Midterm Test

READ THIS PAGE FIRST. Please do not discuss this exam with people who haven’t taken it. Your exam should contain 5 problems on 10 pages. Officially, it is worth 40 points (out of a total of 200).

This is an open-book test. You have two hours 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: ____________

1. __________/10
   Login of person to your Left: __________Right: __________

2. __________/10
   Discussion section number or time: __________________________

3. __________/10
   Discussion TA: __________________________

4. __________/

5. __________/10
   Lab section number or time: __________________________

TOT __________/40 Lab TA: __________________________
1. [10 points]
   
   a. How can you check to see if a number is less than 0 using only == and the bit operators (&, |
   ^, ~, <<, >>, >>>)?

   b. The following program compiles correctly. What does the main program (in D) print?

```java
class A {
    int z = 2;
    void f () { this.g (); }
    void g () { System.out.printf("A:%d%n", z); }
    int h () { return z; }
}

class B extends A {
    int z = 15;
    void g () { System.out.printf("h:%d z:%d%n", h(), z); }
}

class C extends A {
    int z = 42;
    void f () { this.g (); }
}

class D {
    public static void main (String[] args) {
        A c1 = new C();
        C c2 = new C();
        A b1 = new B();
        B b2 = new B();
        c1.f ();
        c2.f ();
        b1.f ();
        b2.f ();
    }
}
```
c. Succinctly describe the result of calling the following function:

```java
int p (int x) {
    int n;
    n = 0;
    while (x != 0) {
        n = n ^ x;
        x = x >>> 1;
    }
    return n & 1;
}
```

d. In the function for part (c) above, how would the result differ if we replaced >>> with >>?

e. What is the result of compiling and executing the following? Briefly explain your answer.

```java
abstract class A {
    abstract void f ();
}

class B {
    void f () { printf ("Hello, world!"); }
}

public class Main {
    public static void main (String[] args) {
        Object b = new B();
        g ((A) b);
    }

    static void g (A x) { x.f (); }
}
```
2. [10 points] Provide simple and tight asymptotic bounds for each of the following. Here, “simple” means roughly “no unnecessary terms or constants” and “tight” means “either the largest $\Omega(\cdot)$ and smallest $O(\cdot)$ bounds you can find or, if possible, a $\Theta(\cdot)$ bound.”

   a. $9x^2 + 3x + 14\log x$

   b. $\sum_{i=0}^{N} \sum_{j=0}^{i} j$

   c. The running time, as a function of $N$, for $\text{foo}(N)$, for $\text{foo}$ declared. (I’m asking for running time here, not worst-case time. So we’re asking for upper and lower bounds on how long the program will run with input $N$.)

```c
void foo(int N){
    int x;
    for(x = 0; x < N; x += 1) {
        int y;
        for (y = 0; y < x; y += 1) {
            bar(x,y);    // bar runs in constant time
        }
    }
}
```

More parts on the next page.
d. The running time, as some function of low and high, for search, declared below. That is, define a suitable function \( s(low, high) \), and then give a bound for \( C_{search}(N) \), in terms of \( N \), where \( N = s(low, high) \). (E.g., “Cost of \( \text{search}(N) = O(N^2) \), where \( N = \lg(high + low) \)”). (Again, I’m asking for upper and lower bounds on the time, not just a bound on the worst-case time).

\[
\text{bool search (int A[], int value, int low, int high) }
\begin{align*}
&\text{if(high < low)} \\
&\quad \text{return false;} \\
&\quad \text{int mid = (low + high) / 2;} \\
&\quad \text{if (A[mid] > value)} \\
&\quad\quad \text{return search(A, value, low, mid - 1);} \\
&\quad\quad \text{else if (A[mid] < value)} \\
&\quad\quad\quad \text{return search(A, value, mid + 1, high);} \\
&\quad \text{return true;} \\
\}
\]

e. The running time (not just worst-case), as a function of \( n \), for \( G(n) \), for \( G \) declared

\[
\text{void G(int n) }
\begin{align*}
&\text{if (n == 1)} \\
&\quad \text{return;} \\
&\quad \text{for(int i = 0; i < n; i += 1)} \\
&\quad\quad \text{G(n-1);} \\
\}
\]
3. [1 point] What is an example of preadaptation?

4. [10 points] Using the following class definitions:

   ```java
   class IntList {
       // 'final' means head can’t be changed after the constructor
       // sets it.
       public final int head;
       public IntList tail;
       public IntList (int head, IntList tail) {
           this.head = head; this.tail = tail;
       }
   }
   class IntList2 {
       public final IntList head;
       public IntList2 tail;
       public IntList2 (IntList head, IntList2 tail) {
           this.head = head; this.tail = tail;
       }
   }
   ```

   fill in the methods below and on the next page to agree with their comments. Define any additional
   methods you’d like. HINT: don’t try to make things efficient. You’ll probably find it easier to
   create one list at a time.

   /* a. */
   /** Slice the list L into a list of N lists such that list #k contains
    * all the items in L that are equal to k modulo N, in their original
    * order. For example, if N is 3 and L contains [9, 2, 7, 12, 8, 1, 6],
    * then the result is [ [9, 12, 6], [7, 1], [2, 8] ]. The operation
    * is destructive (it may destroy the original list) and creates no new
    * IntList objects (it will, of course, create new IntList2 objects).
    */
   static IntList2 dslice (IntList L, int N) { // FILL THIS IN
   
   }
/* b. */

/** A list of N lists such that list #k contains all the items in L
 * that are equal to k modulo N, in their original order. For
 * example, if N is 3 and L contains [9, 2, 7, 12, 8, 1, 6],
 * then the result is [ [9, 12, 6], [7, 1], [2, 8] ]. The operation
 * is nondestructive (the original contents of L are not changed).
 */
static IntList2 slice (IntList L, int N) { // FILL THIS IN

} // ADDITIONAL METHODS MAY GO HERE.
5. The class FilteredList represents a read-only view of a List that selects only certain of its members. In this problem, you are to fill in part of its implementation. For example, if L is any kind of object that implements List<String> (that is, the standard java.util.List), then writing

```
List<String> FL = new FilteredList<String> (L, filter);
```
gives a list containing all items, x, in L for which filter.test (x) is true. Here, filter is of type Predicate:

```
interface Predicate<T> {
    boolean test (T x);
}
```

(Don’t worry about that Predicate<T>, even though we haven’t talked about it explicitly. It just means “For any type, T, ….”) The object pointed to by FL above is supposed to be a view of L; when L changes, so does FL (since FL is a read-only view, we aren’t going to worry about the other direction).

a. Fill in the indicated places below to achieve this effect. Do this “from scratch.” That is, do not use the standard AbstractList or AbstractSequentialList classes.

```java
public class FilteredList<T> {
    // Put private instance variables here.

    public FilteredList (List<T> L, Predicate<T> filter) { // FILL IN

    }

    public int size () { // FILL IN

    }

    public T get (int k) { // FILL IN

    }

    }
```

continues on next page
public Iterator<T> iterator () {
    return ______________________________________
}

// Put additional methods and nested classes here.
b. Here is a second formulation of the slice problem from above. Fill it in, using the FilteredList abstraction from part (a) above. We suggest that the returned value be an ArrayList<List<Integer>>.

```java
/** A list of N lists such that list #k contains all the items in L
 * that are equal to k modulo N, in their original order. For
 * example, if N is 3 and L contains [9, 2, 7, 12, 8, 1, 6],
 * then the result is [[9, 12, 6], [7, 1], [2, 8]]. The operation
 * is nondestructive (the original contents of L are not changed).
 */
static List<List<Integer>> slice (List<Integer> L, int N) { // FILL THIS IN
```

// Put any other methods and classes you need here.