Nick Troccoli
CS 106A

Section #5
July 24, 2017

Section Handout #5: Arrays

Portions of this handout by Marty Stepp

1. Array Simulation

Write the final array contents when the following method is passed each array below:

```
public void mystery(int[] nums) {
    for (int i = 0; i < nums.length - 1; i++) {
         if (nums[i] > nums[i + 1]) {
             nums[i + 1]++;
         }
     }
 }
```

2. Index Of

Write a method named **indexOf** that returns the index of a particular value in an array of integers. The method should return the index of the first occurrence of the target value in the array. If the value is not in the array, it should return `-1`. For example, if an array stores the following values:

```
int[] a = {42, 7, -9, 14, 8, 39, 42, 8, 19, 0};
```

Then the call `indexOf(a, 8)` should return 4 because the index of the first occurrence of value 8 in the array is at index 4. The call `indexOf(a, 2)` should return `-1` because value 2 is not in the array.

3. Unique Numbers

Write a method named **numUnique** that accepts an array of integers as a parameter and returns the number of unique values in the array. The array is guaranteed to be in sorted order, which means that duplicates will be grouped together. For example, if an array stores the following values:

```
int[] list = {5, 7, 7, 7, 22, 22, 23, 35, 35, 40, 40, 40}
```

…then the call `numUnique(list)` should return 6 because this list has 6 unique values `(5, 7, 22, 23, 35, 40)`. It is possible that the list might not have any duplicates. If list instead stored:

```
int[] list = {1, 2, 11, 17, 24, 25, 26, 31, 34, 37, 40, 41}
```

then a call on the method would return 12 because this list contains 12 different values. If passed an empty list, your method should return 0.

4. Banish

Write a method named **banish** that accepts two arrays of integers a1 and a2 as parameters and removes all occurrences of a2's values from a1. An element is "removed" by shifting all subsequent elements one index to the left to cover it up, placing a 0 into the last index. The original relative ordering of a1's elements should be retained. For example, suppose the following two arrays are declared and the following call is made:

```
int[] a1 = {42, 3, 42, 11, 42, 42, 17, 0, 2222, 4, 9, 1};
int[] a2 = {42, 2222, 9};
banish(a1, a2);
```

After the call has finished, the contents of a1 should become:

```
[3, 11, 17, 0, 4, 1, 0, 0, 0, 0, 0, 0]
```

Notice that all occurrences of the values 42, 2222, and 9 have been removed and replaced by 0s at the end of the array, and the remaining values have shifted left to compensate.

5. Collapse

Write a method named **`collapse`** that accepts an array of integers as a parameter and returns a new array containing the result of replacing each pair of integers with the sum of that pair. For example, if an array called list stores the values `{7, 2, 8, 9, 4, 13, 7, 1, 9, 10}`, then the call of collapse(list); should return a new array containing `{9, 17, 17, 8, 19}`. The first pair from the original list is collapsed into 9 (7 + 2), the second pair is collapsed into 17 (8 + 9), and so on. If the list stores an odd number of elements, the final element is not collapsed. For example, if the list had been `{1, 2, 3, 4, 5}`, then the call would return `{3, 7, 5}`. Your method should not change the array that is passed as a parameter.

6. Find Median

Write a method named **findMedian** that is passed an array of integers (representing temperatures) and returns the median of the temperatures. The median is defined as the temperature for which half the temperatures are less than or equal to the median, and half are greater than or equal to the median. You can assume the number of temperatures is odd. All temperatures range from 0 to 100 (inclusive).

7. The Sieve of Eratosthenes

Write a program that uses the "Sieve of Eratosthenes" to print a list of prime numbers between 2 and 1000. In the third century B.C., the Greek astronomer Eratosthenes developed an algorithm for finding all the prime numbers up to some upper limit N. To apply the algorithm, you start by writing down a list of the integers between 2 and N. For example, if N is 10, you would write down the following list:

2 3 4 5 6 7 8 9 10

You then underline the first number in the list and cross off every multiple of that number. Thus, after executing the first step of the algorithm, you will underline 2 and cross off every multiple of 2:

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~

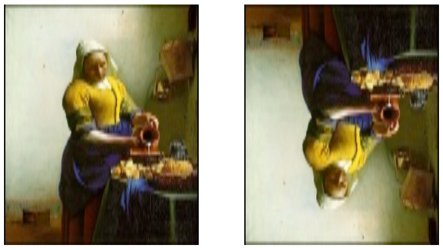
From here, you repeat the process: underline the first number in the list that is neither crossed nor underlined, then cross off its multiples. Eventually, every number in the list will either be underlined or crossed out, as below. The underlined numbers are prime.

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~

Images and Pixels (2D Arrays)

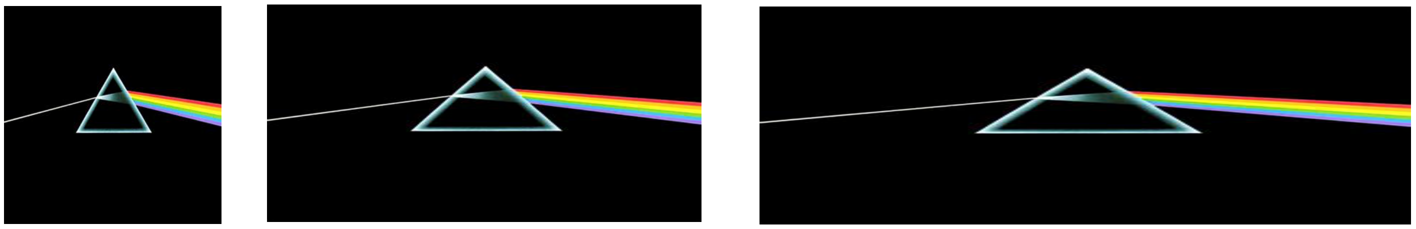
8. Flip Vertical.

Write a method **`flipVertical`** that reverses a picture in the vertical dimension. Thus, if you had a `GImage` containing the image on the left (of Jan Vermeer's The Milkmaid, c. 1659), calling `flipVertical` on that image would modify the pixels of that image as shown on the right:



9. Stretch.

Write a method **`stretch`** that takes a `GImage` and an `int` representing a scale factor as parameters, and modifies the image to horizontally stretch it by the given factor. For example, if the factor is 2, stretch the image to be twice as wide, or if the factor is 3, stretch it to be 3x as wide. As an example, here's a before-and-after comparison of the cover of Pink Floyd's "The Dark Side of the Moon" stretched by 2x and 3x:



10. 2-D Array Simulation. Consider the following method.

```
private void mystery2D(int[][] numbers) {
   for (int r = 0; r < numbers.length; r++) {
       for (int c = 0; c < numbers[0].length - 1; c++) {
           if (numbers[r][c + 1] > numbers[r][c]) {
               numbers[r][c] = numbers[r][c + 1];
           }
       }
   }
}
```

If a two-dimensional array numbers is initialized to:

```
3 4 5 6
4 5 6 7
5 6 7 8
```

…what are its contents after the call of `mystery(numbers);` ?