COMP 221 Fall 2024 Homework 1

Due: Friday, September 20, by 11:59 pm

1 Guidelines

Please **type up** and submit a PDF of your solutions. I recommend LaTex (Overleaf is really nice to use online), but you are certainly welcome to use Google Docs or some other word processing program. Just make sure to save it as a PDF before submission.

Programming language: You are welcome to use Python or Java.

Do your own work: Homework should be individual work. You may discuss problems with other people, but you must write up your solutions yourself. I will not say that the web is off limits, but handing in solutions you find online as if they are your own is also not acceptable.

Make sure you have your name on your homework!

- **Question 1**

  From Skiena, Question 2-1. What value is returned by the following function? Express your answer as a function of n. Write the order of growth in terms of big-Θ. Hint: be very precise with summation variables.

Algorithm 1: function mystery(n)

```
let $r = 0$
for $i = 1 : n - 1$ do
  for $j = i + 1 : n$ do
    for $k = 1 : n$ do
      r = r + 1
    end for
  end for
end for
return r
```

- **Question 2**

  From Skiena, Question 2-2. What value is returned by the following function? Express your answer as a function of n. Write the order of growth in terms of big-Θ.

- **Question 3 – Programming**

  This question requires you to use a graphics library for either Java or Python. For Python, I strongly recommend using `imageTools.py` that we used in COMP 123 (file provided with this homework). For Java, `BufferedImage` is what I used (make sure to include the library in your program).

  Convolution is an image analysis process by which we create a new image from an original one (see: https://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution). A pixel in the new image located at (x, y) will have its value determined by the values of the pixel located at (x, y) in the original image, AND its neighbors. These values are all scaled by a kernel, a 3x3 matrix that applies weights to all the original pixel values (some kernels may be larger than 3x3).

Algorithm 2: function pesky(n)

```
let $r = 0$
for $i = 1 : n$ do
    for $j = 1 : i$ do
        for $k = j : i + j$ do
            r = r + 1
        end for
    end for
end for
return r
```

When looking at a picture, how do we determine what is the boundary, or edge, of an object? Edges in an image are usually determined by sharp gradients - dramatic changes in color or light. Edge detection is a specialized area of computer vision where we use an algorithm to find the edges (those dramatic changes in color or light) in an image. Sobel came up with a pair of 3x3 kernel filters that can help us find edges. One kernel is for the horizontal component, and the other is for the vertical.

$$\begin{array}{cc} \text{x-filter} & \text{y-filter} \\ \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} & \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \end{array}$$

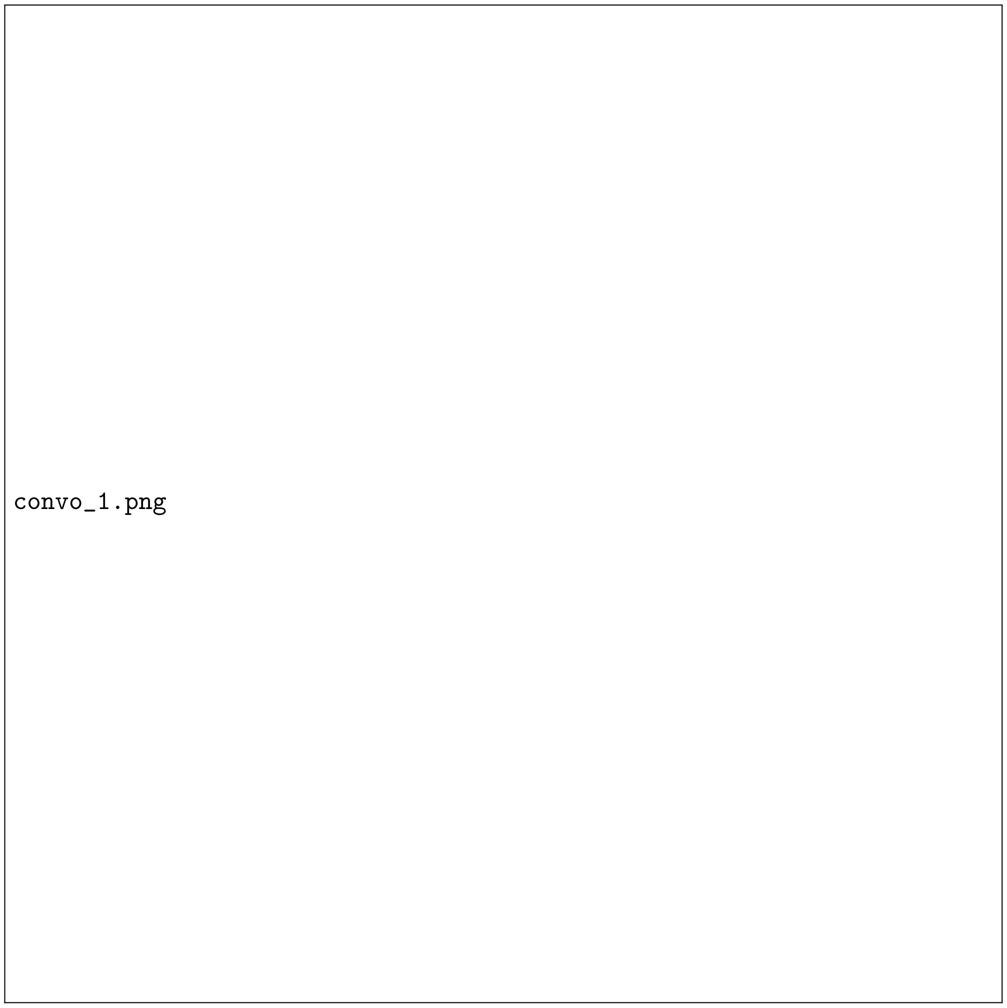
To calculate the convolution, we have to do something like this:

- Suppose we have an original image, named `origImage`. At pixel location (x, y) in `origImage`, let's say the RGB values are `(r5, g5, b5)`.
- We want to use this information to calculate the RGB values of the image containing the edges of `origImage` (let's call it `edgeImg`)
- To do that, we need to look at the 3x3 group of pixels around and including (x, y), and multiply each of those RGB values with the respective kernel value

  For example, for the x-filter:

  - -1 (at $(0, 0)$ of the x-filter kernel) $*(r1, b1, g1)$ at pixel location $(x - 1, y - 1)$. We need to do it 3 times - one each for the red, green, and blue channels
  - $0 * (r2, b2, g2)$ at pixel location $(x, y - 1)$
  - $1 * (r3, b3, g3)$ at pixel location $(x + 1, y - 1)$
  - $-2 * (r4, b4, g4)$ at pixel location $(x - 1, y)$
  - $0 * (r5, b5, g5)$ at pixel location (x, y)
  - $2 * (r6, b6, g6)$ at pixel location $(x + 1, y)$
  - $-1 * (r7, b7, g7)$ at pixel location $(x - 1, y + 1)$
  - $0 * (r8, b8, g8)$ at pixel location $(x, y + 1)$
  - $1 * (r9, b9, g9)$ at pixel location $(x + 1, y + 1)$

  This image from ResearchGate helps illustrate my point above (slightly different variables).



Your job for this homework assignment is to implement the Sobel filter for an image of your choosing. Submit your code, your original image, and your "edge" image for grading and credit. You will not be graded on your code, unless for partial credit. The edge image should be in grayscale (sometimes called black and white, but strictly speaking, not the same), which means the red, green, and blue channels have the same value.

You will need to apply the x- and y- filters to each of the red, green, and blue channel values of a pixel, and then sum those values (giving you `sumXRed`, `sumXGreen`, and `sumXBlue`, and the same for `sumY` values). Then, average *those* three values together (giving you `Gx` or `sumX` and `Gy` or `sumY` values). Then, calculate G using $G = \sqrt{G_x^2 + G_y^2}$. Finally, set the color of the pixel located at (x, y) in the `edgeImg` image to be (G, G, G).

For **Python** and `imageTools`, useful methods would be:

- `pic = Picture("picturename.jpg")` to create a picture object from a picture on your computer
- `width = pic.getWidth()` to get the width, in pixels, of a picture
- `height = pic.getHeight()` to get the height, in pixels, of a picture
- `newPic = Picture(width, height)` to create a new picture object with the specified width and height

- `(r, g, b) = pic.getColor(x, y)` to get the red, green, and blue channel information for the pixel at (x, y) in picture
- `newPic.setColor(x, y, (r, g, b))` to set the color at pixel (x, y) in `newPic` to the specified r, g, b values
- `pic.show()` to see the picture. I recommend you put an `input("Press any key to continue...")` after the `pic.show()` command so it keeps the picture up until you are done

---

For **Java** and `BufferedImage` (note: you are welcome to use a different library, this was the first one I found that was semi-bearable, and it is what is used to support Kilt-graphics from 127/128)

- To create a new `BufferedImage` object, and to read in a picture file:

```
BufferedImage img = null, modImg = null;
try {
    img = ImageIO.read(new File("picturename.jpg"));
} catch (IOException e) {
    System.out.println("Read in image error...");
}
```

- To save the newly created file on your computer (I haven't found an easy way to just have the window pop up yet...):

```
try {
    File outputfile = new File("saved1.png");
    ImageIO.write(modImg, "png", outputfile);
} catch (IOException e) {
    // handle exception
    System.out.println("Exception in saving...");
}
```

- To create a new `BufferedImage` object of specified height and width:

```
BufferedImage edges = new BufferedImage(pic.getWidth(), pic.getHeight(),
BufferedImage.TYPE_INT_RGB);
```

- To extract red, green, and blue values from pic at (x, y):

```
mycolor = new Color(pic.getRGB(x, y));
r5 = mycolor.getRed();
g5 = mycolor.getBlue();
b5 = mycolor.getGreen();
```

- To set the r, g, b values in `edgeImg` to be (g, g, g):

```
rgb = ((g&0x0ff)<<16)|((g&0x0ff)<<8)|(g&0x0ff);
edges.setRGB(x, y, rgb);
```