Lab 5: Sorting Algorithms

SE2205: Data Structures and Algorithms using Java – Winter 2015
Discussion: March 5th **Due**: Sunday March 15

Modified and reviewed by Alexandra L'Heureux (alheure2@uwo.ca) and Vasundhara Sharma (vvasundh@uwo.ca) , for more information please send an email to Alex (TA, SE2205) or Vasu (TA.SE2205)

A. Rationale and Background

Sorting algorithms are an extremely important aspect of computer science, and tend to be one of the most common aspects of interview questions. This is due to their complexity in requiring the software engineer (you!) to understand concepts such as ADTs, algorithm complexity, recursion versus iteration, and serial base cases.

In this lab you will be tasked with implementing your own sorting algorithms and comparing them against the Java built-in sorting algorithms. The three algorithms you will be responsible for in this lab are: Merge Sort, Quick Sort and Insertion Sort. Merge Sort and Quick Sort will be compared against the Java built-in versions of Merge and Quick Sort. For Insertion Sort, you will be required to visualize the sorting process.

B. Evaluation

You will get credit for your lab when you demonstrate it and get your TA's approval. Submitting your lab work online is required; failure to do so will result in a grade of 0%. Additionally, your TA will ask some questions about your coding during demonstration and you need to articulate your idea clearly. In place of questions, the TA may also ask you to modify your program in a small way - it is expected that you have sufficient understanding of the code to do so.

Your mark will be determined as follows:
Completion: Program Demonstration of Base Requirements (50%)
Understanding: Program Demonstration of Changes or Response to Questions (50%)

Please note that you may only demonstrate the code you have previously submitted to OWL.
Please ensure that the file naming conventions are being followed. You should be able to rename your classes once they are done by right clicking on them, selecting *refactor* and *rename*.

1. Submission Instructions

Number of files : All files created and used in this lab
Files to be submitted : use the following format for all your files
your_uwo_user_name_lab05_XXXX.java,

C. Lab Questions

1. Merge and Quick Sort

a. *ArrayGenerator* Class

This class contains a single static method to generate an integer array.

```
public static Integer[] generateRandomArray(int arraySize, int
maxValue)
```

The parameters designate the size of the array and the maximum value each instance within the array can take.

b. IntegerComparator Class

Java provides an interface called java.util.Comparator. This interface requires you to implement the following method:

```
int compare(T o1, T o2)
```

The IntegerComparator class should implement java.util.Comparator and overwrite the compare method. The new method should compare it's two arguments in order; returning a negative, zero, or positive number for less than, equal to, and greater than respectively.

c. MyArrays class

Java has a utility class called Arrays (java.util.Arrays). This class has two methods:

```
public static void sort(int[] a)
public static <T> void sort(T[] a, Comparator<? super T> c)
```

The first method implements a tuned Quick Sort and the second method implements a modified Merge Sort. You will need to design two similar methods which implement your own Quick and Merge Sort algorithms.

The method declarations in MyArrays are: `public static void sort(int[] a)`

```
public static void sort(Integer[] a, IntegerComparator c)
```

In the first method, you must implement your own Quick Sort algorithm. In your second method you must implement your own Merge Sort algorithm.

This class will represent your main entry to your program (thus, the main method). You will need to do the following within your main:

1. Generate an array of integers

2. Make three copies of the array (note: for both the MyArrays and Arrays Merge Sort implementations, you will need the array to be of type Integer. For the Quick Sort implementations, they should be of type int)

3. Use one copy for each algorithm

4. Validate your implementations of Merge and Quick Sort (do they work?) on a small array size (less than 50 objects)

    a. Print to the screen the initial values in the array

    b. Print to the screen the Merge Sorted array

    c. Print to the screen the Quick Sorted array

    d. Optional: Print to the screen the Java Merge Sorted array

    e. Optional: Print to the screen the Java Quick Sorted array

5. Performance Evaluation: Using a large array size (on the order of 1 million, or larger, for both the array size and maximum value of each integer), test the performance of your implementation versus the Java built-in implementations. Below is a code snippet on how to take time stamps before and after your algorithm runs. (Please refer to Section D to adjust the memory size of the Java Virtual Machine to be able to allocate an array of sufficient size).

```
long start, end, time;
start = end = 0;

start = System.currentTimeMillis();

/*
     Your code here!!!
 */

end = System.currentTimeMillis();

time = end - start;
```

a. Print to the screen the size of the array and max size of each int

b. Print to the screen the running time of your merge sort

c. Print to the screen the running time of your quick sort

d. Print to the screen the running time of Java's merge sort

e. Print to the screen the running time of Java's quick sort

2. Sorting Visualization

a. Introduction

In Question 2 you will be visualizing the process of an *Insertion Sort*. Below is the array you will use to visualize this process:

```
int[] numbers = {31, 19, 76, 24, 94, 99, 21, 74, 40, 73}
```

There are two ways we are expecting you to do this:

1. Printing each step of the insertion sort (i.e. printing the content of the numbers array at each step of the sort. This is the minimum we are expecting and you will be able to receive full marks doing it this way. (see sub-section **b** for further explanation).
2. Visualizing the algorithm using histograms. If you successfully complete the lab using a GUI, you will receive up to 5 bonus marks. (see sub-section **c** for further explanation).

b. Visualizing with the Command Line

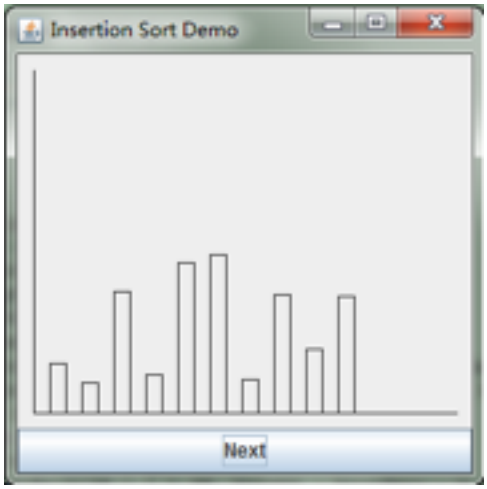
The minimum requirements of this lab is to visualize the algorithm using the command line. For this we are expecting you to use the integer array shown above. Then, print each iteration of the insertion sort to the command line. Each iteration can be viewed as the current state of the integer array. Below is an example of what we are expecting this to look like:

```
Step 0: 31 19 76 24 94 99 21 74 40 73
Step 1: 19 31 76 24 94 99 21 74 40 73
Step 2: 19 31 76 24 94 99 21 74 40 73
Step 3: 19 24 31 76 94 99 21 74 40 73
Step 4: 19 24 31 76 94 99 21 74 40 73
Step 5: 19 24 31 76 94 99 21 74 40 73
Step 6: 19 21 24 31 76 94 99 74 40 73
Step 7: 19 21 24 31 74 76 94 99 40 73
Step 8: 19 21 24 31 40 74 76 94 99 73
Step 9: 19 21 24 31 40 73 74 76 94 99
```

c. Visualizing using Histograms

*This aspect of the lab is **optional**. However, if you are successful in implementing Question 2 with a GUI and histogram, you will receive 5 bonus marks.*

The value of each number can be represented in a histogram by a rectangle. A sample output is shown here:



When the 'Next' button is clicked on, the order of the rectangles will change according to the process of running insertion sorting step by step.

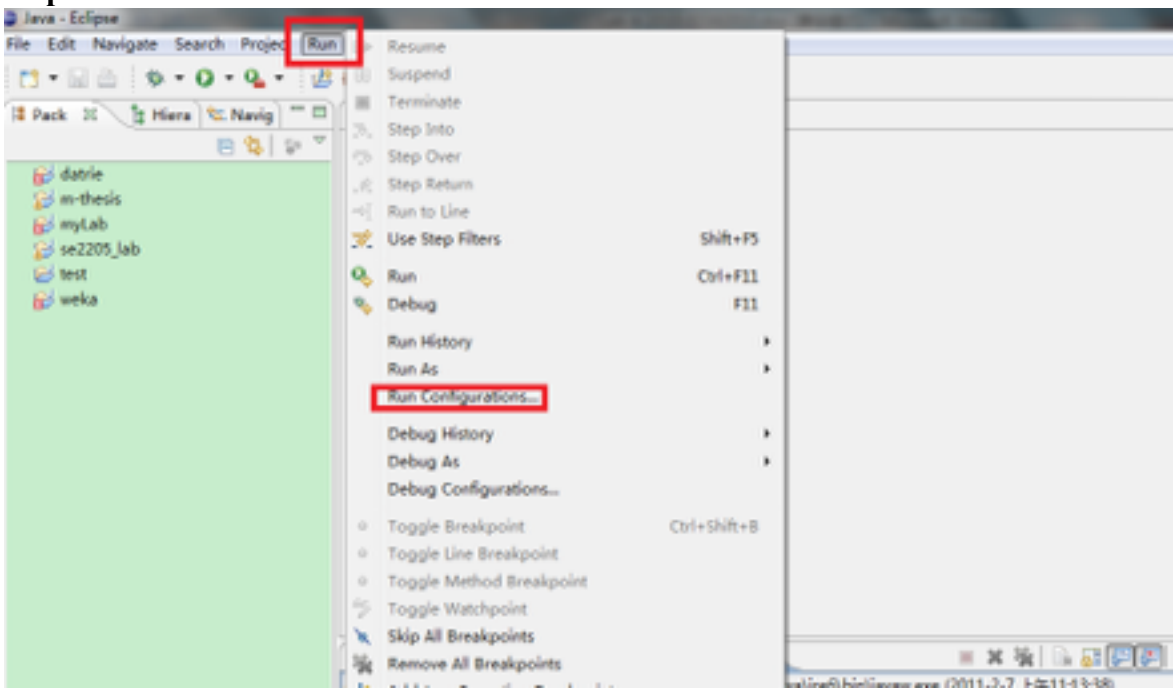
You can use `java.awt.Graphics.drawRect(int x, int y, int width, int height)` and `java.awt.Graphics.drawline(int x1, int y1, int x2, int y2)` to draw rectangles and lines. Please note that the coordinate layouts of the parameters begin at (0,0) in the top left.

You may use any classes you wish to accomplish this. My suggestion is to have one class that extends a JPanel, and another that extends a JFrame. The JFrame class has the main method. In the JFrame class, create the JPanel class and place it within the frame. *You do not need to do it this way, this is simply a suggestion.*

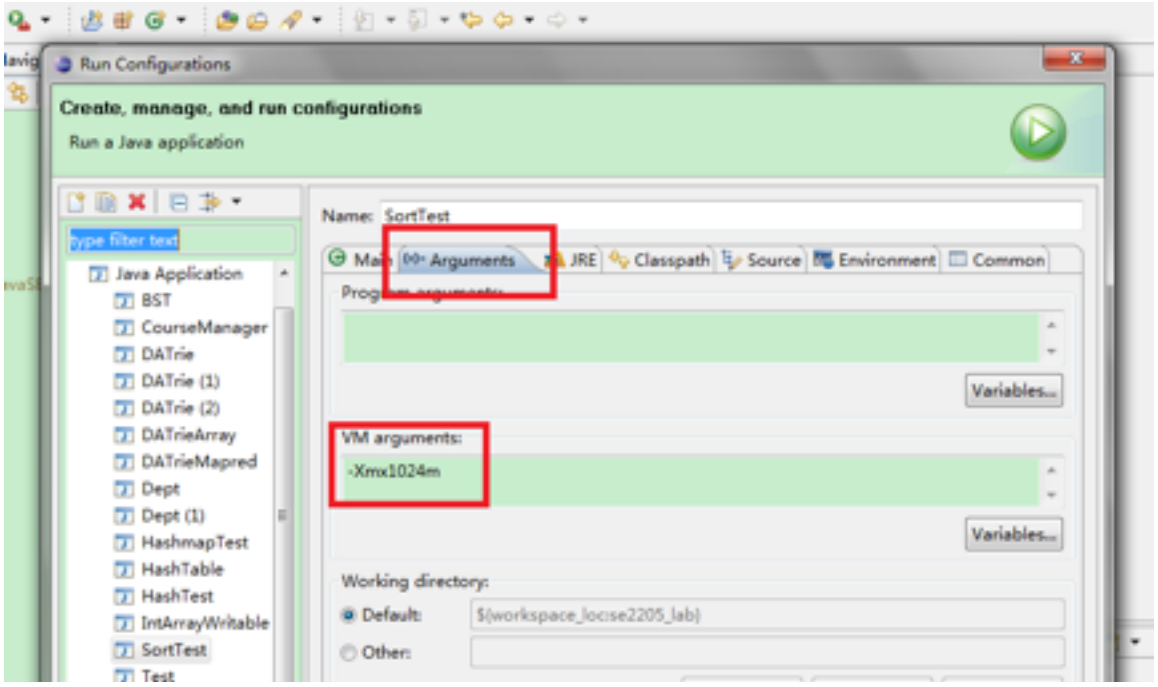
```
public class HistogramPanel extends JPanel implements ActionListener

	// save an index for what step you're on
	// allocate and initialize int[] array
	// constructor etc,

	@Override
	public void paint(Graphics g) {
		super.paint(g);
		// draw lines for each value
	}

	@Override
	public void actionPerformed(ActionEvent arg0) {

		// call the next step insertion sort
		this.repaint();
	}
}
```

D. Increase JVM memory

Step 1:



Step 2:



NOTE: If you cannot provide 1024 MB for your JVM on your machine, please try 512MB or similar smaller numbers. The number of a power of 2 is preferred.

D. Learning Checkpoint

This lab should have introduced you to these concepts of Java:

1. Understanding how different sorting algorithms work
2. Evaluating performance between java implemented methods, and your own
3. Further GUI building with respect to sorting visualization