Homework 1 Questions

Instructions

- Compile and read through the included MATLAB tutorial.
- 2 questions.
- Include code.
- Feel free to include images or equations.
- Please make this document anonymous.
- **Please use only the space provided and keep the page breaks.** Please do not make new pages, nor remove pages. The document is a template to help grading.
- If you really need extra space, please use new pages at the end of the document and refer us to it in your answers.

Submission

- Please zip your folder with **hw1_student id_name.zip** (ex: hw1_20191234_Peter.zip)
- Submit your homework to KLMS.
- An assignment after its original due date will be degraded from the marked credit per day: e.g., A will be downgraded to B for one-day delayed submission.

Questions

Q1: We wish to set all pixels that have a brightness of 10 or less to 0, to remove sensor noise. However, our code is slow when run on a database with 1000 grayscale images.

Image: grizzlypeakg.png

```
A = imread('grizzlypeakg.png');
[m1,n1] = size( A );
for i=1:m1
    for j=1:n1
        if A(i,j) <= 10
            A(i,j) = 0;
        end
    end
end
```

Q1.1: How could we speed it up?

A1.1: We can speed up this sensor noise removal process by creating a binary logical array, B, instead of running a double loop of A. By running B = A $\leq$ 10, B's elements will either be 0 or 1, depending on whether the pixel at corresponding indices of A is less than 10. Then, we can run A(B) = 0 to switch all pixels lower than 10 to 0.

```
A = imread('grizzlypeakg.png');
B = A <= 10;
A(B) = 0;
imshow(A)
```

Q1.2: What factor speedup would we receive over 1000 images? Please measure it.

Ignore file loading; assume all images are equal resolution; don't assume that the time taken for one image $\times 1000$ will equal 1000 image computations, as single short tasks on multitasking computers often take variable time.

A1.2: In order to measure time it takes for the previous and revised versions of sensor removal, I ran two .m files, each containing previous and revised version, 1000 times using a for loop in a separate .m file.
As a result, the original method of using a double loop took 55.479265 seconds, and the revised method using a binary logical array took 41.968613 seconds. Thus, the factor speedup is $55.479265/41.968613 = \boxed{1.3219228}$.

Running original algorithm 1000 times:

```
tic;
for i1 = 1:1000
    Q1_1_original
end
toc;
```

Running revised algorithm 1000 times:

```
tic;
for i1 = 1:1000
    Q1_1_revised
end
toc;
```

Q1.3: How might a speeded-up version change for color images? Please measure it.

Image: grizzlypeak.jpg

A1.3: When run single time, the original method took 8.268699 seconds, and the revised method took 0.107157 second. The gap between the two execution times using colored picture is much greater than the one using previous picture - with the speedup factor of about 77.164338. This might be so because the original method loops around a lot more pixels consisting of red, green, and blue componenets, whereas the revised method finds pixels less than 10 using a logical binary array without the need to loop through the picture's pixels.

Original algorithm:

```
tic;
A = imread('grizzlypeak.jpg');
[m1, n1] = size(A);
for i = 1: m1
    for j = 1 : n1
        if A(i, j) <= 10
            A(i, j) = 0;
        end
    end
end
toc;
imshow(A)
```

Revised algorithm:

```
tic;
A = imread('grizzlypeak.jpg');
B = A <= 10;
A(B) = 0;
toc;
imshow(A)
```

Q2: We wish to reduce the brightness of an image but, when trying to visualize the result, all we sees is white with some weird "corruption" of color patches.

Image: gigi.jpg

```
I = double( imread('gigi.jpg') );
I = I - 20;
imshow( I );
```

Q2.1: What is incorrect with this approach? How can it be fixed while maintaining the same amount of brightness reduction?

A2.1:
1) In double form, all numbers that exceed 1.0 are considered as 1. Therefore, all pixels that exceed 1 have the same brightness - white. We can fix this problem by normalizing pixels to lie between 0 and 1.0. I did so by changing double to im2double.
2) To reduce the brightness, the original code subtracts 20 from all elements of I. However, the elements are now normalized to lie between 0 and 1.0. Therefore, to reduce the brightness by same amount, we need to reduce pixels by 20/255, not 20.

```
I = im2double( imread('gigi.jpg') );
I = I - 20/255;
imshow( I );
```

Q2.2: Where did the original corruption come from? Which specific values in the original image did it represent?

A2.2: When run with original method, the result showed a white picture with some yellowish patches - corruption. When the image is in double form, it considers all pixels higher than 1.0 as 1 - the maximum brightness - and prints them as a white pixel. Thus, yellowish patches represent pixels that are lower than 1.0.

Fixed image: fixedgigi.jpg