1. Jupyter Notebook and NumPy Warmup [20pts]

We will make extensive use of Python's numerical arrays (NumPy) and interactive plotting (Matplotlib) in Jupyter notebooks for the course assignments. The first part of this assignment is intended as a gentle warm up in case you haven't used these tools before. Start by reading through the following tutorials:

If you haven't used Jupyter before, a good place to start is with the introductory documentation here:

https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#starting-the-notebook-server (https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#starting-the-notebook-server)
https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Note
(https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Note
https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Note
(https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Note

This page gives a good introduction to NumPy and many examples of using NumPy along with Matplotlib:

http://www.scipy-lectures.org/intro/numpy/numpy.html (http://www.scipy-lectures.org/intro/numpy/numpy.html)

You should also get comfortable with searching through the documentation as needed

https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html
(https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html)
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html
(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html)

NumPy Array Operations

Describe in words what each of each of the following statements does and what the value of `result` will be (i.e. if you were to execute `print(result)`). You should do this with out actually executing the code but instead just looking it and refering to the NumPy documentation.

[1.1]

```
import numpy as np
a = np.arange(5,15)
result = a[::3]
```

1. find and import the module "numpy" into local space and define the name as "np"
2. create an array starting from 5 to 14, and store the array into variable "a"
3. in a, starting at index 0, use every other 2 element to create a new array; and store the new array into result

---

[1.2]

```
a = np.arange(1,5)
result = a[::-1]
```

1. create a new array starting from 1 to 4, store the new array into "a"
2. use every element in a to create a new array, but in reversed order

---

[1.3]

```
f = np.arange(1840,1860)
g = np.where(f>1850)
result = f[g]
```

1. create an array starting from 1840 to 1859; store the array into "f"
2. create an array whose elements are the index of "f" that holds true for element in f for "greater than 1850"; store the array into "g" (index of element that are greater than 1850)
3. create an array whose elements are f's elements that indexed by g's elements (all elements in f that are greater than 1850)

[1.4]

```
x = np.ones((1,10))
result = x.sum(axis=1)
```

1. create an 1x10 array, every element is 1 in float64 type
2. sum the element of the array on the axis 1, and create an array of the sum

NumPy Coding Exercises

Add or modify the code in the cells below as needed to carry out the following steps.

[1.5]

Use **matplotlib.pyplot.imread** to load in a grayscale image of your choice. If you don't have a grayscale image handy, load in a color image and then convert it to grayscale averaging together the three color channels (use **numpy.mean**).

Finally create an array A that contains the pixels in a 100x100 sub-region of your image and display the image in the notebook using the **matplotlib.pyplot.imshow** function.

HINT: When loading an image with **imread** it is important to example the data type of the returned array. Depending on the image it may be that `I.dtype = uint8` or `I.dtype = float32` . Integer values range in [0..255] while floating point values for an image will be in [0..1]. A simple approach is to always convert images to floats, this will avoid much confusion and potential bugs later on.

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt

#load an image
I = plt.imread('https://upload.wikimedia.org/wikipedia/commons/f/f

#display the shape of the array and data type
print("I.shape=",I.shape,"\nI.dtype=",I.dtype)

#convert to float data type and scale to [0..1] if necessary
if (I.dtype == np.uint8):
    I = I.astype(float) / 256

#I.dtype should now be float

#if your image is color (shape HxWx3), convert to grayscale by ave


#display the image in the notebook using a grayscale colormap
plt.imshow(I,cmap=plt.cm.gray)

#force matplotlib to go ahead and display the plot now
plt.show()

#select out a 100x100 pixel subregion of the image
A = I[0:100, 0:100]

#display the selected subregion
# plt.imshow(0.1*A,cmap=plt.cm.gray)
plt.imshow(A,cmap=plt.cm.gray)
plt.show()

```

```
I.shape= (200, 150)
I.dtype= float32

<Figure size 640x480 with 1 Axes>

<Figure size 640x480 with 1 Axes>
```

[1.6]

In the cell below, describe what happens if you comment out the `plt.show()` lines?

How does the visualization of `A` change if you scale the brightness values (i.e. `plt.imshow(0.1*A,cmap=plt.cm.gray)`)?

Explain what is happening, referring to the **matplotlib** documentation as necessary (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html))

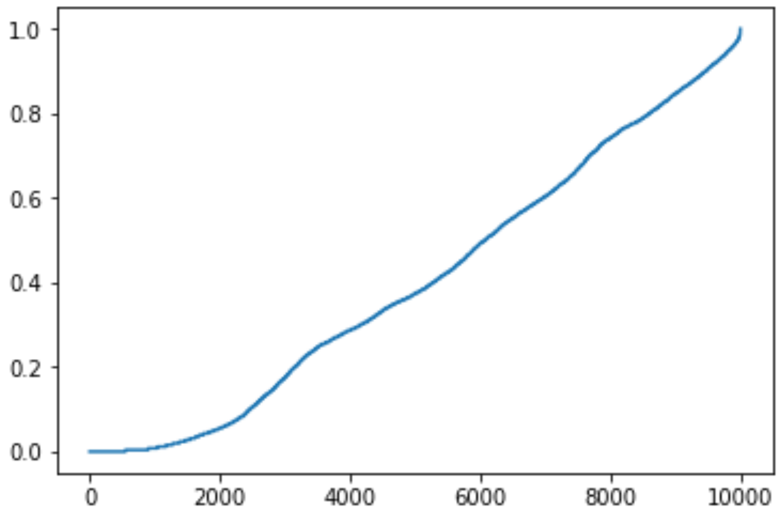
1. Commenting out plt.show() will raise error because show() tells matplotlib to brings the window and start mainloop which is disabled by default.
2. Scale the brightness value does not affect the visulization, because norm = None by default for imshow(). In other words, A's scalar data is noramlized to map color when it's not specified.

[1.7]

Write code in the cell below which (a) puts the values of `A` into a singe 10,000-dimensional column vector `x`, (b) sorts the entries in `x`, and (c) visualizes the contents of the sorted vector `x` by using the **matplotlib.pyplot.plot** function

In [2]:

```
x = np.reshape(A, (1,10000))
x = np.sort(x, axis=1)
plt.plot(x[0])
plt.show()
```

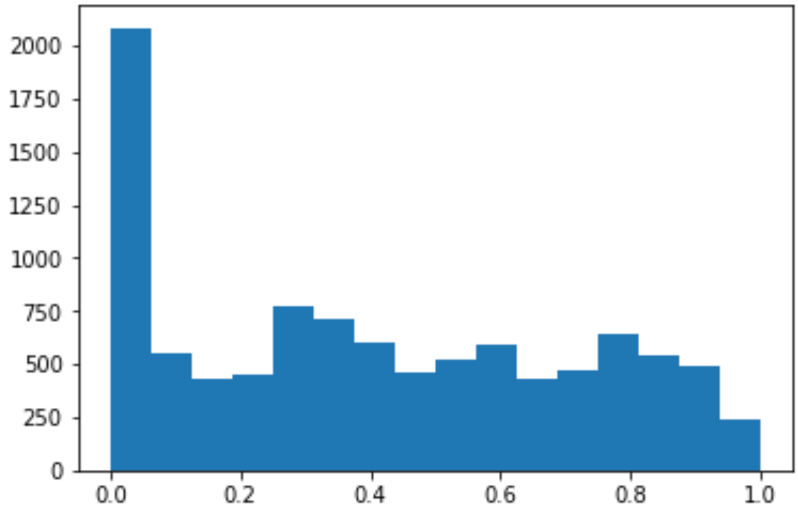


[1.8]

Display a figure showing a histogram of the pixel intensities in `A` using **matplotlib.axes.hist**. Your histogram should have 16 bins. You will need to convert `A` to a vector in order for the histogram to display correctly (otherwise it will show 16 bars for each row of A)

In [3]:

```
#[your code here]

plt.hist(A.flatten(), 16)
plt.show()

```

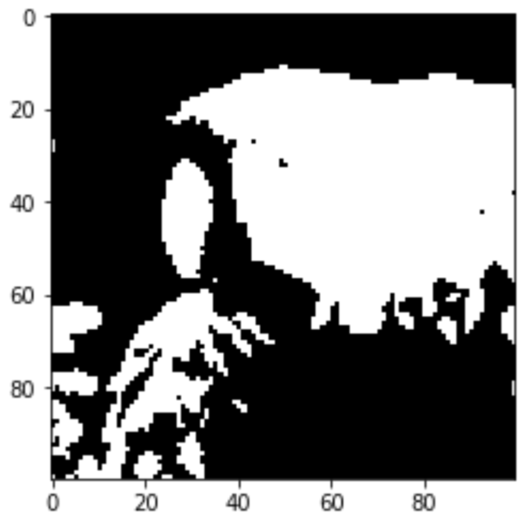


[1.9]

Create and display a new (binary) image the same size as `A` , which is white wherever the intensity in `A` is greater than a threshold specified by a variable `t` , and black everywhere else. Experiment in order to choose a value for the threshold which makes the image roughly half-white and half-black. Also print out the percentage of pixels which are black for your chosen threshold.

In [4]:

```
#[your code here]
t = 0.5
A2 = np.where(A>t, 1, 0)
plt.imshow(A2,cmap=plt.cm.gray)
plt.show()
```

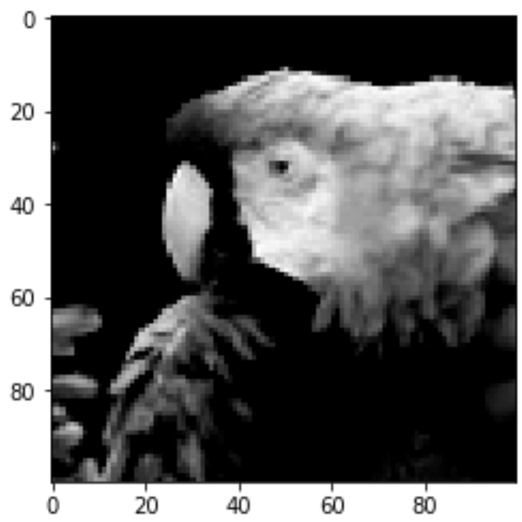


[1.10]

Generate a new grayscale image, which is the same as A, but with A's mean intensity value subtracted from each pixel. After subtracting the mean, set any negative values to 0 and display the result.

In [5]:

```
#[your code here]
mean = np.mean(A)
A3 = np.subtract(A, mean)
A3 = np.where(A3 < 0, 0, A3)
plt.imshow(A3, cmap=plt.cm.gray)
plt.show()
```



[1.11]

Let `y` be a column vector: `y = [1, 2, 3, 4, 5, 6]` so that `y.shape = (6,1)`. Reshape the vector into a matrix `z` using the **numpy.array.reshape** and (**numpy.array.transpose** if necessary) to form a new matrix `z` whose first column is `[1, 2, 3]`, and whose second column is `[4, 5, 6]`. Print out the resulting array `z`

In [6]:

```
#[your code here]
y = np.array([1,2,3,4,5,6])
z = y.reshape(2,3)
print(z)
```

```
[[1 2 3]
 [4 5 6]]
```

[1.12]

Find the minimum value of `A` , if there are multple entries with the same minimum value it is fine to return the first one. Set `r` to be the row in which it occurs and `c` to be the column. Print out `r` , `c` , and `A[r,c]`

In [7]:
```
#[your code here]
min = np.argmin(A)
index = np.unravel_index(min, A.shape)
r = index[0]
c = index[1]
print(r, c, A[r,c])
```

```
0 0 0.0
```

[1.13]

Let v be the vector: `v = [1,8,8,2,1,3,9,8]` . Using the unique function, compute and print the total number of unique values that occur in `v` .

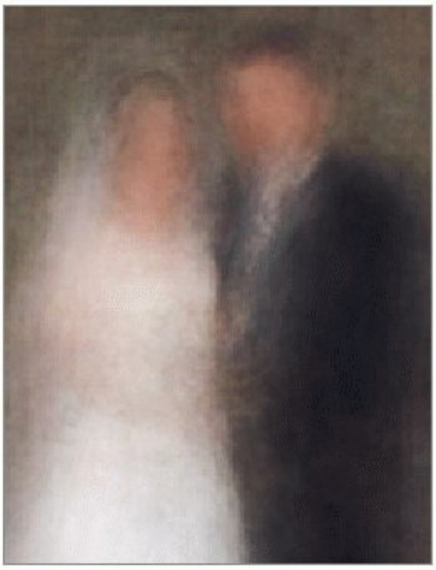
In [8]:
```
#[your code here]
v = np.array([1,8,8,2,1,3,9,8])
u = np.unique(v)
print(u.size)
```

```
5
```

2. Averaging Images [40pts]

In this exercise you will write code which loads a collection of images (which are all the same size), computes a pixelwise average of the images, and displays the resulting average.

The images below give some examples that were generated by averaging "100 unique commemorative photographs culled from the internet" by Jason Salavon. Your program will do something similar.



Newlyweds

Little Leaguer

Kids with Santa

The Graduate

Download the images provided on the Canvas course website for this assignment `averageimage_data.zip` . There are two sets, `set1` and `set2` . Notice that they are all the same size within a single set.

[2.1]

Write a function in the cell below that loads in one of the sets of images and computes their average. You can use the **os.listdir** to get the list of files in the directory. As you load in the images, you should compute an average image on the fly. Color images are represented by a 3-dimensional array of size (HxWx3) where the third dimension indexes the red, green and blue channels. You will want to compute a running average of the red, green and blue slices in order to get your final average color image.

You should encapsulate your code in a function called **average_image** that takes the image directory as an input and returns the average of the images in that directory. Your function should implement some error checking. Specifically your function should skip over any files in the directory that are not images (**plt.imread** will thrown an **OSError** if the file is not an image). It should ignore images that are not color images. Finally, it should also skip any images which are not the same height and width as the first color image you load in.

In [9]:

```python
#
# these are the only modules needed for problem #2
#
import numpy as np
import os
import matplotlib.pyplot as plt

```

In [10]:

```python
def average_image(dirname):
    """
    Computes the average of all color images in a specified direct

    The function ignores any images that are not color images and
    the same size as the first color image you load in

    Parameters
    ----------
    dirname : str
        Directory to search for images

    Returns
    -------
    numpy.array (dtype=float)
        HxWx3 array containing the average of the images found

    """

    #[your code here]
    Iaverage = None
    h = 0
    w = 0
    counter = 0

    tmp = 0

    for file in os.listdir(dirname):
        filename = os.path.join(dirname,file)
        if os.path.isfile(filename):

            #[your code here]
            try:
                I = plt.imread(filename)

                #check color and size
                #I.dtype = uint8

                Ishape = I.shape
                if (len(Ishape) != 3):
                    continue
```

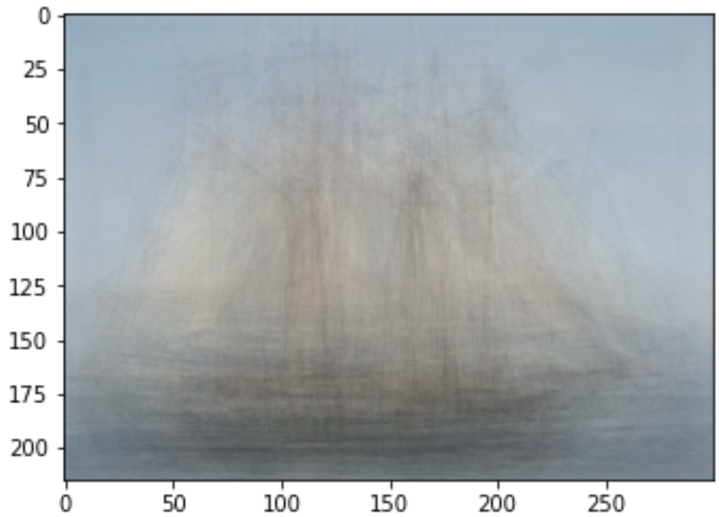
```
                    continue
                if (h == 0 or w == 0):
                    h = Ishape[0]
                    w = Ishape[1]
                    Iaverage = np.zeros((h, w, 3), dtype = np.floa
                else:
                    if (Ishape[0] != h or Ishape[1] != w):
                        continue

            except OSError:
                pass

            finally:
                counter += 1
                Iaverage = np.add(Iaverage, I)

    Iaverage = np.array(Iaverage/counter, dtype = np.uint8)
    return Iaverage
```

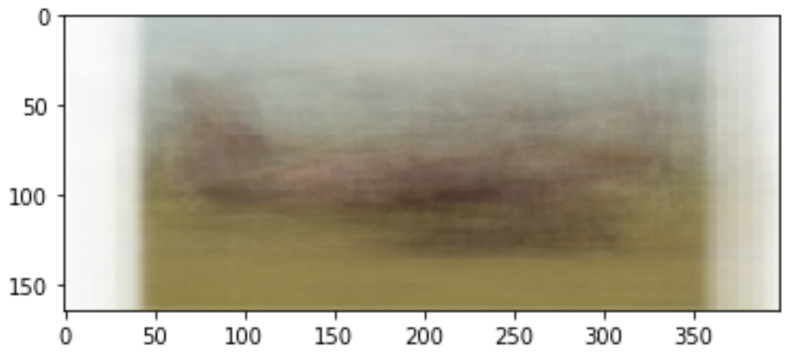
[2.2]

Write code below which calls your **average_image()** function twice, once for each set of images. Display the resulting average images. Also display a single example image from each set for comparison

In [11]:

```
Iav = average_image("averageimage_data/set1/")
plt.imshow(Iav)
plt.show()

Iav = average_image("averageimage_data/set2/")
plt.imshow(Iav)
plt.show()

#[your code here]
```





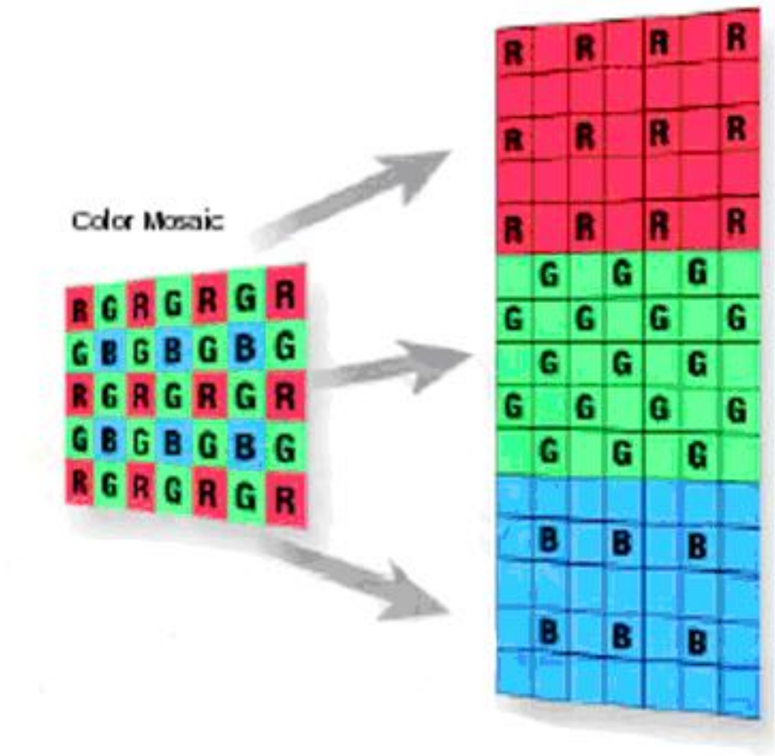
[2.3]

Provide a description of the appearance of the average images. Give an explanation as to why the average image does not look like the individual example images.

The average image's content has blurred shape and outline, but it is possible to guess the content. It looks like a sketch. Due to the set of images have the same content, such as ships for set1, the pixel's color distribution are relatively similar. Therefore, the averaged result will present a blured shape.

3. Color sensor demosaicing [40pts]

As discussed in class, there are several steps to transform raw sensor measurements into nice looking images. These steps include Demosaicing, White Balancing and Gamma Correction. In this problem we will implement the demosaicing step. (see Szeliski Chapter 2.3) In the assignment data directory on Canvas there is a zip file containing raw images from a Canon 20D camera as well as corresponding JPEG images from the camera (`*.JPG`). The raw image files (`*.CR2`) have been converted to 16-bit PGM images (`*.pgm`) using David Coffin's dcraw program to make it easy to load them in as arrays using the supplied code below **read_pgm**



Bayer RGGB mosaic.

The raw image has just one value per pixel. The sensor is covered with a filter array that modifies the sensitivity curve of each pixel. There are three types of filters: "red", "green", and "blue", arranged in the following pattern repeated from the top left corner:

```
R G . . .
G B
.
.
.
```

Your job is to compute the missing color values at each pixel to produce a full RGB image (3 values at each pixel location). For example, for each "green" pixel, you need to compute "blue" and "red" values. Do this by interpolating values from adjacent pixels using the linear interpolation scheme we described in class.

In [12]:

```
#
# these are the only modules needed for problem #3
#
import numpy as np
import matplotlib.pyplot as plt

#
# this function will load in the raw mosaiced data stored in the p
#
def read_pgm(filename):
    """
    Return image data from a raw PGM file as a numpy array
    Format specification: http://netpbm.sourceforge.net/doc/pgm.ht

    """
    infile = open(filename,'r',encoding="ISO-8859-1")

    # read in header
    magic = infile.readline()
    width,height = [int(item) for item in infile.readline().split(
    maxval = infile.readline()

    # read in image data and reshape to 2D array, convert 16bit to
    image = np.fromfile(infile, dtype='>u2').reshape((height, widt
    image = image.astype(float)/65535.
    return image

```

[3.1]

Implement a function `demosaic` which takes an array representing the raw image and returns a standard color image. To receive full credit, you should implement this using NumPy indexing operations like you practiced in the first part of the assignment. You should not need any for loops over individual pixel locations. You can accomplish this by either using array subindexing or alternately by using the imfilter function with the appropriate choice of filter.

In [13]:

```
def demosaic(I):
    """
    Demosaic a Bayer RG/GB image to an RGB image.

    Parameters
    ----------
    I : numpy.array  (dtype=float)
        RG/GB mosaic image of size  HxW

    Returns
    -------
    numpy.array (dtype=float)
    HxWx3 array containing the demosaiced RGB image

    """
    h = I.shape[0]
    w = I.shape[1]
    I_rgb = np.zeros((h,w,3), dtype = np.float32)

    offset_w = 1 if (w%2 == 0) else 2
    offset_h = 1 if (h%2 == 0) else 2

    # R value
    I_rgb[::2, ::2, 0] =  I[::2, 0::2]

    I_rgb[::2, 1:w-1:2, 0] = 0.5 * I[:2:2, 0:w-2:2] + 0.5 * I[::2,

    I_rgb[1:h-1:2, ::, 0] = 0.5 * I[0:h-2:2, ::] + 0.5 * I[2::2, :


    # G value
    I_rgb[::2, 1::2, 1] = I[::2, 1::2]
    I_rgb[1::2, ::2, 1] = I[1::2, ::2]

    I_rgb[::2, 2:w-1:2, 1] = 0.5 * I[::2, 1:w-2:2] + 0.5 * I[::2,
    I_rgb[1::2, 1:w-1:2, 1] = 0.5 * I[1::2, 0:w-2:2] + 0.5 * I[1::


    # B value
    I_rgb[1::2, 1::2, 2] = I[1::2, 1::2]

    I_rgb[1::2, 2:w-1:2, 2] = 0.5 * I[1::2, 1:w-2:2] + 0.5 * I[1::

    I_rgb[2:h-1:2, ::, 2] = 0.5 * I[1:h-2:2, ::] + 0.5 * I[3::2, :


    #[enter your code here]
    return I_rgb
```

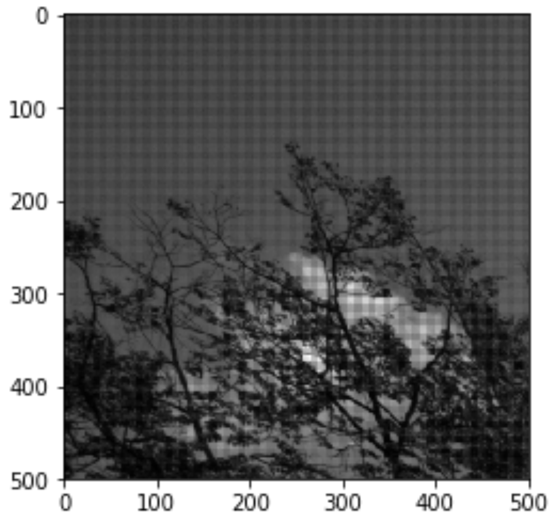
[3.2]

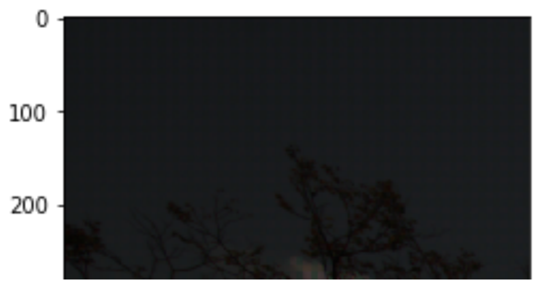
Write code and comments below that demonstrate the results of your demosaic function using `IMG_1308.pgm` . You are encouraged to include multiple examples for illustration. Since the images are so large, work with just the upper-left 500x500 pixel sub-block for illustrations.

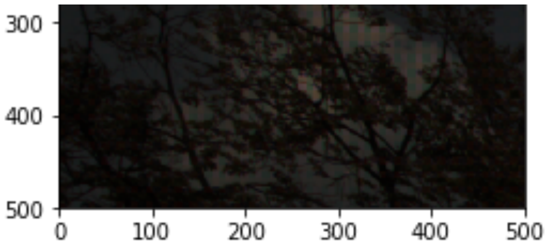
You should display: (a) the original raw image with a grayscale colormap, (b) the resulting RGB image after demosaicing, (c) the corresponding part of the provided JPG file from the camera

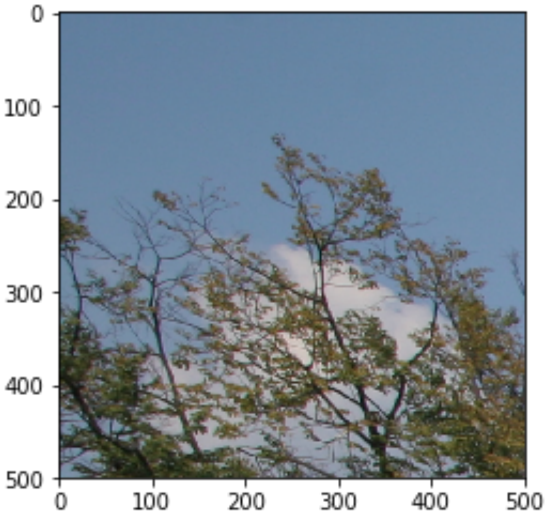
In [14]:

```
Iraw = read_pgm("demosaic/IMG_1308.pgm")

# grayscale
plt.imshow(Iraw[:500:, :500:], cmap = plt.cm.gray)
plt.show()

#RGB
IRGB = demosaic(Iraw[:500:, :500:])
plt.imshow(IRGB)
plt.show()

#JPG
IJPG = plt.imread("demosaic/IMG_1308.jpg")
plt.imshow(IJPG[:500:, :500:])
plt.show()

#[enter your code here]

```









[3.3]

The correctly demosaiced image will appear darker than the JPG version provided. Provide an explanation of why this is the case based on your reading about the digital camera pipeline.

In the pipleline of digital camear, there is gamma correction. Human's perception to light is non-linear, but digital camera is. Therefore the gamma correction will compensate the light. The demosaiced image is not corrected, so it's darker than the JPG version.