

# 数据结构与算法 作业报告

---

第一次



姓名 曹家豪

---

班级 软件 2204 班

---

学号 2226114017

---

电话 13572763245

---

Email caojiahao@stu.xjtu.edu.cn

---

日期 2023-11-27

---

## 目录

任务一：证明 .....	2
任务二： .....	3
1.题目：设计算法要多思考 .....	3
2.数据与算法设计 .....	3
3.部分代码说明 .....	3
4.运行结果： .....	5
5.总结收获： .....	5
任务三 .....	5
1.题目：排序算法的实现 .....	5
2.数据与算法设计： .....	6
3.部分代码说明： .....	7
4.运行结果展示： .....	10
5.总结与收获： .....	10
任务四 .....	10
1.题目：排序算法性能测试和比较 .....	10
2.数据与算法设计： .....	11
3.部分代码说明： .....	11
4.运行结果展示： .....	12
5.总结与收获： .....	14
任务五 .....	15
1.题目：数据分布对排序算法的影响 .....	15
2.数据与算法设计 .....	15
3.部分代码说明： .....	15
4.运行结果展示： .....	16
5.总结与收获： .....	18
任务六 .....	19
1.题目：快速排序的再探讨和应用 .....	19
2.问题一 .....	19
3.问题二 .....	24
4.问题三 .....	26
5.问题四 .....	27
附录 .....	31

## 任务一：证明

1. (a) 令  $n_0 = 1, c = 7, f(n) = \sqrt{n}$

对任意  $n > n_0$  有  $2\sqrt{n} + 6 \leq 7\sqrt{n}$

$$\therefore 2\sqrt{n} + 6 = O(\sqrt{n})$$

(b) 令  $n_0 = 1, c = 1, f(n) = \sqrt{n}$

对任意  $n > n_0$  有  $n^2 \geq n$

$$\therefore n^2 = \Omega(n)$$

(c) 令  $n_0 = 1, c = \log_2 e$

对任意  $n > 1$  有  $\log_2 n = \log_2 e \cdot \frac{\log_2 n}{\log_2 e} = \log_2 e \cdot \ln(n)$

$$\therefore \log_2(n) = \log_2 e \cdot \ln(n)$$

即  $\log_2 n$  既在  $O(\ln(n))$  中又在  $\Omega(\ln(n))$  中

$$\therefore \log_2 n = \Theta(\ln(n))$$

(d) 设存在  $n_0, c$  使得任给  $n > n_0$  有  $4^n \leq c \cdot 2^n$

$$\text{则有 } 2^n(c - 2^n) \geq 0 \rightarrow 2^n \leq c$$

即任给  $n > n_0$  时有  $n \leq \log_2 c$  无法满足

$\therefore$  不存在  $n_0, c$  使得任给  $n > n_0$  有  $4^n \leq c \cdot 2^n$

$$\therefore 4^n \neq O(2^n)$$

2. 令  $n_0 = 1, c = \frac{1}{2}$

$$\text{当 } n = 1 \text{ 时 } T(n) = 1 \geq \frac{1}{2}(\log_2(1))$$

$$\text{设 } n = 2^k \text{ 时 } T(n) \geq \frac{1}{2} n \log_2(n), \quad k \in \mathbb{N}$$

$$T(2^n - a) = 2T(n) + n - \frac{a}{2} \geq n \log_2(n) + n - \frac{a}{2}, \quad 0 < a \leq n$$

$$\text{又 } \frac{1}{2}(2^n - a) \log_2(2^n - a) \leq \frac{1}{2}(2^n - a) \log_2 2^n = n - \frac{a}{2} + \frac{2^n - a}{2} \log_2(n)$$

$$\therefore n \log_2(n) + n - \frac{a}{2} \geq \frac{2^n - a}{2} \log_2(n) + n - \frac{a}{2}$$

$$\therefore T(2^n - a) \geq \frac{2^n - a}{2} \log_2(2^n - a)$$

$$T(2^n - a) = \Omega(2^n - a) \log_2(2^n - a)$$

$$\therefore n \in \mathbb{N}^* \text{ 时 } T(n) = \Omega(n \log(n))$$

## 任务二：

### 1.题目：设计算法要多思考

已知一张图片是对某个事物横截面的扫描结果图，该图片的宽度是  $m$ ，高度是  $n$ ，图片的每一个像素只会由两种颜色之一构成：要么是蓝色，要么是白色。图片中的每一列的颜色分布有如下两种情形：

① 整列所有像素的颜色全是白色；

② 列中像素可以是白色或者蓝色，但在这种情况下，要么所有蓝色像素集中在从上到下，要么所有蓝色像素集中在从下到上，也就是说不会出现蓝色和白色相间的情形。

如果定义每一列的长度为蓝色像素的数量，那么如何求解图片中长度最大的列的长度呢？朴素的算法的时间复杂度是  $O(mn)$ ，但该任务要求完成的算法的时间复杂度必须满足  $O(m\log n)$ 。2 在随实验的附件中有一个 `tomography.png` 图片，同学们可以使用这张图片做为测试数据，图片的大小是  $1200 \times 1600$ ，该图片中最大列长是 1575。对图片的处理可以继续使用在面向对象程序设计课程中构建的 `Picture` 类型。

### 2.数据与算法设计

1) 首先分析问题，需要达到  $O(m\log n)$ 的时间复杂度， $m$ ， $n$  分别为图片的列、行数，则可考虑一种算法，对每一列进行遍历，计算每一列的长度并比较得出最大值，遍历所有行的时间复杂度为  $O(m)$ ，那么当计算每列长度的算法满足时间复杂度为  $O(\log n)$ 时即可满足题意。

2) 要使计算每列长度的算法满足时间复杂度为  $O(\log n)$ ，考虑采用二分法。

3) 这里定义了 `getLength(int i, Picture sample)`方法以获取 `sample` 图像第  $i$  列的长度，其实现逻辑如下：

因为两种颜色时严格分开的，那么只需找到分界点位置即可。定义两个指针 `left` 与 `right` 指向当前的边界，定义中间指针 `pointer` 指向 `left` 与 `right` 中间的位置，当 `pointer` 与 `pointer+1` 位置的颜色相同（需要在 `Picture` 类中定义 `isSameAsNext` 方法）时：

①如果 `pointer` 位置与首位置的颜色相同，说明分界点在右半部分，更新左指针，使 `left=pointer+1`；

②反之说明分界点在左半部分，更新右指针，令 `right=pointer-1`；

直到 `pointer` 到达边界或 `pointer` 与 `pointer+1` 位置颜色不同时终止，此时 `pointer` 即为两种颜色的分界点（或者纯色列的边界），此时只需要根据首位值的颜色判断最终长度是前半部分还是后半部分长度即可。

4) 在主函数中定义 `curlonggest` 为当前的最长列长，初始化为 0，然后开始遍历每一列，调用 `getLength` 方法计算当前列长，当结果大于 `curlonggest` 时，更新 `curlonggest` 的值。遍历结束后，`curlonggest` 则为图片的最大列长度。

### 3.部分代码说明

主函数：

```
1. int curlongest=0;
2. //定义当前最大列长
3. try {
```

```

4.      Picture testimage = new Picture(testpicture);//
      读入图片
5.      for(int i=0;i<testimage.getWidth();i++){
6.          //遍历每一列
7.          int curlength=getLength(i,testimage);
8.          //获取当前列长
9.          if(curlength>=curlongest){
10.             curlongest=curlength;
11.             //若当前列长更大, 则更新 curlongest
12.         }
13.     }
14.     System.out.println("图片的最长列长度为:
      "+curlongest);
15.     } catch (IOException e) {
16.         System.out.println("图片读取失败");
17.     }
18. }

```

getLength 方法:

```

1. public static int getLength(int i, Picture sample) {
2.     Color firstcolor = sample.getColor(i, 0);
3.     //获取首位值颜色
4.     int left = 0;
5.     int right = sample.getHeight() - 1;
6.     int pointer = (left + right) / 2;
7.     //定义头、中、尾指针
8.     while ((pointer < (sample.getHeight() - 1))&&sample
      .isSameAsNext(i, pointer))
9.         //开始二分, 不断循环
10.        //终止条件: pointer 到达分界点或边界
11.        { //pointer 与下一个颜色相同时
12.            if (sample.isSame(i, 0, i, pointer))
13.                //如果 pointer 位置与首位置的颜色相同, 说明分界点在
                右半部分, 更新左指针
14.                {left = pointer + 1;}
15.            else{
16.                //反之说明分界点在左半部分, 更新右指针
17.                right = pointer - 1;
18.            }
19.            pointer = (left + right) / 2;
20.            //更新中间指针
21.        }
22.        if (pointer == (sample.getHeight() - 1)) {
23.            //根据首位值颜色判断返回长度
24.            if (firstcolor.equals(WHITE)) {
25.                return 0;

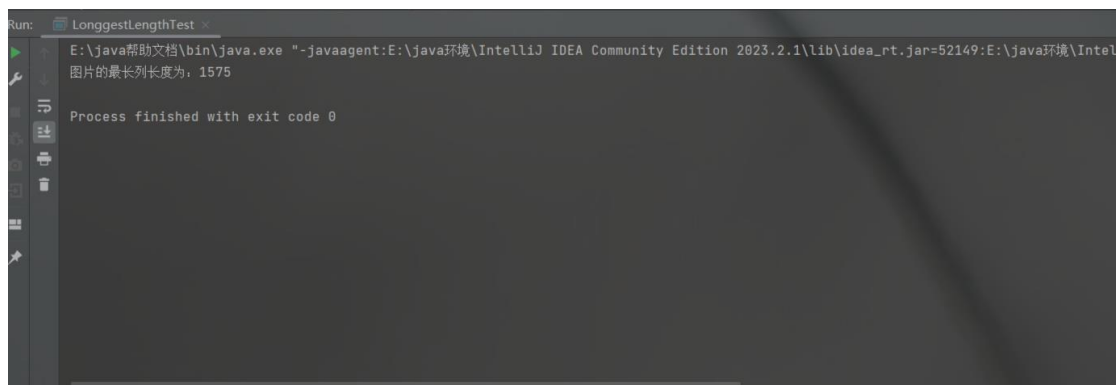
```

```

26.         } else {
27.             return sample.getHeight();
28.         }}
29.         if (firstcolor.equals(BLUE)) {
30.             return pointer + 1;
31.         }
32.         else {
33.             return sample.getHeight() - (pointer + 1);
34.         }
35.     }

```

## 4.运行结果：



## 5.总结收获：

- 1) 对算法的时间复杂度有了更深的理解，能够根据给定的时间复杂度判读采用何种算法
- 2) 更熟练地掌握与运用二分思想解题
- 3) 解决问题过程中有许多对边界值的判断，一定程度上提高了处理边界的能力

## 任务三

### 1.题目：排序算法的实现

参照 Insertion 类的实现方式，为其他四个排序算法实现相对应的类类型。这些类类型中有可能 需要相配合的成员方法，请同学们灵活处理。其中 Shell 排序中的间隔递减序列采用如下函数：

$h_1 = 1 \quad h_i = h_{i-1} * 3$

要求：

每个排序算法使用课堂上所讲授的步骤，不要对任何排序算法进行额外的优化；

对每个排序算法执行排序之后的数据可以调用 *SortAlgorithm* 类型中的成员方法 *isSorted* 进行测试，检查是否排序成功。

## 2.数据与算法设计：

### 1) 主要工作原理：

①选择排序：开启一个循环，每轮从未排序区间选择最小的元素，将其放到已排序区间的末尾缩小未排序区间范围。

②希尔排序：选择一个间隔序列（本题中为：N/3, N/9, N/27...1）。按照间隔序列将原始数组分成若干个子数组。对每个子数组进行插入排序。逐步缩小间隔序列，重复前两个步骤，直到最后一次以间隔为 1 的插入排序完成排序。

③快速排序：选择数组中的某个元素作为“轴值”，将所有小于轴值的元素移到其左侧，而大于轴值的元素移到其右侧。对左侧与右侧序列再次进行这一过程，直至序列无法分割。

④归并排序：归并排序包括“划分”和“归并”两个阶段，其中划分阶段通过递归不断地将数组从中点处分开，将长数组的排序问题转换为短数组的排序问题。合并阶段当子数组长度为 1 时终止划分，开始合并，持续地将左右两个较短的有序数组合并为一个较长的有序数组，直至结束。

2) 在该任务中，Insertion、Selection、Shell、Quicksort、MergeSort 都是虚拟类 SortAlgorithm 的子类，因而要实现 SortAlgorithm 中的 sort 方法，具体实现借助上面提到的工作原理。对于选择排序与希尔排序不再过多解释，主要阐述该任务中对快速排序与归并排序的实现：

### 3) 快速排序的实现

①主函数逻辑如下所示

■ 算法的基本步骤如下：

- 如果纪录序列S中的元素个数是0或1，则返回
- 取S中任一元素v，称为轴值(pivot)
- 将S-{v}（S中的其余元素）划分成两个不相交的集合： $L=\{x \in S-\{v\} | x < v\}$  和  $G=\{x \in S-\{v\} | x \geq v\}$
- 递归对L和G纪录序列进行快速排序
- 合并{L, v, G}即为最终的排序序列

### ②轴值选择：

该任务中选取首位置，中间位置与末位置元素的中间值为轴值，再完成轴值选取的过程中同时完成对这三个位置数据的排序

③划分阶段逻辑如下所示



- 将轴值和最右边界的值进行交换
- 为纪录序列两端设置两个游标
  - 左游标=left
  - 右游标=right-1
- 将两个游标朝相向的方向移动
  - 左游标移过小于轴值的所有元素
  - 右游标移过大于轴值的所有元素
  - 当左游标小于右游标则交换两个游标所指的元素
  - 重复这个步骤直到当左游标大于右游标则分割完成
- 将左游标和right所指的元素进行交换

4) 归并排序的实现:

①当序列长度大于 1 时，持续递归划分左半序列与右半序列，划分结束后，实现归并操作  
 ②归并的实现采用双指针，分别指向左半序列和右半序列的起始位置，开辟一个新序列，开始遍历，比较两个指针指向的值大小，将较小值放入新序列中并更新指针，其中一个指针到达末尾后停止，将另一个序列的剩余元素依次放入新序列，最后将新序列拷贝至原序列。

5) 主函数:

在主函数中利用 GenerateData 类产生规模为 20 和 10000 的各五个随机序列，定义五个 SortAlgorithm 类对象分别指向五种算法，利用类中的 sort 方法对随机序列排序，利用 isSorted 方法对排序后的数据进行测试，展示对规模为 20 的排序结果和两种规模的测试结果。

### 3.部分代码说明:

1) QuickSort:

```
1. public void sort(Comparable[] objs,int left,int right){
2.     if(left<right)
3.         //当序列长度大于1 时进行划分
4.         {
5.             //获取轴值位置
6.             int pivotIndex=partition(objs,left,right);
7.             //递归左子序列
```



```

8.         sort(objs,left,pivotIndex-1);
9.         //递归右子序列
10.        sort(objs,pivotIndex+1,right);
11.    }
12. }
13. public int partition(Comparable[] objs,int left,int right)
14.     //按照轴值划分
15. {
16.     int index=findMiddle(objs,left,right);
17.     //取得轴值下标, 同时完成对首、中、尾位置的排序
18.     int low=left,high=right-1;
19.     exchange(objs,right,index);
20.     //将轴值先放在末尾
21.     Comparable pivot=objs[right];
22.     while(low<high){
23.         while(true){
24.             //左游标寻找大于轴值的元素
25.             if(less(pivot,objs[low])||(low>=high)){
26.                 break;
27.             }
28.             low++;
29.         }
30.         while(true){
31.             //右游标寻找小于轴值的元素
32.             if(less(objs[high],pivot)|| (high<=low)){
33.                 break;
34.             }
35.             high--;
36.         }
37.         if(low<high){
38.             //交换两游标指向元素
39.             exchange(objs,low,high);
40.         }
41.     }
42.     //将轴值归位
43.     exchange(objs,low,right);
44.     //返回轴值位置
45.     return low;
46. }

```

## 2)MergeSort:

```

1.     public void sort(Comparable[] objs,int left,int right){
2.         if(left>=right)
3.             return;

```

```

4.         else{
5.             int mid=(left+right)/2;
6.             //划分左子序列
7.             sort(objs,left,mid);
8.             //划分右子序列
9.             sort(objs,mid+1,right);
10.            //归并
11.            merge(objs,left,mid,right);
12.        }
13.    }
14.    public void merge(Comparable[] objs,int left,int mid,int
        t right){
15.        //采用双指针
16.        int frontPoint=left;
17.        int behindPoint=mid+1;
18.        //开辟新数组用于中转
19.        Comparable[] tmp=new Comparable[right-left+1];
20.        int k=0;
21.        while((frontPoint<=mid)&&(behindPoint<=right)){
22.            //将两个指针指向的较小值存入新数组,同时更新指针
23.            if(less(objs[frontPoint],objs[behindPoint])){
24.                tmp[k++]=objs[frontPoint++];
25.            }
26.            else{
27.                tmp[k++]=objs[behindPoint++];
28.            }
29.        }
30.        //将剩余数据依次放入新数组
31.        while(frontPoint<=mid){
32.            tmp[k++]=objs[frontPoint++];
33.        }
34.        while(behindPoint<=right){
35.            tmp[k++]=objs[behindPoint++];
36.        }
37.        //完成拷贝
38.        for(int i=left;i<=right;i++){
39.            objs[i]=tmp[i-left];
40.        }
41.    }
42.

```

## 4.运行结果展示：

```
对数据规模为20的随机序列测试：
随机数组1经插入排序后的结果为：0.0 0.05 0.1 0.15000000000000002 0.2 0.25 0.3 0.35 0.3999999999999997 0.4499999999999996 0.4999999999999995 0.5499999999999994 0.5999999999999993 0.6499999999999992 0.6999999999999991 0.749999999999999 0.7999999999999989 0.8499999999999988 0.8999999999999987 0.9499999999999986 0.9999999999999985

验证为：true
随机数组2经选择排序后的结果为：0.0 0.05 0.1 0.15000000000000002 0.2 0.25 0.3 0.35 0.3999999999999997 0.4499999999999996 0.4999999999999995 0.5499999999999994 0.5999999999999993 0.6499999999999992 0.6999999999999991 0.749999999999999 0.7999999999999989 0.8499999999999988 0.8999999999999987 0.9499999999999986 0.9999999999999985

验证为：true
随机数组3经希尔排序后的结果为：0.0 0.05 0.1 0.15000000000000002 0.2 0.25 0.3 0.35 0.3999999999999997 0.4499999999999996 0.4999999999999995 0.5499999999999994 0.5999999999999993 0.6499999999999992 0.6999999999999991 0.749999999999999 0.7999999999999989 0.8499999999999988 0.8999999999999987 0.9499999999999986 0.9999999999999985

验证为：true
随机数组4经快速排序后的结果为：0.0 0.05 0.1 0.15000000000000002 0.2 0.25 0.3 0.35 0.3999999999999997 0.4499999999999996 0.4999999999999995 0.5499999999999994 0.5999999999999993 0.6499999999999992 0.6999999999999991 0.749999999999999 0.7999999999999989 0.8499999999999988 0.8999999999999987 0.9499999999999986 0.9999999999999985

验证为：true
随机数组5经归并排序后的结果验证为：true
对规模为10000的随机序列测试：
随机数组6经插入排序后的结果验证为：true
随机数组7经选择排序后的结果验证为：true
随机数组8经希尔排序后的结果验证为：true
随机数组9经快速排序后的结果验证为：true
随机数组10经归并排序后的结果验证为：true

Process finished with exit code 0
```

```
0.4999999999999994 0.5499999999999993 0.6 0.65 0.7000000000000001 0.7500000000000001 0.8000000000000002 0.8500000000000002 0.9000000000000002 0.9500000000000003
0.4999999999999994 0.5499999999999993 0.6 0.65 0.7000000000000001 0.7500000000000001 0.8000000000000002 0.8500000000000002 0.9000000000000002 0.9500000000000003
0.4999999999999994 0.5499999999999993 0.6 0.65 0.7000000000000001 0.7500000000000001 0.8000000000000002 0.8500000000000002 0.9000000000000002 0.9500000000000003
0.4999999999999994 0.5499999999999993 0.6 0.65 0.7000000000000001 0.7500000000000001 0.8000000000000002 0.8500000000000002 0.9000000000000002 0.9500000000000003
0.4999999999999994 0.5499999999999993 0.6 0.65 0.7000000000000001 0.7500000000000001 0.8000000000000002 0.8500000000000002 0.9000000000000002 0.9500000000000003
```

## 5.总结与收获：

- 1) 学会自己独立写出选择、插入、希尔、快速、归并五种排序算法
- 2) 在快速排序与归并排序的实现中运用了递归操作，更加了解递归思想的运用，对递归终止的条件也有所学习
- 3) 学习了双指针法进行序列数据的移动

## 任务四

### 1.题目：排序算法性能测试和比较

完成对每一个排序算法在数据规模为：28 、29 、210、……、216 的均匀分布的随机数据序列、正序序列和逆序序列的排序时间统计。

要求：

在同等规模的数据量和数据分布相同下，要做  $T$  次运行测试，用平均值做为此次测试的结果，用以排除因数据的不同和机器运行当前的状态等因素造成的干扰；（在 SortTest 类型的 test 方法参数中有对每次数据规模下的测试次数的指定）

将所有排序算法的运行时间结果用图表的方式进行展示，X 轴代表数据规模，Y 轴代表运行时间。（如果因为算法之间运行时间差异过大而造成显示上的问题，可以通过将运行时间使用取对数的方式调整比例尺）

对实验的结果进行总结：从一个算法的运行时间变化趋势和数据规模的变化角度，从同样的数据规模和相同数据分布下不同算法的时间相对差异上等角度进行阐述。

## 2.数据与算法设计：

1) 使用提供的 GenerateData、SortTest 和 LineXYDemo 类，在修改 LineXYDemo 中的数据集时注意：

①X 轴为数据规模

②Y 轴为对应算法的排序时长。需要用到 SortTest 类中的 test 方法，注意指定次数  $T$

③修改“标题”“横轴标题”“纵轴”标题等，修改不同曲线的图例

④原本提供的 test 方法有误，进行了改进（每次测试时，测试原序列的拷贝，保证再次测试时原序列仍然无序）

2) 依照 LineXYDemo 完成对五种算法对应图表类的创建，在主函数中创建五个对应对象，完成展示操作

3) 做  $T$  次测试取平均值以排除干扰

4) 此外，对运行时间的结果进行了取对数操作使图像更好展示

## 3.部分代码说明：

以 InsertionTestDemo 为例：在创建数据集时：

```
1. private XYDataset createDataset() {
2.     // 本样例中想要显示的是三组数据的变化图
3.     // X 数组是三组数据共同拥有的 x 坐标值；Y1、Y2 和 Y3 数组
    分别存储了三组数据对应的 y 坐标值
4.     // 共有 9 个节点值
5.     int[] X = new int[9];
6.     // 建立 X 坐标
7.     double[] Y1 = new double[9];
8.     double[] Y2 = new double[9];
9.     double[] Y3 = new double[9];
10.    // 建立三种 Y 坐标分别对应三种数据序列
11.    for (int j = 8; j <= 16; j++) {
12.        int N = (int) pow(2, j);
13.        X[j - 8] = N;
14.        //X 为数据规模
15.        Double[] randomData = GenerateData.getAnotherRandomData(N);
16.        Y1[j - 8] = Math.log(SortTest.test(new InsertionTestDemo(), randomData, 10));
```

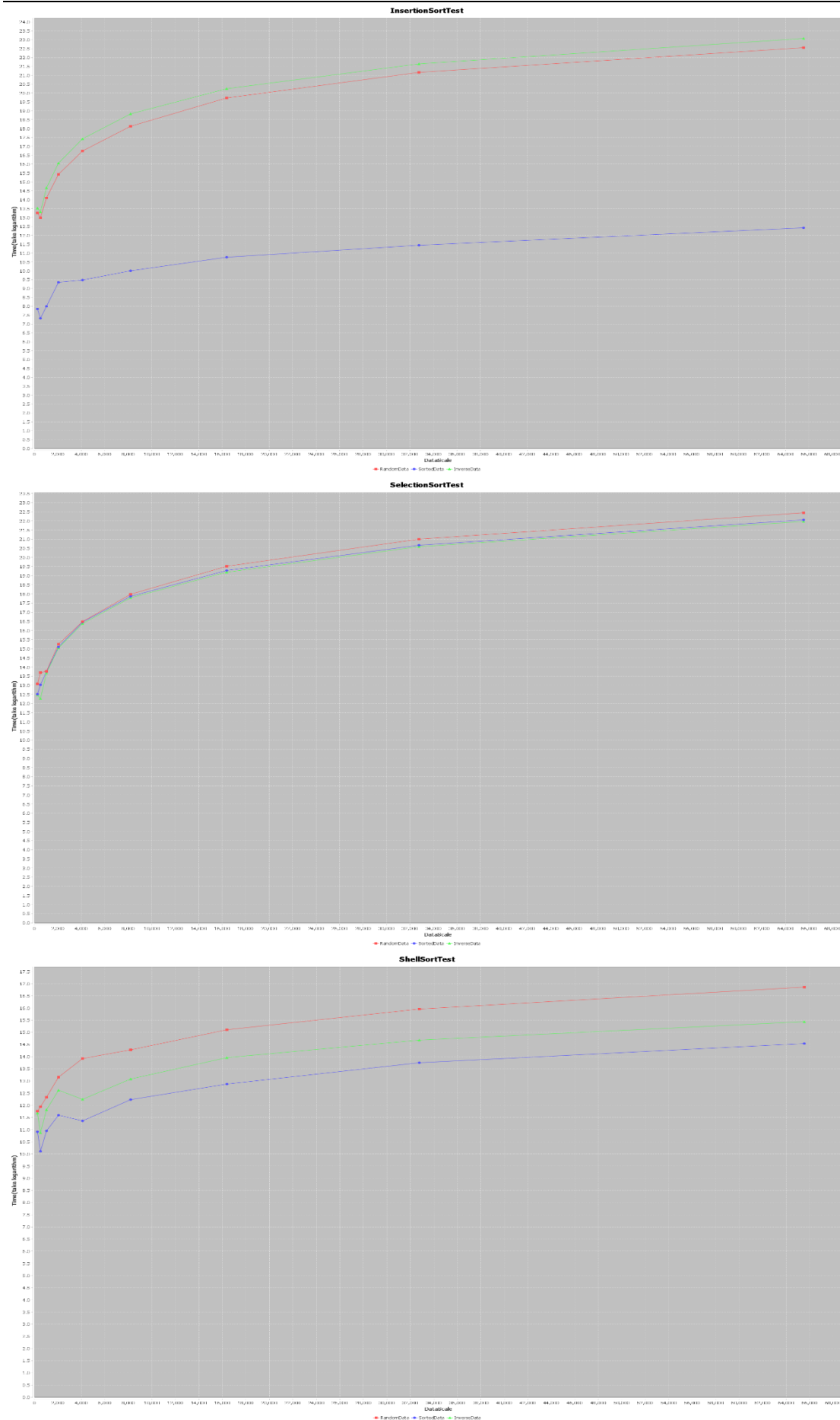
```

17.         Double[] sortedData = GenerateData.getAnotherSo
           rtedData(N);
18.         Y2[j - 8] = Math.log(SortTest.test(new Insertio
           n(), sortedData, 10));
19.         Double[] inverseData = GenerateData.getAnotherI
           nversedData(N);
20.         Y3[j - 8] = Math.log(SortTest.test(new Insertio
           n(), inverseData, 10));
21.         //Y 为该算法对不同种类数据序列测试 10 次用时的平均值
22.     }
23.     double[][] Y = {Y1, Y2, Y3};
24.     // jfreechart 中使用 XYSeries 对象存储一组数据的(x,y)的
           序列, 因为有三组数据所以创建三个 XYSeries 对象
25.     XYSeries[] series = {new XYSeries("RandomData"), ne
           w XYSeries("SortedData"), new XYSeries("InverseData")};
26.     int N = X.length;
27.     int M = series.length;
28.     for (int i = 0; i < M; i++)
29.         for (int j = 0; j < N; j++)
30.             series[i].add(X[j], Y[i][j]);
31.     // 因为在该图表中显示的数据序列不止一组, 所以在
           jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对
           象中
32.     XYSeriesCollection dataset = new XYSeriesCollection
           ();
33.     for (int i = 0; i < M; i++)
34.         dataset.addSeries(series[i]);
35.     return dataset;
36. }

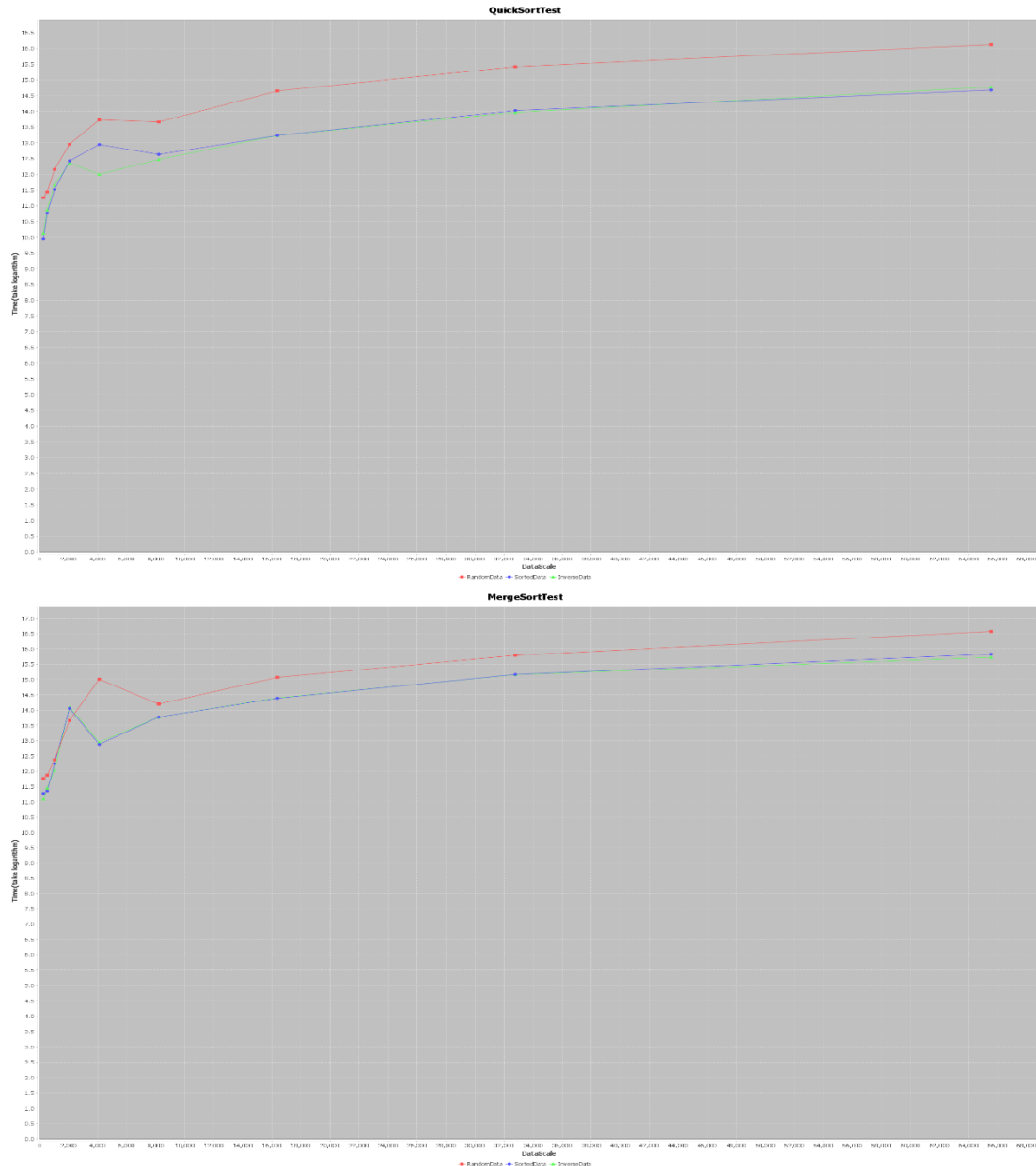
```

## 4.运行结果展示:

(红色曲线表示随机序列, 蓝色曲线表示顺序序列, 绿色曲线表示逆序序列)  
(运行时间进行了取对数操作)







## 5.总结与收获：

总结：基本上随着数据规模扩大，五种排序的用时都在增加

1) 对于相同算法：

总体上插入排序和选择排序的测试时长随数据规模变化要快于其他三种排序算法（除了对顺序序列进行插入排序）

①插入排序：时长随数据规模增长一直较快，且对于顺序序列的测试时长最优，远小于另外两种测试时长，对于逆序序列的测试时长最劣

②选择排序：时长随数据规模增长一直较快，对于随机序列的测试时长最劣，另外两者没有极为显著的差异

③希尔排序：时长随数据规模增长相对较快，对于随机序列的测试时长最劣，对顺序序列的测试时长最优；

④快速排序：时长随数据规模增长较为稳定，对于随机序列的测试时长最劣，另外两种测试结果没有显著差异，且在不同数据规模下各有优劣

⑤归并排序：时长随数据规模增长较为稳定，对于随机序列的测试时长最劣，另外两种测试结果没有显著差异，且在不同数据规模下各有优劣

2) 对于相同数据规模和分布

①随机数据：插入排序与选择排序差异不大且劣于其他三种，其他三种排序算法差异不大，快速排序最佳

②顺序数据：插入排序始终最优，选择排序最劣，其他三种算法测试时间大致为：希尔排序<快速排序<归并排序

③逆序数据：插入排序于选择排序差异不大且插入排序更劣，其他三种排序算法差异不大，快速排序最佳

## 任务五

### 1.题目：数据分布对排序算法的影响

完成了任务3和任务4之后，现要求为GenerateData类型再增加一种数据序列的生成方法，该方法要求生成分布不均匀数据序列：1/2的数据是0，1/4的数据是1，1/8的数据是2和1/8的数据是3。对这种分布不均匀的数据完成如同任务4的运行测试，检查这样的数据序列对于排序算法的性能影响。要求同任务4。（此时，可以将任务4、任务5的运行测试结果做一个纵向比较，用以理解数据序列分布的不同对同一算法性能的影响，如果能从每个排序算法的过程去深入分析理解则更好。

### 2.数据与算法设计：

同任务四一致，只是更换了数据序列，为GenerateData类增添了三种新方法，即生成随机、顺序、逆序的满足题意的序列，再更换生成图表类中的X轴数据即可。

### 3.部分代码说明：

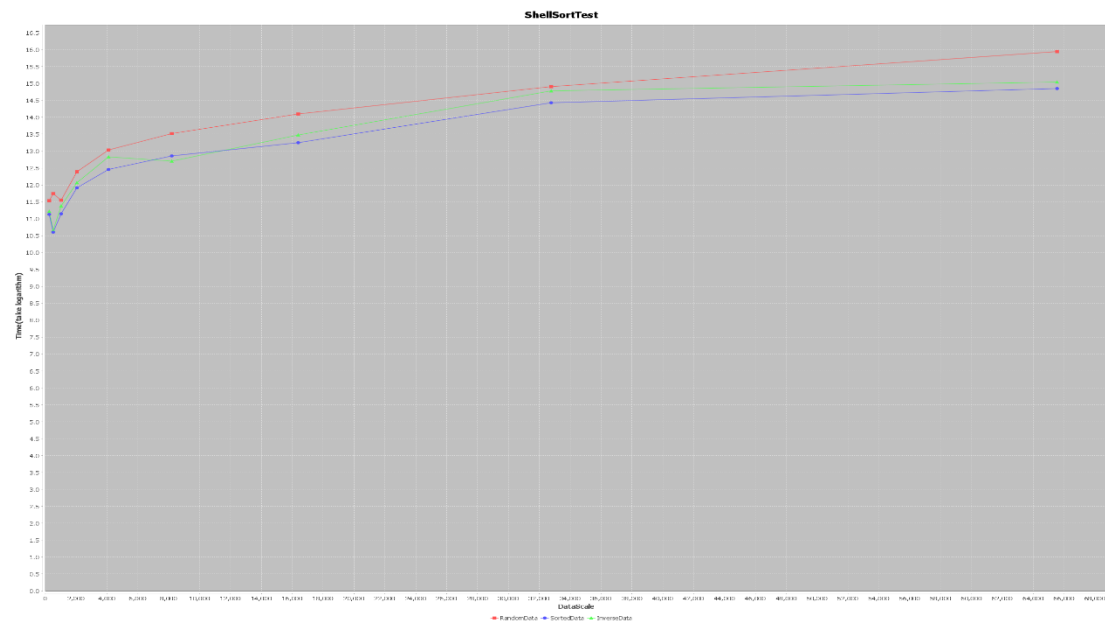
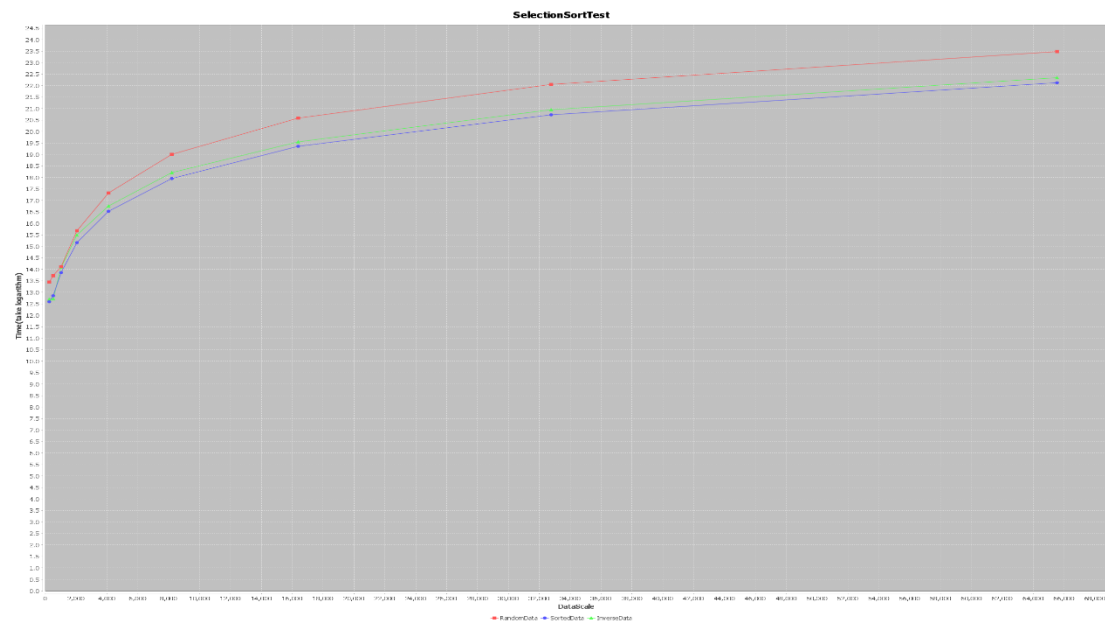
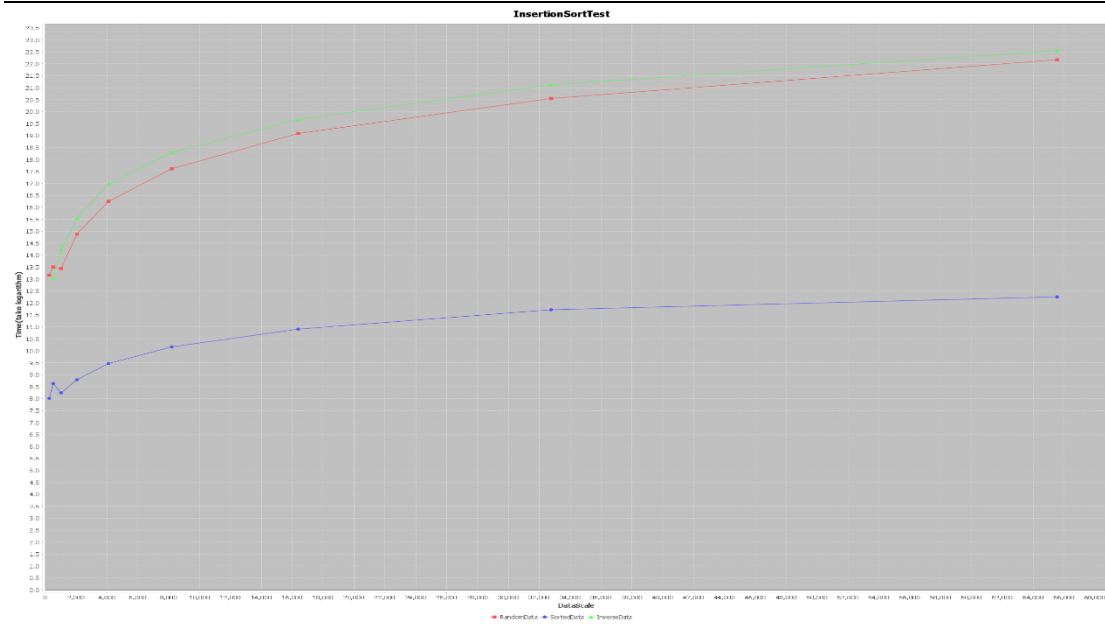
```
1. public static Double[] getAnotherRandomData(int N){
2.     //打乱有序数据即可
3.     Double[] numbers = getAnotherSortedData(N);
4.     shuffle(numbers, 0, numbers.length);
5.     return numbers;
6. }
7. public static Double[] getAnotherSortedData(int N){
8.     //顺序序列
9.     Double[] numbers = new Double[N];
10.    int i=0;
11.    //按顺序将相应数据确定下来
12.    while(i<(N/2)){
13.        numbers[i]=0.0;
14.        i++;
15.    }
16.    while(i<(3*N/4)){
```

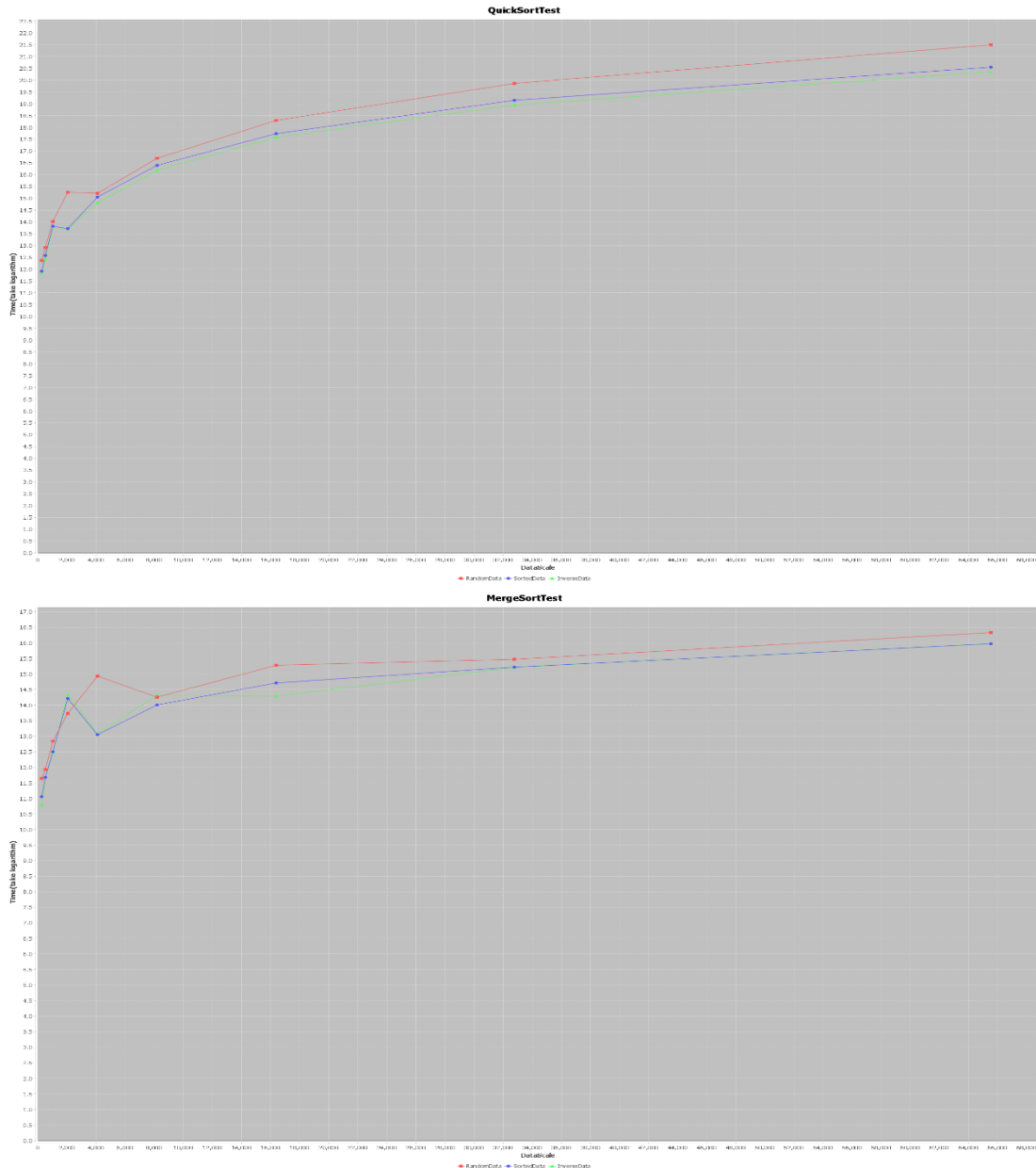
```

17.         numbers[i]=1.0;
18.         i++;
19.     }
20.     while(i<(7*N/8)){
21.         numbers[i]=2.0;
22.         i++;
23.     }
24.     while (i<N){
25.         numbers[i]=3.0;
26.         i++;
27.     }
28.     return numbers;
29. }
30. public static Double[] getAnotherInversedData(int N){
31.     // 逆序序列
32.     Double[] numbers = new Double[N];
33.     int i=0;
34.     // 按逆序将指定数据确定下来
35.     while (i<(N/8)){
36.         numbers[i]=3.0;
37.         i++;
38.     }
39.     while(i<(N/4)){
40.         numbers[i]=2.0;
41.         i++;
42.     }
43.     while(i<(N/2)){
44.         numbers[i]=1.0;
45.         i++;
46.     }
47.     while(i<N){
48.         numbers[i]=0.0;
49.         i++;
50.     }
51.     return numbers;
52. }

```

#### 4.运行结果展示：





## 5.总结与收获：

总结：基本上随着数据规模扩大，五种排序的用时都在增加

1) 对于相同算法：

总体上插入排序和选择排序的测试时长随数据规模变化要快于其他三种排序算法（除了对顺序序列进行插入排序）

①插入排序：时长随数据规模增长一直较快，且对于顺序序列的测试时长最优，远小于另外两种测试时长，对于逆序序列的测试时长最劣

②选择排序：时长随数据规模增长一直较快，对于随机序列的测试时长最劣，顺序序列欲逆序序列差异没有很显著，顺序序列相对更优

③希尔排序：时长随数据规模增长逐渐变快，对于随机序列的测试时长最劣，对顺序序列的测试时长最优；

④快速排序：时长随数据规模增长较为稳定，对于随机序列的测试时长最劣，另外两种测试结果差异不是很显著，逆序序列相对更优

⑤归并排序：时长随数据规模增长较为稳定，对于随机序列的测试时长最劣，另外两种测试结果没有显著差异，且在不同数据规模下各有优劣

2) 对于相同数据规模和分布

①随机数据：插入排序与选择排序差异不大，插入排序略优与选择排序，且显著劣于其他三种，其他三种排序算法测试时间大致为：希尔排序<归并排序<快速排序（按照时长从小到大）

②顺序数据：插入排序始终最优，选择排序始终最劣，其他三种算法测试时间大致为：希尔排序<归并排序<快速排序（按照时长从小到大）

③逆序数据：插入排序与选择排序差异不大，其他三种排序算法测试时间大致为：希尔排序<归并排序<快速排序（按照时长从小到大）

3) 与任务四纵向比对

①插入排序：更换后的分布不均匀数据序列使得相同数据规模下测试时长略微增大，因为重复数据的插入会更加快速

②选择排序：更换后的分布不均匀数据序列使得相同数据规模下测试时长略微增大，变化不大

③希尔排序：更换后的分布不均匀数据序列使得相同数据规模下测试时长略微减小，变化不大

④快速排序：更换后的分布不均匀数据序列使得相同数据规模下测试时长显著增大，因为过多的重复元素使得快速排序在轴值划分时容易令一边子序列的长度很小，使得划分次数增多。

⑤归并排序：数据序列变化影响不大，但仍使得相同规模数据下测试时长减小，因为在合并阶段，重复数据的合并操作会更加快速。

## 任务六

### 1.题目：快速排序的再探讨和应用

快速排序算法被誉为 20 世纪科学和工程领域的十大算法之一。前面的任务只是对快速排序的初识，下面从几个方面再更深入了解它：

### 2.问题一

#### 2.1 题目：优化快速排序

当待排序的数据量小于某个阈值时将递归的快速排序调用改为直接插入 排序调用，按照这种策略的优化的快速排序算法参照任务 4 的要求进行测试，并与任务 4 中没有优化的快速排序算法的执行时间进行对比；

#### 2.2 数据与算法设计：

只需更改原本 QuickSort 类中主函数的部分逻辑，当左边界离右边界的距离小过 5 时，切换为插入排序，直接调用 Insertion 即可，其他情况下仍保持原本快速排序结构。

修改生成图表的类，使其中 test 调用的算法为改进后的快速排序

#### 2.3.部分代码说明：



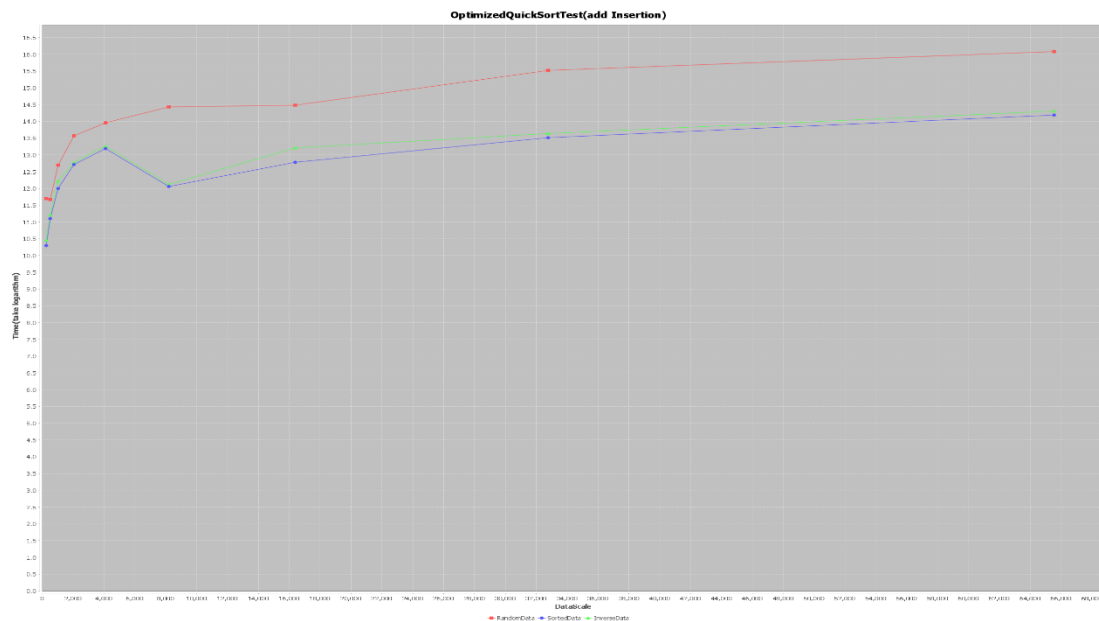
将原本的主函数

```
1. public void sort(Comparable[] objs,int left,int right){
2.     if(left<right)
3.         //当序列长度大于1 时进行划分
4.         {
5.             //获取轴值位置
6.             int pivotIndex=partition(objs,left,right);
7.             //递归左子序列
8.             sort(objs,left,pivotIndex-1);
9.             //递归右子序列
10.            sort(objs,pivotIndex+1,right);
11.        }
12.    }
```

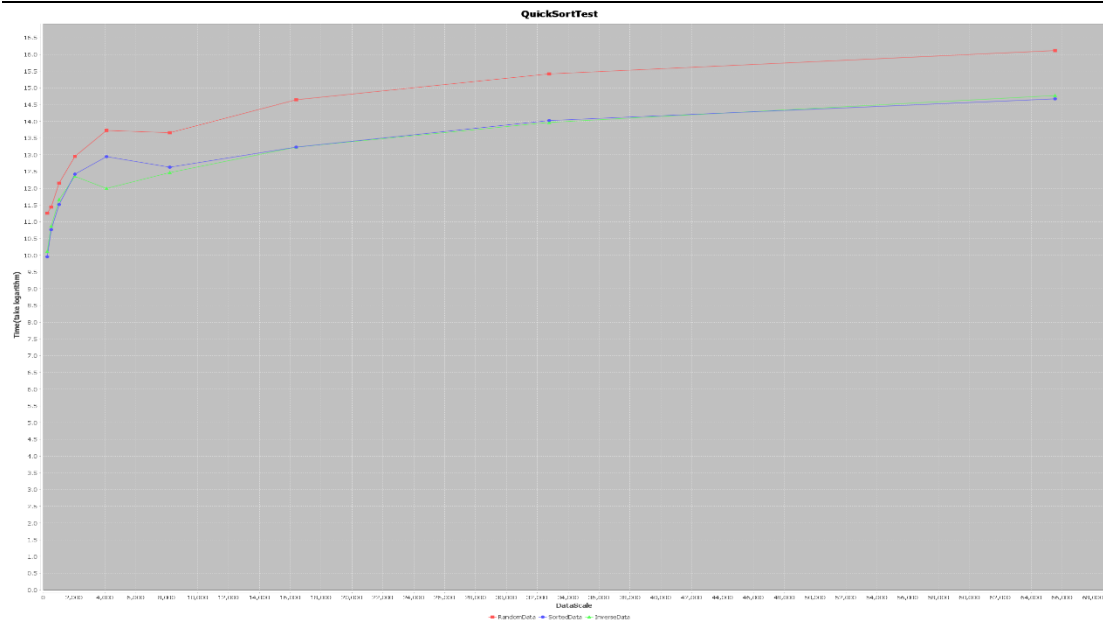
改为

```
1. SortAlgorithm tmp=new Insertion();
2. public void sort(Comparable[] objs,int left,int right){
3.     if(right<=left+5){//确定阈值为5，当数据量小于5 时，调用插入排序
4.         tmp.sort(objs,left,right);
5.         return;}
6.     int pivotIndex=partition(objs,left,right);
7.     sort(objs,left,pivotIndex-1);
8.     sort(objs,pivotIndex+1,right);
9. }
```

2.4.运行结果展示：



与未优化的快速排序比对



同时测得具体数据：（带\*为优化后快排得到数据）

当数据规模为2的8次方时:

随机数据序列测试结果: 186950.0

\*随机数据序列测试结果: 141320.0

顺序数据序列测试结果: 38300.0

\*顺序数据序列测试结果: 22970.0

逆序数据序列测试结果: 46420.0

\*逆序数据序列测试结果: 27860.0

////////////////////////////////////

////////////////////////////////////

当数据规模为2的9次方时:

随机数据序列测试结果: 139630.0

\*随机数据序列测试结果: 120640.0

顺序数据序列测试结果: 75540.0

\*顺序数据序列测试结果: 53790.0

逆序数据序列测试结果: 75980.0

\*逆序数据序列测试结果: 69760.0

////////////////////////////////////

////////////////////////////////////

当数据规模为2的10次方时:

随机数据序列测试结果: 312840.0

\*随机数据序列测试结果: 174320.0

顺序数据序列测试结果: 101380.0

\*顺序数据序列测试结果: 99780.0

逆序数据序列测试结果: 121470.0

\*逆序数据序列测试结果: 124860.0

////////////////////////////////////

////////////////////////////////////

```

当数据规模为2的11次方时:
随机数据序列测试结果: 610130.0
*随机数据序列测试结果: 363850.0
顺序数据序列测试结果: 210300.0
*顺序数据序列测试结果: 191550.0
逆序数据序列测试结果: 663970.0
*逆序数据序列测试结果: 215950.0
////////////////////////////////////
////////////////////////////////////
当数据规模为2的12次方时:
随机数据序列测试结果: 2132320.0
*随机数据序列测试结果: 416340.0
顺序数据序列测试结果: 107940.0
*顺序数据序列测试结果: 91660.0
逆序数据序列测试结果: 133430.0
*逆序数据序列测试结果: 106660.0
////////////////////////////////////
////////////////////////////////////
当数据规模为2的13次方时:
随机数据序列测试结果: 1067610.0
*随机数据序列测试结果: 1008210.0
顺序数据序列测试结果: 402070.0
*顺序数据序列测试结果: 333050.0
逆序数据序列测试结果: 426390.0
*逆序数据序列测试结果: 293420.0
////////////////////////////////////
////////////////////////////////////

```

```

当数据规模为2的14次方时：
随机数据序列测试结果：2357700.0
*随机数据序列测试结果：2189310.0
顺序数据序列测试结果：554240.0
*顺序数据序列测试结果：469670.0
逆序数据序列测试结果：667890.0
*逆序数据序列测试结果：598130.0
////////////////////////////////////
////////////////////////////////////
当数据规模为2的15次方时：
随机数据序列测试结果：5226230.0
*随机数据序列测试结果：4522860.0
顺序数据序列测试结果：965900.0
*顺序数据序列测试结果：915240.0
逆序数据序列测试结果：1180320.0
*逆序数据序列测试结果：966730.0
////////////////////////////////////
////////////////////////////////////
当数据规模为2的16次方时：
随机数据序列测试结果：1.04435E7
*随机数据序列测试结果：1.122132E7
顺序数据序列测试结果：2134430.0
*顺序数据序列测试结果：1950480.0
逆序数据序列测试结果：2386700.0
*逆序数据序列测试结果：2140230.0
////////////////////////////////////

```

## 2.5.总结与收获：

根据数据可以直观得出：优化后的快速排序始终比原排序用时更少，而在数据规模变得很大之后，优化的效果相对不够明显

## 3.问题二

### 3.1 题目：

在实际应用中经常会出现含有大量重复元素的数组，例如可能需要将大量人员资料按照生日排序，或者按照性别排序。给出使用①中完成的快速排序在数据规模为  $2^{16}$  的情况下，数据的重复率分别为 50%、60%、80%和 100%的运行时间的变化趋势图。结合①中的运行数据，给出观察结果

### 3.2.算法设计

需要完成的任务有：

- 1) 在 GenerateData 类中编写新方法，使得其能生成满足题意要求的数据序列，在此采用 random 类，先生成一个随机数，令序列中满足题意规模的数据等于该随机值，其余值随机设定，之后打乱该序列
- 2) 在生成图表的类中需要修改 X 轴值，使其代表数据的重复率

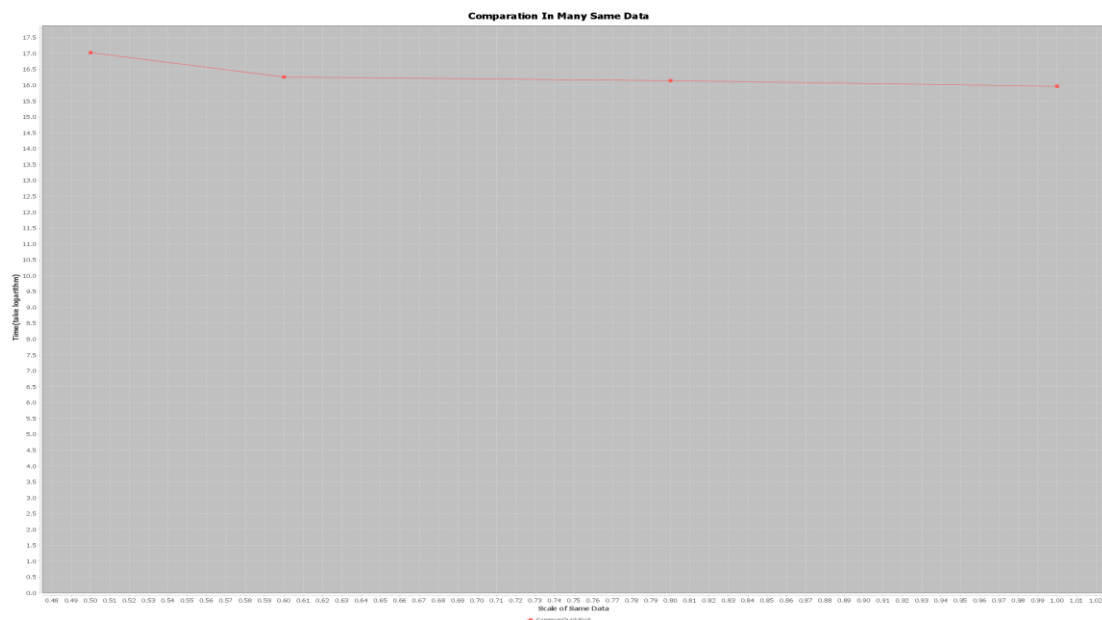
### 3.3.部分代码说明

```

1. public static Double[] getfiftypercentSameData(int N){
2.     Double[] numbers=new Double[N];
3.     Double samedata=random.nextDouble()*N;
4.     //随机确定重复数据的值
5.     for(int i=0;i<N;i++){
6.         if(i<(N/2)){
7.             //使要求规模下的数据为重复值
8.             numbers[i]=samedata;}
9.         else {
10.            //为剩余数据赋随机值
11.            numbers[i]=random.nextDouble()*N;}
12.    }
13.    //打乱序列
14.    shuffle(numbers,0,N);
15.    return numbers;
16. }

```

### 3.4 运行结果展示



### 3.5 总结

发现在数据重复率为 0.5, 0.6, 0.8, 1 的情况下，优化后快速排序算法的测试时间始终小于对随机序列的测试时间，且测试时间随着数据重复率的增多而减小



## 4.问题三

### 4.1.题目

当遇到②中的重复性很高的数据序列时，快速排序还有很大的改进空间，方法是改进快速排序的划分方法，即将原有的二路划分（划分为比轴值大和不小于轴值）变成三路划分（划分为比轴值小，等于轴值和比轴值大）。三路划分的思路来自于经典的荷兰国旗问题。现要求自行查找三路划分的逻辑并实现之（高效的三路划分算法可以参考 J.Bentley 和 D.McIlroy 的实现）。另外，用新的划分算法实现的快速排序重新对②完成实验并比较

### 4.2.数据与算法设计

1) 三路划分法改变了原本 QuickSort 中的 sort 主函数，一种设计思想如下。它从左到右遍历数组一次，维护一个指针 lt 使得  $a[lo..lt-1]$  中的元素都小于  $v$  ( $v$  为轴值)，一个指针 gt 使得  $a[gt+1..hi]$  中的元素都大于  $v$ ，一个指针 i 使得  $a[lt..i-1]$  中的元素都等于  $v$ ， $a[i..gt]$  中的元素都还未确定，如图 2.3.4 所示。一开始 i 和 lo 相等，我们使用 Comparable 接口（而非 less()）对  $a[i]$  进行三向比较来直接处理以下情况：

$a[i]$  小于  $v$ ，将  $a[lt]$  和  $a[i]$  交换，将 lt 和 i 加一；

$a[i]$  大于  $v$ ，将  $a[gt]$  和  $a[i]$  交换，将 gt 减一；

$a[i]$  等于  $v$ ，将 i 加一。

这些操作都会保证数组元素不变且缩小  $gt-i$  的值（这样循环才会结束）。另外，除非和切分元素相等，其他元素都会被交换。

2) 改变生成图标的类，使其算法指向实现后的三路划分快速排序算法

### 4.3.部分代码说明

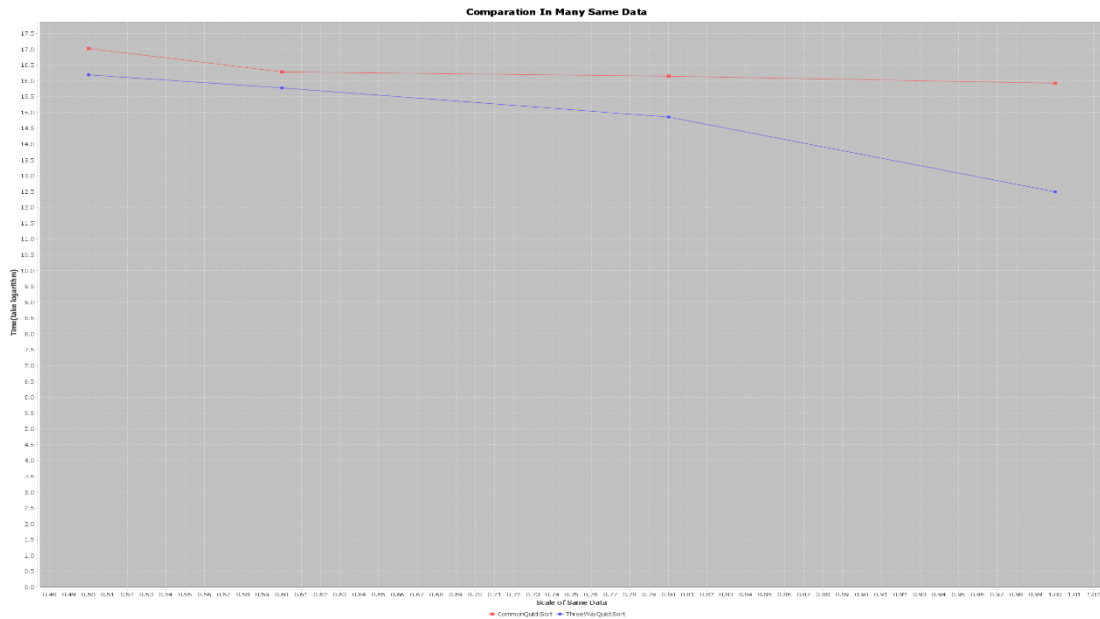
```
1. public void sort(Comparable[] objs,int left,int right){
2.     if(left<right){
3.         int lt=left,i=left+1,gt=right;//维护三个指针
4.         exchange(objs,findMiddle(objs,left,right),left)
5.         ;
6.         Comparable v=objs[left];
7.         while(i<=gt){
8.             int cmp=objs[i].compareTo(v);
9.             if(cmp<0){
10.                // gt 使得 a[gt+1..hi]中的元素都大于 v
11.                exchange(objs,lt++,i++);
12.            }
13.            //lt 使得 a[lo..lt-1]中的元素都小于 v
14.            else if(cmp>0){
15.                exchange(objs,i,gt--);
16.            }
17.            // i 使得 a[lt..i-1]中的元素都等于 v
18.            else{
19.                i++;
20.            }
21.        }
22.    }
23. }
```

```

20.          }
21.          sort(objs,left,lt-1);
22.          sort(objs,gt+1,right);
23.          }
24.      }
    
```

#### 4.4.运行结果展示

(红色曲线为原快排，蓝色曲线为三路划分快排)



#### 4.4.总结

根据图像可以直观看出，随着重复率变化，测试时长的变化趋势并没有改变，但是三路划分快速排序始终要快于原快速排序，且数据重复率越高，优化效果更明显，可以说明三路划分快速排序对于高重复率的序列有较好的应用效果

### 5.问题四

#### 5.1.题目：

在  $N$  个数据中找第  $k$  小元素 ( $1 \leq k \leq N$ ) 的问题可以有若干方法，请给出你能想到的方法，并简要分析每个方法的时间复杂度。在若干方法中，可以利用快速排序思想高效实现，请尽量独立思考，并最终给出设计思想、具体实现、测试以及时间复杂度分析

#### 5.2.多种方法

##### 1) 排序后取第 $k$ 小的元素：

方法：将  $N$  个数据进行排序，然后直接取第  $k$  个元素即为第  $k$  小的元素。

时间复杂度：对  $N$  个元素进行排序的时间复杂度取决于采用的算法，本次实验中的五种排序算法时间复杂度为  $O(N^2)$  和  $O(N \log N)$ ，然后取第  $k$  个元素的时间复杂度为  $O(1)$ ，因此总的时间复杂度根据选择排序算法的不同分别为  $O(N^2)$  或  $O(N \log N)$ 。

##### 2) 分治法：

方法：类似归并排序的思想，将  $N$  个数据分成两部分，然后分别在两部分中查找第  $k$  小的元素，最后根据比较结果选择在左边还是右边进行继续查找。

时间复杂度：平均情况下的时间复杂度为  $O(N\log N)$ 。

### 3) 快速选择算法：

方法：类似快速排序的思想，选择一个 pivot 元素，然后将数据分为两部分，一部分小于 pivot，一部分大于 pivot。根据 pivot 的位置，可以确定第 k 小的元素在左边还是右边，然后继续在对应的部分中进行查找。

时间复杂度：见 5.3.4

## 5.3.快速排序思想解决方法

### 5.3.1 设计思想

1) 选择一个基准元素 (pivot)，并将序列中的其他元素按照与基准元素的大小关系分为两部分：小于基准元素的部分和大于基准元素的部分。在这一步之后，基准元素所在的位置将是它在排好序的序列中的位置。

2) 如果基准元素的位置恰好是第 K 小的位置，那么基准元素就是我们要找的第 K 小的元素，直接返回基准元素。

3) 如果基准元素的位置大于第 K 小的位置，那么我们只需要在基准元素的左侧部分（小于基准元素的部分）继续查找第 K 小的元素。

4) 如果基准元素的位置小于第 K 小的位置，那么我们只需要在基准元素的右侧部分（大于基准元素的部分）继续查找第 K 小的元素。

### 5.3.2 具体实现

```
1. public class QuickSelect {
2.     // 查找第k小的元素
3.     public static Comparable findKthSmallest(Comparable[] arr, int k) {
4.         if (k < 1 || k > arr.length) {
5.             System.out.println("k 超出了序列范围");
6.         }
7.         return quickSelect(arr, 0, arr.length - 1, k - 1);
8.         // 调用quickSelect方法查找第k小的元素
9.     }
10.    // 快速选择算法
11.    private static Comparable quickSelect(Comparable[] arr,
12.        int left, int right, int k) {
13.        if (left == right) {
14.            return arr[left]; // 如果左右边界相等，返回当前元素
15.        }
16.        int pivotIndex = partition(arr, left, right); // 获取基准元素的索引
17.        if (k < pivotIndex) {
18.            return quickSelect(arr, left, pivotIndex - 1, k); // 在左侧子数组中递归查找第k小的元素
19.        } else if (k > pivotIndex) {
20.            return quickSelect(arr, pivotIndex + 1, right, k); // 在右侧子数组中递归查找第k小的元素
21.        }
22.    }
23.}
```

```

20.         } else {
21.             return arr[k]; // 如果 k 等于基准元素的索引, 返回基
                准元素
22.         }
23.     }
24.
25.     // 分区操作, 用于确定基准元素的位置
26.     private static int partition(Comparable[] arr, int left
        , int right) {
27.         int i = left;
28.         int j = right + 1;
29.         Comparable pivot = arr[left]; // 选择左边界元素作为
                基准元素
30.         while (true) {
31.             while (less(arr[++i], pivot)) { // 从左往右找到第
                一个大于等于基准元素的元素
32.                 if (i == right) {
33.                     break;
34.                 }
35.             }
36.             while (less(pivot, arr[--j])) { // 从右往左找到第
                一个小于等于基准元素的元素
37.                 if (j == left) {
38.                     break;
39.                 }
40.             }
41.             if (i >= j) {
42.                 break; // 如果 i 大于等于 j, 跳出循环
43.             }
44.             exchange(arr, i, j); // 交换 i 和 j 位置的元素
45.         }
46.         exchange(arr, left, j); // 将基准元素放到正确的位置
47.         return j; // 返回基准元素的索引
48.     }
49.     // 比较元素大小
50.     private static boolean less(Comparable v, Comparable w)
        {
51.         return v.compareTo(w) < 0; // 如果 v 小于 w, 返回 true
52.     }
53.     // 交换元素位置
54.     private static void exchange(Comparable[] arr, int i, i
        nt j) {
55.         Comparable temp = arr[i];
56.         arr[i] = arr[j];
57.         arr[j] = temp; // 交换 i 和 j 位置的元素
    
```

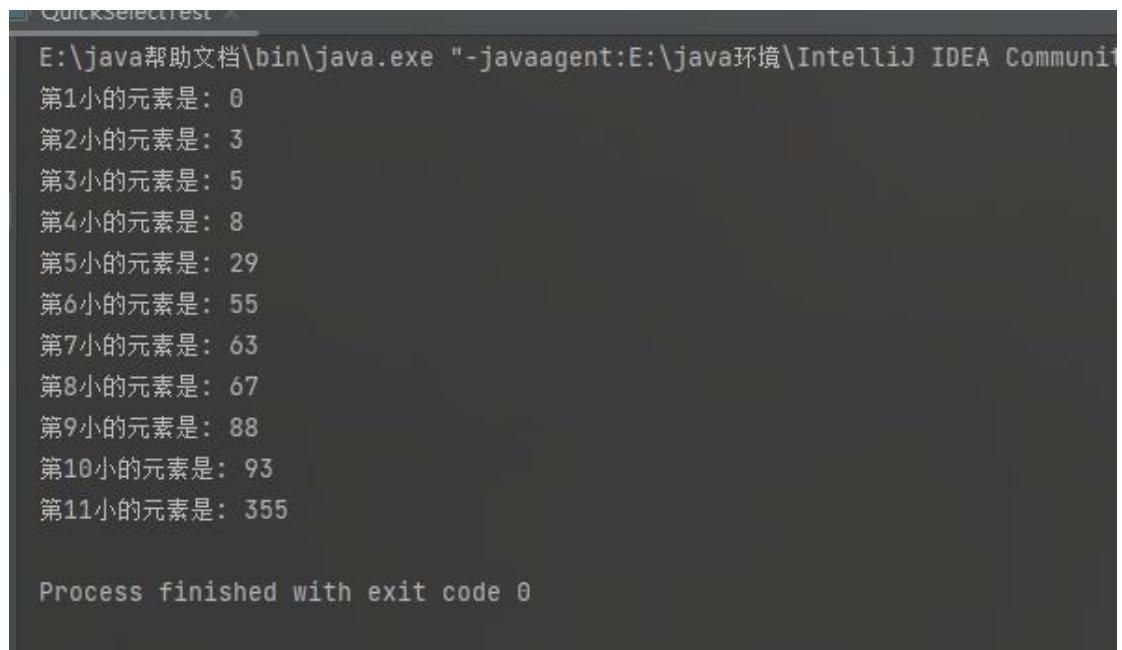
```
58.    }  
59. }  
60.
```

### 5.3.3 测试

编写测试类：

```
1. public class QuickSelectTest {  
2.     public static void main(String[] args) {  
3.         Integer[] arr = { 5, 67, 3, 355, 63, 29, 0, 8, 88,  
        55, 93}; // 定义一个整数数组  
4.         for (int k = 1; k <= arr.length; k++) {  
5.             Comparable result = QuickSelect.findKthSmallest  
                (arr, k); // 调用快速选择算法查找第 k 小的元素  
6.             System.out.println("第" + k + "小的元素  
        是: " + result); // 输出结果  
7.         }  
8.     }  
9. }
```

测试结果如下：



```
QuickSelectTest  
E:\java帮助文档\bin\java.exe "-javaagent:E:\java环境\IntelliJ IDEA Communit  
第1小的元素是: 0  
第2小的元素是: 3  
第3小的元素是: 5  
第4小的元素是: 8  
第5小的元素是: 29  
第6小的元素是: 55  
第7小的元素是: 63  
第8小的元素是: 67  
第9小的元素是: 88  
第10小的元素是: 93  
第11小的元素是: 355  
  
Process finished with exit code 0
```

### 5.3.4 时间复杂度分析

发现快速选择算法与快速排序算法十分相似，都需要首先完成划分过程，再对划分后的特定区域进行递归查找，划分的最好情形是每次都能划分为等长的两半，最坏情形则是所有数据都汇聚一端，假设用随取得的轴值总是在数组的中间，那每次都会将数组分成等长（或长度相差 1）的两半，再假设如果一直到最后子数组长度为 1 才找到第 k 大/小的值，那就一共要进行  $\lfloor \log n \rfloor + 1$  次 partition。

第一次 partition 要遍历的长度是  $n$ ，第二次要遍历的长度是  $n/2$ ，第三次要遍历的长度是  $n/4$ ……倒数第二次遍历的长度是 2，最后一次遍历的长度是 1。这是一个首项为 1，公比为 2 的等比数列，记  $ax = 2^{(x-1)}$ ，其中  $x$  为项数。这个等比数列的前  $x$  项和为  $2^x - 1$ 。因为一共要求  $\lfloor \log n \rfloor + 1$  次 partition， $2^{(\lfloor \log n \rfloor + 1)} - 1 \approx 2n - 1$ ，所以时间复杂度是  $O(2n - 1) = O(n)$ 。

## 附录

```

1. package practice2;
2.
3. import java.awt.*;
4. import java.io.IOException;
5.
6. public class LonggestLengthTest {
7.     public static Color BLUE=new Color(0,0,255);
8.     public static Color WHITE=new Color(255,255,255);
9.     public static void main(String[] args){
10.         String testpicture="E:\\java practice\\homeworkofdatast
            ruct\\src\\practice2\\tomography.png";
11.         int curlongest=0;//定义当前最大列长
12.         try {
13.             Picture testimage = new Picture(testpicture);//
                读入图片
14.             for(int i=0;i<testimage.getWidth();i++){
15.                 //遍历每一列
16.                 int curlength=getLength(i,testimage);
17.                 //获取当前列长
18.                 if(curlength>=curlongest){
19.                     curlongest=curlength;
20.                     //若当前列长更大,则更新 curlongest
21.                 }
22.             }
23.             System.out.println("图片的最长列长度为:
                "+curlongest);
24.         } catch (IOException e) {
25.             System.out.println("图片读取失败");
26.         }
27.     }
28.     public static int getLength(int i, Picture sample) {
29.         Color firstcolor = sample.getColor(i, 0);
30.         //获取首位值颜色
31.         int left = 0;
32.         int right = sample.getHeight() - 1;
33.         int pointer = (left + right) / 2;
34.         //定义头、中、尾指针
35.         while ((pointer < (sample.getHeight() - 1))&&sample
            .isSameAsNext(i, pointer))
36.             //开始二分,不断循环
37.             //终止条件: pointer 到达分界点或边界
38.             { //pointer 与下一个颜色相同时

```



```

39.         if (sample.isSame(i, 0, i, pointer))
40.             //如果 pointer 位置与首位置的颜色相同, 说明分界点在
               右半部分, 更新左指针
41.             {left = pointer + 1;}
42.         else{
43.             //反之说明分界点在左半部分, 更新右指针
44.             right = pointer - 1;
45.         }
46.         pointer = (left + right) / 2;
47.         //更新中间指针
48.     }
49.     if (pointer == (sample.getHeight() - 1)) {
50.         //根据首位值颜色判断返回长度
51.         if (firstcolor.equals(WHITE)) {
52.             return 0;
53.         } else {
54.             return sample.getHeight();
55.         }}
56.     if (firstcolor.equals(BLUE)) {
57.         return pointer + 1;
58.     }
59.     else {
60.         return sample.getHeight() - (pointer + 1);
61.     }
62. }
63. }
64.

```

Ctrl+M

```

1. package practice2;
2.
3. import javax.imageio.ImageIO;
4. import java.awt.*;
5. import java.awt.image.BufferedImage;
6. import java.io.File;
7. import java.io.IOException;
8.
9. public class Picture {
10.     private BufferedImage image;
11.     public int width;
12.     private int height;
13.     public Picture(String filename) throws IOException {
14.         File file = new File(filename);
15.         image = ImageIO.read(file);
16.         width = image.getWidth();

```

```

17.         height = image.getHeight();
18.     }
19.     public Picture(int width, int height){
20.         this.width = width;
21.         this.height = height;
22.         image = new BufferedImage(width, height, BufferedIm
            age.TYPE_INT_RGB);
23.     }
24.     public int getWidth(){
25.         return width;
26.     }
27.     public int getHeight(){
28.         return height;
29.     }
30.     public void setColor(int col, int row, Color c){
31.         image.setRGB(col, row, c.getRGB());
32.     }
33.     public Color getColor(int col, int row){
34.         int rgb = image.getRGB(col, row);
35.         return new Color(rgb);
36.     }
37.     public void save(String filename) throws IOException {
38.         String suffix = filename.substring(filename.lastInd
            exOf('.')+1);
39.         if (suffix.equalsIgnoreCase("png") || suffix.equals
            IgnoreCase("jpg"))
40.             ImageIO.write(image, suffix, new File(filename)
            );
41.     }
42.     public void darker(){
43.         for (int i = 0; i < width; i++)
44.             for (int j = 0; j < height; j++){
45.                 Color c = getColor(i, j);
46.                 setColor(i, j, c.darker());
47.             }
48.     }
49.     public boolean isSameAsNext(int x,int y){
50.         if(getColor(x,y).equals(getColor(x,y+1)))
51.             return true;
52.         else
53.             return false;
54.     }
55.     public boolean isSame(int x1,int y1,int x2,int y2){
56.         if(getColor(x1,y1).equals(getColor(x2,y2)))
57.             return true;

```

```
58.         else return false;
59.     }
60.}
61.
```

Ctrl+M

```
1. package practice3;
2. import java.util.Random;
3.
4. public class GenerateData {
5.     private static Random random = new Random();
6.
7.     public static Double[] getfiftypercentSameData(int N){
8.         Double[] numbers=new Double[N];
9.         Double samedata=random.nextDouble()*N;
10.        //随机确定重复数据的值
11.        for(int i=0;i<N;i++){
12.            if(i<(N/2)){
13.                //使要求规模下的数据为重复值
14.                numbers[i]=samedata;}
15.            else {
16.                //为剩余数据赋随机值
17.                numbers[i]=random.nextDouble()*N;}
18.        }
19.        //打乱序列
20.        shuffle(numbers,0,N);
21.        return numbers;
22.    }
23.    public static Double[] getsixtypercentSameData(int N){
24.        Double[] numbers=new Double[N];
25.        Double samedata=random.nextDouble()*N;
26.        for(int i=0;i<N;i++){
27.            if(i<(N*0.6)){
28.                numbers[i]=samedata;}
29.            else {
30.                numbers[i]=random.nextDouble()*N;}
31.        }
32.        shuffle(numbers,0,N);
33.        return numbers;
34.    }
35.    public static Double[] geteightypercentSameData(int N){
36.        Double[] numbers=new Double[N];
37.        Double samedata=random.nextDouble()*N;
38.        for(int i=0;i<N;i++){
39.            if(i<(N*0.8)){
```

```

40.         numbers[i]=samedata;}
41.     else {
42.         numbers[i]=random.nextDouble()*N;}
43.     }
44.     shuffle(numbers,0,N);
45.     return numbers;
46. }
47. public static Double[] getAllSameData(int N){
48.     Double[] numbers=new Double[N];
49.     Double samedata=random.nextDouble()*N;
50.     for(int i=0;i<N;i++){
51.         numbers[i]=samedata;}
52.     return numbers;
53. }
54. // 生成一个长度为N 的均匀分布的数据序列
55. public static Double[] getRandomData(int N){
56.     Double[] numbers = getSortedData(N);
57.     shuffle(numbers, 0, numbers.length);
58.     return numbers;
59. }
60. public static Double[] getAnotherRandomData(int N){
61.     //打乱有序数据即可
62.     Double[] numbers = getAnotherSortedData(N);
63.     shuffle(numbers, 0, numbers.length);
64.     return numbers;
65. }
66. // 生成一个长度为N 的正序的数据序列
67. public static Double[] getSortedData(int N){
68.     Double[] numbers = new Double[N];
69.     double t = 0.0;
70.     for (int i = 0; i < N; i++){
71.         numbers[i] = t;
72.         t = t + 1.0/N;
73.     }
74.     return numbers;
75. }
76. public static Double[] getAnotherSortedData(int N){
77.     //顺序序列
78.     Double[] numbers = new Double[N];
79.     int i=0;
80.     //按顺序将相应数据确定下来
81.     while(i<(N/2)){
82.         numbers[i]=0.0;
83.         i++;
84.     }

```

```

85.         while(i<(3*N/4)){
86.             numbers[i]=1.0;
87.             i++;
88.         }
89.         while(i<(7*N/8)){
90.             numbers[i]=2.0;
91.             i++;
92.         }
93.         while (i<N){
94.             numbers[i]=3.0;
95.             i++;
96.         }
97.         return numbers;
98.     }
99.     // 生成一个长度为N 的逆序的数据序列
100.    public static Double[] getInversedData(int N){
101.        Double[] numbers = new Double[N];
102.        double t = 1.0;
103.        for (int i = 0; i < N; i++){
104.            t = t - 1.0/N;
105.            numbers[i] = t;
106.        }
107.        return numbers;
108.    }
109.    public static Double[] getAnotherInversedData(int N)
110.    {
111.        // 逆序序列
112.        Double[] numbers = new Double[N];
113.        int i=0;
114.        // 按逆序将指定数据确定下来
115.        while (i<(N/8)){
116.            numbers[i]=3.0;
117.            i++;
118.        }
119.        while(i<(N/4)){
120.            numbers[i]=2.0;
121.            i++;
122.        }
123.        while(i<(N/2)){
124.            numbers[i]=1.0;
125.            i++;
126.        }
127.        while(i<N){
128.            numbers[i]=0.0;
129.            i++;

```

```

129.         }
130.
131.
132.         return numbers;
133.     }
134.     // 将数组 numbers 中的[left,right) 范围内的数据随机打乱
135.     private static void shuffle(Double[] numbers, int left, int right){
136.         int N = right - left;
137.         Random rand = new Random();
138.         for(int i = 0; i < N; i++){
139.             int j = i + rand.nextInt(N-i);
140.             exchange(numbers, i+left, j+left);
141.         }
142.     }
143.     private static void exchange(Double[] numbers, int i, int j){
144.         double temp = numbers[i];
145.         numbers[i] = numbers[j];
146.         numbers[j] = temp;
147.     }
148.
149.     public static void main(String[] args) {
150.         Double[] numbers = getRandomData(1000);
151.         for(int i = 0; i < 100; i++)
152.             System.out.printf("%5.3f ", numbers[i]);
153.     }
154. }
155.

```

Ctrl+M

```

1. package practice3;
2. public class Insertion extends SortAlgorithm {
3.     public void sort(Comparable[] objs){
4.         int N = objs.length;
5.         sort(objs,0,N-1);
6.     }
7.     public void sort(Comparable[] objs,int left,int right){
8.         for(int i = left+1; i <= right; i++){
9.             for(int j = i; j > left && less(objs[j], objs[j-1]); j--){
10.                 exchange(objs, j, j-1);
11.             }
12.         }
13.     }

```

```

14. }
15. package practice3;
16. public class MergeSort extends SortAlgorithm{
17.     public void sort(Comparable[] objs){
18.         int left=0,right=objs.length-1;
19.         sort(objs,left,right);
20.     }
21.     public void sort(Comparable[] objs,int left,int right){
22.         if(left>=right)
23.             return;
24.         else{
25.             int mid=(left+right)/2;
26.             //划分左子序列
27.             sort(objs,left,mid);
28.             //划分右子序列
29.             sort(objs,mid+1,right);
30.             //归并
31.             merge(objs,left,mid,right);
32.         }
33.     }
34.     public void merge(Comparable[] objs,int left,int mid,int
        t right){
35.         //采用双指针
36.         int frontPoint=left;
37.         int behindPoint=mid+1;
38.         //开辟新数组用于中转
39.         Comparable[] tmp=new Comparable[right-left+1];
40.         int k=0;
41.         while((frontPoint<=mid)&&(behindPoint<=right)){
42.             //将两个指针指向的较小值存入新数组,同时更新指针
43.             if(less(objs[frontPoint],objs[behindPoint])){
44.                 tmp[k++]=objs[frontPoint++];
45.             }
46.             else{
47.                 tmp[k++]=objs[behindPoint++];
48.             }
49.         }
50.         //将剩余数据依次放入新数组
51.         while(frontPoint<=mid){
52.             tmp[k++]=objs[frontPoint++];
53.         }
54.         while(behindPoint<=right){
55.             tmp[k++]=objs[behindPoint++];
56.         }
57.         //完成拷贝

```



```

58.         for(int i=left;i<=right;i++){
59.             objs[i]=tmp[i-left];
60.         }
61.     }
62. package practice3;
63.
64. public class Quicksort extends SortAlgorithm{
65.     public void sort(Comparable[] objs){
66.         int left=0,right=objs.length-1;
67.         sort(objs,left,right);
68.     }
69.     public void sort(Comparable[] objs,int left,int right){
70.         if(left<right)
71.             //当序列长度大于1 时进行划分
72.             {
73.                 //获取轴值位置
74.                 int pivotIndex=partition(objs,left,right);
75.                 //递归左子序列
76.                 sort(objs,left,pivotIndex-1);
77.                 //递归右子序列
78.                 sort(objs,pivotIndex+1,right);
79.             }
80.     }
81.     public int partition(Comparable[] objs,int left,int right)
82.     {
83.         //按照轴值划分
84.         int index=findMiddle(objs,left,right);
85.         //取得轴值下标, 同时完成对首、中、尾位置的排序
86.         int low=left,high=right-1;
87.         exchange(objs,right,index);
88.         //将轴值先放在末尾
89.         Comparable pivot=objs[right];
90.         while(low<high){
91.             while(true){
92.                 //左游标寻找大于轴值的元素
93.                 if(less(pivot,objs[low])||(low>=high)){
94.                     break;
95.                 }
96.                 low++;
97.             }
98.             while(true){
99.                 //右游标寻找小于轴值的元素
100.                if(less(objs[high],pivot)|| (high<=low)){
101.                    break;

```

```

102.         }
103.         high--;
104.     }
105.     if(low<high){
106.         // 交换两游标指向元素
107.         exchange(objs,low,high);
108.     }
109. }
110. // 将轴值归位
111. exchange(objs,low,right);
112. // 返回轴值位置
113. return low;
114. }
115. /*int index=findMiddle(objs,left,right);
116. int low=left;
117. Comparable pivot=objs[index];
118. exchange(objs,left,index);
119.
120. while(left<right){
121.     while(true){
122.         if(less(objs[right],pivot)|| (left==right
123.     )){
124.         break;
125.     }
126.     right--;
127.     while(true){
128.         if(less(pivot,objs[left])|| (left==right
129.     )){
130.         break;
131.     }
132.     left++;
133. }
134.     exchange(objs,left,right);
135. }
136.     exchange(objs,left,low);
137.     return left;*/
138.
139. public int findMiddle(Comparable[] objs,int left,int
140.     right){
141.         //Comparable first=objs[left];
142.         Comparable second=objs[(left+right)/2];
143.         Comparable third=objs[right];

```

```

143.         if((less(first,second)&&less(second,third))||les
           s(third,second)&&less(second,first)){
144.             return (left+right)/2;
145.         }
146.         else if((less(second,third)&&less(third,first))|
           |(less(first,third)&&less(third,second))){
147.             return right;
148.         }
149.         else{
150.             return left;
151.         }*/
152.         int mid=(left+right)/2;
153.         if(less(objs[right],objs[left]))
154.             exchange(objs,right,left);
155.         if(less(objs[right],objs[mid]))
156.             exchange(objs,right,mid);
157.         if(less(objs[mid],objs[left]))
158.             exchange(objs,mid,left);
159.         return mid;
160.     }
161. }
162. }
163. package practice3;
164.
165. public class Selection extends SortAlgorithm{
166.     public void sort(Comparable[] objs){
167.         for(int i=0;i< objs.length-1;i++){
168.             int curmaxpoint=i;
169.             for(int j=i;j<objs.length;j++){
170.                 if(less(objs[j],objs[curmaxpoint])){
171.                     curmaxpoint=j;
172.                 }
173.             }
174.             exchange(objs,curmaxpoint,i);
175.         }
176.     }
177. }
178. package practice3;
179.
180. public class Shell extends SortAlgorithm{
181.     public void sort(Comparable[] objs){
182.         int N=objs.length;
183.         for(int i=N/3;i>0;i/=3) {
184.             for (int j = i; j < N; j++) {

```

```

185.         for (int k = j; (k >= i) && (less(objs[k]
           ], objs[k - i])); k -= i) {
186.             exchange(objs, k, k - i);
187.         }
188.     }
189. }
190. for(int j=1;j<N;j++) {
191.     for (int k = j; (k >= 1) && (less(objs[k], objs[
           k - 1])); k -= 1) {
192.         exchange(objs, k, k - 1);
193.     }
194. }
195. }
196. }
197.
198.
199.

```

Ctrl+M

```

1. package practice3;
2.
3. public abstract class SortAlgorithm {
4.     public abstract void sort(Comparable[] objs);
5.     public void sort(Comparable[] objs,int left,int right){
6.     }
7.     protected void exchange(Comparable[] numbers, int i, in
           t j){
8.         Comparable temp;
9.         temp = numbers[i];
10.        numbers[i] = numbers[j];
11.        numbers[j] = temp;
12.    }
13.    protected boolean less(Comparable one, Comparable other
           ){
14.        return one.compareTo(other) < 0;
15.    }
16.    protected void show(Comparable[] numbers){
17.        int N = numbers.length;
18.        int line = 0;
19.        for(int i = 0; i < N; i++){
20.            System.out.printf("%s ", numbers[i]);
21.            line++;
22.            if(line % 20 == 0) System.out.println();
23.        }
24.        System.out.println();

```

```

24.     }
25.     public boolean isSorted(Comparable[] numbers){
26.         int N = numbers.length;
27.         for(int i = 0; i < N-1; i++)
28.             if(numbers[i].compareTo(numbers[i+1]) > 0) return false;
29.         return true;
30.     }
31.
32. }
33.

```

Ctrl+M

```

1. package practice3;
2.
3. public class Test {
4.     public static void main(String[] args) {
5.         SortAlgorithm sortmethod1 = new Insertion();
6.         SortAlgorithm sortmethod2 = new Selection();
7.         SortAlgorithm sortmethod3 = new Shell();
8.         SortAlgorithm sortmethod4 = new Quicksort();
9.         SortAlgorithm sortmethod5 = new MergeSort();
10.        System.out.println("对数据规模为 20 的随机序列测试: ");
11.        Double[] testShell0 = GenerateData.getRandomData(20);
12.        Double[] testInsertion0 = GenerateData.getRandomData(20);
13.        Double[] testMerge0 = GenerateData.getRandomData(20);
14.        Double[] testQuick0 = GenerateData.getRandomData(20);
15.        Double[] testSelection0 = GenerateData.getRandomData(20);
16.        sortmethod1.sort(testInsertion0);
17.        sortmethod2.sort(testSelection0);
18.        sortmethod3.sort(testShell0);
19.        sortmethod4.sort(testQuick0);
20.        sortmethod5.sort(testMerge0);
21.        System.out.printf("随机数组 1 经插入排序后的结果为");
22.        sortmethod1.show(testInsertion0);
23.        System.out.printf("验证为: ");
24.        System.out.println(sortmethod1.isSorted(testInsertion0));
25.        System.out.printf("随机数组 2 经选择排序后的结果为: ");
26.        sortmethod1.show(testSelection0);

```

```

27.      System.out.printf("验证为: ");
28.      System.out.println(sortmethod2.isSorted(testSelecti
        on0));
29.      System.out.printf("随机数组 3 经希尔排序后的结果为: ");
30.      sortmethod1.show(testShell0);
31.      System.out.printf("验证为: ");
32.      System.out.println(sortmethod3.isSorted(testShell0)
        );
33.      System.out.printf("随机数组 4 经快速排序后的结果为: ");
34.      sortmethod1.show(testQuick0);
35.      System.out.printf("验证为: ");
36.      System.out.println(sortmethod4.isSorted(testQuick0)
        );
37.      System.out.printf("随机数组 5 经归并排序后的结果为: ");
38.      sortmethod1.show(testMerge0);
39.      System.out.printf("随机数组 5 经归并排序后的结果验证为:
        ");
40.      System.out.println(sortmethod2.isSorted(testMerge0)
        );
41.      System.out.println("对规模为 10000 的随机序列测试: ");
42.      Double[] testShell = GenerateData.getRandomData(100
        00);
43.      Double[] testInsertion = GenerateData.getRandomData
        (10000);
44.      Double[] testMerge = GenerateData.getRandomData(100
        00);
45.      Double[] testQuick = GenerateData.getRandomData(100
        00);
46.      Double[] testSelection = GenerateData.getRandomData
        (10000);
47.      sortmethod1.sort(testInsertion);
48.      sortmethod2.sort(testSelection);
49.      sortmethod3.sort(testShell);
50.      sortmethod4.sort(testQuick);
51.      sortmethod5.sort(testMerge);
52.      System.out.printf("随机数组 6 经插入排序后的结果验证为:
        ");
53.      System.out.println(sortmethod1.isSorted(testInserti
        on));
54.      System.out.printf("随机数组 7 经选择排序后的结果验证为:
        ");
55.      System.out.println(sortmethod2.isSorted(testSelecti
        on));
56.      System.out.printf("随机数组 8 经希尔排序后的结果验证为:
        ");

```

```

57.         System.out.println(sortmethod3.isSorted(testShell))
           ;
58.         System.out.printf("随机数组 9 经快速排序后的结果验证为:
           ");
59.         System.out.println(sortmethod4.isSorted(testQuick))
           ;
60.         System.out.printf("随机数组 10 经归并排序后的结果验证
           为: ");
61.         System.out.println(sortmethod2.isSorted(testMerge))
           ;
62.     }
63. }
64.

```

Ctrl+M

```

1. package practice4;
2.
3. import org.jfree.chart.ChartFactory;
4. import org.jfree.chart.ChartPanel;
5. import org.jfree.chart.JFreeChart;
6. import org.jfree.chart.plot.PlotOrientation;
7. import org.jfree.chart.plot.XYPlot;
8. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
9. import org.jfree.chart.ui.ApplicationFrame;
10. import org.jfree.chart.ui.RectangleInsets;
11. import org.jfree.data.xy.XYDataset;
12. import org.jfree.data.xy.XYSeries;
13. import org.jfree.data.xy.XYSeriesCollection;
14. import practice3.GenerateData;
15. import practice3.Insertion;
16.
17. import java.awt.*;
18.
19. import static java.lang.Math.pow;
20.
21. public class InsertionTestDemo extends ApplicationFrame {
22.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创建工作
23.     public InsertionTestDemo(String title) {
24.         super(title);
25.         XYDataset dataset = createDataset();           //
           创建记录图中坐标点的数据集
26.         JFreeChart chart = createChart(dataset);       //
           使用上一步已经创建好的数据集生成一个图表对象
27.         ChartPanel chartPanel = new ChartPanel(chart); //
           将上一步已经创建好的图表对象放置到一个可以显示的 Panel 上

```



```

28.      // 设置 GUI 面板 Panel 的显示大小
29.      chartPanel.setPreferredSize(new Dimension(500, 270)
30.      );
31.      setContentPane(chartPanel);           //
        这是 JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的视频
        中进行了讲解。
32.
33.      private JFreeChart createChart(XYDataset dataset) {
34.          // 使用已经创建好的 dataset 生成图表对象
35.          // JFreechart 提供了多种类型的图表对象，本次实验是需要使
        用 XYLine 型的图表对象
36.          JFreeChart chart = ChartFactory.createXYLineChart(
37.              "InsertionSortTest",          // 图表的标题
38.              "DataScale",                  // 横
        轴的标题名
39.              "Time(take logarithm)",
        // 纵轴的标题名
40.              dataset,                      // 图表对象中
        使用的数据集对象
41.              PlotOrientation.VERTICAL,      // 图表显示的
        方向
42.              true,                          // 是否显示图
        例
43.              false,                        // 是否需要生
        成 tooltips
44.              false                         // 是否需要生
        成 urls
45.      );
46.      // 下面所做的工作都是可选操作，主要是为了调整图表显示的
        风格
47.      // 同学们不必在意下面的代码
48.      // 可以将下面的代码去掉对比一下显示的不同效果
49.      chart.setBackgroundPaint(Color.WHITE);
50.      XYPlot plot = (XYPlot) chart.getPlot();
51.      plot.setBackgroundPaint(Color.lightGray);
52.      plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.
        0, 6.0));
53.      plot.setDomainGridlinePaint(Color.WHITE);
54.      plot.setRangeGridlinePaint(Color.WHITE);
55.      XYLineAndShapeRenderer renderer = (XYLineAndShapeRe
        nderer) plot.getRenderer();
56.      renderer.setDefaultShapesVisible(true);
57.      renderer.setDefaultShapesFilled(true);
58.      return chart;

```

```

59.     }
60.
61.
62.     private XYDataset createDataset() {
63.         // 本样例中想要显示的是三组数据的变化图
64.         // X 数组是三组数据共同拥有的 x 坐标值; Y1、Y2 和 Y3 数组
           分别存储了三组数据对应的 y 坐标值
65.         // 共有 9 个节点值
66.         int[] X = new int[9];
67.         // 建立 X 坐标
68.         double[] Y1 = new double[9];
69.         double[] Y2 = new double[9];
70.         double[] Y3 = new double[9];
71.         // 建立三种 Y 坐标分别对应三种数据序列
72.         for (int j = 8; j <= 16; j++) {
73.             int N = (int) pow(2, j);
74.             X[j - 8] = N;
75.             // X 为数据规模
76.             Double[] randomData = GenerateData.getAnotherRa
               ndomData(N);
77.             Y1[j - 8] = Math.log(SortTest.test(new Insertio
               n(), randomData, 10));
78.             Double[] sortedData = GenerateData.getAnotherSo
               rtedData(N);
79.             Y2[j - 8] = Math.log(SortTest.test(new Insertio
               n(), sortedData, 10));
80.             Double[] inverseData = GenerateData.getAnotherI
               nversedData(N);
81.             Y3[j - 8] = Math.log(SortTest.test(new Insertio
               n(), inverseData, 10));
82.             // Y 为该算法对不同种类数据序列测试 10 次用时的平均值
83.         }
84.         double[][] Y = {Y1, Y2, Y3};
85.         // jfreechart 中使用 XYSeries 对象存储一组数据的(x,y)的
           序列, 因为有三组数据所以创建三个 XYSeries 对象
86.         XYSeries[] series = {new XYSeries("RandomData"), ne
               w XYSeries("SortedData"), new XYSeries("InverseData")};
87.         int N = X.length;
88.         int M = series.length;
89.         for (int i = 0; i < M; i++)
90.             for (int j = 0; j < N; j++)
91.                 series[i].add(X[j], Y[i][j]);
92.         // 因为在该图表中显示的数据序列不止一组, 所以在
           jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对
           象中

```

```

93.         XYSeriesCollection dataset = new XYSeriesCollection
           ();
94.         for (int i = 0; i < M; i++)
95.             dataset.addSeries(series[i]);
96.         return dataset;
97.     }
98. }
99.

```

Ctrl+M

```

1. package practice4;
2.
3. import org.jfree.chart.ChartFactory;
4. import org.jfree.chart.ChartPanel;
5. import org.jfree.chart.JFreeChart;
6. import org.jfree.chart.plot.PlotOrientation;
7. import org.jfree.chart.plot.XYPlot;
8. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
9. import org.jfree.chart.ui.ApplicationFrame;
10. import org.jfree.chart.ui.RectangleInsets;
11. import org.jfree.data.xy.XYDataset;
12. import org.jfree.data.xy.XYSeries;
13. import org.jfree.data.xy.XYSeriesCollection;
14. import practice3.GenerateData;
15. import practice3.Insertion;
16. import practice3.MergeSort;
17.
18. import java.awt.*;
19.
20. import static java.lang.Math.pow;
21.
22. public class MergeTestDemo extends ApplicationFrame {
23.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创建工作
24.     public MergeTestDemo(String title) {
25.         super(title);
26.         XYDataset dataset = createDataset();           //
           创建记录图中坐标点的数据集
27.         JFreeChart chart = createChart(dataset);       //
           使用上一步已经创建好的数据集生成一个图表对象
28.         ChartPanel chartPanel = new ChartPanel(chart); //
           将上一步已经创建好的图表对象放置到一个可以显示的Panel上
29.         // 设置GUI面板Panel的显示大小
30.         chartPanel.setPreferredSize(new Dimension(500, 270)
           );

```

```

31.         setContentPane(chartPanel);                                //
           这是 JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的视频
           中进行了讲解。
32.     }
33.
34.     private JFreeChart createChart(XYDataset dataset) {
35.         // 使用已经创建好的 dataset 生成图表对象
36.         // JFreechart 提供了多种类型的图表对象，本次实验是需要使用
           用 XYLine 型的图表对象
37.         JFreeChart chart = ChartFactory.createXYLineChart(
38.             "MergeSortTest",    // 图表的标题
39.             "DataScale",        // 横
           轴的标题名
40.             "Time(take logarithm)",
           // 纵轴的标题名
41.             dataset,            // 图表对象中
           使用的数据集对象
42.             PlotOrientation.VERTICAL,    // 图表显示的
           方向
43.             true,                // 是否显示图
           例
44.             false,              // 是否需要生
           成 tooltips
45.             false                // 是否需要生
           成 urls
46.         );
47.         // 下面所做的工作都是可选操作，主要是为了调整图表显示的风格
48.         // 同学们不必在意下面的代码
49.         // 可以将下面的代码去掉对比一下显示的不同效果
50.         chart.setBackgroundPaint(Color.WHITE);
51.         XYPlot plot = (XYPlot) chart.getPlot();
52.         plot.setBackgroundPaint(Color.lightGray);
53.         plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.
           0, 6.0));
54.         plot.setDomainGridlinePaint(Color.WHITE);
55.         plot.setRangeGridlinePaint(Color.WHITE);
56.         XYLineAndShapeRenderer renderer = (XYLineAndShapeRe
           nderer) plot.getRenderer();
57.         renderer.setDefaultShapesVisible(true);
58.         renderer.setDefaultShapesFilled(true);
59.         return chart;
60.     }
61.
62.

```

```

63.     private XYDataset createDataset() {
64.         // 本样例中想要显示的是三组数据的变化图
65.         // X 数组是三组数据共同拥有的 x 坐标值; Y1、Y2 和 Y3 数组
           分别存储了三组数据对应的 y 坐标值
66.         int[] X = new int[9];
67.         double[] Y1 = new double[9];
68.         double[] Y2 = new double[9];
69.         double[] Y3 = new double[9];
70.         for (int j = 8; j <= 16; j++) {
71.             int N = (int) pow(2, j);
72.             X[j - 8] = N;
73.             Double[] randomData = GenerateData.getAnotherRa
               ndomData(N);
74.             Y1[j - 8] = Math.log(SortTest.test(new MergeSor
               t(), randomData, 10));
75.             Double[] sortedData = GenerateData.getAnotherSo
               rtedData(N);
76.             Y2[j - 8] = Math.log(SortTest.test(new MergeSor
               t(), sortedData, 10));
77.             Double[] inverseData = GenerateData.getAnotherI
               nversedData(N);
78.             Y3[j - 8] = Math.log(SortTest.test(new MergeSor
               t(), inverseData, 10));
79.         }
80.         double[][] Y = {Y1, Y2, Y3};
81.         // jfreechart 中使用 XYSeries 对象存储一组数据的(x,y)的
           序列, 因为有三组数据所以创建三个 XYSeries 对象
82.         XYSeries[] series = {new XYSeries("RandomData"), ne
               w XYSeries("SortedData"), new XYSeries("InverseData")};
83.         int N = X.length;
84.         int M = series.length;
85.         for (int i = 0; i < M; i++)
86.             for (int j = 0; j < N; j++)
87.                 series[i].add(X[j], Y[i][j]);
88.         // 因为在该图表中显示的数据序列不止一组, 所以在
           jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对
           象中
89.         XYSeriesCollection dataset = new XYSeriesCollection
               ();
90.         for (int i = 0; i < M; i++)
91.             dataset.addSeries(series[i]);
92.
93.         return dataset;
94.     }
95.

```

```

96. }
97. package practice4;
98.
99. import org.jfree.chart.ChartFactory;
100. import org.jfree.chart.ChartPanel;
101. import org.jfree.chart.JFreeChart;
102. import org.jfree.chart.plot.PlotOrientation;
103. import org.jfree.chart.plot.XYPlot;
104. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
105. import org.jfree.chart.ui.ApplicationFrame;
106. import org.jfree.chart.ui.RectangleInsets;
107. import org.jfree.data.xy.XYDataset;
108. import org.jfree.data.xy.XYSeries;
109. import org.jfree.data.xy.XYSeriesCollection;
110. import practice3.GenerateData;
111. import practice3.Insertion;
112. import practice3.Quicksort;
113.
114. import java.awt.*;
115.
116. import static java.lang.Math.pow;
117.
118. public class QuickTestDemo extends ApplicationFrame {
119.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创建工作
120.     public QuickTestDemo(String title) {
121.         super(title);
122.         XYDataset dataset = createDataset();
123.         // 创建记录图中坐标点的数据集
124.         JFreeChart chart = createChart(dataset);
125.         // 使用上一步已经创建好的数据集生成一个图表对象
126.         ChartPanel chartPanel = new ChartPanel(chart);
127.         // 将上一步已经创建好的图表对象放置到一个可以显示的Panel上
128.         // 设置GUI面板Panel的显示大小
129.         chartPanel.setPreferredSize(new Dimension(500, 270));
130.         setContentPane(chartPanel);
131.         // 这是JavaGUI的步骤之一，不用过于关心，面向对象课程综合训练的
            // 视频中进行了讲解。
132.     }
133.
134. private JFreeChart createChart(XYDataset dataset) {
135.     // 使用已经创建好的dataset生成图表对象

```

```

132.          // JFreechart 提供了多种类型的图表对象，本次实验是需要使用XYLine 型的图表对象
133.          JFreeChart chart = ChartFactory.createXYLineChart(
134.              "QuickSortTest",          // 图表的标题
135.              "DataScale",              /
            / 横轴的标题名
136.              "Time(take logarithm)",
            // 纵轴的标题名
137.              dataset,                  // 图表对象中使用的数据集对象
138.              PlotOrientation.VERTICAL,  // 图表显示的方向
139.              true,                     // 是否显示图例
140.              false,                   // 是否需要生成 tooltips
141.              false                     // 是否需要生成 urls
142.          );
143.          // 下面所做的工作都是可选操作，主要是为了调整图表显示的风格
144.          // 同学们不必在意下面的代码
145.          // 可以将下面的代码去掉对比一下显示的不同效果
146.          chart.setBackgroundPaint(Color.WHITE);
147.          XYPlot plot = (XYPlot) chart.getPlot();
148.          plot.setBackgroundPaint(Color.lightGray);
149.          plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 6.0));
150.          plot.setDomainGridlinePaint(Color.WHITE);
151.          plot.setRangeGridlinePaint(Color.WHITE);
152.          XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer();
153.          renderer.setDefaultShapesVisible(true);
154.          renderer.setDefaultShapesFilled(true);
155.          return chart;
156.      }
157.
158.
159.      private XYDataset createDataset() {
160.          // 本样例中想要显示的是三组数据的变化图
161.          // X 数组是三组数据共同拥有的 x 坐标值；Y1、Y2 和 Y3 数组分别存储了三组数据对应的 y 坐标值
162.          int[] X = new int[9];
163.          double[] Y1 = new double[9];

```



```

164.         double[] Y2 = new double[9];
165.         double[] Y3 = new double[9];
166.         for (int j = 8; j <= 16; j++) {
167.             int N = (int) pow(2, j);
168.             X[j - 8] = N;
169.             Double[] randomData = GenerateData.getAnotherRandomData(N);
170.             Y1[j - 8] = Math.log(SortTest.test(new Quick
                sort(), randomData, 10));
171.             Double[] sortedData = GenerateData.getAnotherSortedData(N);
172.             Y2[j - 8] = Math.log(SortTest.test(new Quick
                sort(), sortedData, 10));
173.             Double[] inverseData = GenerateData.getAnotherInversedData(N);
174.             Y3[j - 8] = Math.log(SortTest.test(new Quick
                sort(), inverseData, 10));
175.         }
176.         double[][] Y = {Y1, Y2, Y3};
177.         // jfreechart 中使用 XYSeries 对象存储一组数据的
            (x,y)的序列, 因为有三组数据所以创建三个 XYSeries 对象
178.         XYSeries[] series = {new XYSeries("RandomData"),
            new XYSeries("SortedData"), new XYSeries("InverseData")};
179.         int N = X.length;
180.         int M = series.length;
181.         for (int i = 0; i < M; i++)
182.             for (int j = 0; j < N; j++)
183.                 series[i].add(X[j], Y[i][j]);
184.         // 因为在该图表中显示的数据序列不止一组, 所以在
            jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对
            象中
185.         XYSeriesCollection dataset = new XYSeriesCollect
            ion();
186.         for (int i = 0; i < M; i++)
187.             dataset.addSeries(series[i]);
188.
189.         return dataset;
190.     }
191.
192. }
193. package practice4;
194.
195. import org.jfree.chart.ChartFactory;
196. import org.jfree.chart.ChartPanel;
197. import org.jfree.chart.JFreeChart;

```

```

198. import org.jfree.chart.plot.PlotOrientation;
199. import org.jfree.chart.plot.XYPlot;
200. import org.jfree.chart.renderer.xy.XYLineAndShapeRender
    er;
201. import org.jfree.chart.ui.ApplicationFrame;
202. import org.jfree.chart.ui.RectangleInsets;
203. import org.jfree.data.xy.XYDataset;
204. import org.jfree.data.xy.XYSeries;
205. import org.jfree.data.xy.XYSeriesCollection;
206. import practice3.GenerateData;
207. import practice3.Insertion;
208. import practice3.Selection;
209.
210. import java.awt.*;
211.
212. import static java.lang.Math.pow;
213.
214. public class SelectionTestDemo extends ApplicationFrame
    {
215.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创
        建工作
216.     public SelectionTestDemo(String title) {
217.         super(title);
218.         XYDataset dataset = createDataset();
            // 创建记录图中坐标点的数据集
219.         JFreeChart chart = createChart(dataset);
            // 使用上一步已经创建好的数据集生成一个图表对象
220.         ChartPanel chartPanel = new ChartPanel(chart);
            // 将上一步已经创建好的图表对象放置到一个可以显示的 Panel 上
221.         // 设置 GUI 面板 Panel 的显示大小
222.         chartPanel.setPreferredSize(new Dimension(500, 2
            70));
223.         setContentPane(chartPanel);
            // 这是 JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的
            视频中进行了讲解。
224.     }
225.
226.     private JFreeChart createChart(XYDataset dataset) {
227.         // 使用已经创建好的 dataset 生成图表对象
228.         // JFreechart 提供了多种类型的图表对象，本次实验是需
            要使用 XYLine 型的图表对象
229.         JFreeChart chart = ChartFactory.createXYLineChar
            t(
230.             "SelectionSortTest", // 图表的标题

```

```

231.         "DataScale",                                     /
           / 横轴的标题名
232.         "Time(take logarithm)",
           // 纵轴的标题名
233.         dataset,                                         // 图表对
           象中使用的数据集对象
234.         PlotOrientation.VERTICAL,                         // 图表显
           示的方向
235.         true,                                             // 是否显
           示图例
236.         false,                                           // 是否需
           要生成 tooltips
237.         false                                           // 是否需
           要生成 urls
238.     );
239.     // 下面所做的工作都是可选操作，主要是为了调整图表显
           示的风格
240.     // 同学们不必在意下面的代码
241.     // 可以将下面的代码去掉对比一下显示的不同效果
242.     chart.setBackgroundPaint(Color.WHITE);
243.     XYPlot plot = (XYPlot) chart.getPlot();
244.     plot.setBackgroundPaint(Color.lightGray);
245.     plot.setAxisOffset(new RectangleInsets(5.0, 5.0,
           5.0, 6.0));
246.     plot.setDomainGridlinePaint(Color.WHITE);
247.     plot.setRangeGridlinePaint(Color.WHITE);
248.     XYLineAndShapeRenderer renderer = (XYLineAndShap
           eRenderer) plot.getRenderer();
249.     renderer.setDefaultShapesVisible(true);
250.     renderer.setDefaultShapesFilled(true);
251.     return chart;
252. }
253.
254.
255.     private XYDataset createDataset() {
256.         // 本样例中想要显示的是三组数据的变化图
257.         // X 数组是三组数据共同拥有的 x 坐标值；Y1、Y2 和 Y3
           数组分别存储了三组数据对应的 y 坐标值
258.         int[] X = new int[9];
259.         double[] Y1 = new double[9];
260.         double[] Y2 = new double[9];
261.         double[] Y3 = new double[9];
262.         for (int j = 8; j <= 16; j++) {
263.             int N = (int) pow(2, j);
264.             X[j - 8] = N;

```

```

265.         Double[] randomData = GenerateData.getAnotherRandomData(N);
266.         Y1[j - 8] = Math.log(SortTest.test(new Selection(), randomData, 10));
267.         Double[] sortedData = GenerateData.getAnotherSortedData(N);
268.         Y2[j - 8] = Math.log(SortTest.test(new Selection(), sortedData, 10));
269.         Double[] inverseData = GenerateData.getAnotherInversedData(N);
270.         Y3[j - 8] = Math.log(SortTest.test(new Selection(), inverseData, 10));
271.     }
272.     double[][] Y = {Y1, Y2, Y3};
273.     // jfreechart 中使用 XYSeries 对象存储一组数据的 (x,y) 的序列, 因为有三组数据所以创建三个 XYSeries 对象
274.     XYSeries[] series = {new XYSeries("RandomData"), new XYSeries("SortedData"), new XYSeries("InverseData")};
275.     int N = X.length;
276.     int M = series.length;
277.     for (int i = 0; i < M; i++)
278.         for (int j = 0; j < N; j++)
279.             series[i].add(X[j], Y[i][j]);
280.     // 因为在该图表中显示的数据序列不止一组, 所以在 jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对象中
281.     XYSeriesCollection dataset = new XYSeriesCollection();
282.     for (int i = 0; i < M; i++)
283.         dataset.addSeries(series[i]);
284.
285.     return dataset;
286. }
287.
288. }
289. package practice4;
290.
291. import org.jfree.chart.ChartFactory;
292. import org.jfree.chart.ChartPanel;
293. import org.jfree.chart.JFreeChart;
294. import org.jfree.chart.plot.PlotOrientation;
295. import org.jfree.chart.plot.XYPlot;
296. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
297. import org.jfree.chart.ui.ApplicationFrame;

```

```

298. import org.jfree.chart.ui.RectangleInsets;
299. import org.jfree.data.xy.XYDataset;
300. import org.jfree.data.xy.XYSeries;
301. import org.jfree.data.xy.XYSeriesCollection;
302. import practice3.GenerateData;
303. import practice3.Insertion;
304. import practice3.Shell;
305.
306. import java.awt.*;
307.
308. import static java.lang.Math.pow;
309.
310. public class ShellTestDemo extends ApplicationFrame {
311.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创
    建工作
312.     public ShellTestDemo(String title) {
313.         super(title);
314.         XYDataset dataset = createDataset();
            // 创建记录图中坐标点的数据集
315.         JFreeChart chart = createChart(dataset);
            // 使用上一步已经创建好的数据集生成一个图表对象
316.         ChartPanel chartPanel = new ChartPanel(chart);
            // 将上一步已经创建好的图表对象放置到一个可以显示的 Panel 上
317.         // 设置 GUI 面板 Panel 的显示大小
318.         chartPanel.setPreferredSize(new Dimension(500, 2
            70));
319.         setContentPane(chartPanel);
            // 这是 JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的
            视频中进行了讲解。
320.     }
321.
322.     private JFreeChart createChart(XYDataset dataset) {
323.         // 使用已经创建好的 dataset 生成图表对象
324.         // JFreechart 提供了多种类型的图表对象，本次实验是需
            要使用 XYLine 型的图表对象
325.         JFreeChart chart = ChartFactory.createXYLineChar
            t(
326.             "ShellSortTest", // 图表的标题
327.             "DataScale", //
            / 横轴的标题名
328.             "Time(take logarithm)",
            // 纵轴的标题名
329.             dataset, // 图表对
            象中使用的数据集对象

```

```

330.                PlotOrientation.VERTICAL,           // 图表显
                    示的方向
331.                true,                               // 是否显
                    示图例
332.                false,                             // 是否需
                    要生成 tooltips
333.                false                               // 是否需
                    要生成 urls
334.            );
335.            // 下面所做的工作都是可选操作, 主要是为了调整图表显
                    示的风格
336.            // 同学们不必在意下面的代码
337.            // 可以将下面的代码去掉对比一下显示的不同效果
338.            chart.setBackgroundPaint(Color.WHITE);
339.            XYPlot plot = (XYPlot) chart.getPlot();
340.            plot.setBackgroundPaint(Color.lightGray);
341.            plot.setAxisOffset(new RectangleInsets(5.0, 5.0,
                    5.0, 6.0));
342.            plot.setDomainGridlinePaint(Color.WHITE);
343.            plot.setRangeGridlinePaint(Color.WHITE);
344.            XYLineAndShapeRenderer renderer = (XYLineAndShap
                    eRenderer) plot.getRenderer();
345.            renderer.setDefaultShapesVisible(true);
346.            renderer.setDefaultShapesFilled(true);
347.            return chart;
348.        }
349.
350.
351.        private XYDataset createDataset() {
352.            // 本样例中想要显示的是三组数据的变化图
353.            // X 数组是三组数据共同拥有的 x 坐标值; Y1、Y2 和 Y3
                    数组分别存储了三组数据对应的 y 坐标值
354.            int[] X = new int[9];
355.            double[] Y1 = new double[9];
356.            double[] Y2 = new double[9];
357.            double[] Y3 = new double[9];
358.            for (int j = 8; j <= 16; j++) {
359.                int N = (int) pow(2, j);
360.                X[j - 8] = N;
361.                Double[] randomData = GenerateData.getAnothe
                    rRandomData(N);
362.                Y1[j - 8] = Math.log(SortTest.test(new Shell
                    (), randomData, 10));
363.                Double[] sortedData = GenerateData.getAnothe
                    rSortedData(N);

```

```

364.          Y2[j - 8] = Math.log(SortTest.test(new Shell
           (), sortedData, 10));
365.          Double[] inverseData = GenerateData.getAnotherInversedData(N);
366.          Y3[j - 8] = Math.log(SortTest.test(new Shell
           (), inverseData, 10));
367.      }
368.      double[][] Y = {Y1, Y2, Y3};
369.      // jfreechart 中使用 XYSeries 对象存储一组数据的
           (x,y) 的序列, 因为有三组数据所以创建三个 XYSeries 对象
370.      XYSeries[] series = {new XYSeries("RandomData"),
           new XYSeries("SortedData"), new XYSeries("InverseData")};
371.      int N = X.length;
372.      int M = series.length;
373.      for (int i = 0; i < M; i++)
374.          for (int j = 0; j < N; j++)
375.              series[i].add(X[j], Y[i][j]);
376.      // 因为在该图表中显示的数据序列不止一组, 所以在
           jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对象中
377.      XYSeriesCollection dataset = new XYSeriesCollection();
378.      for (int i = 0; i < M; i++)
379.          dataset.addSeries(series[i]);
380.
381.      return dataset;
382.  }
383.
384.  }
385.

```

Ctrl+M

```

1. package practice4;
2.
3. import practice3.GenerateData;
4. import practice3.Quicksort;
5. import practice3.SortAlgorithm;
6.
7. import java.lang.reflect.Array;
8. import java.util.Arrays;
9.
10. public class SortTest {
11.     // 使用指定的排序算法完成一次排序所需要的时间, 单位是纳秒
12.     public static double time(SortAlgorithm alg, Double[] numbers){

```



```

13.         double start = System.nanoTime();
14.         alg.sort(numbers);
15.         double end = System.nanoTime();
16.         return end - start;
17.     }
18.     // 为了避免一次测试数据所造成的不公平, 对一个实验完成 T 次测试, 获得 T 次测试之后的平均时间
19.     public static double test(SortAlgorithm alg, Double[] numbers,int T)
20.     {
21.         double totalTime = 0;
22.         for(int i = 0; i < T; i++){
23.             Double[] copyArray= Arrays.copyOf(numbers,numbers.length);
24.             totalTime += time(alg, copyArray);
25.         }
26.         return totalTime/T;
27.     }
28.     // 执行样例, 仅供参考。
29.     // 由于测试数据的规模大小, 算法性能, 机器性能等因素, 请同学们耐心等待每次程序的运行。
30.     public static void main(String[] args) {
31.         int[] dataLength = {100, 1000};
32.         double[] elapsedTime = new double[dataLength.length];
33.         SortAlgorithm alg = new Quicksort();
34.         for(int i = 0; i < dataLength.length; i++)
35.             elapsedTime[i] = test(alg, GenerateData.getRandomData(dataLength[i]), 5);
36.         for(double time: elapsedTime)
37.             System.out.printf("%.4f ", time);
38.         System.out.println();
39.     }
40. }
41.

```

Ctrl+M

```

1. package practice4;
2.
3. public class MainTest {
4.     public static void main(String[] args) {
5.         InsertionTestDemo demo1 = new InsertionTestDemo("Demo1");
6.         demo1.pack();
7.         demo1.setVisible(true);

```

```

8.      SelectionTestDemo demo2 = new SelectionTestDemo("De
      mo2");
9.      demo2.pack();
10.     demo2.setVisible(true);
11.     ShellTestDemo demo3 = new ShellTestDemo("Demo3");
12.     demo3.pack();
13.     demo3.setVisible(true);
14.     QuickTestDemo demo4 = new QuickTestDemo("Demo4");
15.     demo4.pack();
16.     demo4.setVisible(true);
17.     MergeTestDemo demo5 = new MergeTestDemo("Demo5");
18.     demo5.pack();
19.     demo5.setVisible(true);
20.     }
21. }
22.

```

Ctrl+M

```

1. package practice6;
2.
3. import practice3.Insertion;
4. import practice3.SortAlgorithm;
5.
6. public class OptimizedQuickSort1 extends SortAlgorithm {
7.     SortAlgorithm tmp=new Insertion();
8.     public void sort(Comparable[] objs){
9.         int left=0,right=objs.length-1;
10.        sort(objs,left,right);
11.    }
12.    public void sort(Comparable[] objs,int left,int right){
13.        if(right<=left+5){
14.            tmp.sort(objs,left,right);
15.            return;}
16.        int pivotIndex=partition(objs,left,right);
17.        sort(objs,left,pivotIndex-1);
18.        sort(objs,pivotIndex+1,right);
19.    }
20.    public int partition(Comparable[] objs,int left,int right) {
21.        int index=findMiddle(objs,left,right);
22.        int low=left,high=right-1;
23.        exchange(objs,right,index);
24.        Comparable pivot=objs[right];
25.        while(low<high){
26.            while(true){

```

```

27.         if(less(pivot,objs[low])||(low>=high)){
28.             break;
29.         }
30.         low++;
31.     }
32.     while(true){
33.         if(less(objs[high],pivot)||(high<=low)){
34.             break;
35.         }
36.         high--;
37.     }
38.     if(low<high){
39.         exchange(objs,low,high);
40.     }
41. }
42. exchange(objs,low,right);
43. return low;
44. }
45. /*int index=findMiddle(objs,left,right);
46. int low=left;
47. Comparable pivot=objs[index];
48. exchange(objs,left,index);
49.
50. while(left<right){
51.     while(true){
52.         if(less(objs[right],pivot)||(left==right)){
53.             break;
54.         }
55.         right--;
56.     }
57.     while(true){
58.         if(less(pivot,objs[left])||(left==right)){
59.             break;
60.         }
61.         left++;
62.     }
63.
64.     exchange(objs,left,right);
65. }
66. exchange(objs,left,low);
67. return left;*/
68.
69. public int findMiddle(Comparable[] objs,int left,int ri
    ght){
70.     /*Comparable first=objs[left];

```

```

71.         Comparable second=objs[(left+right)/2];
72.         Comparable third=objs[right];
73.         if((less(first,second)&&less(second,third))||less(t
             hird,second)&&less(second,first)){
74.             return (left+right)/2;
75.         }
76.         else if((less(second,third)&&less(third,first))||(l
             ess(first,third)&&less(third,second))){
77.             return right;
78.         }
79.         else{
80.             return left;
81.         }*/
82.         int mid=(left+right)/2;
83.         if(less(objs[right],objs[left]))
84.             exchange(objs,right,left);
85.         if(less(objs[right],objs[mid]))
86.             exchange(objs,right,mid);
87.         if(less(objs[mid],objs[left]))
88.             exchange(objs,mid,left);
89.         return mid;
90.     }
91.
92. }
93.

```

Ctrl+M

```

1. package practice6;
2.
3. import org.jfree.chart.ChartFactory;
4. import org.jfree.chart.ChartPanel;
5. import org.jfree.chart.JFreeChart;
6. import org.jfree.chart.plot.PlotOrientation;
7. import org.jfree.chart.plot.XYPlot;
8. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
9. import org.jfree.chart.ui.ApplicationFrame;
10. import org.jfree.chart.ui.RectangleInsets;
11. import org.jfree.data.xy.XYDataset;
12. import org.jfree.data.xy.XYSeries;
13. import org.jfree.data.xy.XYSeriesCollection;
14. import practice3.GenerateData;
15. import practice3.Insertion;
16. import practice3.QuickSort;
17. import practice4.SortTest;
18.

```

```

19. import java.awt.*;
20.
21. import static java.lang.Math.pow;
22.
23. public class PictureTest extends ApplicationFrame {
24.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创建工作
25.     public PictureTest(String title) {
26.         super(title);
27.         XYDataset dataset = createDataset();           //
                创建记录图中坐标点的数据集
28.         JFreeChart chart = createChart(dataset);       //
                使用上一步已经创建好的数据集生成一个图表对象
29.         ChartPanel chartPanel = new ChartPanel(chart); //
                将上一步已经创建好的图表对象放置到一个可以显示的Panel上
30.         // 设置GUI 面板Panel 的显示大小
31.         chartPanel.setPreferredSize(new Dimension(500, 270)
                );
32.         setContentPane(chartPanel);                   //
                这是JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的视频
                中进行了讲解。
33.     }
34.
35.     private JFreeChart createChart(XYDataset dataset) {
36.         // 使用已经创建好的dataset 生成图表对象
37.         // JFreechart 提供了多种类型的图表对象，本次实验是需要使
                用XYLine 型的图表对象
38.         JFreeChart chart = ChartFactory.createXYLineChart(
39.             "OptimizedQuickSortTest(add Insertion)",
                // 图表的标题
40.             "DataScale",                               // 横
                轴的标题名
41.             "Time(take logarithm)",
                // 纵轴的标题名
42.             dataset,                                     // 图表对象中
                使用的数据集对象
43.             PlotOrientation.VERTICAL,                   // 图表显示的
                方向
44.             true,                                         // 是否显示图
                例
45.             false,                                       // 是否需要生
                成 tooltips
46.             false                                        // 是否需要生
                成 urls
47.         );

```

```

48.          // 下面所做的工作都是可选操作，主要是为了调整图表显示的
           风格
49.          // 同学们不必在意下面的代码
50.          // 可以将下面的代码去掉对比一下显示的不同效果
51.          chart.setBackgroundPaint(Color.WHITE);
52.          XYPlot plot = (XYPlot) chart.getPlot();
53.          plot.setBackgroundPaint(Color.lightGray);
54.          plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.
           0, 6.0));
55.          plot.setDomainGridlinePaint(Color.WHITE);
56.          plot.setRangeGridlinePaint(Color.WHITE);
57.          XYLineAndShapeRenderer renderer = (XYLineAndShapeRe
           nderer) plot.getRenderer();
58.          renderer.setDefaultShapesVisible(true);
59.          renderer.setDefaultShapesFilled(true);
60.          return chart;
61.      }
62.
63.
64.      private XYDataset createDataset() {
65.          // 本样例中想要显示的是三组数据的变化图
66.          // X 数组是三组数据共同拥有的 x 坐标值；Y1、Y2 和 Y3 数组
           分别存储了三组数据对应的 y 坐标值
67.          int[] X = new int[9];
68.          double[] Y1 = new double[9];
69.          double[] Y2 = new double[9];
70.          double[] Y3 = new double[9];
71.          for (int j = 8; j <= 16; j++) {
72.              int N = (int) pow(2, j);
73.              X[j - 8] = N;
74.              Double[] randomData = GenerateData.getRandomDat
                   a(N);
75.              Y1[j - 8] = Math.log(SortTest.test(new Optimize
                   dQuickSort1(), randomData, 10));
76.              Double[] sortedData = GenerateData.getSortedDat
                   a(N);
77.              Y2[j - 8] = Math.log(SortTest.test(new Optimize
                   dQuickSort1(), sortedData, 10));
78.              Double[] inverseData = GenerateData.getInversed
                   Data(N);
79.              Y3[j - 8] = Math.log(SortTest.test(new Optimize
                   dQuickSort1(), inverseData, 10));
80.          }
81.          double[][] Y = {Y1, Y2, Y3};

```

```

82.      // jfreechart 中使用 XYSeries 对象存储一组数据的(x,y)的
      序列, 因为有三组数据所以创建三个 XYSeries 对象
83.      XYSeries[] series = {new XYSeries("RandomData"), ne
      w XYSeries("SortedData"), new XYSeries("InverseData")};
84.      int N = X.length;
85.      int M = series.length;
86.      for (int i = 0; i < M; i++)
87.          for (int j = 0; j < N; j++)
88.              series[i].add(X[j], Y[i][j]);
89.      // 因为在该图表中显示的数据序列不止一组, 所以在
      jfreechart 中需要将多组数据序列存放到一个 XYSeriesCollection 对
      象中
90.      XYSeriesCollection dataset = new XYSeriesCollection
      ();
91.      for (int i = 0; i < M; i++)
92.          dataset.addSeries(series[i]);
93.
94.      return dataset;
95.  }
96.
97. }

```

Ctrl+M

```

1. package practice6;
2.
3. public class QuickSelectTest {
4.     public static void main(String[] args) {
5.         Integer[] arr = { 5, 67, 3, 355, 63, 29, 0, 8, 88,
5.         55, 93}; // 定义一个整数数组
6.         for (int k = 1; k <= arr.length; k++) {
7.             Comparable result = QuickSelect.findKthSmallest
7.             (arr, k); // 调用快速选择算法查找第 k 小的元素
8.             System.out.println("第" + k + "小的元素
8.             是: " + result); // 输出结果
9.         }
10.    }
11. }
12. package practice6;
13.
14. public class QuickSelect {
15.     // 查找第 k 小的元素
16.     public static Comparable findKthSmallest(Comparable[] a
16.     rr, int k) {
17.         if (k < 1 || k > arr.length) {
18.             System.out.println("k 超出了序列范围");

```

```

19.     }
20.     return quickSelect(arr, 0, arr.length - 1, k - 1);
    // 调用quickSelect方法查找第k小的元素
21. }
22.
23. // 快速选择算法
24. private static Comparable quickSelect(Comparable[] arr,
    int left, int right, int k) {
25.     if (left == right) {
26.         return arr[left]; // 如果左右边界相等, 返回当前元
    素
27.     }
28.     int pivotIndex = partition(arr, left, right); // 获
    取基准元素的索引
29.     if (k < pivotIndex) {
30.         return quickSelect(arr, left, pivotIndex - 1, k
    ); // 在左侧子数组中递归查找第k小的元素
31.     } else if (k > pivotIndex) {
32.         return quickSelect(arr, pivotIndex + 1, right,
    k); // 在右侧子数组中递归查找第k小的元素
33.     } else {
34.         return arr[k]; // 如果k等于基准元素的索引, 返回基
    准元素
35.     }
36. }
37.
38. // 分区操作, 用于确定基准元素的位置
39. private static int partition(Comparable[] arr, int left
    , int right) {
40.     int i = left;
41.     int j = right + 1;
42.     Comparable pivot = arr[left]; // 选择左边界元素作为
    基准元素
43.     while (true) {
44.         while (less(arr[++i], pivot)) { // 从左往右找到第
    一个大于等于基准元素的元素
45.             if (i == right) {
46.                 break;
47.             }
48.         }
49.         while (less(pivot, arr[--j])) { // 从右往左找到第
    一个小于等于基准元素的元素
50.             if (j == left) {
51.                 break;
52.             }

```



```

53.         }
54.         if (i >= j) {
55.             break; // 如果 i 大于等于 j, 跳出循环
56.         }
57.         exchange(arr, i, j); // 交换 i 和 j 位置的元素
58.     }
59.     exchange(arr, left, j); // 将基准元素放到正确的位置
60.     return j; // 返回基准元素的索引
61. }
62. // 比较元素大小
63. private static boolean less(Comparable v, Comparable w)
    {
64.     return v.compareTo(w) < 0; // 如果 v 小于 w, 返回 true
65. }
66. // 交换元素位置
67. private static void exchange(Comparable[] arr, int i, int j) {
68.     Comparable temp = arr[i];
69.     arr[i] = arr[j];
70.     arr[j] = temp; // 交换 i 和 j 位置的元素
71. }
72. }

1. package practice6;
2.
3.
4. import org.jfree.chart.ChartFactory;
5. import org.jfree.chart.ChartPanel;
6. import org.jfree.chart.JFreeChart;
7. import org.jfree.chart.plot.PlotOrientation;
8. import org.jfree.chart.plot.XYPlot;
9. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
10. import org.jfree.chart.ui.ApplicationFrame;
11. import org.jfree.chart.ui.RectangleInsets;
12. import org.jfree.data.xy.XYDataset;
13. import org.jfree.data.xy.XYSeries;
14. import org.jfree.data.xy.XYSeriesCollection;
15. import practice3.GenerateData;
16. import practice3.Insertion;
17. import practice3.QuickSort;
18. import practice3.SortAlgorithm;
19. import practice4.SortTest;
20.
21. import java.awt.*;
22.
23. import static java.lang.Math.pow;

```

```

24.
25. public class samedataTest1 extends ApplicationFrame {
26.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创建工作
27.     public samedataTest1(String title) {
28.         super(title);
29.         XYDataset dataset = createDataset();           //
        创建记录图中坐标点的数据集
30.         JFreeChart chart = createChart(dataset);       //
        使用上一步已经创建好的数据集生成一个图表对象
31.         ChartPanel chartPanel = new ChartPanel(chart); //
        将上一步已经创建好的图表对象放置到一个可以显示的Panel上
32.         // 设置GUI 面板Panel 的显示大小
33.         chartPanel.setPreferredSize(new Dimension(500, 270)
        );
34.         setContentPane(chartPanel);                   //
        这是JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的视频
        中进行了讲解。
35.     }
36.
37.     private JFreeChart createChart(XYDataset dataset) {
38.         // 使用已经创建好的dataset 生成图表对象
39.         // JFreechart 提供了多种类型的图表对象，本次实验是需要使用
        XYLine 型的图表对象
40.         JFreeChart chart = ChartFactory.createXYLineChart(
41.             "Comparation In Many Same Data",        // 图
        表的标题
42.             "Scale of Same Data",
        // 横轴的标题名
43.             "Time(take logarithm)",
        // 纵轴的标题名
44.             dataset,                                  // 图表对象中
        使用的数据集对象
45.             PlotOrientation.VERTICAL,                // 图表显示的
        方向
46.             true,                                     // 是否显示图
        例
47.             false,                                    // 是否需要生
        成 tooltips
48.             false                                    // 是否需要生
        成 urls
49.         );
50.         // 下面所做的工作都是可选操作，主要是为了调整图表显示的
        风格
51.         // 同学们不必在意下面的代码
52.         // 可以将下面的代码去掉对比一下显示的不同效果
    
```

```

53.      chart.setBackgroundPaint(Color.WHITE);
54.      XYPlot plot = (XYPlot) chart.getPlot();
55.      plot.setBackgroundPaint(Color.lightGray);
56.      plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.
0, 6.0));
57.      plot.setDomainGridlinePaint(Color.WHITE);
58.      plot.setRangeGridlinePaint(Color.WHITE);
59.      XYLineAndShapeRenderer renderer = (XYLineAndShapeRe
nderer) plot.getRenderer();
60.      renderer.setDefaultShapesVisible(true);
61.      renderer.setDefaultShapesFilled(true);
62.      return chart;
63.  }
64.
65.
66.  private XYDataset createDataset() {
67.      // 本样例中想要显示的是三组数据的变化图
68.      // X 数组是三组数据共同拥有的 x 坐标值; Y1、Y2 和 Y3 数组
分别存储了三组数据对应的 y 坐标值
69.      double[] X = {0.5,0.6,0.8,1};
70.      double[] Y1 = new double[4];
71.      double[] Y2 = new double[4];
72.      int SCALE=(int)pow(2,16);
73.      SortAlgorithm commonquicksort=new QuickSort2();
74.      //SortAlgorithm threewayquicksort=new ThreeWayQuick
Sort();
75.      Y1[0]=SortTest.test(commonquicksort,GenerateData.ge
tfiftypercentSameData(SCALE),10);
76.      Y1[1]=SortTest.test(commonquicksort,GenerateData.ge
tsixtypercentSameData(SCALE),10);
77.      Y1[2]=SortTest.test(commonquicksort,GenerateData.ge
teightypercentSameData(SCALE),10);
78.      Y1[3]=SortTest.test(commonquicksort,GenerateData.ge
tallSameData(SCALE),10);
79.      System.out.println(""+Y1[0]+" "+Y1[1]+" "+Y1[2]+" "
+Y1[3]);
80.      /*Y2[0]=Math.Log(SortTest.test(threewayquicksort,Ge
nerateData.getfiftypercentSameData(SCALE),10));
81.      Y2[1]=Math.log(SortTest.test(threewayquicksort,Gene
rateData.getsixtypercentSameData(SCALE),10));
82.      Y2[2]=Math.log(SortTest.test(threewayquicksort,Gene
rateData.geteightypercentSameData(SCALE),10));
83.      Y2[3]=Math.log(SortTest.test(threewayquicksort,Gene
rateData.getallSameData(SCALE),10));*/
84.      double[][] Y = {Y1};

```

```

85.         XYSeries[] series = {new XYSeries("CommonQuickSort"
        )});
86.         int N = X.length;
87.         int M = series.length;
88.         for (int i = 0; i < M; i++)
89.             for (int j = 0; j < N; j++)
90.                 series[i].add(X[j], Y[i][j]);
91.         // 因为在该图表中显示的数据序列不止一组，所以在
        jfreechart 中需要将多组数据序列存放到一个XYSeriesCollection 对
        象中
92.         XYSeriesCollection dataset = new XYSeriesCollection
        ();
93.         for (int i = 0; i < M; i++)
94.             dataset.addSeries(series[i]);
95.
96.         return dataset;
97.     }
98.
99. }
100. package practice6;
101.
102. import practice3.SortAlgorithm;
103. public class ThreeWayQuickSort extends SortAlgorithm {
104.     public void sort(Comparable[] objs){
105.         int left=0,right=objs.length-1;
106.         sort(objs,left,right);
107.     }
108.     public void sort(Comparable[] objs,int left,int righ
        t){
109.         if(left<right){
110.             int lt=left,i=left+1,gt=right;
111.             exchange(objs,findMiddle(objs,left,right),le
                ft);
112.             Comparable v=objs[left];
113.             while(i<=gt){
114.                 int cmp=objs[i].compareTo(v);
115.                 if(cmp<0){
116.                     exchange(objs,lt++,i++);
117.                 }
118.                 else if(cmp>0){
119.                     exchange(objs,i,gt--);
120.                 }
121.                 else{
122.                     i++;
123.                 }

```

```

124.         }
125.         sort(objs,left,lt-1);
126.         sort(objs,gt+1,right);
127.     }
128. }
129.     public int findMiddle(Comparable[] objs,int left,int
        right){
130.
131.         int mid=(left+right)/2;
132.         if(less(objs[right],objs[left]))
133.             exchange(objs,right,left);
134.         if(less(objs[right],objs[mid]))
135.             exchange(objs,right,mid);
136.         if(less(objs[mid],objs[left]))
137.             exchange(objs,mid,left);
138.         return mid;
139.     }
140. }
141. package practice6;
142.
143. import practice3.GenerateData;
144. import practice3.Quicksort;
145. import practice3.SortAlgorithm;
146. import practice4.QuickTestDemo;
147. import practice4.SortTest;
148.
149. import static java.lang.Math.pow;
150.
151. public class Main {
152.     public static void main(String[] args){
153.         //int N=(int)pow(2,16);
154.         //System.out.println(SortTest.test(new QuickSort
            2(),GenerateData.getRandomData(N),10 ));
155.
156.         samedataTest1 demo6 = new samedataTest1("Demo6")
            ;
157.         demo6.pack();
158.         demo6.setVisible(true);
159.         /* int T=10;
160.         SortAlgorithm tt1=new Quicksort();
161.         SortAlgorithm tt2=new OptimizedQuickSort1();
162.         for(int j=8;j<=16;j++){
163.             System.out.println("////////////////////////////////
                //////////////////////////////////");

```

```

164.          System.out.println("当数据规模为 2 的"+j+"次方
           时: ");
165.          int N=(int)pow(2,j);
166.          Double[] randomData=GenerateData.getRandomDa
           ta(N);
167.          System.out.println("随机数据序列测试结果:
           "+SortTest.test(tt1,randomData,T));
168.          Double[] randomData2=GenerateData.getRandomD
           ata(N);
169.          System.out.println("*随机数据序列测试结果:
           "+SortTest.test(tt2,randomData2,T));
170.          Double[] sortedData=GenerateData.getSortedDa
           ta(N);
171.          System.out.println(("顺序数据序列测试结果:
           "+SortTest.test(tt1,sortedData,T)));
172.          Double[] sortedData2=GenerateData.getSortedD
           ata(N);
173.          System.out.println(("*顺序数据序列测试结果:
           "+SortTest.test(tt2,sortedData2,T)));
174.          Double[] inverseData=GenerateData.getInverse
           dData(N);
175.          System.out.println(("逆序数据序列测试结果:
           "+SortTest.test(tt1,inverseData,T)));
176.          Double[] inverseData2=GenerateData.getInvers
           edData(N);
177.          System.out.println(("*逆序数据序列测试结果:
           "+SortTest.test(tt2,inverseData2,T)));
178.          System.out.println("////////////////////
           //////////////////////");
179.          */
180.          /*int SCALE=(int)pow(2,16);
181.          SortAlgorithm commonquicksort=new QuickSort2();
182.          System.out.println(Math.log(SortTest.test(common
           quicksort,GenerateData.getfiftypercentSameData(SCALE),10)))
           );
183.          SortAlgorithm te=new QuickSort2();
184.          Double[] tes=GenerateData.getAnotherRandomData(5
           000);
185.          te.sort(tes);
186.          System.out.println(te.isSorted(tes));*/
187.          }
188.      }

```