CS102 – Algorithms and Programming II
Programming Assignment 6
Spring 2023

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:

  CS102_Sec1_Asgn6_YourSurname_YourName.zip
- Replace the variables “Sec1”, “YourSurname” and “YourName” with your actual section, surname and name.
- You may ask questions on Moodle and during your section’s lab.
- Upload the above zip file to Moodle by the deadline (if not significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TAs and tutors during your section’s lab.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Sorting Pixels

You are going to implement a Java program for sorting the pixels of an image based on brightness value. You are going to implement a **2D Bubble Sort** algorithm. We would like to visualize the sorting at each turn of the algorithm. To this end, you need to implement methods for loading and displaying images on the screen. You may use **BufferedImage** and **ImageIO** Java classes to accomplish this task.

In an **ImageSorter** class implement a **void loadImage(string fileName)** method for importing the image file with the given file name. This class should store the loaded image as a **BufferedImage**. Include a **displayImage** method for displaying the stored **BufferedImage** on screen. You are free to arrange your frame and panel in the way you desire. For example, ImageSorter could extend **JFrame** or **JPanel**, which would influence how **displayImage** method functions.

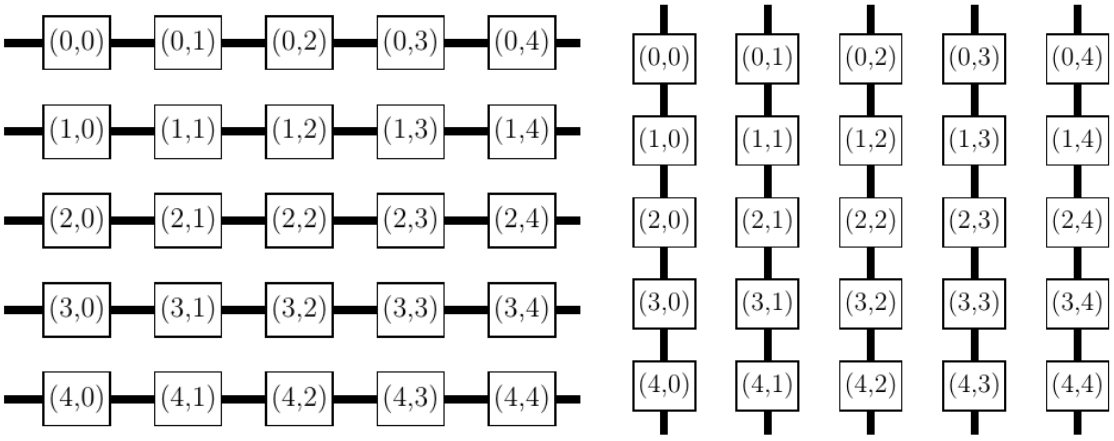
Sorting requires comparing, yet comparing two vectors (entities with more than one variable) is not trivial. For example, the color of a pixel is represented by 3 float variables in the [0,1] range: values for Red (R), Green (G), and Blue (B). To compare two colors, we are going to use their brightness. You can calculate the brightness of color using the following luminance formula:

$$\text{Brightness}(R,G,B) = (0.2126 * R + 0.7152 * G + 0.0722 * B)$$

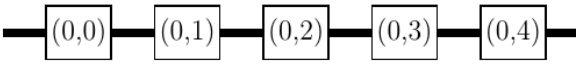
Then you will be able to compare two different colors and hence sort them.

The sorting technique you are going to use is an adaptation of Bubble Sort to 2D matrices. An image can be considered a 2D color matrix. The basis of Bubble Sort is to swap any adjacent unordered pairs until all are in order. Since we are going to visualize the sort as it takes place, you need to implement a step of the sort to be called by a timer repeatedly.

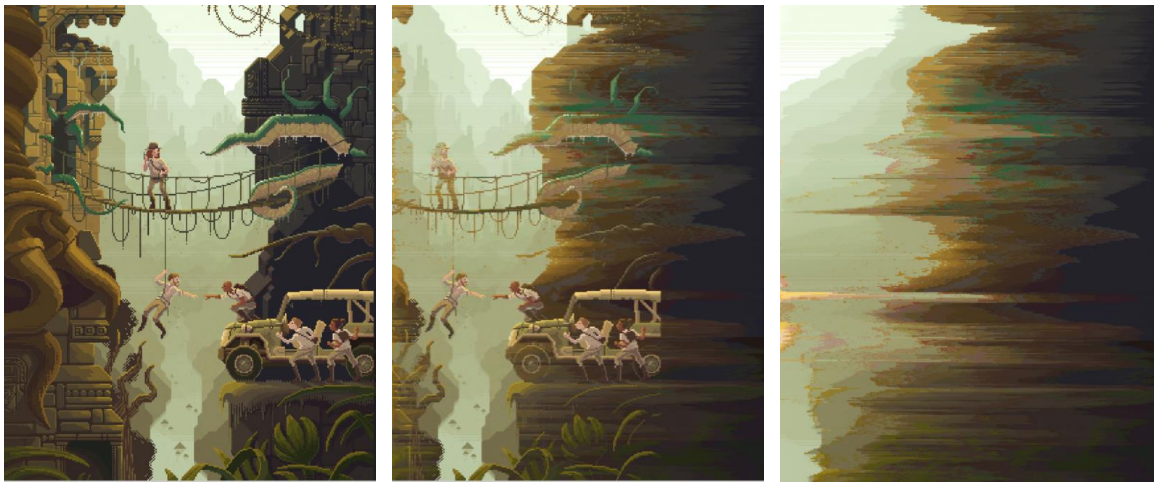
A 2D matrix can be thought as a bunch of horizontal or vertical arrays:



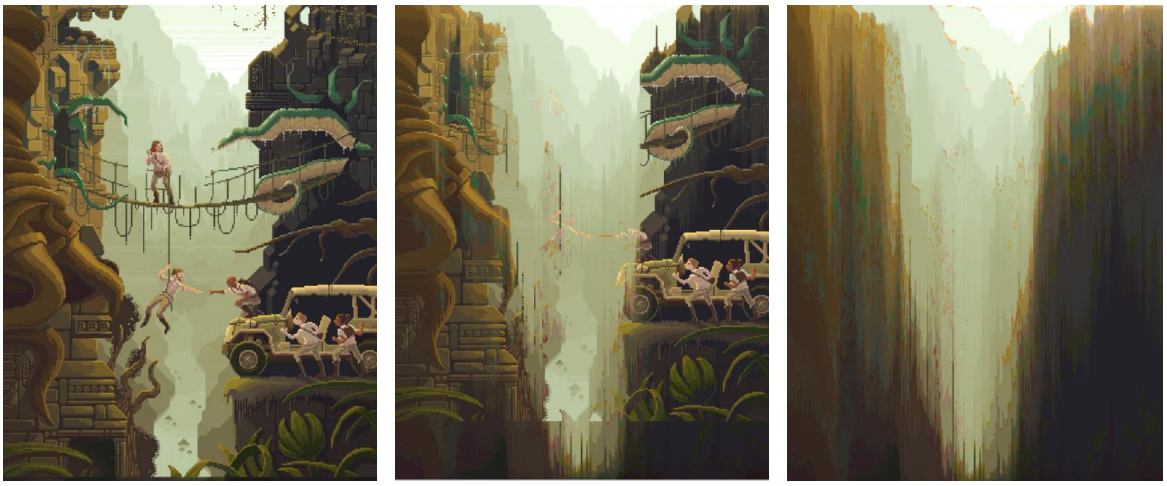
Implement a void **horizontalStep()** method that performs one pass of Bubble Sort in the horizontal direction, considering the image as a bunch of horizontal arrays. For example, this should make one pass on the following first line, followed by all the other lines.



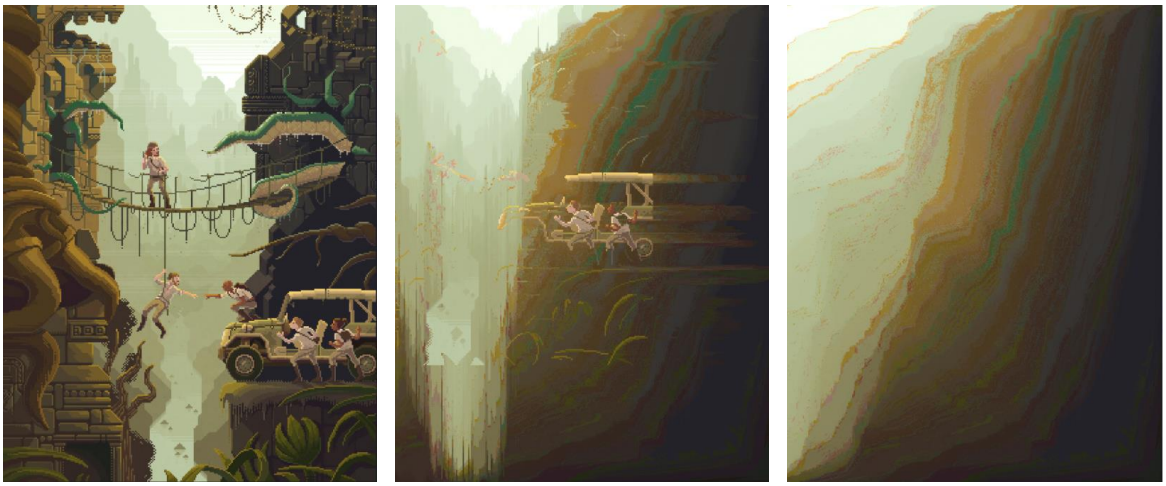
This method will not complete the sort, for an image of width N, this method should be called N times at worst case to complete the sort. For example, repeatedly calling the **horizontalStep()** method would sort each horizontal line of the image based on color's brightness value, resulting in the following images. (You will call this **horizontalStep** method with a timer so that we can see the intermediate steps.)



Implement a void **verticalStep()** method that performs one pass of Bubble Sort in the vertical direction, considering the image as a bunch of vertical arrays. This time, repeated calls of **verticalStep()** would result in each column of the image being sorted based on brightness.



Implement a void **diagonalStep()** that calls **horizontalStep**, **verticalStep**, and **displayImage** methods in it. Include a void **startAnimatedBubbleSort()** method that starts a timer which calls **diagonalStep** to visualize the steps of 2D Bubble Sort. The brightest pixel should end up in the top left corner and the darkest pixel should end up in the bottom right corner of the image.



Make sure your algorithm works with different sizes of images. You can assume that the maximum image size is your screen resolution, thus you do not need to consider zooming out the image to show it completely. Once the sorting is complete (no pixel pair needs swapping) you should stop the timer. Implement keyboard controls (*java.awt.event.KeyListener*) to adjust the speed of animation (timer's delay) and to restart the sorting (reloading the original image). You may use any image to test your code.

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- Horizontal step method should be complete, we should be able to see its effects on the image that you show.

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.