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.

```
  class Point{
    private double x;
    private double y;
    public Point(double x, double y){
      this.x = x;
      this.y = y;
    }
    public double getX(){
      return this.x;
    }
    public double getY(){
      return this.y;
    }
  }
```

```
//Terry Speicher and Jason Ivey

/**
 * Collection of sorting algorithms to be compared, along with a few utilitarian methods
 * @author Terry Speicher
 * @author Jason Ivey
 */
public class SortAlgorithms{

  /**
   * Constructor doesn't have to do anything
   */
  public SortAlgorithms(){

  }

  /**
   * Not a complete comparison of all items to see if they are in order, but quickly checks
   * consecutive items to make sure that the smaller item is first.
   * @param data Point[] of some size to be processed
   * @return boolean Return true if each consecutive item is less than the one after it.
   */
  public boolean isSorted(Point [] data){

    boolean sortedYN = true;
    int i = 0;
    while (i < data.length - 2 && sortedYN == true) {
      if (data[i].getX() > data[i+1].getX())
        sortedYN = false;
      i++;

    }
    return sortedYN;
  }  //end isSortedYN


  /**
   * Compare the first 100 items in each given Point array to see if they are equal.  Used
   * as a testing routine to work out some bugs when learning how to pass the data by
   * reference.
   * @param data Point[] of some size to be processed
   * @param test Point[] of some size to be processed
   * @return boolean Return true if the first 100 x coordinates are the same in the
   * two arrays
   */
  public boolean looksEqual(Point [] data, Point [] test){

    boolean lookSame = true;
    for (int i = 0 ; i < 100 ; i++)
      if (data[i].getX() != test[i].getX())
      lookSame = false;
    return lookSame;
  } //end looksEqual

//  ***********  Begin BubbleSort *****************

  /**
   * BubbleSort was coded by the authors from scratch.  This is the worst case sort because
   * it does not even adjust the inner loop to start at the first nonsorted element - it
   * always starts at the beginning.
   * @param d Point[] of some size to be processed
   */
  public void bubbleSort(Point [] d){
    for(int x = 0; x < d.length; x++){
      for(int y = 0; y < d.length-1; y++){
        if ( d[y].getX() > d[y+1].getX() ) {
          Point temp = d[y];
          d[y] = d[y+1];
          d[y+1] = temp;
        }
      }
    }
  }

//************* End Bubble Sort *******************

//  *************  Begin QuickSort Methods  ******************

  /**
   * Copied from the textbook.  Improved on by adding a check in the partitioning
   * routine to check ”if (endOfLeftList != scan)” before swapping, because we found
   * that the swap routine was sometimes being called to swap an element with itself if
   * the ”endOfLeftList” equaled ”scan”.
   * @param data Point[] of some size to be processed
   */
  public void quickSort(Point [] data){
    doQuickSort(data,0,data.length-1);


  }  // end quickSort

  /**
   * sub method to start recursion process.  Required to
   * pass in the two indexes for processing the sub array.
   * Initially, we pass in the absolute beginning and
   * absolute ending elements.
   * @param array Point[] to be sorted
   * @param start int that shows where the start of the subarray to be sorted is
   * @param end int that shows where the end of the bubarray to be sorted is
   **/
  private void doQuickSort(Point[] array, int start, int end){
    int pivotPoint;
    if (start < end){
      pivotPoint = partition(array, start, end);
      doQuickSort(array,start,pivotPoint-1);
      doQuickSort(array,pivotPoint+1,end);
    }
  } // end doQuickSort

  /**
   * Partition the given array from array[start] to array[end] and put
   * pivot element in middle.  Then move all elements with values less
   * than the pivot point to the left of the pivot point and move
   * all elements with values greater than the pivot point to the right
   * of the pivot point.
   * @param array Point[] to be sorted
   * @param start int that shows where the start of the subarray to be partitioned is
   * @param end int that shows where the end of the bubarray to be partitioned is
   **/
  private int partition(Point[] array, int start, int end){

    double pivotValue;
    int endOfLeftList;
    int mid;

    mid = (start + end) / 2;

    //take the first element of the array and swap
    //it with the middle element.  I don't know why,
    //except to keep with the idea of using the middle
    //element as the pivot
    swap(array,start,mid);

    pivotValue = array[start].getX();
    endOfLeftList = start;
    for (int scan = start + 1; scan <= end; scan++){
      if (array[scan].getX() < pivotValue){
        endOfLeftList++;
        if (endOfLeftList != scan)
          swap(array,endOfLeftList,scan);
      }
    }//end of for

    swap(array,start,endOfLeftList);

    return endOfLeftList;

  }//end partition

  /**
   * Swap elements array[a] and array[b]
   * @param array Point[] with elements to be swapped
   * @param a index of an element to be swapped
   * @param b index of other element to be swapped
   **/
  private void swap(Point[] array, int a, int b){

    Point temp = array[a];
    array[a] = array[b];
    array[b] = temp;

  }//end swap

// **************  End QuickSort Methods   *****************

//  *************  Begin MergeSort Method  ******************

  /***************************************************************************************
    *    With online help from:
    *    Title: howotodoinjava.com
    *    Author: Lokesh Gupta
    *    Date: October 23, 2015
    *    Availability: http://howtodoinjava.com/algorithm/merge-sort-java-example/
    *
    ****************************************************************************************/
  /**
   * Merge Sort method to sort an array of data points
   * @param data Point[] of some size to be processed
   */
  public void mergeSort(Point[] data){

    if (data.length <= 1){
      return;
    }

    Point[ ] half1 = new Point[ data.length / 2 ];
    Point[ ] half2 = new Point[ data.length - half1.length ];

    System.arraycopy(data, 0, half1, 0, half1.length);
    System.arraycopy(data, half1.length, half2, 0, half2.length);

    mergeSort(half1);
    mergeSort(half2);

    merge(half1, half2, data);
    return;

    /**
     * Main recursive method to merge sorting.
     * divides Points array into smaller pieces.
     * Makes use of merge(Point[],Point[],Point[]) to combine and sort data
     **/
  }
  /**
```

```
   * Merge elements of array[a] and array[b] into result
   * @param half1 Point[] half to be combined with brother in order
   * @param half2 Point[] brother of half1
   * @param result Point[] to be returned when brothers combined and sorted.
   **/
  private static void merge(Point[] half1, Point[] half2, Point[] result){
    int x = 0;
    int y = 0;
    int merge = 0;

    while(x < half1.length && y < half2.length ){
      if(half1[x].getX() < half2[y].getX()){
        result[merge] = half1[x];
        x++;
      }
      else{
        result[merge] = half2[y];
        y++;
      }
      merge++;
    }
    System.arraycopy(half1, x, result, merge, half1.length - x);
    System.arraycopy(half2, y, result, merge, half2.length - y);
  }
  //  *************  End mergeSort Methods  *****************

  //  *************  Begin insertionSort Methods  *****************

  /*************************************************************************************
    *    With online help from:
    *    Title: http://www.java2novice.com/
    *    Author: N/A
    *    Date: N/A
    *    Availability: http://www.java2novice.com/java-interview-programs/insertion-sort/
    *************************************************************************************/


  /**
   * Sort Point[] by inserting unsorted points into the correct positions in sorted Point[]
   * @param array Point[] array to be sorted
   **/

  public static void insertionSort(Point array[]) {
    int n = array.length; // Limiter initalized to length of Point[] array
    for (int j = 1; j < n; j++) {
      Point key = array[j]; // Key to comparison
      int i = j-1; // Interloop counter, one less than 'j'
      while ( (i > -1) && ( array[i].getX() > key.getX() ) ) {
        array [i+1] = array [i];
        i--;
      }
      array[i+1] = key; // Inserting key into sorted portion of array.
    }
  }

  //  *************  End insertionSort Methods  *****************


  //  *************  Begin selectionSort Methods  *****************

  /*************************************************************************************
    *    With online help from:
    *    Title: Sorting.java
    *    Author: Lewis/Loftus
    *    Date: N/A
    *    Availability: http://www.ics.uci.edu/~stasio/winter06/Lectures/Lec7code/ComparableExample/Sorting.java
    *************************************************************************************/

  /**
   * Sorts the Point array of objects using the selection sort algorithm.
   * @param input Point[] array to be sorted
   **/

  public static void selectionSort (Point[] input)
  {
    int min; // Min value
    Point temp; // Temp storage of Points

    for (int index = 0; index < input.length-1; index++)
    {
      min = index; // Initializing min
      for (int scan = index+1; scan < input.length; scan++)
        if (input[scan].getX() < input[min].getX())
        min = scan; // update min

      // Swap the values
      temp = input[min];
      input[min] = input[index];
      input[index] = temp;
    }
  }


//  **************   Cocktail Shaker Sort   ********************
//http://www.javacodex.com/Sorting/Cocktail-Sort

  /**
   * Cocktail Shaker Sort
   * @param array Point[] of some size to be processed
   */
  public static void cocktailShakerSort( Point[] array ){
    boolean swapped;
    do {
      swapped = false;
      for (int i =0; i<=  array.length  - 2;i++) {
        if (array[ i ].getX() > array[ i + 1 ].getX()) {
          //test whether the two elements are in the wrong order
          Point temp = array[i];
          array[i] = array[i+1];
          array[i+1]=temp;
          swapped = true;
        }
      }
      if (!swapped) {
        //we can exit the outer loop here if no swaps occurred.
        break;
      }
      swapped = false;
      for (int i= array.length - 2;i>=0;i--) {
        if (array[ i ].getX() > array[ i + 1 ].getX()) {
          Point temp = array[i];
          array[i] = array[i+1];
          array[i+1]=temp;
          swapped = true;
        }
      }
      //if no elements have been swapped, then the list is sorted
    } while (swapped);
  }

  /**
   * Force Push Sort was coded by Terry Speicher to show how he *thought* the Cocktail
   * Shaker was supposed to work.  It is kind of a ”double ended” selection sort.
   * @param p Point[] of some size to be processed
   */
  public void forcePush(Point [] p){

    Point temp = new Point(0.0,0.0);

    Point swapper = new Point(0.0,0.0);

    for (int start = 0, end = p.length -1 ; start < end  ; start++, end--){
      if (start != end) { //only 1 element.  We are done.
        if ( (end - start) == 1) {
          //only two elements left
          //compare and swap if necessary
          if (p[start].getX() > p[end].getX()){
            temp = p[start];
            p[start] = p[end];
            p[end] = temp;
          }
        } else {
          //more than 2 elements - go through and find max and min and swap
          double maxValue;
          double minValue;
          int minValueIndex;
          int maxValueIndex;

          minValue = maxValue = p[start].getX();  //set min and max to first element
          minValueIndex = maxValueIndex = start;

          for (int i = start + 1 ; i <= end ; i++){  //go through array to find max and mins
            if (p[i].getX() < minValue) {
              minValue = p[i].getX();
              minValueIndex = i;
            } else
              if (p[i].getX() > maxValue) {
              maxValue = p[i].getX();
              maxValueIndex = i;
            }
          }  //after this for statement, we know the locations of the max and min elements

          swapper = p[start];
          p[start] = p[minValueIndex];
          p[minValueIndex] = swapper;

          //this handles the funny case where the max value was in the start position
          //but then we moved it when we swapped it with the element that actually should
          //end up in the start position.  That was the swap above, which would leave our
          //max element in the position with the index of 'minValueIndex'.
          if (maxValueIndex == start)
            maxValueIndex = minValueIndex;

          swapper = p[end];
          p[end] = p[maxValueIndex];
          p[maxValueIndex] = swapper;


        }

      }
    }//end main for loop
  } //end forcePush

  /**
   * Utilitarian method to print out the x value from an array of Points.  This was used
   * to find the odd case in the Force Push sort where the largest element in the portion
   * of the array to be sorted was in the first slot and therefore got moved when the
   * smallest element was swapped into its place.  The method was left in the class for
   * documentation purposes only.
   * @param a Point[] of some size to be processed
   */
  public void printArray(Point [] a){
    for (int i = 0 ; i < a.length ; i++) {
      System.out.printf("%.2f ",a[i].getX());
    }
    System.out.println();
  }


}
```

```
//Terry Speicher and Jason Ivey

/**
 * Read file into an array and present different sized sub arrays of those points
 * to each different sort routines and record timed results
 * @author Terry Speicher
 * @author Jason Ivey
 */
import java.util.*;
import java.util.Arrays;
import java.io.*;

public class SortAlgorithmsTester{

  private static Point[] data = new Point[100000];

  /**
   * Main body of Sort Tester.
   * @param args standard header String[]
   */
  public static void main(String args[]){

    /** Create int array representing the number of elements that will be taken from the
      * beginning of the array of random items
      */
    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,
      200,300,400,500,600,700,800,900,1000,2000,3000,4000,5000,6000,7000,
      8000,9000,10000,data.length};

    // Set the number of times the data will be tested and averaged.
```

```java
    int iterations = 10;
    //results table is 9 by however many testCases there are
    long [][] resultsTable = new long[8][testCases.length];
    boolean testing = false; //Turn on/off verbose intermediate findings

    read(); //one time read of data[]

    SortAlgorithms sort = new SortAlgorithms();  //create class of sort methods

    //Main counter for determined number of test cases
    for (int counter = 1 ; counter <= iterations ; counter++){

      //Visual output of loop # currently being processed
      System.out.println("Iteration#" + counter + " of " + iterations);

      //Print headings
      if (testing) System.out.print("Sort/#Items,");
      for (int i = 0 ; i < testCases.length ; i++)
        if (testing) System.out.print(testCases[i] + ",");
      if (testing) System.out.println();

      //Print each sort name and test results from each test of n elements
      if (testing) System.out.print("BubbleSort,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.bubbleSort(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[1][i] += estimatedTime;
      }
      if (testing) System.out.println();

      //Keep printing name of sort and results
      if (testing) System.out.print("ForcePush,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.forcePush(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[2][i] +=estimatedTime;
      }
      if (testing) System.out.println();

      if (testing) System.out.print("MergeSort,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.mergeSort(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[3][i] +=estimatedTime;
      }
      if (testing) System.out.println();

      if (testing) System.out.print("QuickSort,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.quickSort(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[4][i] +=estimatedTime;
      }
      if (testing) System.out.println();

      if (testing) System.out.print("InsertionSort,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.insertionSort(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[5][i] +=estimatedTime;
      }
      if (testing) System.out.println();

      if (testing) System.out.print("SelectionSort,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.selectionSort(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[6][i] +=estimatedTime;
      }
      if (testing) System.out.println();

      if (testing) System.out.print("ShakerSort,");
      for (int i = 0 ; i < testCases.length ; i++){
        long startTime = System.nanoTime();
        sort.cocktailShakerSort(Arrays.copyOfRange(data,0,testCases[i]));
        long estimatedTime = System.nanoTime() - startTime;
        if (testing) System.out.print(estimatedTime + ",");
        resultsTable[7][i] +=estimatedTime;
      }
      if (testing) System.out.println();

    }

    //In verbose mode, print out time totals table
    if (testing) {
      System.out.println("-------------------------Totals over " + iterations + " iterations ------------------------------");     System.out.print("Sort/#Items,");
      for (int i = 0 ; i < testCases.length ; i++)
        System.out.print(testCases[i] + ",");
      System.out.println();

      for (int a = 1; a <=7 ; a++){
        switch (a) {
          case 1:
            System.out.print("Bubble,");
            break;
          case 2:
            System.out.print("ForcePush,");
            break;
          case 3:
            System.out.print("Merge,");
            break;
          case 4:
            System.out.print("Quick,");
            break;
          case 5:
            System.out.print("Insertion,");
            break;
          case 6:
            System.out.print("Selection,");
            break;
          case 7:
            System.out.print("Shaker,");
            break;
        }  //end switch case

        for (int j = 0 ; j < resultsTable[0].length ; j++){
          System.out.print(resultsTable[a][j] + ",");
        }
        System.out.println();


      }
    }

    //Same loop as above, but print out the averages instead of the total time
    System.out.println("-------------------------Averages over " + iterations + " iterations ----------------------------");

    System.out.print("Sort/#Items,");
    for (int i = 0 ; i < testCases.length ; i++)
      System.out.print(testCases[i] + ",");
    System.out.println();

    for (int a = 1; a <=7 ; a++){
      switch (a) {
        case 1:
          System.out.print("Bubble,");
          break;
        case 2:
          System.out.print("ForcePush,");
          break;
        case 3:
          System.out.print("Merge,");
          break;
        case 4:
          System.out.print("Quick,");
          break;
        case 5:
          System.out.print("Insertion,");
          break;
        case 6:
          System.out.print("Selection,");
          break;
        case 7:
          System.out.print("Shaker,");
          break;
      }  //end switch case

      for (int j = 0 ; j < resultsTable[0].length ; j++){
        System.out.print(resultsTable[a][j]/ iterations+ ",");
      }
      System.out.println();


    }

  }

  /**
   * Read in the file with the pairs of point (x,y) coordinates and place in the data[]
   *
   */
  private static void read(){

    try {
      File file = new File("100000Points.txt");
      Scanner scan = new Scanner(file);

      int x = 0;
      String line;
      String token[];
      while(scan.hasNext()){
        line = scan.nextLine();
        token = line.split("\t");
        data[x] = new Point(Double.parseDouble(token[0]), Double.parseDouble(token[1]));
        x++;
      }

      //confirm number of elements read
      System.out.println("We have " + x + " points");
      scan.close();
    }
    catch (Exception e){
      System.out.println("Error in reading file");
    }
  }


}
```