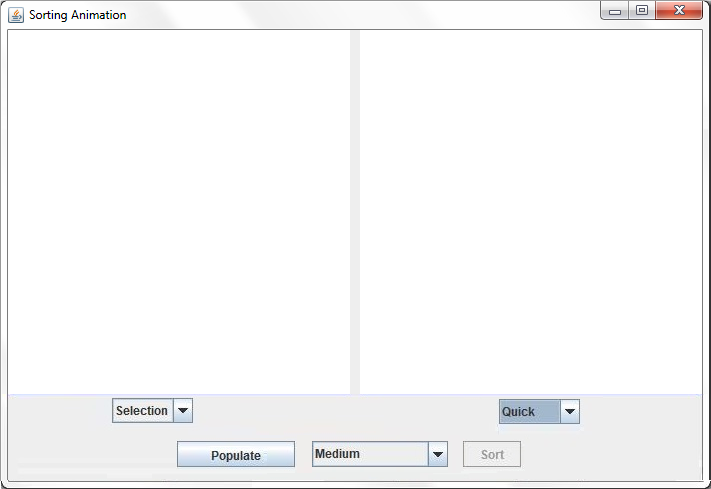
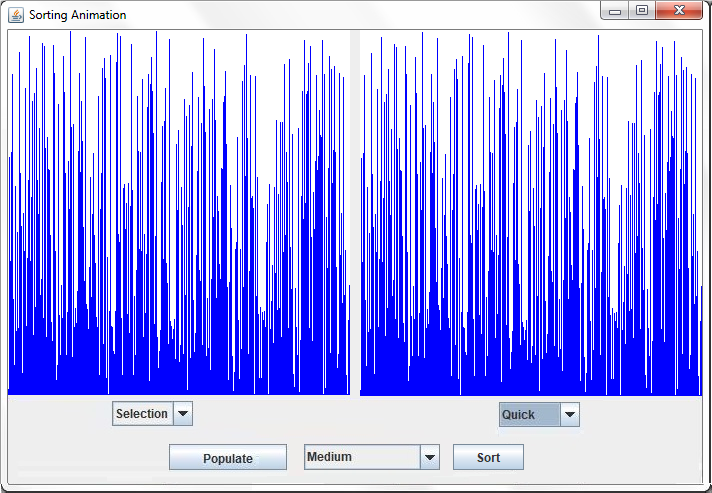
**CSCI 470-1/502-1 Assignment 5 – Concurrency and Animation Summer 2019**

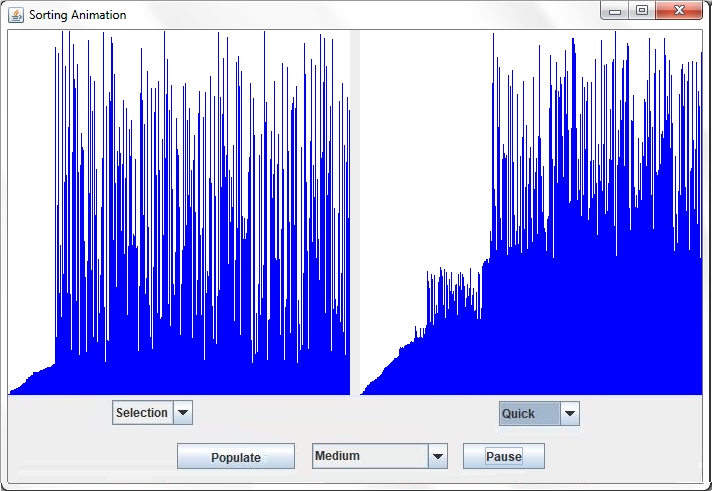
**150 points**

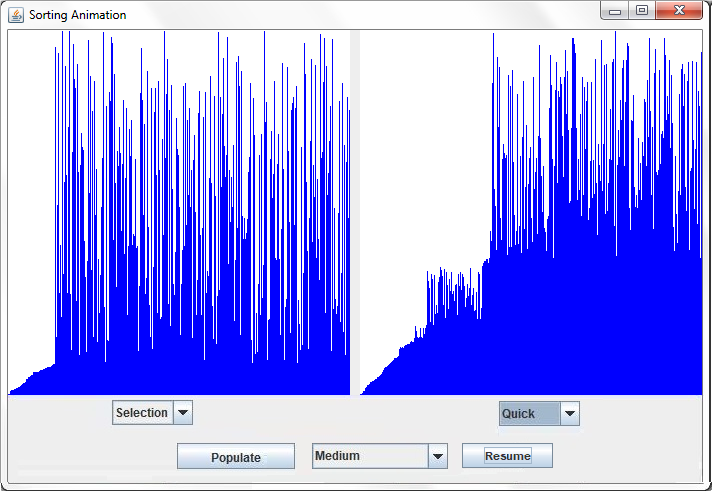
For this assignment you will write a Java application using graphics and concurrency (multithreading) to display an animated version of various common sorting algorithms. The basic version of this application is fairly simple and straightforward. Also, please note the extra credit opportunity listed near the end of this document.



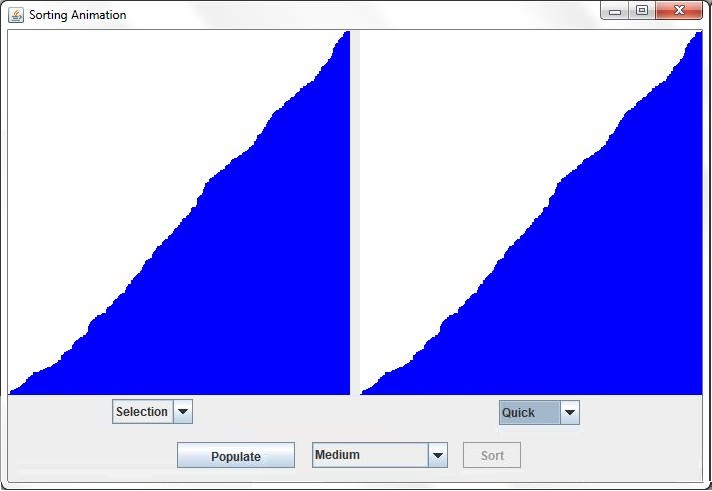
*The sorting animation application shown at startup.*

*  
After the arrays have been populated and the sorts chosen, but before sorting has begun.*

*  
The sorts in progress.*



*The sorts have been paused.*

*  
After both sorts have completed.*

**The Classes**

Create an app class named SortAnimationApp to represent the sorting animation application and inherits from JFrame. It should have a pair of SortPanel objects as data members. The class should have a custom constructor that passes a title bar string to the superclass constructor and adds the two SortPanel objects to the application’s layout. It should also have a main() method that creates an instance of the application, sets the default close operation and size, and makes the application visible. Be sure that you run it using SwingUtilities like this:

SwingUtilities.invokeLater(new Runnable()

{

@Override

public void run()

{

new SortAnimationApp();

}

});

There should be a single set of buttons controlling the sorting. This means there should be one Populate button, a JComboBox from which the user can choose the speed of the sorting animations (Slow, Medium or Fast) and the Sort/Pause/Resume button as shown in the illustrations above.

The SortPanel class encapsulates the display and controls for the sorting animation and inherits from JPanel. As shown above, each SortPanel should have just a JComboBox or JSpinner to choose which sorting algorithm to use. It also has a SortAnimationPanel as one of its data members. The constructor for the class should manage the layout for the controls and animation panel. This class can also handle events from the buttons.

The SortAnimationPanel class will display the visual results of sorting and should inherit from JPanel and implement the Runnable interface. (Hint: You may want to make this an “inner” or “nested” class of SortPanel. Doing so will give this class access to the SortPanel data members without any need for an object reference.)

Define an object reference to an array of integers (either in the SortPanel class or in the SortAnimationPanel class). When the “Populate” button is pressed, create a new array of integers; the size of this array should be the width of the animation panel. Fill the array with random values in the range 0 to the height of the animation panel (Hint: Review the Java Random class and its method nextInt() for a good way to do this). Once the array is populated, call repaint() for both panels to display the same array values, disable the “Populate” button, and enable the “Sort” button.

You should override the paintComponent() method for the SortAnimationPanel class. After you call the superclass version of the method, get the dimensions of the panel and clear it. Then, if the array of integers is not null, draw a series of lines on the panel surface representing the array elements. The height of each line should represent the integer value of the corresponding integer array element.

When the “Sort” button is pressed, disable the “Sort” button and create a new Thread object from your Runnable SortAnimationPanel. Call the Thread object’s start() method to start the sort in both panels.

The run() method should call the appropriate sort method based on the user’s selection for each of the two panels to sort the array of integers in each panel in ascending order. As the sorting algorithm progresses, call repaint() any time two elements are swapped. After each pass through the sort algorithm’s outer loop (or equivalent), have the thread sleep for a short time (say 100 milliseconds). Effectively, the thread will pause for a short time every time an element is put into sorted order.

To be clear, program the GUI so that **both** panels are populated with exactly the same set of vertical lines with a single Populate button press and that they all begin at exactly the same time with a single Sort button press.

You should provide ***at least*** three different sorting algorithms that the user can choose from for each panel. Possible candidates include selection sort, insertion sort, bubble sort, heap sort, quicksort, shell sort, merge sort, etc. Pick at least one of the more efficient (i.e., better than O(N2) – Big O notation below) algorithms. C++ and/or Java code for these sorts can all be found online.

When the array is completely sorted, enable the “Populate” button so that the user can start the process over.

**Extra Credit Opportunities**

Add a third sort panel. (5 points)

Force mutual exclusivity between each of the two or three sort algorithms. In other words, the user is unable to choose the same sort for more than one sort panel. (10 points)

Submit your .java files on Blackboard as before.

**Example of Big O Notation**

O(N2) represents an algorithm whose performance is directly proportional to the square of the size of the input data set. This is common with algorithms that involve nested iterations over the data set. Deeper nested iterations will result in O(N3), O(N4), etc.

bool ContainsDuplicates(IList<string> elements)

{

for (var outer = 0; outer < elements.Count; outer++)

{

for (var inner = 0; inner < elements.Count; inner++)

{

// Don't compare with self

if (outer != inner)

{

if (elements[outer] == elements[inner])

return true;

}

}

}

return false;

}