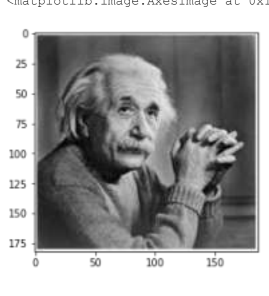
Solution 1

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
```

In [2]:
```
einstein = cv2.imread("einstein.jpg", cv2.IMREAD_GRAYSCALE)
plt.imshow(einstein, cmap='gray')
```

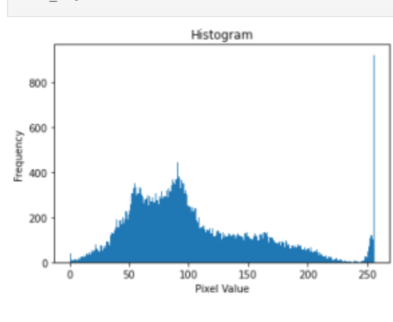
Out[2]: <matplotlib.image.AxesImage at 0x1e586d2cd60>



1.a)

In [3]:
```
def hist_img(image):
    hist = plt.hist(image.flatten(), 256, [0, 256])
    # Show the histogram
    plt.title('Histogram')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.show()
```

In [4]:
```
hist_img(einstein)
```

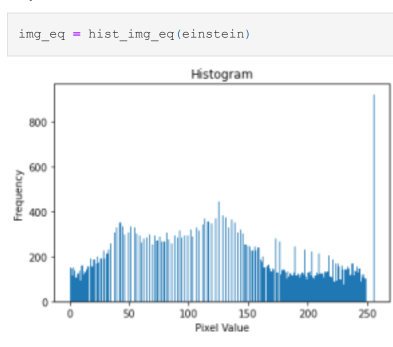


1.b)

In [5]:
```
def hist_img_eq(image):
    hist, bin_edges = np.histogram(image.flatten(),256, [0,256])
    cdf = hist.cumsum()
    cdf_n = (cdf-cdf.min())*255/ (cdf.max()-cdf.min())
    table = np.interp(np.arange(256), bin_edges[:-1], cdf_n).astype(np.uint8)
    img_eq = table[image]
    hist_eq = plt.hist(img_eq.flatten(), 256, [0, 256])
    # Show the histogram
    plt.title('Histogram')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.show()
    return img_eq
```

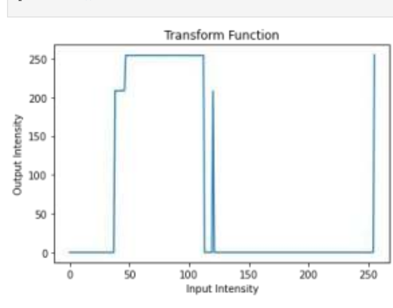
1.c)

In [6]:
```
img_eq = hist_img_eq(einstein)
```



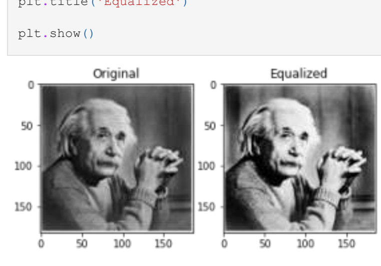
1.d)

In [7]:
```
hist_eq, bin_edges_eq = np.histogram(img_eq.flatten(),256, [0,256])
hist, bin_edges = np.histogram(einstein.flatten(),256, [0,256])
cdf_eq = hist_eq.cumsum()
trans_func = np.interp(hist, hist_eq, np.arange(256))
plt.plot(trans_func)
plt.title('Transform Function')
plt.xlabel('Input Intensity')
plt.ylabel('Output Intensity')
plt.show()
```



1.e)

In [8]:
```
# Display the original and equalized images
plt.subplot(121)
plt.imshow(einstein, cmap='gray')
plt.title('Original')

plt.subplot(122)
plt.imshow(img_eq, cmap='gray')
plt.title('Equalized')

plt.show()
```



Solution 2

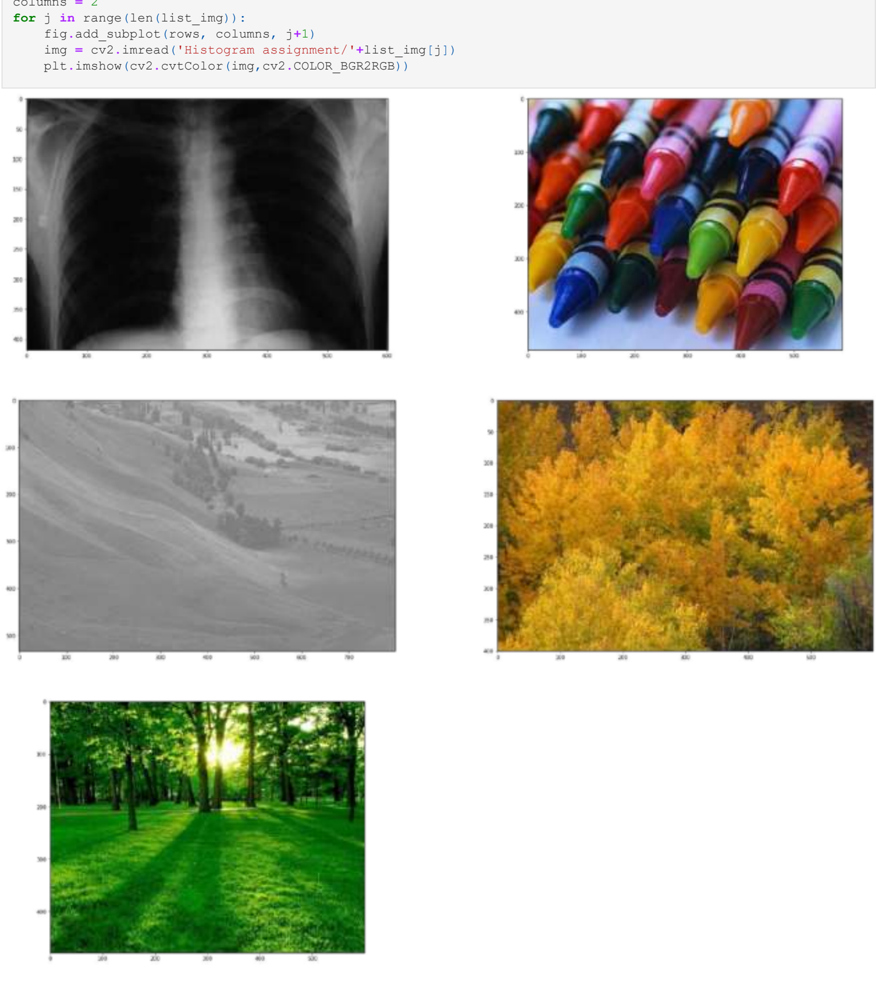
In [9]:
```
import zipfile
with zipfile.ZipFile('Histogram assignment-20230301T131449Z-001.zip', 'r') as zip_ref:
    # Extract all files to the specified extract path
    zip_ref.extractall()
```

In [10]:
```
import os
list_img = os.listdir('Histogram assignment')
```

In [11]:
```
len(list_img)
```

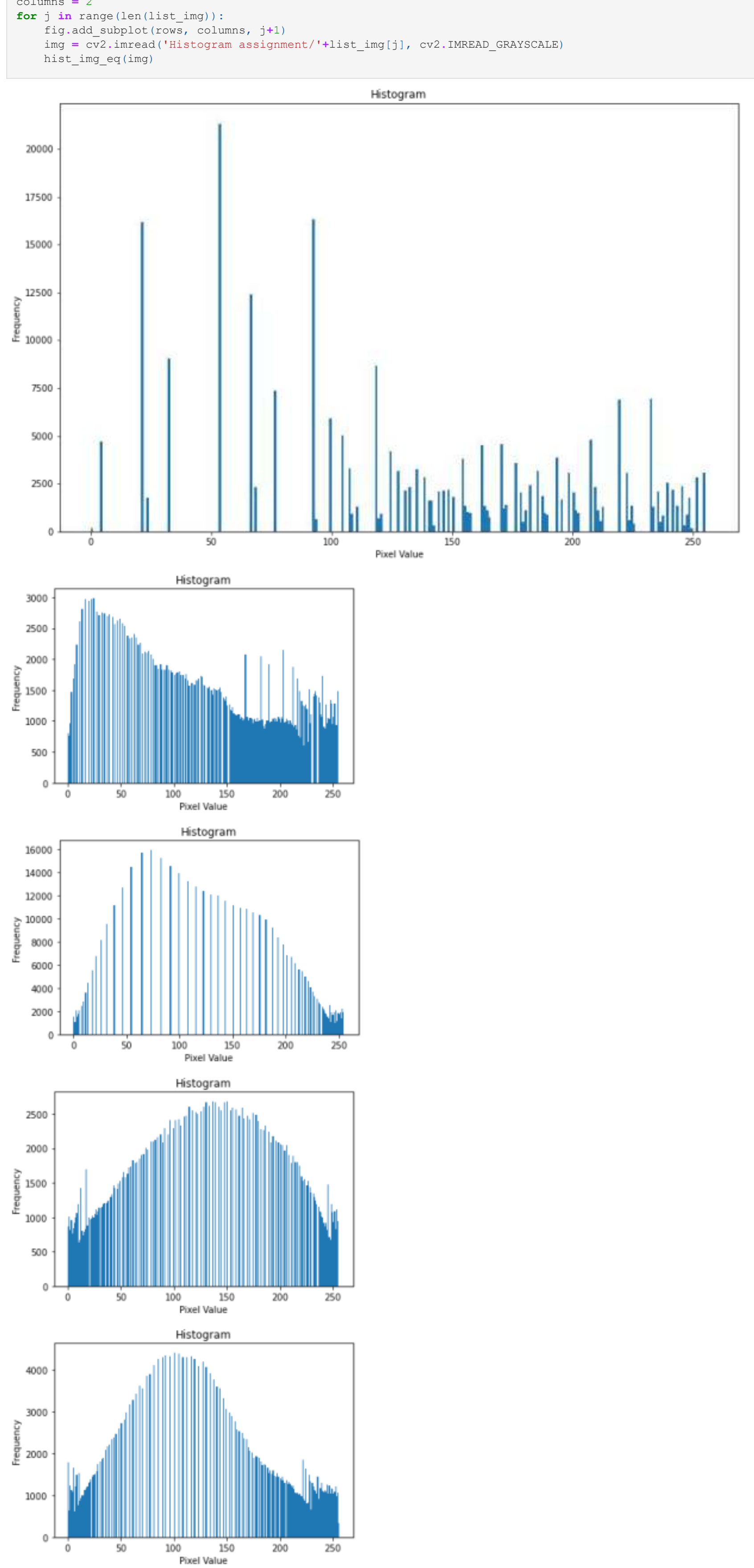
Out[11]: 5

In [12]:
```
fig = plt.figure(figsize=(30, 30))
rows = 3
columns = 2
for j in range(len(list_img)):
    fig.add_subplot(rows, columns, j+1)
    img = cv2.imread('Histogram assignment/'+list_img[j])
    plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

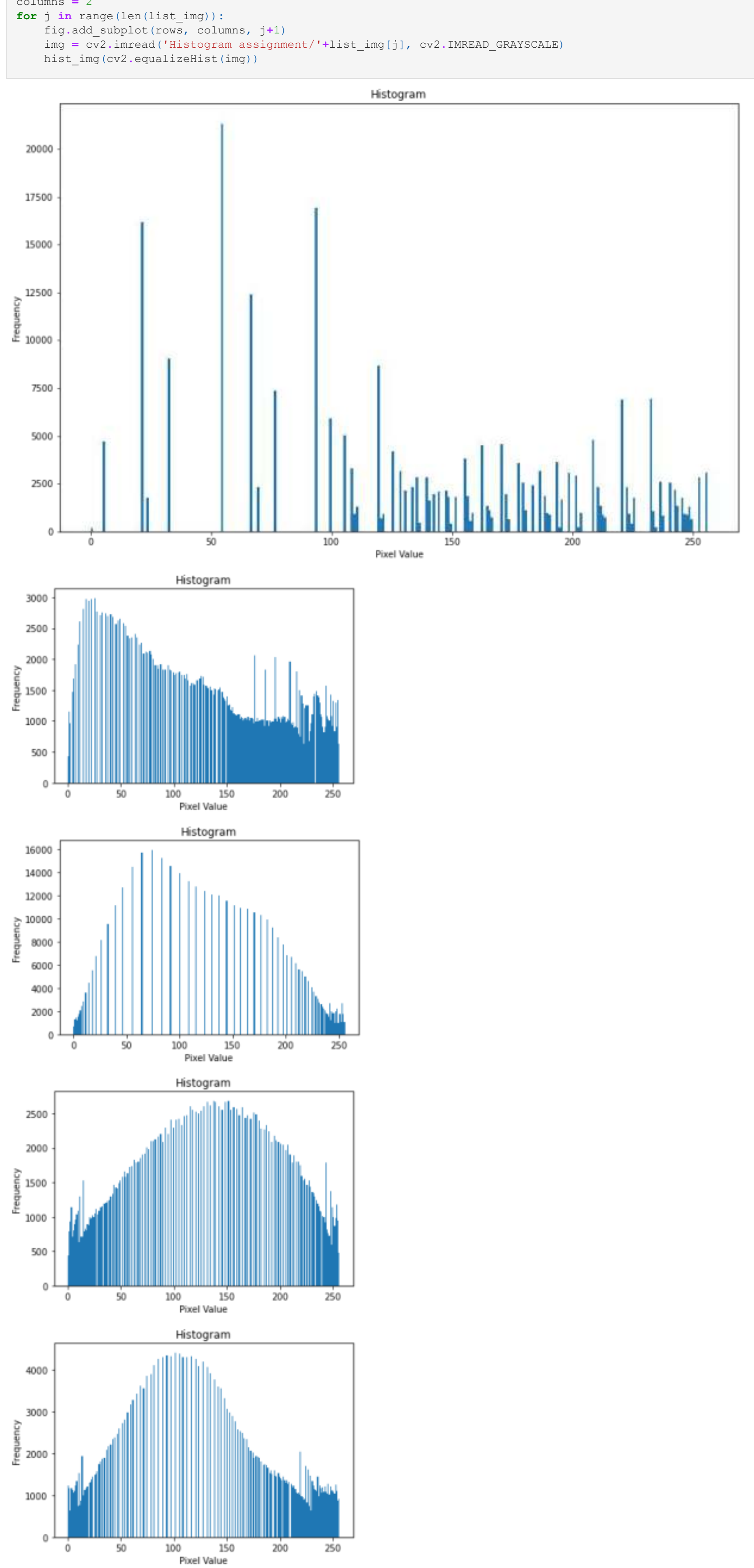


2.a)

In [13]:
```
fig = plt.figure(figsize=(30, 30))
rows = 3
columns = 2
for j in range(len(list_img)):
    fig.add_subplot(rows, columns, j+1)
    img = cv2.imread('Histogram assignment/'+list_img[j], cv2.IMREAD_GRAYSCALE)
    hist_img_eq(img)
```



In [14]:
```
fig = plt.figure(figsize=(30, 30))
rows = 3
columns = 2
for j in range(len(list_img)):
    fig.add_subplot(rows, columns, j+1)
    img = cv2.imread('Histogram assignment/'+list_img[j], cv2.IMREAD_GRAYSCALE)
    hist_img(cv2.equalizeHist(img))
```



If we look at the histogram obtained by both the medthods for each of the images, we can see the histograms look very similar.

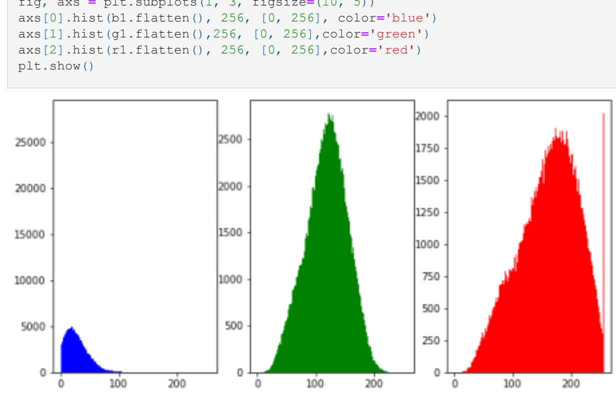
2.b)

The intensities obtained are not uniform. In histogram equalization our target is to redistribute the intensities to maximize te contrast of the image so that it becomes appealing to the human eyes. The aim of histogram equalization is not create a uniform curve of intensity values. Hence, depending upon the inensities of the image, the intensities obtained can be non uniform.

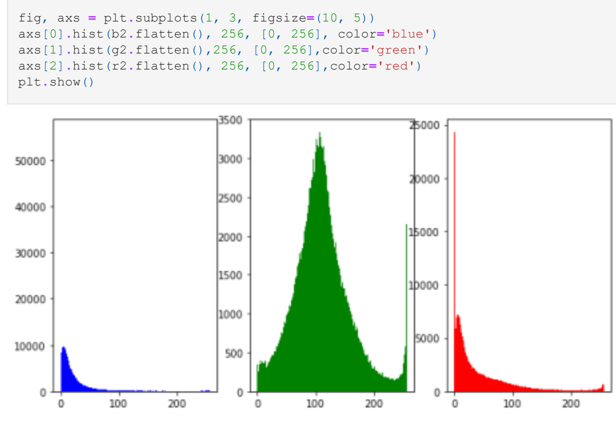
Solution 3

3.a)

In [15]:
```
img1 = cv2.imread('Histogram assignment/image1.jpg')

# Split the image into its color channels
b1, g1, r1 = cv2.split(img1)

fig, axs = plt.subplots(1, 3, figsize=(10, 5))
axs[0].hist(b1.flatten(), 256, [0, 256], color='blue')
axs[1].hist(g1.flatten(),256, [0, 256],color='green')
axs[2].hist(r1.flatten(), 256, [0, 256],color='red')
plt.show()
```



In [16]:
```
img2 = cv2.imread('Histogram assignment/image2.jpg')

# Split the image into its color channels
b2, g2, r2 = cv2.split(img2)

fig, axs = plt.subplots(1, 3, figsize=(10, 5))
axs[0].hist(b2.flatten(), 256, [0, 256], color='blue')
axs[1].hist(g2.flatten(),256, [0, 256],color='green')
axs[2].hist(r2.flatten(), 256, [0, 256],color='red')
plt.show()
```



3.b)

In [17]:
```
b1_hist, _ = np.histogram(b1.ravel(), 256, [0, 256])
b2_hist, _ = np.histogram(b2.ravel(), 256, [0, 256])
b1_val , b1_id, b1_counts = np.unique(b1.flatten(), return_inverse = True,return_counts=True )
b2_val, b2_counts = np.unique(b2.flatten(), return_counts=True)
cdf1_b = np.cumsum(b1_counts)/b1.size
#cdf1_n_b = cdf1_b * float(b1_hist.max()) / cdf1_b.max()

cdf2_b = np.cumsum(b2_counts)/b2.size
#cdf2_n_b = cdf2_b * float(b2_hist.max()) / cdf2_b.max()

b_interp = np.interp(cdf1_b, cdf2_b, b2_val)
b_match = b_interp[b1_id].reshape(b1.shape)
```

In [18]:
```
g1_hist, _ = np.histogram(g1.ravel(), 256, [0, 256])
g2_hist, _ = np.histogram(g2.ravel(), 256, [0, 256])
g1_val , g1_id, g1_counts = np.unique(g1.flatten(), return_inverse = True,return_counts=True )
g2_val, g2_counts = np.unique(g2.flatten(), return_counts=True)
cdf1_g = np.cumsum(g1_counts)/g1.size
#cdf1_n_b = cdf1_b * float(b1_hist.max()) / cdf1_b.max()

cdf2_g = np.cumsum(g2_counts)/g2.size
#cdf2_n_b = cdf2_b * float(b2_hist.max()) / cdf2_b.max()

g_interp = np.interp(cdf1_g, cdf2_g, g2_val)
g_match = g_interp[g1_id].reshape(g1.shape)
```

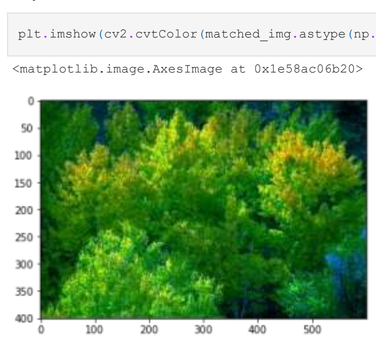
In [19]:
```
r1_hist, _ = np.histogram(r1.ravel(), 256, [0, 256])
r2_hist, _ = np.histogram(r2.ravel(), 256, [0, 256])
r1_val , r1_id, r1_counts = np.unique(r1.flatten(), return_inverse = True,return_counts=True )
r2_val, r2_counts = np.unique(r2.flatten(), return_counts=True)
cdf1_r = np.cumsum(r1_counts)/r1.size
#cdf1_n_b = cdf1_b * float(b1_hist.max()) / cdf1_b.max()

cdf2_r = np.cumsum(r2_counts)/r2.size
#cdf2_n_b = cdf2_b * float(b2_hist.max()) / cdf2_b.max()

r_interp = np.interp(cdf1_r, cdf2_r, r2_val)
r_match = r_interp[r1_id].reshape(r1.shape)
```

In [20]:
```
matched_channels = [b_match, g_match, r_match]
matched_img = cv2.merge(matched_channels)
```

3.c)

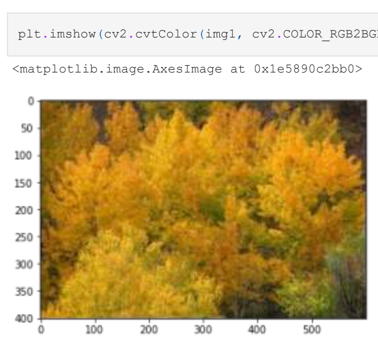
In [21]:
```
plt.imshow(cv2.cvtColor(matched_img.astype(np.uint8), cv2.COLOR_RGB2BGR))
```

Out[21]: <matplotlib.image.AxesImage at 0x1e58ac06b20>



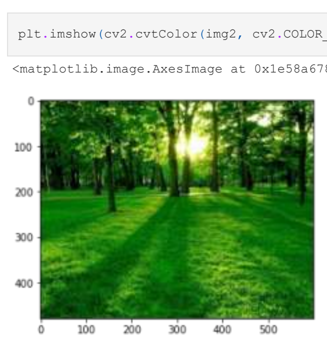
In [22]:
```
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_RGB2BGR))
```

Out[22]: <matplotlib.image.AxesImage at 0x1e5890c2bb0>



In [23]:
```
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_RGB2BGR))
```

Out[23]: <matplotlib.image.AxesImage at 0x1e58a6781f0>



3.d)

Image1 had a lot of yellow colour wherease image2 had green, we can see after matching image1 has also become greenish.

Solution 4

4.a)

In [24]:
```
k = 99
```

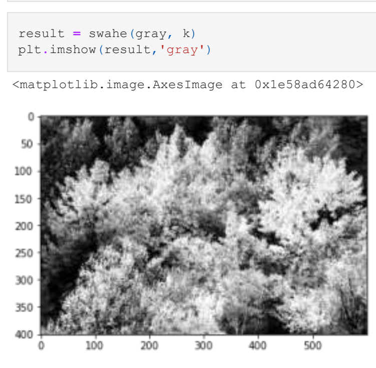
4.b)

In [25]:
```
gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
```

In [26]:
```
def sliding_window(image, k):
    img_crop = []
    for i in range(0, image.shape[0] - k):
        for j in range(0, image.shape[1] -k):
            img_crop.append((i, j, image[i:i+k+1, j:j+k+1]))
    return img_crop

def swahe(image, k):
    # Split the image into small windows and apply histogram equalization on each window
    result = np.zeros_like(image)
    for (i, j, window) in sliding_window(image, k):
        window_eq = cv2.equalizeHist(window)
        result[i:i+k+1, j:j+k+1] = window_eq

    return result
```

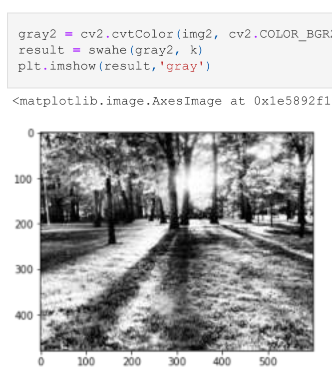
In [27]:
```
result = swahe(gray, k)
plt.imshow(result,'gray')
```

Out[27]: <matplotlib.image.AxesImage at 0x1e58ad64280>



In [28]:
```
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
result = swahe(gray2, k)
plt.imshow(result,'gray')
```

Out[28]: <matplotlib.image.AxesImage at 0x1e5892f1a60>

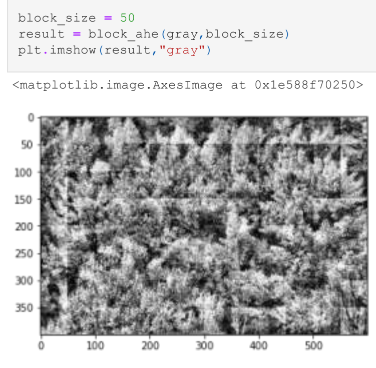


4.c)

In [29]:
```
def block_ahe(image, block_size):
    # Divide the image into blocks of block_size
    h, w = image.shape
    blocks_h = h // block_size
    blocks_w = w // block_size
    blocks = [image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size] for i in range(blocks_h) for

    # Apply histogram equalization on each block separately
    for i, block in enumerate(blocks):
        blocks[i] = cv2.equalizeHist(block)

    # Combine the blocks into a single image
    result = np.vstack([np.hstack(blocks[j*blocks_w:(j+1)*blocks_w]) for j in range(blocks_h)])

    return result
```

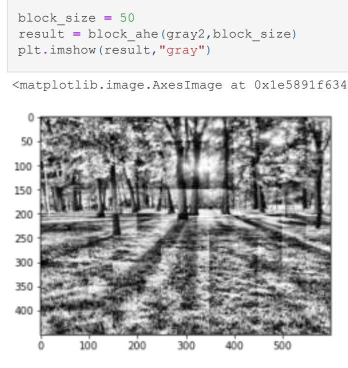
In [30]:
```
block_size = 50
result = block_ahe(gray,block_size)
plt.imshow(result,"gray")
```

Out[30]: <matplotlib.image.AxesImage at 0x1e588f70250>



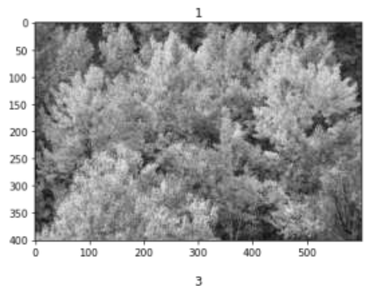
In [31]:
```
block_size = 50
result = block_ahe(gray2,block_size)
plt.imshow(result,"gray")
```

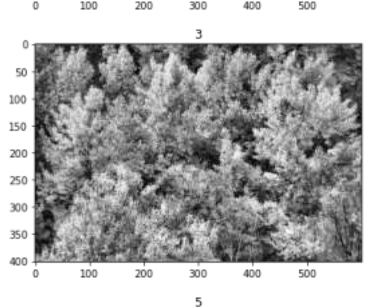
Out[31]: <matplotlib.image.AxesImage at 0x1e5891f6340>

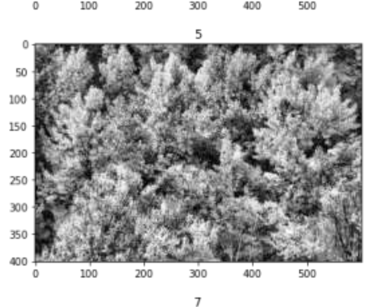


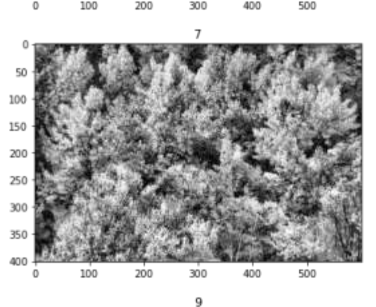
4.d)

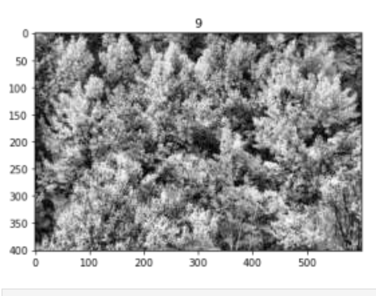
In [32]:
```
for clip_limit in range(1,10,2):
    clahe = cv2.createCLAHE(clipLimit=clip_limit)
    gray1_clahe = clahe.apply(gray)
    plt.imshow(gray1_clahe, "gray")
    plt.title(clip_limit)
    plt.show()
```



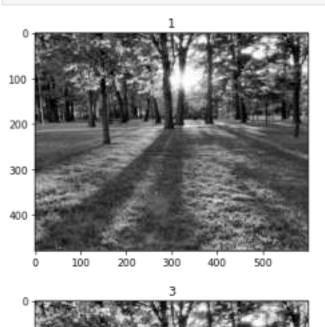


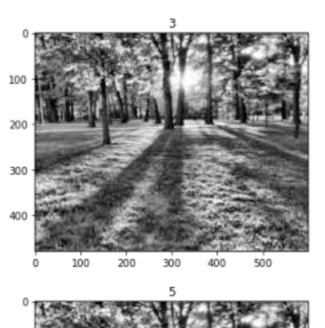


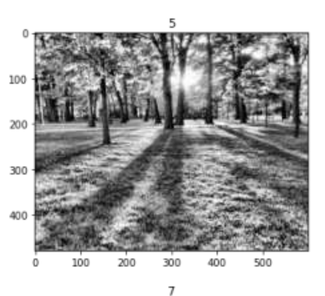


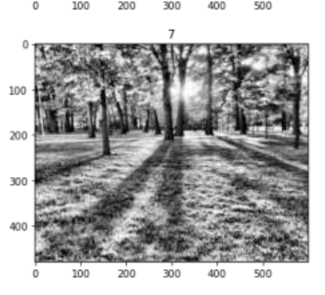


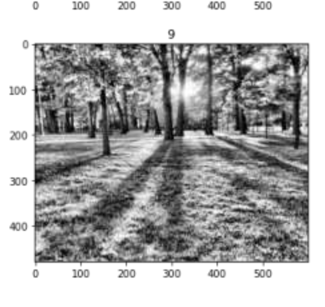
In [33]:
```python
for clip_limit in range(1,10,2):
    clahe = cv2.createCLAHE(clipLimit=clip_limit)
    gray2_clahe = clahe.apply(gray2)
    plt.imshow(gray2_clahe, "gray")
    plt.title(clip_limit)
    plt.show()
```











4.e)

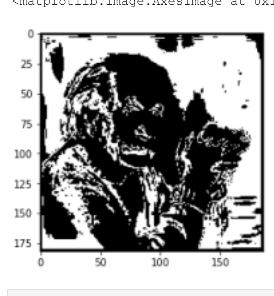
If we look at the output we get by both CLAHE and SWAHE then we can see that CLAHE performs better, SWAHE has some pixellation in the output images.

Solution 5

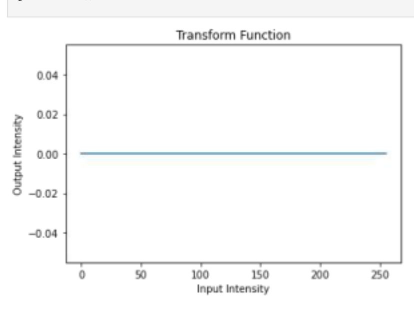
5.a)

In [34]:
```python
range_highlight = (50,100)
h,w = einstein.shape
img_highlight = np.zeros((h,w),dtype='uint8')
for i in range(h):
    for j in range(w):
        if einstein[i,j]>range_highlight[0] and einstein[i,j]<range_highlight[1]:
            img_highlight[i,j] = 255
        else:
            img_highlight[i,j] = 0
plt.imshow(img_highlight, "gray")
```

Out[34]: <matplotlib.image.AxesImage at 0x1e58a95a490>



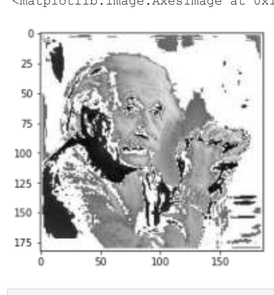
In [35]:
```python
hist, bin_edges = np.histogram(einstein.flatten(),256, [0,256])
hist_eq, bin_edges_eq = np.histogram(img_highlight.flatten(),256, [0,256])

trans_func = np.interp(hist, hist_eq, np.arange(256))
plt.plot(trans_func)
plt.title('Transform Function')
plt.xlabel('Input Intensity')
plt.ylabel('Output Intensity')
plt.show()
```



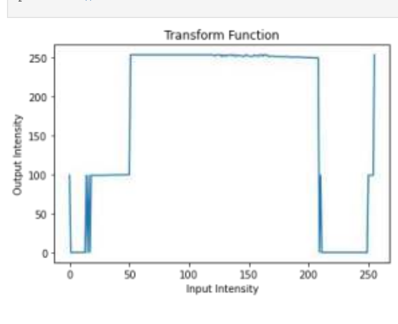
5.b)

In [36]:
```python
range_highlight = (50,100)
h,w = einstein.shape
img_highlight = np.zeros((h,w),dtype='uint8')
for i in range(h):
    for j in range(w):
        if einstein[i,j]>range_highlight[0] and einstein[i,j]<range_highlight[1]:
            img_highlight[i,j] = 255
        else:
            img_highlight[i,j] = einstein[i,j]
plt.imshow(img_highlight, "gray")
```

Out[36]: <matplotlib.image.AxesImage at 0x1e58a724c10>



In [37]:
```python
hist, bin_edges = np.histogram(einstein.flatten(),256, [0,256])
hist_eq, bin_edges_eq = np.histogram(img_highlight.flatten(),256, [0,256])

trans_func = np.interp(hist, hist_eq, np.arange(256))
plt.plot(trans_func)
plt.title('Transform Function')
plt.xlabel('Input Intensity')
plt.ylabel('Output Intensity')
plt.show()
```

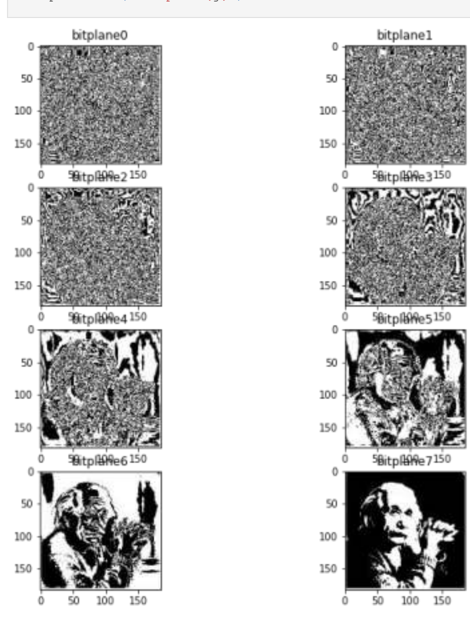


Solution 6

6.a)

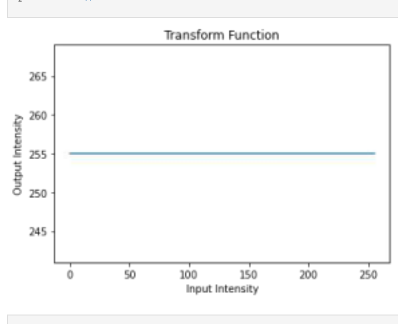
In [38]:
```python
einstein_bin = []
for i in range(h):
    for j in range(w):
        einstein_bin.append(np.binary_repr(einstein[i][j], width=8))

bitplanes = np.zeros((8,h,w), np.uint8)

for j in range(0,8):
    b = 2**j
    bitplanes[j] = (np.array([int(i[7-j]) for i in einstein_bin],dtype = np.uint8)*b).reshape(h,w)
```

In [39]:
```python
fig = plt.figure(figsize=(10, 10))
rows = 4
columns = 2
for j in range(8):
    fig.add_subplot(rows, columns, j+1)
    plt.imshow(bitplanes[j],"gray")
    plt.title(f'bitplane{j}')
```

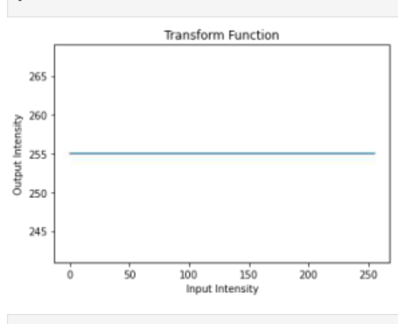


6.b)

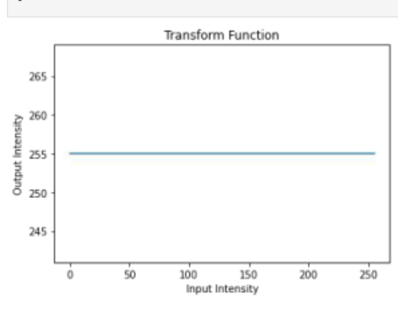
In [40]:
```python
hist, bin_edges = np.histogram(einstein.flatten(),256, [0,256])
hist_0, bin_edges_0 = np.histogram(bitplanes[0].flatten(),256, [0,256])

trans_func = np.interp(hist, hist_0, np.arange(256))
plt.plot(trans_func)
plt.title('Transform Function')
plt.xlabel('Input Intensity')
plt.ylabel('Output Intensity')
plt.show()
```



In [41]:
```python
hist, bin_edges = np.histogram(einstein.flatten(),256, [0,256])
hist_3, bin_edges_3 = np.histogram(bitplanes[3].flatten(),256, [0,256])

trans_func3 = np.interp(hist, hist_3, np.arange(256))
plt.plot(trans_func3)
plt.title('Transform Function')
plt.xlabel('Input Intensity')
plt.ylabel('Output Intensity')
plt.show()
```



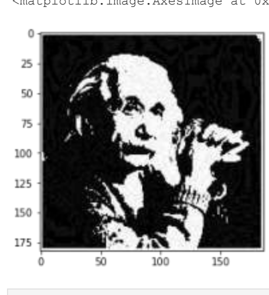
In [42]:
```python
hist, bin_edges = np.histogram(einstein.flatten(),256, [0,256])
img7 = bitplanes[7]
hist_7, bin_edges_7 = np.histogram(img7.flatten(),256, [0,256])

trans_func7 = np.interp(hist, hist_7, np.arange(256))
plt.plot(trans_func7)
plt.title('Transform Function')
plt.xlabel('Input Intensity')
plt.ylabel('Output Intensity')
plt.show()
```



6.c)

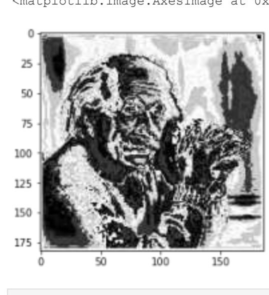
In [43]:
```python
reconstruct = bitplanes[0]+bitplanes[3]+bitplanes[7]
plt.imshow(reconstruct,'gray')
```

Out[43]: <matplotlib.image.AxesImage at 0x1e58a783070>



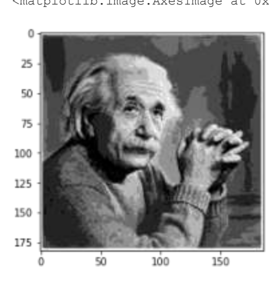
In [44]:
```python
reconstruct = bitplanes[2]+bitplanes[4]+bitplanes[6]
plt.imshow(reconstruct,'gray')
```

Out[44]: <matplotlib.image.AxesImage at 0x1e58a85dac0>



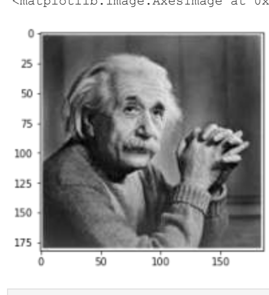
In [45]:
```python
reconstruct = bitplanes[5]+bitplanes[6]+bitplanes[7]
plt.imshow(reconstruct,'gray')
```

Out[45]: <matplotlib.image.AxesImage at 0x1e58eb27370>



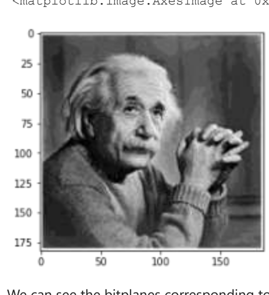
In [46]:
```python
plt.imshow(einstein, "gray")
```

Out[46]: <matplotlib.image.AxesImage at 0x1e58b8002e0>



In [47]:
```python
reconstruct =bitplanes[4]+bitplanes[5]+bitplanes[6]+bitplanes[7]
plt.imshow(reconstruct,'gray')
```

Out[47]: <matplotlib.image.AxesImage at 0x1e58b866d90>



We can see the bitplanes corresponding to more significant bits when merged give a better image. Minimum number of planes depends upon our requirements, bitplanes 5,6 and 7 also give a fairly good image. Though, bitplanes 4,5,6 and 7 together give a better image.

In [ ]: