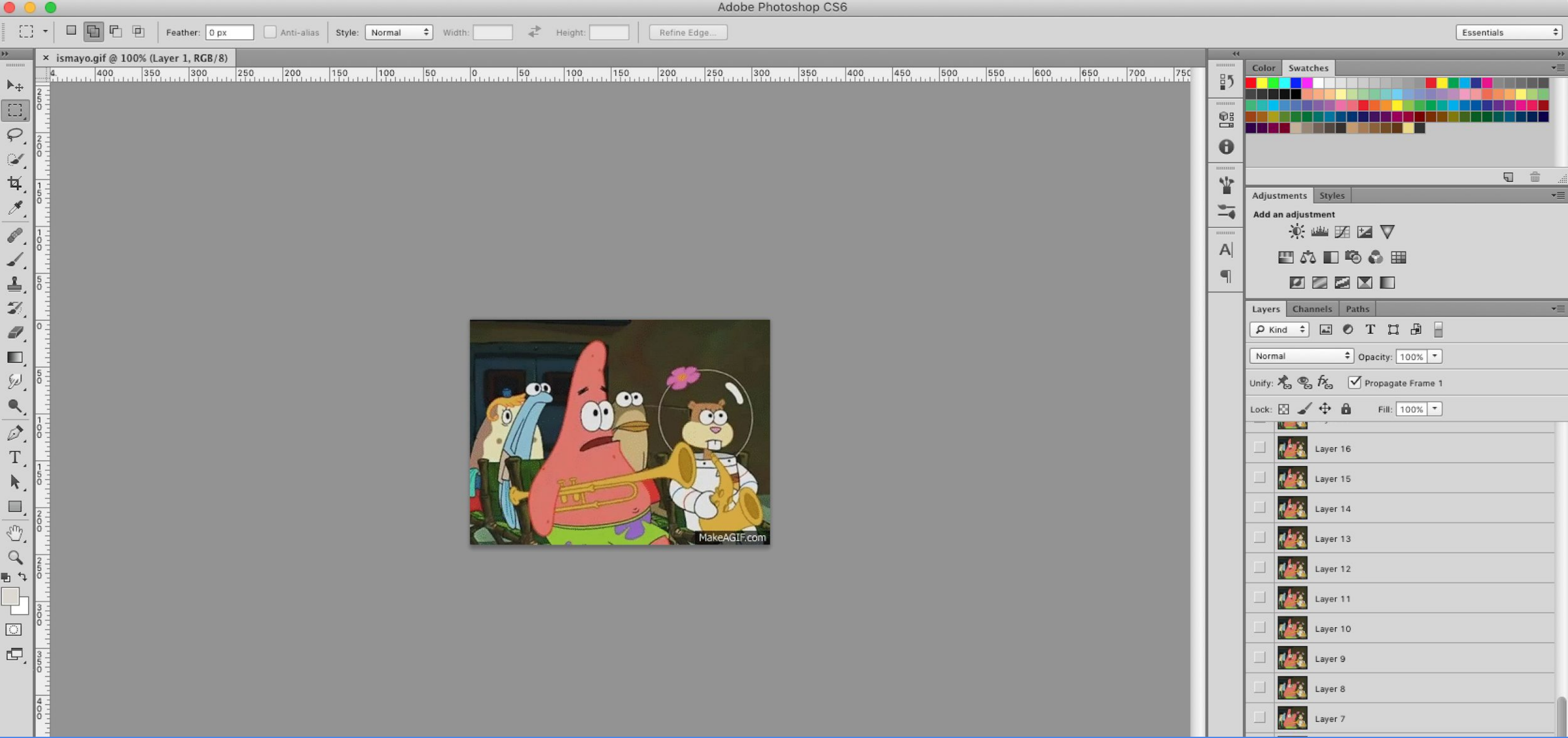
YOUR EARLY ASSIGNMENT HELP | ASSIGNMENT FIVE

YEAH! Hours: ImageShop

Garrick Fernandez <gfaerr@stanford.edu>
Robbie Jones <rmjones@stanford.edu>

Due Mon. Feb. 26th at 11:00am PST

Questions or feedback? tinyurl.com/yeah-hours



ismayo.gif @ 100% (Layer 1, RGB/8)

Feather: 0 px

Style: Normal

Width:

Height:

Normal

Opacity: 100%

☑ Propagate Frame 1

Fill: 100%

Layer 16

Layer 15

Layer 14

Layer 13

Layer 12

Layer 11

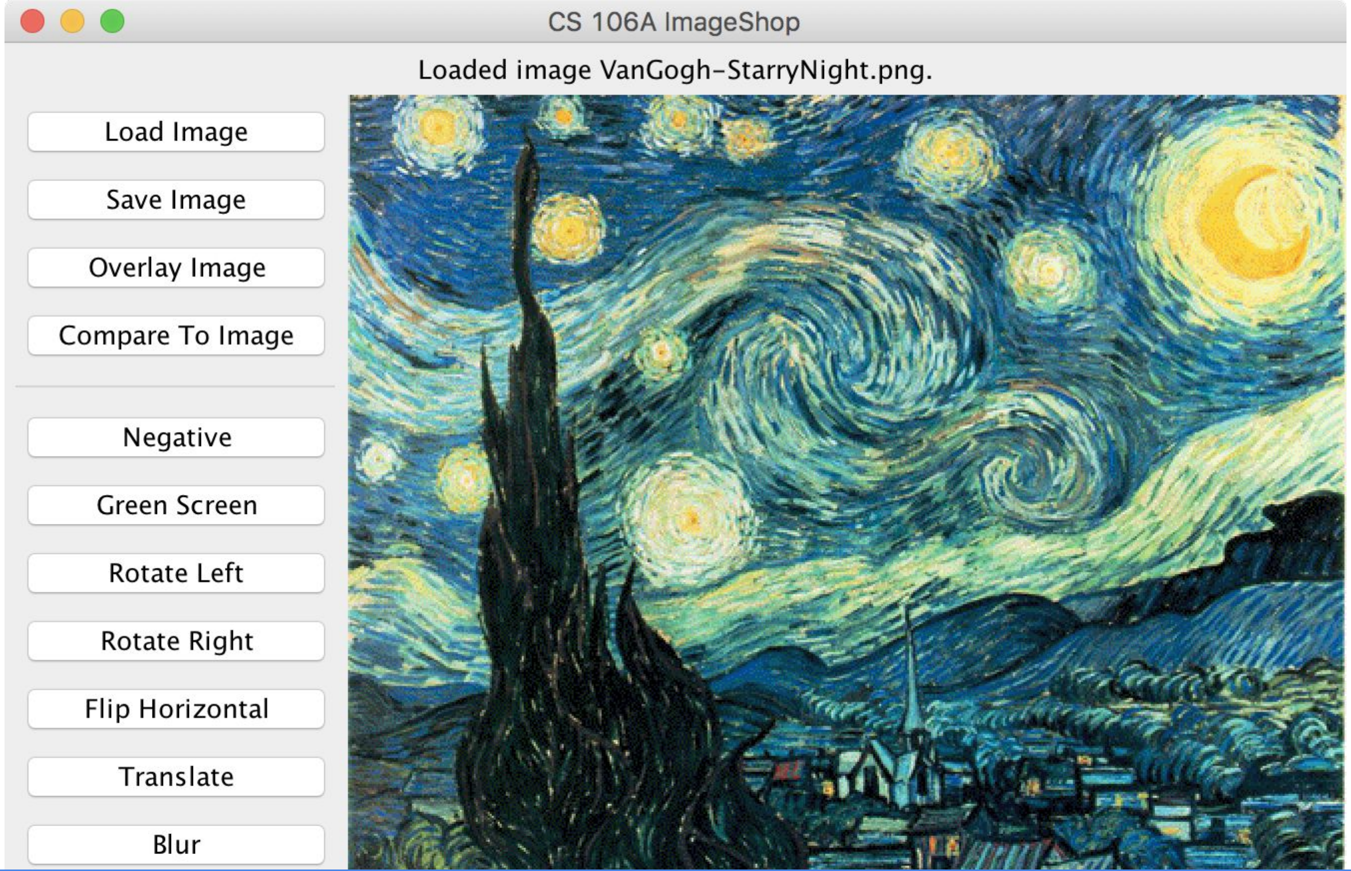
Layer 10

Layer 9

Layer 8

Layer 7

tinyurl.com/yeah-hours



CS 106A ImageShop

Loaded image VanGogh-StarryNight.png.

tinyurl.com/yeah-hours

Why ImageShop, why?

An introduction to bigger programs! Think about the given interface and how your code communicates with it.

Boost familiarity with `GImage` and its methods.

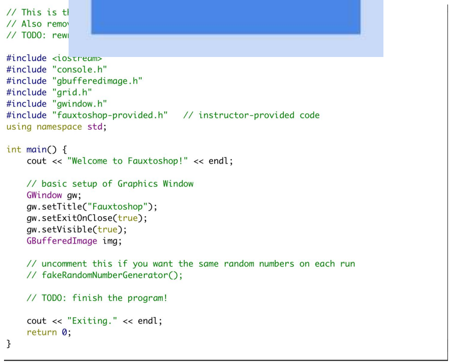
Practice using 2D arrays and manipulating them.

Translating between the pixels in a `GImage` and a 2D array of integers!

Skills For Approaching Problems

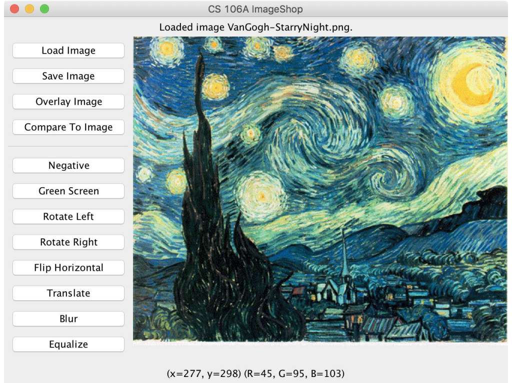
- ☆ Quickly **intuit** what a problem is asking for.
- ✍ Learn how to **draft and design** good code.
- {} Pull bits and snippets from our **coding toolbox**.
- ❐ Anticipate **edge cases** and **test** for errors.

A Primer: ImageShop ✍



```
// This is t
// Also remo
// TODO: rew

#include <iostream>
#include "console.h"
#include "gbufferedimage.h"
#include "grid.h"
#include "gwindow.h"
#include "fauxtoshop-provided.h"   // instructor-provided code
using namespace std;

int main() {
    cout << "Welcome to Fauxtoshop!" << endl;

    // basic setup of Graphics Window
    GWindow gw;
    gw.setTitle("Fauxtoshop");
    gw.setExitOnClose(true);
    gw.setVisible(true);
    GBufferedImage img;

    // uncomment this if you want the same random numbers on each run
    // fakeRandomNumberGenerator();

    // TODO: finish the program!

    cout << "Exiting." << endl;
    return 0;
}
```

their code in
ImageShopProgram.java



CS 106A ImageShop

Loaded image VanGogh–StarryNight.png.

(x=277, y=298) (R=45, G=95, B=103)

our code (!) in
ImageShopAlgorithms.java

How do these two interact?

tinyurl.com/yeah-hours

A Primer: ImageShop

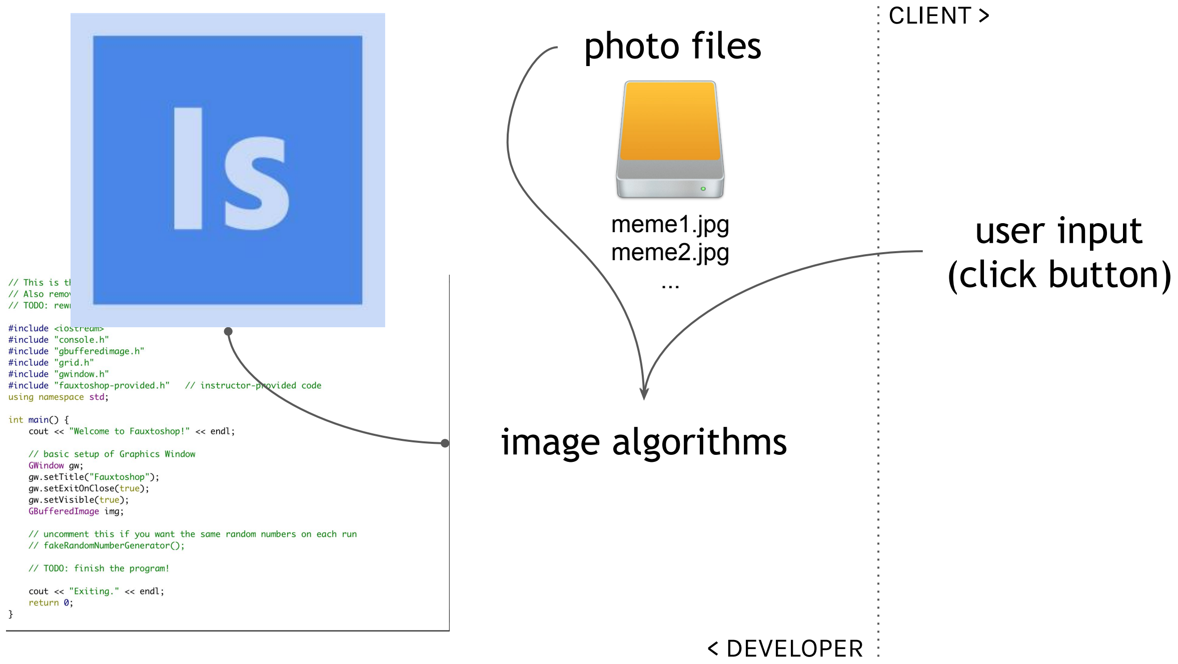


photo files

meme1.jpg
meme2.jpg
...

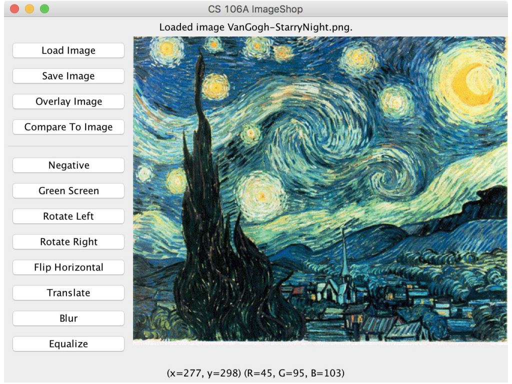
image algorithms

CLIENT >

user input
(click button)

< DEVELOPER

```
// This is th
// Also remo
// TODO: rewr

#include <iostream>
#include "console.h"
#include "gbufferedimage.h"
#include "grid.h"
#include "gwindow.h"
#include "fauxtoshop-provided.h"   // instructor-provided code
using namespace std;

int main() {
    cout << "Welcome to Fauxtoshop!" << endl;

    // basic setup of Graphics Window
    GWindow gw;
    gw.setTitle("Fauxtoshop");
    gw.setExitOnClose(true);
    gw.setVisible(true);
    GBufferedImage img;

    // uncomment this if you want the same random numbers on each run
    // fakeRandomNumberGenerator();

    // TODO: finish the program!

    cout << "Exiting." << endl;
    return 0;
}
```

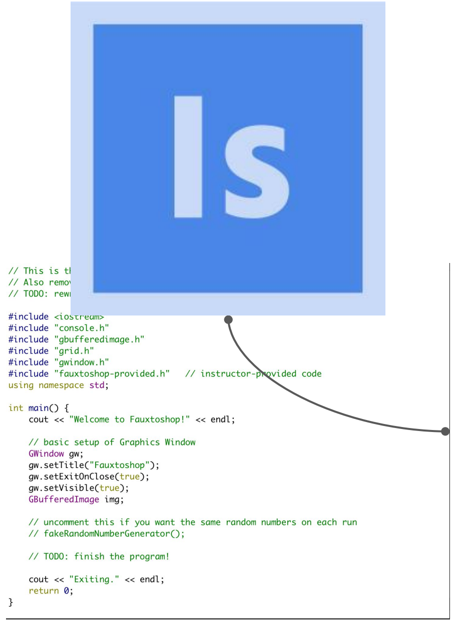


CS 106A ImageShop

Loaded image VanGogh-StarryNight.png.

(x=277, y=298) (R=45, G=95, B=103)

A Primer: ImageShop ✍



```
// This is th
// Also remo
// TODO: rew

#include <iostream>
#include "console.h"
#include "gbufferedimage.h"
#include "grid.h"
#include "gwindow.h"
#include "fauxtoshop-provided.h"   // instructor-provided code
using namespace std;

int main() {
    cout << "Welcome to Fauxtoshop!" << endl;

    // basic setup of Graphics Window
    GWindow gw;
    gw.setTitle("Fauxtoshop");
    gw.setExitOnClose(true);
    gw.setVisible(true);
    GBufferedImage img;

    // uncomment this if you want the same random numbers on each run
    // fakeRandomNumberGenerator();

    // TODO: finish the program!

    cout << "Exiting." << endl;
    return 0;
}
```

photo files

meme1.jpg
meme2.jpg
...

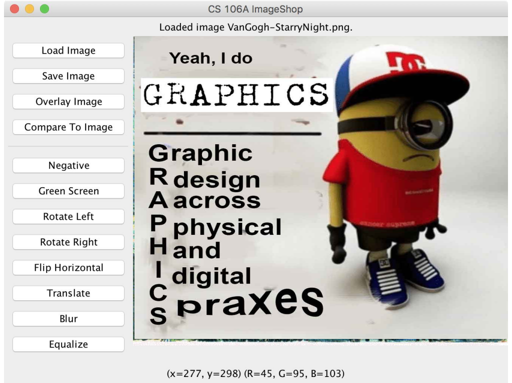
CLIENT >

user input
(click button)

image algorithms

display output

< DEVELOPER

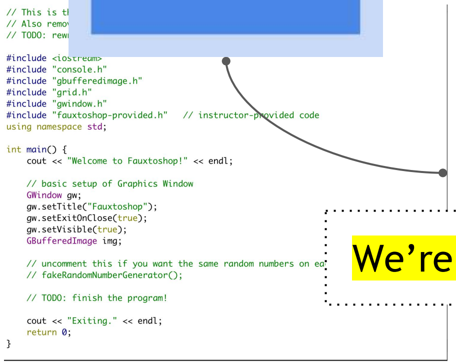


Loaded image VanGogh-StarryNight.png.

(x=277, y=298) (R=45, G=95, B=103)

tinyurl.com/yeah-hours

A Primer: ImageShop



```
// This is th
// Also remov
// TODO: rewr

#include <iostream>
#include "console.h"
#include "gbufferedimage.h"
#include "grid.h"
#include "gwindow.h"
#include "fauxtoshop-provided.h"   // instructor-provided code
using namespace std;

int main() {
    cout << "Welcome to Fauxtoshop!" << endl;

    // basic setup of Graphics Window
    GWindow gw;
    gw.setTitle("Fauxtoshop");
    gw.setExitOnClose(true);
    gw.setVisible(true);
    GBufferedImage img;

    // uncomment this if you want the same random numbers on ea
    // fakeRandomNumberGenerator();

    // TODO: finish the program!

    cout << "Exiting." << endl;
    return 0;
}
```

photo files

meme1.jpg
meme2.jpg
…

CLIENT >

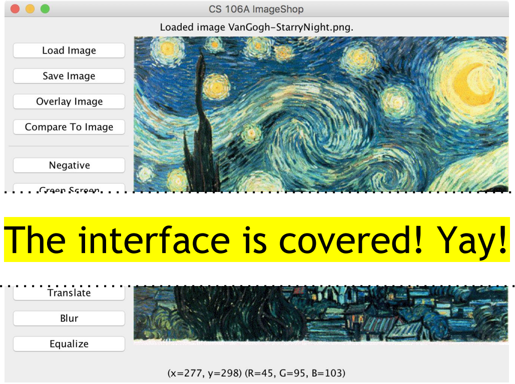
user input
(click button)

image algorithms

We’re in charge of the algorithms!

display output

< DEVELOPER



Loaded image VanGogh-StarryNight.png.

(x=277, y=298) (R=45, G=95, B=103)

The interface is covered! Yay!

tinyurl.com/yeah-hours

Breaking Up the Problem

How do we break up the problem into approachable milestones?

✍

Breaking Up the Problem

First, let’s figure out how to operate on 2D arrays and figure out what that means for our image.

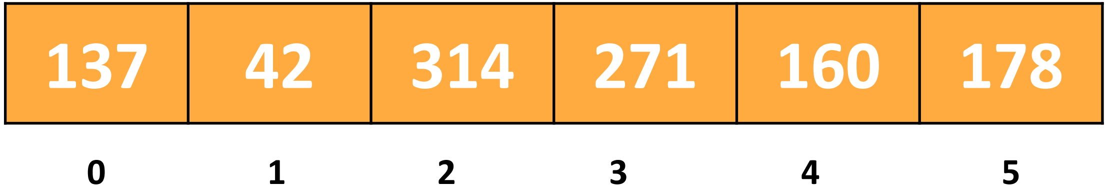
Then, let’s understand how `GImages` work and how to use them and convert between them and 2D arrays.

Last, let’s fill out our algorithms one by one!

1D and 2D Arrays

Slides courtesy of Nick Troccoli. Thanks Nick :)

Arrays: the basics



137	42	314	271	160	178
0	1	2	3	4	5

- An array stores an **ordered sequence** of multiple objects.
  - Can access objects by index using [].
- All stored objects have the same type, which you choose.
- Can store *any* type, even primitive types (**`int`**, **`boolean`**, **`double`**, etc.).
- Fixed size; cannot grow or shrink once created.

Arrays: how to make

137	42	314	271	160	178
0	1	2	3	4	5

- To create a new array, specify the type of the array and the size:

```
Type[] arr = new Type[size]
```

ex:

```
int[] numList = new int[6];
```

- All elements are initialized to the type's *default value* (0 for numeric types, `false` for `booleans`, `null` for objects).

Arrays: accessing

137	42	314	271	160	178
0	1	2	3	4	5

- To access an element of the array, use the square brackets to choose the index:

```
arr[index]
```

ex. (using the above array):

```
int num = numList[2]; // 314
```

Arrays: accessing

137	42	314	271	160	178
0	1	2	3	4	5

- To loop over every element in an array, just use a `for` loop:

```
for (int i = 0; i < arr.length; i++) {
    Type elem = arr[i];
    // use elem...
}
```

Arrays: accessing

137	42	314	271	160	178
0	1	2	3	4	5

- To loop over every element in an array, just use a `for` loop:

```
// prints out every element in numList
for (int i = 0; i < numList.length; i++) {
    int elem = numList[i];
    println(elem);
}
```

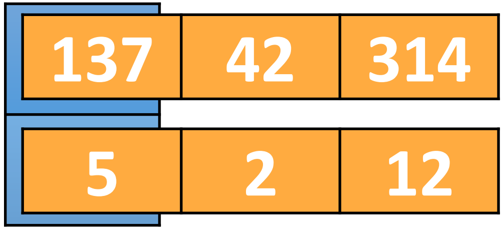
Arrays: persistence

- Like with objects, if an array is passed as a parameter to a method and that method changes its elements, those changes *will* persist when that method is done.

```
private void doSomething(int[] nums) {
   nums[0] = 2;
}
// elsewhere...
int[] arr = new int[5];
doSomething(arr); // now arr[0] = 2!
```

2D Arrays are grids!

- You can think of a 2D array as a **grid**. ***arr***[***row***][***col***] selects the element in the grid at position (***row***, ***col***).



137	42	314
5	2	12

numGrid

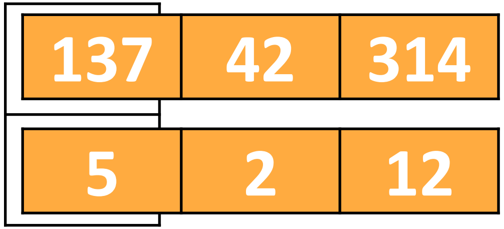
```
int num = numGrid[1][2]; // 12
```

- You can also think of a 2D array as an array of arrays. ***arr***[***row***] selects the 1D array consisting of the columns in row ***row***.

```
int[] secondRow = numGrid[1];
int num = secondRow[2]; // 12
```

2D Arrays

- To create a new 2D array, specify the type using *two* pairs of brackets, and specify both the width and height:



137	42	314
5	2	12

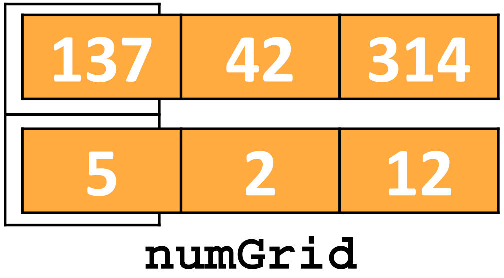
```
Type[][] arr = new Type[rows][cols];
```

ex:

```
int[][] numGrid = new int[2][3];
```

2D Arrays

- The height of a 2D array is the length of the array itself. The width is the length of any of the *element arrays*.



ex.

```
int gridHeight = numGrid.length; // 2
int gridWidth = numGrid[0].length; // 3
```

2D Arrays

- You can iterate over a 2D array just like with a 1D array, but you need 2 loops instead of 1.

Ex. What does the grid below look like?

```
int[][] arr = new int[4][5];
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[0].length; col++) {
        arr[row][col] = row + col;
    }
}
```

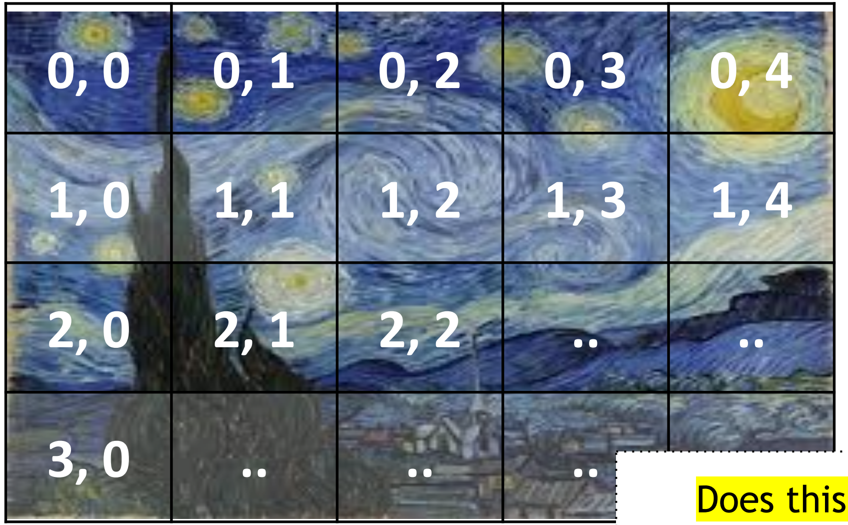
```
int[][] arr = new int[4][5];
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[0].length; col++) {
        arr[row][col] = row + col;
    }
}
```

Careful not to confuse rows and cols!

	0	1	2	3	4
0					
1					
2					
3					

Does this look like anything?

```
int[][] arr = new int[4][5];
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[0].length; col++) {
        arr[row][col] = row + col;
    }
}
```



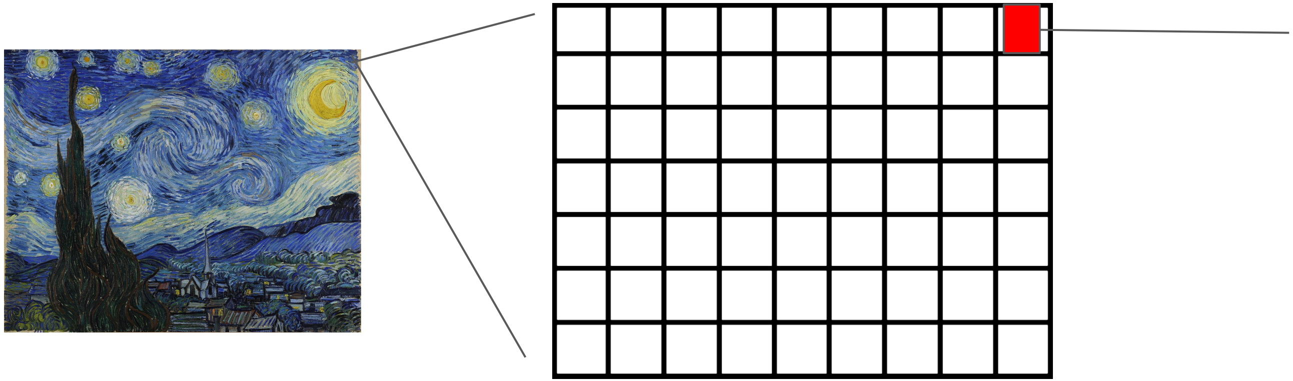
	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	0, 4
1	1, 0	1, 1	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	..	..
3	3, 0	..	..	..	

Does this look like anything?

GImages are 2D arrays of pixels!

Think of each pixel as an int representing the RGB value at that pixel

Thousands of pixels/ints comprise a single image!



#ff0000 means:

(255, 0, 0)

Red: 255

Green: 0

Blue: 0

How to use a pixel

A pixel contains all three integers (red, green, blue) as a single int. Sometimes, we need to be able to extract the individual values from it!

```
// Extract the red, green, and blue values individually
int red = GImage.getRed(pixel);
int green = GImage.getGreen(pixel);
int blue = GImage.getBlue(pixel);
red = 0; // Get rid of the red, for example
// Create a new pixel
int pixel = GImage.createRGBPixel(red, green, blue);
```

Notice that these methods are called on the GImage class *itself*, not on an object of the class!

Breaking Up the Problem

First, let’s figure out how to operate on 2D arrays and figure out what that means for our image.

Then, let’s understand how `GImages` work and how to use them and convert between them and 2D arrays.

Last, let’s fill out our algorithms one by one!

From GImages to 2D arrays

If GImages are 2D arrays of pixels/ints, how do we convert from one into the other?

There are some helpful methods to convert between the two!

```
// Turn a GImage into a 2D array of pixels
int[][] pixels = img.getPixelArray();

// Create a new img given a 2D array of pixels
GImage img = new GImage(pixels);
```

Breaking Up the Problem

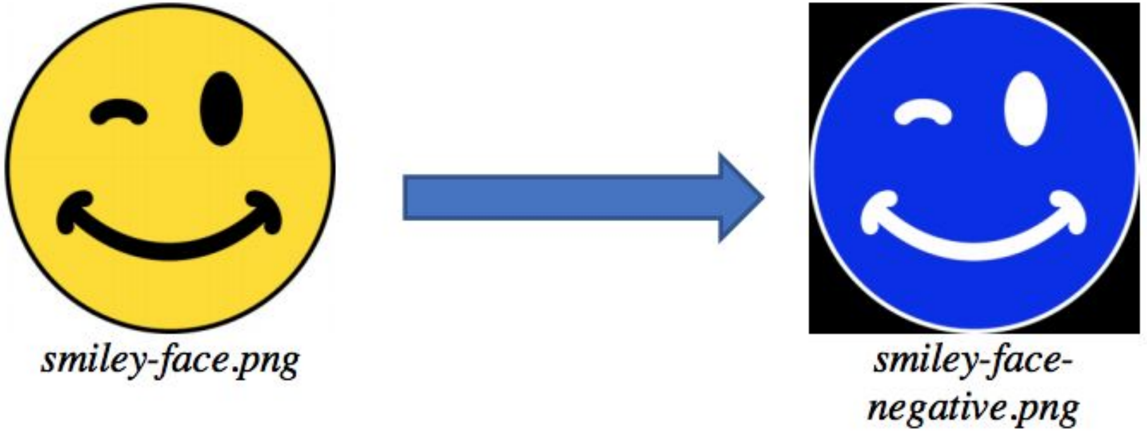
First, let’s figure out how to operate on 2D arrays and figure out what that means for our image.

Then, let’s understand how `GImages` work and how to use them and convert between them and 2D arrays.

Last, let’s fill out our algorithms one by one!

Negative

K → 255 - K



Negative

The basic ideas used here will be prevalent throughout the assignment

- Looping through entire image
- Grabbing (and possibly manipulating) pixel at each location
- Returning **new** image

Green Screen

Lights, Camera, Green Screen!

- Main idea: make all green pixels transparent (alpha = 0)
- Problem: not all pixels are **exactly** green
- Solution: treat a pixel as green if its green color component is at *least 2x as large as the maximum of its red and blue components.*

Lights, Camera, Green Screen!

Helpful methods (again!) from the book / Java documentation:

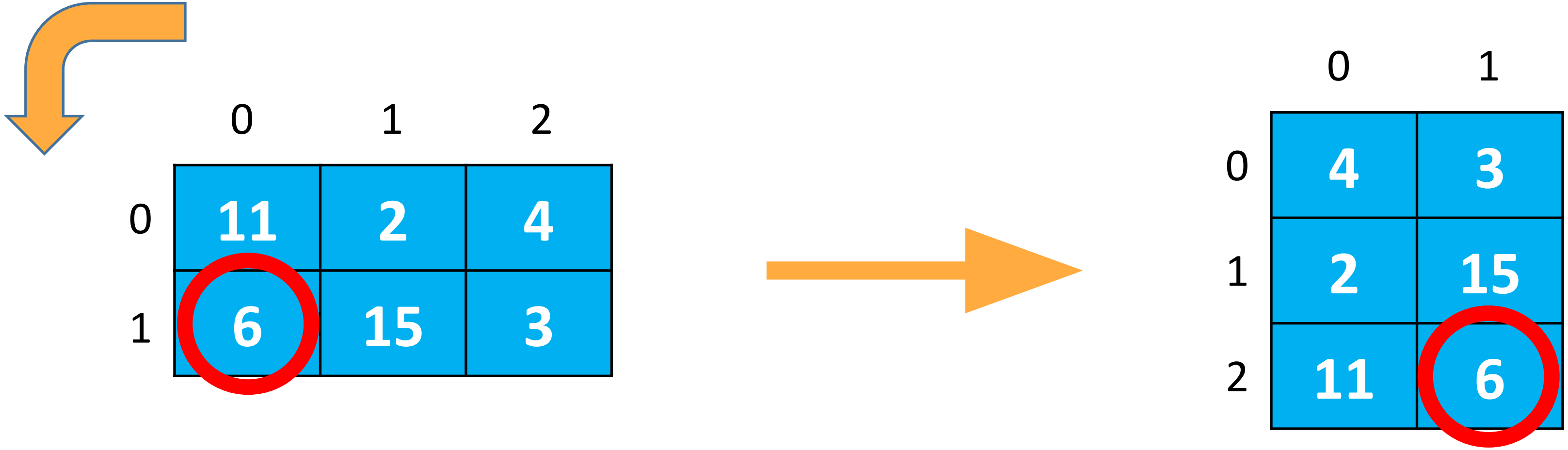
```
int red = GImage.getRed(PIXEL);
int green = GImage.getGreen(PIXEL);
int blue = GImage.getBlue(PIXEL);
int pixel = GImage.createRGBPixel(RED, GREEN, BLUE, ALPHA);
int maxNum = Math.max(NUM1, NUM2);
```

Flip Horizontal, Rotate Left, and Rotate Right

Flip Horizontal, Rotate Left, Rotate Right

- Tip: draw out pictures of pixel movement

Ex. Rotate Left:



	0	1	2
0	11	2	4
1	6	15	3

	0	1
0	4	3
1	2	15
2	11	6

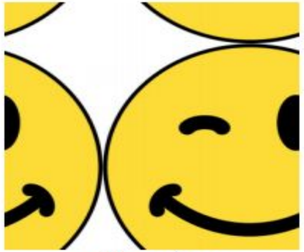
Notice: pixel’s **column #** in new image is its **row #** in original image.

How do we calculate the new **row #**?

(Lost in) Translation

smiley-face.png

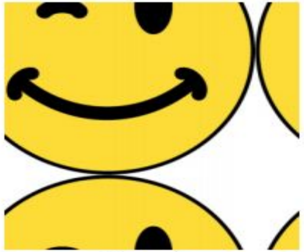
dx=100, dy=50



smiley-face-translate-100-50.png

smiley-face.png

dx=-45, dy=-91



smiley-face-translate-negative-45-91.png

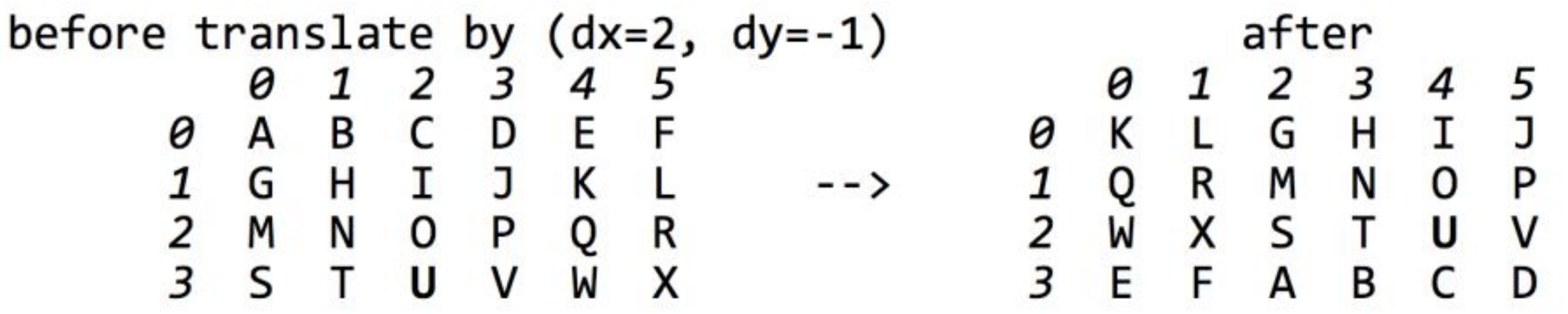
(Lost in) Translation

Big idea: pixels need to “roll over” by a certain (dx, dy) to a new location

Problem: pixels can’t go off the edge of the image

Question: how do we make pixels wrap around to the other side of the image?

Simpler Example



```
before translate by (dx=2, dy=-1)                after
        0  1  2  3  4  5                   0  1  2  3  4  5
     0  A  B  C  D  E  F                0  K  L  G  H  I  J
     1  G  H  I  J  K  L       -->      1  Q  R  M  N  O  P
     2  M  N  O  P  Q  R                2  W  X  S  T  U  V
     3  S  T  U  V  W  X                3  E  F  A  B  C  D
```

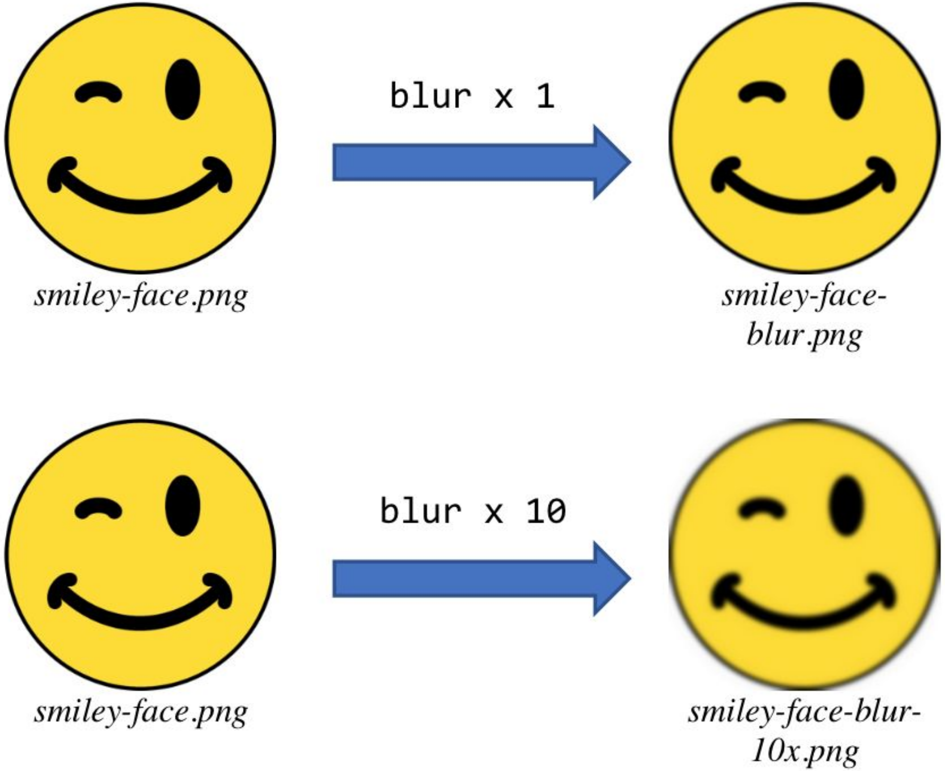
Notice the wraparound!

6 → 0

7 → 1

NOTE: dx can be greater than width of image! (and similar for dy)

There are other really cool types of blurs, including Gaussian blur!



smiley-face.png

blur x 1

smiley-face-blur.png

smiley-face.png

blur x 10

smiley-face-blur-10x.png

Blur

For every pixel p at position (r,c), we take the average of all the RGB values over the nine positions from r-1 to r+1 and c-1 to c+1.

Can we imagine the **pseudocode**?

Blur

For every pixel p at position (r,c), we take the average of all the RGB values over the nine positions from r-1 to r+1 and c-1 to c+1.

Can we imagine the **pseudocode**?

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by # neighbors)
```

Blur

For every pixel p at position (r,c), we take the average of all the RGB values over the nine positions from r-1 to r+1 and c-1 to c+1.

Can we imagine the **pseudocode**?

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by # neighbors)
```

What’s this gonna look like?

Images, Arrays, And Double-`For`s

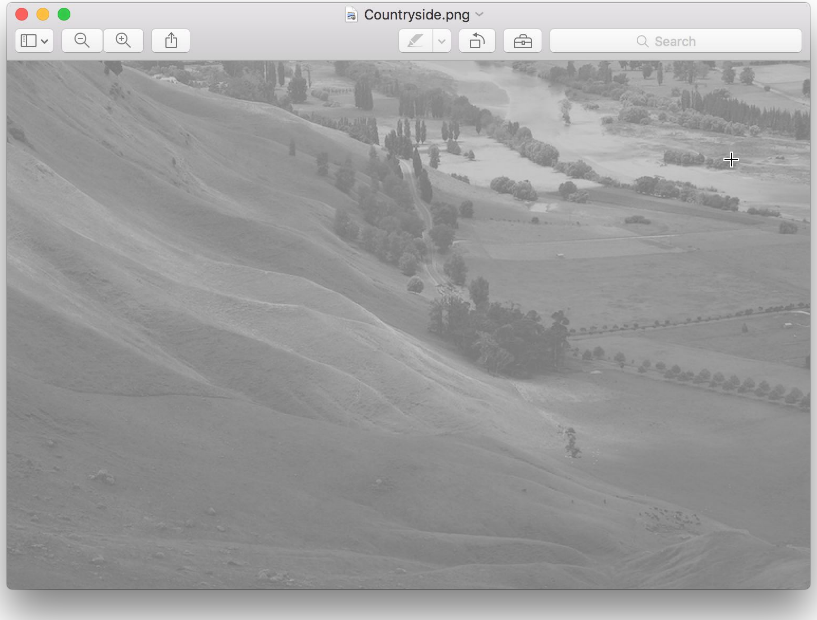
Convert our image to an `int[][]`, and use a double `for` loop to go over each pixel.

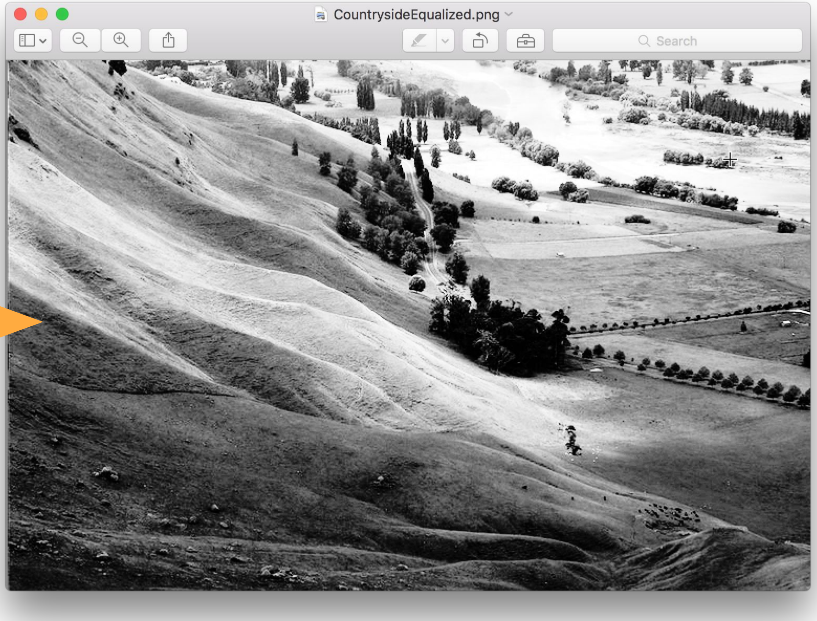
```
// converting a GImage to an array of ints
int[][] original = img.getPixelArray();
```

```
// some helpful array bits...you could also check the docs!
int rows = original.length;
int cols = original[0].length;
// If a row number is in bounds, it will be 0 to rows-1...
```

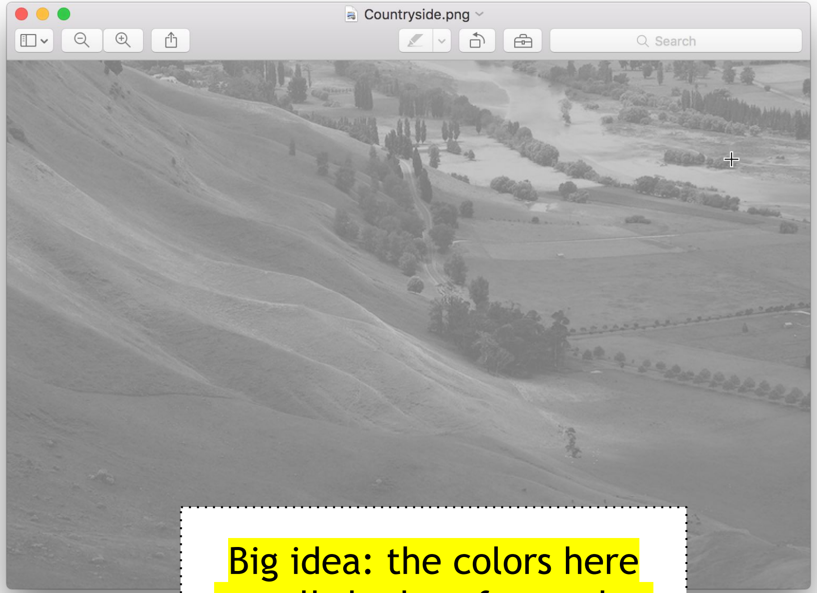
Equalize

Histogram Equalization

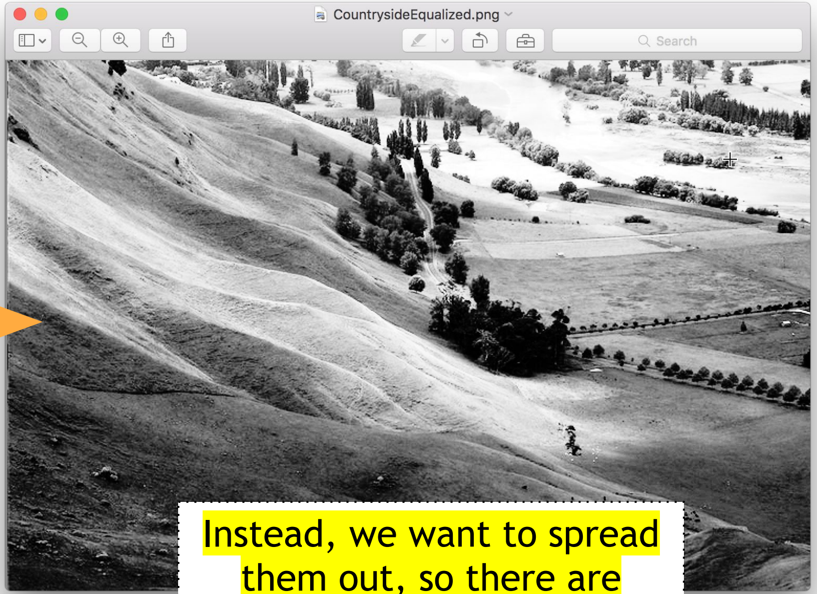




Histogram Equalization



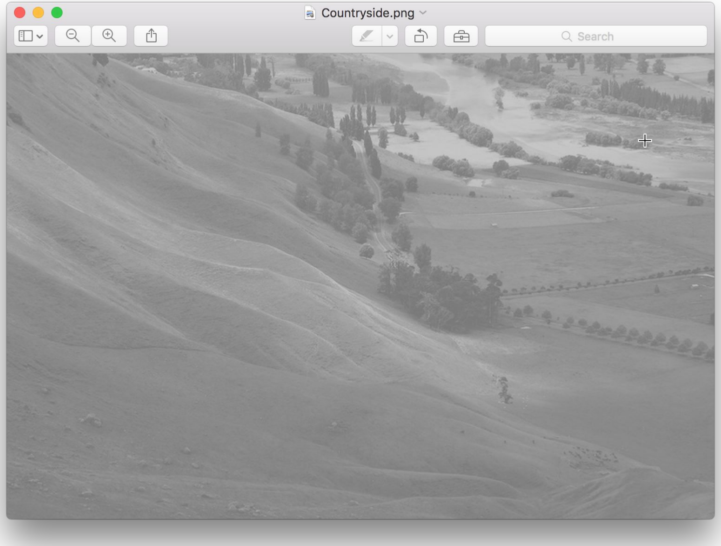
Big idea: the colors here are all shades of grey that are “close” to one another.

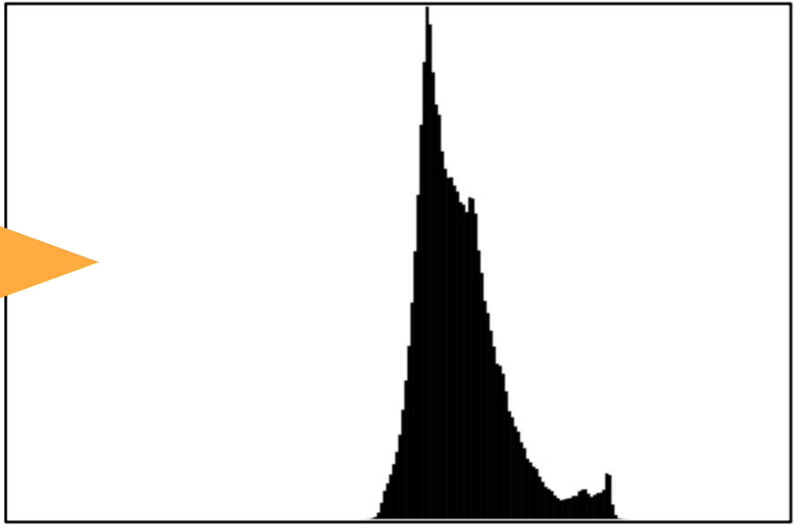


Instead, we want to spread them out, so there are more “extreme” values or greater contrast.

Histogram Equalization

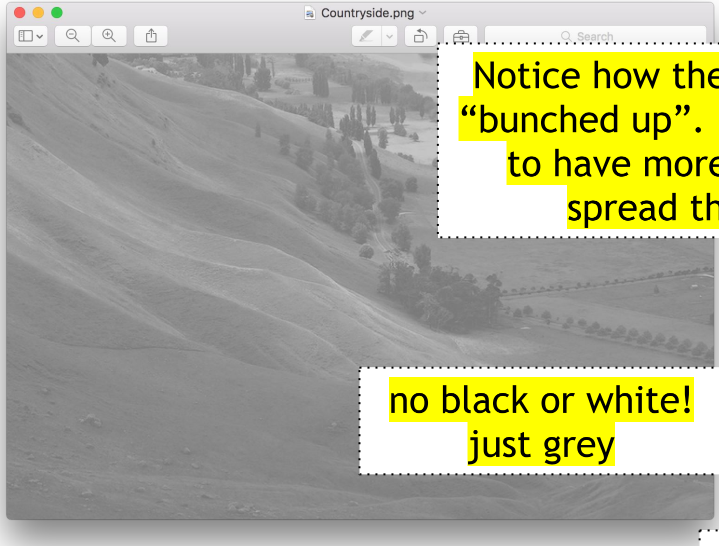
- Main idea: spread out *luminosity values* as much as possible
- Step 1: compute the luminosity histogram.





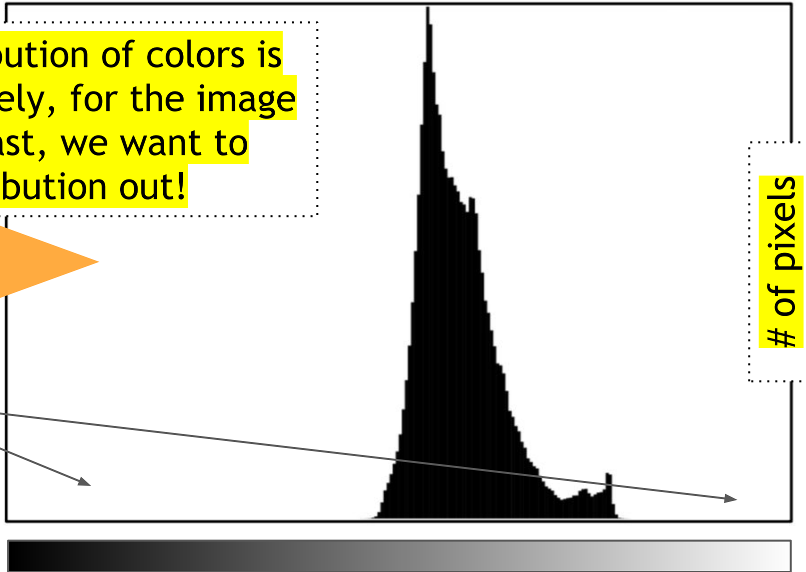
Histogram Equalization

- Main idea: spread out *luminosity values* as much as possible
- <u>Step 1</u>: compute the luminosity histogram.



Notice how the distribution of colors is "bunched up". Intuitively, for the image to have more contrast, we want to spread the distribution out!

no black or white! just grey



# of pixels

color

Histogram Equalization

- Step 1: compute the luminosity histogram.

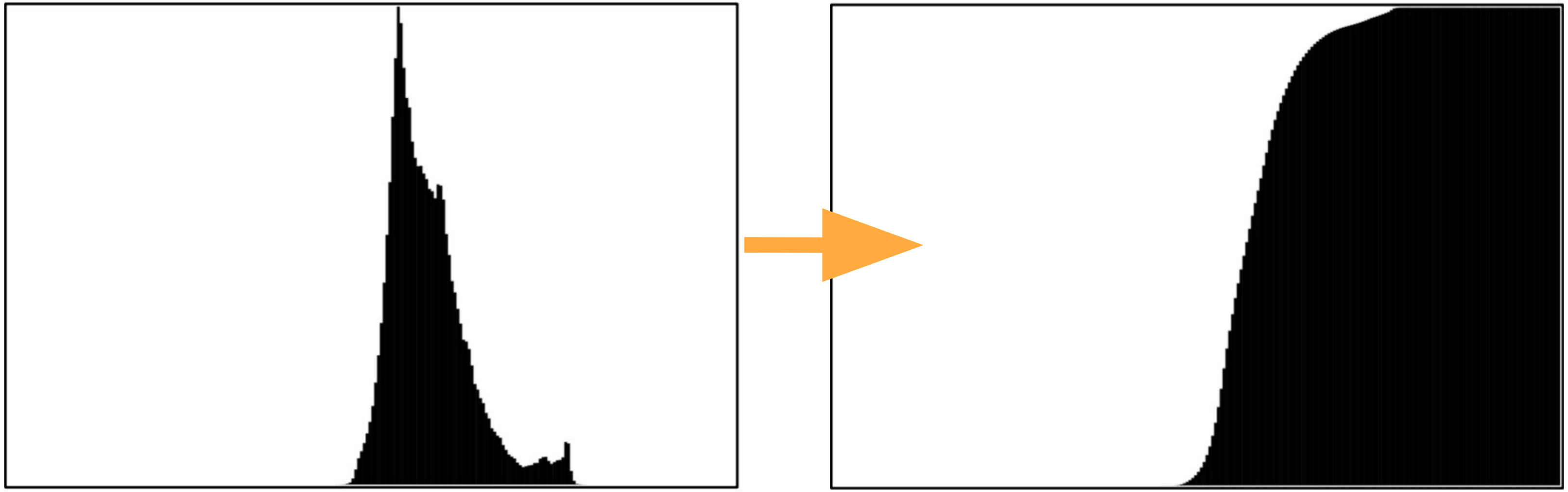
4	24	36	1	0	…	0
0	1	2	3	4	…	255

`histogram[i]` = # of pixels with luminosity *i*

```
int luminosity = computeLuminosity(red, green, blue) // helpful!
```

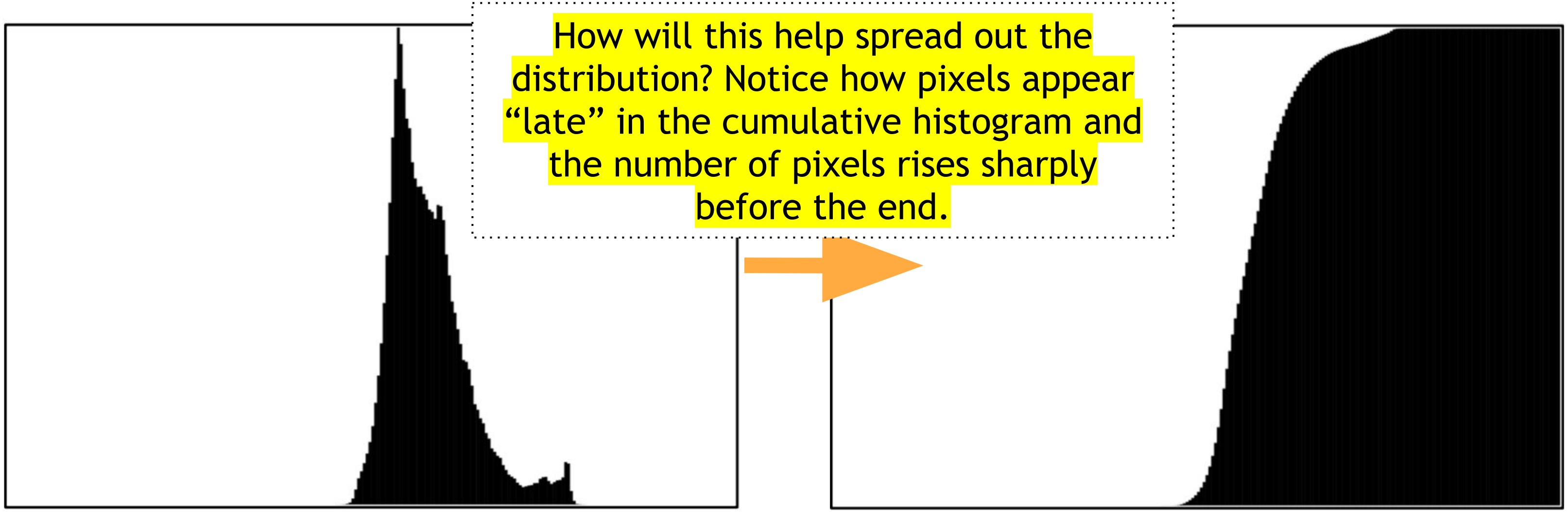
Histogram Equalization

- Step 2: use the luminosity histogram to compute the cumulative luminosity histogram



Histogram Equalization

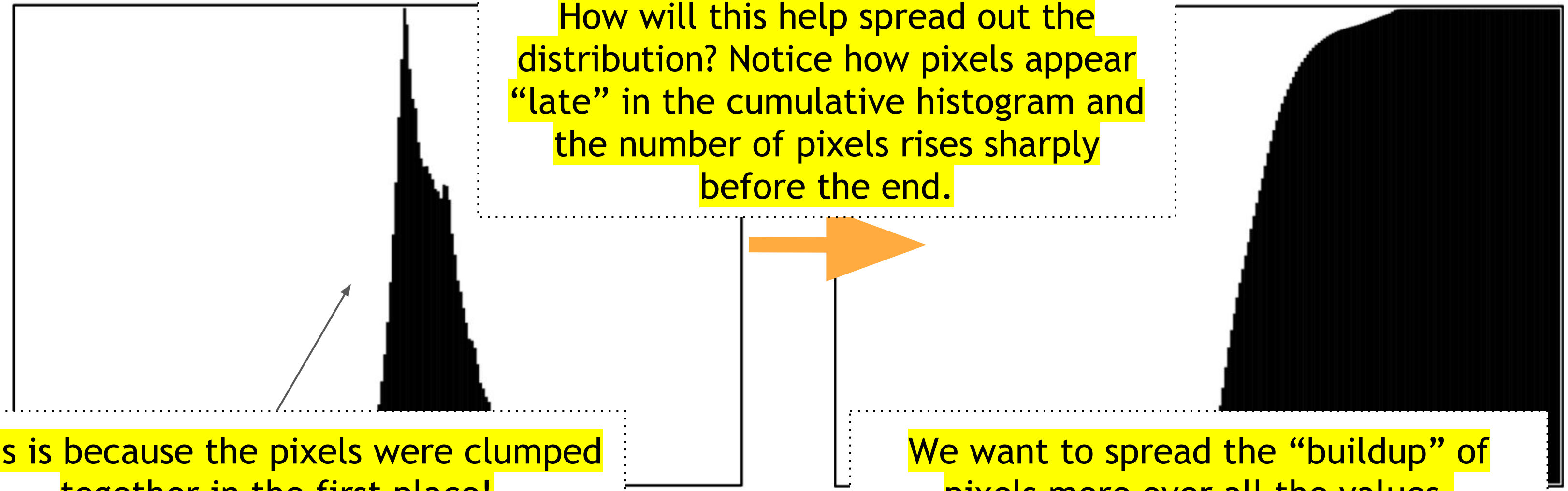
- Step 2: use the luminosity histogram to compute the cumulative luminosity histogram



How will this help spread out the distribution? Notice how pixels appear “late” in the cumulative histogram and the number of pixels rises sharply before the end.

Histogram Equalization

- Step 2: use the luminosity histogram to compute the cumulative luminosity histogram



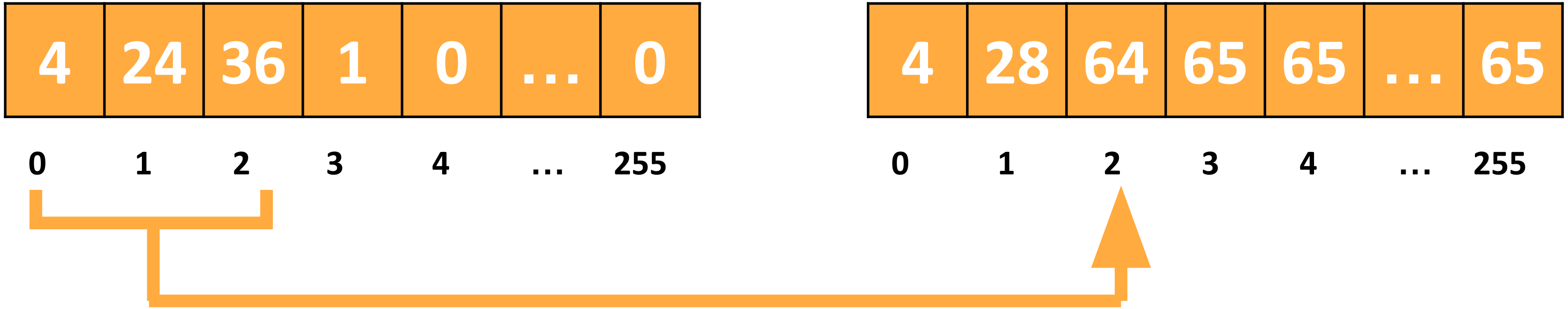
How will this help spread out the distribution? Notice how pixels appear “late” in the cumulative histogram and the number of pixels rises sharply before the end.

This is because the pixels were clumped together in the first place!

We want to spread the “buildup” of pixels more over all the values.

Histogram Equalization

- <u>Step 2</u>: use the luminosity histogram to compute the cumulative luminosity histogram.



4	24	36	1	0	…	0
0	1	2	3	4	…	255

4	28	64	65	65	…	65
0	1	2	3	4	…	255

`cumulativeHistogram[i]` = # of pixels with luminosity ≤ *i*

Careful not to confuse this with the previous histogram!

Histogram Equalization

- Step 3: use the cumulative luminosity histogram to compute new luminosities for each pixel.

$$new\ pixel\ luminosity = \frac{255\ * cumulative\ histogram[\boldsymbol{L}]}{total\ \#\ pixels}$$

where $\boldsymbol{L}$ is the original luminosity for the pixel. Change each pixel to have red, green, and blue values equal to its new luminosity.

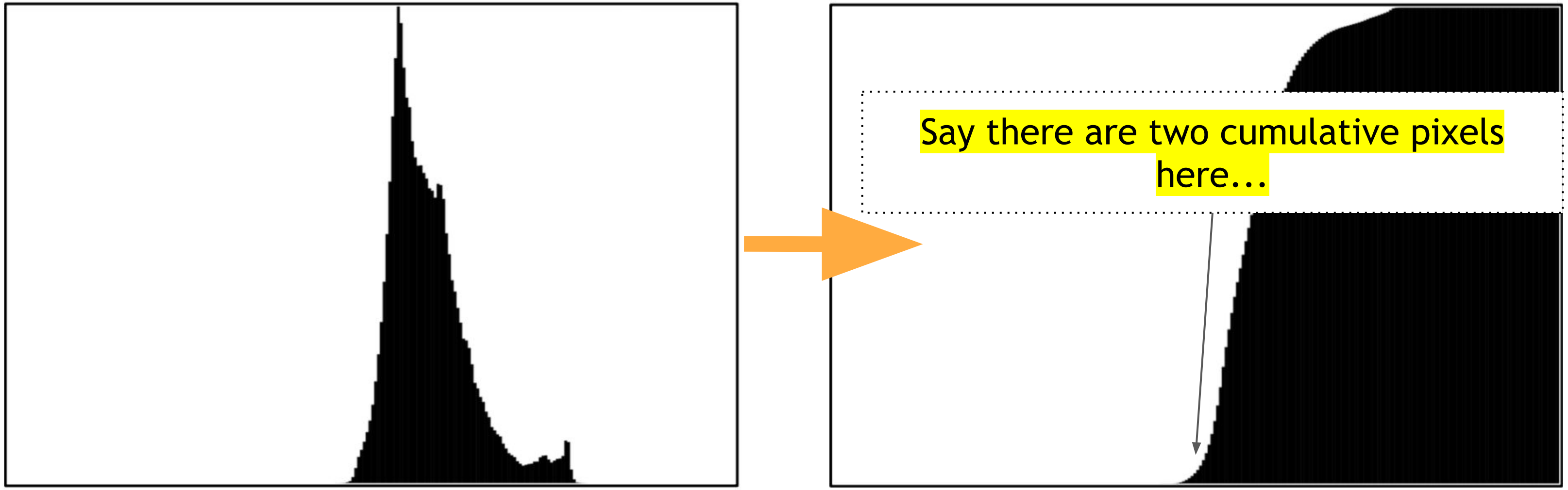
Histogram Equalization

- Step 3: use the cumulative luminosity histogram to compute a new luminosity for each pixel given its old luminosity.

4	28	64	65	65	…	65
0	1	2	3	4	…	255

Old L	New L
0	≈ 16
1	≈ 110
2	≈ 251
3	255
…	…
255	255

How Does this help?

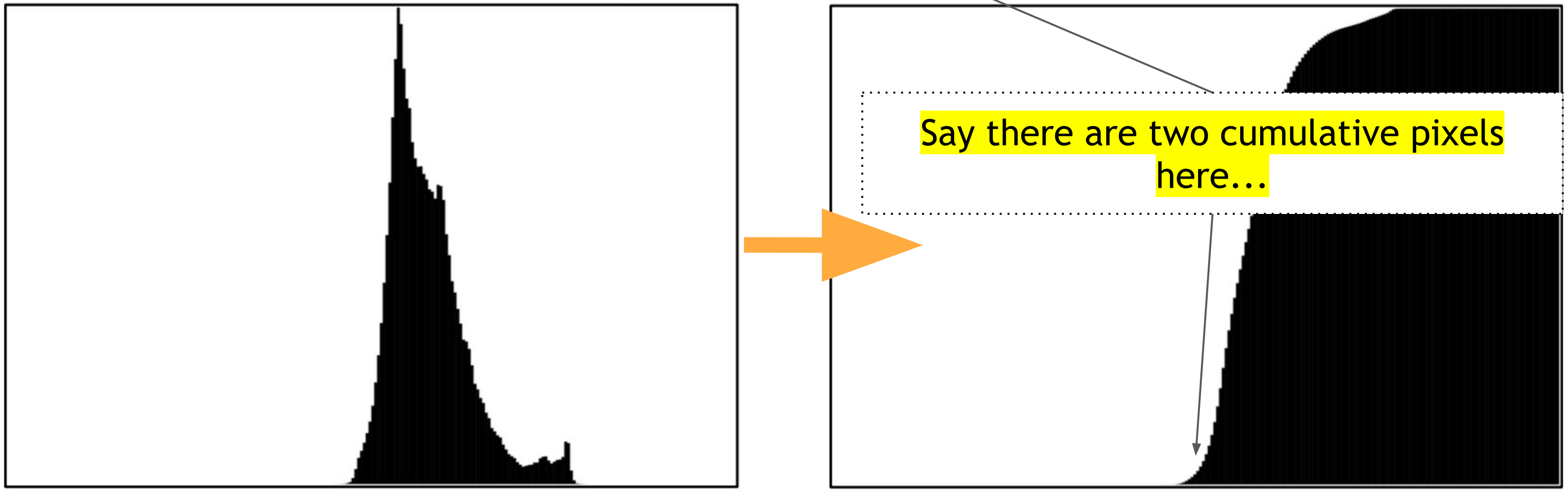


Say there are two cumulative pixels here…

Histogram Equalization

$$255\ x\ \frac{\#\ pixels\ with\ luminosity \le this\ pixel's\ luminosity}{\#\ total\ pixels}$$

This will be 255 times a number close to 0.

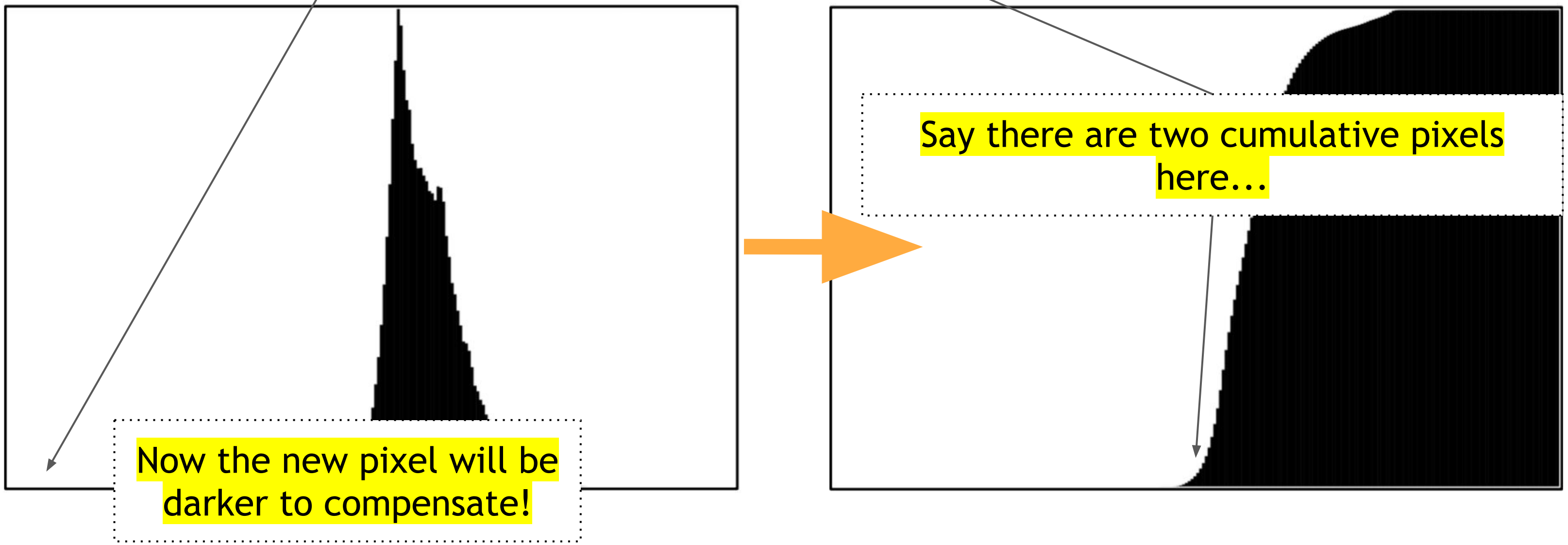


Say there are two cumulative pixels here…

Histogram Equalization

$$255\ x\ \frac{\#\ pixels\ with\ luminosity \leq this\ pixel's\ luminosity}{\#\ total\ pixels}$$

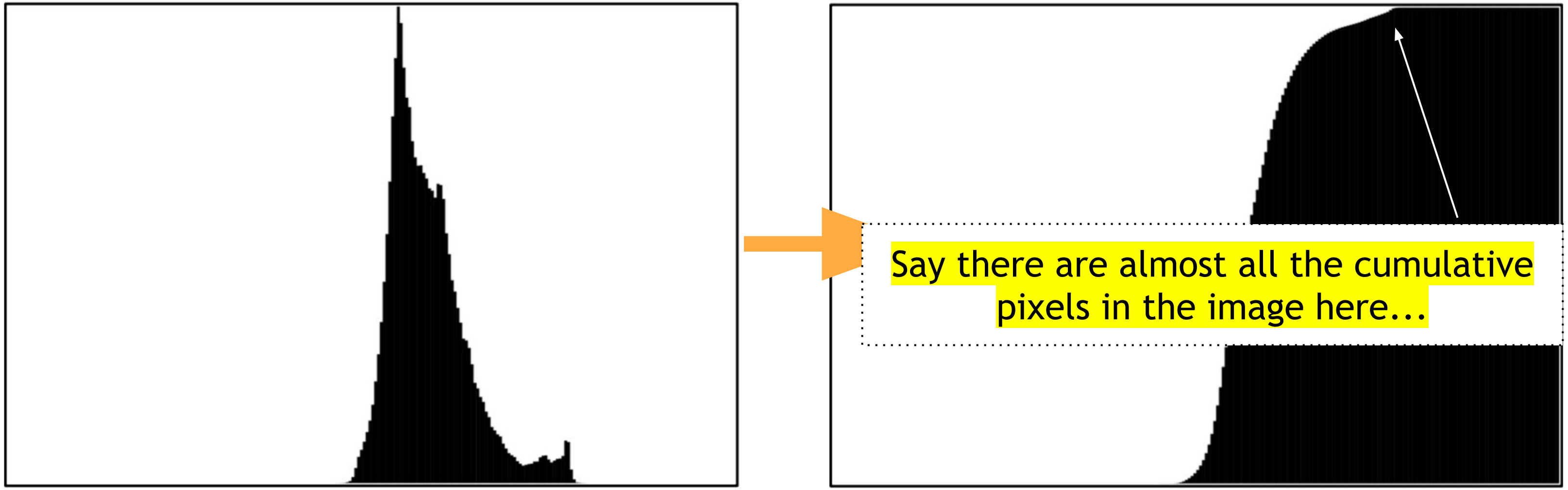
This will be a small number.



Say there are two cumulative pixels here…

Now the new pixel will be darker to compensate!

Histogram Equalization

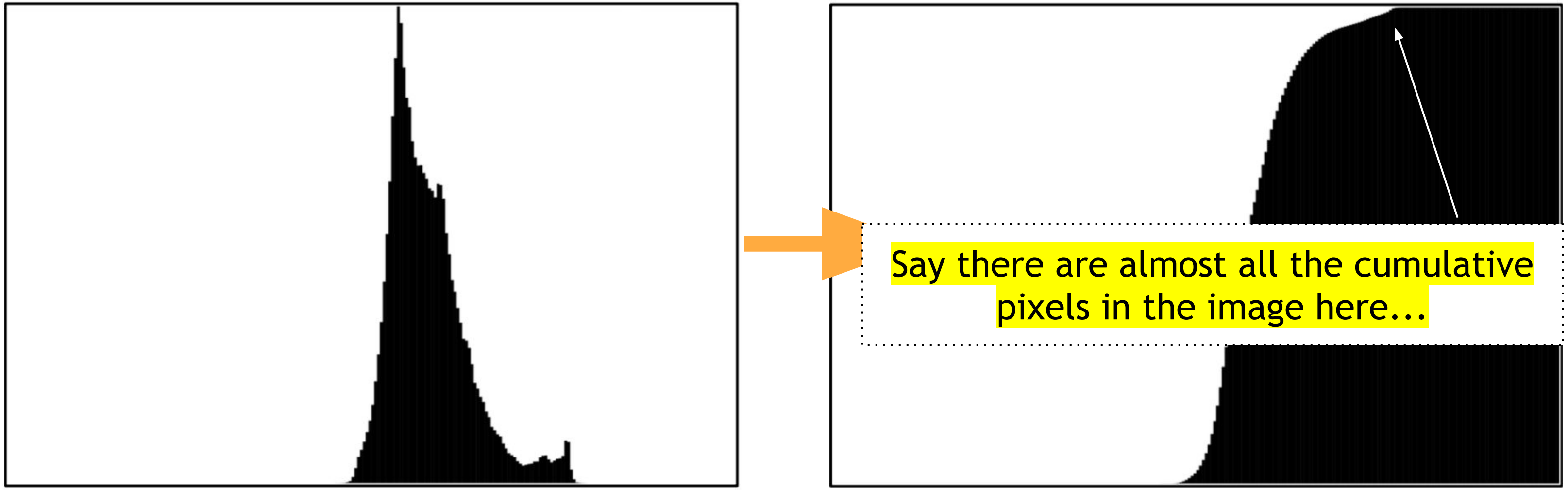


Say there are almost all the cumulative pixels in the image here...

Histogram Equalization

$$255\ x\ \frac{\#\ pixels\ with\ luminosity \le this\ pixel's\ luminosity}{\#\ total\ pixels}$$

Then we have 255 times a number close to 1!

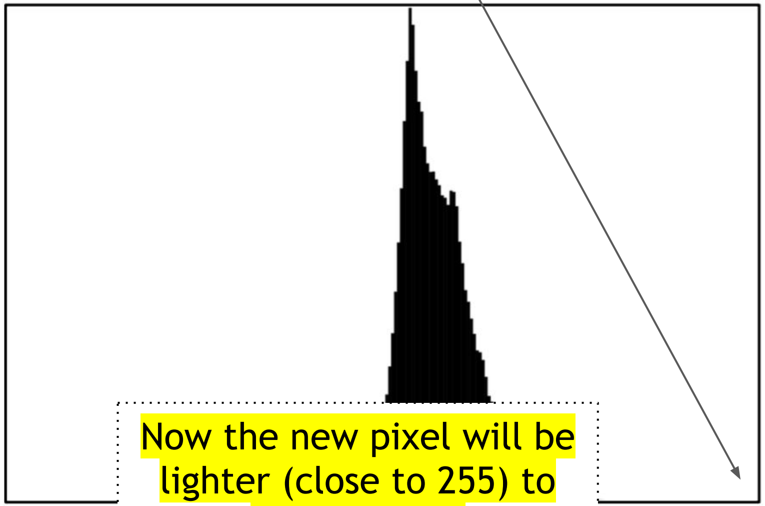


Say there are almost all the cumulative pixels in the image here...

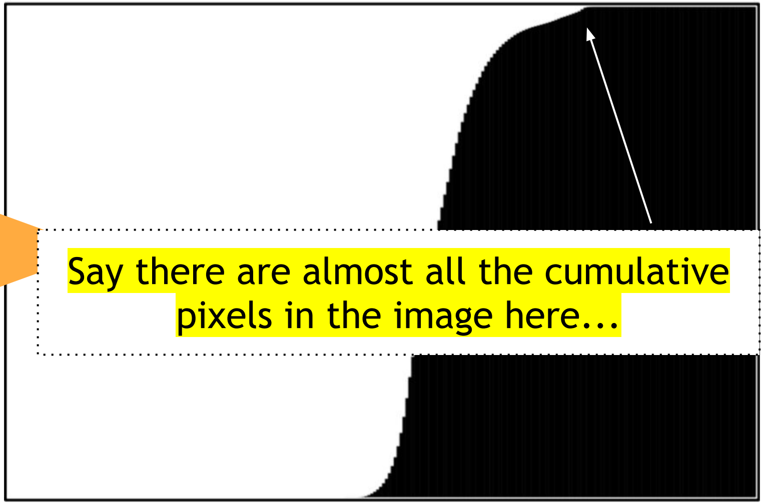
Histogram Equalization

$$255\ x\ \frac{\#\ pixels\ with\ luminosity \le this\ pixel's\ luminosity}{\#\ total\ pixels}$$

This will be 255 times a number close to 1!



Now the new pixel will be lighter (close to 255) to compensate!



Say there are almost all the cumulative pixels in the image here...

Histogram Equalization

Summary:

1. Compute the histogram for the original image.
2. Compute the cumulative histogram from the image histogram.
3. Change each original pixel to have red, green, and blue values equal to its new luminosity from the given formula.

Hint: **decompose into helper methods!** (e.g. computing the histogram and computing the cumulative histogram).

Tips and Tricks

Read read *read* the documentation! It's on the course website.

Check the given functions in the handout. You may have some of your work cut out for you! Read the handout too (no ivars allowed!)

Think about algorithms conceptually, and then move up to pseudocode, and then implement them for real! Then, most importantly, **test them**!

The **Compare To Image** option in the program will be great for testing.

Run the demo JAR from the course website!

{}

Documentation is your friend.
As with any good friend, use ‘em.

Tips and Tricks: The Pixel Problem

Back to blur...do we see any problems with it?

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by # neighbors)
```

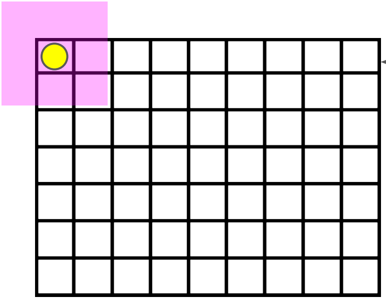
Hmmm...

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by #
```

Let’s look at this step as it repeats.

tinyurl.com/yeah-hours

Hmmm...

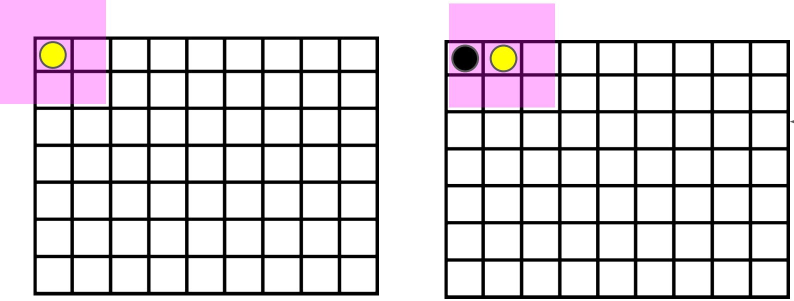


We blur over the pixels in range of this pixel p1...

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by #
```

Let’s look at this step as it repeats.

Hmmm...

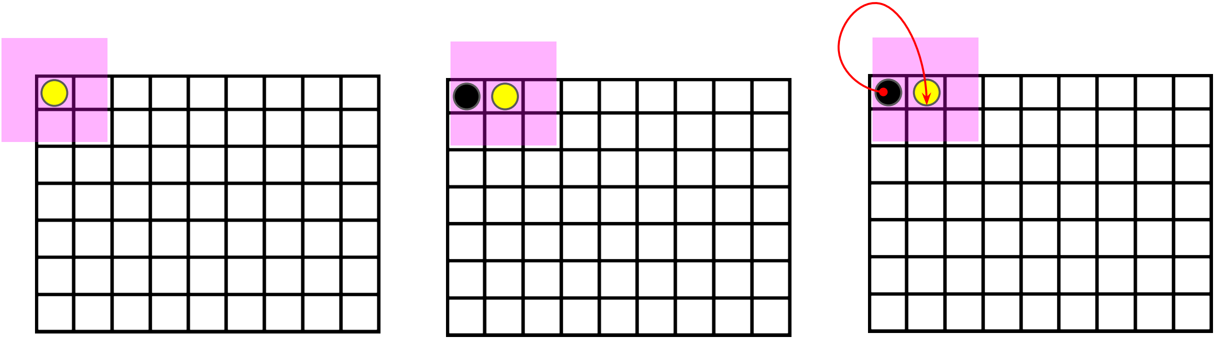


We move to the next pixel p2...

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by #
```

Let’s look at this step as it repeats.

Hmmm…



Oops! We're using a blurred pixel to calculate the next pixel's value.

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by #
```

Let's look at this step as it repeats.

tinyurl.com/yeah-hours

Tips and Tricks: The Pixel Problem

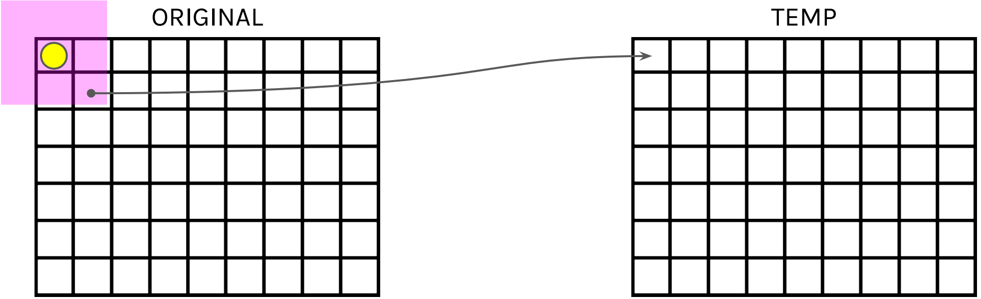
Back to blur…there’s something about it!

How do we put the blurred pixel p prime in p’s place without overwriting p? We don’t want to select p again and blur an already blurred pixel…

```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by # neighbors)
```

Tips and Tricks: Temps

Intuition: to avoid double-scattering, we can create a temporary `int[][]` to map our new pixels to.

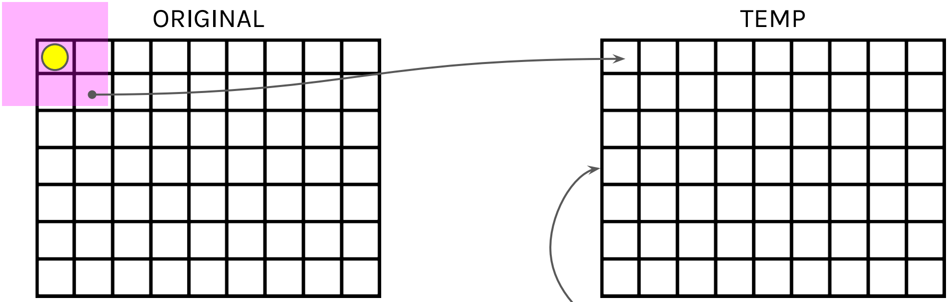


```
// for each pixel in the image:
//     for each neighbor in range, including that pixel
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by #
```

Let’s look at this step as it repeats.

Tips and Tricks: Temps

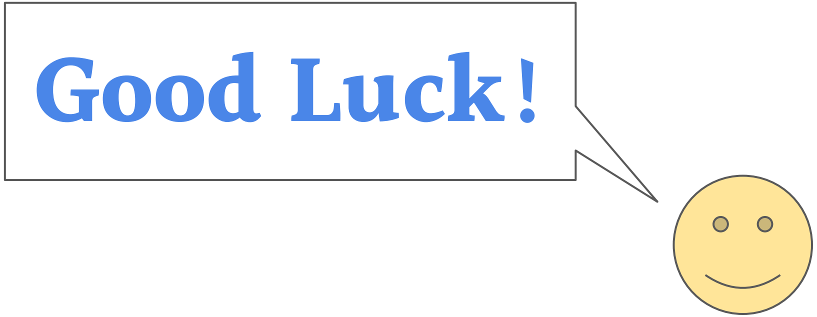
Temporary arrays are useful when you don’t want data in a new image you’re making to **bleed over & influence** said image!



```
// for each pixel in the image:
//     for each neighbor in range, including
//         tally up the total RGB
//     the new pixel is the avg. RBG (divide by #
```

Nothing in temp is being used to create the new image.

Let’s look at this step as it repeats.



Good Luck!