Homework Turnin

Email:	rgalanos@fcps.edu
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	03-035
Receipt ID:	8d09fd353d977ab7c631a88cd2336a72

Warning: Your turnin is 1 day late. Assignment 03-035 was due Thursday, November 3, 2016, 11:30 PM.

Replacing prior submission from Fri 2016/11/04 11:08am.

Turnin Successful!

The following file(s) were received:

Merge_Quick.java (4333 bytes)

```
    /*
    Calls methods in the classes Merge and QuickSort.
    Students are to write the Merge and QuickSort classes.
    */
import java.util.*;
import java.io.*;
public class Merge_Quick
{
   public static void main(String[] args) throws Exception
   {
      int n = (int)(Math.random()*100);
      double[] array = new double[n];
      for(int k = 0; k < array.length; k++)
         array[k] = Math.random();
      print(array);
      MergeSort.sort(array);
      //QuickSort.sort(array);
      print(array);
      if(check(array))
         System.out.println("In order!");
      else
         System.out.println("oops!");
   }
   public static void print(double[] array)
   {
      for(double theNumber : array )     //doing something to each
         System.out.println(theNumber);
      System.out.println();
   }
   public static boolean check(double[] a)
   {
      boolean bool = true;
      for(int x=0;x<a.length-1;x++)
      {
         if(a[x]>a[x+1])
            bool = false;
      }
      return bool;
   }
}
```

```
////////////////////////////////////////////////
// from Lambert & Osborne, p. 482 - 485
class MergeSort
{
   private static final int CUTOFF = 10;  // for small lists, recursion isn't worth it
   public static void sort(double[] array)
   {
      double[] copyBuffer = new double[array.length];
      mergeSortHelper(array, copyBuffer, 0, array.length - 1);
   }

   private static void mergeSortHelper(double[] array, double[] copyBuffer,
                                                              int low, int high)
   {
         // if ( high - low  < CUTOFF )              //switch to selection sort  when
            // Selection.sort(array, low, high);     //each list gets small enough
         // else
      if (low < high)
      {
         int middle = (low + high) / 2;
         mergeSortHelper(array, copyBuffer, low, middle);
         mergeSortHelper(array, copyBuffer, middle + 1, high);
         merge(array, copyBuffer, low, middle, high);
      }
   }

   public static void merge(double[] array, double[] copyBuffer,
                                    int low, int middle, int high)
      // array          array that is being sorted
      // copyBuffer     temp space needed during the merge process
      // low            beginning of first sorted subarray
      // middle         end of first sorted subarray
      // middle + 1     beginning of second sorted subarray
      // high           end of second sorted subarray
   {
         /* write the merge method  */

      copyBuffer = array;
      int low2 = middle+1;
      int index = low;

      while(low != middle+1 && low2 != high+1)
      {
         if(array[low]<=array[middle+1])
         {
            copyBuffer[index]=array[low];
            low++;
            index++;
         }
         else
         {
            copyBuffer[index] = array[low2];
            low2++;
            index++;
         }
      }

      while(low != middle+1)
      {
         copyBuffer[index]=array[low];
         low++;
         index++;
      }
      while(low2 != high+1)
      {
         copyBuffer[index] = array[low2];
         low2++;
         index++;
      }

      array = copyBuffer;
   }
}

////////////////////////////////////////////////
class QuickSort
{
   public static void sort(double[] array)
   {
      quickSort(array,0,array.length-1);
   }
```

```
   private static void quickSort(double[] array, int first, int last)
   {
      int splitPt;
      if(first<last)
      {
         splitPt = split(array, first, last);
         quickSort(array,first,splitPt-1);
         quickSort(array,splitPt+1,last);
      }
   }
   private static int split(double[] info, int first, int last)
   {
      int splitPt = first;
      double pivot = info[first];

      while(first<=last)
      {
         if(info[first] <= pivot)
            first++;
         else if (info[last]>=pivot)
            last--;
         else
         {
            swap(info, first, last);
            first++;
            last--;
         }
      }
      swap(info, last, splitPt);
      splitPt = last;
      return splitPt;
   }
   private static void swap(double[] array, int a, int b)
   {
      double temp = array[a];
      array[a]=array[b];
      array[b]=temp;
   }
}
```