Name: NetID:

Prelim 1

CS 2110, 12 March 2019, 7:30 PM

	1	**2**	**3**	**4**	**5**	**6**	**Total**
Question	Name	Short answer	OO	Recursion	Loop invariants	Exception handling	
Max	1	32	23	19	13	12	100
Score							
Grader							

The exam is closed book and closed notes. Do not begin until instructed.

You have **90 minutes**. Good luck!

Write your name and Cornell **NetID**, **legibly**, at the top of the first page, and your Cornell ID Number (7 digits) at the top of pages 2-8! There are 6 questions on 8 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your pages are still stapled together. If not, please use our stapler to reattach all your pages!

We have scrap paper available. If you do a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam so that we can make sense of what you handed in.

Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space provided.

In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that it's good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam. I will not talk about the exam with anyone in this course who has not yet taken prelim 1.

(signature)

1. Name (1 point)

Write your name and NetID, **legibly**, at the top of page 1. Write your Student ID Number (the 7 digits on your student ID) at the top of pages 2-8 (so each page has identification).

2. Short Answer (32 points)

(a) 6 points. Below are three expressions. To the right of each, write its value.

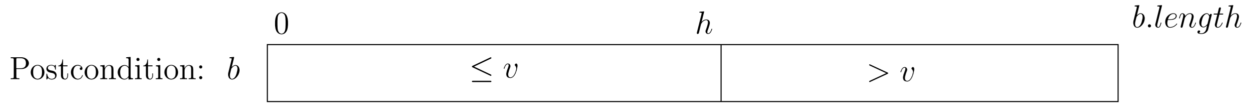
1. `(char) ('G'- 2)`
2. `new Integer(6) == new Integer(6)`
3. `(int) (9.2/2) == (int) (9.2/2)`

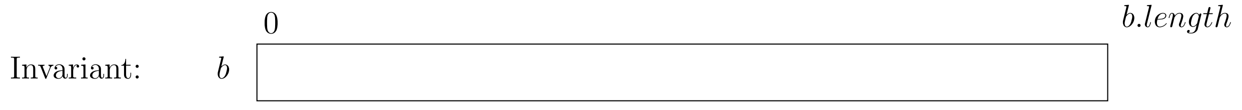
(b) 7 points. Circle T or F in the table below.

(a)	T	F	`if (0) { return "Fail"; }` is not valid Java.
(b)	T	F	Selection sort is stable.
(c)	T	F	The tightest worst-case bound for Mergesort is $O(n * log\ n)$ because its depth of recursion may be proportional to the log of the length n of the array.
(d)	T	F	This declaration overrides method `toString()`: `public String toString(int x) { return "" + x; }`
(e)	T	F	We know that `List` is an interface. Therefore, this declaration is illegal: `List c;`
(f)	T	F	Because `List` is an interface, this statement is illegal: `List c= new List();`
(g)	T	F	Two versions of an overloaded method can have different numbers of parameters.

(c) 4 points. Write the 4-step algorithm for executing the procedure call `z(8.1, 6)` .

(d) 3 points. Binary search. The precondition of binary search is that array `b` is sorted. The postcondition is given below. If `v` is in `b` it indicates that the rightmost occurrence of `v` will be found. Complete the loop invariant, which appears below the postcondition. There is no need to mention that `b` is sorted, since that is always true.





(e) 6 points. Complexity

1. Suppose a method calls a procedure that requires $300 * n$ operations once and then calls another procedure *log n* times, and this second procedure requires $n/2$ operations. Below, circle the tightest (smallest) asymptotic time complexity of this method.

$O(n)$ $O(n\ log\ n)$ $O(n^2)$

2. To the right of the code below, write the tightest (smallest) asymptotic time complexity (in terms of n) of the code. It is known that $n > 5$.

```
double s= 0;
for (int h= 1; h <= n; h= h+1) {
     for (int j= 1; j < n; j= j + n/5) {
          s= s + h/j;
     }
}
```

(f) 6 points. Testing. Below is the signature for function findFurthest.

```
/** Return the value in b that is furthest from zero.
 * Note: -3 is further from zero than 2.
 * If several array values are the same distance from zero,
 * return the one with smallest index.
 * If b is null or empty, return 0.  */
public static int findFurthest(int[] b)
```

Write six (6) distinct test cases in the space below. We don't need formal `assertEquals` calls. Instead, say what is in `b` (or give its values as a list).

One should be convinced the function works as specified if it passes all six of these test cases. Testing nearly identical cases 6 times does not count.

3. Object-Oriented Programming (23 points)

Below and on the next page are three classes, `Plane`, `PassengerPlane`, and `Trip`. Unnecessary parts of classes are omitted. There is no need for assert statements for preconditions.

(a) 3 points Implement `Plane`'s constructor.

(b) 3 points Implement `Plane`'s method `compareTo`.

(c) 7 points Implement `Plane`'s method `equals`.

(d) 6 points Implement `PassengerPlane`'s constructor.

(e) 4 points Implement `PassengerPlane`'s method `equals`.

```
/** An instance contains the serial number of a Plane and its Trips. */
class Plane implements Comparable<Plane> {
  private String serialNum;      // Plane's serial number (not null)
  private ArrayList<Trip> trips; // list of Trips this Plane took

  /** Constructor: a Plane with serial number serialNum and no Trips.
    * Precondition: serialNum is not null. */
  public Plane(String serialNum) { // TODO: Part a





  }

  /** Return the number of Trips this Plane has taken. */
  public int numTrips() { // ... implementation omitted ... }

  /** Compare total number of trips of the two planes
    * as per specification in interface Comparable. */
  @Override public int compareTo(Plane p) {  // TODO: Part b





   }

  /** Return true iff this Plane and ob are of the same class and
      have the same serial number. */
  @Override public boolean equals(Object ob) { // TODO: Part c





  }
}
```

```
/** PassengerPlane has seats and passengers on the plane. */
class PassengerPlane extends Plane {
  int numSeats;                 // Number of seats
  ArrayList<String> passengers; // List of passengers on the plane (not null)

  /** Constr.: instance with serial number s, number of seats seats, no passengers.
    * Precondition: s is not null */
  public PassengerPlane(String s, int seats) { // TODO: Part d





  }

  /** Return true iff this object and ob are of the same class, they have
    * the same name, and they have the same number of seats. */
  public boolean equals(Object ob) { // TODO: Part e






  }
}

/** A Trip between two airports. */
class Trip {
  // Implementation omitted. It's not needed to answer the questions above.
}
```

4. Recursion (19 Points)

(a) 4 points The Hofstadter F and M sequences are defined recursively as follows:

```
public static int F(int n) {
  if (n == 0) return 1;
  return n - M(F(n - 1));
}
```

```
public static int M(int n) {
  if (n == 0) return 0;
  return n - F(M(n - 1));
}
```

Write the calls made in evaluating the call `M(2)` in the order they are called, starting with `M(2)`.

Class Road, below, maintains information about roads leaving Rome. A road either starts at Rome or forks from another road. A road may end or fork into one or two other roads.

(b) 7 points. Complete the body of function timeFromRome(). Use recursion; do not use loops.

(c) 8 points. Complete the body of function isTradeRoute(). Use recursion; do not use loops.

```
Example:                ROME
                         |
                         | A
A starts at Rome.        o
A forks into B, C.      / \
B, C's source is A.    /   \
B has no forks.       / B   \ C
C forks into D, E.   /       \
D, E's source is C.           o
D, E have no forks.          / \
                          D /   \ E
```

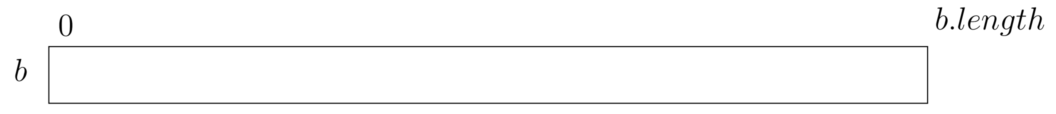
```
public class Road {
  private Road source; // Road that leads to this road (null if it starts at Rome)
  private Road leftFork;   // left fork of road, null if none
  private Road rightFork;  // right fork of road, null if none
  private int travelTime;  // time it takes to travel along this road in hours.
  private boolean hasTradePost; // true if this road has a trading post.

  /** Return the time it takes to travel from Rome to the end of this road */
  public int timeFromRome() {









  }

  /** Return true if this road is part of a trade route.
    * A road is part of a trade route if it has a trading post
    * or either of its forks are part of a trade route. */
  public boolean isTradeRoute() {









   }
}
```

5. Loop Invariants (13 points)

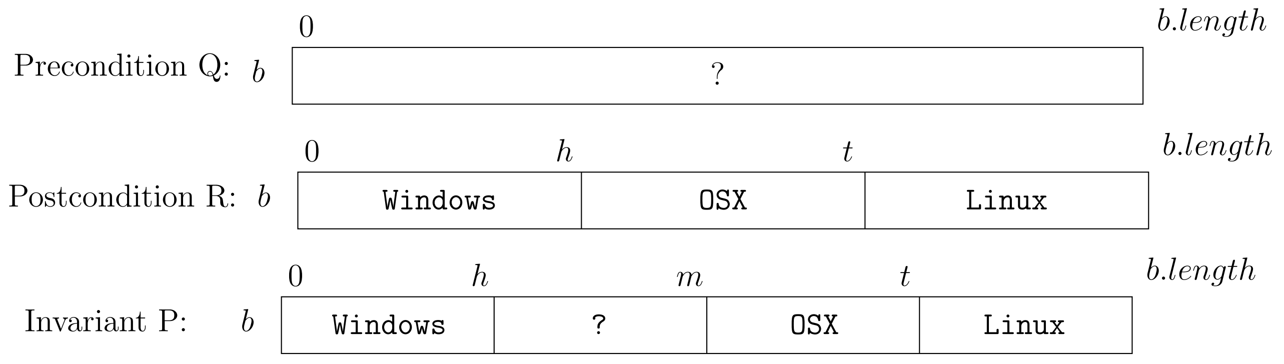
(a) 3 points Consider the assertion

```
b[0..k] are prime  && k ≤ j  && b[j+1...b.length-1] are composite (non-prime)
```

Draw it as an array diagram below:



(b) 10 points Below are the precondition, postcondition, and invariant of a loop that sorts an array b of Students based on the operating system on their laptops (Windows, OSX, Linux):



1. Write the initialization
   code that truthifies P:

2. Write a while-loop condition that makes
   the loop terminate when R is true:

3. Write the repetend. Use `swap(b, i, j)` to swap `b[i]` and `b[j]`. Instances of class Student have three functions `usesWindows()` (true if the student's operating system is Windows), `usesOSX()` (true if the student's os is OSX), and `usesLinux()` (true if the student's os is Linux).

6. Exception handling (12 Points)

In method `foo` to the right, `S0`, `S1`, `S3`, `S5`, `S7`, and `S9` are statements. Below, method `p` calls `foo`.

```
static void p() {
    ...
    foo();
    ...
}
```

Below are four situations that could happen when the call on `foo` in method `p` is executed. Answer the question for each of these situations.

```
public static void foo()  {
    S0
    try { S1 }
    catch (Error e) { S3 }
    catch (ArithmeticException e) { S5 }
    catch (Throwable e) { S7 }
    S9
}
```

(a) 3 points Suppose `S1` throws a `Throwable`. Is it caught by the third catch clause, and if not, what happens to that `Throwable` object?

(b) 3 points Suppose `S0` throws an `ArithmeticException`. Is it caught by the second catch clause, and if not, what happens to that `ArithmeticException` object?

(c) 3 points Suppose `S3` is executed and it does not throw anything. What is executed next?

(d) 3 points Suppose `S5` throws an `Error`. Is it caught by the third catch clause, and if not, what happens to that `Error` object?