**Algorithm Description:**

Instantiating the SeamCarver class defines a picture stored in the class, and creates a matrix that will be used in the Find Seam functions. (W\*H) – to setup the matrix

getPicture function returns the picture stored in the class. (1)

Width function returns the width of the picture as stored in the picture class. (1)

Height function returns the height of the picture as stored in the picture class. (1)

Energy throws an exception if either the x or y is outside of the boundaries of the picture, otherwise it finds the difference between the top, bottom and left, right pixels, squares the difference, and adds them together. (1) – a lot of math but just constant time math, no loops or recursion

findHorizontalSeam starts left and goes right looking at the entire height of the picture calculating the energies and keeping a least weight path to each vertice visited. It does this by adding the weight of the least previous path it can reach to its own weight as topological sort does. Once it is done we take the least path overall and traverse back through its path to collect all the indices and return them. T(n)=(W\*H) - the topological portion of the algorithm runs across the width and the height. Then later it runs through just the width, so T(n)=W\*H+W which is T(n)=W\*H

findVerticalSeam starts top and goes to the bottom looking at the entire width of the picture calculating the energies and keeping a least weight path to each vertice visited. It does this by adding the weight of the least previous path it can reach to its own weight as topological sort does. Once it is done we take the least path overall and traverse back through its path to collect all the indices and return them. T(n)=(W\*H) - the topological portion of the algorithm runs across the width and the height. Then later it runs through just the height, so T(n)=W\*H+H which is T(n)=W\*H

removeHorizontalSeam creates a temporary picture file to keep track of the new picture. The function starts left and goes right looking at the entire height at each x index adding the pixels of the original image to the temporary one as it goes. Once the function gets to a point in the height that is equal to the indice for the current x it skips that indice for the original image and adds the rest of the pixels from the original image to the temporary image normally. It sets the temporary image to the class image when it is done. T(n)=(W\*H), the algorithm runs through the entire width and height of the picture minus 1 from the height due to the removal of a horizontal seam, so T(n)=W\*(H-1), I am just calling it T(n)=W\*H.

removeVerticalSeam creates a temporary picture file to keep track of the new picture. The function starts from the top and goes to the bottom looking at the entire width at each y index adding the pixels of the original image to the temporary one as it goes. Once the function gets to a point in the width that is equal to the indices for the current y it skips that indice for the original image and adds the rest of the pixels from the original image to the temporary image normally. It sets the temporary image to the class image when it is done. T(n)=(W\*H), the algorithm runs through the entire width and height of the picture minus 1 from the width due to the removal of a vertical seam, so T(n)=(W-1)\*H, I am just calling it T(n)=W\*H.

**Any Help Received:**

Discussed some implementation ideas with David Ericson, Jonathan Pearl, and Kendrick Kiggins to a degree at which I decided what the best implementation option was from the discussion. Example: I talked with Jonathan Pearl on an implementation idea I had to do the Find Seams function to which he added to my idea. (The original idea was to find the least weighted path as I run through the possible paths by going through the entire possible paths one at a time, instead Jon suggested collecting all the paths and then checking the end of all the paths to decide which path to use).