Programming Assignment 2

CSC 225 Summer 2023

June 26, 2023

1 Visualizing Sorting Algorithms

In this assignment you will implement a number of sorting algorithms. Code for visualizing these sorting algorithms has been written for you so that you are able to see them work in real time.

You must implement the following sorting algorithms which sort a given input sequence, S, of distinct integers.

Sorting algorithm	Run-time efficiency
`MergeSort`	$O(n \log n)$
`QuickSort`	$O(n^2)$
`InsertionSort`	$O(n^2)$
`RadixSort`	$O(dn)$, where d is the maximum number of digits it requires to represent any integer in S

You may not use library sorting calls to help you sort (e.g. `Collections.sort()`). You must write each sorting algorithm yourself.

Each list will return the sorted sequence of S from smallest integer at position 0 to largest integer at position $n-1$, where n is the number of integers in S.

2 Logistics and Details

You will be sorting sequences of type `List<Integer>`. You can review the `List` interface and its methods here: `https://docs.oracle.com/javase/8/docs/api/java/util/List.html`.

You will receive five Java files: `P2Tester.java`, `GraphAlgorithms.java`, `PixelWriter.java`, `PixelVertex.java`, `PixelGraph.java`. All five files are required in order to run the visualizing program. However, you will be working only in `GraphAlgorithms.java`; when submitting your work, submit only `GraphAlgorithms.java`.

2.1 `P2Tester.java`

`P2Tester.java` contains the main code for creating the visualization program. You are not expected to understand all the code, but if you are a curious you are welcome to play around with it. However, your `GraphAlgorithms.java` submission must run and be correct using the **original** version of `P2Tester.java`.

To run the tester, compile all above Java files and then simply

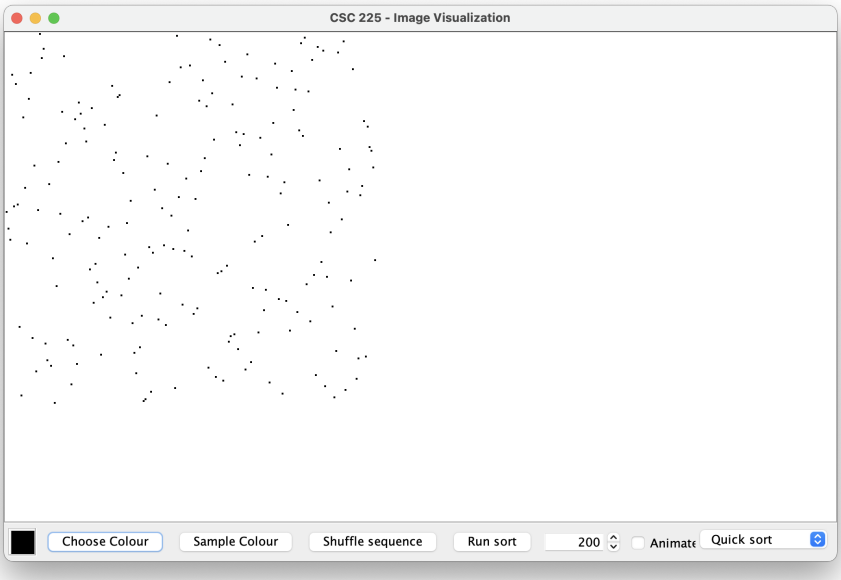
```
java P2Tester
```

This should open a new window with a white canvas and dots (see Fig. 1).

The 'random' black pixels on the white canvas represent the `List<Integer>` you will be sorting, where the index of each `Integer` is plotted on the x-axis and the value on the y-axis. When sorted correctly, they should form a diagonal line from the top left corner to the bottom right corner (see Fig. 2).

The number of pixels in this line is initially set to 200 and can be adjusted using the textbox located in the bottom menu in between the 'Run Sort' button and the 'Animate?' checkbox. The textbox will accept

Figure 1: Opening screen of `P2Tester`



integers between 1 and 2000 inclusive, although **it is recommended that you stay in the 100-300 range** due to zoom constraints. Changing the number of pixels in the line will automatically reshuffle the sequence.

To manually reshuffle the sequence, click on the `Shuffle Sequence` button.

Select which sort you would like to test using the drop-down menu in the far-right of the bottom menu. Then click on the `Run sort` button. If you have the `Animate?` checkbox checked when you run the sort, you will be able to see an animation of the sort.

You are able to zoom in and out using the 'wheel' on your mouse or trackpad or using the up and down arrow keys.

2.1.1 Bill Bird's Legacy, Acknowledgements, and Flood Fill

The base code for this assignment was written by Bill Bird with modifications from Anthony Estey and B. Jiao.

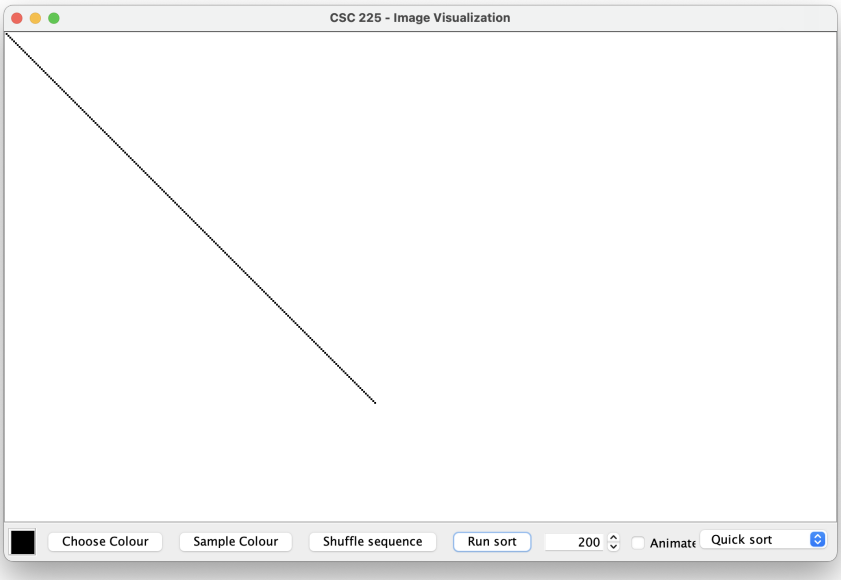
It was originally a program where students implemented the common image processing method 'Flood Fill' using Depth-First Search (DFS) and Breadth-First Search (BFS). Although I have modified it to visualize sorting algorithms, I have decided to keep the DFS and BFS functionality for those who are interested in implementing these methods on their own time. More explicitly, the DFS and BFS portion of the code in `GraphAlgorithms.java` **will not be marked**.

To use the BFS and DFS options in the tester, select BFS or DFS from the drop-down menu, choose a colour using `Choose Colour` or `Sample Colour` and then click on any pixel in the image (note: sometimes you may need to click 2-3 times for it to work). Note: since neither BFS nor DFS will be implemented, it will not work until you implement these methods first.

2.2 `GraphAlgorithms.java`

`GraphAlgorithms.java` is where you will be writing all of your code. **Do not change the name or any of the provided code in this file.** Simply add your solutions to the existing code. In particular, the method headers given **cannot** be changed. You are allowed (and encouraged) to write helper functions where appropriate.

Figure 2: A correctly sorted sequence results in a diagonal line.



You will be responsible for drawing your sequence on the canvas at various stages of each sort. You can do this by calling the `drawSequence(List<Integer> sequence, PixelWriter writer)` method. You must pass the same `writer` from each sort's method header arguments when calling `drawSequence()`. This method will automatically update and draw pixel positions on the canvas as your list elements change position.

Hint: in MergeSort, QuickSort, and RadixSort call `drawSequence()` after you have finished merging or concatenating sub-lists.

2.3 Hints

MergeSort, QuickSort, and InsertionSort should be straightforward to implement provided you have been following along in lectures.

2.3.1 RadixSort Hints

RadixSort is significantly harder to implement, not because the algorithm is more difficult, but because it is difficult to translate into Java code. The following hints are meant as a suggestion and are not mandatory to implement, although they will hopefully make your life much easier and make your code easier to read when being graded.

- Consider starting by writing the helper method `BucketSort` first.
- Consider temporarily converting all integers in S into their `String` representations.
- You may want to look into Java's `String`: in particular the `format` method: `https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#format-java.util.Locale-java.lang.String-java.lang.Object...-`.
- You may want to familiarize yourself with Java's `Integer`, in particular the `valueOf` method: `https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#valueOf-int-`.
- Recall that generic-typing of `List<E>` allows for lists of lists to be made: `List<List<E>>`.

2.3.2 General Hints

- In general for this assignment (and good coding practices), consider breaking each sorting algorithm into logical pieces which will form helper methods. Try to avoid one long method.
- Similarly to Programming Assignment 1, consider familiarizing yourself with `List` (`https://docs.oracle.com/javase/8/docs/api/java/util/List.html`) and `ArrayList` (`https://www.google.com/search?client=opera-gx&q=arraylist+java&sourceid=opera&ie=UTF-8&oe=UTF-8`)
- If you would like resources on Generics in Java, see `https://www.baeldung.com/java-generics`, `https://docs.oracle.com/javase/tutorial/java/generics/types.html` or `https://www.geeksforgeeks.org/generics-in-java/`

3 Submission

You must submit all your work in Java. The only file you **must** work in is `GraphAlgorithms.java`. **Do not change the name or any of the provided code in this file.** Simply add your solutions to the existing code. You may modify the other files on your own time, but your submission will be marked using the original state of these files.

You are given five files: `P2Tester.java`, `GraphAlgorithms.java`, `PixelWriter.java`, `PixelGraph.java`, and `PixelVertex.java`.

When submitting your assignment, submit only `GraphAlgorithms.java`.

4 Evaluation

The programming assignment will be marked out of 25, based on a combination of automated testing and human inspection.

Part	Max points
MergeSort	5
QuickSort	5
InsertionSort	5
RadixSort	10

The following score ranges will apply to this assignment:

Score	Description
0-5	Submission does not compile
5-15	Compiles but mostly incorrect results, inefficient, missing methods, or poorly written code
15-20	Mostly correct results, mostly follows run-times listed, code of dubious quality
20-25	All methods instantiated with correct results, follows run-times listed, well-written code