$O(n\log n)$ 时间。

该算法的实现留作练习题（参见编程练习题 22.11）。

复习题

22.24 什么是凸包？

22.25 描述如何使用卷包裹算法来找到凸包。列表 H 需要使用 `ArrayList` 或者 `LinkedList` 来实现吗？

22.26 描述如何使用格雷厄姆算法来找到凸包。为什么算法使用栈来存储凸包中的点？

关键术语

average-case analysis（平均情况分析）
backtracking approach（回溯法）
best-case input（最佳情况输入）
big *O* notation（大 *O* 标记）
brute force（穷举法）
constant time（常量时间）
convex hull（凸包）
divide-and-conquer approach（分而治之法）
dynamic programming approach（动态编程法）
exponential time（指数时间）
growth rate（增长率）
logarithmic time（对数时间）
quadratic time（二次时间）
space complexity（空间复杂度）
time complexity（时间复杂度）
worst-case input（最差情况输入）

本章小结

1. 大 *O* 标记是分析算法性能的理论方法。它估计算法的执行时间随着输入规模的增加会有多快的增长。因此，可以通过检查两个算法的增长率来比较它们。
2. 导致最短执行时间的输入称为最佳情况输入，而导致最长执行时间的输入称为最差情况输入。最佳情况和最差情况都不具有代表性，但是最差情况分析非常有用。你可以确保算法永远不会比最差情况还慢。
3. 平均情况分析试图在所有可能的相同规模的输入中确定平均时间。平均情况分析是比较理想的，但是完成很困难，因为对于许多问题，要确定不同输入实例的相对概率和分布是相当困难的。
4. 如果执行时间与输入规模无关，我们就说该算法耗费了常量时间，以符号 $O(1)$ 表示。
5. 线性查找耗费 $O(n)$ 时间。具有 $O(n)$ 时间复杂度的算法称为线性算法，它表现为线性增长率。二分查找耗费 $O(\log n)$ 时间。具有 $O(\log n)$ 时间复杂度的算法称为对数算法，它表现为对数增长率。
6. 选择排序的最差情况时间复杂度为 $O(n^2)$。具有 $O(n^2)$ 时间复杂度的算法称为平方级算法，它表现为平方级增长率。
7. 汉诺塔问题的时间复杂度是 $O(2^n)$。具有 $O(2^n)$ 时间复杂度的算法称为指数算法，它表现为指数增长率。
8. 求出给定下标处的斐波那契数可以使用动态编程在 $O(n)$ 时间内求解。
9. 动态编程是通过解决子问题，然后将子问题的结果结合来获得整个问题的解的过程。动态编程的关键思想是只解决子问题一次，并将子问题的结果存储以备后用，从而避免了重复的子问题的求解。
10. 欧几里得的 GCD 算法需要 $O(\log n)$ 时间。
11. 所有小于等于 n 的素数可以在 $O\left(\frac{n\sqrt{n}}{\log n}\right)$ 时间内找到。
12. 使用分而治之法可以在 $O(n\log n)$ 时间内找到最近点对。
13. 分而治之法将问题分解为子问题，解决子问题，然后将子问题的解答合并从而获得整个问题的解答。和动态编程不一样的是，分而治之法中的子问题不会交叉。子问题类似初始问题，但是具有更小的尺寸，因此可以应用递归来解决这样的问题。

14. 可以使用回溯法解决八皇后问题。
15. 回溯法渐进地寻找一个备选方案，一旦确定该备选方案不可能是一个有效方案，则放弃掉，继而寻找一个新的备选方案。
16. 使用卷包裹法可以在 $O(n^2)$ 时间内找到一个点集的凸包，使用格雷厄姆算法则需要 $O(n\log n)$ 时间。

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

*22.1 （最大连续递增的有序子串）编写一个程序，提示用户输入一个字符串，然后显示最大连续递增的有序子串。分析你的程序的时间复杂度。下面是一个运行示例：

```
Enter a string:abcabcdgabxy ↵Enter
abcdg
```

```
Enter a string: abcabcdgabmnsxy ↵Enter
abmnsxy
```

**22.2 （最大增序子序列）编写一个程序，提示用户输入一个字符串，然后显示最大的增序子串。分析你的程序的时间复杂度。下面是一个运行示例：

```
Enter a string: Welcome ↵Enter
Welo
```

*22.3 （模式匹配）编写一个程序，提示用户输入两个字符串，然后检测第二个字符串是否是第一个字符串的子串。假定在字符串中相邻的字符是不同的。（不要使用 `String` 类中的 `indexOf` 方法。）分析你的算法的时间复杂度。你的算法至少需要 $O(n)$ 时间。下面是该程序的一个运行示例：

```
Enter a string s1: Welcome to Java ↵Enter
Enter a string s2: come ↵Enter
matched at index 3
```

*22.4 （模式匹配）编写一个程序，提示用户输入两个字符串，然后检测第二个字符串是否是第一个字符串的子串。（不要使用 `String` 类中的 `indexOf` 方法。）分析你的算法的时间复杂度。下面是该程序的一个运行示例：

```
Enter a string s1: Mississippi ↵Enter
Enter a string s2: sip ↵Enter
matched at index 6
```

*22.5 （同样个数的子序列）编写一个程序，提示用户输入一个以 0 结束的整数序列，找出有同样数字的最长的子序列。下面是该程序的一个运行示例：

```
Enter a series of numbers ending with 0:
2 4 4 8 8 8 8 2 4 4 0 ↵Enter
The longest same number sequence starts at index 3 with 4 values of 8
```

*22.6 （GCD 的执行时间）编写一个程序，使用程序清单 22-3 和程序清单 22-4 中的算法，求下标从 40 到 45 的每两个连续的斐波那契数的 GCD，获取其执行时间。你的程序应该打印如下所示的一个表格：

	40	41	42	43	44	45
程序清单 22.3 GCD						
程序清单 22.4 GCDEuclid						

（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;
```

**22.7 （最近的点对）22.8 节介绍了一个使用分而治之方法求最近点对的算法。实现这个算法，使其满足下面的要求：

- 使用和编程练习题 20.4 相同的方式定义类 Point 和 CompareY。
- 定义一个名为 Pair 的类，它的数据域 p1 和 p2 表示两个点，名为 getDistance() 的方法返回这两个点之间的距离。
- 实现下面的方法：

```
/** Return the distance of the closest pair of points */
public static Pair getClosestPair(double[][] points)

/** Return the distance of the closest pair of points */
public static Pair getClosestPair(Point[] points)

/** Return the distance of the closest pair of points
  * in pointsOrderedOnX[low..high]. This is a recursive
  * method. pointsOrderedOnX and pointsOrderedOnY are
  * not changed in the subsequent recursive calls.
  */
public static Pair distance(Point[] pointsOrderedOnX,
    int low, int high, Point[] pointsOrderedOnY)

/** Compute the distance between two points p1 and p2 */
public static double distance(Point p1, Point p2)

/** Compute the distance between points (x1, y1) and (x2, y2) */
public static double distance(double x1, double y1,
    double x2, double y2)
```

**22.8 （不大于 10 000 000 000 的所有素数）编写一个程序，找出不大于 10 000 000 000 的所有素数。大概有 455 052 511 个这样的素数。你的程序应该满足下面的要求：

- 应该将这些素数都存储在一个名为 PrimeNumber.dat 的二进制数据文件中。当找到一个新素数时，将该数字追加到这个文件中。
- 为了判定一个新数是否是素数，程序应该从数据文件加载这些素数到一个大小为 10 000 的 long 型的数组中。如果数组中没有任何数是这个新数的除数，继续从该数据文件中读取下 10 000 个素数，直到找到除数或者读取完文件中的所有数字。如果没找到除数，这个新的数字就是素数。
- 因为执行该程序要花很长时间，所以应该把它作为 UNIX 机器上的一个批处理任务来运行。如果机器被关闭或重启，程序应该使用二进制数据文件中存储的素数来继续，而不是从零开始启动。

**22.9 （几何：找到凸包的卷包裹算法）22.10.1 节介绍了为一个点集找到一个凸包的卷包裹算法。假定使用 Java 的坐标系统表示点，使用下面的方法实现该算法：

```
/** Return the points that form a convex hull */
public static ArrayList<Point2D> getConvexHull(double[][] s)
```

Point2D 在 9.6 节中定义。

编写一个测试程序，提示用户输入点集的大小以及点，然后显示构成一个凸包的点的信息。下面是一个运行示例：

```
How many points are in the set? 6 Enter
Enter 6 points: 1 2.4 2.5 2 1.5 34.5 5.5 6 6 2.4 5.5 9 Enter
The convex hull is
  (1.5, 34.5) (5.5, 9.0) (6.0, 2.4) (2.5, 2.0) (1.0, 2.4)
```

22.10 （素数的个数）编程练习题 22.8 将素数存储在一个名为 PrimeNumbers.dat 的文件中。编写一个程序，找出小于或等于 10、100、1 000、10 000、100 000、1 000 000、10 000 000、100 000 000、1 000 000 000、10 000 000 000 的素数个数。你的程序应该从 PrimeNumbers.dat 文件中读取数据。

**22.11 （几何：寻找凸包的格雷厄姆算法）22.10.2 节介绍了为一个点集寻找凸包的格雷厄姆算法。假定使用 Java 的坐标系统表示点。使用下面的方法实现该算法：

```
/** Return the points that form a convex hull */
public static ArrayList<MyPoint> getConvexHull(double[][] s)

MyPoint is a static inner class defined as follows:
  private static class MyPoint implements Comparable<MyPoint> {
    double x, y;

    MyPoint rightMostLowestPoint;

    MyPoint(double x, double y) {
      this.x = x; this.y = y;
    }

    public void setRightMostLowestPoint(MyPoint p) {
      rightMostLowestPoint = p;
    }

    @Override
    public int compareTo(MyPoint o) {
      // Implement it to compare this point with point o
      // angularly along the x-axis with rightMostLowestPoint
      // as the center, as shown in Figure 22.10b. By implementing
      // the Comparable interface, you can use the Array.sort
      // method to sort the points to simplify coding.
    }
  }
```

编写一个测试程序，提示用户输入点集的大小和点，然后显示构成一个凸包的点。下面是一个运行示例：

```
How many points are in the set? 6 Enter
Enter 6 points: 1 2.4 2.5 2 1.5 34.5 5.5 6 6 2.4 5.5 9 Enter
The convex hull is
  (1.5, 34.5) (5.5, 9.0) (6.0, 2.4) (2.5, 2.0) (1.0, 2.4)
```

*22.12 （最后的 100 个素数）编程练习题 22.8 将素数存储在一个名为 PrimeNumbers.dat 的文件中。编写一个高效程序，从该文件中读取最后 100 个素数。

（提示：不要从文件中读取所有的数字，跳过文件中最后 100 个数之前的所有数。）

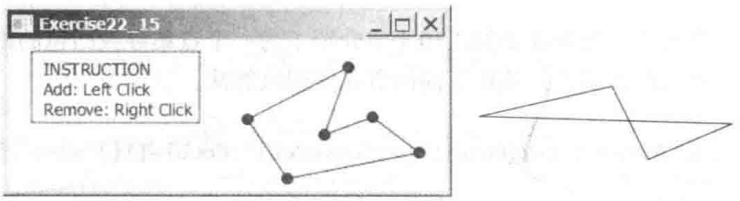
**22.13 （几何：凸包动画）编程练习题 22.11 为从控制台输入的点集中找到凸包。编写一个程序，可以让用户通过单击鼠标左 / 右键来添加 / 移除点，然后显示凸包，如图 22-8c 所示。

*22.14 （素数的执行时间）编写一个程序，使用程序清单 22-5 ～程序清单 22-7 中的算法，找出小于 8 000 000、10 000 000、12 000 000、14 000 000、16 000 000 和 18 000 000 的所有素数，获取其执行时间。你的程序应该打印如下所示的一个表格：

	8000000	10000000	12000000	14000000	16000000	18000000
程序清单 22.5						
程序清单 22.6						
程序清单 22.7						

**22.15 （几何：无交叉多边形）编写一个程序，可以让用户通过单击鼠标左 / 右键来添加 / 移除点，然

后显示一个连接所有点的无交叉多边形，如图 22-11a 所示。如果一个多边形有两条或者更多的边是相交的，则认为是交叉多边形，如图 22-11b 所示。使用如下算法来从一个点集中构建一个多边形。



a) b) 交叉多边形

图 22-11 a) 编程练习题 22.15 为一个点集显示一个无交叉多边形；b) 在一个交叉多边形中，两条或者更多的边是相交的

步骤 1：给定一个点集 S，选择 S 中的最右下角点 p_0。

步骤 2：将 S 中的点按照以 p_0 为原点的 x 轴夹角进行排序。如果出现同样的值，即两个点具有同样的角度，则认为离 p_0 较近的那个点具有更大的角度。S 中的点现在排序为 p_0，p_1，p_2，…，p_{n-1}。

步骤 3：排好序的点形成了一个无交叉多边形。

**22.16 （线性查找动画）编写一个程序，显示线性查找的动画。创建一个包含从 1 到 20 的 20 个不同数字并且顺序随机的数组。数组元素以直方图显示，如图 22-12 所示。你需要在文本域中输入一个查找键值。单击 step 按钮将引发程序执行算法中的一次比较，重绘直方图并且其中一个条形显示查找的位置。这个按钮同时冻结文本域以防止其中的值被改变。当算法结束时，在 border 面板的顶部标签中显示状态，从而给出用户信息。单击 Reset 按钮创建一个新的随机数组，从而开始一次新的查找。这个按钮也使得文本域可以编辑。

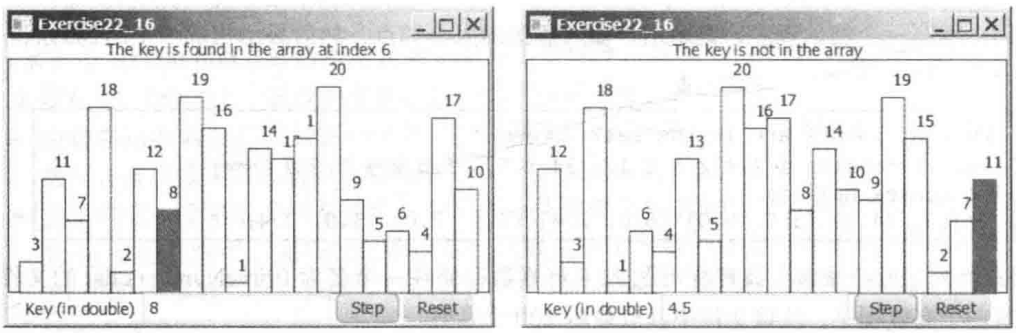


图 22-12 程序显示线性查找的动画

**22.17 （最近点对的动画）编写一个程序，可以让用户通过单击鼠标左 / 右键来添加 / 移除点，然后显示一条连接最近点对的直线，如图 22-4 所示。

**22.18 （二分查找动画）编写一个程序，显示二分查找的动画。创建一个包含从 1 到 20 的顺序数字的数组。数组元素以直方图显示，如图 22-13 所示。你需要在文本域中输入一个搜索键值。单击 Step 按钮将引发程序执行算法中的一次比较。使用淡灰色来绘制代表目前查找范围内的数字的条形，使用黑色绘制表示查找范围的中间数的条形。Step 按钮同时冻结文本域以防止其中的值被改变。当算法结束时，在 border 面板的顶部标签中显示状态信息。单击 Reset 按钮创建一个新的随机数组，从而开始一次新的查找。这个按钮也使得文本域可以编辑。

*22.19 （最大块）编程练习题 8.35 描述了寻找最大块的问题。设计一个动态编程的算法，从而在 $O(n^2)$ 时间内求解这个问题。编写一个测试程序，显示一个 10 × 10 的方格矩阵，如图 22-14a 所示。

矩阵中的每个元素为 0 或者 1，单击 Refresh 按钮可以随机生成。在一个文本域的中央显示每个数字。对每个条目使用一个文本域。允许用户改变条目的值。单击 Find Largest Block 按钮找到包含 1 值的最大子块。高亮显示块中的数字，如图 22-14b 所示。参见 www.cs.armstrong.edu/liang/animation/FindLargestBlock.html 上提供的交互式测试。

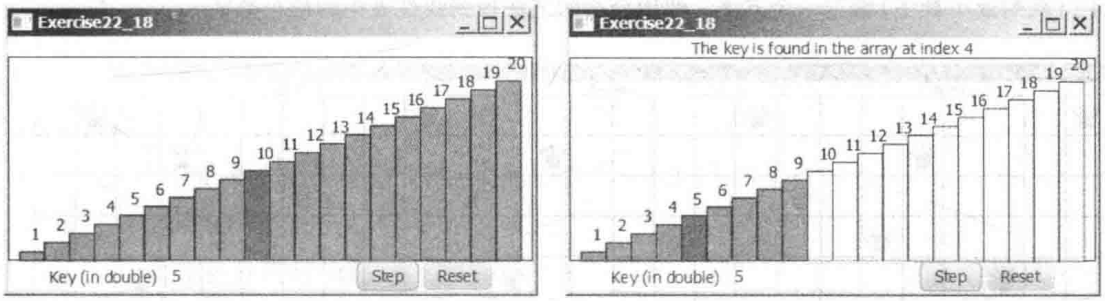
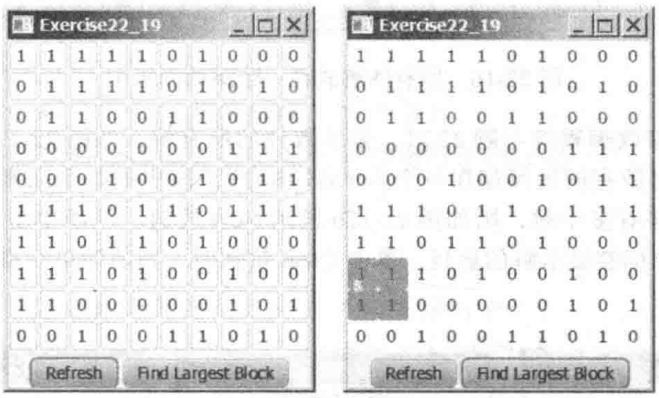


图 22-13 程序显示二分查找的动画

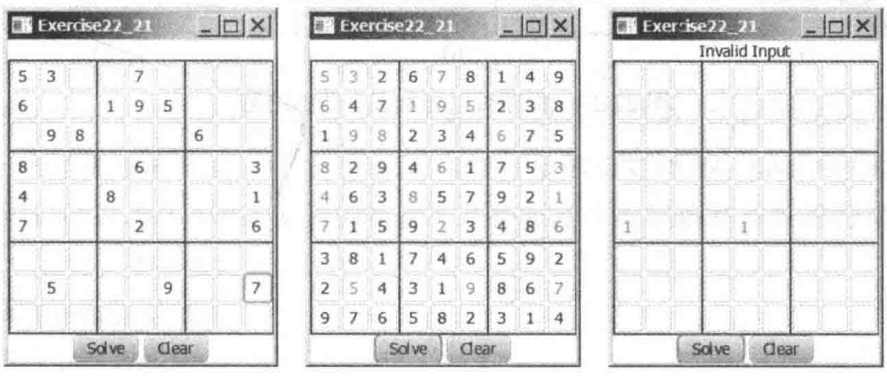


a) b)

图 22-14 程序找到包含 1 的最大块

***22.20 （游戏：多个数独的解答）补充材料 VI.A 给出了数独问题的完整求解。数独问题可能有多个解答。修改补充材料 VI.A 中的 Sudoku.java 显示解决方案的总数。如果多个解决方案存在，则显示两个解决方案。

***22.21 （游戏：数独）补充材料 VI.C 给出了数独问题的完整求解。编写一个程序，提示用户从文本域输入数字，如图 22-15a 所示。单击 Solve 按钮显示结果，如图 22-15b ～图 22-15c 所示。



a) b) c)

图 22-15 解决数独问题的程序

***22.22 （游戏：递归数独）为数独问题编写一个递归的解法。

***22.23 （游戏：多个八皇后问题的解答）编写一个程序，在一个滚动面板中显示八皇后问题的所有可能解，如图 22-16 所示。对于每个解，使用标签标记解决方案的数字。（提示：将所有解的面板放在一个 HBox 中，然后将其放入一个 ScrollPane 中。）

**22.24 （找到最小数字）编写一个方法，使用分而治之法找到线性表中的最小数字。

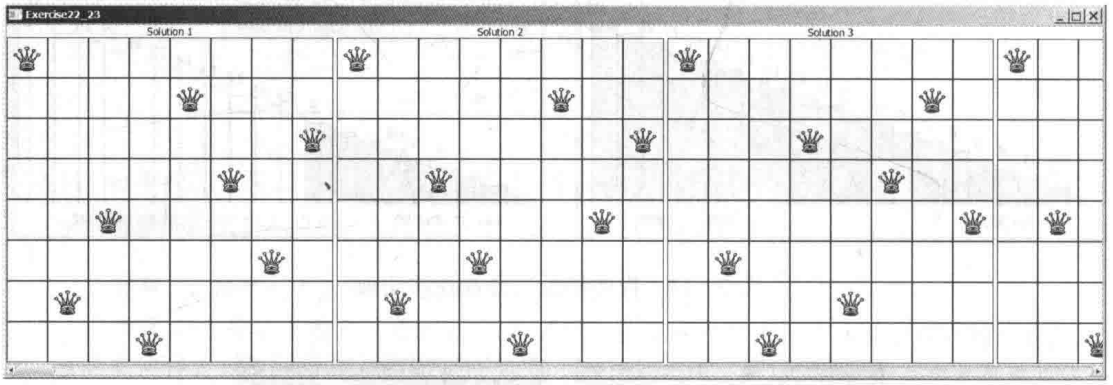
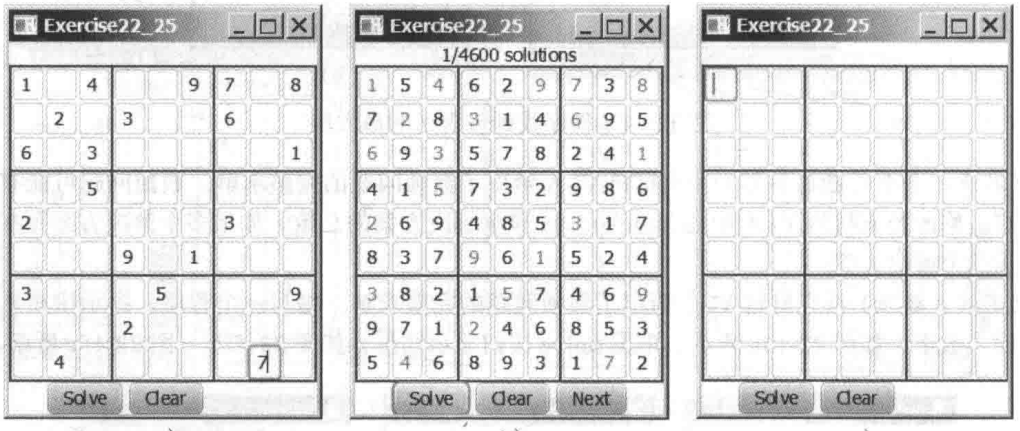


图 22-16 所有的解放在一个滚动面板中

***22.25 （游戏：数独）修改编程练习题 22.21，显示数独的所有解，如图 22-17a 所示。当单击 Solve 按钮时，程序将所有的解保存在一个 ArrayList 中。表中的每个元素都是一个二维的 9×9 网格。如果程序有多个解，则如图 22-17b 显示 Next 按钮。可以单击 Next 按钮显示下一个解，同样有一个标签显示解的数目。单击 Clear 按钮时，则清除单元格，隐藏 Next 按钮，如图 22-17c 所示。



a) b) c)

图 22-17 程序可以显示多个数独的解