CSE 11 Winter 2016 **START EARLY!**
Program Assignment #5 (200 points)
Due: 12 March 2016 at 1159pm (2359, Pacific Standard Time)

PROGRAM #5 : AnimatedSort

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

This program focuses on integrating a great deal of what has been covered in this course. This is a large program, but it builds on Program #4.

You are to create a Swing-based interface that animates the steps of two sorting algorithms: merge sort and bubble sort. The following screenshot shows a histogram of values that are unsorted.

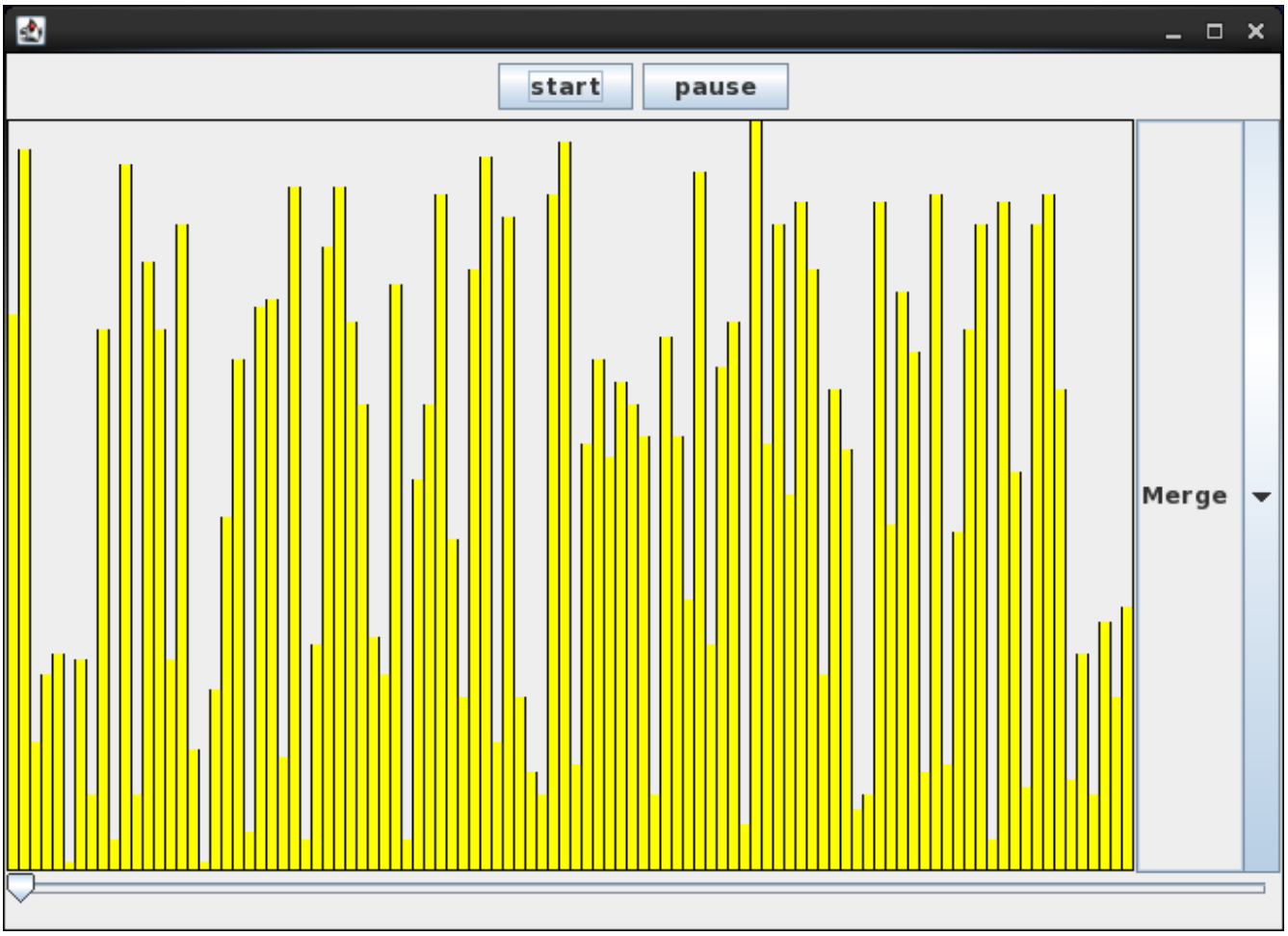

Figure 1 - An Unsorted Array of Double. Displayed as a Histogram

The various components of the interface

- At the top are two control buttons – start and pause. Start begins the sorting process, pause freezes the sorting process (and unfreezes when pressed again). Hitting the Start button at anytime, restarts the sort from the beginning.

- At the bottom is a speed slider. The slider controls animation performance
- At the right-hand side, a JComboBox enables the user to choose which sorting algorithm to use
- The histogram display, shows the current state of the partially sorted array.

Your Program: AnimatedSort

Basic Idea:
The basic idea of the animation is to highlight at least one of the values being compared during a sort. We will illustrate with a `testinput` that has seven numbers

```
5 10 12 15.0 20 5 10 15.0 17 2 8 6 4 16
```

It should create the following graphical output when the command is entered

```
% java AnimatedSort 300 200 testinput
```

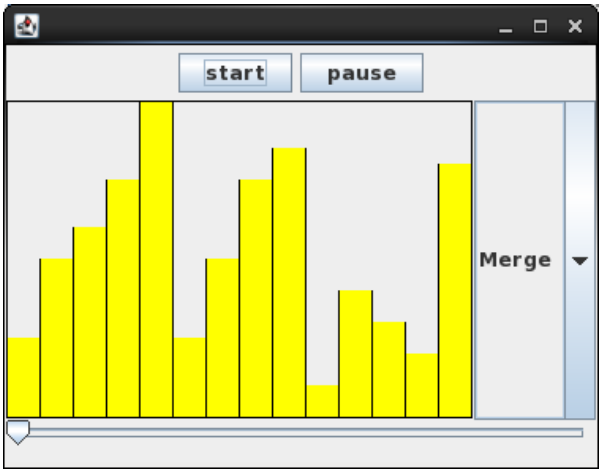

Figure 2 - Example with 14 numbers. The chart is approximately 300x200 pixels. The buttons, slider and ComboBox take additional space.

As in assignment 4, you start the program with command-line arguments. The first two numbers of the command are the width and height of the histogram part of the display (in pixels). The third argument is the name of the input file that contains only non-negative doubles.

- Each bar in the histogram has identical width
- Each bar is colored yellow
- The largest value is full height
- All other values are of proportional height with respect to the largest value. For example, the values 5 are ¼ the height of the largest value (20)
- There are black lines between the bars and black framed rectangle around the histogram
- This histogram display is to be no wider than the width specified.

What follows are several screenshots that show different aspects of the program.

A. Choosing between two sort algorithms.

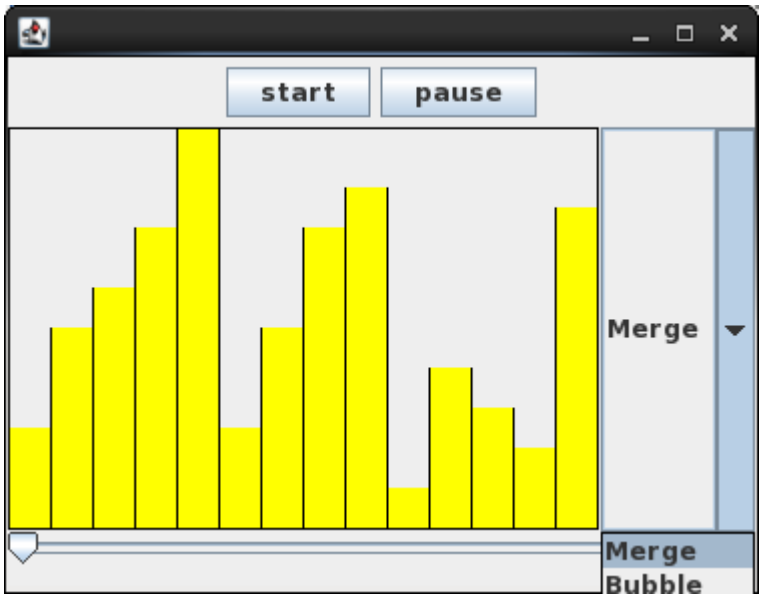

Figure 3 - Choosing Bubble or Merge sorting algorithm

B. Starting the Sort – Press the Start Button

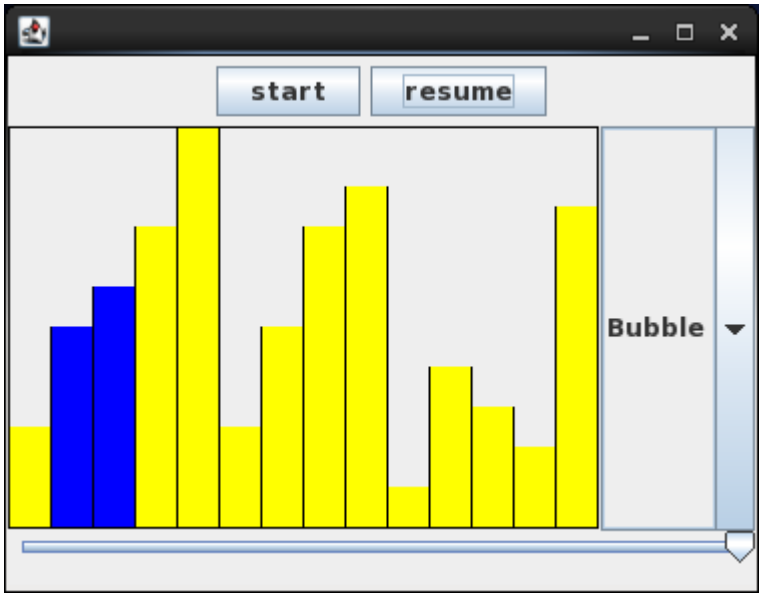

Figure 4 - Start up of bubble sort. The blue bars are the array values that are currently being compared. In this picture, the pause button has been pushed. Notice that its label has also changed to "resume". Also the SpeedSlider at the bottom is set to slowest animation speed.

C. Fully-Sorted Array

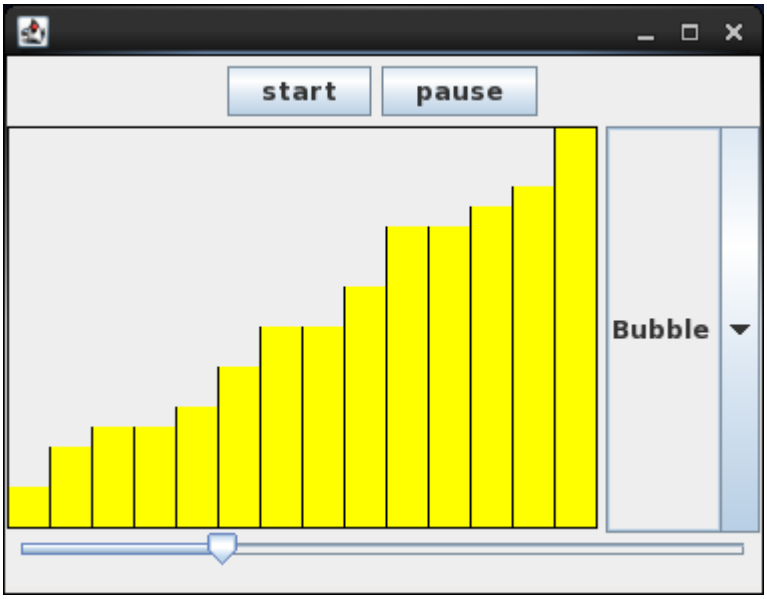

Figure 5 - Fully Sorted Display

D. Partially-Sorted Array using Merge Sort

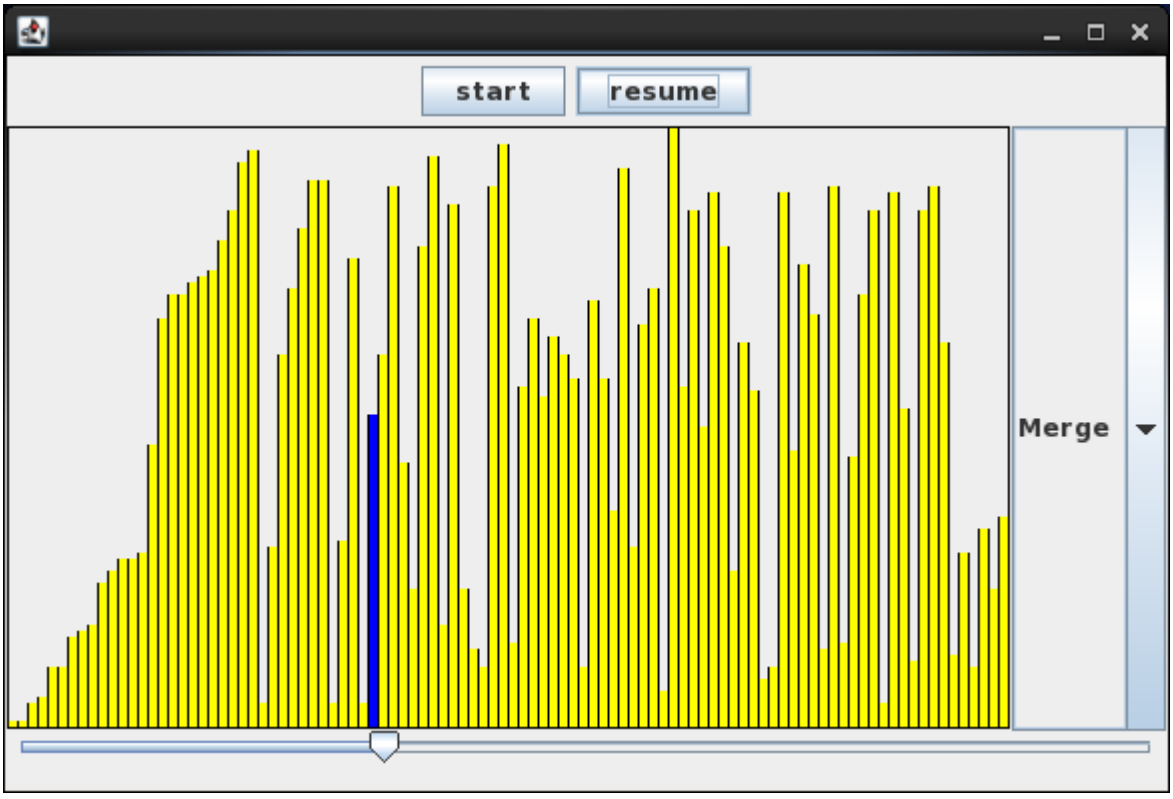

Figure 6 – Partially-sorted Using the Merge Sort Algorithm. Note that in this screenshot only one blue bar is showing. Also notice that about the first 25% of the values are in sorted order.

Java Class Files

- Sorter.java

- MergeSort.java
- BubbleSort.java
- AnimatedSort.java

The above files are the ONLY files that you will turn in. If you want to define other classes, define them as helper classes within these files.

Suggested Development Plan

This is a more complicated program than your previous program. The following is a discussion of a possible way to go about developing the code.

IMPORTANT: After you have debugged each step, save a copy of all of your source files.

Step 1. Convert YOUR Histogram.java to AnimatedSort.java.
Some notable changes in requirements from Histogram

- AnimatedSort will sort non-negative doubles instead of integers.

You will likely need to modify some of the logic of your existing code.

Step 2. Create a Sorter inheritance hierarchy, using Sorter as the parent class.

It is worthwhile noting that your sort methods only differ in their sorting algorithms. Any other setup is common to both classes and can be accomplished in Sorter.

The first step is to create a BubbleSort program that is NOT a graphics program. BubbleSort is much simpler than Merge Sort and can be coded in just a small number of lines. You have many choices about how to create the constructor for your sorters. There is nothing wrong with your constructor re-reading the input file.

Have your main method of your BubbleSort program read the file, print out unsorted elements, sort the elements, and then print the result. Check that your output is actually sorted.

Do not create static methods (other than main). Your graphics program will need to be able create instances of BubbleSort (and eventually MergeSort) to sort an input file.

Step 3. Connect your BubbleSort to your AnimatedSort class.

This is the first “tricky” part of your program. And it requires some thought on your part. The ultimate goal is to animate each “step” that a sort takes. For this part of the development, don’t worry about how to create the “blue” bars. But you do have to figure out how you reflect the changes made to your data array (as it is being sorted) are shown graphically.

Some things to note when you are working this out

- You will be updating the graphical location of bars on your display. Try to think about the bars as logical objects. For example, the first bar corresponds to the index=0 entry of your data array.
- Draw a picture of your classes (e.g., MyWindow, Grid, Sorter, and BubbleSort) and figure out what messages you need to send to instances of classes. For example, a BubbleSort instance is going to need to be able to "tell" the Grid instance that the bar that was located at index K has been moved to index J.
- Don't be afraid to generate small methods that simplify the logic of your code. Also, even though Grid is defined in your AnimatedSort class, you can use it in your Sorter classes.
- Try to not have your sorter know too much about the actual details of the graphics display.

A program at this stage, might simply load the file, display it graphically, create a BubbleSort instance and then request that BubbleSort instance to sort itself. When sorting, each time you swap two elements in the array, tell the Grid instance that the elements have been swapped so that the graphical display can update itself.

Step 4 – Slow down that animation and change bar colors

If you have completed step 3, you have essentially a histogram class that will very quickly animate a bubble sort. Now you want to slow it down and change the colors of the two elements being compared.

- Look at some of the Threading examples and look up the sleep method (it takes milliseconds as arguments.)
- Modify your sort method to
    - Change the colors of the bars that your comparing to blue
    - Sleep (pause execution) for 50 – 100 ms.
    - Change the colors of the bars back to yellow
    - Swap elements, if needed (updating the GUI if the elements were actually swapped)

Step 5 – Add a Start Button to your GUI to control when the sort begins. Add a Slider.

This sounds simple (adding the Button is straightforward). And Adding the Slider is straightforward, too. When the start button is pressed, the `sort` method of your sorter will need to be invoked. This takes place in an `actionPerformed` method of your graphics program.

However, if you just invoke sort directly, the `actionPerformed` method will not return until the sort has completed. That will make your graphics processing stall because you are blocking the Event Dispatch Thread. To get around this problem, you can use Threads. Your `actionPerformed` method can create a new thread. And then start that thread running (the `Runnable` that is used as the argument of the Thread constructor would invoke your Sorter's sort method in the Runnable's `run()` method).

When you have completed Step 5, you are well on your way to the end of assignment.

Step 6 - Add Speed Control and Pause Control to your Program

You added the slider in step 5, but didn't necessarily connect it to anything. You should solve this problem at this stage. You need to invoke a method on your Sorter that will update the time to wait between steps. You should also code the logic of the pause button. There are no "magic" methods in the Thread class that will enable you to stop your sorting code and then start it again. Instead, figure out where to code this logic in your sort method.

Verify that the speed affects your code.

Step 7 – Enable restart of a Sort from the beginning

When you restart a sort, it's often simplest to just build a new Sorter instance and perhaps a new Grid instance However, you will want to make sure that the sorting thread of an existing instance (if it exists) exits (and therefore the run() method of the thread completes). If you have pause working, it should be simple to add an additional capability to "stop" the sort wherever it is and exit the sorting method.

Step 8 – Build Merge Sort

At this stage you can build a non-graphical version of Merge sort. Debug it like you did when you initially created BubbleSort. Then add the ability to update the graphical representation. When that works, change your code to use MergeSort instead of BubbleSort. If you do this correctly, it should only be a few lines of code that change on your AnimatedSort class. If you find yourself changing many lines, think about how to use Sorter references instead of BubbleSort references in your code wherever you can.

A note on Merge sort animation. Merge sort requires a temporary array during the merging process. If you look at Figure 6, only a single blue bar is shown. This is perfectly fine. Only one is showing because this particular step of the merge has not quite completed and the "other" blue bar is actually in the temporary array and is not being displayed.

Step 9 – Create the ability to choose between Merge and Bubble Sort in your GUI
The final step is to add the ability to choose between a bubble sort and a merge sort. Add the JComboBox to the right-hand side of your GUI.

Step 10 – Make sure no GUI errors on standard error occur
For example, make certain that

- Pressing Pause before any sort has started (or any sort has completed) does not result in any errors
- Moving the speed slider before the first time a sort has been performed does not cause any errors.

Below is a list of requirements. If a "Reason Statement" is given, then that statement should go after "Reason:" in the usage statement. You may add other information to the reason (for example, including the error string (using `getMessage()` of the `Exception` class). If you add more information, please keep it on the same line.

I. Errors that must be handled by your program

Most of these requirements are very similar to your Histogram project. Some small details (like Histogram is changed to AnimatedSort and integers are changed to doubles) do exist.

Requirement 1. When an error is detected, your program should output to *stderr* (the standard error stream) a two-line statement of the following form and then exit with a non-zero exit code (use `System.exit()`)

```
Usage: AnimatedSort [ width height] filename
Reason: a reason for the usage statement
```

Requirement 2. Your program should take either one (1) or three (3) arguments. If one argument is given, the single argument is interpreted as a file name. If three arguments are given, the first is interpreted as the width of histogram, the second as the height, and the third is the filename.

Reason statement: Improper number of arguments

Requirement 3. Bars must show at least 2 yellow pixels in width (a vertical black line also consumes a pixel). That is, when graphed, at least yellow pixels are visible

Reason statement: Too many bins for pixel width

Requirement 4. All elements in the input file are doubles. If anything other than a double is present in the file, then the usage statement should be given

Reason statement: Non-double in file

Requirement 5. All doubles in the input file are non-negative

Reason Statement: Value is negative

Requirement 6. Program must handle any file system errors. including file does not exist, and incorrect permissions

Reason statement: File system error

Requirement 7. Program must handle empty files

Reason statement: No values to display

Requirement 8. Program must handle obviously incorrect grids (e.g. with non-positive width or height)

Reason statement: Invalid Dimensions

Requirement 9. Program must handle when width or height given on command line is not an integer
Reason statement: Invalid Integer

Requirement 10. If more than one error is possible given specific inputs (eg. invalid grid and invalid data), only ONE of the errors should be given (the first one your program finds)

II. Functional Requirements of your Program

There are many additional requirements. A number of the first requirements are similar to your Histogram code.

Requirement 1. The JPanel that holds the bars can be no larger the (width + 2) x (height + 2) pixels. This allows you to treat the width x height as the area for the display of bars. The extra pixels allows you to frame the histogram in a black-colored rectangle. The window will be significantly larger than width x height to accommodate your slider, buttons, and combo box. Let Swing size the rest of these components for you.

Requirement 2. The Histogram must be framed with a black rectangle drawn with lines that are one (1) pixel wide

Requirement 3. A single line, colored black, this is one pixel wide, separates each bar. The top of each bar does not have to be framed in black. Two black lines next to each other (each being 1 pixel wide to create a visual that is a 2-pixel-wide line is incorrect)

Requirement 4. The bars are colored yellow

Requirement 5. The largest value in the input file is full height. All other graphed values have pixel heights that are proportional to this maximum value

Requirement 6. If width and height are not given on the command line, you are to use the following defaults

- width = 600
- height = 400

Requirement 7. The window should be large enough to show the entire Histogram without resizing by the user. You will need to override the definition of `getPreferredSize` in the Grid class so that Swing can properly size your histogram and then properly size your window for you.

Requirement 8. ~~The main program should wait for the user to hit a key before exiting. You may use the same code as given in the example.~~ The user simply hits the 'X' in the window frame to close the window.

Requirement 9. All of your classes and methods must be commented using Javadoc-style comments. Your name and email should go in as Javadoc style at the top of your program. See your previous assignments for examples.

Requirement 10. Your program should never end with a stack trace. Any errors should be gracefully handled with usage statements.

Requirement 11. Swing should generate no errors. Other than your usage statement there should be no output on standard output or standard error. Please note, printing to standard output is encouraged while you are developing your code.

Requirement 12. The program shows the unsorted array when it first starts.

Requirement 13. The program does not begin the animated sorting process unless the Start button is pressed by the user

Requirement 14. Bar coloring. In the case of bubble sort, both of the array elements that are being compared to determine if a swap should be made, are colored blue. In the case of merge sort, both of the array elements that are being compared should be colored blue. For Merge sort, because of the use of temporary storage and a likely implementation of merge sort, only one of these bars may actually be visible at particular phases. At least one of these bars should be visible during the merge.

Requirement 15. **Pause button**. At startup, the pause button displays the word "pause". When the pause button is pressed, it displays the word "resume". When pressed again, it displays "pause". When you Start a sort the pause button should be forced to display "pause".

If a sort is running, pause should prevent the sort from making progress (you cannot accomplish this by using methods in the thread class). Pressing the button again should resume the sort at exactly the same place.

Hint: don't attempt to use any methods from the Thread class to "freeze" your sort, write this logic directly into your code

Requirement 16. Speed Slider. The slider controls how quickly the sort progresses in time. The simplest way is for the value of the slider to govern how long sort waits (sleeps) at each compare step before going onto the next step. When the slider is all the way to the left, the program should run at its fastes (sleep interval is smallest), when all the way to the right, it should be at its slowest

Fastest: no more than 100 comparisons/sec
Slowest: no slower than 1 comparison every 1 seconds

The speed slider takes effect immediately.

Requirement 17. Restart. When the start button is pressed, the sort should begin again at the unsorted array. Any running sort should be terminated. The current state of the speed slider should be used to govern the speed of the restarted sort.

Requirement 18. Sort Selection. A ComboBox allows the user to select which sort to use. It only

takes affect when the start button is pressed. While a sort is active, selecting the sort has no affect on the running code.

Requirement 19. Window Resize. Resizing does not affect the width of bars or computed size of the JPanel. You do not have to handle window resizing.

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR4" program that will turn in the files
AnimatedSort.java Sorter.java BubbleSort.java MergeSort.java

No other files will be turned in and they **must be named exactly as above.** BundlePR5 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

$ **/home/linux/ieng6/cs11wa/public/bin/bundlePR5**

The output of the turnin should be similar to what you have seen in your previous programming assignments

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

My Grid doesn't exactly fit in the JPanel, what should I do? When your Grid is added to the Window, the Window should be packed, using the pack() method defined in AWT. Your Grid class will need to override the definition of `getPreferredSize()`

What public methods can I add? Anything you want (methods and constructors). Don't use public class or instance variables. You may use protected and private instance variables. You may define public constants (static final) as you see fit.

Can I add classes? Yes. If you add them, they should be "helper" classes and coded within one of the four source files that you turn in.

If the user just gives bad input, do I just print usage and reason statement? Yes. Make sure it is output to the standard error output and that your code exits with a non-zero exit code

Do I have to check for all kinds of crazy inputs? Programming with Exceptions can help make this a fairly easy task.

Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon. This is a complex program. The professor's solution was more than 700 totals lines when added together.

Do I have to use Threads? It might be possible to do this program without creating a Thread to run your sort(), but threads is much simpler.

Do I need to use synchronized methods? No. While there are multiple threads (main, event dispatch, sort), there shouldn't be any race conditions.

Do I have to use an inheritance hierarchy for my Sorters? No. You could define an interface instead and have your sorters implement the interface. Inheritance makes a bit more sense, but using interface isn't "wrong".

How does my sorter know about the graphics Grid and its methods for displaying the histogram? Short answer. You tell it. Either through methods you design, or through the constructor. You then have to decide *How* you have your sort method instruct the Grid to change color and location of certain bars. You might create a helper class that represents a single graphed bar. One (good) approach is to have your sorter tell a bar to sets its color. Your sorter should be able to tell your Grid to place a particular bar at a specific index.

Can I use MY histogram code from the previous assignment? Yes.

Can I use my friend's histogram code from the previous assignment? No.

If my histogram code didn't work right in program 4 how can I do program 5? Come talk to your tutors/TA/Professor to get your code working. We can show you the professor's solution. We can also help you debug your Histogram code to get it right. However, tutors/TAs are instructed to NOT help you debug your histogram code if you ask after Tuesday of 10th week.

MergeSort and BubbleSort examples are on the web, can I just copy/paste them? No. But you may (and should) use them as reference for your own implementation. This assignment requires you to understand how these algorithms work. Re-coding them is good practice (and doesn't really take all that long since you have many worked examples).

I don't understand paintGraphics or how the original program really works, can you explain? `paintComponent()` is how AWT/Swing redraw a graphics screen. AWT decides when to ask your component to repaint itself (via paintComponent()). It might decide to this if you move the window on your screen, if it becomes uncovered via mouse click or any number of (events) reasons. paintComponent() can be called at anytime. In the sample code, the method repaint() is called. This puts a repaint() request onto Event Dispatch Queue (controlled by AWT) and then returns. When the Dispatch thread gets around to handling the repaint request, your paintComponent method will be invoked.

What does SwingUtiltilities.invokeLater() do? When you give invokeLater an object reference (say its called myGrObj), a request is placed onto the Event Dispatch Thread to invoke the run() method of myGrObj (only objects that implement Runnable are valid objects for invokeLater). At some point in time "later", the run() method of myGrObj will be called.

What IS the Event Dispatch Thread? Graphics programs have all kinds of events (mouse movement, keyboard, windows closing, resizing, etc). The Dispatch thread is in charge of informing Objects that a particular event has actually occurred. When an event occurs (say a mouse press), the event is put onto the event dispatch queue (We call this enqueing a request to the dispatch thread). Think of the queue as an inbox of work to-be-done. The Event Dispatch Thread takes an event off its dispatch queue (out of its inbox) and tells every Object that has registered interest in that particular event that the event (and its particulars) has occurred. Say it is a MousePress event, and three Objects have interest in the MousePress event, each of the three Object's appropriate handler is called. They are called one-at-a-time. When all of the handlers of all the Objects have completed for a particular event, the dispatch thread goes on to the next event in its dispatch queue. Note. This program is not defining any events to handle. (you will in Program 5)

How should I exit the program? The user should hit the 'X' in the Window bar supplied by the windowing system.

I have a question on input files. Are values separated by new lines or spaces or something else? They are separated by white space. If you are properly using Scanner, then programs with multiple lines of input or all values on a single line should work.