Vincent Lee
I pledge my honor that I have abided by the Stevens Honor System.

Driver.py runs main.py parameterized. Main.py runs snic with the specified image and the compression value as outlined in the paper's algorithm.

Snic.py reuses the center initialization and center highlighting from hw3. The center initialization is the center of 50x50 pixel blocks in the image.

```
def initialCenters(image):
    # centers are 50 apart starting at 25,25
    colors = []
    centers = []
    xsize, ysize, _ = image.shape
    xblocks = xsize // 50
    yblocks = ysize // 50
    for i in range(xblocks):
        for j in range(yblocks):
            x = 25 + i * 50
            y = 25 + j * 50
            centers.append([x,y])
            colors.append(image[x][y])
    return centers, colors
```

```
# to color the centeroids
def colorCenters(image,centers):
    xsize, ysize, _ = image.shape
    for center in centers:
        center[0] = int(center[0])
        center[1] = int(center[1])
        if center[0]!=0 and center[0]!=xsize and center[1]!= 0 and center[1]!=ysize:
            image[center[0]-1:center[0]+2,center[1]-1:center[1]+2] = [1,0,0]
    return image
```

The snic algorithm uses a min heap, places all the initial centroids with distance 0 in the heap and loops until the heap is empty. It utilizes a label map specifying if the element is already in a cluster or not. If it is not it assigns it the cluster from which it has the minimum distance as outlined in the paper using an s value and user specified compression value to calculate the distance. It adds the surrounding 8 pixels in the heap using the directions list which outlines the direction of the surrounding pixels. The distance is from the pixel to centroid.
The only major change I have done from the algorithm outlined in the paper is the order of the values stored in the heap. Python by default uses the first index so I place the distance in the first index of the tuple that I place in the heap.I also keep a count of the elements in the cluster in order to maintain a running average of pixel position and color.

```
def snic(image, compactnessFactor):
    m = compactnessFactor
    heap = []
    centers, colors = initialCenters(image)
    s = (image.shape[0] * image.shape[1] / len(centers)) ** (1/2)
    xsize, ysize, _ = image.shape
    ret = numpy.zeros(image.shape)
    labelMap = numpy.zeros((xsize,ysize))
    # elements in cluster are x, y, R, G, B, count
    clusters = [[0,0,0,0,0,0] for i in range(len(centers))]
    directions =[ [-1, 0], [-1,-1] , [-1,1] , [1,-1], [1, 0] , [1,1], [0,1], [0,-1] ]
    for i in range(len(centers)):
        center = centers[i]
        color = colors[i]
        # order is distance, k, R, G, B
        # sort by distance
        element = (0,i+1,color[0],color[1],color[2],center[0],center[1])
        heapq.heappush(heap,element)
    while len(heap) != 0:
        target = heapq.heappop(heap)
        distance, k, colorR, colorG, colorB, x, y = target
        if labelMap[x][y] == 0:
            labelMap[x][y] = k
            old = updateCluster(clusters, k-1, [x,y,colorR,colorG,colorB])
            for move in directions:
                moveX = x + move[0]
                moveY = y + move[1]
                if moveX >= 0 and moveX < xsize and moveY >= 0 and moveY < ysize:
                    if labelMap[moveX][moveY] == 0:
                        targetColor = image[moveX,moveY]
                        target = [moveX, moveY, targetColor[0], targetColor[1],targetColor[2]]
                        d = elementDistance(target,old,s,m)
                        e = (d, k, targetColor[0], targetColor[1], targetColor[2], moveX, moveY)
                        heapq.heappush(heap,e)
    for i in range(xsize):
        for j in range(ysize):
            clusterIndex = labelMap[i][j]
            target = clusters[int(clusterIndex-1)]
            ret[i][j] = [target[2]/255,target[3]/255,target[4]/255]
    return drawBorders(labelMap,ret)
```

UpdateCenters updates the average pixel index of the cluster and the color by maintaining a count in each cluster.

```
def updateCluster(clusters, index, element):
    cluster = clusters[index]
    # increment count
    cluster[5] += 1
    elements = cluster[5]
    # the index must be an integer
    cluster[0] = ((cluster[0] * (elements-1)) + element[0]) / cluster[5]
    cluster[1] = ((cluster[1] * (elements-1)) + element[1])  / cluster[5]
    for i in range(2,5):
        cluster[i] = ((cluster[i] * (elements-1)) + element[i]) / cluster[5]
```

Draw Border uses the label map to figure out which elements are in the same cluster. It checks if towards the right and below are in the same group else it draws a black pixel.

```
def drawBorders(labelMap, ret):
    xsize, ysize = labelMap.shape
    for i in range(xsize-1):
        for j in range(ysize-1):
            if labelMap[i][j] != labelMap[i+1][j] or labelMap[i][j] != labelMap[i][j+1]:
                ret[i][j] = [0,0,0]
    return ret
```

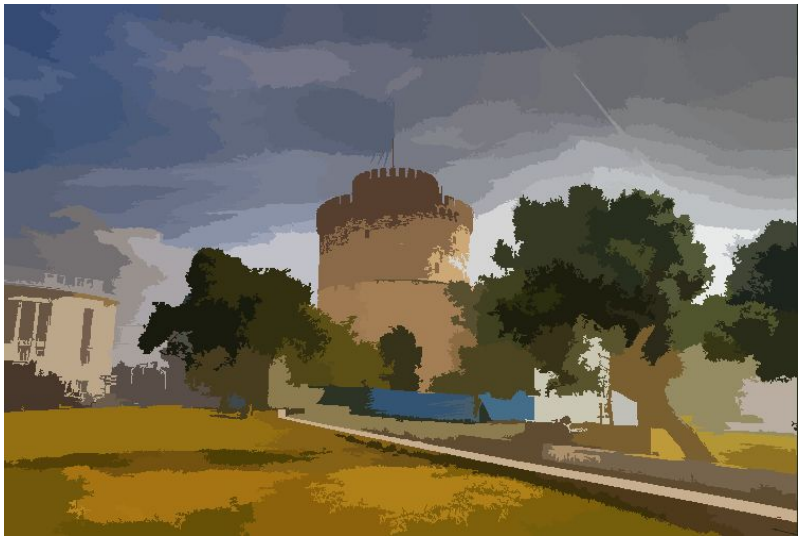
The distance follows the algorithm exactly.

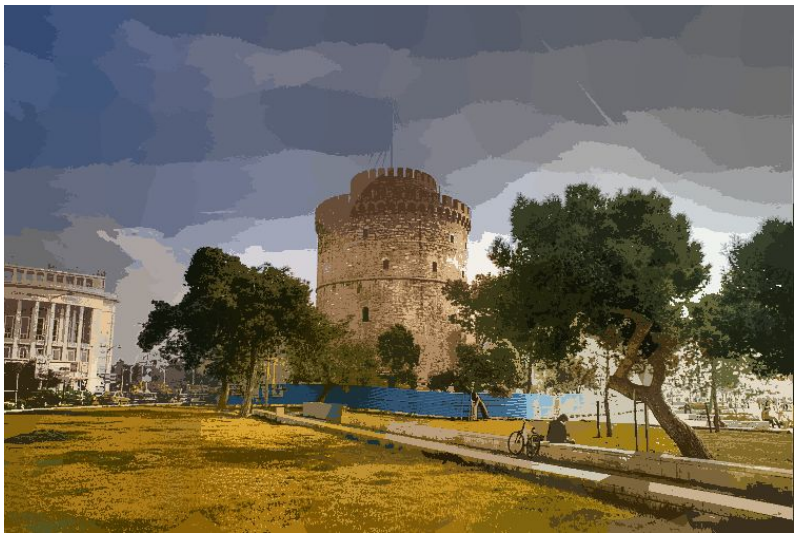
```
def squaredDifference(a,b):
    total = 0
    for i in range(len(a)):
        total += (a[i] - b[i]) ** 2
    return total

def elementDistance(a,b,s,m):
    pixelDistance = squaredDifference([a[0],a[1]],[b[0],b[1]]) / s
    colorDistance = squaredDifference(a[2:],b[2:]) / m
    return (pixelDistance + colorDistance) ** (1/2)
```

Compared to the slic runtime this is much shorter as it only has to run through the algorithm once to do the clustering. It is also a lot less sensitive to high textured areas. In hw3 we were told to divide the pixel indices by half allowing the pixel color to have more of an effect in the average center and which cluster the pixel belonged to. In this the s value is determined by the image and the m value is user defined.

I ran skimage's slic in order to get what the algorithm is intended to look like when it converges rather than what my homework did. The code for skimage's slick is in librarySlic.py.


Snic no borders


HW3 Slic no borders


Skimage Slic no borders

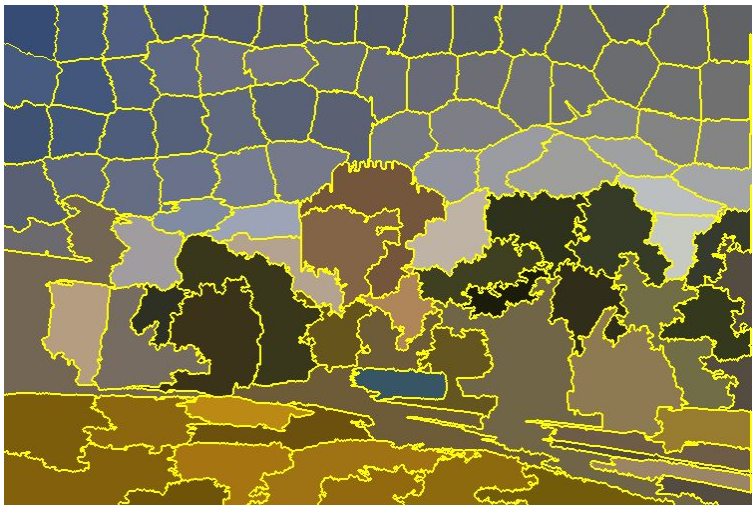
The homework implementation did well in the top half/ areas of little texture however it gets worse as it gets down to more textured areas where the online implementation is effective all around. The snic implementation is able to handle the textured areas almost as well as the skimage slic. There's some textured areas which are in high detail however that can be attributed to parameter tuning of the compressionValue. With a theoretically proven correct compression value we could get rid of the high detail in snic. Snic also results in a single pixel region in the middle of the castle. This might be due to the fact that we did a depth first search with the heap. All the pixels surrounding the single had already been assigned.


Snic centers


HW3 Slic centers


Snic borders


Hw3 slic borders


Skimage slic borders

Snic borders and centers

Slic borders and centers