ASSIGNMENT NO:3 **DATE: / /2018**

PROGRAM TITLE:Perform Histogram Equalization on an image.

PROGRAM CODE:

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.imageio.ImageIO;


public class HistogramEQ {

    private static BufferedImage original, equalized;

    public static void main(String[] args) throws IOException {

        File original_f = new File("./img/test.jpg");
        String output_f = "hist-o";
        original = ImageIO.read(original_f);
        equalized = histogramEqualization(original);
        writeImage(output_f);

    }

    private static void writeImage(String output) throws
IOException {
        File file = new File(output+".jpg");
        ImageIO.write(equalized, "jpg", file);
    }

    private static BufferedImage
histogramEqualization(BufferedImage original) {

        int red;
        int green;
        int blue;
        int alpha;
        int newPixel = 0;

        // Get the Lookup table for histogram equalization
        ArrayList<int[]> histLUT =
histogramEqualizationLUT(original);

        BufferedImage histogramEQ = new
BufferedImage(original.getWidth(), original.getHeight(),
original.getType());

        for(int i=0; i<original.getWidth(); i++) {
            for(int j=0; j<original.getHeight(); j++) {

                // Get pixels by R, G, B
                alpha = new Color(original.getRGB (i,
j)).getAlpha();
                red = new Color(original.getRGB (i, j)).getRed();
```

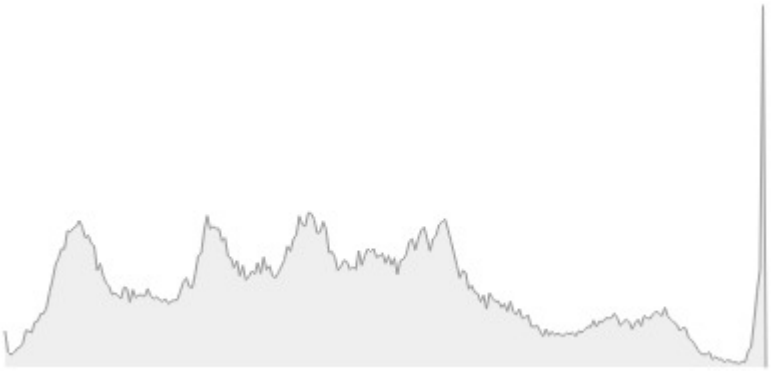
```
                green = new Color(original.getRGB (i,
j)).getGreen();
                blue = new Color(original.getRGB (i, j)).getBlue();

                // Set new pixel values using the histogram lookup
table
                red = histLUT.get(0)[red];
                green = histLUT.get(1)[green];
                blue = histLUT.get(2)[blue];

                // Return back to original format
                newPixel = colorToRGB(alpha, red, green, blue);

                // Write pixels into image
                histogramEQ.setRGB(i, j, newPixel);

            }
        }

        return histogramEQ;

    }

    // Get the histogram equalization lookup table for separate R,
G, B channels
    private static ArrayList<int[]>
histogramEqualizationLUT(BufferedImage input) {

        // Get an image histogram - calculated values by R, G, B
channels
        ArrayList<int[]> imageHist = imageHistogram(input);

        // Create the lookup table
        ArrayList<int[]> imageLUT = new ArrayList<int[]>();

        // Fill the lookup table
        int[] rhistogram = new int[256];
        int[] ghistogram = new int[256];
        int[] bhistogram = new int[256];

        for(int i=0; i<rhistogram.length; i++) rhistogram[i] = 0;
        for(int i=0; i<ghistogram.length; i++) ghistogram[i] = 0;
        for(int i=0; i<bhistogram.length; i++) bhistogram[i] = 0;

        long sumr = 0;
        long sumg = 0;
        long sumb = 0;

        // Calculate the scale factor
        float scale_factor = (float) (255.0 / (input.getWidth() *
input.getHeight()));

        for(int i=0; i<rhistogram.length; i++) {
            sumr += imageHist.get(0)[i];
            int valr = (int) (sumr * scale_factor);
            if(valr > 255) {
                rhistogram[i] = 255;
            }
```

```
            else rhistogram[i] = valr;

            sumg += imageHist.get(1)[i];
            int valg = (int) (sumg * scale_factor);
            if(valg > 255) {
                ghistogram[i] = 255;
            }
            else ghistogram[i] = valg;

            sumb += imageHist.get(2)[i];
            int valb = (int) (sumb * scale_factor);
            if(valb > 255) {
                bhistogram[i] = 255;
            }
            else bhistogram[i] = valb;
        }

        imageLUT.add(rhistogram);
        imageLUT.add(ghistogram);
        imageLUT.add(bhistogram);

        return imageLUT;

    }

    // Return an ArrayList containing histogram values for separate
R, G, B channels
    public static ArrayList<int[]> imageHistogram(BufferedImage
input) {

        int[] rhistogram = new int[256];
        int[] ghistogram = new int[256];
        int[] bhistogram = new int[256];

        for(int i=0; i<rhistogram.length; i++) rhistogram[i] = 0;
        for(int i=0; i<ghistogram.length; i++) ghistogram[i] = 0;
        for(int i=0; i<bhistogram.length; i++) bhistogram[i] = 0;

        for(int i=0; i<input.getWidth(); i++) {
            for(int j=0; j<input.getHeight(); j++) {

                int red = new Color(input.getRGB (i, j)).getRed();
                int green = new Color(input.getRGB (i,
j)).getGreen();
                int blue = new Color(input.getRGB (i,
j)).getBlue();

                // Increase the values of colors
                rhistogram[red]++; ghistogram[green]++;
bhistogram[blue]++;

            }
        }

        ArrayList<int[]> hist = new ArrayList<int[]>();
        hist.add(rhistogram);
        hist.add(ghistogram);
        hist.add(bhistogram);
```

```
        return hist;

    }

    // Convert R, G, B, Alpha to standard 8 bit
    private static int colorToRGB(int alpha, int red, int green,
int blue) {

        int newPixel = 0;
        newPixel += alpha; newPixel = newPixel << 8;
        newPixel += red; newPixel = newPixel << 8;
        newPixel += green; newPixel = newPixel << 8;
        newPixel += blue;

        return newPixel;

    }

}
```

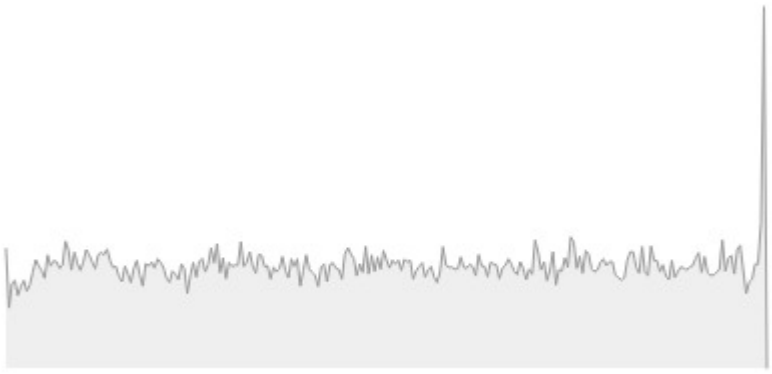
OUTPUT:

L: Original Image



R: Histogram of the Original Image

L: Equalized Image



R: Histogram of the Equalized Image