CS61B, Fall 2009 **HW #2** **P. N. Hilfinger**

Due: Mon, 14 September 2009 (at midnight)

We have skeleton files for your solutions available in `~cs61b/code/hw2` and also in the Subversion repository. Following the procedure of Lab #2, you can obtain the latter with

```
% cd ~/svnwork
% svn copy $STAFFREPOS/hw2 hw2
% svn commit hw2 -m "Initial HW2 skeleton"
```

on the instructional machines (where `$STAFFREPOS` is shorthand for

```
svn://cs61b-ta@torus.cs.berkeley.edu/home/ff/cs61b-ta/SVN
```

1. In Lab #2, you started an implementation of type `arith.Rational`. Complete this implementation now. Copy the files

```
Makefile Root1.java Root2.java tests.cmd tests.std
arith/Rational.java arith.RationalTest.java
```

from Lab #2 into your `hw2` directory (try this:

```
cd ~/svnwork/hw2
# Don't forget the dot at the end of the next command!
svn copy ~/svnwork/lab2/{*.java,tests.std,tests.cmd,arith} .
svn commit
```

which copies them and puts them under version control in hw2). Now add appropriate methods to `Rational.java` and adapt `Root2.java` to produce answers comparable to those of `Root1`. Be careful! Since you are using `long` instead of "real" integers, it's easy for intermediate results to overflow, producing strange results, so you'll have to be unambitious for now. Add tests (both unit tests and regression tests) of your implementation. I'm not dictating the interface you choose for `Rational`, but I do suggest that you use `BigDecimal` for hints, as in the lab.

2. Complete the following Java function so that it performs as indicated in its comment. The files `IntList.java` and `IntList2.java` in the template contain the declarations of the classes `IntList` and `IntList2`. Put your answers in a file called `Lists.java`, for which we've also provided a template. You may find the functions in the file `Utils.java` to be useful for testing.

```
/** The list of lists formed by breaking up L into "natural runs":
 *  that is, maximal ascending sublists, in the same order as
 *  the original.  For example, if L is (1, 3, 7, 5, 4, 6, 9, 10),
 *  then result is the three-item list ((1, 3, 7), (5), (4, 6, 9, 10)).
 *  Destructive: creates no new IntList items, and may modify the
 *  original list pointed to by L. */
static IntList2 naturalRuns (IntList L) {
    /* *Fill in here* */
}
```

3. Complete the following Java functions so that they perform as indicated in their comments. Remember that some arrays can have zero elements. Put your answers to this problem (all parts) in a file named `Arrays.java`, for which we've provided a template. You may find the contents of the file `Utils.java` useful in testing your answers.

a.
```
/** A new array consisting of the elements of A followed by the
 *  the elements of B. */
static int[] catenate(int[] A, int[] B) {
    /* *Fill in here* */
}
```

b.
```
/** The array formed by removing LEN items from A,
 *  beginning with item #START (counts from 0). */
static int[] remove (int[] A, int start, int len) {
    /* *Fill in here* */
}
```

Continued...

c.
```
/** The array of arrays formed by breaking up A into
 *  maximal ascending lists, without reordering.
 *  For example, if A is {1, 3, 7, 5, 4, 6, 9, 10}, then
 *  returns the three-element array
 *  {{1, 3, 7}, {5}, {4, 6, 9, 10}}. */
static int[][] naturalRuns (int[] A) {
    /* *Fill in here* */
}
```