Source Code

by Jason Ivey & Terry Speicher using: LaTeX & Dr Java.

October 2016

1 Goal

To read the 100kpoint.txt file and sort the data within it providing reliable data on the run-time of each algorithm.

```
1   class Point{
2     private double x;
3     private double y;
4     public Point(double x, double y){
5       this.x = x;
6       this.y = y;
7     }
8     public double getX(){
9       return this.x;
10    }
11    public double getY(){
12      return this.y;
13    }
14  }
```

```
1  //Terry Speicher and Jason Ivey
2
3  /**
4   * Collection of sorting algorithms to be compared, along with a few utilitarian methods
5   * @author Terry Speicher
6   * @author Jason Ivey
7   */
8  public class SortAlgorithms{
9
10   /**
11    * Constructor doesn't have to do anything
12    */
13   public SortAlgorithms(){
14
15   }
16
17   /**
18    * Not a complete comparison of all items to see if they are in order, but quickly checks
19    * consecutive items to make sure that the smaller item is first.
20    * @param data Point[] of some size to be processed
21    * @return boolean Return true if each consecutive item is less than the one after it.
22    */
23   public boolean isSorted(Point [] data){
24
25     boolean sortedYN = true;
26     int i = 0;
27     while (i < data.length - 2 && sortedYN == true) {
28       if (data[i].getX() > data[i+1].getX())
29         sortedYN = false;
30       i++;
31
32     }
33     return sortedYN;
34   }  //end isSortedYN
35
36
37   /**
38    * Compare the first 100 items in each given Point array to see if they are equal.  Used
39    * as a testing routine to work out some bugs when learning how to pass the data by
40    * reference.
41    * @param data Point[] of some size to be processed
42    * @param test Point[] of some size to be processed
```

```
43      * @return boolean Return true if the first 100 x coordinates are the same in the
44      * two arrays
45      */
46     public boolean looksEqual(Point [] data, Point [] test){
47
48       boolean lookSame = true;
49       for (int i = 0 ; i < 100 ; i++)
50         if (data[i].getX() != test[i].getX())
51         lookSame = false;
52       return lookSame;
53     } //end looksEqual
54
55   //  ***********  Begin BubbleSort *****************
56
57     /**
58      * BubbleSort was coded by the authors from scratch.  This is the worst case sort because
59      * it does not even adjust the inner loop to start at the first nonsorted element - it
60      * always starts at the beginning.
61      * @param d Point[] of some size to be processed
62      */
63     public void bubbleSort(Point [] d){
64       for(int x = 0; x < d.length; x++){
65         for(int y = 0; y < d.length-1; y++){
66           if ( d[y].getX() > d[y+1].getX() ) {
67             Point temp = d[y];
68             d[y] = d[y+1];
69             d[y+1] = temp;
70           }
71         }
72       }
73     }
74
75   //************* End Bubble Sort *******************
76
77   //  *************  Begin QuickSort Methods  ******************
78
79     /**
80      * Copied from the textbook.  Improved on by adding a check in the partitioning
81      * routine to check "if (endOfLeftList != scan)" before swapping, because we found
82      * that the swap routine was sometimes being called to swap an element with itself if
83      * the "endOfLeftList" equaled "scan".
84      * @param data Point[] of some size to be processed
85      */
86     public void quickSort(Point [] data){
87       doQuickSort(data,0,data.length-1);
88
89
90     }  // end quickSort
91
92     /**
93      * sub method to start recursion process.  Required to
94      * pass in the two indexes for processing the sub array.
95      * Initially, we pass in the absolute beginning and
96      * absolute ending elements.
97      * @param array Point[] to be sorted
98      * @param start int that shows where the start of the subarray to be sorted is
99      * @param end int that shows where the end of the bubarray to be sorted is
100     **/
101    private void doQuickSort(Point[] array, int start, int end){
102      int pivotPoint;
103      if (start < end){
104        pivotPoint = partition(array, start, end);
105        doQuickSort(array,start,pivotPoint-1);
106        doQuickSort(array,pivotPoint+1,end);
107      }
108    } // end doQuickSort
109
110    /**
111     * Partition the given array from array[start] to array[end] and put
112     * pivot element in middle.  Then move all elements with values less
113     * than the pivot point to the left of the pivot point and move
114     * all elements with values greater than the pivot point to the right
115     * of the pivot point.
116     * @param array Point[] to be sorted
117     * @param start int that shows where the start of the subarray to be partitioned is
118     * @param end int that shows where the end of the bubarray to be partitioned is
119     **/
120    private int partition(Point[] array, int start, int end){
121
122      double pivotValue;
123      int endOfLeftList;
```

```
124       int mid;
125
126       mid = (start + end) / 2;
127
128       //take the first element of the array and swap
129       //it with the middle element.  I don't know why,
130       //except to keep with the idea of using the middle
131       //element as the pivot
132       swap(array,start,mid);
133
134       pivotValue = array[start].getX();
135       endOfLeftList = start;
136       for (int scan = start + 1; scan <= end; scan++){
137         if (array[scan].getX() < pivotValue){
138           endOfLeftList++;
139           if (endOfLeftList != scan)
140             swap(array,endOfLeftList,scan);
141         }
142       }//end of for
143
144       swap(array,start,endOfLeftList);
145
146       return endOfLeftList;
147
148     }//end partition
149
150     /**
151      * Swap elements array[a] and array[b]
152      * @param array Point[] with elements to be swapped
153      * @param a index of an element to be swapped
154      * @param b index of other element to be swapped
155      **/
156     private void swap(Point[] array, int a, int b){
157
158       Point temp = array[a];
159       array[a] = array[b];
160       array[b] = temp;
161
162     }//end swap
163
164   // **************  End QuickSort Methods   *****************
165
166   //  *************  Begin MergeSort Method  ******************
167
168     /*****************************************************************************
169       *    With online help from:
170       *    Title: howtodoinjava.com
171       *    Author: Lokesh Gupta
172       *    Date: October 23, 2015
173       *    Availability: http://howtodoinjava.com/algorithm/merge-sort-java-example/
174       *
175       *****************************************************************************/
176     /**
177      * Merge Sort method to sort an array of data points
178      * @param data Point[] of some size to be processed
179      */
180     public void mergeSort(Point[] data){
181
182       if (data.length <= 1){
183         return;
184       }
185
186       Point[ ] half1 = new Point[ data.length / 2 ];
187       Point[ ] half2 = new Point[ data.length - half1.length ];
188
189       System.arraycopy(data, 0, half1, 0, half1.length);
190       System.arraycopy(data, half1.length, half2, 0, half2.length);
191
192       mergeSort(half1);
193       mergeSort(half2);
194
195       merge(half1, half2, data);
196       return;
197
198       /**
199        * Main recursive method to merge sorting.
200        * divides Points array into smaller pieces.
201        * Makes use of merge(Point[],Point[],Point[]) to combine and sort data
202        **/
203     }
204     /**
```

```
205      * Merge elements of array[a] and array[b] into result
206      * @param half1 Point[] half to be combined with brother in order
207      * @param half2 Point[] brother of half1
208      * @param result Point[] to be returned when brothers combined and sorted.
209      **/
210     private static void merge(Point[] half1, Point[] half2, Point[] result){
211       int x = 0;
212       int y = 0;
213       int merge = 0;
214
215       while(x < half1.length && y < half2.length ){
216         if(half1[x].getX() < half2[y].getX()){
217           result[merge] = half1[x];
218           x++;
219         }
220         else{
221           result[merge] = half2[y];
222           y++;
223         }
224         merge++;
225       }
226       System.arraycopy(half1, x, result, merge, half1.length - x);
227       System.arraycopy(half2, y, result, merge, half2.length - y);
228     }
229     //  *************  End mergeSort Methods  ******************
230
231     //  *************  Begin insertionSort Methods  ******************
232
233     /*****************************************************************************
234       *    With online help from:
235       *    Title: http://www.java2novice.com/
236       *    Author: N/A
237       *    Date: N/A
238       *    Availability: http://www.java2novice.com/java-interview-programs/insertion-sort/
239       *****************************************************************************/
240
241
242     /**
243      * Sort Point[] by inserting unsorted points into the correct positions in sorted Point[]
244      * @param array Point[] array to be sorted
245      **/
246
247     public static void insertionSort(Point array[]) {
248       int n = array.length; // Limiter initalized to length of Point[] array
249       for (int j = 1; j < n; j++) {
250         Point key = array[j]; // Key to comparison
251         int i = j-1; // Interloop counter, one less than 'j'
252         while ( (i > -1) && ( array[i].getX() > key.getX() ) ) {
253           array [i+1] = array [i];
254           i--;
255         }
256         array[i+1] = key; // Inserting key into sorted portion of array.
257       }
258     }
259
260     //  *************  End insertionSort Methods  ******************
261
262
263     //  *************  Begin selectionSort Methods  ******************
264
265     /*****************************************************************************
266       *    With online help from:
267       *    Title: Sorting.java
268       *    Author: Lewis/Loftus
269       *    Date: N/A
270       *    Availability: http://www.ics.uci.edu/~stasio/winter06/Lectures/Lec7code/ComparableExample
          /Sorting.java
271       *****************************************************************************/
272
273     /**
274      * Sorts the Point array of objects using the selection sort algorithm.
275      * @param input Point[] array to be sorted
276      **/
277
278     public static void selectionSort (Point[] input)
279     {
280       int min; // Min value
281       Point temp; // Temp storage of Points
282
283       for (int index = 0; index < input.length-1; index++)
284       {
```

```
285         min = index; // Initializing min
286         for (int scan = index+1; scan < input.length; scan++)
287           if (input[scan].getX() < input[min].getX())
288           min = scan; // update min
289 
290         // Swap the values
291         temp = input[min];
292         input[min] = input[index];
293         input[index] = temp;
294       }
295     }
296 
297 
298   //  **************   Cocktail Shaker Sort   ********************
299   //http://www.javacodex.com/Sorting/Cocktail-Sort
300 
301     /**
302      * Cocktail Shaker Sort
303      * @param array Point[] of some size to be processed
304      */
305     public static void cocktailShakerSort( Point[] array ){
306       boolean swapped;
307       do {
308         swapped = false;
309         for (int i =0; i<=  array.length  - 2;i++) {
310           if (array[ i ].getX() > array[ i + 1 ].getX()) {
311             //test whether the two elements are in the wrong order
312             Point temp = array[i];
313             array[i] = array[i+1];
314             array[i+1]=temp;
315             swapped = true;
316           }
317         }
318         if (!swapped) {
319           //we can exit the outer loop here if no swaps occurred.
320           break;
321         }
322         swapped = false;
323         for (int i= array.length - 2;i>=0;i--) {
324           if (array[ i ].getX() > array[ i + 1 ].getX()) {
325             Point temp = array[i];
326             array[i] = array[i+1];
327             array[i+1]=temp;
328             swapped = true;
329           }
330         }
331         //if no elements have been swapped, then the list is sorted
332       } while (swapped);
333     }
334 
335     /**
336      * Force Push Sort was coded by Terry Speicher to show how he *thought* the Cocktail
337      * Shaker was supposed to work.  It is kind of a "double ended" selection sort.
338      * @param p Point[] of some size to be processed
339      */
340     public void forcePush(Point [] p){
341 
342       Point temp = new Point(0.0,0.0);
343 
344       Point swapper = new Point(0.0,0.0);
345 
346       for (int start = 0, end = p.length -1 ; start < end  ; start++, end--){
347         if (start != end) { //only 1 element.  We are done.
348           if ( (end - start) == 1) {
349             //only two elements left
350             //compare and swap if necessary
351             if (p[start].getX() > p[end].getX()){
352               temp = p[start];
353               p[start] = p[end];
354               p[end] = temp;
355             }
356           } else {
357             //more than 2 elements - go through and find max and min and swap
358             double maxValue;
359             double minValue;
360             int minValueIndex;
361             int maxValueIndex;
362 
363             minValue = maxValue = p[start].getX();  //set min and max to first element
364             minValueIndex = maxValueIndex = start;
365 
```

```
366             for (int i = start + 1 ; i <= end ; i++){  //go through array to find max and mins
367               if (p[i].getX() < minValue) {
368                 minValue = p[i].getX();
369                 minValueIndex = i;
370               } else
371                 if (p[i].getX() > maxValue) {
372                 maxValue = p[i].getX();
373                 maxValueIndex = i;
374               }
375             }  //after this for statement, we know the locations of the max and min elements
376
377             swapper = p[start];
378             p[start] = p[minValueIndex];
379             p[minValueIndex] = swapper;
380
381             //this handles the funny case where the max value was in the start position
382             //but then we moved it when we swapped it with the element that actually should
383             //end up in the start position.  That was the swap above, which would leave our
384             //max element in the position with the index of 'minValueIndex'.
385             if (maxValueIndex == start)
386               maxValueIndex = minValueIndex;
387
388             swapper = p[end];
389             p[end] = p[maxValueIndex];
390             p[maxValueIndex] = swapper;
391
392
393           }
394
395         }
396       }//end main for loop
397     } //end forcePush
398
399     /**
400      * Utilitarian method to print out the x value from an array of Points.  This was used
401      * to find the odd case in the Force Push sort where the largest element in the portion
402      * of the array to be sorted was in the first slot and therefore got moved when the
403      * smallest element was swapped into its place.  The method was left in the class for
404      * documentation purposes only.
405      * @param a Point[] of some size to be processed
406      */
407     public void printArray(Point [] a){
408       for (int i = 0 ; i < a.length ; i++) {
409         System.out.printf("%.2f ",a[i].getX());
410       }
411       System.out.println();
412     }
413
414
415 }
```

```
1  //Terry Speicher and Jason Ivey
2
3  /**
4   * Read file into an array and present different sized sub arrays of those points
5   * to each different sort routines and record timed results
6   * @author Terry Speicher
7   * @author Jason Ivey
8   */
9  import java.util.*;
10 import java.util.Arrays;
11 import java.io.*;
12
13 public class SortAlgorithmsTester{
14
15   private static Point[] data = new Point[100000];
16
17   /**
18    * Main body of Sort Tester.
19    * @param args standard header String[]
20    */
21   public static void main(String args[]){
22
23     /** Create int array representing the number of elements that will be taken from the
24       * beginning of the array of random items
25       */
26     int [] testCases = {2,2,3,4,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,
27       200,300,400,500,600,700,800,900,1000,2000,3000,4000,5000,6000,7000,
28       8000,9000,10000,data.length};
29
30     // Set the number of times the data will be tested and averaged.
```

```
31      int iterations = 10;
32      //results table is 9 by however many testCases there are
33      long [][] resultsTable = new long[8][testCases.length];
34      boolean testing = false; //Turn on/off verbose intermediate findings
35
36      read(); //one time read of data[]
37
38      SortAlgorithms sort = new SortAlgorithms();  //create class of sort methods
39
40      //Main counter for determined number of test cases
41      for (int counter = 1 ; counter <= iterations ; counter++){
42
43        //Visual output of loop # currently being processed
44        System.out.println("Iteration#" + counter + " of " + iterations);
45
46        //Print headings
47        if (testing) System.out.print("Sort/#Items,");
48        for (int i = 0 ; i < testCases.length ; i++)
49          if (testing) System.out.print(testCases[i] + ",");
50        if (testing) System.out.println();
51
52        //Print each sort name and test results from each test of n elements
53        if (testing) System.out.print("BubbleSort,");
54        for (int i = 0 ; i < testCases.length ; i++){
55          long startTime = System.nanoTime();
56          sort.bubbleSort(Arrays.copyOfRange(data,0,testCases[i]));
57          long estimatedTime = System.nanoTime() - startTime;
58          if (testing) System.out.print(estimatedTime + ",");
59          resultsTable[1][i] += estimatedTime;
60        }
61        if (testing) System.out.println();
62
63        //Keep printing name of sort and results
64        if (testing) System.out.print("ForcePush,");
65        for (int i = 0 ; i < testCases.length ; i++){
66          long startTime = System.nanoTime();
67          sort.forcePush(Arrays.copyOfRange(data,0,testCases[i]));
68          long estimatedTime = System.nanoTime() - startTime;
69          if (testing) System.out.print(estimatedTime + ",");
70          resultsTable[2][i] +=estimatedTime;
71        }
72        if (testing) System.out.println();
73
74        if (testing) System.out.print("MergeSort,");
75        for (int i = 0 ; i < testCases.length ; i++){
76          long startTime = System.nanoTime();
77          sort.mergeSort(Arrays.copyOfRange(data,0,testCases[i]));
78          long estimatedTime = System.nanoTime() - startTime;
79          if (testing) System.out.print(estimatedTime + ",");
80          resultsTable[3][i] +=estimatedTime;
81        }
82        if (testing) System.out.println();
83
84        if (testing) System.out.print("QuickSort,");
85        for (int i = 0 ; i < testCases.length ; i++){
86          long startTime = System.nanoTime();
87          sort.quickSort(Arrays.copyOfRange(data,0,testCases[i]));
88          long estimatedTime = System.nanoTime() - startTime;
89          if (testing) System.out.print(estimatedTime + ",");
90          resultsTable[4][i] +=estimatedTime;
91        }
92        if (testing) System.out.println();
93
94        if (testing) System.out.print("InsertionSort,");
95        for (int i = 0 ; i < testCases.length ; i++){
96          long startTime = System.nanoTime();
97          sort.insertionSort(Arrays.copyOfRange(data,0,testCases[i]));
98          long estimatedTime = System.nanoTime() - startTime;
99          if (testing) System.out.print(estimatedTime + ",");
100         resultsTable[5][i] +=estimatedTime;
101       }
102       if (testing) System.out.println();
103
104       if (testing) System.out.print("SelectionSort,");
105       for (int i = 0 ; i < testCases.length ; i++){
106         long startTime = System.nanoTime();
107         sort.selectionSort(Arrays.copyOfRange(data,0,testCases[i]));
108         long estimatedTime = System.nanoTime() - startTime;
109         if (testing) System.out.print(estimatedTime + ",");
110         resultsTable[6][i] +=estimatedTime;
111       }
```

```
112         if (testing) System.out.println();
113
114         if (testing) System.out.print("ShakerSort,");
115         for (int i = 0 ; i < testCases.length ; i++){
116           long startTime = System.nanoTime();
117           sort.cocktailShakerSort(Arrays.copyOfRange(data,0,testCases[i]));
118           long estimatedTime = System.nanoTime() - startTime;
119           if (testing) System.out.print(estimatedTime + ",");
120           resultsTable[7][i] +=estimatedTime;
121         }
122         if (testing) System.out.println();
123
124       }
125
126       //In verbose mode, print out time totals table
127       if (testing) {
128         System.out.println("-------------------------Totals over " + iterations + " iterations
              ---------------------------");     System.out.print("Sort/#Items,");
129         for (int i = 0 ; i < testCases.length ; i++)
130           System.out.print(testCases[i] + ",");
131         System.out.println();
132
133         for (int a = 1; a <=7 ; a++){
134           switch (a) {
135             case 1:
136               System.out.print("Bubble,");
137               break;
138             case 2:
139               System.out.print("ForcePush,");
140               break;
141             case 3:
142               System.out.print("Merge,");
143               break;
144             case 4:
145               System.out.print("Quick,");
146               break;
147             case 5:
148               System.out.print("Insertion,");
149               break;
150             case 6:
151               System.out.print("Selection,");
152               break;
153             case 7:
154               System.out.print("Shaker,");
155               break;
156           }  //end switch case
157
158           for (int j = 0 ; j < resultsTable[0].length ; j++){
159             System.out.print(resultsTable[a][j] + ",");
160           }
161           System.out.println();
162
163
164         }
165       }
166
167       //Same loop as above, but print out the averages instead of the total time
168       System.out.println("-------------------------Averages over " + iterations + " iterations
              ---------------------------");
169
170       System.out.print("Sort/#Items,");
171       for (int i = 0 ; i < testCases.length ; i++)
172         System.out.print(testCases[i] + ",");
173       System.out.println();
174
175       for (int a = 1; a <=7 ; a++){
176         switch (a) {
177           case 1:
178             System.out.print("Bubble,");
179             break;
180           case 2:
181             System.out.print("ForcePush,");
182             break;
183           case 3:
184             System.out.print("Merge,");
185             break;
186           case 4:
187             System.out.print("Quick,");
188             break;
189           case 5:
190             System.out.print("Insertion,");
```

```
191             break;
192           case 6:
193             System.out.print("Selection,");
194             break;
195           case 7:
196             System.out.print("Shaker,");
197             break;
198         }  //end switch case
199 
200         for (int j = 0 ; j < resultsTable[0].length ; j++){
201           System.out.print(resultsTable[a][j]/ iterations+ ",");
202         }
203         System.out.println();
204 
205 
206       }
207 
208     }
209 
210     /**
211      * Read in the file with the pairs of point (x,y) coordinates and place in the data[]
212      *
213      */
214     private static void read(){
215 
216       try {
217         File file = new File("100000Points.txt");
218         Scanner scan = new Scanner(file);
219 
220         int x = 0;
221         String line;
222         String token[];
223         while(scan.hasNext()){
224           line = scan.nextLine();
225           token = line.split("\t");
226           data[x] = new Point(Double.parseDouble(token[0]), Double.parseDouble(token[1]));
227           x++;
228         }
229 
230         //confirm number of elements read
231         System.out.println("We have " + x + " points");
232         scan.close();
233       }
234       catch (Exception e){
235         System.out.println("Error in reading file");
236       }
237     }
238 
239 
240   }
```