Throughout this test, assume that the following definition is available.

Class IntList {
    Public int head;
    Public IntList tail;

    /** The IntList whose head is HEAD and whose tail is TAIL. */
    public IntList (int head, IntList tail) {
        this.head = head; this.tail = tail;
    }
}
1. [10 points] For each of the following assertions, indicate whether it is true or false. For items that are false, demonstrate clearly and briefly that the statement is false.

   a. $1000 + 1/n \in \Theta(1)$

   b. $1/n \in \Theta(1)$

   c. $x^2 + 1000x \in \Theta(1)$

   d. The worst-case time for executing the following statements is in $\Omega(N^2)$:

   ```java
   For (int I = 0; i < N; i += 1)
       If (A[i] > x) {
           For (int j = I; j < N; j += 1) {
               S += A[j];
           }
           break;
       }
   }
   ```

   e. The worst-case time for executing the call $f(x)$ is $\Theta(2^x)$. Assume that the function flip takes constant time and generates a random Boolean value each time it’s called (as if according to the flip of a coin). It can, in principle, produce any possible sequence of return values.

   ```java
   Void f (int y) {
       If (y == 0)
           System.out.println (“Bingo!”);
       If (flip ())
           F(y-1);
       If (flip())
           F(y-1);
   }
   ```
2. [10 points] The partition method shown on the next page re-arranges the times in a non-empty IntList so that all those items that are less than the original first item of the list (called the pivot) come first, all those equal to the original first item come second, and all those larger come last. Aside from this constraint on their order, the times of the list can be mixed up arbitrarily relative to the order of the original list. For example, if \( L \) is initially the IntList

\[
[42, 55, 19, 21, 110, 30, 42, 2, 62, 19, 42, 70, 14]
\]

then the pivot is 42, and if we call

\[
L = \text{partition}(L);
\]

Then afterwards, the list \( L \) could be

\[
[14, 19, 2, 30, 21, 19, 42, 42, 42, 55, 110, 62, 70]
\]

(I say “could be” because the order of the first six items and the last four items is arbitrary). The partition method is destructive: no new IntList items are created, and the order of the original list is lost. The method below works by changing the .tail fields to re-arrange the list, and never re-assigns the .head fields. Fill in the blanks in the method on the next page to make it work properly.
/** The items in L, re-arranged so that all items < pivot come first,
* all those = pivot come next, and all those > pivot come last,
* where pivot is the initial value of L.head.  L may not be null.
*/
Static IntList partition (IntList L) {
    // Strategy: Go through L from first to last and accumulate the result list as we
    // go.  At any time,
    //  first: points to the first IntList object of the result list.
    //  mid: points to the last object in the result list whose
    //       .head is equal to the pivot.
    //  last: points to the last object in the result list.  Its .tail always points to the
    //       IntList objects that haven’t been processed yet.
    // For example, after processing the first 8 objects of the sample
    // list above, we might have
    // [2, 30, 21, 19, 42, 42, 55, 110, 62, 19, 42, 70, 14]
    // ^        ^        ^
    // first       mid      last
    //
    IntList first, mid, last;
    Int pivot = L.head;
    Last = first = mid = L;

    While (_____________________ != null) {
        IntList q = last.tail;  // q is next item to be processed
        If (____________________) {
            // Move item at q from its current location to the front.
            Last.tail = q.tail;
            ______________ = first;
            first = ______________;
        } else if (__________________) {
            // Move item at q from its current location to middle
            last.tail = q.tail;
            ______________ = mid.tail;
            mid = mid.tail = ______________;
        } else {
            // Just extend the result list.
            Last = ______________;
        }
    }
}
3. [10 points] Fill in the blanks in the description below (there are blanks on this page and the next):
You are writing a text editor, and you decide to include a facility for undoing editing operations and for re-running sequences of editing operations. For this purpose, you define an interface that is supposed to capture the idea of an editing command:

```java
interface command {
    /** Perform the action represented by THIS. */
    void doIt();
    /** Reverse the action represented by THIS, returning things to
     * their state before doIt() was called. */
    void doIt();
}
```

One of the classes of your editor handles overall command processing like this:

```java
class Controller {
    List transcript = new ArrayList();
    /** Execute C and add it to the transcript */
    void execute (command c) {
        c.doIt(); transcript.add (c)
    }
    /** Remove the last command from the transcript and undo it. */
    void undo () {
        command c = ____________________________________;
        c.undo ();
    }
}
```

Finally, your editor interprets user commands and then carries them out by calling methods of an Editor class that holds and manipulate the actual text.

```java
class Editor {
    /** The full text being edited. */
    private StringBuffer theText = new StringBuffer();
    // The StringBuffer class, a kind of modifiable version of String,
    // is described in 10.4 and Figure 10.4 of Programming in Java

    private myController = new Controller();
    ....
    /** Insert INSERTION into theText at position K. */
    void insert (String insertion, int k) {
        myController.execute (new Inserter (theText, insertion, k));
    }
    ... // and so on for deletion, substitution, etc.
}
```
Continued from previous page

    // Note: The number and placement of blank lines below is supposed
    // to be strongly suggestive of how many lines of program text you
    // really need.

    Class Inserter _________________________ {  
        Inserter (StringBuffer b, String , int position) {  
            ________________  
        }  
    }  

    // Private fields
    ________________  
    ________________  
    ________________

    // Methods
    ________________  
    ________________  
    ________________  
    ________________  
    ________________  

    }  

4. [2 points] Who were the antagonists at the Battle of Thermopylae?
5. [10 points] Give short, concise answers to the following:

a. Show what is printed by

```java
System.out.println ((15 << 28 >> 28);
System.out.println ((15 << 28 >>> 29);
```

(Reminder: Java ints are 32 bits long).

b. Somewhere in your program you create an IntList object:

```java
positionList = new IntList (x, otherList);
```

and you have checked with gjdb that otherList is non-null when you execute this statement. But later, your program blows up with a NullPointerException on

```java
if (positionList.tail.head > positionList.head) { ….
```

Even though positionList is unchanged. Assuming that there are a lot of statements executed between the assignment to positionList and this statement, how could you best use gjdb to find where your program is setting positionList.tail to null?

c. The following function cannot work as promised. Why not?

```java
/** Assuming that the numbers in L are initially
 * in ascending order, insert X into the list at
 * the right point to maintain ascending order. */
void insert (IntList L, int x) {
    for (IntList p = L; L != null; L = L.tail) {
        ...
    }
    ...
```
d. A student decides to represent a matrix of doubles with an ArrayList of ArrayLists of Doubles (a Double is a “wrapper object” that contains a double). The idea is that the matrix should be expandable. He writes the following method to add a column to the matrix

```java
/** Expand MATRIX on the right with a column of 0’s */
void addAColumn (List matrix) {
    for (int row = 0; row < matrix.size (); row += 1)
        ((List) matrix.get (row)).add (new Double (0.0));
}
```

Mysteriously, he discovers that after executing

```java
addAColumn (B);
```
His matrix B, which initially had 100 rows and 100 columns, now has 100 rows and 200 columns. What was probably wrong?

e. I use a text-formatting system called LaTeX to write tests, readers, and lectures. By hand I can run it like this:

```
Latex lect20.tex
```

And it will produce an output file called lect20.dvi. I want to set up a makefile in that directory containing lectures so that gmake will run the latex program on any lectures that need it. Each time I create a new lecture, I just add the name of the new file to a line in the makefile that looks like this:

```
Latex_SRCS = lect1.text lect2.tex etc.
```

What else should I put in the makefile to make this all work—that is, so that I only have to change the LATEX_SRCS line when I add a new lecture?