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 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 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.

snic.py

```
64  def snic(image, compactnessFactor):
65      m = compactnessFactor
66      heap = []
67      centers, colors = initialCenters(image)
68      s = (image.shape[0] * image.shape[1] / len(centers)) ** (1/2)
69      xsize, ysize, _ = image.shape
70      ret = numpy.zeros(image.shape)
71      labelMap = numpy.zeros((xsize,ysize))
72      # elements in cluster are x, y, R, G, B, count
73      clusters = [[0,0,0,0,0,0] for i in range(len(centers))]
74      directions =[ [-1, 0], [-1,-1] , [-1,1] , [1,-1], [1, 0] , [1,1], [0,1], [0,-1] ]
75      for i in range(len(centers)):
76          center = centers[i]
77          color = colors[i]
78          # order is distance, k, R, G, B
79          # sort by distance
80          element = (0,i+1,color[0],color[1],color[2],center[0],center[1])
81          heapq.heappush(heap,element)
82      while len(heap) != 0:
83          target = heapq.heappop(heap)
84          distance, k, colorR, colorG, colorB, x, y = target
85          if labelMap[x][y] == 0:
86              labelMap[x][y] = k
87              updateCluster(clusters, k-1, [x,y,colorR,colorG,colorB])
88              for move in directions:
89                  moveX = x + move[0]
90                  moveY = y + move[1]
91                  if moveX >= 0 and moveX < xsize and moveY >= 0 and moveY < ysize:
92                      if labelMap[moveX][moveY] == 0:
93                          old = [x,y, colorR,colorG, colorB]
94                          targetColor = image[moveX,moveY]
95                          target = [moveX, moveY, targetColor[0], targetColor[1],targetColor[2]]
96                          d = elementDistance(target,old,s,m)
97                          e = (d, k, targetColor[0], targetColor[1], targetColor[2], moveX, moveY)
98                          heapq.heappush(heap,e)
99      for i in range(xsize):
100         for j in range(ysize):
101             clusterIndex = labelMap[i][j]
102             target = clusters[int(clusterIndex-1)]
103             ret[i][j] = [target[2]/255,target[3]/255,target[4]/255]
104     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.

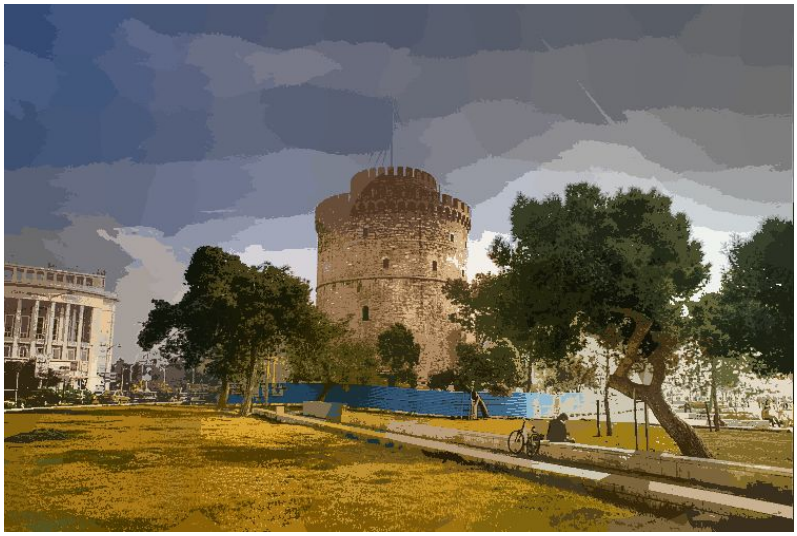
```
def squaredDifference(a,b):
    total = 0
    for i in range(len(a)):
        total += (a[i] - b[i]) ** 2
    return total

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 the 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 and m values are well chosen so that the clusters can be well formed.

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. Results are in librarySlic.py.


Snic no borders


HW3 Slic no borders


Skimage Slic no borders

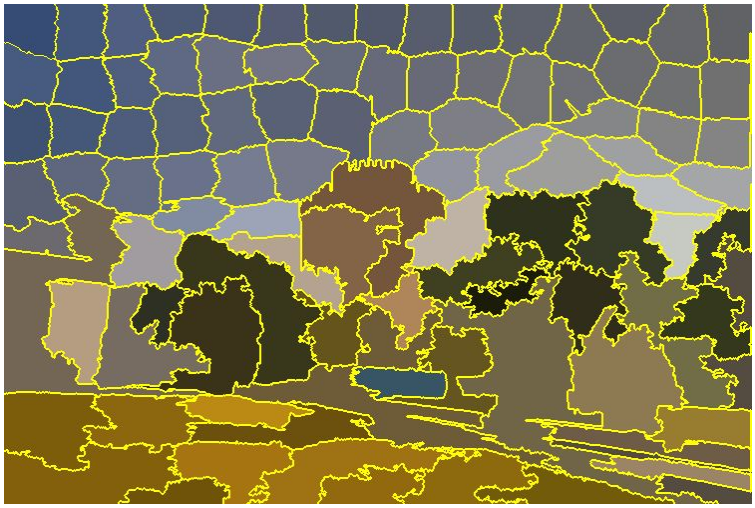
The homework implementation did well in the top half/ areas of little texture however it gets worse as it gets down to more textured areas where the online implementation is fine all around. The snic implementation has larger groupings but is able to handle the textured areas quite well.


Snic centers


HW3 Slic centers

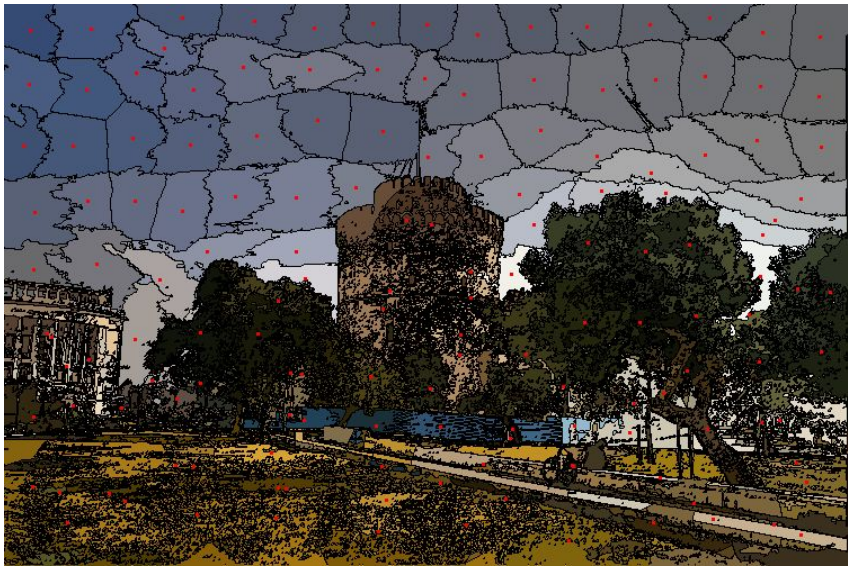

Snic borders


Hw3 slic borders


Skimage slic borders

There are a lot more borders in hw3 as it could not deal with textured areas. However, there are segments in snic which are tiny. Skimage implementation looks like a good mix of bother. The issue in hw3 was the coefficients. Dividing by 2 did not help, perhaps multiplying by 2 or reducing the color by a coefficient in order to give more weight to the distance would have helped with the textured regions. The small regions in the snic are due to this depth first approach.


Snic borders and centers


Slic borders and centers