AP Computer Science Name ____

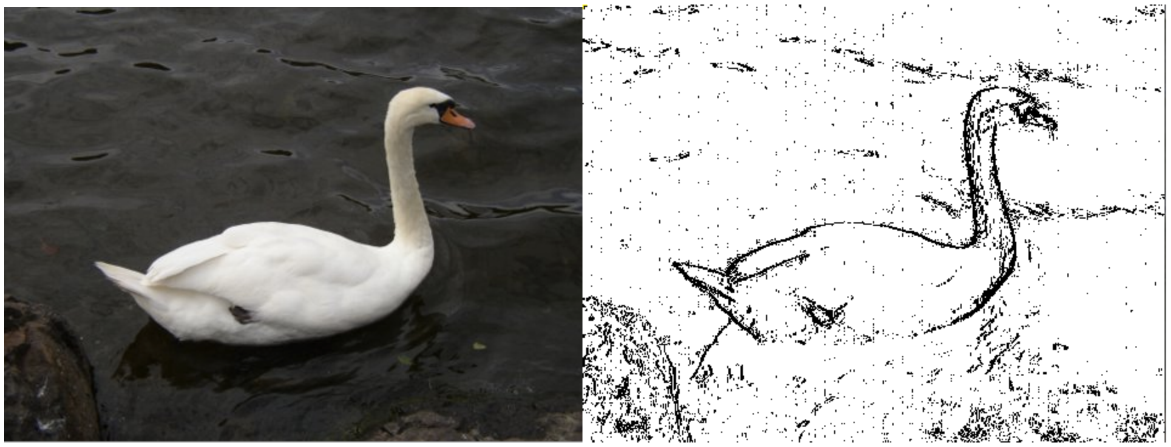
Picture Lab 6: Edge Detection Date ______________________________ Period __________

Objective: To work on methods to do simple edge detection.

Background:
Detecting edges is a common image processing problem. For example, digital cameras often feature face detection. Some robotic competitions require the robots to find a ball using a digital camera, so the robot needs to be able to "see" a ball.

One way to look for an edge in a picture is to compare the color at the current pixel with the pixel in the next column to the right. If the colors differ by more than some specified amount, this indicates that an edge has been detected and the current pixel color should be set to black. Otherwise, the current pixel is not part of an edge and its color should be set to white (see the figures below). How do you calculate the difference between two colors? The formula for the difference between two points (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Likewise, the difference between two colors $(red_1, green_1, blue_1)$ and $(red_2, green_2, blue_2)$ is $\sqrt{(red_2 - red_1)^2 + (green_2 - green_1)^2 + (blue_2 - blue_1)^2}$. The **colorDistance** method in the **Pixel** class uses this calculation to return the difference between the current pixel color and a passed color.

Below is a swan picture and its edge detected version to the right. The method is **edgeDetection** in **Picture.java**. The edge detection algorithm measures the color distance between each pixel and its immediate right (next column) neighbor to determine if the pixel is at an edge. If the color distance between the two pixels is greater than a set threshold value, then an edge is detected and the resulting picture's pixel is turned black. Otherwise, the resulting pixel is turned white denoting no edge. For the picture below, each pixel measured the color distance to its right pixel neighbor (next column over).

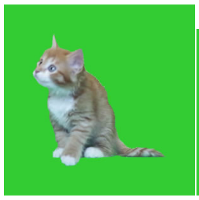
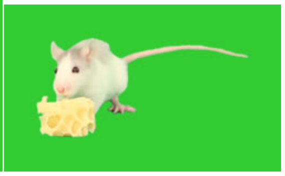


Activity:
A1) Write an edge detection method that detects edges by comparing each pixel with the pixel below.

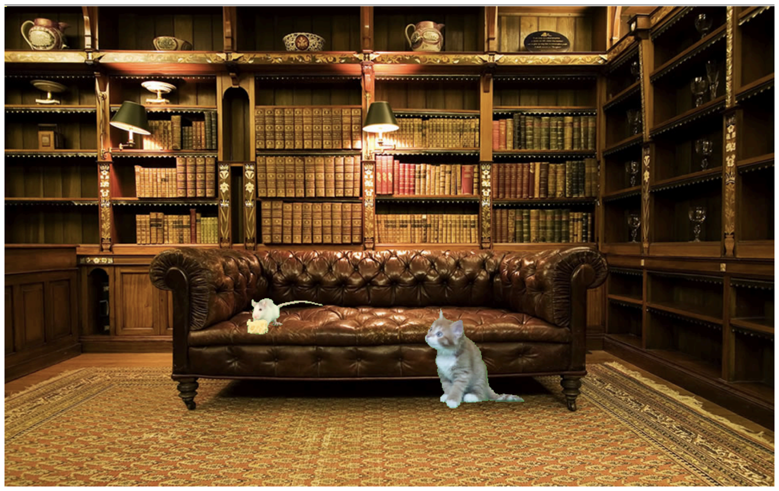
Create the new method `Picture edgeDetectionBelow(int threshold)` inside **Picture.java** that uses <u>the pixel below (next row)</u> to calculate the color distance and returns a black-and-white edge picture.

Green Screen

Another popular effect is the “green screen”. On television, they make a person or object “appear” somewhere else. This is done by superimposing a picture of the subject in front of a picture of a location. For example, below are two photos of a cat and mouse each with a green background. The third photo is a couch in a library.

By scaling and superimposing the cat and mouse onto the library background, the two animals appear to be in the same room together.



Activity:

A2) Write a method to take three images, two with green screen and the third a background, and return an image of two superimposed on the third that looks natural and in scale. Points will be deducted if the images are out of scale or look unnatural. Use **PictureExplorer** to find the color of the green screen and use that color in your algorithm. You may use your own images or use the ones provided.

Create the new method `Picture greenScreen()` inside **Picture.java** that returns a green screen result.

Challenge: (not graded)

A3) Rotate a picture any angle θ, center the picture, and fix any pixel “drop out”. The transformation formulas for calculating a rotated pixel are the following:

$$x_1 = x_0 \cos\theta - y_0 \sin\theta$$
$$y_1 = x_0 \sin\theta + y_0 \cos\theta$$

where (x_0, y_0) is the original location and (x_1, y_1) is the new location. Remember, columns represent the x-direction and rows represent the y-direction.