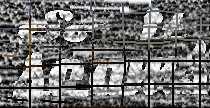
**Carved Lab**

**Secret Message in Image**

**“So it seams”**

****

**Algorithm Description**

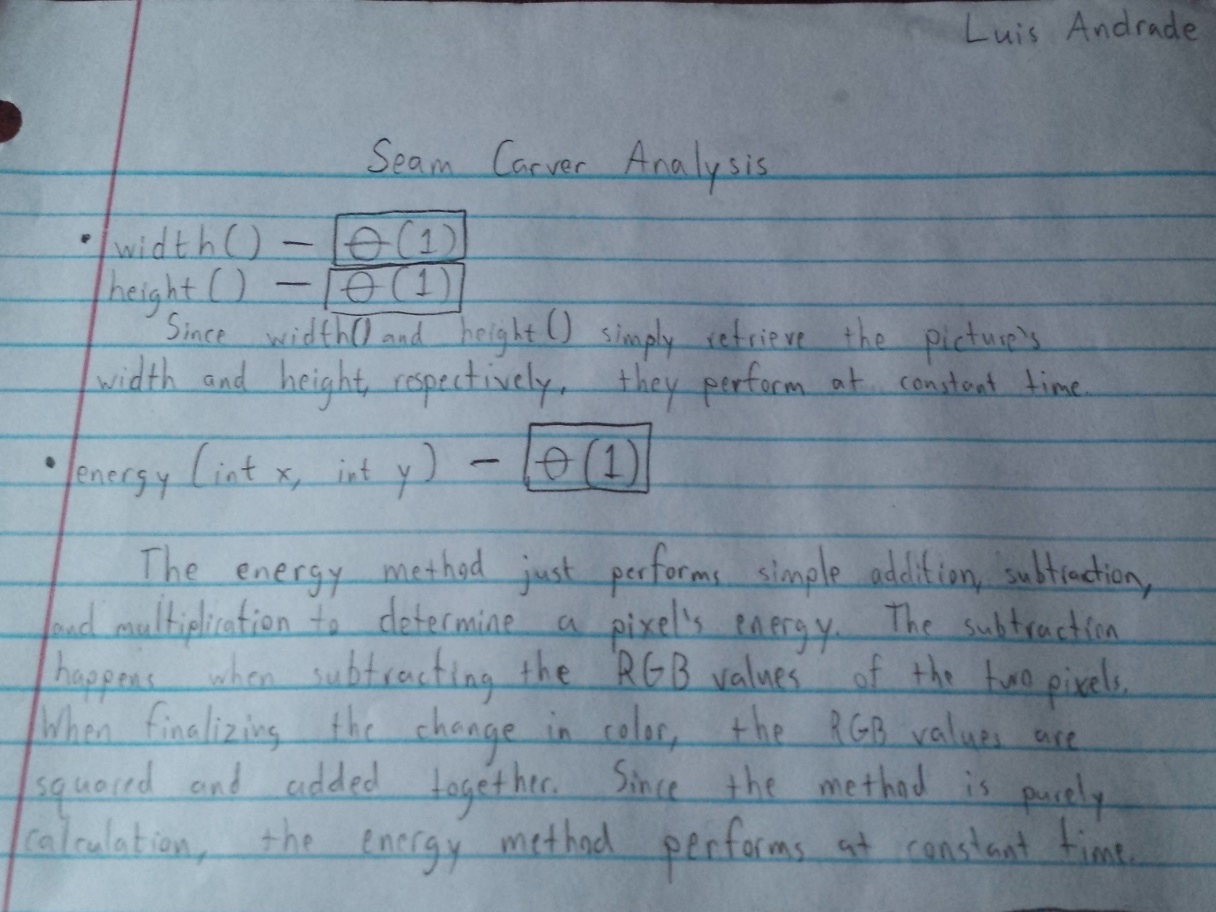
The algorithm begins by taking in a new Picture object through the constructor, which is stored in a global variable so that the entire SeamCarver class can access it. The width, height, and energy methods perform at constant time. The width method simply returns the width of the picture in integers, and the height method returns the height of the picture. The find methods for horizontal and vertical seams give each pixel an energy score passed on the change in color from one pixel to the next. Then, the path of lowest energy cost is determined and stored into an array that holds the indices of the pixels to be removed. This array is used within the remove methods of each seam type to determine which set of pixels to remove from the picture and resize the final image.

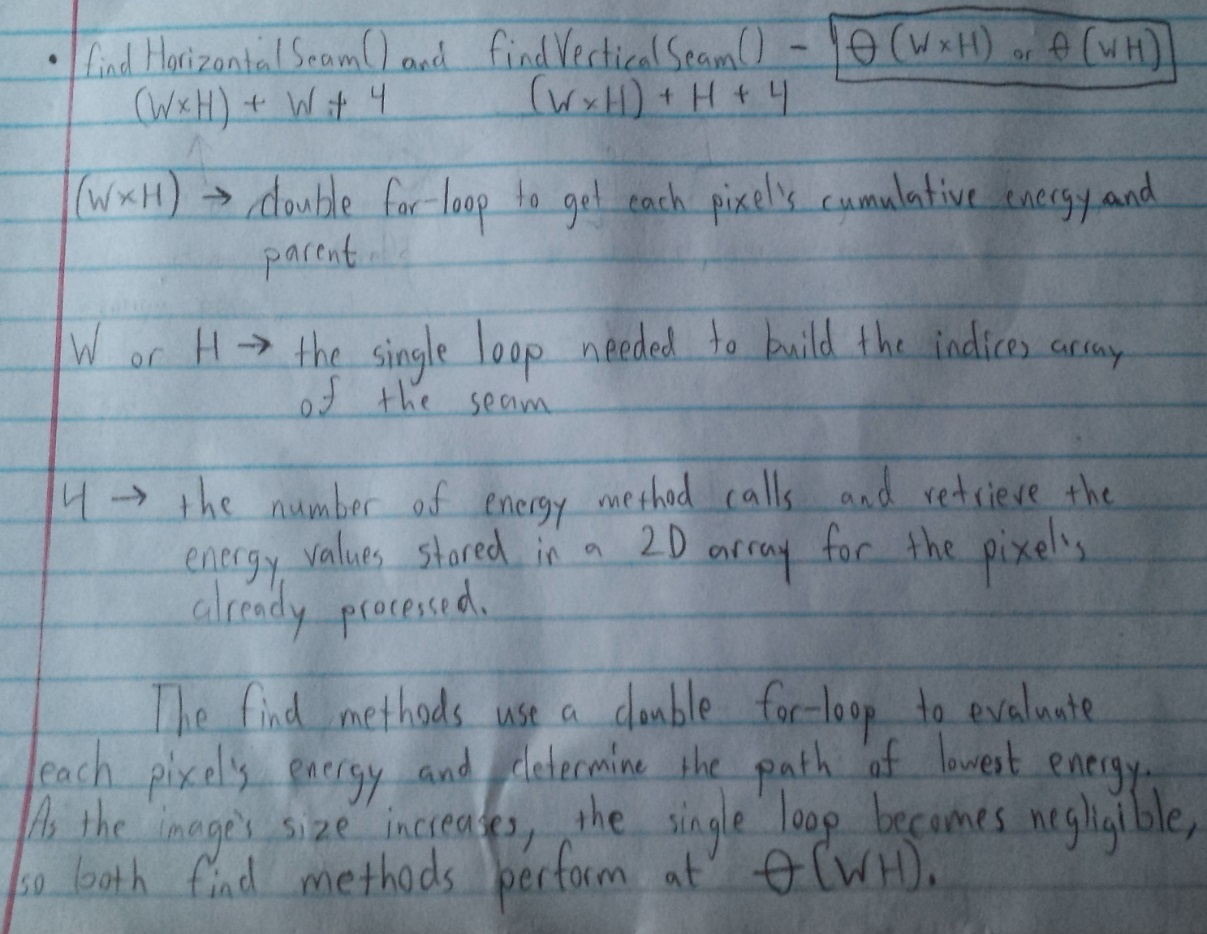
The energy method works by first checking if the x and y coordinates of a pixel are within the image’s range. If not, an IndexOutOfBoundsException is thrown. If they are, then the change in color is determined from one pixel to the next. The RGB values of the right and left pixels from the current pixel are subtracted, squared, and then added together to determine the change in color in the x-direction. This is repeated for the top and bottom pixels of the current pixel to determine the change in color in the y-direction. These squared values are then added together to get the overall energy score of a pixel.

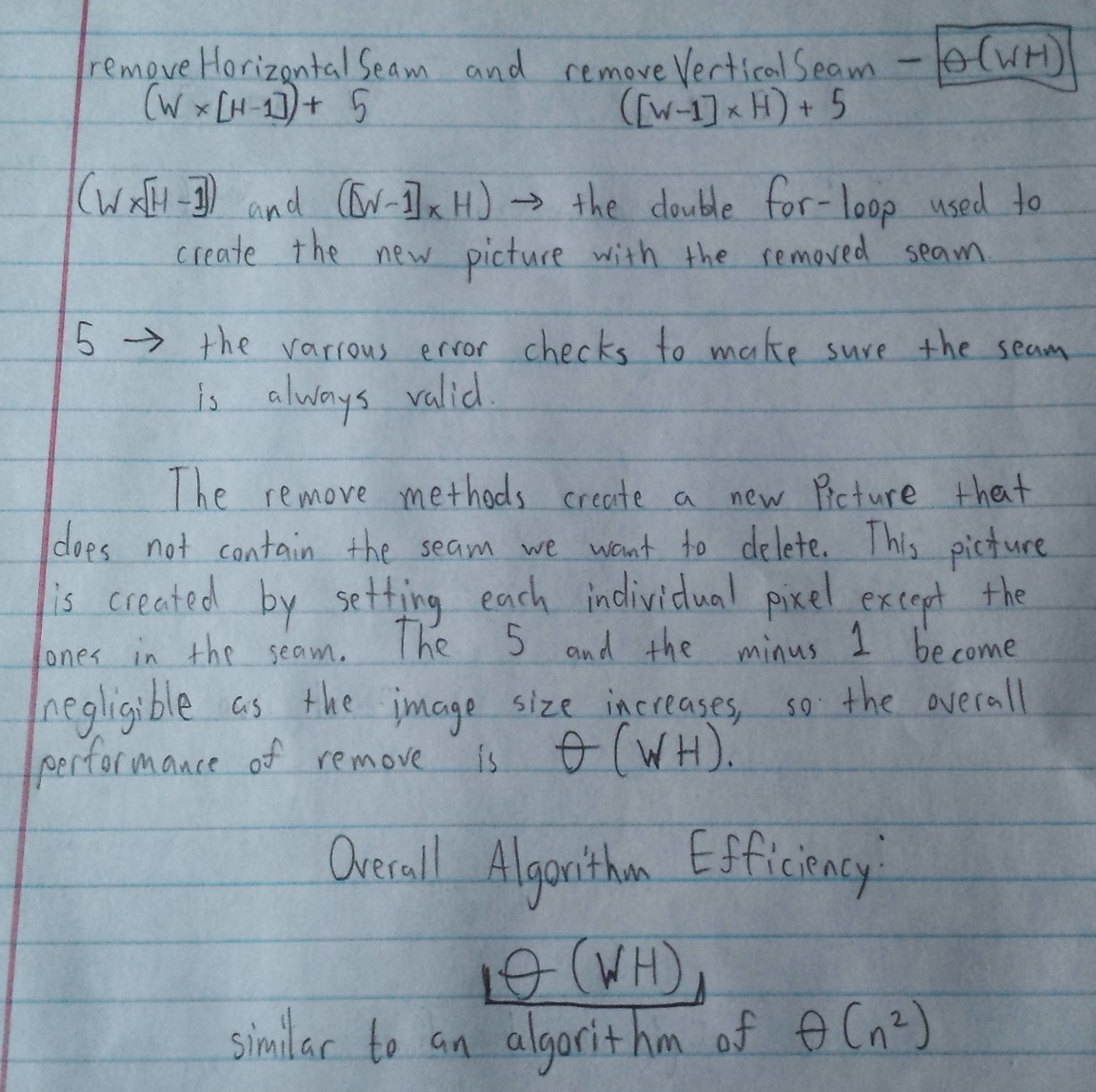
The findHorizontalSeam creates a seam from left to right by determining the energy cost of each possible path and finding the one with least energy. The method first creates two temporary 2D arrays to represent vertices in a graph mapped to the pixels of the picture. The first 2D array holds the cumulative energy weights of each vertex in order to determine the path of minimum energy. The second 2D array holds the indices of a pixels parent. Since the pixels are implicitly connected by a directed acyclic graph, the index at each vertex represents which of the three pixels connected to the pixel at the vertex has the lowest energy. With the 2D arrays created, the method starts filling in both arrays by going through each pixel in the image. For each pixel, the energy of the pixel is calculated. We then look at the parent pixels in the previous column that are connected to that pixel and determine which of those three pixels has the lowest energy. Once determined, that pixel’s energy and the lowest parent’s energy are added together and then stored at the pixel’s vertex. The parent pixel’s index is stored at the same vertex, but in the second 2D array. Once the algorithm reaches the image’s last column, the algorithm begins checking and storing the index of the vertex with the lowest cumulative energy. Once all of that is done and the lowest energy path is determined, the path is built by going backwards using the second 2D array or parent pointers and storing the indices gathered previously, starting at the vertex that held the lowest cumulative energy. This path ends up being an array of indices, where each index in the array represents the X-coordinate of a pixel to be removed and the value within the index represents the Y-coordinate of the pixel to be removed. The findVerticalSeam method works similarly to the findHorizontalSeam method, but instead creates a seam from top to bottom and returns an array of indices where each index in the array represents the Y-coordinate of the pixel in the image to be removed, and the value in the index represents the X-coordinate of the pixel to be removed.

The removeHorizontalSeam takes in the indices array that was created in the find method and shrinks the current image down from top to bottom by one row of pixels. First, error checking is done to see if the array’s length is equal to the image’s width, and that the image’s width is not 1 or less. If these pass, the algorithm starts removing pixels by creating a new picture object whose height is one less than the original picture. For each x-coordinate in the image, the y-coordinate of the pixel to remove is retrieved from the indices array. If the coordinate is within the image’s range and the y-coordinate is not off by more than one from the previous pixel, then the algorithm loops through each y-coordinate and sets the color of each pixel in the new image to the color of the previous image. However, when the y-coordinate equals to the y-coordinate of the pixel to remove, the algorithm skips that pixel and gets all the other pixels after it. Basically, this method replicates vertically shifting all of the pixels below the pixel to be removed up by one. The new image created represents this shift and the original picture is set to this new picture object with the seam of pixels removed. The removeVerticalSeam works in a similar. However, error checking involves the height of the image and the method shrinks the image down from left to right by one row of pixels, effectively shifting all of the pixels past the removed pixels over to the left by one.

**Algorithm Analysis**







**Help Received**

* Most of the help I received involved the find methods, where I was trying to understand how to traverse the pixels in an image in order to find the path of lowest energy. Most of this help involved understanding how the pixels were connected to each other in a manner similar to a directed acyclic graph, and how to begin building the seam after the lowest energy path was determined. This helped me get on the right train of thought and to come to the necessary conclusions myself to understand how seam carving worked.
* I also read a Wikipedia article on seam carving to better understand how the find methods functioned and how to build the possible paths using a dynamic programming solution.