Lab 1

In this lab, you will implement the array with constant time initialization that we discussed in class yesterday and (hopefully) run some timing experiments.

**Initializable array**

Begin with my given code, which is available on the course webpage under Assignments (for a zip archive with an Eclipse project) and in `/home/courses/cs180i/lab1` (for individual files). There are two files: `Array.java`, which is a skeleton for the `Array` class, and `ArrayTest.java`, which is a set of JUnit test cases for `Array`.

The skeleton of `Array` declares all the necessary variables so you “just” have to complete the methods:

- For the constructor, you just need to allocate all the arrays, set the default value to 0, and set the number of changes to 0.
- For initialize, you just need to update the default value and set the number of changes to 0.
- For access, you need to use `where` to see where in `whichChanged` the desired index occurs. If you get a valid location (between 0 and the number of changes) and the desired index is at that location, then you return the corresponding location in `changes`. Otherwise, you return the default value.
- For assign, you follow the same validity check as for access. This determines if you need to add a new change (modifying `where`, `whichChanged`, and `numChanges`) or just update the appropriate cell of `changes`.

Once you’ve written these, make sure that your code passes the test cases.

If you have more time, see if you can time these against the implementations in built-in arrays. The general strategy for timing a piece of code is to get the time immediately before it, run the code, get the time immediately afterwards, and then subtract one time from the other. The call `System.currentTimeMillis()` gives you the current time in milliseconds (1000ths of a second) as a `long` (essentially an `int` that can store larger numbers). Write a program that uses this strategy to time a large number of initializes (I used 100,000); the large number is needed because each call to initialize is very quick and also because they take different amounts of time. Collect timing for just the initializations, not the time to create the array.

Also time the same number of initializations of a Java array (using a `for` loop). Try this for different array sizes (I used ten thousand, twenty thousand, etc) and see how the running times compare. Do the times behave as you’d expect?

Next do the same to compare the running times of assign; time a piece of code that uses assign many times to set different array elements. Finally, do the same for access.