Dan Minik

Write Up

15-8 a)

When you increase m by one, you increase the number of possible choices for the next row for lowest cost by 3*n (3 choices for each pixel wide). This then means that for the next row, another 3*n new choices are available to choose all the way for m rows and with this new change of choices, the next row is having more possibilities as well. So for increasing m, you are increasing the possible seams by m^{3n} which means as m grows, the possible seams grows exponentially.

b)

The algorithm for the seam with the lowest disruption value would be similar to finding the seam with the lowest energy given by

d[i,j] = d[i,j] if i = 0

d[I,j] + min(d[i-1][j-1], d[i-1][j], d[i-1][j+1]) if i > 0

for this algorithm, it would have to iterate through every pixel which would be $O(n^2)$ and for every pixel, it would have to make 3 choices based on finding the min value above it. So going down the array it would take $O(3n^2)$. Then to backtrack back up the possible seams it would take m time for the height of the array, and at each pixel, would have to make another three decisions which would be O(3m) so $T(n) = O(3n^2)+ O(3m)$.

Script

```
from skimage import data, io, filters
import numpy as np
import copy

#returns a W x H array, the energy at each pixel in img
def dual_gradient_energy(img):
   edges = filters.sobel(img)
   return edges

#an array of H integers, for each row it returns the column of the seam

def find_seam(img):

   seams = copy.copy(img)

   #starting at index 1 because the total cost of the first row is the same as the original
values
   #calculates the paths
   for i in range(1, len(seams)):
       for j in range(0, len(seams[0])):
           #handles if we are on the left most row of the image
           if(j == 0):
               seams[i][j] = min(seams[i-1][j], seams[i-1][j+1]) + seams[i][j]
           #handles if we are on the right most row of the image
           elif (j == (len(edges[0]) - 1)):
               seams[i][j] = min(seams[i-1][j-1], seams[i-1][j]) + seams[i][j]
           #handles if we are in the middle of the image
           else:
               seams[i][j] = min(seams[i - 1][j - 1], seams[i - 1][j], seams[i-1][j+1]) +
seams[i][j]

   #key = index for lowest cost at the end of seam
   key = 0;
   for i in range(0, len(seams[0]) - 1):
       if(seams[len(seams) - 1][i + 1] > seams[len(seams) - 1][i]):
           key = i + 1;

   # Now we start at seams[len(seams) - 1][key] and backtrack up seams, saving the lowest cost
path to the new list
   # called path
   path = [0 for i in range(len(seams))]
   path[len(seams) - 1] = key
   #place holder for lowest value index
   current_key = key
   for i in range(len(seams) - 1, -1, -1):
       if (current_key == 0):
           if(seams[i][current_key] < seams[i][current_key + 1]):
               path[i] = current_key
               current_key = current_key
           else:
```

```
                path[i] = current_key + 1
                current_key = current_key + 1
        elif (current_key == (len(seams[0])-1)):
            if (seams[i][current_key] < seams[i][current_key - 1]):
                path[i] = current_key
                current_key = current_key
            else:
                path[i] = current_key - 1
                current_key = current_key - 1
        else:
            if(seams[i][current_key - 1] < (seams[i][current_key] and seams[i][current_key + 
1])):
                path[i] = current_key - 1
                current_key = current_key - 1
            elif(seams[i][current_key] < (seams[i][current_key - 1] and seams[i][current_key + 
1])):
                path[i] = current_key
                current_key = current_key
            else:
                path[i] = current_key + 1
                current_key = current_key + 1
    return path

#displays the image after the energy function with the seam drawn on the image

def plot_seam(img, seam):
    for i in range(0, len(seam)):
        img[i][seam[i]] = 1
    io.imshow(img)
    io.show()

#modify img to remove the seam, returns the new array of floats without the seam
def remove_seam(img, seam):
    height = len(img)
    width = len(img[0]) - 1
    new_img = np.zeros((height, width), np.float32)
    for i in range(0, len(new_img)):
        new_img_counter = 0
        for j in range(0, len(new_img[0])):
            if j != seam[i]:
                new_img[i][new_img_counter] = img[i][j]
                new_img_counter += 1
            else:
                j += 1
    return new_img



img = data.imread('/Users/danminik/Desktop/Junior Year/Algorithms/Project2Python/Test.png', 
True)
```

```
edges = dual_gradient_energy(img)
seam = find_seam(edges)
plot_seam(edges, seam)
new_img = remove_seam(img, seam)
io.imshow(new_img)
io.show()
```