COSC-111-2021F - Introduction to Computer Science I

Lab09: Visualizing and listening to sorting algorithms (Due 11/1)

Please read these instructions fully and carefully before diving into writing code or even before starting to think about your code: they contain clarifications and examples that you need to understand before moving on.

We haven't covered MergeSort yet in class, but we'll do so on Monday. Meanwhile you can start on the InsertionSort part of this lab. Because of this issue, this lab assignment is due on *Sunday 11/1 at 11.59pm* (but there will be another lab out next Friday)

Lab Prep

You must have done and *understood* the readings on InsertionSort and MergeSort and for the lab, or this assignment will be quite hard. You will need the `StdAudio.java` file in your project `src` directory. You can download it from https://introcs.cs.princeton.edu/java/stdlib/StdAudio.java.

Visualizing and listening to sorting algorithms

You will write two programs that visualize the progress of sorting algorithms, respectively InsertionSort and MergeSort, and "play the music" of these algorithms. To get an idea of what these programs will do, see the example videos `InsertionSort.mov` and `MergeSort.mov` linked below (the first one is 5 minutes long, which you will never get back, so maybe just watch 20-30 seconds of it at the beginning, towards the middle, and towards the end). Along the way, you will also develop a small class so we keep practicing the concepts of classes, objects, instance variables, instance methods, and so on.

The Value class

You will start by writing a `Value` class which will represent the values in the arrays that your sorting algorithms will sort. A `Value` object is characterized by two quantities:

1. an `int` instance variable named, for example, `val`, which stores, upon construction of the object, the value that the caller wants this object to represent; and

2. a *sound*, stored as a `double[]` instance variable named, e.g., `sound`, which contains *samples*, where a sample is defined as in the paragraph "Sampling" on page 156 of S&W (within the subsection "Standard Audio" in Sect. 1.5, which was assigned as a reading for this lab). The number of the samples in the sound, i.e., the length of the array `sound`, is a *constant* decided by you. We suggest to start with something like 4410 samples, which corresponds to a sound of .1 seconds of duration (to understand why, you need to have read the aforementioned paragraph), and then try different values while testing your program . The array is initially empty (i.e., the constructor for `Value` only constructs a `new` array of the length decided by you, and assigns its address to `sound`, but it *does not fill the array*, thus the constructor does not take any parameter that describes the sound). The array will be filled later by the method `setSound()` (described below).

The `Value` class must have the following public instance methods:

- `int getValue()`: returns the value of `val`;
- `double[] getSound()`: returns the value of `sound`;
- `int compareTo(Value other)`: compares the value represented by `this` object with the value represented by the `other` object and returns an appropriate value (either -1, or 0, or 1) to represent the relationship between these two values.
- `void setSound(int hz)`: sets the values in `sound` to be samples from a sine wave with a frequency of `hz` Hertz. The technical details are a bit overwhelming, but the `for` loop in the code at the end of the "Sampling" paragraph shows how to fill the array `sound` with the correct samples (with minor changes). The `setSound` method should *not* play the sound: it just fills the instance variable `sound` with the right values.
- `void draw(...)`, where we did not list any parameter, but your method will *definitively take parameters*: draw `this` object using methods from the `StdDraw` class. Your method will have to take parameters to know *where* to draw `this` object, and possibly other information needed for drawing (e.g., the color to draw it in, or anything else that you think is important). The examples videos linked below show the values being drawn as vertical columns, where the height of a column is proportional to the value represented by the object. Please, *be creative*: you do not have to draw the `Value` objects as vertical bars: you can use triangles, circles, flowers, text, anything you would like, and you can use different colors, provided that the drawing for a `Value` object has *size proportional to the value of the variable `val` of `this` object*. For an example, see the `InsertionSort-30-ExemplarySolution.mov` video linked at the bottom of the page, which shows the work of a student in Spring'20.

We strongly suggest that you create a test program to test the `Value` class by creating objects of this class, setting sounds, getting and playing the sound (using the `StdAudio.play` method), drawing the object, get and print the value, ...

InsertionSort

Now that the `Value` class is ready, you can start implementing your programs to visualize and listen to the sorting algorithms. You will start by writing a program `InsertionSort`. The program takes exactly one positive integer `n` as the only command line argument. It prints error messages and exits if it receives a different number of arguments or if the received integer is not positive.

It then creates `n` objects of class `Value`, and stores them into an array. Each of these objects represents a random value between 1 and `n` (the value should be passed to the constructor; because the values are chosen at random, it is possible that multiple objects will represent the same value).

The program will set the sound of each `Value` object in such a way that the object with the smallest value (which you can find by just iterating over the elements of the array) has a sound corresponding to a concert A (i.e., `hz=440`), and every other `Value` object has a sound corresponding to a higher note the larger is the difference between its value `x` and the smallest value `min`, according to the formula `hz = 440 * Math.pow(2, (x - min) / 12.0)`.

Your program will then draw the objects one by one in the order that they appear in the array and, after drawing each `Value` object, it will play the sound of the object just drawn.

Then, it will sort the elements of the array using the InsertionSort algorithm, modified in such a way that, *every time* that two values are switched of position, your program will redraw all the objects one by one and play their sounds, using a different color just for the object that is being moved "to the left" of the array. Your program should not use double buffering and pauses from `StdDraw`, but it will use `StdDraw.clear()` in strategic points. The example video `InsertionSort.mov` below shows you an example, where the original position of the object being moved to the left of the array is colored in yellow, and the object being moved is colored in orange. Once again, *be creative*: you do not have to draw all the elements on a line: you could arrange them in a circle, or on an arc, or in any other creative way. For an example, see the `InsertionSort-30-ExemplarySolution.mov` video linked at the bottom of the page, which shows the work of a student in Spring'20.

When run with a sufficiently high value of the command line argument, this program will allow you to hear and visualize the progress of the sorting algorithm, which is quite musically pleasing (or not?)

MergeSort

The `MergeSort` program is similar to the `InsertionSort` program above, except that it should redraw the `Value` objects and play their sounds after each "merge" phase and every time the algorithm hits base case, using a different color to draw the objects involved in the merge phase or in the base case (in other words, the objects in the sub-array being looked at by this call to the recursive mergeSort method should be drawn in a different color). The example video `MergeSort.mov` below shows an example.

When run with a sufficiently high value of the command line argument, this program will allow you to hear and visualize the progress of a recursive algorithm.

Also, the fact that the video for `MergeSort` is about 4 minutes long and the one for `InsertionSort` is about 20 minutes, should make you notice the difference of time complexity between the two algorithms.

Submission

Please submit only the files `InsertionSort.java`, `MergeSort.java`, `StdDraw.java`, `StdAudio.java`, and `Value.java` which you can find in the `src` folder (a.k.a., directory) of your project folder, which should be indicated in the "Project" panel on the left of the IntelliJ interface, next to the label denoting your whole project.

- InsertionSort-20-ExemplarySolution.mov October 23 2020, 10:09 AM
- InsertionSort-30-sound.mov April 1 2020, 8:25 PM
- MergeSort-30-sound.mov April 1 2020, 8:25 PM

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Sunday, November 1, 2020, 11:59 PM
Time remaining	3 days 5 hours
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet.

◄ SW-StdAudioReadingsForLab09-pp155to159

Jump to...

Practice exercises ►

Amherst College
www.amherst.edu
askIT@amherst.edu
(413) 542-2000