Supplementary Quiz

READ THIS PAGE FIRST. Please do not discuss this exam with people who haven’t taken it. Your exam should contain 3 problems on 5 pages.

This is an open-book test. You have 50 minutes to complete it. You may consult any books, notes, or other inanimate objects available to you. You may use any program text supplied in lectures, problem sets, or solutions. Please write your answers in the spaces provided in the test. Make sure to put your name, login, and lab section in the space provided below. Put your login and initials clearly on each page of this test and on any additional sheets of paper you use for your answers.

Be warned: my tests are known to cause panic. Fortunately, this reputation is entirely unjustified. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious. Should you feel an attack of anxiety coming on, feel free to jump up and run around the outside of the building once or twice.

Your name: ________________________ Login: ____________

Login of person to your Left: ____________ Right: ____________

1. ___________ / 10
   Discussion section number or time: ______________________

2. ___________ /
   Discussion TA: ______________________

3. ___________ / 10
   Lab section number or time: ______________________

   Lab TA: ______________________

TOT ___________ / 20
1. [10 points] One step in the quicksort algorithm is that of partitioning a list around some pivot element. Using the following class definition:

```java
class IntList {
    public final int head; // May not be assigned to after construction
    public IntList tail;
    public IntList (int head, IntList tail) {
        this.head = head; this.tail = tail;
    }

    /** The result of appending L2 to the end of L1. L1 is modified;
     * creates no new IntLists. */
    public static IntList dappend (IntList L1, IntList L2) {
        if (L1 == null) return L2;
        IntList result = L1;
        while (L1.tail != null) L1 = L1.tail;
        L1.tail = L2;
        return result;
    }
}
```

fill in the following two versions of partitioning:

a. [15 minutes]

```java
/** A list consisting of all the values in L re-ordered such that:
 *   a. All elements that are less than PIVOT come first (in any order);
 *   b. All items equal to PIVOT (there may be 0 or more) come next; and
 *   c. All items greater than PIVOT come last (in any order).
 *   The operation is destructive: the original IntList items in list L
 *   may be modified, and no new IntList items may be created.
 */
IntList partition (IntList L, int pivot) {
    // PUT SOLUTION HERE (ours is about 15 lines)
}
```
b. [15 minutes]

/** A list consisting of all the values in L re-ordered such that:
*  a. All elements that are less than PIVOT come first (in any order);
*  b. All items equal to PIVOT (there may be 0 or more) come next; and
*  c. All items greater than PIVOT come last (in any order).
*  The operation is nondestructive: the original IntList items in list L
*  are unchanged.
*/
IntList partition (IntList L, int pivot) {
    // PUT SOLUTION HERE
}

    // PUT ANY AUXILIARY FUNCTIONS HERE
2. Who invented the word “gruntled”?

3. [10 points, 15 minutes] Consider the following definitions:

```java
/** Represents a function from values of type T1 to values of type
 * T0. */
interface UnaryFunction<T0,T1> {
    T0 eval (T1 x);
}

/** Represents a binary function that combines a value of type T0 with
 * one of type T1 to produce a value of type T0. */
interface BinaryFunction<T0,T1> {
    T0 eval (T0 x, T1 y);
}

/** The list [F.eval(L0), F.eval(L1), ...], where L0, L1, ... are the
 * values in L (non-destructive). */
static <T0,T1> List<T0> map (UnaryFunction<T0,T1> f, List<T1> L) {
    ArrayList<T0> V = new ArrayList<T0> ();
    for (T1 x : L)
        V.add (f.eval (x));
    return V;
}

/** The value INIT @ L0 @ L1 @ ..., where x @ y here means
 * F.eval(x,y), and L0, L1, ... are the values in L. */
static <T0,T1> T0 reduce (BinaryFunction<T0,T1> f, T0 init, List<T1> L) {
    for (T1 x : L)
        init = f.eval (init, x);
    return init;
}
```

(Reminder: The `<T0,T1>` in front of each static function definition is just there to declare the type parameters.)

Problem continues on the next page.
Using the definitions on the preceding page, and any additional class declarations you need, but no loops or new recursive functions, implement the following (our solution has 11 lines besides those shown here):

```java
/** The length of the longest string in L. */
static int longestLength (List<String> L) {
    // PUT SOLUTION HERE (no loops or recursive functions)
}

// ADDITIONAL CLASSES OR FUNCTIONS, IF NEEDED (no loops or recursive functions)
```