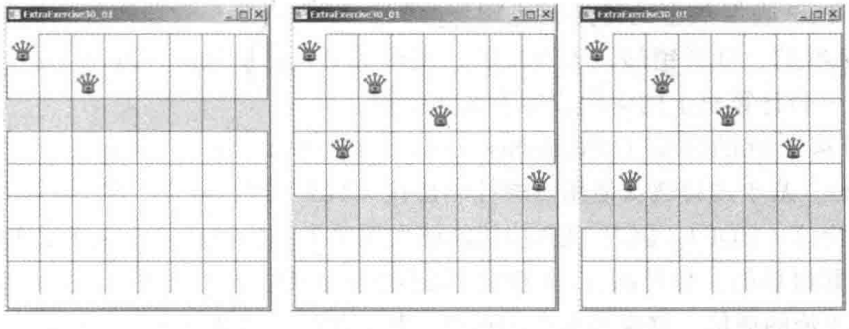


图 30-33　为八皇后问题动画显示中间搜索步骤