Multimedia Forensics Exercise 2

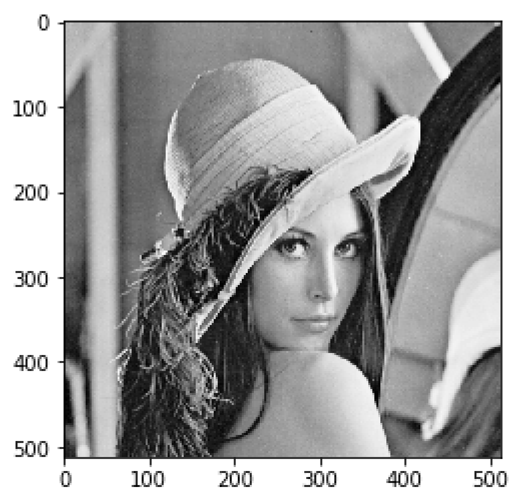
Tientso Ning

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct
%matplotlib inline
```

Step 1 and Step 2

In [2]:
```
#read the original image as grayscale
img = cv2.imread("./Lab2_JPEG/lena_512.bmp", 0)
plt.imshow(img, cmap="Greys_r") #show image to check
```

Out[2]: `<matplotlib.image.AxesImage at 0x7f6c27cca080>`



In [3]:
```
#zigzag function stolen from the internet
def zigzag (a):
    return np.concatenate([np.diagonal(a[::-1,:], i)[::(2*(i % 2)-1)] for i in
range(1-a.shape[0], a.shape[0])])
```

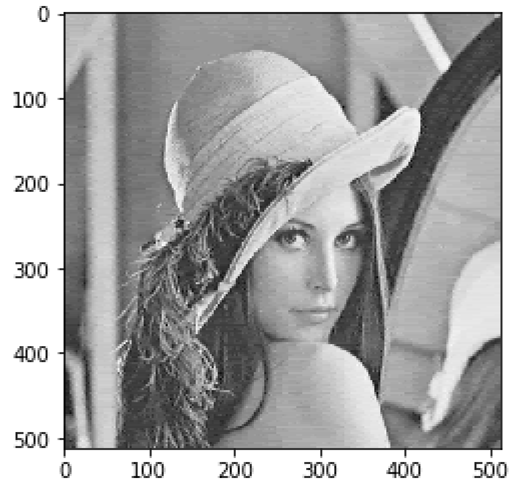
```
In [4]: #one-time compression function
def compress (image, QF1, QF2):
    '''
    compresses the image using DCT transform and Q matrix
    '''
    img = np.copy(image) #copy

    #this is the quantization matrix
    Q50 = np.array([[16,11,10,16,24,40,51,61],[12,12,14,19,26,58,60,55],[14,13
,16,24,40,57,69,56],[14,17,22,29,51,87,80,62],[18,22,37,56,68,109,103,77],[24,
35,55,64,81,104,113,92],[49,64,78,87,103,121,120,101],[72,92,95,98,112,100,103
,99]], order='F')

    #QF1
    if QF1 > 50:
        QM1 = np.around(Q50*(np.ones(Q50.shape)*((100-QF1)/50)))
        QM1.astype(int)

    elif QF1 < 50:
        QM1 = np.around(Q50*(np.ones(Q50.shape)*(50/QF1)))
        QM1.astype(int)

    elif QF1 == 50:
        QM1 = Q50

    else:
        print("error")

    QM1.astype(float)

    #QF2
    if QF2 > 50:
        QM2 = np.around(Q50*(np.ones(Q50.shape)*((100-QF2)/50)))
        QM2.astype(int)

    elif QF2 < 50:
        QM2 = np.around(Q50*(np.ones(Q50.shape)*(50/QF2)))
        QM2.astype(int)

    elif QF2 == 50:
        QM2 = Q50

    else:
        print("error")

    QM2.astype(float)

    #Set up the DCT values
    dct_domain = np.zeros(img.shape) #should just be same shape
    dct_quantized = np.zeros(img.shape)
    dct_dequantized = np.zeros(img.shape)
    dct_restored = np.zeros(img.shape)
    dct_quantized_coeff = np.zeros((64,(img.shape[0]//8)*(img.shape[1]//8)))

    dct_domain2 = np.zeros(img.shape) #for the second compression
    dct_quantized2 = np.zeros(img.shape)
```

```
    dct_quantized2_coeff = np.zeros((64,(img.shape[0]//8)*(img.shape[1]//8)))

    #subtract the img by 128 to center around the 0
    img = np.copy(img) - (128*np.ones(img.shape))

    #JPEG Encoding
    k = 0
    for i in range(0, img.shape[0], 8): #row 8x8

        for j in range(0, img.shape[1], 8):

            block = img[i:i+8,j:j+8] #set the block
            win1 = dct(block, norm='ortho')
            dct_domain[i:i+8,j:j+8] = win1

            win2 = np.around(win1/QM1)
            dct_quantized[i:i+8,j:j+8] = win2
            dct_quantized_coeff[:,k] = zigzag(win2) #to get the coeff for pair
wise
            k += 1

    #JPEG Decoding
    for i in range(0, img.shape[0], 8):

        for j in range(0, img.shape[1], 8):

            win2 = dct_quantized[i:i+8, j:j+8]
            win3 = win2*QM1 #dequantization of DCT coeff
            dct_dequantized[i:i+8, j:j+8] = win3
            win4 = idct(win3, norm='ortho')
            dct_restored[i:i+8, j:j+8] = win4

    #Set up the relevant return info
    img_recon = np.copy(dct_restored) + (128*np.ones(dct_restored.shape)) #do
n't forget to uncenter
    globalDCT = np.copy(dct_quantized)
    pairwiseDCT = np.copy(dct_quantized_coeff)

    #JPEG Encoding2
    k=0
    for i in range(0, img_recon.shape[0], 8): #row 8x8

        for j in range(0, img_recon.shape[1], 8):

            block = img_recon[i:i+8,j:j+8] #set the block
            win5 = dct(block, norm='ortho')
            dct_domain2[i:i+8,j:j+8] = win5

            win6 = np.around(win5/QM2)
            dct_quantized2[i:i+8,j:j+8] = win2
            dct_quantized2_coeff[:,k] = zigzag(win6)
            k += 1

    #Set up the relevant 2x return info
    globalDCT_double = np.copy(dct_quantized2)
    pairwiseDCT_double = np.copy(dct_quantized2_coeff)
```

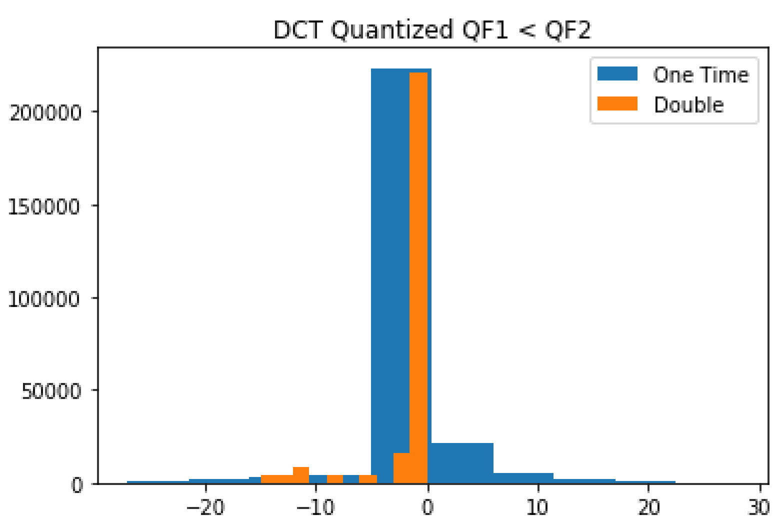
```
    return img_recon, globalDCT, pairwiseDCT, globalDCT_double, pairwiseDCT_do
uble
```

In [5]:
```
#compress the image once and see
part1, part1DCT_g, part1DCT_pw, part1DCT_g_double, part1DCT_pw_double = compre
ss(img, 60, 85) #compress with QF1=60, QF2=85
plt.imshow(part1, cmap="Greys_r") #show onetime compressed image
```

Out[5]: <matplotlib.image.AxesImage at 0x7f6c25fa1c18>



In [6]:
```
#see the global analysis of the DCT values
plt.hist(part1DCT_g.flatten(), label="One Time")
plt.hist(part1DCT_g_double.flatten(), label="Double")
plt.title("DCT Quantized QF1 < QF2")
plt.legend()
plt.show()
```

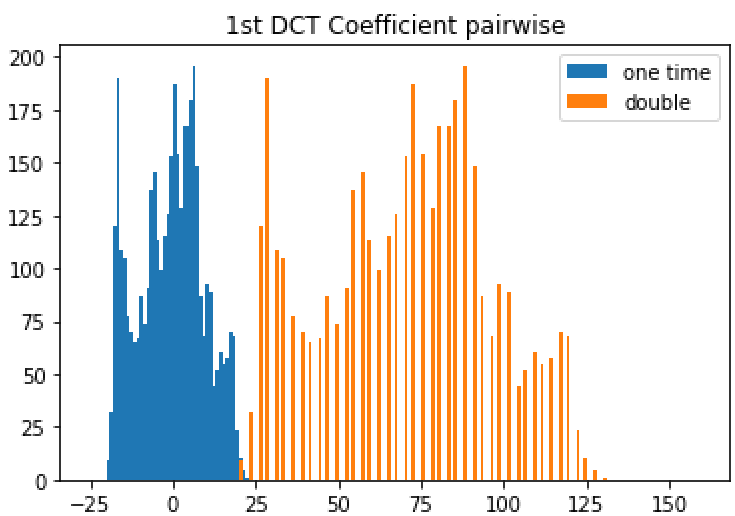


We expected to see some better results than that was shown here like the pictures we saw in class, where some of the bins are missing. The quality here is not good, but this is what we expect when we do a global analysis, so for the second part we will do pairwise comparisons.

In [7]:
```
#establish the bin sizes
for i in range(0,10):
    min_dct = min(min(part1DCT_pw[i,:]), min(part1DCT_pw_double[i,:]))
    max_dct = max(max(part1DCT_pw[i,:]), max(part1DCT_pw_double[i,:]))

    x_bin = []
    for i in range(int(min_dct),int(max_dct)+1):
        x_bin.append(i)
```

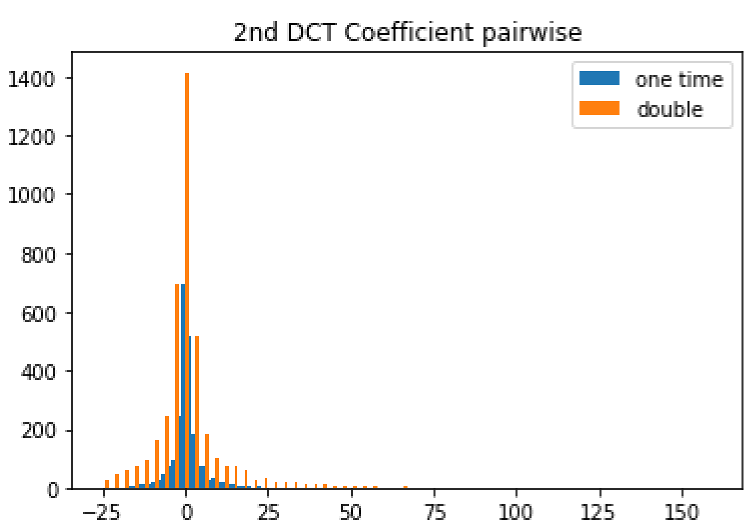
In [8]:
```
#pairwise analysis of DCT coefficients
plt.hist(part1DCT_pw[0,:], bins=x_bin, label="one time")
plt.hist(part1DCT_pw_double[0,:], bins=x_bin, label="double")
plt.legend()
plt.title("1st DCT Coefficient pairwise")
```

Out[8]:
```
Text(0.5, 1.0, '1st DCT Coefficient pairwise')
```



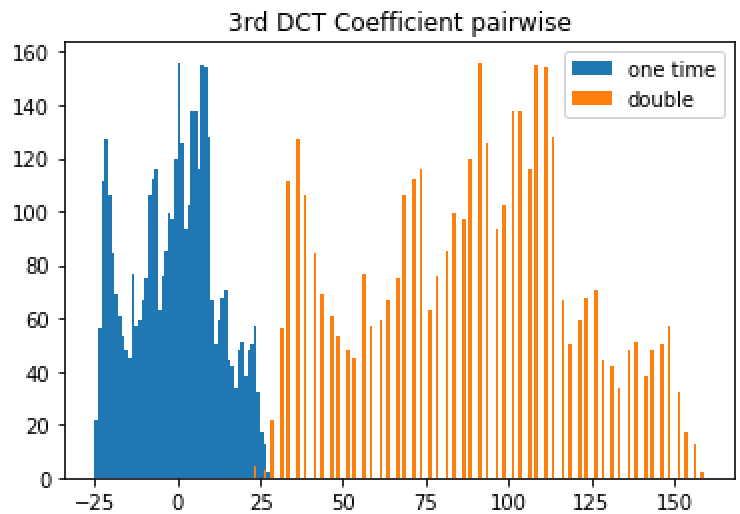
In [9]:
```
plt.hist(part1DCT_pw[1,:], bins=x_bin, label="one time")
plt.hist(part1DCT_pw_double[1,:], bins=x_bin, label="double")
plt.legend()
plt.title("2nd DCT Coefficient pairwise")
```

Out[9]:
```
Text(0.5, 1.0, '2nd DCT Coefficient pairwise')
```



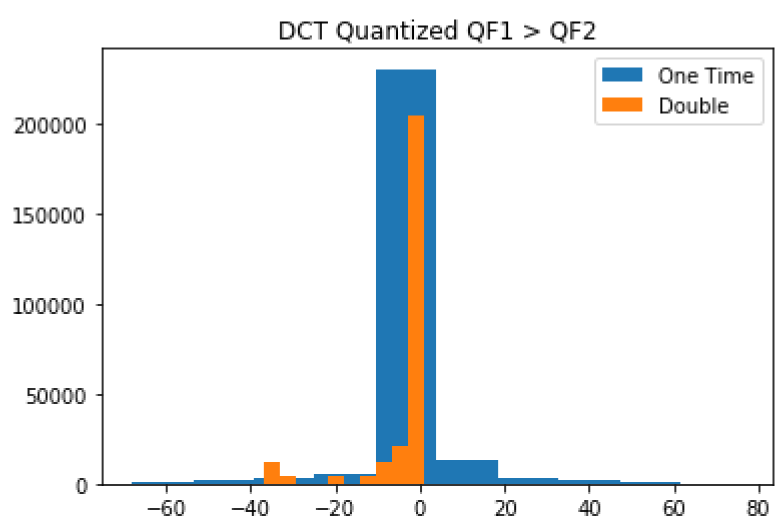
In [10]:
```python
plt.hist(part1DCT_pw[2,:], bins=x_bin, label="one time")
plt.hist(part1DCT_pw_double[2,:], bins=x_bin, label="double")
plt.legend()
plt.title("3rd DCT Coefficient pairwise")
```

Out[10]: `Text(0.5, 1.0, '3rd DCT Coefficient pairwise')`



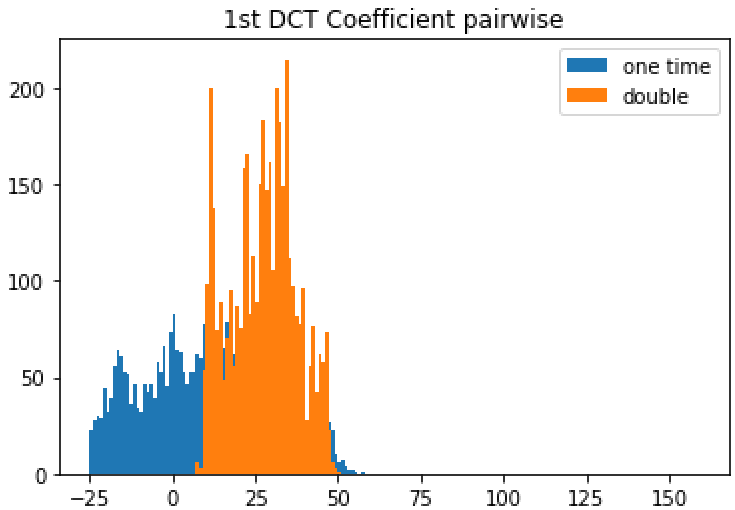
Now we can see the pairwise coefficients, and we can really see the difference. We notice the results that we expect, which is this type of lost bins in the second compression.

In [11]:
```python
#show the histogram for QF1 > QF2 Global
part1, part1DCT_g, part1DCT_pw, part1DCT_g_double, part1DCT_pw_double = compre
ss(img, 85, 60) #compress with QF2=60, QF1=85
plt.hist(part1DCT_g.flatten(), label="One Time")
plt.hist(part1DCT_g_double.flatten(), label="Double")
plt.title("DCT Quantized QF1 > QF2")
plt.legend()
plt.show()
```



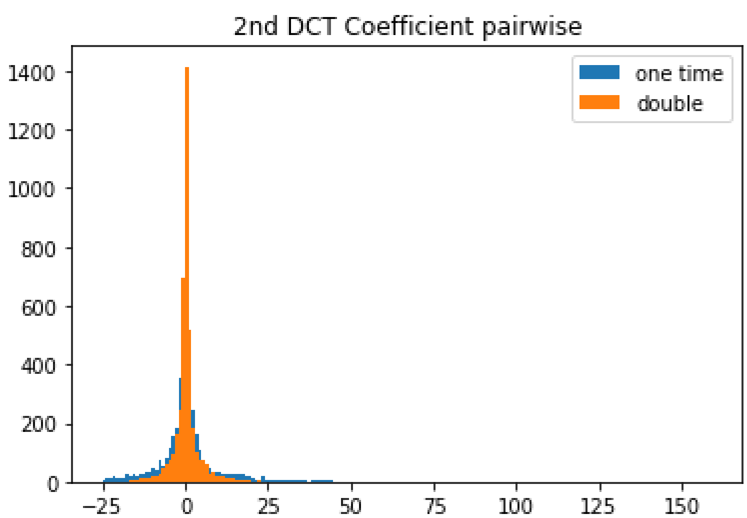
In [12]:
```python
#pairwise analysis of DCT coefficients QF1 < QF2
plt.hist(part1DCT_pw[0,:], bins=x_bin, label="one time")
plt.hist(part1DCT_pw_double[0,:], bins=x_bin, label="double")
plt.legend()
plt.title("1st DCT Coefficient pairwise")
```

Out[12]: Text(0.5, 1.0, '1st DCT Coefficient pairwise')

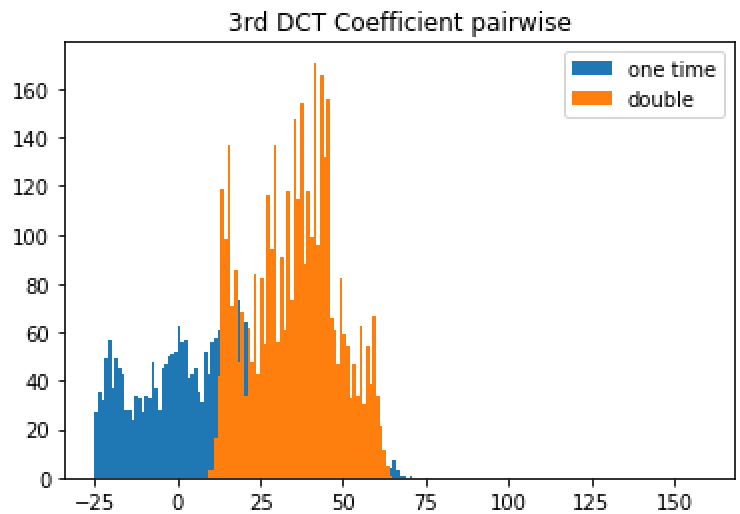


In [13]:
```python
plt.hist(part1DCT_pw[1,:], bins=x_bin, label="one time")
plt.hist(part1DCT_pw_double[1,:], bins=x_bin, label="double")
plt.legend()
plt.title("2nd DCT Coefficient pairwise")
```

Out[13]: Text(0.5, 1.0, '2nd DCT Coefficient pairwise')



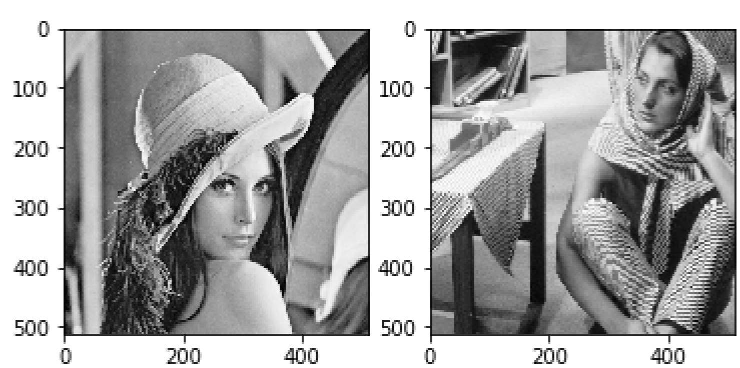
```
In [14]: plt.hist(part1DCT_pw[2,:], bins=x_bin, label="one time")
         plt.hist(part1DCT_pw_double[2,:], bins=x_bin, label="double")
         plt.legend()
         plt.title("3rd DCT Coefficient pairwise")

Out[14]: Text(0.5, 1.0, '3rd DCT Coefficient pairwise')
```



We notice that when the QF1 < QF2, the binning effect doesn't occur when we do the pairwise coefficient analysis. The global analysis is still hard to separate, but altering the QF for the second compression makes it hard to separate the images and detect.

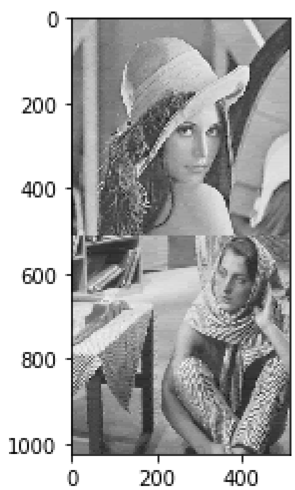
Step 3

```
In [15]: #load images and show
         img1 = cv2.imread("./updates/lena_512.bmp",0)
         img2 = cv2.imread("./updates/barbara512.bmp",0)
         f,ax = plt.subplots(1,2)
         ax[0].imshow(img1, cmap="Greys_r")
         ax[1].imshow(img2, cmap="Greys_r")

Out[15]: <matplotlib.image.AxesImage at 0x7f6c2455deb8>
```



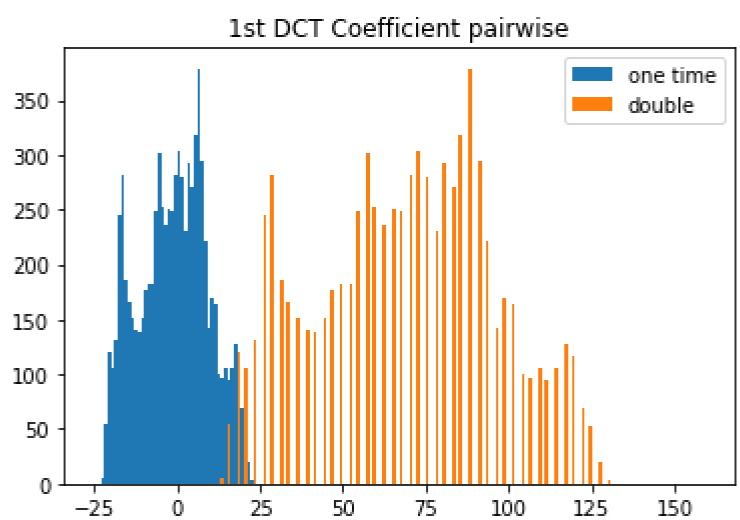
In [16]:
```
#compress the first image and concatenate with the uncompressed second image
part2, part2DCT_g, part2DCT_pw, part2DCT_g_double, part2DCT_pw_double = compre
ss(img1, 60, 85)
part2_concat = np.concatenate((part2,img2))

#compress the concatenated image to get a double compression with single compr
ession
part2, part2DCT_g, part2DCT_pw, part2DCT_g_double, part2DCT_pw_double = compre
ss(part2_concat, 60, 85)

#show the image
plt.imshow(part2, cmap="Greys_r")
```

Out[16]: <matplotlib.image.AxesImage at 0x7f6c244ba5f8>



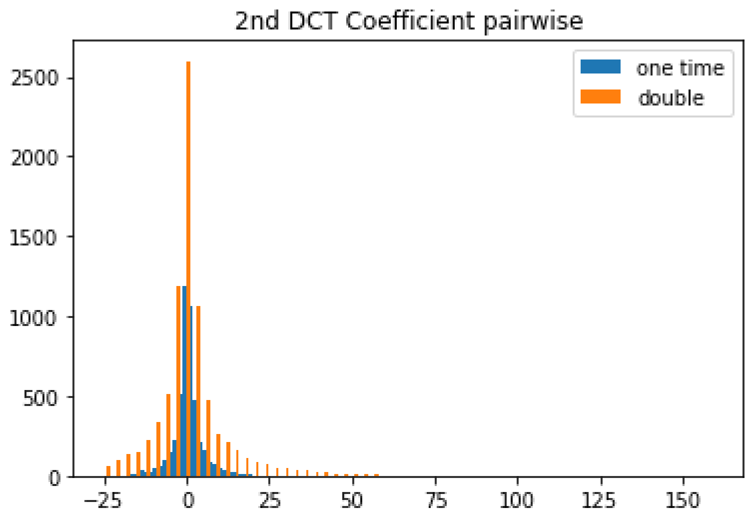
In [17]:
```
#pairwise analysis of DCT coefficients
plt.hist(part2DCT_pw[0,:], bins=x_bin, label="one time")
plt.hist(part2DCT_pw_double[0,:], bins=x_bin, label="double")
plt.legend()
plt.title("1st DCT Coefficient pairwise")
```

Out[17]: Text(0.5, 1.0, '1st DCT Coefficient pairwise')



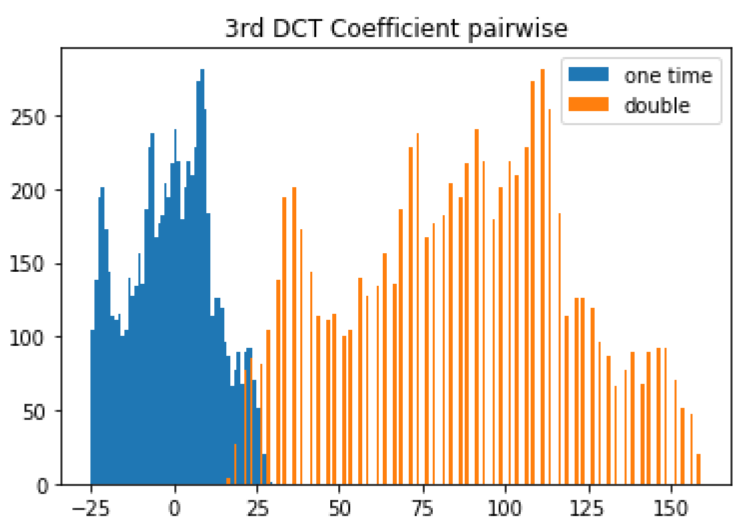
In [18]:
```
plt.hist(part2DCT_pw[1,:], bins=x_bin, label="one time")
plt.hist(part2DCT_pw_double[1,:], bins=x_bin, label="double")
plt.legend()
plt.title("2nd DCT Coefficient pairwise")
```

Out[18]: Text(0.5, 1.0, '2nd DCT Coefficient pairwise')



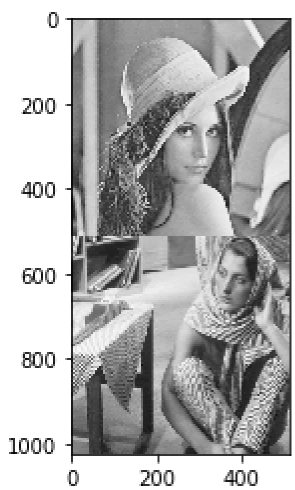
In [19]:
```
plt.hist(part2DCT_pw[2,:], bins=x_bin, label="one time")
plt.hist(part2DCT_pw_double[2,:], bins=x_bin, label="double")
plt.legend()
plt.title("3rd DCT Coefficient pairwise")
```

Out[19]: Text(0.5, 1.0, '3rd DCT Coefficient pairwise')



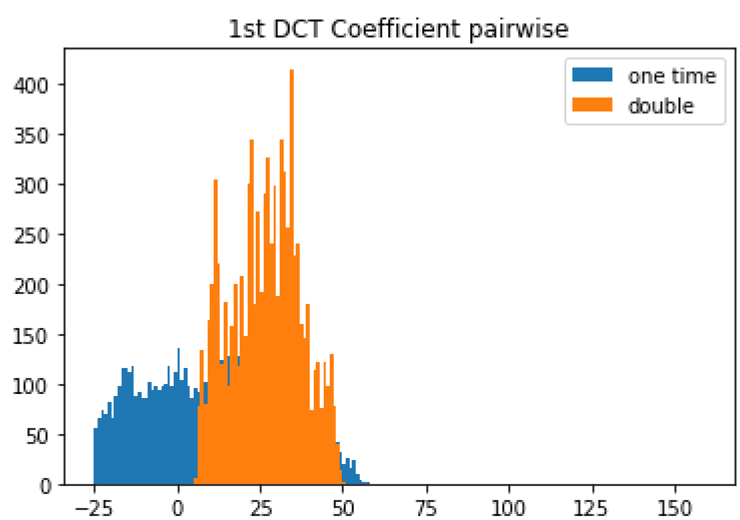
In [20]:
```
#compress the first image and concatenate with the uncompressed second image
part2, part2DCT_g, part2DCT_pw, part2DCT_g_double, part2DCT_pw_double = compre
ss(img1, 85, 60)
part2_concat = np.concatenate((part2,img2))

#compress the concatenated image to get a double compression with single compr
ession
part2, part2DCT_g, part2DCT_pw, part2DCT_g_double, part2DCT_pw_double = compre
ss(part2_concat, 85, 60)

#show the image
plt.imshow(part2, cmap="Greys_r")
```

Out[20]: <matplotlib.image.AxesImage at 0x7f6c1f7ad160>



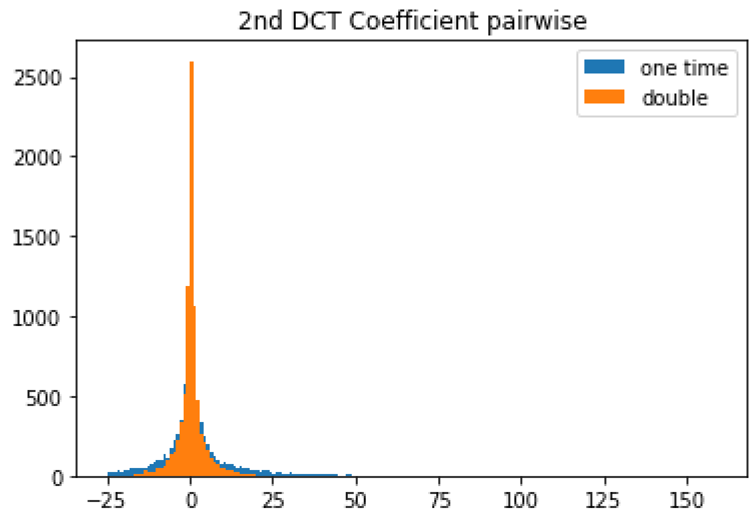
In [21]:
```
#pairwise analysis of DCT coefficients
plt.hist(part2DCT_pw[0,:], bins=x_bin, label="one time")
plt.hist(part2DCT_pw_double[0,:], bins=x_bin, label="double")
plt.legend()
plt.title("1st DCT Coefficient pairwise")
```

Out[21]: Text(0.5, 1.0, '1st DCT Coefficient pairwise')



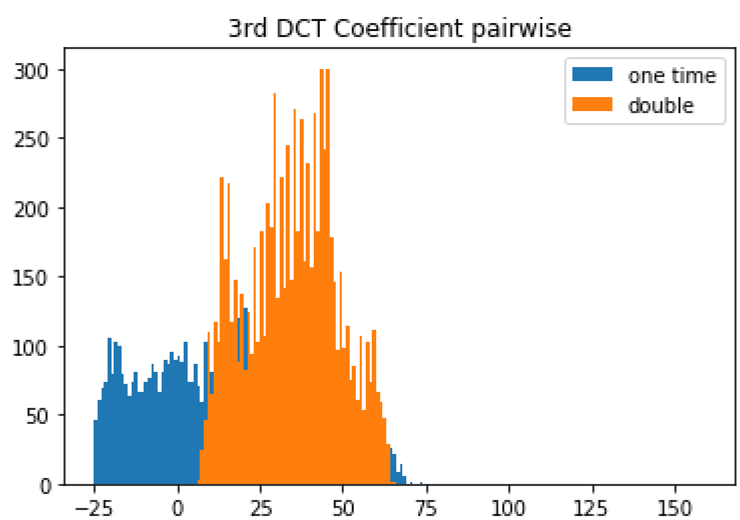
In [22]:
```
plt.hist(part2DCT_pw[1,:], bins=x_bin, label="one time")
plt.hist(part2DCT_pw_double[1,:], bins=x_bin, label="double")
plt.legend()
plt.title("2nd DCT Coefficient pairwise")
```

Out[22]: `Text(0.5, 1.0, '2nd DCT Coefficient pairwise')`



In [23]:
```
plt.hist(part2DCT_pw[2,:], bins=x_bin, label="one time")
plt.hist(part2DCT_pw_double[2,:], bins=x_bin, label="double")
plt.legend()
plt.title("3rd DCT Coefficient pairwise")
```

Out[23]: `Text(0.5, 1.0, '3rd DCT Coefficient pairwise')`



Here we can again see that we do get clear binning in the case of QF1 < QF2. We see the same results in QF1 > QF2 where the binning is not clear. From the visual image, we can only tell which section is double compressed because we concatenated the images. In the field, you would have to compare segments of a singular image that you expect a copy and paste to have occurred, making this exercise a lot more difficult. The expectation is that however, if you compare the correct regions and detect binning, you can tell that the particular section has been tampered with!