Name: NetID:

CS2110 Final Exam SOLUTION

17 May 2018, 7:00PM–9:30PM

	1	2	3	4	5	6	7	8	Total
Question	Name	Short Answer	Sorting	Recursion	Object Oriented	Data Structures	Graphs	Hashing	
Max	0	22	10	7	21	15	16	9	100
Score									

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

You have **150 minutes**. Good luck!

Write your name and Cornell **NetID**, **legibly**, at the top of **every** page! There are 6 questions on 16 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 the final.

_____

(signature)

1. Name (0 points)

Write your name and NetID, **legibly**, at the top of **every** page of this exam.

2. Short Answer (22 points)

(a) True / False (10 points) **Circle** T or F in the table below.

(a)	T	F	The try-catch block can be used to handle both Errors and Exceptions. True.
(b)	T	F	Two nested for-loops, each with a control variable that range over the numbers 1..n, takes expected time $O(n * n)$. False. It depends on what the body of the inner loop does. If it consists of the statement return; the time is $O(1)$.
(c)	T	F	Instance methods of a class generally have access to its static fields. True.
(d)	T	F	To create and start a new thread, (1) have a class C that extends class $Thread$, (2) in C implement method $run()$, and (3) create an instance M of class C. False. You also have to call $M.start()$.
(e)	T	F	A tree can have at most one cycle. False. A tree cannot have any cycles.
(f)	T	F	There can be two shortest paths between two nodes of a graph. True.
(g)	T	F	Once a JLabel has been created, its text cannot be changed. False. You changed a label's text in A6.
(h)	T	F	Primitive type char extends primitive type int. False. char is numeric, but primitives do not extend each other.
(i)	T	F	After a HashMap is resized, all values will be in the same bucket as they were before the resize. False. Every element in the HashMap is rehashed.
(j)	T	F	A function that returns a random int in the range 0..size-1 each time it is called is a valid hash function. False. The function needs to return the same value every time it receives the same argument.

(b) Testing (3 points) Consider the following test for function `boolean mystery(int)`.

```
@Test
public void testMystery() {
  assertEquals(true, M.mystery(1));
  assertEquals(false, M.mystery(321));
  assertEquals(true, M.mystery(222));
}
```

(i) Suppose function `mystery(int i)` returns true if the digits of `i` are in ascending order and false otherwise. Will it pass the test cases in `testMystery`?
Yes

(ii) Suppose function `mystery(int i)` returns true if the digits of `i` form a palindrome and false otherwise. Will it pass the test cases in `testMystery`?
Yes

(iii) Write the specification (in English) of a different version of `mystery(int i)` that would pass the test cases above (do not use any of the functions from i..ii above).
Many possible examples. Here are two. "Return true iff the digits of i are all less than 4." "Return true iff no digit in i is 0."

(c) Classes and Interfaces (3 points) Consider the following interfaces and classes:

```
public interface Valuable {
  public double getValue();
}
public interface Movable {
  public void increaseSpeed();
  public void decreaseSpeed();
}
public class Rock { ... };
public class Jewel extends Rock implements Valuable { ... };
public class Car implements Valuable, Movable { ... };
public class SportsCar extends Car implements Valuable { ... };
```

(I) For each of the following classes, write down the functions that **must** be implemented in the class for that class to compile (possibly, no functions are needed). Do not include functions if they are not required for compilation.
Rock: none
Jewel: getValue
Car: getValue, increaseSpeed, decreaseSpeed
SportsCar: none

(II) Assume you have implemented all the required functions from part I (and no extra functions). For each of the following snippets, could the code result in a compile-time error, a run-time error, or no error? (Assume each snippet is executed separately.)

(i) `Movable car= new Car();` No error.

(ii) `Jewel ruby= new Valuable();` Compile-time error. Cannot create a Valuable.

(iii) `Car jaguar= new SportsCar();` No error.

(iv) `Valuable crownJewel= (Jewel)(new Rock());` Run-time error.

(d) Complexity (3 points) Match algorithms (i)-(vi) with their tightest runtime complexity by writing the appropriate letter next to them. (Some letters may be used twice or not at all.)

(i) Getting the second-to-last item in a singly linked list E

(ii) Getting the second-to-last item in an array list C

(iii) Binary search B

(iv) Insertion Sort A

(v) Merge Sort F

(vi) Linear search E

(A) $O(n^2)$

(B) $O(\log n)$

(C) $O(1)$

(D) $O(n^3)$

(E) $O(n)$

(F) $O(n \log n)$

(e) Concurrency (3 points) Assume that the initial value of `int x` is 5 and consider the following code in two threads:

Thread 1:	**Thread 2:**
`x= 2 * x;`	`x= x % 3;`

What are the possible values of `x` after execution of the two threads?
The possible values are (1) 1: Do x= 2*x; x= x%3;.
(2) 4: Do x= x%3; x= 2*x;.
(3) 1: Threads store x, which is 5, in different registers; x= 2*5 is done; then x= 5%3; is done.
(4) 10: Threads store x, which is 5, in different registers; x= 5%3 is done; then x= 2*5 is done.

3. Sorting (10 Points)

(a) 3 points. Consider the following sorting algorithms for sorting an array of size n: InsertionSort, SelectionSort, MergeSort, QuickSort, and HeapSort. For the questions below, assume that each sort is implemented as described in lecture and in the JavaHyperText.

(i) Which of the above sorting techniques are stable?

InsertionSort and MergeSort

(ii) Which of the above sorting techniques require only $O(1)$ space?

InsertionSort, SelectionSort, HeapSort

(iii) Which of the above sorting techniques run in worst-case time $O(n \log n)$?

MergeSort and HeapSort

(b) 3 points. Consider using quicksort to sort the array {18, 23, 12, 7, 26, 19, 16, 24} in ascending order. The first element of a section is always selected as the pivot.

(i) Draw the array just after quicksort partitions the array and before it makes the first recursive call. Circle the current pivot.

12, 7, 16, 18, 19, 23, 24, 26 (the numbers before and after the pivot could be in any order.)

(ii) What is the maximum possible depth of recursion (how many quicksort stack frames can be on the call stack)?

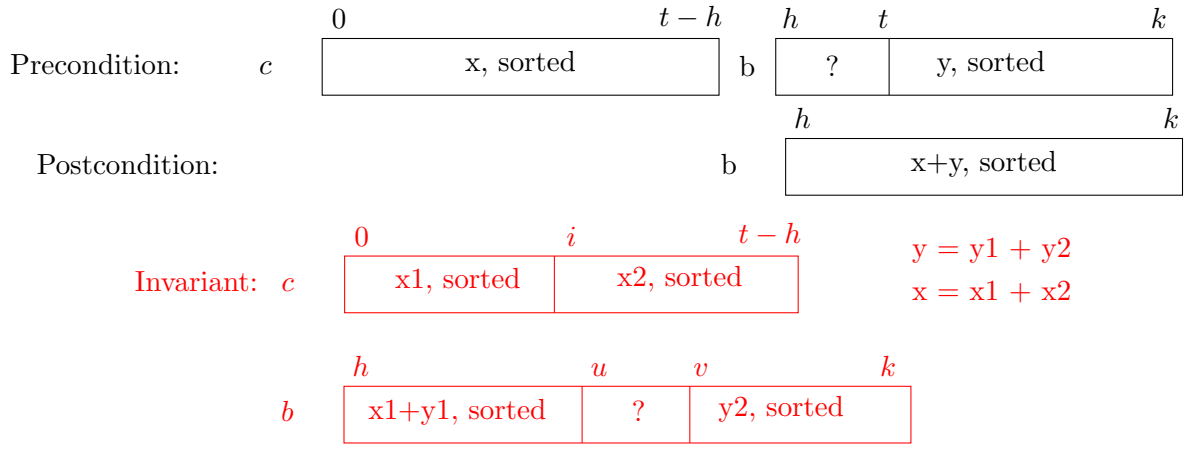
4. Call 1: produce array shown in (i).
Call 2, sorting 19, 23, 24, 26. Partition creates 19, 23, 24, 26.
Call 3, sorting 23, 24, 26. Partition creates 23, 24, 26.
Call 4: sorting 24, 26. No more calls after partitioning because partitions have size < 2.

(c) 4 points You are an intern at Tech Co. for Summer 2019. Your boss asks you to implement mergesort. So far, you have implemented method `mergesort`:

```
/** Sort b[h..k] */
public static void mergesort(int[] b, int h, int k]) {
    if (k+1-k < 2) return;   // base case: size of b[h..k] < 2

    // recursive case: recursively sort each half and merge the two halves
```

```
    int t= (h+k)/2;
    mergesort(b, h, t);
    mergesort(b, t+1, k);
    merge(b, h, t, k);
}
```

You now need to implement method `merge`. You remember the precondition and postcondition for merge (below), but you need to reconstruct the invariant in order to make sure you correctly implement `merge`. Write the invariant for `merge` below (just draw the invariant, don't implement the method). Be sure to include the invariant diagrams for both arrays c and b. x and y stand for the list of values initially in c and $b[t+1..k]$.



4. Recursion (7 points)

Consider the following class `BST`, which implements a binary search tree. We have intentionally omitted most of the class invariant.

```
public class BST<E extends Comparable<E>> {
    public E value;       // value in this node
    public BST<E> left;   // left subtree --null if none
    public BST<E> right;  // right subtree --null if none
    ...
    public void insert(E v) {...} // insert element v into the BST
                                  // according to Comparable<E>
    public ArrayList<E> toArrayList() {...}
}
```

Below, implement method `toArrayList`, which appears in class BST. You can use method `addAll` of class `ArrayList`.

```
/** Return an ArrayList<E> containing the elements of this BST
 *  in sorted order according to E's Comparable instance. */
public ArrayList<E> toArrayList() {
    ArrayList<E> al= new ArrayList<E>();
    if (left != null)  al.addAll(left.toArrayList());
    al.add(value);
    if (right != null) al.addAll(right.toArrayList());
    return al;
 }
```

5. Object-Oriented Programming (21 points)

Below are classes `Vehicle` and `Bus`.

```
public abstract class Vehicle {
  private int numWheels;

  /** Constr: Vehicle with n wheels. */
  public Vehicle(int n) {
     numWheels= n;
  }

  /** Return the vehicle's speed. */
  public abstract int getSpeed() {
     return 30;
  }
}
```

```
public class Bus extends Vehicle {
   private int numSeats;

   /** Constr: Bus with ns seats */
   public Bus(int ns) {
     numSeats= ns;
     numWheels= 4;
   }

   /** Return the bus's speed*/
   public @Override int getSpeed() {
      return 45;
   }
}
```

(a) 3 points List three separate problems with this implementation.

- The `Bus` constructor calls `super();` as its first statement. There is no `Vehicle` constructor with 0 parameters, so `super();` is a syntax error.
- Abstract method `getSpeed` is implemented in `Vehicle`.
- The constructor for `Bus` references `numWheels`, which is private.

(b) 5 points Below is class `Course`. Complete the constructor and method `equals`.

```
public class Course {
    private String name;          // The name of the course.
    private int courseID;         // The course id.

    /** Constructor: a course with name name, and course id id.*/
    public Course(String name, int id) {
        this.name= name;
        courseID= id;
     }

    /** Return true iff this and ob are of the same class and
        this and ob have the same name and id */
    public @Override boolean equals(Object ob)  {
        if (ob == null || ob.getClass() != getClass()) return false;
        Course c= (Course) ob;
        return name.equals(c.name) && courseID == c.courseID;
     }
}
```

(c) 10 points Below is class `Student` and interface `Offered`. Assume that method `equals` has already been implemented in class `Student`.

```
public class Student {
  private int studentID;

  public Student(int id){
    studentID= id;
  }
  /** Return true iff this and ob
    * are of the same class and
    * have the same student ID. */
  public @Override boolean
         equals(Object ob) {...}
}
```

```
public interface Offered {

  /** Add s to the course.
   *  Throw IllegalArgumentException
   *  if s is null or course is full. */
   public void addStudent(Student s)
          throws IllegalArgumentException;
}
```

Complete the constructor and method `equals` for `OfferedCourse`, below. Add any necessary methods. You can use `Arrays.equals` to compare arrays.

```
public class OfferedCourse extends Course implements Offered {
    private int maxS;             // The maximum capacity of the course.
    private int numS;             // The number of students in this course.
    private Student[] students;   // The students for this course
                                  // are in students[0...numS-1].

    /** Constructor: a course with name name, course id id, max capacity max,
     *  and no students */
    public OfferedCourse(String name, int id, int max) {
        super(name, id);
        maxS= max;
        students= new Student[maxS];
    }

   /** Return true iff this and ob are of the same class and
    *  have the same name, id, capacity, and students. */
    public @Override boolean equals(Object ob) {
        if (!super.equals(ob)) return false;
        OfferedCourse c= (OfferedCourse)ob;
        if (c.maxS != maxS) return false;
        return Arrays.equals(students, c.students);
    }
```

```
     /** Add s to the course. Throw an IllegalArgumentException
      *  if s is null or cannot be added. */
    public void addStudent(Student s) throws IllegalArgumentException {
        if (s == null || maxS == numS) {
            throw new IllegalArgumentException();
        }
        students[numS]= s;
        numS= numS + 1;
    }

}
```

(d) 3 points You decide to create a class `ConcurrentCourse`, below, for an implementation of a course in a multi-threaded environment.

```
public class ConcurrentCourse extends Course {
    private List<Student> students= new LinkedList<>(); // The students in this course.
    private int numStudents; // The number of students in this course.

    // Constructor, etc. goes here...

    /** Enroll s in this course. */
    public void enrollStudent(Student s) {
        synchronized(students) {students.add(s); numStudents++;}
    }

    /** Remove s from the course if currently enrolled. */
    public void removeStudent(Student s) {
        if (students.contains(s)) {
            synchronized(students) {students.remove(s); numStudents--;}
        }
    }
}
```

Is this class thread-safe? Explain your answer.

No. The call to `contains` inside `removeStudent` is not inside a synchronized block. This could lead to a race condition where the list is being modified by one thread as the other is calling `contains`, which for a linked list can lead to a NullPointerException. Alternatively, the student could have been removed by another thread between the check and the synchronized block. Therefore, calling `removeStudent` with the same argument with two different threads, could result in `numStudents` being decremented twice.

6. Data Structures (15 points)

(a) Stacks and Queues (2 points)

Write the printed output of the following two blocks of code:

```
Stack<Character> ls1= new Stack<>();
String s= "Hello"; }
for (int i= 0; i != s.length(); i++) { ls1.push(s.charAt(i)); }
while (!ls1.isEmpty()) { System.out.print(ls1.pop()); }
```

olleH

```
Queue<Character> ls2= new LinkedList<>()
String s1= "Hello";
for (int i= 0; i != s1.length(); i++) { ls2.add(s1.charAt(i)); }
while (!ls2.isEmpty()) { System.out.print(ls2.poll()); }
```

Hello

(b) Implementation Choice (3 points)

What data structure(s) would you use to implement each of the following abstractions to achieve the given time complexity? You may use several data structures in a single implementation.

Choices: ArrayList, LinkedList, Binary Search Tree, Red Black Tree, Binary Heap, HashTable

A stack with worst-case O(1) `push` and `pop` operations and expected O(1) `contains` operation. No answer. Use LinkedList for push/pop. If you add a HashTable to keep track of where values are? push/pop no longer worst case O(1).

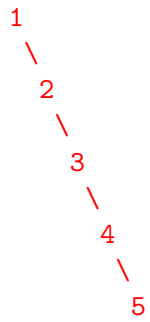
A dictionary or map with expected O(1) `add`, `contains`, and `update` operations. HashTable

A set that allows iteration over its entries in sorted order and $O(\log n)$ worst-case `add` and `remove` operations. Red Black Tree

(c) Binary Search Trees (6 points)

(i) 2 points Draw the Binary Search Trees that result from inserting the five integers on the left below, in order from left to right, into an initially empty tree. Then do the same thing for the five elements on the right below.

5, 7, 3, 2, 9 1, 2, 3, 4, 5



(ii) 1 point What advantage does a Red-Black Tree have over a Binary Search Tree?
A balanced binary tree guarantees $O(\log n)$ worst-case running time for the standard tree operations. A binary search tree has $O(n)$ worst-case time.

(iii) 3 points Consider the following class `BinaryTree` with method `isBST`.

```
public class BinaryTree {
      private int value;        // The value stored at this node.
      private BinaryTree left;  // Left child ---null if no left child.
      private BinaryTree right; // Right child ---null if no right child.
      ...
      /** Return true iff this tree is a Binary Search Tree. */
      public boolean isBst() {
         return (left == null  ||  left.value < value)  &&
                (right == null ||  right.value > value);
      }
}
```

There is a bug in method `isBST`. Draw a tree that, when used in the following test case, exposes this bug. That is, draw a tree that does not satisfy the Binary Search Tree invariant but for which method `isBST` still returns true.

```
public class TreeTester {
    @Test
    public void testIsBst() {
       BinaryTree yourTree= ...;
       assertFalse(yourTree.isBst());
    }
}
```

In a Binary Search Tree, the values of all nodes in the left subtree must be less than the root value. Method BST checks only that the left child's node is less.

```
\
12
```

(d) Heaps (4 points)

Consider the following min-heap, where the letters are values and the ints are priorities.

```
        (A, 3)
        /    \
    (B, 5)   (C, 7)
    /   \
(D, 6)   (E, 9)
```

Draw the heap after performing each of the following operations in order —the three operations are executed sequentially. For method pop, also write the value that is returned.

`updatePriority(A, 8);`

```
        (B, 5)
        /    \
    (D, 6)   (C, 7)
    /   \
(A, 8)   (E, 9)
```

`pop();` return B

```
        (D, 6)
        /    \
    (A, 8)   (C, 7)
    /
(E, 9)
```

`add(F, 2);`

```
        (F, 2)
        /    \
    (D, 6)   (C, 7)
    /   \
(E, 9)   (A, 8)
```

7. Graphs (16 points)

(a) Implementation (9 points)

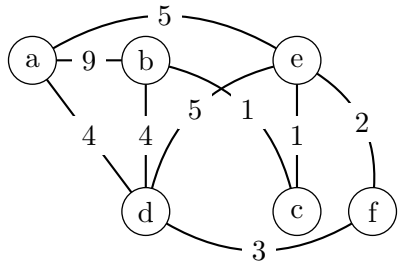
Assume that there exists a class `Node` with a constructor that has no parameters. Write a Java class `Graph` that implements a directed graph. Assume that the graph will be sparse and make your implementation as efficient as possible for a sparse graph. You need not write specs for the methods, but give the class invariant. Class `Graph` should have

1. Whatever data structures are needed to maintain the graph,
2. A constructor with no parameters, giving a graph with no nodes and no edges,
3. A public procedure `addNode(Node n)`, which adds a node to the graph that is not connected to any other node but does nothing if v is already in the graph, and
4. A public procedure `addEdge(Node u, Node v)`, which adds a directed edge from `u` to `v` (if both `u` and `v` are nodes of the graph) but does nothing if either node doesn't exist.

```
public class Graph {
  /** Each node of the graph has an entry in adjList, giving
    * as value the list of neighbors of this Node. */
  private HashMap<Node, LinkedList<Node>> adjList;

  public Graph() {
    adjList= new HashMap<Node, LinkedList<Node>>();
  }

  public void addNode(Node n){
    if (!adjList.containsKey(n)){
       adjList.put(n, new LinkedList<Node>());
    }
  }

  public void addEdge(Node u, Node v){
    if(adjList.containsKey(u) && adjList.containsKey(v)){
       adjList.get(u).add(v);
    }
  }
}
```

(b) Dijkstra's Shortest Path Algorithm (3 points)

Lets say that we start running Dijkstra's algorithm from node a in the graph shown below. Complete the table below with the estimated shortest path length to b after each iteration. We've filled in the first iteration for you, feel free to add as many additional columns as you need.



Iteration	1	2	3	4
length	9			

8, 8, 7

(d) Spanning Trees (4 points)

If an undirected connected graph has n nodes, how many edges does a spanning tree have? $n - 1$

Do Prim's and Kruskal's algorithms always produce the same spanning tree? If they do, please explain why in a few words. If not, when *do* they produce the same spanning tree?
They produce the same spanning tree when the edge weights are unique.

8. Hashing (9 points)

Consider an implementation of `HashSet<E>` using an array `b` of length 6 and linear probing. Do not be concerned with resizing. Use the hash function defined below.

```
public int hashCode(String s) { return s.length; }
```

(a) (3 points)

Draw array `b` after the following calls to method add of the HashSet. If there is no value for a specific index, leave it blank.

```
add("Gries"); add("Steven"); add("Ramya");
```

b

"Steven"	"Ramya"				"Gries"

(b) (4 points)

Draw b after the following calls —cross off (do not erase) values that are in the HashSet but considered removed.

```
remove("Gries"); add("Grant");
```

b

"Steven"	"Ramya"	"Grant"			"~~Gries~~"

(c) (2 points)

Define a hash function that would result in more collisions for the above items added.

Any function that hashes the four added elements to the same index works. One possible answer:

```
public int hashCode(String s) { return 0; }
```