数据结构与算法
作业报告

第一次

姓名

班级

学号

电话

Email

日期

目录

任务 1
1、 题目 2
2、 解答过程 3
3、 总结 4
任务 2
1、 题目 5
2、 程序设计及代码说明 5
3、 运行结果展示 8
4、 总结 8
任务 3
1、 题目 9
2、 程序设计及代码说明 9
3、 运行结果展示 14
4、 总结 15
任务 4
1、 题目 16
2、 程序设计及代码说明 16
3、 运行结果展示 18
4、 总结 20
任务 5
1、 题目 21
2、 程序设计及代码说明 21
3、 运行结果展示 24
4、 总结 27
任务 6
1、 题目 29
2、 程序设计及代码说明 29
3、 运行结果展示 35
4、 总结和收获 40

任务 1 证明

1) 使用 O、Ω和Θ的定义，证明下面每一个等式：
   a) $2\sqrt{n}+6=O(\sqrt{n})$
   b) $n^2=\Omega(n)$
   c) $log_2(n)=\Theta(\ln(n))$
   d) $4^n \neq O(2^n)$
2) 使用数学归纳法证明$T(n)=\Omega(nlog(n))$。T(n)的定义式如下：

$$T(n)=\begin{cases}2T\left(\left\lceil\frac{n}{2}\right\rceil\right)+\frac{n}{2}, & n>1\\ 1, & n=1\end{cases}$$

解答过程:

1)

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

证：设 $T(n)=2\sqrt{n}+6$，取 $c=8$，$n_0=1$，$f(n)=\sqrt{n}$

则对任意 $n>n_0$，有 $T(n)\le c\,f(n)$，即 $2\sqrt{n}+6\le 8\sqrt{n}\Rightarrow \sqrt{n}\ge 1$

则 $T(n)$ 在集合 $O(f(n))$ 中，即 $2\sqrt{n}+6=O(\sqrt{n})$

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

证：设 $T(n)=n^2$，取 $c=1$，$n_0=1$，~~$f(n)$~~ $g(n)=n$

则 $n>n_0$ 时，$T(n)-cg(n)=n^2-n=n(n-1)=n(n-n_0)>0$

故 $T(n)\ge cg(n)$，即 $n^2\ge n$，即 $n^2=\Omega(n)$

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

证：由定义知 $\log_2(n)=O(\log_2(n))$，即 $\log_2(n)=c(\log_2(n))$，c 为常数

当 $c=\log_e 2$ 时，$\log_2(n)=\log_e 2\cdot\log_2(n)=\log_e(n)=\ln(n)$

故 $\log_2(n)=O(\ln(n))$

同理 $\log_2(n)=\Omega(\ln(n))$

故 $\log_2(n)=\Theta(\ln(n))$

2) 证明 $T(n)=\Omega(n\log(n))$，$T(n)=\begin{cases}2T(\lfloor\frac{n}{2}\rfloor)+\frac{n}{2}, & n>1\\ 1, & n=1\end{cases}$

证：①当 $n=1$ 时：

$T(1)=1$，$n\log(n)=0$，$T(1)=\Omega(0)$

②当 $n>1$ 时：

假设取 $\frac{n}{2}$ 时等式成立，即 $T(\frac{n}{2})=\Omega(n\log(n))$

则 $T(n)=2T(\frac{n}{2})+\frac{n}{2}=2n\log(n)+\frac{n}{2}$

由于下限与常数无关，且 $n>1$ 时 $n\log(n)>n$.

则 $T(n)=\Omega(n\log(n))$

综上所述：$T(n)=\Omega(n\log(n))$

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

证：设 $f(n)=2^n$，则 $f^2(n)=(2^n)^2=4^n$

显然 $f^2(n)\neq O(f(n))$

故 $4^n\neq O(2^n)$

总结:

在对时间复杂度的学习中，应熟练掌握平均时间复杂度、最坏情况时间复杂度、最好情况时间复杂度相关内容的知识，三者分别以θ符号、大 O 符号、Ω符号表示，且上述三者在计算中可直接忽略除增长速度最快的项的所有项的项、增长速度最快的项的常数系数等，也应熟练掌握计算某程序时间复杂度的方法；对于数学归纳法，应熟记通常先讨论 n=1 的情形是否成立，再在假设取值 n-1 的情况下依旧成立时证明取值为 n 的情况下成立。综上所述可得所需证明的式子是成立的。

任务 2 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(mlogn)。[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*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。

设计：

一．编写 Search{}类：

编写 main（）函数（应加上 throws IOException 异常处理以保证程序正常运行）：

1. 使用 Picture 类的有参构造创建 Picture 类对象 picture，参数为需要查找的图片名字，即“tomography.png”；

2. 开始编写二分查找的代码

（1）创建 int 类型参数 max，用于记录已经循环过的列蓝色像素最长的值；

（2）使用 for 循环，次数为图片宽度像素次，即 picture.getWidth 次，用于从左至右依次调取图片的每一列；

（3）For 循环内，创建两个 int 类型参数，分别为 firstColor、lastColor，分别用于获取当前列中第一个元素的绿色度（也可以使用红色度且两者效果相同，但本人后续代码均使用的是绿色度作为判断）即最后一个元素的绿色度，方便后续判断该列是否有蓝色像素且如果有蓝色像素的情况下其是从上到下延伸还是从下向上延伸；再创建三个 int 类型变量 n（当前位置）、k（上一个上限位置或最右侧）、l（上一个下限位置或最左侧），其初始值分别为 picture.getHeight（）/2、picture.getHeight（）、0，用于后续二分查找过程中的逐步逼近；

（4）For 循环内，调用 while 循环，while 循环内使用 if 语句判断该列第一个像素的绿色度是否为 0（为 0 即代表该像素呈现蓝色），

若该列第一个像素的绿色度为 0，使用 if 语句判断下一个像素的绿色度是否与该像素的绿色度相等，若不相等则找到临界的像素，其位置即为当前行数的值，将其与 for 循环外创建的 max 的值进行比较，若比 max 大则使 max 的值等于该行行数，并使用 break 终止该次循环；

若下一个像素的绿色度与本像素相等，则开始二分查找，判断 n（数组最中间）的像素是否为蓝色，若为蓝色则向右二分查找，记录下限 l=n，当前位置 n=（n+k）/2，

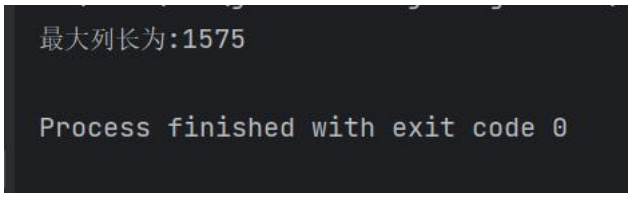
若为白色则向左二分查找，记录上限 k=n，当前位置 n=（n+l）/2；重复上述过程直至找出临界的像素才能终止对该列像素的二分查找；

若该列的最后一个像素的绿色度为 0，使用 if 语句判断上一个像素的绿色度是否与该像素的绿色度相等，若不相等则找到临界的像素，其位置即为当前行数的值，将其与 for 循环外创建的 max 的值进行比较，若比 max 大则使 max 的值等于该行行数，并使用 break 终止该次循环；若上一个像素的绿色度与本像素相等，则开始二分查找，判断 n（数组最中间）的像素是否为蓝色，若为蓝色则向左二分查找，记录上限 k=n，当前位置 n=（n+l）/2，若为白色则向右二分查找，记录上限 l=n，当前位置 n=（n+k）/2；重复上述过程直至找出临界的像素才能终止对该列像素的二分查找；

若第一个像素和最后一个像素的绿色度均不为 0，即第一个像素和最后一个像素都是白色，那么该列并没有蓝色像素，直接使用 break 语句终止该次循环；

（5）在 For 循环外输出 max 的值。

运行结果展示:



总结:

完成该任务需熟练掌握对已知图片的查找调用以及二分查找的原理，前者的学习与运用已在面向对象程序设计实验中进行，故本次作业的该环节非常轻松，对于后者，因为二分查找每一次都能排除一半的数据，故其单次循环时间复杂度应为 O（logn），总的时间复杂度为 O（nlogn），比一般的顺序查找（时间复杂度为 O（n^2））要快，所以选择使用二分查找。

任务 3 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：

$$\begin{cases} h_1 = 1 \\ h_i = h_{i-1} * 3 \end{cases}$$

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。

设计：

1.编写 Selection 类，用于完成对数组的选择排序：

(1).注意 Selection 类应继承 SortAlgorithm 类，方便调用其中的 less（）方法和 exchange（）方法；

(2).编写 sort（）方法：

（a）.调用 for 循环，i 从 0 开始，次数为待排序数组的长度-1 次，创建 int 类型变量 minIndex，令其等于当前循环的次数-1，目的为记录最小元素所在的索引处；

（b）.在上述 for 循环内部再调用一次 for 循环，j 初始值为 i+1，且 j 应小于数组的长度，使用 if 语句判断 objs[j]和 objs[minIndex]的大小，若数组在位置 j 的值比在 minIndex 小，则使用 exchange 方法交换两者的值；

2.编写 Shell 类，用于完成对数组的希尔排序：

(1).注意 Shell 类应继承 SortAlgorithm 类，方便调用其中的 less（）方法和 exchange（）方法；

(2).重写 sort（）函数：

(a).创建整数类型变量 h，其初始值为 1，使用 while 循环，条件为 h<a.length/2，在循环内使 h=3*h，当 h>a.length/2 时跳出循环，此时的 h 即为增长量；

(b).调用 while 循环，循环条件为 h>=1，使用 for 循环，i 初始值为 h，i<a.length，i++，在此 for 循环内部再调用 for 循环，j 初始值为 i，j>=h，j-=h，此时待插入元素是 a[j]，使用 if 语句和 less 方法比较 a[j]和 a[j-h]的大小，若前者小则交换两者位置，目的是使相隔 h 个位置的元素组成的新数组能从左至右以从小到大的规律排序，若 a[j-h]的值比 a[j]的值要小，符合上述所说的条件，则使用 break 语句结束此次循环；

(c).在 for 循环外，while 循环内，令 h/=2，目的是 while 循环的每一次循环时对数组排序的增长量的值是原来的 0.5 倍，用于完成更精细的排序，直至最后一次 h=1 时类似于冒泡排序的第一次排序；

3.编写 Quick 类，用于完成对数组的快速排序：

(1).注意 Quick 类应继承 SortAlgorithm 类，方便调用其中的 less（）方法和 exchange（）方法；

(2).重写 sort 方法：

(a).定义两个整数类型变量 left 和 right，其初始值分别为 0 和 objs.length-1；

(b).使用栈模拟递归调用：创建 Stack<Interger>类型变量 stack，分别对 right 和 left 使用 stack.push 方法；

(c).调用 while 循环，条件是！stack.isEmpty（），令 left=stack.pop（）、right=stack.pop。在此 while 循环内再次调用 while 循环，循环条件为 right>left，定义整数类型变量 partition=partition(objs，left，right)（partition（）方法在(d)中介绍），判断 partition-left 是否>right-partition，若是则对 partition-1 和 left 使用 stack.push 方法，并让 left 的值等于 partition+1；若 partition-left>right-partition 不成立，则对 right 和 partition+1 使用 stack.push 方法，并让 right 的值等于 partition-1；

(3).编写 partition 方法，参数为 Comparable[] objs，int leftobjs，int rightobjs：

(a).定义 Comparable 类型变量 key=objs[leftobjs], 定义两个整数类型变量 left=leftobjs、right=rightobjs+1,作为两个指针，分别指向待切分元素的最小索引处和最大索引处的下一位置；

(b).调用 while 循环, 循环内再前后调用两次 while 循环（相互独立），第一次 while 循环条件为 less(key,objs[--right]),

即判断数组从最右侧开始依次向左的元素是否大于分界值 key，若大于则继续指向其左侧的元素，若 right==leftobjs（即指针指向了最左侧元素）或者分界值大于指针指向的元素，则跳出此 while 循环转而进入第二个 while 循环，其循环条件是 less（objs[++left],key），即从数组最左侧向右的元素小于分界值 key，再使用 if 语句，若 left==rightobjs 则使用 break 语句终止循环，否则使指针继续向右索引。结束第二个 while 循环后再调用 if 语句判断 left 是否小于 right，若 left>=right，则使用 break 语句跳出 while 循环（此时已经对数组在 leftobjs 和 rightobjs 间的元素完成排序），否则使用 exchange 方法交换左右两个指针索引的元素的值；

(c).while 循环外，再次用 exchange 方法交换分界值 key 处元素与两指针指向的元素（此时左右两指针已指向同一元素），后 return right 返回 right 的值。

4.编写 Mergesort 类，用于完成对数组的归并排序:

(1).注意 Quick 类应继承 SortAlgorithm 类，方便调用其中的 less（）方法和 exchange（）方法；

(2).创建 Comparable 类型数组 assist，作为归并时的辅助数组；

(3).重写 sort 方法：初始化数组 assist，令其长度为 objs.length，即待排序数组的长度。定义两个整数类型变量

lift 和 right，其值分别为 0 和 objs.length-1，用于记录数据中的最小和最大索引。

(4).重载 sort 方法，参数为 Comparable[] objs，int left，int right，其中 objs 即为待排序数组，left 和 right 是步骤(3)中所定义的两个整数类型常量。使用 if 语句判断 right 是否小于 left，若是则 return 结束循环，若不是则继续下面的操作。定义整数类型变量 mid=(right-left)/2,用于后续对待排序数组进行分组排序。调用两次重载后的 sort 方法，第一次的参数为 objs、left、mid，第二次为 objs、mid+1、right。调用 merge 方法（具体方法实现在下一步骤中），参数为 objs、left、mid、right，用于归并上述两个 sort 方法排序后的数据;

(5).编写 merge 方法，用于对两数组进行归并，参数有 Comparable[] objs,int left,int mid,int right:

(a).定义三个指针 p、p1、p2，其值分别等于 left、left、mid+1，p 用于对辅助数组 assist 中第 p+1 个元素进行处理，p1 和 p2 用于指向(4)中经过 sort 方法排序后的两个数组并对其进行比较。

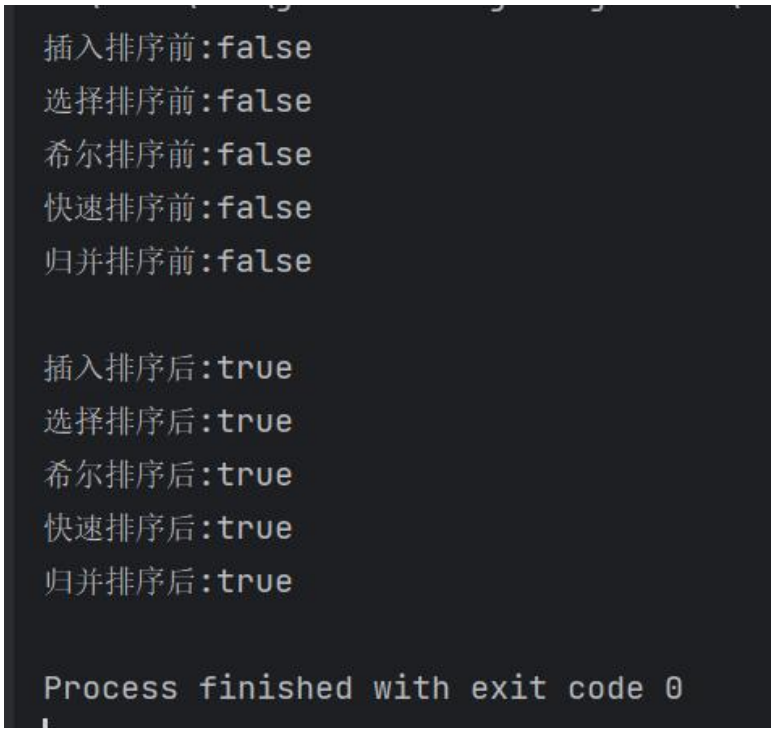
(b).使用 while 循环，循环条件为 p1<=mid&&p2<=right，再使用 if 语句 less 方法判断待排序数组在 p1 位置处和 p2 位置处元素值的大小，若 p1 处值小则辅助数组 assist 在 p 处位置的元素等于待排序数组 objs 在 p1 处元素，否则等于在 objs 在 p2 处元素。当 p1 或 p2 中的一个已经遍历完其中的一个

数组时，使用 while 循环直接将 assist 在 p 处的元素等于 objs 未遍历结束的指针所指位置处的元素，并用 p++、p1++、p2++使遍历依次往后进行；

(c).调用 for 循环，定义整数类型变量 index=left，index<=right，index++，让待排序数组 objs 在 index 位置处的元素的值等于辅助数组 assist 在 index 位置处的元素的值；

运行结果展示：

输出对各种排序算法排序前数组是否是正序分布，再对各数组使用不同的排序方法进行排序，输出对已完成排序的数组是否是正序分别，若是则输出 true，否则输出 false：



总结与收获:

1.在对本任务要求的代码实现中，进一步加深了对插入排序、选择排序、希尔排序、快速排序、归并排序的原理的印象及巩固完善掌握了对上述排序算法的代码书写的相关知识体系。

2.再次复习巩固了继承、多态相关的知识，并将之运用到对不同排序算法的代码实现中。

任务 4 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：2^8、2^9、2^{10}、……、2^{16} 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。

设计：（本任务使用了三个 java.class）

1.SortTest 类（老师给定的代码，用于对排序算法进行测试）

2.LineXYDemo 类（也是给定的代码，但做了一些改动，下述仅为改动部分的设计）：

(1) .添加有参构造器 LineXYDemo（String title，int i），其中 i 影响了对测试使用的数组的元素分布，i=1、2、3 时测试使用的数组分别为随机数组、正序数组、倒序数组。

(2) .改写 createDataset（）方法，用于测量比较各排序算法对随机序列排序所消耗的时间：

(a).定义整数类型数组 dataLength，用于给定生成数组的长度。由于任务要求为对规模为 2 的八次方到十六次方的数组测试，故

dataLength={256,512,1024,2048,4096,8192,16384,32768,65536}

(b).定义 double 类型数组 elapsedTime，此数组长度为 dataLength 数组的长度；

(c).创建 Insertion 对象 alg，调用 for 循环，循环次数等于 dataLength 的长度，令 elapsedTime[i]=test(alg,GenerateData.getRandomData(dataLength[i]),5),用于记录 dataLength 中的每一个数组规模所需消耗的时间，再在循环外定义 double 类型数组 Y1，其长度等于 dataLength 数组的长度，调用 for 循环将上述步骤记录的 elapsedTime 取对数后的值依次拷贝到 Y1 数组中；

(d).分别创建 Selection 对象 alg2、Shell 对象 alg3、Quick 对象 alg4、Mergesort 对象 alg5，将其排序消耗的时间取对数后依次存入 Y2、Y3、Y4、Y5 数组中，其余步骤和(c)中相同；

(e).设 X、Y 轴，X 为 double 类型数组，按照任务的要求将其长度取为 9，分别为{8,9,10,11,12,13,14,15,16}，每个数字代表的是 2 的几次方，Y 为 double[ ][ ]数组，有{Y1,Y2,Y3,Y4,Y5}。再定义 XYSeries 类型数组对象 series，以此为每一种排序算法的测试结果取名标注，便于区分。

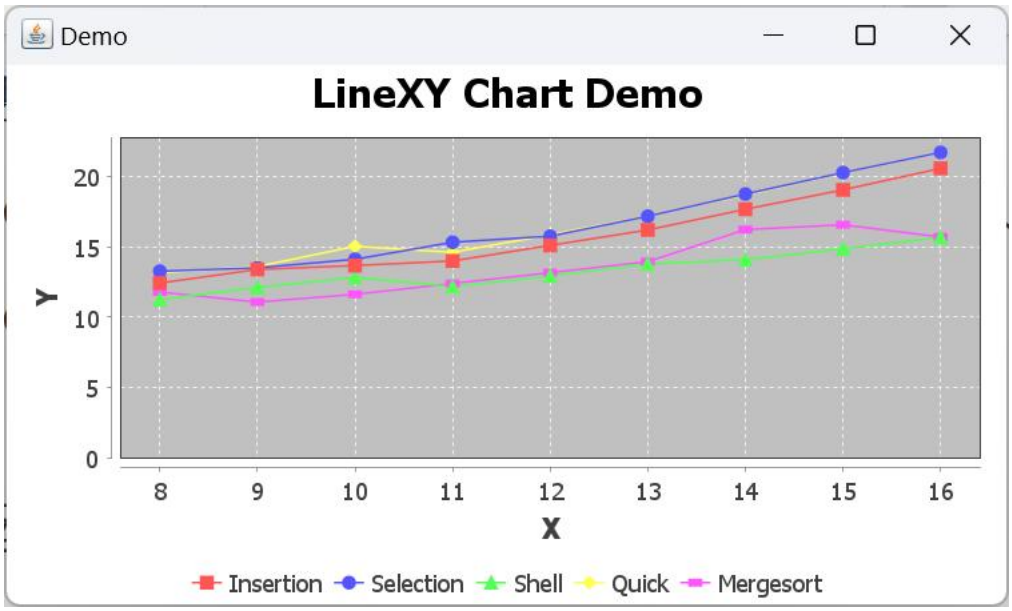
(f).编写 createDataset2（）方法和 createDataset3（）方法，其与 createDataset（）方法的唯一不同在于调用 test 方法时分别将 GenerateData.getRandomData（dataLength[i]）改为 GenerateData.getSortedData（dataLength[i]）、GenerateData.getInversedomData（dataLength[i]），用于测量不同算法对正序序列、逆序序列排序所需消耗的时间。

3.编写 Test 类，用于对上述代码进行测试：

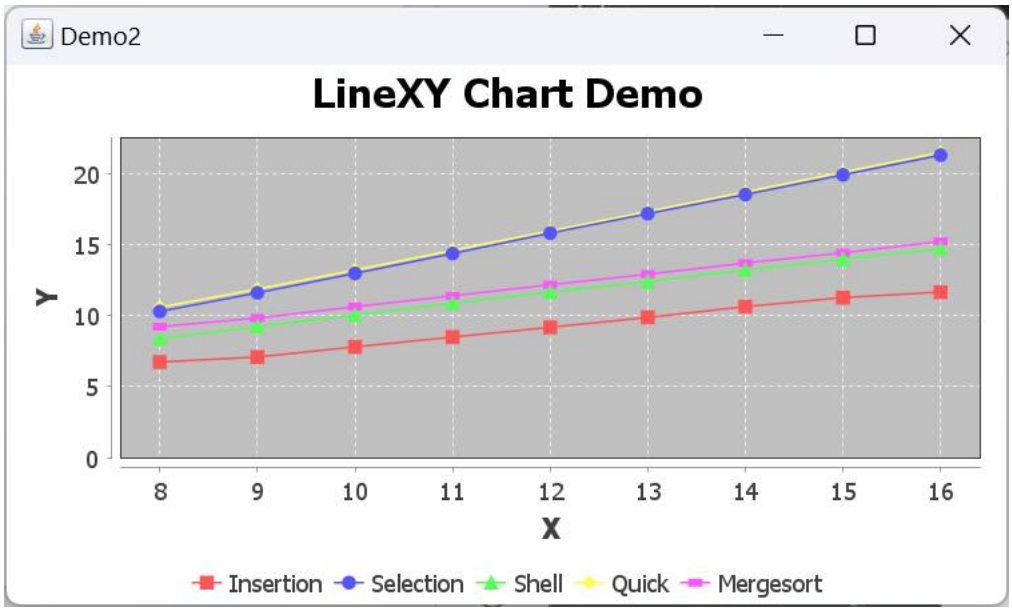
调用 LineXYDemo 的有参构造器创建三个 LineXYDemo 对象 demo、demo2、demo3，其 int 类型参数取值依次为 1、2、3，用于测试随机序列、正序序列、逆序序列下各排序算法在不同规模下的时间消耗量。调用 demo.pack()方法及 demo.setVisible(true)方法以得到图表，获取 demo2 和 demo3 图表的步骤和获取 demo 图表类似。

运行结果展示：

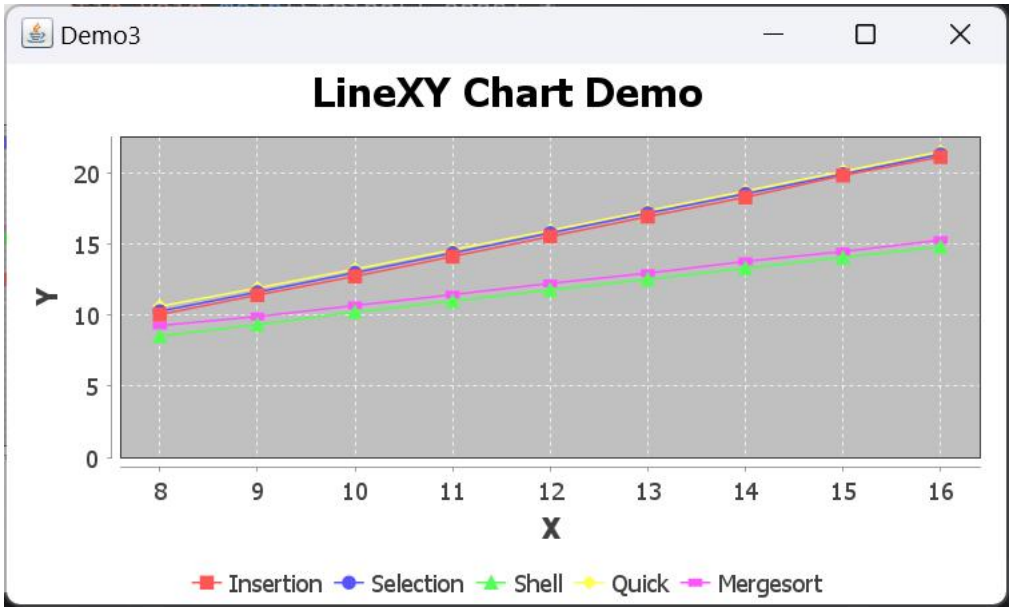
1.各排序算法对随机序列排序在不同规模下的时间消耗量（X 轴的值代表 2 的多少次方，Y 轴的值代表以纳秒为单位并取对数后的消耗时间值）：



2.各排序算法对正序序列排序在不同规模下的时间消耗量（X 轴的值代表 2 的多少次方，Y 轴的值代表以纳秒为单位并取对数后的消耗时间值）：



3.各排序算法对逆序序列排序在不同规模下的时间消耗量（X 轴的值代表 2 的多少次方，Y 轴的值代表以纳秒为单位并取对数后的消耗时间值）：



总结与收获：

1.算法的运行时间随序列规模变化规律：对所有算法而言，所排序的序列规模越大，消耗的时间越多。

2.同规模不同序列不同排序算法之间的差异：

(1).随机序列：规模大时，插入排序消耗时间较少，希尔排序和归并排序次之，选择排序和快速排序消耗时间最多；

(2).正序序列：无论规模希尔排序和归并排序消耗时间较少，插入排序次之，选择排序和快速排序消耗时间最多；

(3).逆序序列：无论规模希尔排序和归并排序消耗时间较少，插入排序选择排序和快速排序相当且消耗时间较多；

任务 5 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。）

设计：

1.给 GenerateData 类型添加新方法 getData（int N）：

(1) .创建 double 类型数组 numbers，其长度为 N;

(2) .调用 for 循环，从 i=0 开始循环 N 次，循环内使用 if 语句给 numbers[i]赋值，i<N/2 时其值为 0.0，i<N*3/4 时其值为 1.0, i<N*7/8 时其值为 2.0, 剩余情况则取值为 3.0;

(3) .使用 shuffle（）方法将上述正序序列 numbers 中元素位置打乱;

(4) .return 语句返回 numbers。

2.改写 LineXYDemo 类型：

(1) .改写 createDataset（）方法：对任务 5 中的 createDataset()方法的改写类似于任务 4 中 createDataset、createDataset2、createDataset3 的改写，只需将每一个 SortAlgorithm 对象所用到的 test 方法中的 GenerateData.getRandomData(dataLength[i]),5)改写为 GenerateData.getData(dataLength[i]),5)，其余改写均与任务 4 中三种方法的改写相同。

3.创建 Test 类型，用于测试上述代码是否能成功运行：

编写 main 函数，创建 LineXYDemo 对象 demo，标题为 Demo，调用 demo.pack()方法及 demo.setVisible(true)方法，后运行即可得到各种排序算法对给定的序列排序消耗的时间量的线性图表。

4.创建 CompareToWork4 包，目的是得到各种排序算法分别对任务 4 和任务 5 所涉及的四种序列排序所需消耗的时间量的线性图表，该包内涉及改写的 LineXYDemo 类型及 Test 测试类型。

(1) .改写 LineXYDemo 类型：

(a).编写有参构造 LineXYDemo(String title, int i)，其中 i 的取值影响了对序列（随机、正序、逆序、特殊）的选取；

(b).调用 if-else if 语句判断 i 的值，i==1 时调用 createDataset（）方法；i==2 时调用 createDataset2（）方法；i==3 时调用 createDataset3 方法；i==4 时调用 createDataset4（）方法；i==5 时调用 createDataset5（）方法；

(c).编写 createDataset（）方法：

代码大体与前几个任务中的 createDataset（）方法的改写相同，但方法从比较不同排序算法对同一序列排序消耗的时间变为比较同一排序算法对不同序列排序消耗的时间，即创建的对象均为 Insertion（）类型对象，因需比较的序列分为 4 种，故创建 4 个对象 alg、alg2、alg3、alg4，分别用数组 Y1、Y2、Y3、Y4 储存对同一序列不同规模所消耗的时间量；创建 XYSeries 数组类型时给不同序列排序消耗时间量的线性图线取名，便于区分；

(d).按照(c)中步骤依次编写 createDataset2 方法、createDataset3 方法、createDataset4 方法、createDataset5 方法，仅将创建的对象分别改为 Selection（）类型对象、Shell（）类型对象、Quick（）类型对象、Mergesort（）类型对象。

(2) .编写 Test 类类型，用于测试上述代码是否实现对同一排序算法在对不同序列不同规模进行排序所消耗时间量的对比：

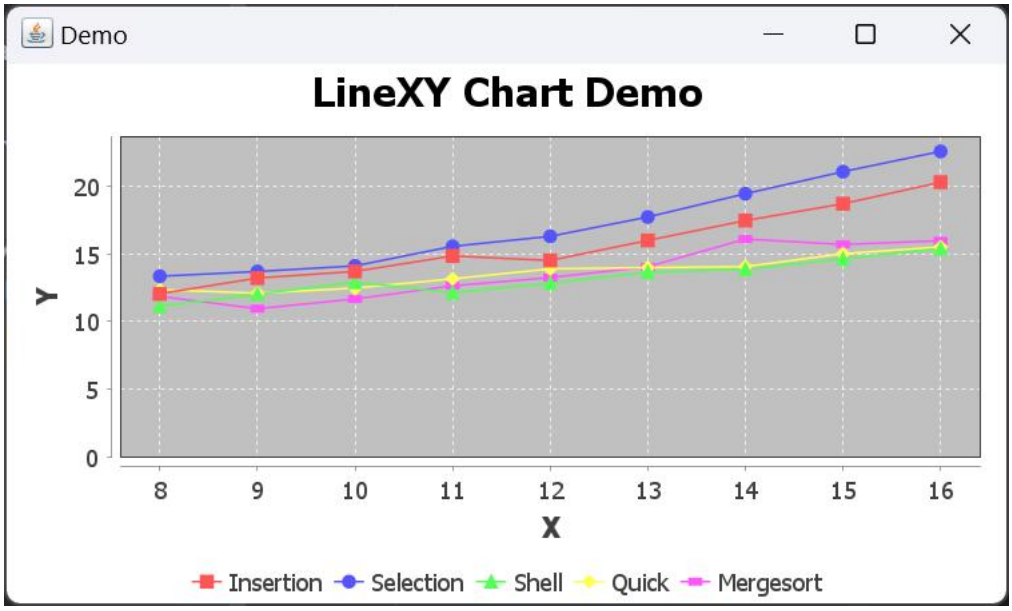
(a).先编写完成插入排序的相关内容比较：

调用 LineXYDemo 类有参构造器创建对象 demo，其标题为 Demo，i 取值为 1，调用 demo.pack（）方法、demo.setVisible(true)方法。

(b).编写选择排序、希尔排序、快速排序、归并排序的相关内容比较，步骤类似上述(a)中编写插入排序的内容，仅将 i 依次取值为 2、3、4、5 即可。

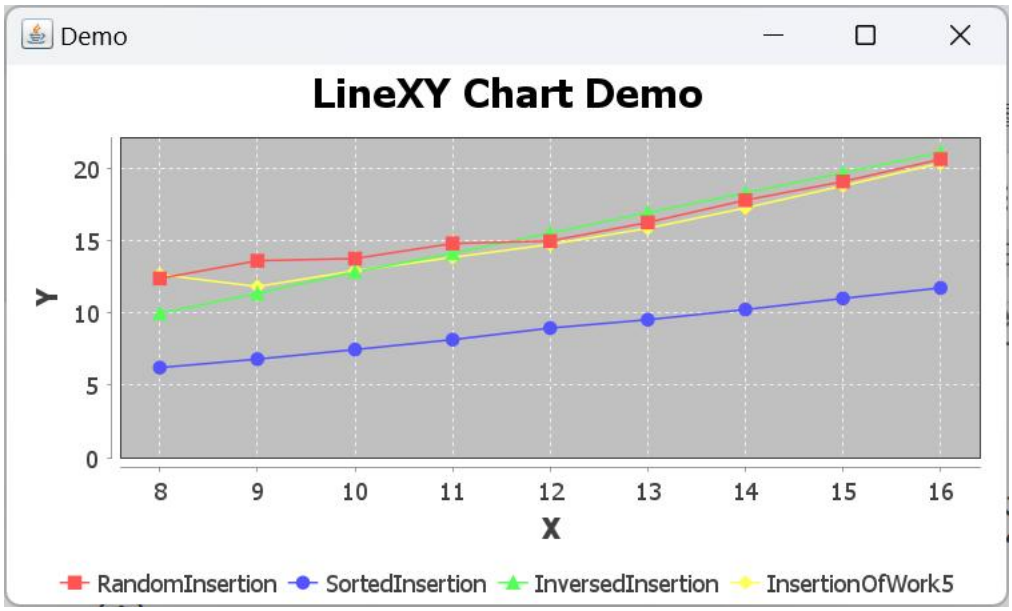
运行结果展示:

1.不同排序算法对任务 5 要求的序列进行排序所消耗的时间的比较（X 轴的值代表 2 的多少次方，Y 轴的值代表以纳秒为单位并取对数后的消耗时间值）:

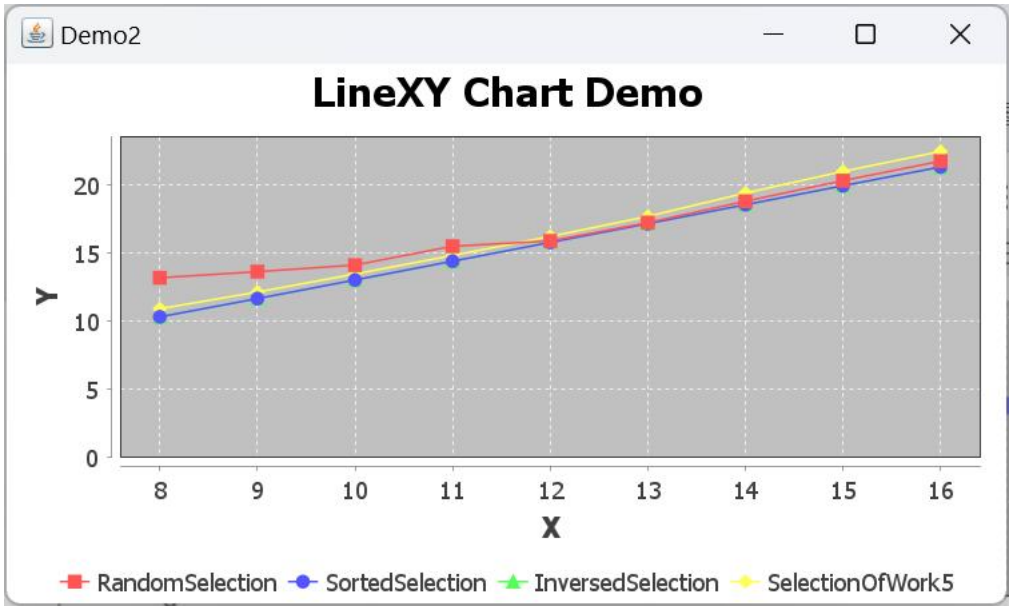


2. 同一排序算法对不同序列进行排序所消耗的时间的比较:

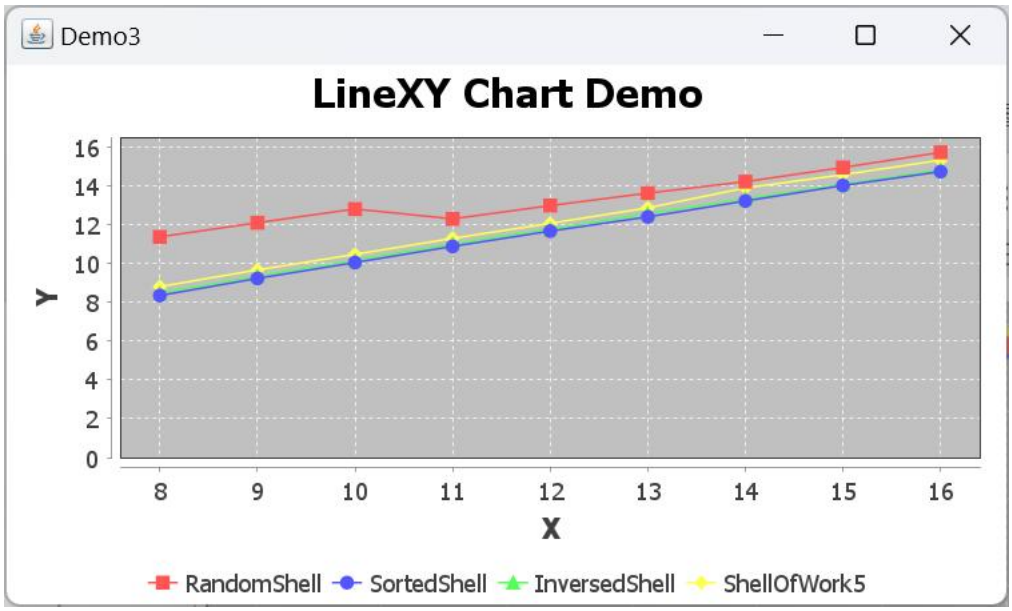
(1).插入排序


Demo
LineXY Chart Demo
Y
X
0
5
10
15
20
8
9
10
11
12
13
14
15
16
RandomInsertion
SortedInsertion
InversedInsertion
InsertionOfWork5

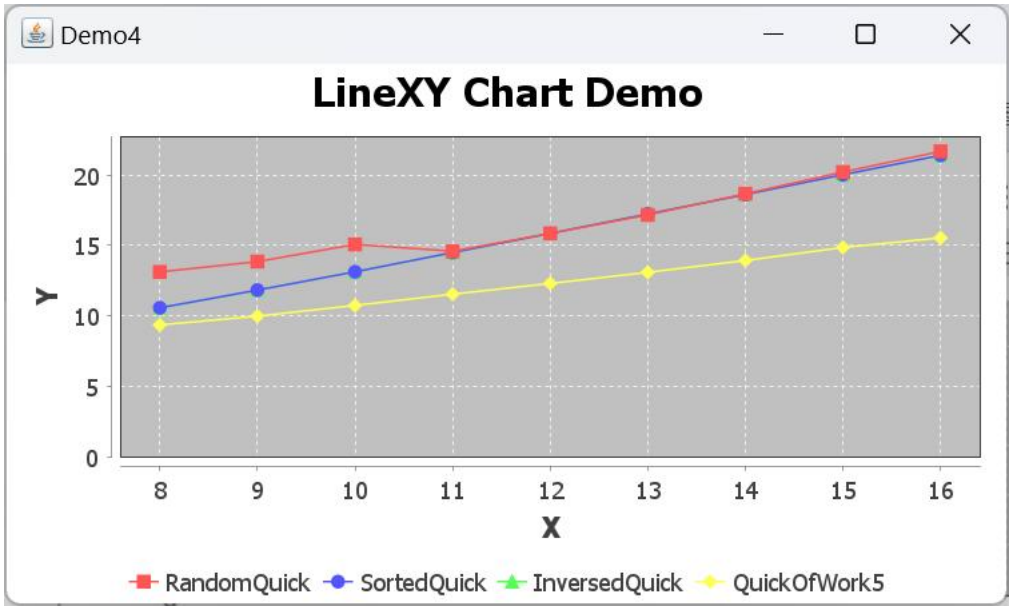
(2).选择排序


Demo2
LineXY Chart Demo
Y
X
0
5
10
15
20
8
9
10
11
12
13
14
15
16
RandomSelection
SortedSelection
InversedSelection
SelectionOfWork5

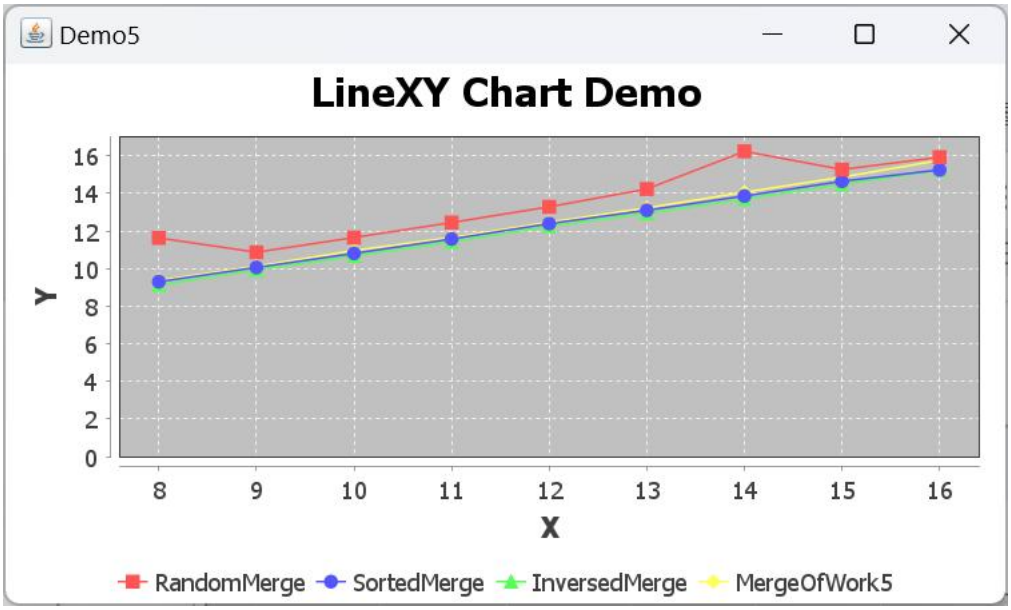
(3).希尔排序


Demo3
LineXY Chart Demo
16
14
12
10
8
6
4
2
0
Y
8
9
10
11
12
13
14
15
16
X
RandomShell
SortedShell
InversedShell
ShellOfWork5

(4).快速排序


Demo4
LineXY Chart Demo
20
15
10
5
0
Y
8
9
10
11
12
13
14
15
16
X
RandomQuick
SortedQuick
InversedQuick
QuickOfWork5

(5).归并排序



总结与收获:

1.对重复率高的序列排序时，序列规模很大时使用选择排序所需消耗的时间高于插入排序所需消耗的时间，而插入排序所需消耗的时间又明显高于使用希尔排序、快速排序、归并排序所需消耗的时间;

2.同一排序算法对不同序列进行排序时所需消耗的时间并不一定相同（这里的相同指消耗时间的指数级大小）：

(1).插入排序对正序序列排序的时间明显小于对逆序、随机、重复率高的序列排序的时间。

(2).选择排序在规模小时对随机序列排序消耗时间最多，规模大时各序列消耗时间近乎相等。

(3).希尔排序在规模小时对随机序列排序消耗时间最多，规模大时各序列消耗时间近乎相等。

(4).快速排序在规模小时随机序列排序消耗时间最长，规模大时对随机序列、正序序列、逆序序列排序消耗时间近乎相等；无论规模大小，重复率高的序列排序消耗时间总是最少。

(5).归并排序在规模小时对随机序列排序消耗时间最多，规模大时各序列消耗时间近乎相等。

任务 6 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：

① 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 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 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.Mcllroy 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 在 N 个数据中找第 k 小元素($1 \leqslant k \leqslant N$)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。

设计：

一．优化快速排序

1.编写 ImprovedQuick 类型

(1).拷贝 Quick 类型的代码，ImprovedQuick 类型仅在 Quick 类型的基础上对 sort（）进行修改并编写 insertionSort（）方法，下述将分别对两个改动进行阐述。

(2).编写 insertionSort(Comparable[] objs, int left, int right) 方法：调用 for 循环，设整数类型变量 i=left+1，i<right+1，i++，在此循环内部再调用 for 循环，设整数类型变量 j=i，j>left&&less(objs[j],objs[j-1]),j--，循环内部调用 exchange（）方法交换 objs 在 j 处元素的值和在 j-1 处元素的值；

(3).改写 sort（）方法：在原 Quick 类型 sort（）方法的第一个内部 while 循环内进行改写。设一个 整数类型变量，

其值由编写者自己决定，使用 if 语句判定当 right-left+1 的值<=设定的整数类型变量时，调用 insertionSort（objs，left，right）方法使用直接插入排序对 objs 在 left 至 right 处元素进行排序，后编写 break 语句终止循环；剩余步骤与原 Quick 类型中 sort 方法相同。

2.改写 LineXYDemo 类型：

(1).编写有参构造器 LineXYDemo(String title,int i), 其中 title 是输出显示的图表的标题，i 将影响对排序的序列的选取。使用 if 语句，判定 i==1 时调用 createDataset（）方法，i==2 时调用 createDataset2（）方法，i==3 时调用 createDataset3（）方法，i==4 时调用 createDataset4（）方法；

(2).改写 createDataset（）方法：创建两个 SortAlgorithm 类型对象，前一个使用改良前的快速排序对随机序列排序，后一个使用改良后的快速排序对随机序列排序，其余代码均与原方法相同。

(3).编写 createDataset2（）方法、createDataset3（）方法、createDataset4（）方法。其步骤均与(2)中改写 createDataset（）方法相同，仅需将随机序列依次更改为正序序列、逆序序列、重复率高的序列。

2.编写 Test 类型：

使用 LineXYDemo 的有参构造器创建四个 LineXYDemo 对象，分别输入想要的标题，i 的值依次取 1、2、3、4,分别对四个对象调用 pack（）方法和 setVisible（true）方法以获取其图表。

二．对重复率高的序列的快速排序

1.给 GenerateData 类型添加 4 个新方法：

(1).编写 getData2（int N）方法，用于获取重复率为 50%的序列。先定义 Double 类型数组 numbers，长度为 N，使用 for 循环让 i 从 0 开始循环 N 次，使用 if 语句判断 i<N/2 时给 numbers[i]赋值为 0.0，否则给其赋值为（double）i-N/2+1。For 循环外使用 shuffle 方法将 numbers 的元素顺序打乱，最后使用 return 返回 numbers。

(2).编写 getData3（int N）方法、getData4（int N）方法，分别用于获取重复率为 60%、80%的序列。代码大体与 getData2（int N）相同，仅需修改 if 语句的判定条件及 else 语句对 numbers[i]的赋值：if 判定条件分别改为 i<N*6/10、i<N*8/10,对 numbers[i]的赋值分别为（double）i-N*6/10+1、（double）i-N*8/10+1；

(3).编写 getData4（int N）方法，用于获取重复率为 100%的序列：先定义 Double 类型数组 numbers，长度为 N，

使用 for 循环让 i 从 0 开始循环 N 次，给 numbers[i]赋值为 0.0，最后使用 return 返回 numbers。

2.改写 LineXYDemo（）类型（仅对 createDataset（）方法进行修改）：

(1).创建四个 Quick（）对象，依次使其对 getData2（）、getData3（）、getData4（）、getData5（）的序列排序并记录消耗时间，定义 double 类型数组 Y1 记录四个时间；

(2).对 ImprovedQuick（）的消耗时间记录类似（1），使用

3.编写 test 类型：编写 main 方法，创建 LineXYDemo 对象 demo，对其调用 demo.pack()方法和 demo.setVisible(true)方法即可生成对应图表。

三．编写三路划分的快速排序

1.编写 ThreeWayQuick 类型：

(1).注意其应继承 SortAlgorithm 类型；

(2).拷贝原有的 Quick 类类型的代码，仅需对部分代码进行修改即可：在 sort（Comparable[] objs）方法中使用 if 语句判断 left<right 后进行的操作从原来的二路划分改写为三路划分，即分别将小于、等于、大于轴值的部分入栈。

2.改写 LineXYDemo 类型：直接拷贝（二）中的 LineXYDemo 类型，仅将每次创建的 ImprovedQuick 对象改为 ThreeWayQuick 对象即可，再在创建 XYSeries 类型数组时将第二个对象名 ImprovedQuick 改为 ThreeWayQuick，使图表注释得以修改。

四．在 N 个数据中找第 k 小元素(1≤k ≤N)

本人提供了两种方法：

1.选择排序法

(1).sort 方法的构造大体和任务 3 中的 Selection 类型的 sort 方法相同，仅需多定义一个整数类型变量 k，完成 k 次循环排序而非 n 次；

(2).编写 testSelection 方法：定义 long 类型变量 start，使用 System.currentTimeMillis () 方法令 start 记录当前时间，再创建 Selection 类对象 alg，使用 sort 方法对序列排序，再定义 long 类型变量 end，使用 System.currentTimeMillis （）方法令 start 记录当前时间，最后输出 end-start 即可得到排序消耗的时间。

(3).编写 main 函数进行测试，定义整数类型变量 N，其值大小决定了生成序列的规模，对其使用 testSelection 方法以测试消耗的时间。

2.快速排序法

(1).原理类似任务 3 中的 Quick 类型，仅做一些小改动：定义一个整数类型变量 k，其值为查找的第 k 小的元素。设定轴值，将比轴值小的元素放在轴值左边，比轴值大的元素放在右边，通过比较轴值的位置和 k 的大小来判断继续向左递归查找还是向右递归查找，通过此方法找到第 k 小的元素。

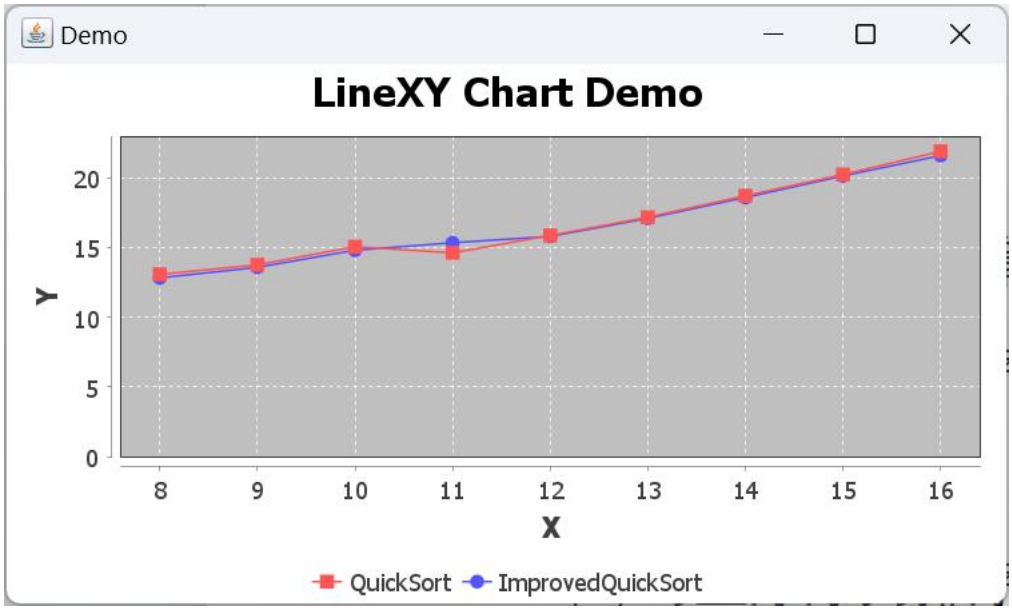
(2).编写 testQuick 方法：定义 long 类型变量 start，使用 System.currentTimeMillis（）方法令 start 记录当前时间，再创建 Quick 类对象 alg，使用 sort 方法对序列排序，再定义 long 类型变量 end，使用 System.currentTimeMillis（）方法令 start 记录当前时间，最后输出 end-start 即可得到排序消耗的时间。

(3).编写 main 函数进行测试，定义整数类型变量 N，其值大小决定了生成序列的规模，对其使用 testQuick 方法以测试消耗的时间。

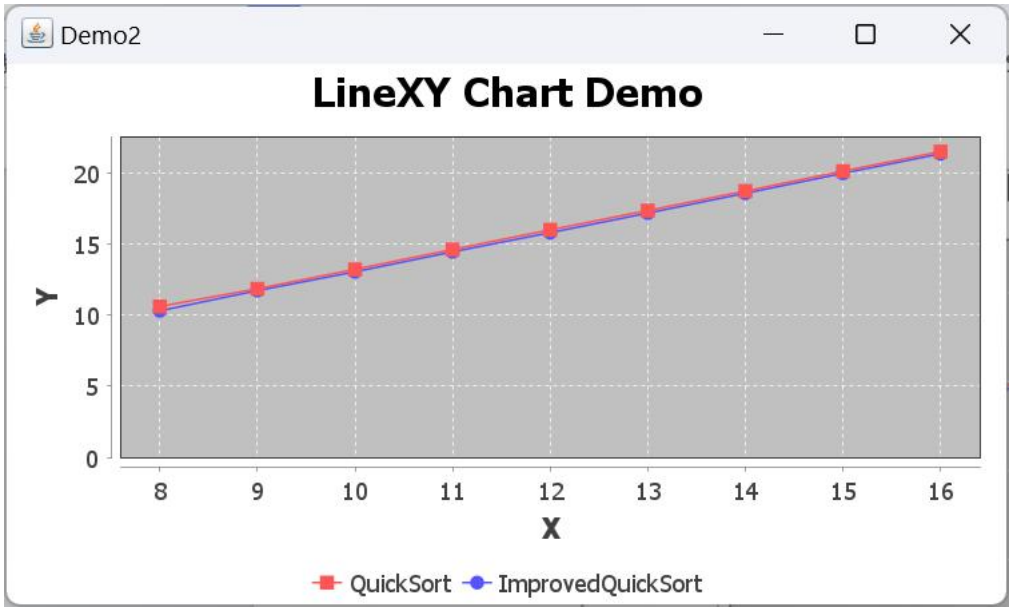
运行结果展示:

一．优化快速排序算法并与之前的进行对比:

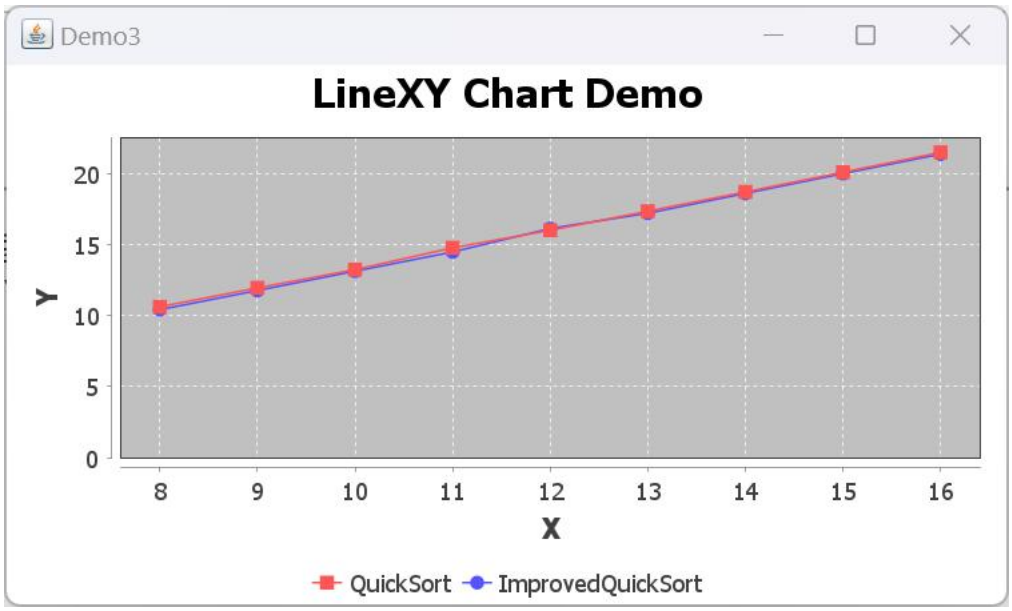
(1).对随机序列排序:



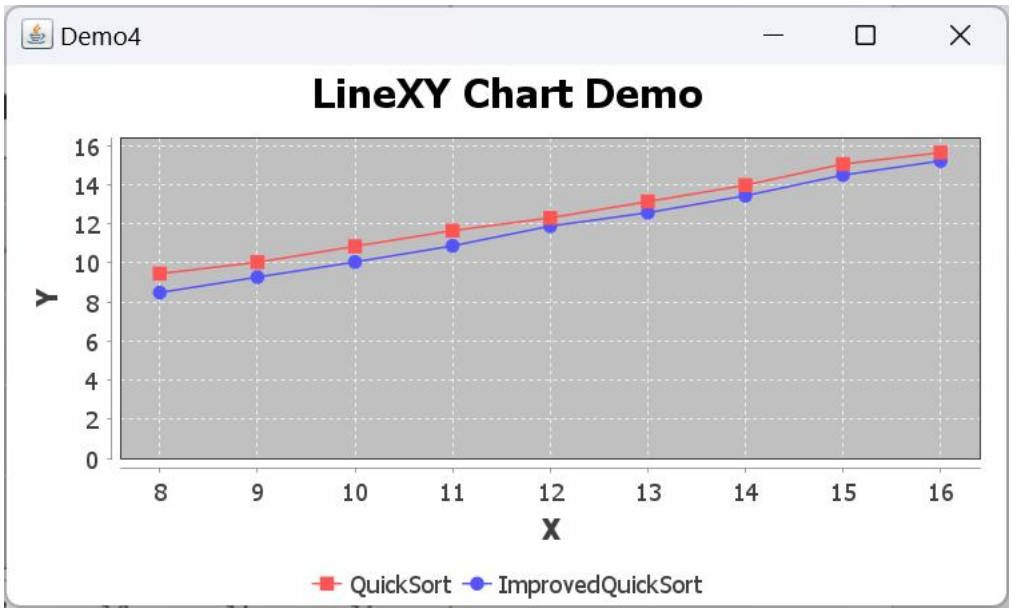
(2).对正序序列排序:



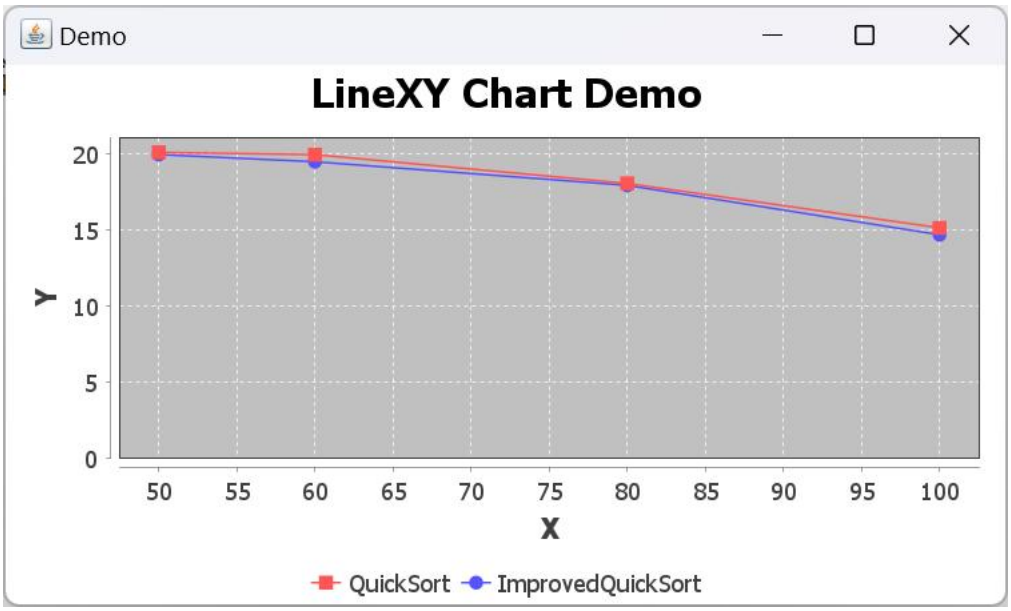
(3)对逆序序列排序:



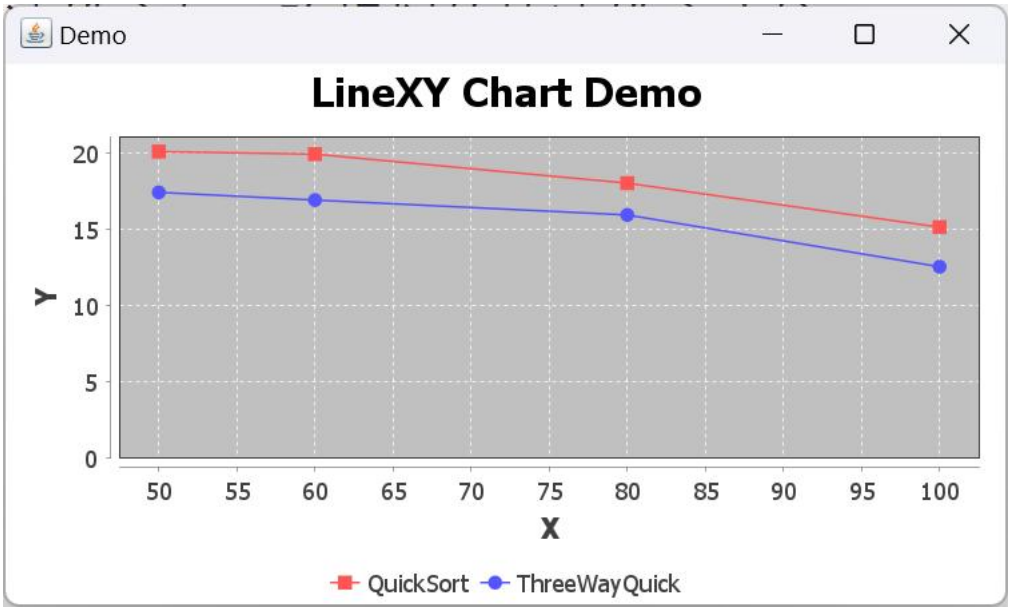
(4).对重复率高的序列排序:



二．在给定规模下队不同重复率的序列排序对比:



三．三路规划的快速排序与二路规划的快速排序比较:



四．不同方法查找第 k 小个元素

(1).选择排序（规模为 65536 即 2 的 16 次方）

（a）.k=10

```
该序列规模为65536的序列
需要找的是第10小的元素,所消耗的时间为（毫秒为单位）：
16

Process finished with exit code 0
```

（b）.k=100

```
该序列规模为65536的序列
需要找的是第100小的元素,所消耗的时间为（毫秒为单位）：
32
```

（c）.k=1000

```
该序列规模为65536的序列
需要找的是第1000小的元素,所消耗的时间为（毫秒为单位）：
143

Process finished with exit code 0
```

（d）.k=10000

```
该序列规模为65536的序列
需要找的是第10000小的元素,所消耗的时间为（毫秒为单位）：
1245
```

(2).快速排序（规模为 2 的 20 次方）

（a）.k=10

```
该序列规模为1048576的序列
需要找的是第10小的元素,所消耗的时间为（毫秒为单位）：
103

Process finished with exit code 0
```

（b）.k=100

```
该序列规模为1048576的序列
需要找的是第100小的元素,所消耗的时间为（毫秒为单位）：
111

Process finished with exit code 0
```

（c）.k=1000

```
该序列规模为1048576的序列
需要找的是第1000小的元素,所消耗的时间为（毫秒为单位）：
94

Process finished with exit code 0
```

（d）.k=10000

```
该序列规模为1048576的序列
需要找的是第10000小的元素,所消耗的时间为（毫秒为单位）：
64

Process finished with exit code 0
```

（e）.k=100000

```
该序列规模为1048576的序列
需要找的是第100000小的元素,所消耗的时间为（毫秒为单位）：
52

Process finished with exit code 0
```

（f）.k=1000000

```
该序列规模为1048576的序列
需要找的是第1000000小的元素,所消耗的时间为（毫秒为单位）：
62

Process finished with exit code 0
```

总结:

1.将快速排序优化后，设立了一个阈值，当划分出的数组的长度小于阈值时将直接对此数组进行直接插入排序。当阈值取 100 时，得出上述代码展示的图表，可看出经改良的快速排序算法对重复率高的序列进行排序相较于原本的快速排序算法的效率要高，其余序列差别不大（可能是序列规模太小或者阈值取值问题？）；

2.用上述未改良的快速排序算法和改良后的快速排序算法对四种不同重复度的序列（重复率分别为 50%、60%、80%、100%）排序并绘制出图表，可看出重复率越高，改良后的快速排序相较于未改良的快速排序效率增高越显著，且重复率越高，消耗的时间越少;

3.使用三路划分的思想再次改进快速排序算法，再用此算法与未改良的快速排序算法进行测试，测试方式如（2），通过绘制的图表可看出三路划分的快速排序算法在对高重复率的序列排序的效率明显高于二路划分的快速排序算法;

4.在 N 个数据中找到第 k 小元素:

(1).使用选择排序

对同一规模的序列取不同 k 值进行测试，发现随着 k 的增大，所消耗的时间也增大，两者呈正相关关系。通过计算得出其时间复杂度为 O（kn），理由是为找到第 k 小元素仅需循环 k 次，而当 k=n 时时间复杂度为 O（n^2）。

(2).使用快速排序

对同一规模的序列取不同 k 值进行测试，发现随着 k 的增大，所消耗的时间先减小后增大，而 k 越接近数组长度的一半，所消耗的时间越少。通过计算得其平均时间复杂度为 O（n），最差情况为 O（n^2）。