Before running any commands, run: npm install to install all needed components

Part 1)

Part A)

To run Part 1A: node part1a.js

Input: 810147810.jpg

Output: newImage.jpeg

The processes of greyscaling works by averaging out the RGB values of a pixel and assigning this new average value to that pixel in each of its R G and B slots (Cook).

Part B)

To run Part1B: node part1b.js

Input: 810147810.jpg

Output (blurred): blurredImage.jpg

Output (sharpened): sharpImage.jpeg

Here we used a kernel to convolute each pixel of the matrix. Convolution is the process of multiplying a pixel and its neighbors by a matrix and then adding all these values to a pixel. The specific matrix used is called a kernel. The kernel used to blur was:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

While the kernel used to sharpen was:

|  |  |  |
| --- | --- | --- |
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

(Docks)

Part C)

To run Part1C: node part1c.js

Input: bact2.png

Output (separated image): sepImg2.jpeg

Output(colored image): coloredSep.jpeg

To tackle this problem, I took a rather simple solution. I selected the pixel in the very far corner and set it as my default background pixel. From here I compared every pixel’s hex value to my default pixel’s hex value. If the difference between this pixel and my default background pixel was less than a set tolerance constant (which was derived through trial and error) I colored the pixel black, otherwise I colored it white. After this, I went through the now black and white image, coloring each white pixel a random color. Finally I went through the image one more time, any time I encountered a pixel that was not black, I found its color and colored all non black pixels adjacent to it this color, thus spreading the color over the object. As my function would hit the next pixel, it would detect the newly colored pixel and repeat the process over and over.

Assuming an nXn pixel picture, the looping though of each pixel in the photo takes O(n^2) time. Inside this loop, checking each pixel against the set background pixel is O(1), coloring each black or white pixel is O(1). Looping through each pixel again is O(n^2). Comparing each pixel to check if it is black is O(1). Coloring the pixel a random color is O(1). Assigning the random color is O(1). Finally looping through the pixels of the new photo is O(n^2), checking each pixels color and spreading it to all non black pixels is O(1). This leads to a O(3n^2) or O(n^2).

Part 2:

To run Part2: node part2.js

The algorithm I used to solve this problems is Kahn’s algorithm for Topological Sorting. The algorithm works by finding the nDegree for each node in a tree. The nDegree is the amount of incoming edges to a node. First the algorithm queues up all nodes with an nDegree of 0, from here it loops until the queue is empty. It dequeues the first member, and decrements the nDegree of any child nodes of the member. Then it checks to see if any new nodes should be added to the queue. (GeeksforGeeks). In order to calculate the nDegree, you must loop through all the edges to find how many nodes feed into a node. I handle this with the actual assignment of the variable but it is still a O(E) operation. After this we must queue and loop through all of the vertices but only once, which is a O(V) operation. This leads to a runtime of O(V + E).

Citations:

Cook, J. (2018). *Converting color to grayscale*. [online] Johndcook.com. Available at: https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/ [Accessed 1 May 2018].

Docs.gimp.org. (2018). *8.2. Convolution Matrix*. [online] Available at: https://docs.gimp.org/en/plug-in-convmatrix.html [Accessed 1 May 2018].

GeeksforGeeks. (2018). *Kahn's algorithm for Topological Sorting - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/ [Accessed 1 May 2018].