|  |  |
| --- | --- |
| **数据结构与算法 作业报告** | |
| 第一次 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **姓名** | 郝飞洋 |
| **班级** | 软件2202班 |
| **学号** | 2221411150 |
| **电话** | 13383536923 |
| **Email** | 958015679@qq.com |
| **日期** | 2023-11-27 |

**目录**

[任务1 2](#_Toc152448527)

[任务2 2](#_Toc152448528)

[题目 2](#_Toc152448529)

[数据设计 2](#_Toc152448530)

[算法设计 2](#_Toc152448531)

[主干代码说明 3](#_Toc152448532)

[运行结果展示 5](#_Toc152448533)

[总结和收获 5](#_Toc152448534)

[任务3 5](#_Toc152448535)

[题目 5](#_Toc152448536)

[数据设计 5](#_Toc152448537)

[算法设计 6](#_Toc152448538)

[主干代码说明 8](#_Toc152448539)

[运行结果展示（仅选一种算法测试结果展示） 15](#_Toc152448540)

[总结和收获 15](#_Toc152448541)

[任务4 15](#_Toc152448542)

[题目 15](#_Toc152448543)

[数据设计 15](#_Toc152448544)

[算法设计 16](#_Toc152448545)

[主干代码说明 16](#_Toc152448546)

[运行结果展示 19](#_Toc152448547)

[规律阐述 20](#_Toc152448548)

[任务5 21](#_Toc152448549)

[题目：数据分布对排序算法的影响 21](#_Toc152448550)

[数据设计 21](#_Toc152448551)

[算法设计 22](#_Toc152448552)

[主干代码说明 22](#_Toc152448553)

[运行结果展示 23](#_Toc152448554)

[规律阐述 23](#_Toc152448555)

[任务6 23](#_Toc152448556)

[题目：快速排序的再探讨和应用 23](#_Toc152448557)

[附录：每个题的源代码 24](#_Toc152448558)

[任务2 24](#_Toc152448559)

[任务3 25](#_Toc152448560)

[任务4 29](#_Toc152448561)

[任务5 34](#_Toc152448562)

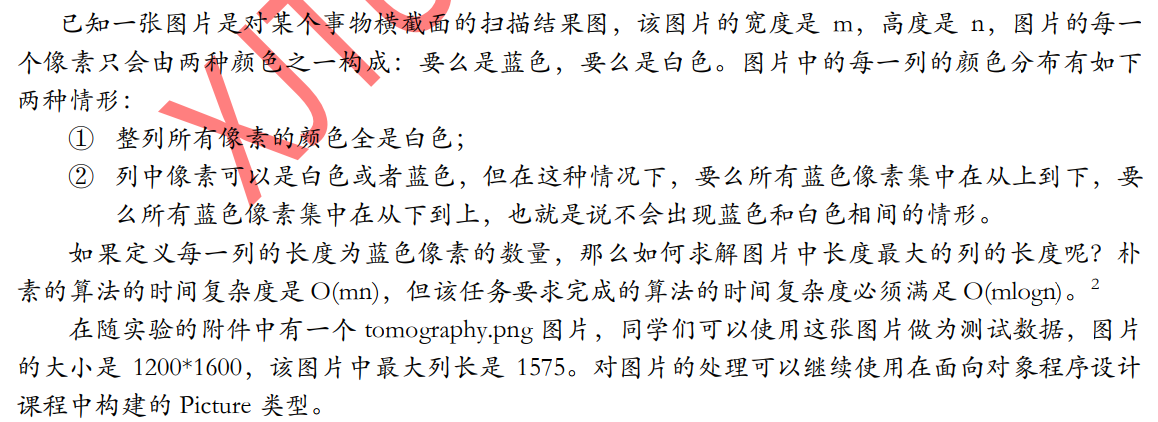
[任务6 34](#_Toc152448563)

# 任务1

附于整个文档最后

# 任务2

## 题目



## 数据设计

利用面向对象课程中构建的Picture类型对图片进行处理

将每一列都视作一个排好序的线性表，通过索引来确定其“长度”

设置一个记录最大值的变量来对所求值进行存储

## 算法设计

题目中要求算法的时间复杂度必须满足O(mlogn)，所以自然想到用分治+递归。具体地说，就是用二分法来确定蓝白交界处的位置，并通过其索引来判断该列的长度。

具体算法如下：

**长度计算算法：**

判断首尾颜色是否相同，

若是，则长度为0，算法结束

有两个指针分别指向首元素和尾元素

无条件的循环：

判断首元素和中间元素是否相等：

若是，首元素指针 <-- 中间元素指针

若否，尾元素指针 <-- 中间元素指针

判断两个指针是否指向相邻位置：

若是，

判断这列中首元素的颜色是否是蓝色：

若是，则长度=首元素指针+1，算法结束

若否，则长度=图片的高度-尾元素指针，算法结束

**最大列长算法：**

初始化max=0，遍历所有的列，调用列长计算算法，将各个列的列长与max比较，若大于max，则将其值赋给max。

## 主干代码说明

1. import java.awt.\*;
2. import java.io.IOException;

导入一些必要的包

1. public class MaxColumnLength {
2. public static void main(String[] args) throws IOException {
3. Picture testPicture = new Picture("F:\\1学校课程\\3\\数据结构\\数据结构实验1\\homework1(1)[\\tomography.png](file:///\\tomography.png)");

设置max初始值为0

1. int max = 0;

遍历图片的宽度方向：

1. for(int c = 0; c < testPicture.getWidth(); c++){

根据前述算法进行实现：

1. int len = 0;
2. if(testPicture.getColor(c,0).equals(testPicture.getColor(c, testPicture.getHeight()-1))){
3. continue;
4. }
5. int low = 0;
6. int high = testPicture.getHeight()-1;
7. while(true){
8. int mid = low + (high - low) / 2;
9. if(testPicture.getColor(c,low).equals(testPicture.getColor(c,mid))){
10. low = mid;
11. }else{
12. high = mid;
13. }
14. if(high - low <= 1){
15. if(testPicture.getColor(c,0).equals(Color.white)){
16. len = testPicture.getHeight() - high;
17. break;
18. }else{
19. len = low + 1;
20. break;
21. }
22. }
23. }

判断是否大于已知的最大值：

1. if(len > max){
2. max = len;
3. }
4. }

输出结果：

1. System.out.println(max);
2. }
3. public int getLength(Picture pic){
4. return 0;
5. }
6. }

## 运行结果展示



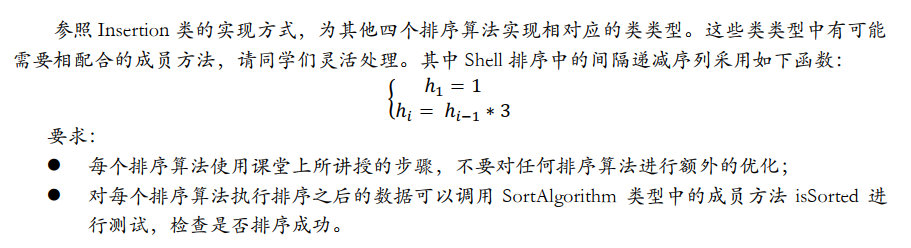
## 总结和收获

通过自己设计算法，体验到设计算法的技巧和其中需要的细致思考。

# 任务3

## 题目

排序算法的实现



## 数据设计

使用提供的模板进行算法编写。

排序对象为Comparable类的数组，在排序过程中的辅助变量用合适的数据类型进行表示。

## 算法设计

选择排序：

外层循环（从0到N）：

设min的初始值为循环变量i

内层循环（从i到N）：

如果当前扫描到的元素比min对应的元素小，

则将当前下标赋给min

交换i对应的值和min对应的值

希尔排序：

第一层循环（用于计算增量，使用题目中给出的增量序列）：

第二层循环（选择每一组开始的元素）：

第三层循环（从每组的第二个元素开始遍历）：

第四层循环（比较该元素与组内前一个元素的大小，若小于前一个元素，则向前交换，不断向前直到不能交换为止）

快速排序：

需要用两个辅助变量high和low来标识目前正在处理的数组的首尾

若high==low，排序结束，算法停止。

根据“三选一法”选择轴值

通过双指针法把数组分成两部分：

初始状态两个指针分别指向首尾

循环：

左指针：只要指向的值比轴值小就向右移

右指针：只要指向的值比轴值大且没有移动到数组左端就向左移

判断两指针位置是否交叉：

若否，将两指针指向的值互换

若是，将左指针所指的值与轴值交换

调用快速排序对low到左指针-1的部分进行排序

调用快速排序对左指针+1到high的部分进行排序

归并排序：

若数组元素个数不为0或1：

将数组从中间一分为二

将两个数组分别进行归并排序

将两个数组合并

合并数组的具体方法：

将原数组1拷贝到一个临时数组中

在下面的过程中，指针指向的是临时数组中的元素，归并结果存放在原来的位置上。

指针1、2分别指向数组1、2的头部

比较两指针所指元素的大小，取较小值加入新数组中，并把这个指针右移

重复上一步，直到有一个指针已经到头：

将另一个指针所指的元素及之后的元素拷贝到新数组中

## 主干代码说明

**快速排序：**

**代码说明已经以注释形式写在代码之中**

**这里使用了多种方法实现未优化的快速排序**

1. public class Quick extends SortAlgorithm {
2. public void sort(Comparable[] objs) {
3. int N = objs.length;
4. quickSort(objs, 0, N - 1);
5. }
6. public static void main(String[] args){
7. Double[] testData = GenerateData.getRandomData(5);
8. Double[] test = new Double[6];
9. for (int i = 0; i < 5; i++) {
10. test[i] = testData[i];
11. System.out.print(test[i] + " ");
12. }
13. System.out.println();
14. SortAlgorithm alg = new Quick();
15. alg.sort(testData);
16. for (int i = 0; i < 5; i++) {
17. System.out.print(testData[i] + " ");
18. }
19. System.out.println(test.equals(testData));
20. System.out.println(alg.isSorted(testData));
21. }
22. private Comparable choosePivot(Comparable[] objs, int low, int high){
23. int mid = low + (high - low) / 2;
24. Comparable pivot = objs[high];
25. if (!(less(objs[low], objs[high]) ^ less(objs[mid], objs[low]))) {
26. pivot = objs[low];
27. exchange(objs, low, high);
28. } else if (!(less(objs[mid], objs[low]) ^ less(objs[high], objs[mid]))) {
29. pivot = objs[mid];
30. exchange(objs, mid, high);
31. }
32. return pivot;
33. }
34. */\*\**
35. \* 老师上课讲的快速排序方法
36. \* @param objs
37. \* @param low
38. \* @param high
39. \*/
40. public void quickSort(Comparable[] objs, int low, int high) {
41. *//基准情形*
42. if (low >= high) {
43. return;
44. }
45. *//选择轴值*
46. Comparable pivot = choosePivot(objs, low, high);
47. *//相向双指针法的操作和递归*
48. int left = low;
49. int right = high - 1;
50. while (true) {
51. while (less(objs[left], pivot)) {
52. left++;
53. }
54. while (less(pivot, objs[right]) && right > 0) {
55. right--;
56. }
57. if (left < right) {
58. exchange(objs, left, right);
59. } else {
60. exchange(objs, left, high);
61. if (left > 1) {
62. quickSort(objs, low, left - 1);
63. }
64. quickSort(objs, left + 1, high);
65. return; *//一定不要忘记这个return，否则会一直循环退不出去！*
66. }
67. }
68. }
69. */\*\**
70. \* 将partition函数分离出来的快速排序算法
71. \* @param objs
72. \* @param low
73. \* @param high
74. \*/
75. public void quick(Comparable[] objs, int low, int high){
76. *//基准情形*
77. if (low >= high) {
78. return;
79. }
80. int pivotIndex = partition\_2(objs, low, high);
81. quick(objs, low, pivotIndex-1);
82. quick(objs, pivotIndex+1, high);
83. }
84. */\*\**
85. \* 单路划分（两指针同向运动）
86. \* @param objs
87. \* @param low
88. \* @param high
89. \* @return
90. \*/
91. private int partition\_1(Comparable[] objs, int low, int high){
92. Comparable pivot = choosePivot(objs, low, high);
93. int i = low; *//i相当于是一个“栈”的top指针，指向目前已扫描到的最后一个比轴值小的元素的下一个位置。*
94. int j = low; *//j往右遍历搜索比轴值小的元素，找到后就放到i左边的那个“栈”里去*
95. *//刚开始时i和j在同一个位置上*
96. while(j < high){
97. if(less(objs[j],pivot)){ *//j找到了比轴值小的元素*
98. if(i != j){
99. exchange(objs,i,j);
100. }
101. *//当一个比轴值小的元素放到左边的“栈”中时，i就会向后移动一位。*
102. i++;
103. }
104. j++;
105. }
106. *//此时i指向的是比轴值小的元素的后一个元素（也就是第一个比轴值大的元素）*
107. exchange(objs,i,high);
108. *//将i指向的值和轴值交换位置，完成划分。*
109. return i;
110. *//返回轴值最后所在的位置。*
111. }
112. */\*\**
113. \* 普通双路划分（两指针相向运动）
114. \* @param objs
115. \* @param low
116. \* @param high
117. \* @return
118. \*/
119. public int partition\_2(Comparable[] objs, int low, int high){
120. Comparable pivot = choosePivot(objs, low, high);
121. int left = low;
122. int right = high - 1;
123. while(left < right){
124. *//其实只要让left指针和right指针碰面即可，就算交错了也必定会是交错到对方已经排过的领域的第一个值.*
125. *//下面的两个循环的顺序是有讲究的，left先动，最后指向的值是第一个大于轴值的值；right先动，最后指向的是最后一个小于轴值的值。*
126. while(left < right && less(objs[left], pivot)){ *//这两个循环中的left<right都是有必要的！*
127. left++;
128. }
129. while(left < right && less(pivot, objs[right])){ *//这两个循环中的比较都是严格小于，这样做可以使排序后等于pivot的元素平均划分到两个区域中，为下一次划分奠定基础*
130. right--;
131. }
132. exchange(objs,left,right);
133. }
134. exchange(objs,left,high);
135. return left;
136. }
137. }

归并排序：

1. public class Merge extends SortAlgorithm {
2. public void sort(Comparable[] objs) {
3. int N = objs.length;
4. mergeSort(objs,0,N-1);
5. }

以下是mergeSort的主干代码：

1. public void mergeSort(Comparable[] objs, int low, int high){

递归的基本情况：

1. if(low == high){
2. return;
3. }
4. int mid = low + (high - low) / 2;

递归调用：

1. mergeSort(objs, low, mid);
2. mergeSort(objs, mid + 1, high);

调用merge函数：

1. merge(objs, low, mid + 1, high);
2. }

这里将merge单独拿出来，方便mergeSort本身的递归调用。

1. private void merge(Comparable[] objs, int start, int start2, int end){
2. int len1 = start2 - start;
3. Comparable[] tmp = new Comparable[len1];
4. for (int i = 0; i < len1; i++) {
5. tmp[i] = objs[i];
6. }
7. int p1 = 0;
8. int p2 = start2;
9. for(int i = start; i <= end; i++){
10. if(less(tmp[p1],objs[p2])){
11. objs[i] = tmp[p1];
12. p1++;
13. if(p1 == len1){
14. break;
15. }
16. }else{
17. objs[i] = objs[p2];
18. p2++;
19. if(p2>end){
20. while(p1<len1){
21. i++;
22. objs[i] = tmp[p1];
23. p1++;
24. }
25. }
26. }
27. }
28. }

在主函数中写辅助验证的代码：

1. public static void main(String[] args){
2. Double[] testData = GenerateData.getRandomData(5);
3. Double[] test = new Double[6];
4. for (int i = 0; i < 5; i++) {
5. test[i] = testData[i];
6. }
7. SortAlgorithm alg = new Merge();
8. alg.sort(testData);
9. System.out.println(test.equals(testData));
10. System.out.println(alg.isSorted(testData));
11. }
12. }

选择排序：

1. public class Selection extends SortAlgorithm {
2. public void sort(Comparable[] objs) {
3. int N = objs.length;
4. for (int i = 0; i < N; i++) {
5. int min = i;
6. for (int j = i; j < N; j++) {
7. if(less(objs[j],objs[min])){
8. min = j;
9. }
10. }
11. exchange(objs,i,min);
12. }
13. }
14. public static void main(String[] args){
15. Double[] testData = GenerateData.getRandomData(5);
16. Double[] test = new Double[6];
17. for (int i = 0; i < 5; i++) {
18. test[i] = testData[i];
19. }
20. SortAlgorithm alg = new Insertion();
21. alg.sort(testData);
22. System.out.println(test.equals(testData));
23. System.out.println(alg.isSorted(testData));
24. }
25. }

希尔排序：

1. public class Shell extends SortAlgorithm {
2. public void sort(Comparable[] objs){
3. int N = objs.length;
4. *//计算每一轮的增量*
5. for(int i = N / 3; i > 0; i /= 3){
6. *//选择每一组的开始元素*
7. for( int j = 0; j < i; j ++){
8. *//开始插入排序，由于下面是和前一个元素比较，所以要从每组的第二个元素开始遍历*
9. for (int k = i + j; k < N; k += i) {
10. *//将每一个元素放到该放的位置上去*
11. for (int l = k; l >= i && less(objs[l],objs[l-i]); l -= i) {
12. exchange(objs,l,l-i);
13. }
14. }
15. }
16. }
17. }
18. public static void main(String[] args){
19. Double[] testData = GenerateData.getRandomData(5);
20. Double[] test = new Double[6];
21. for (int i = 0; i < 5; i++) {
22. test[i] = testData[i];
23. }
24. SortAlgorithm alg = new Shell();
25. alg.sort(testData);
26. System.out.println(test.equals(testData));
27. System.out.println(alg.isSorted(testData));
28. }
29. }

## 运行结果展示（仅选一种算法测试结果展示）



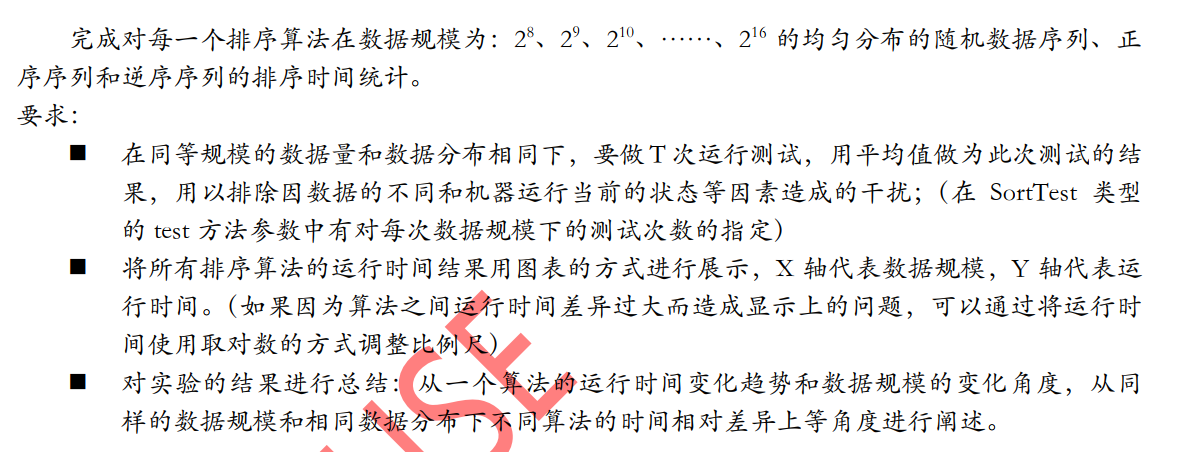
## 总结和收获

通过对排序算法的实现，对各个排序算法有了更深的理解，并对算法实现过程中需要注意的地方更加熟悉了。

# 任务4

## 题目

排序算法性能测试和比较



## 数据设计

使用提供的模板进行算法编写。

利用任务3中写好的排序算法进行完成。

通过对运行时间取对数的方法调整比例尺。

利用java提供的类来产生随机数据。

## 算法设计

通过理解给定作图模板并进行运用来画出图像。

## 主干代码说明

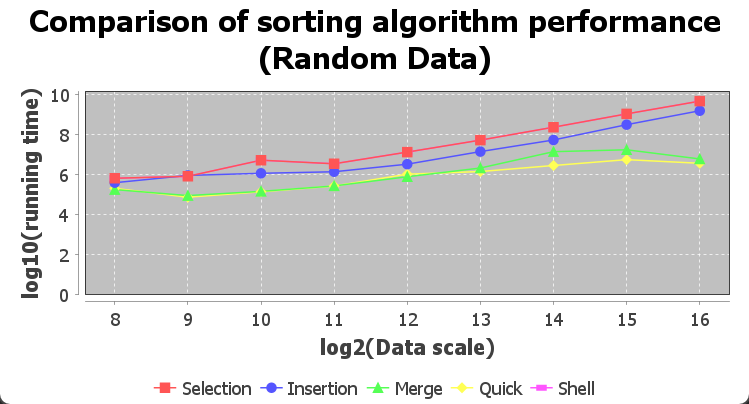
通过使用给定的画图模板来利用jfreechart画图

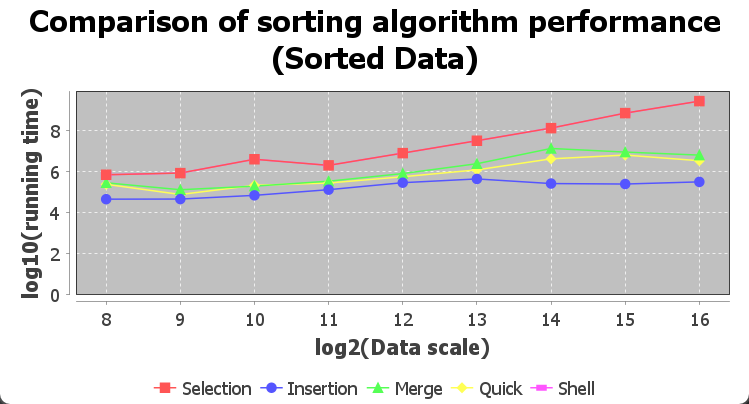
1. import org.jfree.chart.ChartFactory;
2. import org.jfree.chart.ChartPanel;
3. import org.jfree.chart.JFreeChart;
4. import org.jfree.chart.plot.PlotOrientation;
5. import org.jfree.chart.plot.XYPlot;
6. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
7. import org.jfree.chart.ui.ApplicationFrame;
8. import org.jfree.chart.ui.RectangleInsets;
9. import org.jfree.data.xy.XYDataset;
10. import org.jfree.data.xy.XYSeries;
11. import org.jfree.data.xy.XYSeriesCollection;
12. import java.awt.\*;
13. public class LineXYDemo extends ApplicationFrame {
14. *// 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*
15. public LineXYDemo(String title){
16. super(title);
17. XYDataset dataset = createDataset();             *// 创建记录图中坐标点的数据集*
18. 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*
19. 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*
20. *// 设置GUI面板Panel的显示大小*
21. chartPanel.setPreferredSize(new Dimension(500, 270));
22. 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。*
23. }
24. private JFreeChart createChart(XYDataset dataset) {
25. *// 使用已经创建好的dataset生成图表对象*
26. *// 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*
27. JFreeChart chart = ChartFactory.createXYLineChart(
28. "Comparison of sorting algorithm performance(Random Data)",      *// 图表的标题*
29. "Data scale",                           *// 横轴的标题名*
30. "log10(running time)",                           *// 纵轴的标题名*
31. dataset,                       *// 图表对象中使用的数据集对象*
32. PlotOrientation.VERTICAL,      *// 图表显示的方向*
33. true,                          *// 是否显示图例*
34. false,                         *// 是否需要生成tooltips*
35. false                          *// 是否需要生成urls*
36. );
37. *// 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*
38. *// 同学们不必在意下面的代码*
39. *// 可以将下面的代码去掉对比一下显示的不同效果*
40. chart.setBackgroundPaint(Color.WHITE);
41. XYPlot plot = (XYPlot)chart.getPlot();
42. plot.setBackgroundPaint(Color.lightGray);
43. plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 6.0));
44. plot.setDomainGridlinePaint(Color.WHITE);
45. plot.setRangeGridlinePaint(Color.WHITE);
46. XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer();
47. renderer.setDefaultShapesVisible(true);
48. renderer.setDefaultShapesFilled(true);
49. return chart;
50. }
51. private double[][] ArrayLog10(double[][] rawdata){
52. int row = rawdata.length;
53. int column = rawdata[0].length;
54. double[][] result = new double[row][column];
55. for (int i = 0; i < row; i++) {
56. for(int j = 0; j < column; j++){
57. result[i][j] = Math.log10(rawdata[i][j]);
58. }
59. }
60. return result;
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*

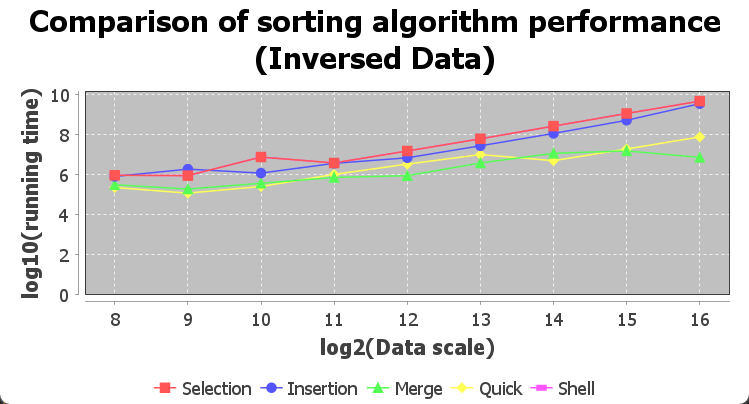
此处输入数据：

1. double[] X = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
2. double[] Y1 = {698440.0000,817360.0000,5733380.0000,5270680.0000,14846220.0000,59159340.0000,273607600.0000,1177341760.0000,4901339440.0000};
3. double[] Y2 = {268400.0000,558500.0000,1187359.8000,2837539.6000,3360580.0000,12852500.2000,52058420.0000,321808119.8000,1413004639.8000};
4. double[] Y3 = {210940.0000,86920.0000,178140.0000,293240.0000,596619.8000,1314800.2000,10481939.8000,12115399.8000,6555679.6000};
5. double[] Y4 = {251300.0000,72000.0000,133200.0000,212540.0000,690520.0000,1040740.0000,2148280.0000,4855980.0000,10460420.0000};
6. double[] Y5 = {163880.0000,132240.0000,298420.0000,184160.0000,694540.0000,1258120.0000,1564240.0000,2983220.0000,6451060.0000};
7. double[][] Y\_raw = {Y1, Y2, Y3, Y4, Y5};
8. double[][] Y = ArrayLog10(Y\_raw);
9. *// 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*
10. XYSeries[] series = {new XYSeries("Selection"), new XYSeries("Insertion"), new XYSeries("Merge"), new XYSeries("Quick"), new XYSeries("Shell")};
11. int N = X.length;
12. int M = series.length;
13. for(int i = 0; i < M; i++)
14. for(int j = 0; j < N; j++)
15. series[i].add(X[j], Y[i][j]);
16. *// 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*
17. XYSeriesCollection dataset = new XYSeriesCollection();
18. for(int i = 0; i < M; i++)
19. dataset.addSeries(series[i]);
20. return dataset;
21. }
22. public static void main(String[] args) {
23. LineXYDemo demo = new LineXYDemo("Comparison of sorting algorithm performance");
24. demo.pack();
25. demo.setVisible(true);
26. }
27. }

## 运行结果展示







## 规律阐述

选择排序：

时间复杂度为O(n^2)，比较次数较多，数据交换次数较少。选择排序在处理数据时，只有在发现当前位置的数据比后面的位置上的数据大时，才会进行数据交换，因此交换次数相对较少。在数据量较大时，选择排序的时间效率较低。

插入排序：

时间复杂度为O(n^2)，比较次数和数据交换次数均较多。插入排序在处理数据时，需要不断地将新数据插入到已排序好的数列中，并保持数列的有序性。因此，插入排序在处理数据量较大、数据分布无规律或数据本身无序的情况下，效率较低。

快速排序：

时间复杂度为O(nlogn)，比较次数和数据交换次数均较多。快速排序是一种分治算法，通过选择一个基准元素将待排序的数列分成两部分，然后对这两部分分别进行排序。在处理数据时，快速排序能够利用分治的思想，将一个较大的数列分成若干个小数列，然后对每个小数列进行递归处理。因此，快速排序在处理数据量较大、数据分布无规律或数据本身无序的情况下，效率较高。

归并排序：

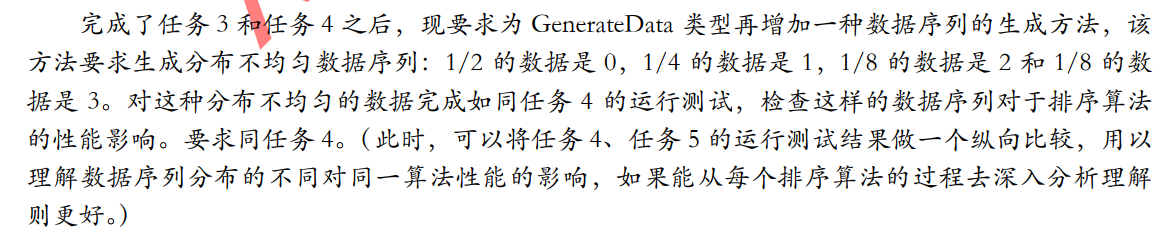
时间复杂度为O(nlogn)，比较次数和数据交换次数均较多。归并排序是一种基于分治思想的排序算法，通过将待排序的数列分成若干个子序列，然后对每个子序列进行递归处理，最终合并得到有序的序列。归并排序在处理数据时，需要将数据进行分解和合并操作，因此需要额外的空间存储临时数据。但是，归并排序具有较好的稳定性，即相同值的元素在排序后保持原有的相对顺序。

希尔排序：

希尔排序的运行时间随着数据量的增加而增加，但相对于冒泡排序和插入排序等算法，其运行时间的变化趋势较为平缓。这是因为希尔排序在每次迭代时都会将数据分成更多的子序列，从而在每个子序列上进行更精细的排序操作。因此，当数据量较大时，希尔排序的效率相对较高。希尔排序在处理不同规模的数据时，其性能表现也不同。对于较小的数据集，希尔排序的效率可能不如其他一些算法（如插入排序），因为希尔排序的额外操作（即划分和合并子序列）会使得运行时间较长。然而，随着数据规模的增加，希尔排序的优势逐渐显现，因为它能够更好地利用缓存和减少内存访问的开销。在相同的数据规模和相同的数据分布下，希尔排序相对于其他一些排序算法（如冒泡排序、插入排序）具有更快的运行速度。这是因为希尔排序在处理数据时，能够更快地缩小数据的范围，从而减少需要比较和交换的元素数量。

# 任务5

## 题目：数据分布对排序算法的影响



## 数据设计

继承上一个题的设计思想。

## 算法设计

通过计算数组长度\*比例来计算赋值的个数，赋值之后再用shuffle函数来将数组打乱。

测试过程与上一个任务大致相同，不再赘述。

## 主干代码说明

1. public static Double[] getNovelData(int N){

创建数组：

1. Double[] numbers = new Double[N];
2. for(int i = 0; i < N/2; i++){
3. numbers[i] = 0.0;
4. }

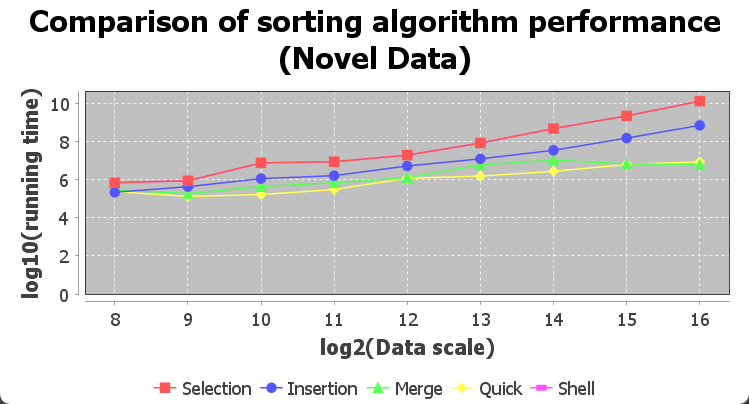
进行赋值：

1. for(int i = N/2; i < 3 \* N / 4; i++){
2. numbers[i] = 1.0;
3. }
4. for(int i = 3 \* N / 4; i < 7 \* N / 8; i++){
5. numbers[i] = 2.0;
6. }
7. for(int i = 7 \* N / 8; i < N; i++){
8. numbers[i] = 3.0;
9. }
10. shuffle(numbers, 0, numbers.length - 1);

打乱数组：

1. return numbers;
2. }

## 运行结果展示



## 规律阐述

在不同数据分布下，每个排序算法的表现都会有所变化。分布不均匀数据可能对插入排序、选择排序等有较大影响，而对于快速排序、归并排序等影响相对较小。每个算法对数据分布的敏感性不同，这种比较有助于选择适当的算法来处理特定类型的数据。

# 任务6

## 题目：快速排序的再探讨和应用

# 附录：每个题的源代码

## 任务2

1. import java.awt.\*;
2. import java.io.IOException;
3. public class MaxColumnLength {
4. public static void main(String[] args) throws IOException {
5. Picture testPicture = new Picture("F:\\1学校课程\\3\\数据结构\\数据结构实验1\\homework1(1)\\tomography.png");
6. int max = 0;
7. for(int c = 0; c < testPicture.getWidth(); c++){
8. int len = 0;
9. if(testPicture.getColor(c,0).equals(testPicture.getColor(c, testPicture.getHeight()-1))){
10. continue;
11. }
12. int low = 0;
13. int high = testPicture.getHeight()-1;
14. while(true){
15. int mid = low + (high - low) / 2;
16. if(testPicture.getColor(c,low).equals(testPicture.getColor(c,mid))){
17. low = mid;
18. }else{
19. high = mid;
20. }
21. if(high - low <= 1){
22. if(testPicture.getColor(c,0).equals(Color.white)){
23. len = testPicture.getHeight() - high;
24. break;
25. }else{
26. len = low + 1;
27. break;
28. }
29. }
30. }
31. if(len > max){
32. max = len;
33. }
34. }
35. System.out.println(max);
36. System.out.println(testPicture.getHeight());
37. }
38. public int getLength(Picture pic){
39. return 0;
40. }
41. }

## 任务3

1. */\*\**
2. \* 对objs[left..right]进行归并排序（公用temp数组版本）
3. \*
4. \* @param objs
5. \* @param left
6. \* @param right
7. \* @param temp
8. \*/
9. public void mergeSort(Comparable[] objs, int left, int right, Comparable[] temp){
10. if(right - left < 16){ *//取THRESHOLD=16*
11. Insertion insertionSort = new Insertion();
12. insertionSort.insertionSort(objs,left,right);
13. return;
14. }
15. int mid = (left + right) / 2;
16. mergeSort(objs, left, mid, temp);
17. mergeSort(objs, mid + 1, right, temp);
18. if (!less(objs[mid+1],objs[mid])){
19. return;
20. }
21. *//合并两个有序区间*
22. for (int i = left; i <= right; i++) {
23. temp[i] = objs[i];
24. }
25. *//这里给出了一种在递归函数中让某个内容“不递归”的方法：作为函数的参数传进去。*
26. *//比如这里的temp数组，是一个全局使用的数组，我们就可以不在函数中创建它，而是在调用处创建之后作为参数传入。*
27. *//数组全局使用有两个好处：*
28. *//1. 避免了将元素复制时产生的下标偏移问题，在调用时，取哪块用哪块*
29. *//2. 如果每一次归并都创建一个新的数组，此时创建和回收数组的性能开销会比较大*
30. int i = left;
31. int j = mid + 1;
32. for (int k = left; k <= right; k++) {
33. if(i == mid+1){
34. objs[k] = temp[j];
35. j++;
36. } else if (j == right + 1) {
37. objs[k] = temp[i];
38. } else if (less(temp[j],temp[i])){*//此处应考虑归并排序算法的稳定性，要使得两元素相等时i优先放入。*
39. objs[k] = temp[j];
40. j++;
41. }else{
42. objs[k] = temp[i];
43. i++;
44. }
45. }
46. }
47. *//归并排序的优化1：在小区间内使用插入排序*
48. *//归并排序的优化2：在归并两个有序数组之前检查是否满足“左最大>=右最小”*
49. */\*关于分治算法：*
50. 分解、解决、合并
51. 应用：归并排序、快速排序、树、回溯算法、动态规划（记忆化递归）
52. \*/
53. */\*\**
54. \* 将partition函数分离出来的快速排序算法
55. \* @param objs
56. \* @param low
57. \* @param high
58. \*/
59. public void quick(Comparable[] objs, int low, int high){
60. *//基准情形*
61. if (low >= high) {
62. return;
63. }
64. int pivotIndex = partition(objs, low, high);
65. quick(objs, low, pivotIndex-1);
66. quick(objs, pivotIndex+1, high);
67. }
68. */\*\**
69. \* 单路划分（两指针同向运动）
70. \* @param objs
71. \* @param low
72. \* @param high
73. \* @return
74. \*/
75. private int partition\_1(Comparable[] objs, int low, int high){
76. Comparable pivot = choosePivot(objs, low, high);
77. int i = low; *//i相当于是一个“栈”的top指针，指向目前已扫描到的最后一个比轴值小的元素的下一个位置。*
78. int j = low; *//j往右遍历搜索比轴值小的元素，找到后就放到i左边的那个“栈”里去*
79. *//刚开始时i和j在同一个位置上*
80. while(j < high){
81. if(less(objs[j],pivot)){ *//j找到了比轴值小的元素*
82. if(i != j){
83. exchange(objs,i,j);
84. }
85. *//当一个比轴值小的元素放到左边的“栈”中时，i就会向后移动一位。*
86. i++;
87. }
88. j++;
89. }
90. *//此时i指向的是比轴值小的元素的后一个元素（也就是第一个比轴值大的元素）*
91. exchange(objs,i,high);
92. *//将i指向的值和轴值交换位置，完成划分。*
93. return i;
94. *//返回轴值最后所在的位置。*
95. }
96. */\*\**
97. \* 普通双路划分（两指针相向运动）
98. \* @param objs
99. \* @param low
100. \* @param high
101. \* @return
102. \*/
103. public int partition\_2(Comparable[] objs, int low, int high){
104. Comparable pivot = choosePivot(objs, low, high);
105. int left = low;
106. int right = high - 1;
107. while(left < right){
108. *//其实只要让left指针和right指针碰面即可，就算交错了也必定会是交错到对方已经排过的领域的第一个值.*
109. *//下面的两个循环的顺序是有讲究的，left先动，最后指向的值是第一个大于轴值的值；right先动，最后指向的是最后一个小于轴值的值。*
110. while(left < right && less(objs[left], pivot)){ *//这两个循环中的left<right都是有必要的！*
111. left++;
112. }
113. while(left < right && less(pivot, objs[right])){ *//这两个循环中的比较都是严格小于，这样做可以使排序后等于pivot的元素平均划分到两个区域中，为下一次划分奠定基础*
114. right--;
115. }
116. exchange(objs,left,right);
117. }
118. exchange(objs,left,high);
119. return left;
120. }
121. private int partition(Comparable[] objs, int left, int right){
122. Comparable pivot = choosePivot(objs,left,right);
123. *//循环不变量：*
124. *//[left..i) < pivot*
125. *//(j..right-1] > pivot*
126. int i = left;
127. int j = right-1;
128. while(i <= j){
129. while(i<=j && less(objs[i],pivot)){ *// i找大于等于轴值的值，因此在小于轴值时要向右移动*
130. i++;
131. }
132. while(i<=j && less(pivot,objs[j])){ *// j找小于等于轴值的值，因此在大于轴值时要向左移动*
133. j--;
134. }
135. if(i<=j){
136. exchange(objs,i,j);
137. i++;
138. j--;
139. }
140. }
141. exchange(objs,i,right);
142. return i;
143. }

其他请参见主干代码分析部分

## 任务4

1. import org.jfree.chart.ChartFactory;
2. import org.jfree.chart.ChartPanel;
3. import org.jfree.chart.JFreeChart;
4. import org.jfree.chart.plot.PlotOrientation;
5. import org.jfree.chart.plot.XYPlot;
6. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
7. import org.jfree.chart.ui.ApplicationFrame;
8. import org.jfree.chart.ui.RectangleInsets;
9. import org.jfree.data.xy.XYDataset;
10. import org.jfree.data.xy.XYSeries;
11. import org.jfree.data.xy.XYSeriesCollection;
12. import java.awt.\*;
13. public class LineXYDemo extends ApplicationFrame {
14. *// 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*
15. public LineXYDemo(String title){
16. super(title);
17. XYDataset dataset = createDataset();             *// 创建记录图中坐标点的数据集*
18. 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*
19. 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*
20. *// 设置GUI面板Panel的显示大小*
21. chartPanel.setPreferredSize(new Dimension(500, 270));
22. 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。*
23. }
24. private JFreeChart createChart(XYDataset dataset) {
25. *// 使用已经创建好的dataset生成图表对象*
26. *// 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*
27. JFreeChart chart = ChartFactory.createXYLineChart(
28. "Comparison of sorting algorithm performance(Repetitive Data)",      *// 图表的标题*
29. "Data Repetitiveness",                           *// 横轴的标题名*
30. "running time",                           *// 纵轴的标题名*
31. dataset,                       *// 图表对象中使用的数据集对象*
32. PlotOrientation.VERTICAL,      *// 图表显示的方向*
33. true,                          *// 是否显示图例*
34. false,                         *// 是否需要生成tooltips*
35. false                          *// 是否需要生成urls*
36. );
37. *// 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*
38. *// 同学们不必在意下面的代码*
39. *// 可以将下面的代码去掉对比一下显示的不同效果*
40. chart.setBackgroundPaint(Color.WHITE);
41. XYPlot plot = (XYPlot)chart.getPlot();
42. plot.setBackgroundPaint(Color.lightGray);
43. plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 6.0));
44. plot.setDomainGridlinePaint(Color.WHITE);
45. plot.setRangeGridlinePaint(Color.WHITE);
46. XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer();
47. renderer.setDefaultShapesVisible(true);
48. renderer.setDefaultShapesFilled(true);
49. return chart;
50. }
51. private double[][] ArrayLog10(double[][] rawdata){
52. int row = rawdata.length;
53. int column = rawdata[0].length;
54. double[][] result = new double[row][column];
55. for (int i = 0; i < row; i++) {
56. for(int j = 0; j < column; j++){
57. result[i][j] = Math.log10(rawdata[i][j]);
58. }
59. }
60. return result;
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. double[] X = {50, 60, 80, 100};
66. double[] Ysr = {646480.0000,810360.0000,5216780.0000,3451820.0000,13326260.0000,53072580.0000,236708440.0000,1101155780.0000,4808753200.0000};
67. double[] Yss = {673840.0000,815200.0000,3853560.0000,1939400.0000,7704600.0000,31027140.0000,128925740.0000,693407720.0000,2670950720.0000};
68. double[] Ysi = {880000.0000,844980.0000,7165340.0000,3632840.0000,14365020.0000,58608360.0000,252736780.0000,1084954120.0000,4502642480.0000};
69. double[] Ysn = {709960.0000,902620.0000,7810680.0000,8972540.0000,19912840.0000,86534640.0000,498192580.0000,2317368140.0000,13764761740.0000};
70. double[] Yir = {386280.0000,898000.0000,1149020.0000,1370900.0000,3291480.0000,14115640.0000,53964540.0000,316715600.0000,1586506900.0000};
71. double[] Yis = {43180.0000,43720.0000,66320.0000,125640.0000,276080.0000,421140.0000,250280.0000,237340.0000,302640.0000};
72. double[] Yii = {764640.0000,1783020.0000,1127500.0000,3463940.0000,6567880.0000,26136200.0000,108678180.0000,495460960.0000,3352051980.0000};
73. double[] Yin = {222700.0000,438700.0000,1159760.0000,1667620.0000,5377860.0000,12731920.0000,35900020.0000,156989740.0000,733090540.0000};
74. double[] Yshr = {175020.0000,148880.0000,361760.0000,322840.0000,784160.0000,982520.0000,2201940.0000,2746840.0000,9321320.0000};
75. double[] Yshs = {170160.0000,183120.0000,370640.0000,405920.0000,191160.0000,442780.0000,320280.0000,644640.0000,1467720.0000};
76. double[] Yshi = {141660.0000,134840.0000,268200.0000,355320.0000,886880.0000,917120.0000,1243420.0000,1090480.0000,2299960.0000};
77. double[] Yshn = {173480.0000,119220.0000,227160.0000,488580.0000,632280.0000,1114120.0000,971400.0000,1399820.0000,3818280.0000};
78. double[] Ymr = {174420.0000,87320.0000,142320.0000,266360.0000,769800.0000,2113820.0000,13971980.0000,17393940.0000,6029860.0000};
79. double[] Yms = {272340.0000,125000.0000,186920.0000,325680.0000,770900.0000,2332820.0000,12997100.0000,8645920.0000,6153220.0000};
80. double[] Ymi = {293540.0000,179160.0000,348220.0000,682320.0000,833840.0000,3623400.0000,11007900.0000,14592100.0000,6858920.0000};
81. double[] Ymn = {299440.0000,188580.0000,446000.0000,706540.0000,1305500.0000,6132160.0000,10919980.0000,6879100.0000,6172660.0000};
82. double[] Yqr = {201260.0000,72520.0000,135620.0000,263440.0000,1036060.0000,1405140.0000,2822540.0000,5558600.0000,3651920.0000};
83. double[] Yqs = {229600.0000,73060.0000,205320.0000,265880.0000,538740.0000,1166500.0000,4042660.0000,6187060.0000,3265560.0000};
84. double[] Yqi = {214460.0000,111620.0000,240120.0000,968260.0000,3171600.0000,9499700.0000,4730680.0000,18094100.0000,72704400.0000};
85. double[] Yqn = {241980.0000,134880.0000,168280.0000,311020.0000,1216200.0000,1576560.0000,2813200.0000,6417160.0000,8831980.0000};
86. double[] Yqur = {338080.0000,100760.0000,107120.0000,182340.0000,708040.0000,1132960.0000,1726960.0000,8226760.0000,4730540.0000};
87. double[] Yqus = {205060.0000,146660.0000,81340.0000,132840.0000,413460.0000,597480.0000,1273940.0000,5646500.0000,5287160.0000};
88. double[] Yqui = {183880.0000,147940.0000,250580.0000,985520.0000,3103540.0000,9707160.0000,4694080.0000,18359560.0000,72049020.0000};
89. double[] Yqun = {225360.0000,144640.0000,199820.0000,249480.0000,1009000.0000,1080220.0000,2061620.0000,10593260.0000,5792220.0000};
90. double[] Yqu = {12475340.0000, 11279740.0000, 11530620.0000, 8851100.0000};
91. double[] Yqt = {960.0000, 1000.0000, 1060.0000, 1040.0000};
92. *//        double[][] Y\_raw = {Yqr, Yqur};*
93. *//        double[][] Y = ArrayLog10(Y\_raw);*
94. double[][] Y = {Yqu, Yqt};
95. *// 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*
96. XYSeries[] series = {new XYSeries("quick\_update"), new XYSeries("quick\_triple")};
97. *//        XYSeries[] series = {new XYSeries("Selection"), new XYSeries("Insertion"), new XYSeries("Merge"), new XYSeries("Quick"), new XYSeries("Shell")};*
98. int N = X.length;
99. int M = series.length;
100. for(int i = 0; i < M; i++)
101. for(int j = 0; j < N; j++)
102. series[i].add(X[j], Y[i][j]);
103. *// 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*
104. XYSeriesCollection dataset = new XYSeriesCollection();
105. for(int i = 0; i < M; i++)
106. dataset.addSeries(series[i]);
107. return dataset;
108. }
109. public static void main(String[] args) {
110. LineXYDemo demo = new LineXYDemo("Comparison of sorting algorithm performance");
111. demo.pack();
112. demo.setVisible(true);
113. }
114. }

## 任务5

1. public static Double[] getNovelData(int N){
2. Double[] numbers = new Double[N];
3. for(int i = 0; i < N/2; i++){
4. numbers[i] = 0.0;
5. }
6. for(int i = N/2; i < 3 \* N / 4; i++){
7. numbers[i] = 1.0;
8. }
9. for(int i = 3 \* N / 4; i < 7 \* N / 8; i++){
10. numbers[i] = 2.0;
11. }
12. for(int i = 7 \* N / 8; i < N; i++){
13. numbers[i] = 3.0;
14. }
15. shuffle(numbers, 0, numbers.length - 1);
16. return numbers;
17. }

## 任务6

1. private void quickSortTriple\_rightPivot(Comparable[] objs, int left, int right){
2. if(left <= right){
3. return;
4. }
5. Comparable pivot = choosePivot(objs,left,right);
6. *//循环不变量*
7. *//[left..lt) < pivot*
8. *//[lt..i) == pivot*
9. *//(gt..right-1] >pivot*
10. int lt = left;
11. int gt = right - 1;
12. int i = left;
13. while(i <= gt){
14. if(less(objs[i],pivot)){
15. exchange(objs, lt, i);
16. lt++;
17. i++;
18. } else if (less(pivot, objs[i])) {
19. exchange(objs, gt, i);
20. gt--;
21. } else {
22. i++;
23. }
24. }
25. *//轴值和第一个比轴值大的元素交换*
26. exchange(objs,right, gt+1);
27. quickSortTriple\_rightPivot(objs,left,lt-1);
28. quickSortTriple\_rightPivot(objs,gt+2,right);
29. }
30. public Comparable kthsmallest\_double(Comparable[] objs, int k){
31. int len = objs.length;
32. int target = k - 1;
33. int left = 0;
34. int right = len - 1;
35. while(true){
36. int pivotIndex = partition(objs, left, right);
37. if(pivotIndex == target){
38. return objs[pivotIndex];
39. } else if (pivotIndex < target) {
40. left = pivotIndex + 1;
41. }else {
42. right = pivotIndex - 1;
43. }
44. }
45. }
46. private int partition(Comparable[] objs, int left, int right){
47. Comparable pivot = choosePivot(objs,left,right);
48. *//循环不变量：*
49. *//[left..i) < pivot*
50. *//(j..right-1] > pivot*
51. int i = left;
52. int j = right-1;
53. while(i <= j){
54. while(i<=j && less(objs[i],pivot)){
55. *// i找大于等于轴值的值，因此在小于轴值时要向右移动*
56. i++;
57. }
58. while(i<=j && less(pivot,objs[j])){
59. *// j找小于等于轴值的值，因此在大于轴值时要向左移动*
60. j--;
61. }
62. if(i<=j){
63. exchange(objs,i,j);
64. i++;
65. j--;
66. }
67. }
68. exchange(objs,i,right);
69. return i;
70. }
71. public Comparable kthsmallest\_triple(Comparable[] objs, int k, int left, int right){
72. *// base case*
73. if ( right - left <= 1 ){
74. if ( left == right ){
75. return objs[left];
76. } else {
77. if (less(objs[right],objs[left])){
78. exchange(objs, left, right);
79. }
80. if ( k==1 ){
81. return objs[left];
82. } else {
83. return objs[right];
84. }
85. }
86. }
87. Comparable pivot = choosePivot(objs,left,right);
88. *//循环不变量*
89. *//[left..lt) < pivot*
90. *//[lt..i) == pivot*
91. *//(gt..right-1] >pivot*
92. int lt = left;
93. int gt = right - 1;
94. int i = left;
95. while(i <= gt){
96. if(less(objs[i],pivot)){
97. exchange(objs, lt, i);
98. lt++;
99. i++;
100. } else if (less(pivot, objs[i])) {
101. exchange(objs, gt, i);
102. gt--;
103. } else {
104. i++;
105. }
106. }
107. *//轴值和第一个比轴值大的元素交换*
108. exchange(objs,right, gt+1);
109. *// 每轮排序完成后，[left, lt-1] < pivot, [gt+2, right] > pivot*
110. if (k-1 <= lt-left-1){
111. return kthsmallest\_triple(objs,k,left,lt-1);
112. } else if (k-1 > lt-left-1 && k-1 < gt-left+2) {
113. return pivot;
114. } else {
115. return kthsmallest\_triple(objs, k-gt-2, gt+2, right);
116. }
117. }