Introduction to Computer Science I
Project 5a
Fancy array footwork

Notice: Due to a snafu on my part, I distributed the solutions to what I had intended for Project 5a. Whoops. So, since the cat was out of the bag, I have updated this assignment, consonant with the instructions that I gave during the lab (when my mistake was revealed). Be sure that you're following these updated instructions.

1 Sorting and searching code

First, login, create a directory, and grab some code:

```
$ mkdir project-5
$ cd project-5
$ cp -v ~sfkaplan/public/COSC-111/project-5a/* .
```

First, use *Emacs* to open and examine `SortIt.java`. This program contains a pre-written `main()` method, as well as the following two methods (whose innards have been provided... rats):

- `public static int search (int[] array, int value)`
  Search for an entry in `array` that contains `value`, and return the index at which it was found; return -1 if `value` does not occur in `array`. The code here performs a *bubble sort*.
- `public static void sort (int[] array)` Order the elements in `array` from least to greatest. The code provided performs a *linear search*.

Running the program: The `SortIt` program is run something like this:

```
$ java SortIt 100 -33
```

The program performs a number of steps:

1. It calls on a method in `Tools.class` to create an array of mostly random values; I say *mostly* because one of those values actually has a prescribed value. The length of this array is provided by the user at the command line.
2. It then calls on `sort()` to sort the array.
3. A `Tools` method is then used to verify that the sort is correct (printing error messages if it is not).
4. Finally, it uses `search()` to find the index of a particular value—one that the user gets to specify at the command line. The location of this value is then printed.

1.1 Making the key methods faster

While the bubble sort and linear search in `SortIt` work, we know that they are slower on large arrays than better solutions. So, you should do the following to create a new copy of this work that you will change:

```
$ cp SortIt.java FastSortIt.java
```

You can then open `FastSortIt.java` with Emacs. Once you do, you should:

1. Change the name of the program. That is, `public class SortIt` must be changed to `public class FastSortIt`.
2. Delete the bodies of the `sort()` and `search()` methods. You are going to rewrite them. Specifically, **you must implement** *mergesort* and *binary search* in these methods, respectively.

1.2 How to write and test your code

I would approach this program in the following sequence:

1. **Write the sorter:** Ignore the `search()` method at first—just have it return `-1` all of the time so that it compiles. In the meantime, complete `sort()` and any supporting methods that you create for it.
2. **Add code to print the array:** There is, among the `Tools`, a helpful method for debugging that prints the contents of the array. Insert call(s) to it in your `main()` method so that you can see the results of your sorting. For example, in `main()`:

   ```
   sort(values);
   Tools.printArray(values); // <- added for debugging
   Tools.verifySort(values);
   ```

3. **Test the sorter:** Compile your code, and then invoke the program. At first, specify very small lengths (and put a dummy value in the place of the value for which the program searches, since that method is not yet written):

   ```
   $ java FastSortIt 1 0
   [0] = 24541
   0 not found in the array

   $ java FastSortIt 3 0
   [0] = 5068
   [1] = 7035
   [2] = 24541
   0 not found in the array
   ```

Use increasing lengths and check, visually, that the values seem to be in order.

4. **Write the searcher:** Now that your sorting works, complete the `search()` method to perform a search.

5. **Insert a known value:** Modify `main()` again to set one of the values in your array to some constant before it is sorted. That is:

```
int[] values = Tools.createValues(arrayLength);
values[0] = 12345; // <- inserted
sort(values);
```

6. **Test the searcher:** Run your program, searching for the known value:

```
$ java SortIt 10 12345
[0] = 5068
[1] = 7035
[2] = 8115
[3] = 12345
[4] = 28338
[5] = 54270
[6] = 77788
[7] = 92985
[8] = 93844
[9] = 97967
Found: [3] = 12345
```

7. **Remove the debugging code:** Remove the lines added to `main()` that print the array and that insert a known value. Your program is likely correct now, so you don't need those any longer.

2 Finding a particular value

Now that you have a working program, it's time to really put it to use. In particular, you need to search an array of **50 million** items for a particular value. What value? Here's the clue:

> **Find the Stone stone.** On it is written the name of a fraternity. Take those greek letters, and replace them with their numeric position in the greek alphabet. For example, if the greek letters were Sigma Xi ($\Sigma\Xi$), then those are, respectively, the 18^{th} and 14^{th} letters of the greek alphabet. Now contatenate those numbers (e.g., 18 and 14 become 1814). That is the value for which you much search.

So, if 1814 were the number that you must find (it isn't), then you would do the following:

```
$ java FastSortIt 2000000 1814
```

Your program, although it will take a few moments to run, should find that value and show you the position number at which that value was found in the sorted array of 2,000,000 values. **Hold onto that position number!**

3 Finding part B

To get to part **B**, visit the following link:

```
https://www.cs.amherst.edu/~sfkaplan/courses/2013/spring/COSC-111/
projects/project-5b-XYZ.pdf
```

Of course, `XYZ` should be replaced with the position number at which the particular value, above, was found.

4 How to submit your work

Use the `cs111-submit` command:

```
  cs111-submit project-5a FastSortIt.java
```

Project 5a is due on Sunday, April 7