This section handout covers a variety of topics from various points in the quarter. Hopefully, you and your section can find a few problems in here to work through that will help you review the material and prepare for the upcoming final exam. Have fun with these problems! I've chosen them specifically because I think they're interesting and rewarding, and I hope that they don't disappoint.

Problem One: Probably Unimportant Pages

When determining the importance of various pages on the Internet, one of the fundamental assumptions made is that important pages end up getting linked to. If a page has no incoming links, it's probably not a very important page.

Suppose that you are given a graph of webpages represented as a `Map<String, List<String>>` that associates each page (using its name as the key) with the list of pages it links to. Write a method

```java
private Set<String> findUnimportantPages(Map<String, List<String>> links)
```

that returns a `Set<String>` containing the names of all pages that have no other pages that link to them. These are the pages that probably aren't very important.

Problem Two: Handling Commas

In last Friday's lecture, we wrote a program that contacted Google, performed a search, and outputted how many pages Google said matched that search query. The method that we wrote returned this value as a `String` for simplicity. One of the reasons for this is that the Google search page automatically inserts commas into the number of pages returned. For example, if a query had one million search results, Google would represent this with the string 1,000,000. Unfortunately, the `Integer.parseInt` method cannot parse strings containing commas, and attempting to do so will cause a `NumberFormatException`.

Write a method

```java
private int parseIntWithCommas(String number)
```

that accepts as input a `String` representing a number with appropriate commas inserted, then returns an `int` containing the value of that number. You can assume that the input string is properly-formatted and represents a number that fits into an `int`.

Problem Three: Searching and Sorting

Recall that the time required to selection sort an array of \( N \) elements is roughly proportional to \( N^2 / 2 \).

1. Suppose that it takes about 4s to sort an array of 10,000 elements with selection sort. About how long will it take to sort an array of 20,000 elements?

2. Suppose that it takes about 4s to sort an array of 10,000 elements with selection sort. About how long will it take to sort two arrays, each of which has 5,000 elements?
Problem Four: Case-Insensitive Maps

One of the requirements of the NameSurfer assignment is to ensure that names are treated case-insensitively. For example, searching for “JEREMY,” “jeremy,” or “Jeremy” should all return the same result.

This is actually a fairly common setup, and to simplify things it might be nice to build our own class that's similar to a HashMap except that the keys are treated case-insensitively. In this problem, you'll build a class called CaseInsensitiveMap that is similar to a Map<String, String>, except that the keys are treated case-insensitively.

Provide an implementation for the three methods in the following class:

```java
public class CaseInsensitiveMap {

    /**
     * Inserts a new key/value pair into the map. If the key already exists
     * in the map, the old key/value pair is overwritten.
     *
     * @param key The key to insert.
     * @param value The value associated with the key.
     */
    public void put(String key, String value) {
        // TODO: Implement this!
    }

    /**
     * Returns the value associated with the given key, returning null if the
     * key doesn't exist.
     *
     * @param key The key to look up.
     * @return The associated value.
     */
    public String get(String key) {
        // TODO: Implement this!
        return null;
    }

    /**
     * Returns whether the given key exists in the map.
     *
     * @param key The key to look up.
     * @return Whether it exists in the map.
     */
    public boolean containsKey(String key) {
        // TODO: Implement this!
        return false;
    }
}
```
Problem Five: Inverting Colors

If you’ll recall from our lecture on arrays, GImages are internally represented on the computer as a two-dimensional array of pixels. Each pixel has a red, green, and blue component associated with them, and these values range from 0 (zero intensity) to 255 (maximum intensity). For example, pure red would have red value 255, green value 0, and blue value 0.

Given a GImage, we can construct the inverse of that GImage by flipping the intensity of each color channel. For example, if we have a pixel with red value 0, green value 255, and blue value 100, the inverse of that pixel would have red value 255, green value 0, and blue value 155. Visually, this represents taking the opposite of each of the colors in the image, so black pixels become white, green pixels become bright purple, etc.

Write a method

```java
private GImage invertImage(GImage toInvert)
```

that accepts as input a GImage and produces a new GImage that’s the inverse of the original GImage. As an example, here’s a before-and-after comparison of the cover of Pink Floyd's “The Dark Side of the Moon” cover (would that make it the light side of the moon?)