|  |  |
| --- | --- |
| **数据结构与算法 作业报告** | |
| 第二次 | |
| 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** | 2023-12-21 |

**目录**

[任务2 3](#_Toc152021492)

[题目 3](#_Toc152021493)

[数据设计 3](#_Toc152021494)

[算法设计 3](#_Toc152021495)

[主干代码说明 4](#_Toc152021496)

[运行结果展示 4](#_Toc152021497)

[总结和收获 5](#_Toc152021498)

[任务3 5](#_Toc152021499)

[题目 5](#_Toc152021500)

[数据设计 5](#_Toc152021501)

[算法设计 5](#_Toc152021502)

[主干代码说明 7](#_Toc152021503)

[运行结果展示（仅选一种算法测试结果展示） 12](#_Toc152021504)

[总结和收获 12](#_Toc152021505)

[任务4 13](#_Toc152021506)

[题目 13](#_Toc152021507)

[数据设计 13](#_Toc152021508)

[算法设计 13](#_Toc152021509)

[主干代码说明 13](#_Toc152021510)

[运行结果展示 17](#_Toc152021511)

[规律阐述 18](#_Toc152021512)

[任务5 19](#_Toc152021513)

[题目：数据分布对排序算法的影响 19](#_Toc152021514)

[数据设计 19](#_Toc152021515)

[算法设计 19](#_Toc152021516)

[主干代码说明 19](#_Toc152021517)

[运行结果展示 20](#_Toc152021518)

[规律阐述 21](#_Toc152021519)

[任务6 21](#_Toc152021520)

[题目：快速排序的再探讨和应用 21](#_Toc152021521)

[优化快速排序 22](#_Toc152021522)

[重复性元素排序 22](#_Toc152021523)

[三路划分算法设计 22](#_Toc152021524)

[三路划分主干代码说明 23](#_Toc152021525)

[找出第k小的元素：方法探索 25](#_Toc152021526)

[找出第k小的元素：快速排序思想、分治法的实现 26](#_Toc152021527)

[找出第k小的元素：算法分析 27](#_Toc152021528)

[规律阐述 27](#_Toc152021529)

[附录：每个题的源代码 28](#_Toc152021530)

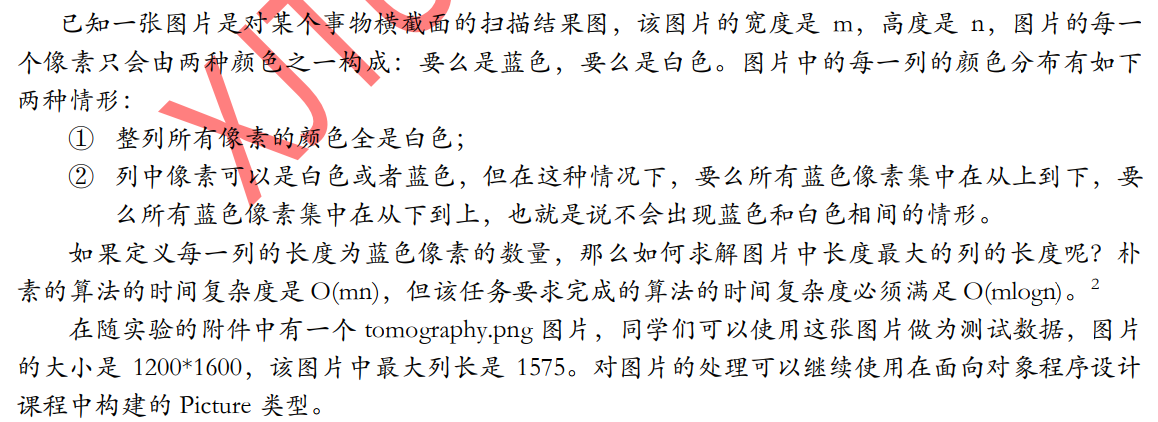
[任务四 28](#_Toc152021531)

[任务五 32](#_Toc152021532)

[任务六： 32](#_Toc152021533)

# 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

## 运行结果展示

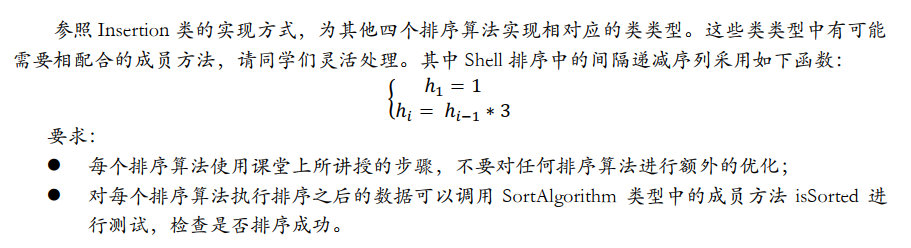
## 总结和收获

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：**

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

主函数中写了一些辅助测试代码：

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. System.out.print(test[i] + " ");
7. }
8. System.out.println();
9. SortAlgorithm alg = new Quick();
10. alg.sort(testData);
11. for (int i = 0; i < 5; i++) {
12. System.out.print(testData[i] + " ");
13. }
14. System.out.println(test.equals(testData));
15. System.out.println(alg.isSorted(testData));
16. }

这是快速排序的主干代码：

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

基准情形

1. if (low >= high) {
2. return;
3. }

选择轴值

1. int mid = low + (high - low) / 2;
2. Comparable pivot = objs[high];
3. if (!(less(objs[low], objs[high]) ^ less(objs[mid], objs[low]))) {
4. pivot = objs[low];
5. exchange(objs, low, high);
6. } else if (!(less(objs[mid], objs[low]) ^ less(objs[high], objs[mid]))) {
7. pivot = objs[mid];
8. exchange(objs, mid, high);
9. }

双指针法的操作和递归

1. int left = low;
2. int right = high - 1;
3. while (true) {
4. while (less(objs[left], pivot)) {
5. left++;
6. }
7. while (less(pivot, objs[right]) && right > 0) {
8. right--;
9. }
10. if (left < right) {
11. exchange(objs, left, right);
12. } else {
13. exchange(objs, left, high);
14. if (left > 1) {
15. quickSort(objs, low, left - 1);
16. }
17. quickSort(objs, left + 1, high);
18. return;

一定不要忘记这个return，否则会一直循环退不出去！

1. }
2. }
3. }
4. }

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）



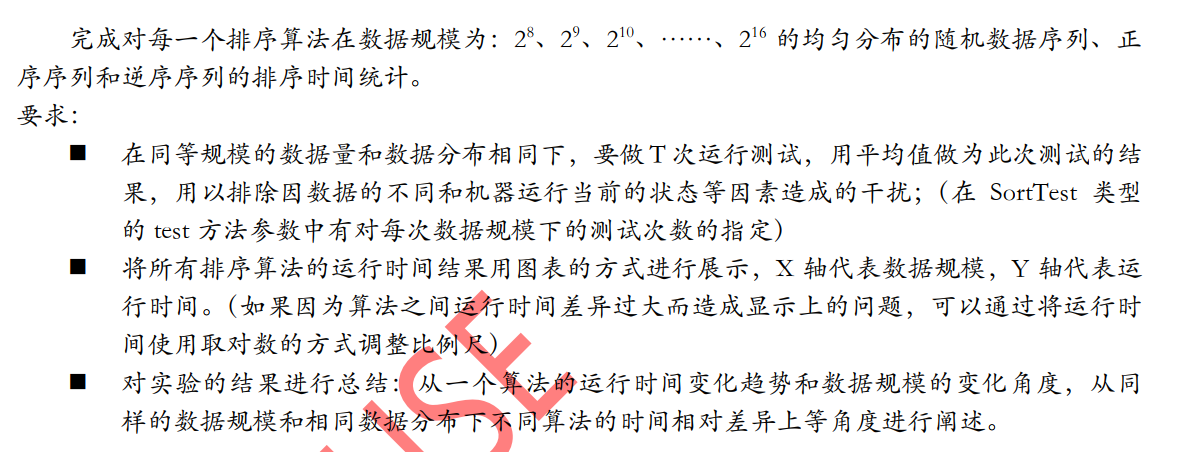
## 总结和收获

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

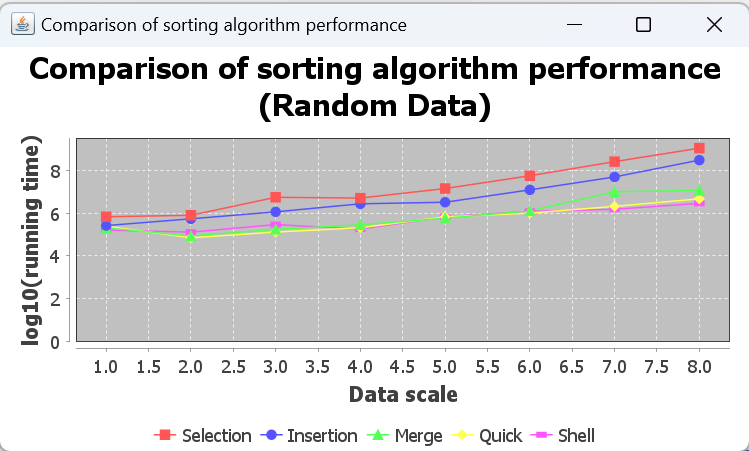
通过使用给定的画图模板来利用jfreechart画图

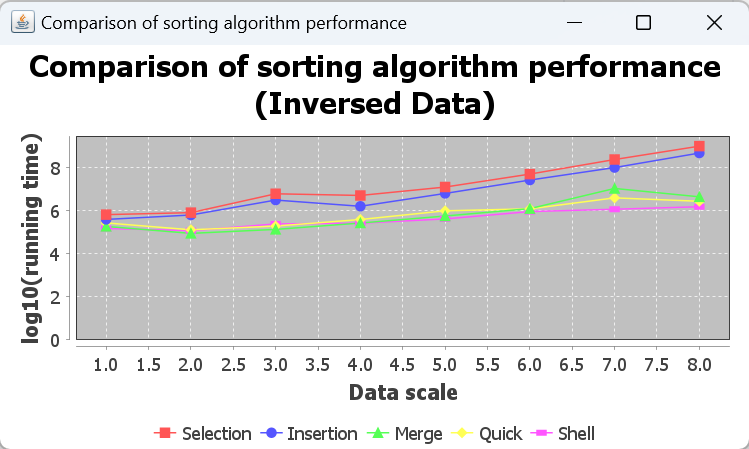
1. import org.jfree.chart.ChartFactory;
2. import org.jfree.chart.ChartPanel;
3. import org.jfree.chart.JFreeChart;
4. import org.jfree.chart.plot.PlotOrientation;
5. import org.jfree.chart.plot.XYPlot;
6. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
7. import org.jfree.chart.ui.ApplicationFrame;
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*

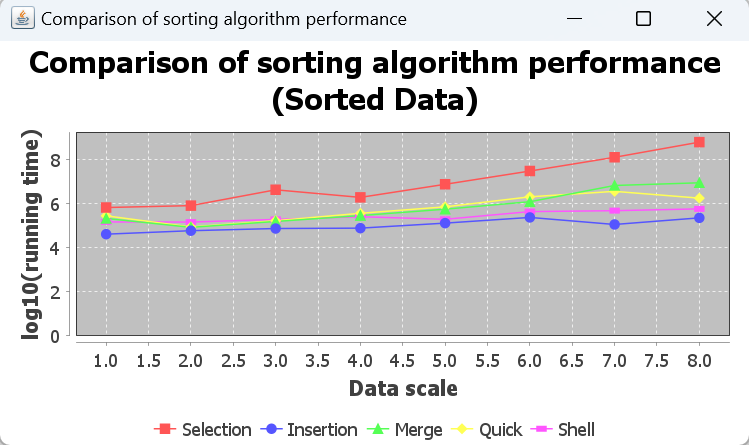
此处输入数据：

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。

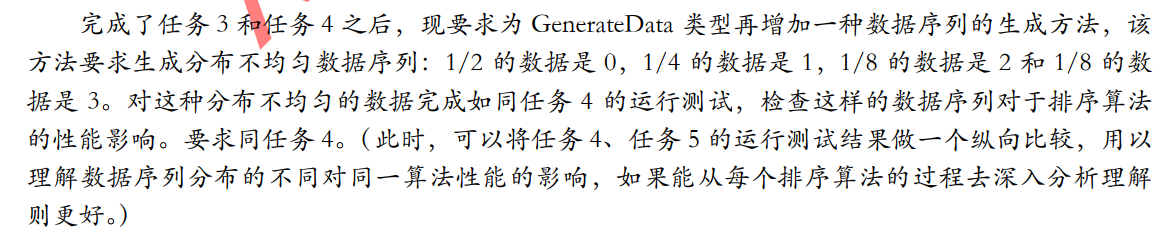
归并排序：时间复杂度为O(nlogn)，比较次数和数据交换次数均较多。归并排序是一种基于分治思想的排序算法，通过将待排序的数列分成若干个子序列，然后对每个子序列进行递归处理，最终合并得到有序的序列。归并排序在处理数据时，需要将数据进行分解和合并操作，因此需要额外的空间存储临时数据。但是，归并排序具有较好的稳定性，即相同值的元素在排序后保持原有的相对顺序。

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. }

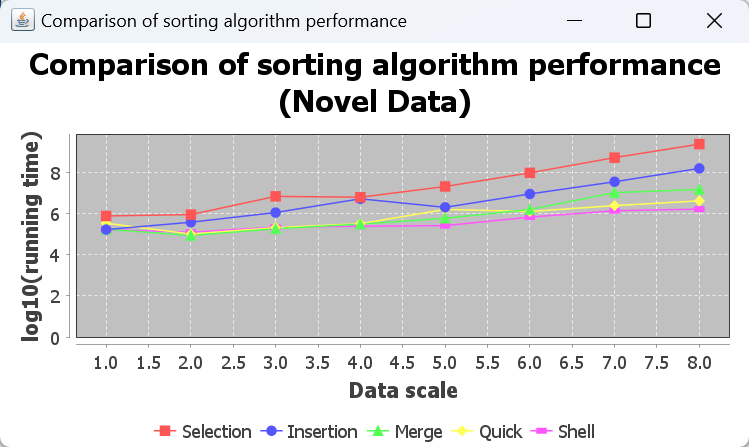
进行赋值：

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

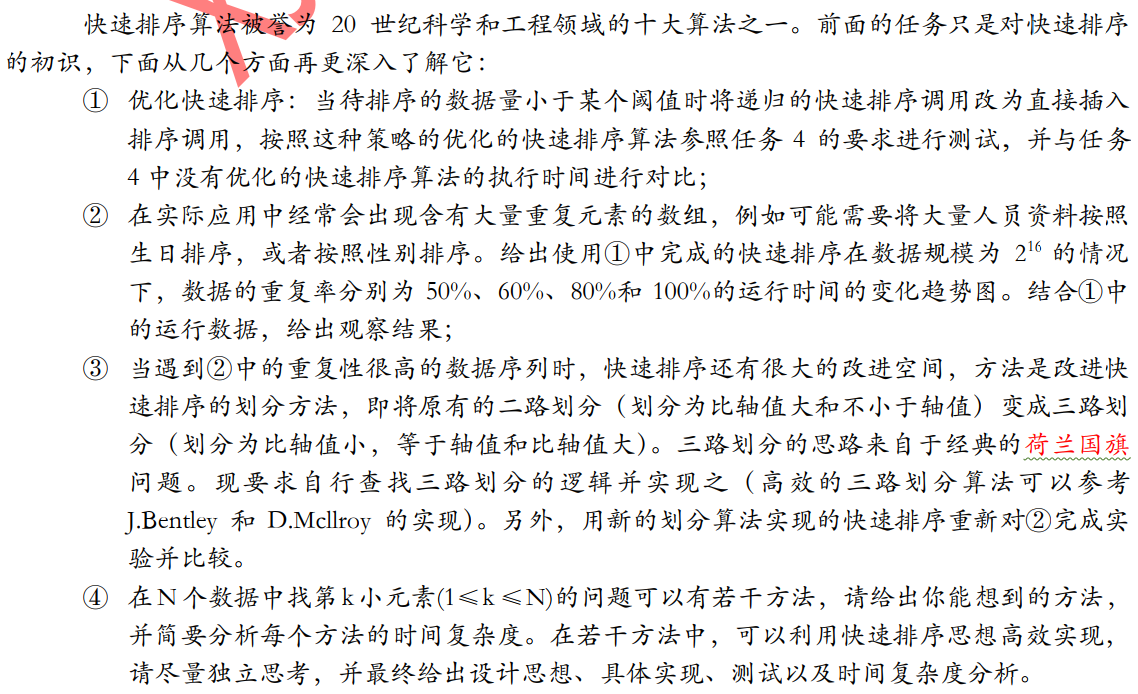


## 规律阐述

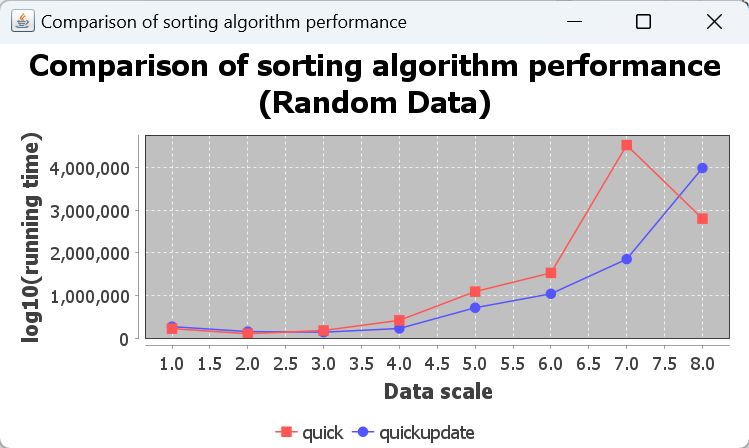
在不同数据分布下，每个排序算法的表现都会有所变化。分布不均匀数据可能对插入排序、选择排序等有较大影响，而对于快速排序、归并排序等影响相对较小。每个算法对数据分布的敏感性不同，这种比较有助于选择适当的算法来处理特定类型的数据。

# 任务6

## 题目：快速排序的再探讨和应用



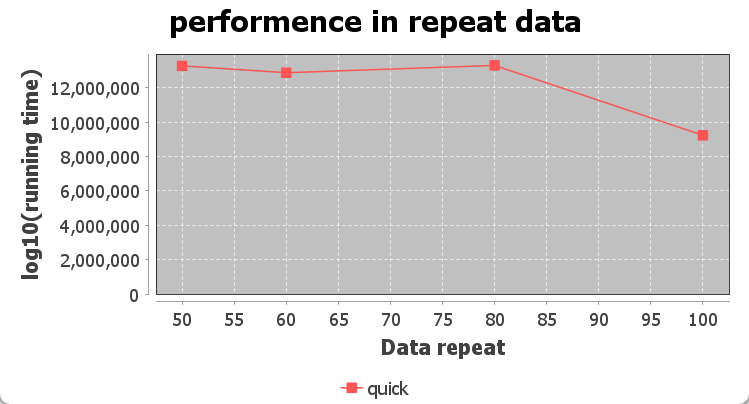
## 优化快速排序



从图中可见，优化后的快速排序算法在较大范围内都很有优势。

## 重复性元素排序

由于操作步骤与前面所述的算法测试大致相同，在此不再赘述。



## 三路划分算法设计

参考资料，通过四个指针实现。具体细节参见主干代码说明。

## 三路划分主干代码说明

1. public class quickSortTriple  extends SortAlgorithm {
2. public void sort(Comparable[] objs) {
3. int N = objs.length;
4. quickSortTriple(objs, 0, N - 1);
5. }
6. private void quickSortTriple(Comparable[] objs, int left, int right) {
7. if (right <= left)
8. return;
9. int p, q, i, j;
10. Comparable pivot;*// 锚点*
11. i = p = left;
12. j = q = right - 1;

每次总是取序列最右边的元素为锚点

1. pivot = objs[right];
2. while (true) {

工作指针i从右向左不断扫描，找小于或者等于锚点元素的元素

1. while (i < right && less(objs[i],pivot)) {
2. */\**
3. \* 找到与锚点元素相等的元素将其交换到p所指示的位置
4. \*/
5. if (objs[i] == pivot) {
6. exchange(objs,i,p);
7. p++;
8. }
9. i++;
10. }
11. */\**
12. \* 工作指针j从左向右不断扫描，找大于或者等于锚点元素的元素
13. \*/
14. while (left <= j && less(pivot,objs[j])) {
15. */\**
16. \* 找到与锚点元素相等的元素将其交换到q所指示的位置
17. \*/
18. if (objs[j] == pivot) {
19. exchange(objs,j,q);
20. q--;
21. }
22. j--;
23. }
24. */\**
25. \* 如果两个工作指针i j相遇则一趟遍历结束
26. \*/
27. if (i >= j)
28. break;
29. */\**
30. \* 将左边大于pivot的元素与右边小于pivot元素进行交换
31. \*/
32. exchange(objs,i,j);
33. i++;
34. j--;
35. }
36. */\**

因为工作指针i指向的是当前需要处理元素的下一个元素

 故而需要退回到当前元素的实际位置，然后将等于pivot元素交换到序列中间

1. \*/
2. i--;
3. p--;
4. while (p >= left) {
5. exchange(objs,i,p);
6. i--;
7. p--;
8. }
9. */\**

 因为工作指针j指向的是当前需要处理元素的上一个元素

    故而需要退回到当前元素的实际位置，然后将等于pivot元素交换到序列中间

1. \*/
2. j++;
3. q++;
4. while (q <= right) {
5. exchange(objs, j, q);
6. j++;
7. q++;
8. }

         递归遍历左右子序列

1. quickSortTriple(objs, left, i);
2. quickSortTriple(objs, j, right);
3. }
4. }

很遗憾的是，优化后的快速排序算法在数据量大时会出现StackOverflow错误，不能得出正常的运行时间，目前正在寻找解决办法，我会加倍努力，在一天之内找到解决办法！

## 找出第k小的元素：方法探索

排序法：将数据按照从小到大的顺序排序，然后返回第k个元素。这种方法的时间复杂度取决于排序算法的复杂度，一般为O (n log n)。

堆法：构建一个最小堆，然后依次删除堆顶元素k次，最后返回堆顶元素。这种方法的时间复杂度为O (n + k log n)，其中O (n)是建堆的复杂度，O (k log n)是删除k次堆顶的复杂度。

分治法：利用快速排序的思想，随机选择一个基准元素，将数据划分为两部分，左边的元素都小于等于基准，右边的元素都大于等于基准。然后根据基准的位置和k的大小，递归地在左边或右边的部分继续查找第k小元素。这种方法的平均时间复杂度为O (n)，最坏情况为O (n2)。

## 找出第k小的元素：快速排序思想、分治法的实现

1. *// 寻找一个序列中第k小的元素*
2. public class KthSmallest {
3. *// 划分算法，返回基准元素的位置*
4. public static int partition(int[] arr, int left, int right) {
5. int pivot = arr[left]; *// 选择第一个元素作为基准*
6. int i = left, j = right;
7. while (i < j) {
8. *// 从右往左找到第一个小于等于基准的元素*
9. while (i < j && arr[j] > pivot) {
10. j--;
11. }
12. *// 将该元素放到左边*
13. if (i < j) {
14. arr[i] = arr[j];
15. i++;
16. }
17. *// 从左往右找到第一个大于等于基准的元素*
18. while (i < j && arr[i] < pivot) {
19. i++;
20. }
21. *// 将该元素放到右边*
22. if (i < j) {
23. arr[j] = arr[i];
24. j--;
25. }
26. }
27. *// 将基准元素放到最终位置*
28. arr[i] = pivot;
29. return i;
30. }
31. *// 查找算法，返回第k小的元素*
32. public static int findKthSmallest(int[] arr, int left, int right, int k) {
33. *// 如果只有一个元素，直接返回*
34. if (left == right) {
35. return arr[left];
36. }
37. *// 划分数组，得到基准元素的位置*
38. int p = partition(arr, left, right);
39. *// 如果基准元素的位置刚好是k-1，说明找到了第k小的元素*
40. if (p == k - 1) {
41. return arr[p];
42. }
43. *// 如果基准元素的位置大于k-1，说明第k小的元素在左边部分*
44. else if (p > k - 1) {
45. return findKthSmallest(arr, left, p - 1, k);
46. }
47. *// 如果基准元素的位置小于k-1，说明第k小的元素在右边部分*
48. else {
49. return findKthSmallest(arr, p + 1, right, k - p - 1);
50. }
51. }
52. *// 测试*
53. public static void main(String[] args) {
54. int[] arr = {5, 3, 7, 2, 9, 1, 4, 6, 8};
55. int k = 4;
56. System.out.println("第" + k + "小的元素是：" + findKthSmallest(arr, 0, arr.length - 1, k));
57. }
58. }

## 找出第k小的元素：算法分析

最好情况：每次划分都能将数组均匀地分为两部分，那么递归的深度为log n，每次划分的时间为O (n)，所以总的时间复杂度为O (n log n)。

平均情况：每次划分都能将数组分为两个长度为n/10和9n/10的部分，那么递归的深度为O (log n)，每次划分的时间为O (n)，所以总的时间复杂度为O (n)。

最坏情况：每次划分都将数组分为一个元素和n-1个元素的两部分，那么递归的深度为O (n)，每次划分的时间为O (n)，所以总的时间复杂度为O (n2)。

## 规律阐述

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。

# 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

参见主干代码分析部分

## 任务四

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*
65. double[] X = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
66. double[] Y1 = {698440.0000,817360.0000,5733380.0000,5270680.0000,14846220.0000,59159340.0000,273607600.0000,1177341760.0000,4901339440.0000};
67. double[] Y2 = {268400.0000,558500.0000,1187359.8000,2837539.6000,3360580.0000,12852500.2000,52058420.0000,321808119.8000,1413004639.8000};
68. double[] Y3 = {210940.0000,86920.0000,178140.0000,293240.0000,596619.8000,1314800.2000,10481939.8000,12115399.8000,6555679.6000};
69. double[] Y4 = {251300.0000,72000.0000,133200.0000,212540.0000,690520.0000,1040740.0000,2148280.0000,4855980.0000,10460420.0000};
70. double[] Y5 = {163880.0000,132240.0000,298420.0000,184160.0000,694540.0000,1258120.0000,1564240.0000,2983220.0000,6451060.0000};
71. double[][] Y\_raw = {Y1, Y2, Y3, Y4, Y5};
72. double[][] Y = ArrayLog10(Y\_raw);
73. *// 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*
74. XYSeries[] series = {new XYSeries("Selection"), new XYSeries("Insertion"), new XYSeries("Merge"), new XYSeries("Quick"), new XYSeries("Shell")};
75. int N = X.length;
76. int M = series.length;
77. for(int i = 0; i < M; i++)
78. for(int j = 0; j < N; j++)
79. series[i].add(X[j], Y[i][j]);
80. *// 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*
81. XYSeriesCollection dataset = new XYSeriesCollection();
82. for(int i = 0; i < M; i++)
83. dataset.addSeries(series[i]);
84. return dataset;
85. }
86. public static void main(String[] args) {
87. LineXYDemo demo = new LineXYDemo("Comparison of sorting algorithm performance");
88. demo.pack();
89. demo.setVisible(true);
90. }
91. }

## 任务五

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. }

## 任务六：

1. *// 寻找一个序列中第k小的元素*
2. public class KthSmallest {
3. *// 划分算法，返回基准元素的位置*
4. public static int partition(int[] arr, int left, int right) {
5. int pivot = arr[left]; *// 选择第一个元素作为基准*
6. int i = left, j = right;
7. while (i < j) {
8. *// 从右往左找到第一个小于等于基准的元素*
9. while (i < j && arr[j] > pivot) {
10. j--;
11. }
12. *// 将该元素放到左边*
13. if (i < j) {
14. arr[i] = arr[j];
15. i++;
16. }
17. *// 从左往右找到第一个大于等于基准的元素*
18. while (i < j && arr[i] < pivot) {
19. i++;
20. }
21. *// 将该元素放到右边*
22. if (i < j) {
23. arr[j] = arr[i];
24. j--;
25. }
26. }
27. *// 将基准元素放到最终位置*
28. arr[i] = pivot;
29. return i;
30. }
31. *// 查找算法，返回第k小的元素*
32. public static int findKthSmallest(int[] arr, int left, int right, int k) {
33. *// 如果只有一个元素，直接返回*
34. if (left == right) {
35. return arr[left];
36. }
37. *// 划分数组，得到基准元素的位置*
38. int p = partition(arr, left, right);
39. *// 如果基准元素的位置刚好是k-1，说明找到了第k小的元素*
40. if (p == k - 1) {
41. return arr[p];
42. }
43. *// 如果基准元素的位置大于k-1，说明第k小的元素在左边部分*
44. else if (p > k - 1) {
45. return findKthSmallest(arr, left, p - 1, k);
46. }
47. *// 如果基准元素的位置小于k-1，说明第k小的元素在右边部分*
48. else {
49. return findKthSmallest(arr, p + 1, right, k - p - 1);
50. }
51. }
52. *// 测试*
53. public static void main(String[] args) {
54. int[] arr = {5, 3, 7, 2, 9, 1, 4, 6, 8};
55. int k = 4;
56. System.out.println("第" + k + "小的元素是：" + findKthSmallest(arr, 0, arr.length - 1, k));
57. }
58. }