Arnoldas Kurbanovas
CSI 526 Project #2

Reporting of encryption/decryption times using full round AES average of 10 runs:

Data file	Size (in pixels)	Size (in KB)	Encryption time using (a) (ms)	Decryption time using (a) (ms)	Encryption time using (b) (ms)	Decryption time using (b) (ms)	Encryption time using (c) (ms)	Decryption time using (c) (ms)
Image1	800x600	1370	1169	573	1157	531	1185	588
Image2	640x480	301	170	99	170	100	173	121
Image3	528x292	19.4	15	7	15	7	15	7

Reporting of encryption/decryption times using round-reduced AES average of 10 runs:

Data file	Size (in pixels)	Size (in KB)	Encryption time using (a) (ms)	Decryption time using (a) (ms)	Encryption time using (b) (ms)	Decryption time using (b) (ms)	Encryption time using (c) (ms)	Decryption time using (c) (ms)
Image1	800x600	1370	416	219	414	219	758	241
Image2	640x480	301	59	41	60	41	61	44
Image3	528x292	19.4	4	3	5	3	6	3

Observations about tables:

For both the full round AES and round-reduced AES run averages the time in ms increased using the pixel by pixel method of reading in the image compared to the professors byte by byte method. The time for every case also increases if you preserve the header of the images which logically makes sense concluding that my findings are correct!

My analysis, observations, findings and work for this assignment:
Read in image content pixel by pixel.
Code:

```
File input_image_fileNew = new File(file_path);
//adding code to read in image pixel by pixel
    BufferedImage image = ImageIO.read(input_image_fileNew);

    int width = image.getWidth();
    int hight = image.getHeight();
```

Store my red green blue and alpha pixel values into an integer pixel 3 dimensional matrix/array having 4 bytes per pixel
Code:

```
//pixel matrix of RGBa
    int[][][] rgbaOut = new int[hight][width][4];

    System.out.println("rgbaOut: "+rgbaOut);

    for(int x = 0; x<image.getWidth(); x++){

          for(int y =0; y<image.getHeight(); y++){
                //color object to split the values of red green blue and alpha
             Color color = new Color(image.getRGB(x, y), true);
             rgbaOut[y][x][0] = color.getRed();
             //System.out.println("rgbaOut[y][x][0] red: " + rgbaOut[y][x][0]);
             rgbaOut[y][x][1]= color.getGreen();
             //System.out.println("rgbaOut[y][x][1] green: " + rgbaOut[y][x][1]);
             rgbaOut[y][x][2] = color.getBlue();
             //System.out.println("rgbaOut[y][x][2] blue: " + rgbaOut[y][x][2]);
             rgbaOut[y][x][3]= color.getAlpha();
             //System.out.println("rgbaOut[y][x][3] alpha: " + rgbaOut[y][x][3]);
          }
    }
```

I convert that 3 dimensional matrix into a 1 dimensional integer array by using a triple nested for loop
Code:

```
//convert from a 3d array to a 1d array
    int[] singleArr = new int[rgbaOut.length*rgbaOut.length*rgbaOut.length];
    //flattening down my 3d array to a 1d array
    int a = 0;
```

```
    for(int i =0; i< rgbaOut.length; i++){
        for(int j=0; j< rgbaOut.length; j++){
            for(int k=0; k<4; k++){
                singleArr[a]= rgbaOut[i][j][k];
                //System.out.println("singleArr: "+singleArr[a]);
                a++;
            }
        }
    }
```

I then converted my new 1 dimensional integer array into a byte array having block size of 4 pixels

Code:

```
//method i created to convert from a 1D single int array to a single byte array=
   public static byte[] intToByteArr(int[] intArr){
        final ByteBuffer buffer = ByteBuffer.allocate(intArr.length *
4).order(ByteOrder.LITTLE_ENDIAN);
        buffer.asIntBuffer().put(intArr);
        return buffer.array();
   }
byte[] input_data;
    input_data = intToByteArr(singleArr);

    //checking byte vals
    for(int z=0; z<input_data.length ; z++){
        //System.out.println("byte input_data values: "+ input_data[z]);
    }
```