Problem 1 Image operations and vectorization (1pt)

Vector operations using numpy can offer a significant speedup over doing an operation iteratively on an image. The problem below will demonstrate the time it takes for both approaches to change the color of quadrants of an image.

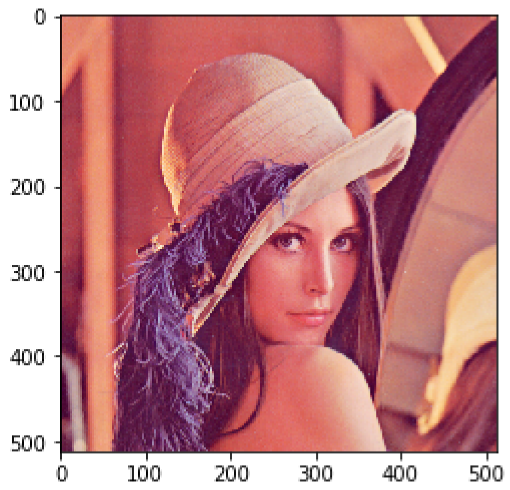
The problem reads an image "Lenna.png" that you will find in the assignment folder. Two functions are then provided as different approaches for doing an operation on the image.

Your task is to follow through the code and fill in the "piazza" function using instructions on Piazza.

In [7]:

```
import numpy as np
import matplotlib.pyplot as plt
import copy
import time

img = plt.imread('Lenna.png')              # read a JPEG image
print("Image shape", img.shape)            # print image size and color depth

plt.imshow(img)                            # displaying the original image
plt.show()
```

```
Image shape (512, 512, 3)
```



In [8]:

```
def iterative(img):

    image = copy.deepcopy(img)              # create a copy of the image matrix
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            if x < image.shape[0]/2 and y < image.shape[1]/2:
                image[x,y] = image[x,y] * [0,1,1]    #removing the red channel
            elif x > image.shape[0]/2 and y < image.shape[1]/2:
                image[x,y] = image[x,y] * [1,0,1]    #removing the green channel
            elif x < image.shape[0]/2 and y > image.shape[1]/2:
                image[x,y] = image[x,y] * [1,1,0]    #removing the blue channel
            else:
                pass
    return image

def vectorized(img):

    image = copy.deepcopy(img)
    a = int(image.shape[0]/2)
    b = int(image.shape[1]/2)
    image[:a,:b] = image[:a,:b]*[0,1,1]
    image[a:,:b] = image[a:,:b]*[1,0,1]
    image[:a,b:] = image[:a,b:]*[1,1,0]

    return image
```

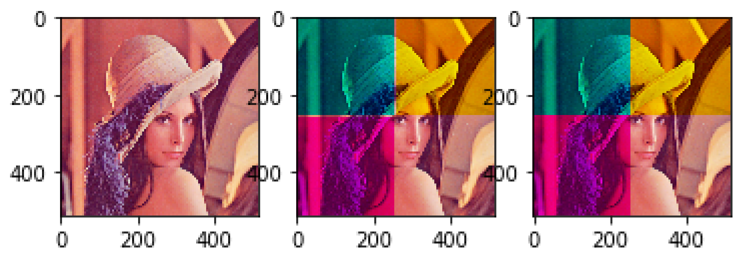
In [9]:

```
# # The code for this problem is posted on Piazza. Sign up for the course if you have not. Then find
# # the function definition included in the post 'Welcome to CSE252A' to complete this problem.
# # This is the only cell you need to edit for this problem.
def piazza():
    start = time.time()
    image_iterative = iterative(img)
    end = time.time()
    print("Iterative method took {0} seconds".format(end-start))
    start = time.time()
    image_vectorized = vectorized(img)
    end = time.time()
    print("Vectorized method took {0} seconds".format(end-start))
    return image_iterative, image_vectorized

# Run the function
image_iterative, image_vectorized = piazza()
```

```
Iterative method took 2.140733242034912 seconds
Vectorized method took 0.018352031707763672 seconds
```

In [10]:

```
# Plotting the results in sepearate subplots

plt.subplot(1, 3, 1)  # create (1x3) subplots, indexing from 1
plt.imshow(img)       # original image

plt.subplot(1, 3, 2)
plt.imshow(image_iterative)

plt.subplot(1, 3, 3)
plt.imshow(image_vectorized)

plt.show()           #displays the subplots

plt.imsave("multicolor_Lenna.png",image_vectorized)    #Saving an image
```



Submission Instructions
Remember to submit a pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via Latex. Upload to Gradescope. **NOTE**: You need to have XeTex installed on your machine to generate PDFs

Problem 2 Further Image Manipulation (7pts)

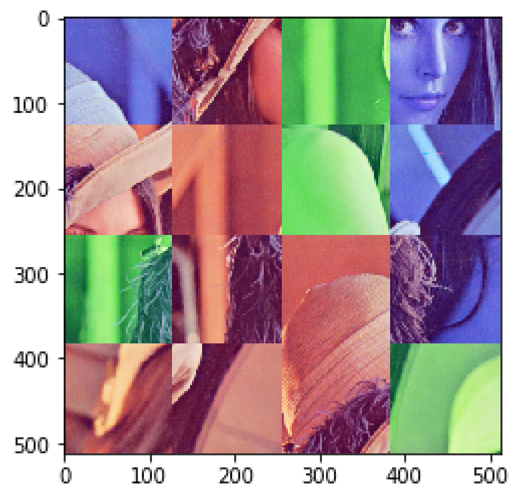
In this problem you will solve a jigsaw puzzle using the 'jigsaw.png' provided with the homework. The solution of this jigsaw is the Lenna image we used above. There are a total of 16 jigsaw pieces of size 128x128x3 which together make up the 512x512x3 image. Not only is Lemma jumbled spatially, but some of the channels in jigsaw pieces are also permuted i.e. **RGB** to **BGR** and **GRB**.

Your task is to put all the pieces to their respective locations and correct the channel permutations. To achieve this task, you are required to complete the three helper functions that will be used to solve this puzzle. You are **NOT** allowed to use any function other than the three provided here. Also, the code needs to be *vectorised* i.e. you are **NOT** allowed to use *for* loops to achieve this task.

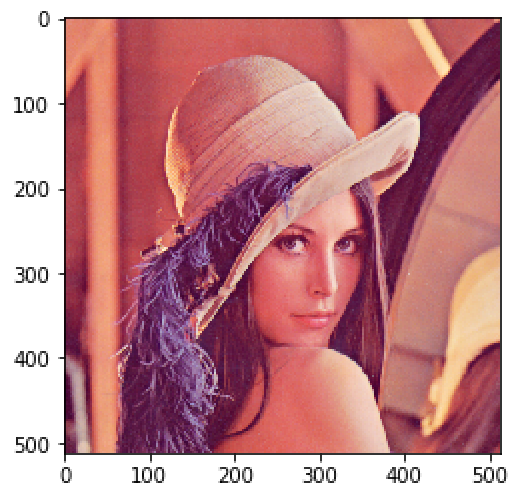
In [66]:

```
def getTile(jigsaw, tile_idx):

    '''
    This function returns a particular jigsaw piece

    jigsaw     : 512x512x3 np.ndarray
    tile_idx : tuple containing the (i,j) location of the piece

    piece    : 128x128x3 np.ndarray
    '''
    assert isinstance(tile_idx,tuple), 'tile index must be a tuple'
    assert len(tile_idx) == 2, 'tile index must specify the row and column index of the jigsaw'


    # Write your code here
    piece = np.zeros((128,128,3))   # modify piece
    piece = jigsaw[tile_idx[0]*128: (tile_idx[0]+1)*128, tile_idx[1]*128: (tile_idx[1]+1)*128,:3]

    return piece.copy()

def permuteChannels(tile, permutation):

    '''
    This function performs a permutation on channel

    tile          : 128x128x3 np.ndarray
    permutation   : tuple containing (i,j,k) channel indices
    tile_permuted : 128x128x3 np.ndarray
    '''

    assert tile.shape == (128,128,3), 'tile size should be 128x128x3'
    assert isinstance(permutation, tuple), 'permutation should be a tuple'
    assert len(permutation) == 3, 'There are only 3 channels'

    #Write your code here
    tile[:,:,[permutation[0],permutation[1],permutation[2]]]=tile[:,:,[0,1,2]]
    tile_permuted = tile.copy()

    return tile_permuted.copy()

def putTile(board, tile, tile_idx):

    '''
    This function put a jigsaw piece at a particular location on the board

    board : 512x512x3 np.ndarray
    tile : 128x128x3 np.ndarray
    tile_idx : tuple containing the (i,j) location of the piece
    img : 512x512x3 np.ndarray
    '''

    assert board.shape == (512,512,3), 'canvas size should be 512x512x3'
    assert tile.shape == (128,128,3), 'tile size should be 128x128x3'
    assert isinstance(tile_idx,tuple), 'tile index must be a tuple'
    assert len(tile_idx) == 2, 'tile index must specify the row and column index of the jigsaw'

    # Write your own code here
    img = board.copy()   # modify img
    img[tile_idx[0]*128:(1+tile_idx[0])*128, tile_idx[1]*128:(1+tile_idx[1])*128, 0:] = tile

    return img

TILE_SIZE = 128
source = [(0,0),(0,1),(0,2),(0,3),
          (1,0),(1,1),(1,2),(1,3),
          (2,0),(2,1),(2,2),(2,3),
          (3,0),(3,1),(3,2),(3,3)]

# Fill in the target list with the corresponding piece locations
target = [(0,2),(2,1),(1,0),(2,2),
          (1,2),(0,0),(3,2),(0,3),
          (2,0),(3,0),(0,1),(3,1),
          (3,3),(1,3),(1,1),(2,3)]

#Fill in the respective channel permutations
channelPermutation = [(2,1,0),(0,1,2),(1,0,2),(2,1,0),
                      (0,1,2),(0,1,2),(1,0,2),(2,1,0),
                      (1,0,2),(0,1,2),(0,1,2),(2,1,0),
                      (0,1,2),(0,1,2),(0,1,2),(1,0,2)]

jigsaw = plt.imread('jigsaw.png')
board = np.ones(jigsaw.shape)

for i in range(16):
    tile = getTile(jigsaw, source[i])
    tile = permuteChannels(tile, channelPermutation[i])
    board = putTile(board, tile, target[i])

print("Jigsaw Puzzle")
plt.imshow(jigsaw)
plt.show()
print("Solution")
plt.imshow(board)
plt.show()
```

```
Jigsaw Puzzle
```



```
Solution
```



Submission Instructions
Remember to submit a pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via Latex. Upload to Gradescope. **NOTE**: You need to have XeTex installed on your machine to generate PDFs