Name: NetID:

Final Exam SOLUTION

CS 2110, May 17, 2016, 9:00 AM

	1	2	3	4	5	6	Total
Question	Short Answer	Object Oriented	Algorithms	Data Structures	Graphs	Concurrency	
Max	6	19	24	25	20	6	100
Score							
Grader							

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** at the top of **every** page! There are 6 questions on 17 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your booklet is 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 its good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam.

_____

(signature)

1. Short Questions (6 points)

(a) Hashing (2 points)

If you override `equals`, you should also override `hashCode` so that it satisfies what one property?

Objects that are equal should have the same hash code.

(b) Exceptions (4 points)

You are given the following code:

```
public static void m(int x){
    try {
        m2(x);
        System.out.println(1);
    } catch (ArithmeticException e) {
        System.out.println(2);
    } catch (Exception e) {
        System.out.println(3);
    }
}

public static void m2(int x) throws IOException {
    System.out.println(4);
    if (x==1)
        throw new IOException();
    if (x==0)
        throw new ArithmeticException();
    System.out.println(5);
}
```

(i) 2 points Write what is printed to the console by `m(1)`.

4
3

(ii) 2 points Write what is printed to the console by `m(0)`.

4
2

2. Object-Oriented Programming (19 points)

(a) Encapsulation (2 points)

Underline every illegal field access, similar to how Eclipse informs you that your code is invalid.

```
public class Super {
    private int priv;
    public int pub;

    public void printSuper(Super sup) {

        System.out.println("{priv=" + sup.priv + ", pub=" + sup.pub + "}");

    }
}

public class Sub extends Super {
    public void printSub(Sub sub) {

        System.out.println("{priv=" + sub.priv + ", pub=" + sub.pub + "}");

    }
}
```

(b) Sharing (2 points)

```
public class Baz {
    static int a= 0;
    int b;

    public Baz() {
        b= a;
        a= a+1;
    }

    public boolean equals(Baz that) {
        return b == that.b;
    }
}
```

Is the result of `(new Baz()).equals(new Baz())` always `true`?
No, it will evaluate to `true` **only** if two `Baz` objects are constructed at the same time in separate threads.

(c) Inheritance (3 points)

Consider the following code.

```
class A {
    private int x;

    public A(int x) {
        this.x= x;
    }

    public void m() {
        System.out.println(x-1);
    }
}

class B extends A{
    private int y;

    public B(int y) {
        super(y-1);
        this.y= y;
    }

    public void m() {
        System.out.println(y+1);
    }
}
```

(i) 1 point What is printed to the console by `(new A(3)).m()`?

2

(ii) 1 point What is printed to the console by `(new B(3)).m()`?

4

(iii) 1 point What is printed to the console by `((A)(new B(3))).m()`?

4

(d) Graphical User Interfaces (4 points)

Class `SearchEngine`, below, provides a search bar in a GUI. The GUI is displayed properly, but clicking the search button does nothing. Your task: make changes to `SearchEngine` so that it will listen for a click of the search button and call method `search` with the appropriate text if that event occurs.

Information to recall:

- `JTextField` has a method `String getText()`
- `JButton` has a method `void addActionListener(ActionListener)`
- `JButton` notifies its action listeners whenever it is clicked and only when it is clicked
- Interface `ActionListener` has a single method `void actionPerformed(ActionEvent)`

```
public class SearchEngine extends JFrame {
    private JTextField searchBar= new JTextField("Enter your search here");
    private JButton submit= new JButton("Search");

    public SearchEngine() {
        Container cp= getContentPane();

        setSize(300, 100);
        setResizable(false);


[add 'implements implements ActionListener' to class header]

        submit.addActionListener(this);


        cp.add(searchBar, BorderLayout.CENTER);
        cp.add(submit, BorderLayout.WEST);
        setVisible(true);
        pack();
    }

    private void search(String input) { ... }


    public void actionPerformed(ActionEvent e) {
         search(searchBar.getText());
    }


}
```

(e) Design (8 points)

You are employed by the Mathematics Department to help design their latest computation engine. While Java comes with a lot of mathematical functions, they operate on floating-point numbers, which are sometimes not precise enough. For example, both `float` and `double` cannot represent $\frac{1}{7}$ perfectly. You are asked to design class `Rational` representing rational numbers. However, the mathematicians need to be able to tradeoff between storage and precision. So you also need to design classes `ByteRational`, `ShortRational`, and `IntRational`, storing the numerator and denominator as `byte`s, `short`s, and `int`s respectively. Your design must satisfy the following additional requirements.

- `ByteRational`, `ShortRational`, and `IntRational` must each have a constructor taking a numerator followed by a denominator with type `byte`, `short`, and `int` respectively. The constructors must throw an `IllegalArgumentException` if the denominator is `0`.
- All `Rational`s must be immutable. That is, once a `Rational` is constructed, one can never change the value it represents. The type-checker must enforce this.
- One should be able to sort a list of `Rational`s. For example, the following code should result in the list $[\frac{871649}{786861}, \frac{5}{2}, \frac{6798}{876}]$:

```
List<Rational> numbers= Arrays.asList(new ByteRational(5, 2),
                                      new ShortRational(6798, 876),
                                      new IntRational(871649, 786861));
Collections.sort(numbers);
```

- `Rational`s must be able to be put into `HashSet`s and used as keys for `HashMap`s.
- There must be no uses of floating-point values.
- All access modifiers must be as restrictive as possible.
- There must be no wrapping/unwrapping between primitive types and their corresponding classes.
- There must be only one `instanceof` check and downcast between classes. There can be multiple casts between primitive types.

On the following blank page, provide the code for `Rational` and `IntRational` *only*. Be careful about overflow and about division! Don't worry too much about run-time efficiency; we care much more about simplicity of design.

```
public abstract class Rational implements Comparable<Rational> {
     protected abstract int|long getNumerator();
     protected abstract int|long getDenominator();

     public int compareTo(Rational that) {
          // must cast to long if getNumerator and getDenominator return int
          long diff = this.getNumerator() * that.getDenominator()
                      - that.getNumerator() * this.getDenominator();
          return diff < 0 ? -1 : diff == 0 ? 0 : 1;
     }
     public boolean equals(Object that) {
          return that instanceof Rational && 0 == compareTo((Rational)that);
     }
     public int hashCode() {
          // might need to cast to long
          // must ensure that (a*b)/(a*c) has same hash as b/c for a != 0
          return (int)(getNumerator() * Short.MAX_VALUE / getDenominator());
     }
}

public class IntRational extends Rational {
    private final int numerator;
    private final int denominator;

    public IntRational(int num, int den) {
        if (den == 0)
            throw new IllegalArgumentException(...);
        numerator= num;
        denominator= den;
    }

    @Override? protected int|long getNumerator() { return numerator; }
    @Override? protected int|long getDenominator() { return denominator; }
}
```

3. Algorithms (24 points)

(a) Sorting (14 points)

Consider the array [0, 5, 6, 3, 4, 9, 1, 2].

(i) 2 points If selection sort is called on that array, which two elements are sorted last, i.e. placed into their correct and final positions?

6 9
OR
9 6

(ii) 4 points Calling merge sort on that array recurses up to depth 3 (considering the first call to be at depth 0). Merge sort can be implemented to exploit parellelism. It can run each recursive call in its own thread, and then process the result after all those threads have finished. As such, it is possible for all calls to merge sort at depth 3 to complete simultaneously, and similarly for depths 2, 1, and 0. Below, fill in the contents of the array after each depth level has completed.

After Depth 3: [0,5,6,3,4,9,1,2]

After Depth 2: [0,5,3,6,4,9,1,2]

After Depth 1: [0,3,5,6,1,2,4,9]

After Depth 0: [0,1,2,3,4,5,6,9]

(iii) 8 points Complete the loop below and its initialization according to the provided precondition, invariant, and postcondition. Introduce variables when necessary.

```
// precondition: input is a non-null int[]

int[] acc= new int[input.length];
int i= 0;

// invariant: acc.length = input.length     AND
//            for j in 0..i-1, acc[j] = sum of input[0..j]

while (i < input.length OR acc.length) {

        acc[i]= (i == 0 ? 0 : acc[i-1]) + input[i];
        i= i+1;

}
// postcondition: acc.length = input.length    AND
//                for j in 0..acc.length-1, acc[j] = sum of input[0..j]
```

(b) Binary Search (2 points)

Consider the array b = [3, 4, 7, 9, 13, 16, 19]. Consider performing a binary search on b, using the binary search algorithm presented in lecture, searching for the rightmost occurrence of 7. What elements from the array would the search look at? List the value of the elements below in the order they are looked at.
9 4 7

(c) Complexity (8 points)

Ross is building a path in his garden using giant slabs of slate. Ross can carry only one slab at a time. Thus for the first slab, he takes zero steps, but for the second slab he takes one step there and one step back, and for the third he takes two steps there and three steps back. Thus, for a straight path of length n, Ross takes $T(n)$ steps where $T(n)$ is defined as follows:

$$T(n) = 0 + 2 + 4 + 6 + \cdots + 2(n-1) = \begin{cases} 0 & \text{for } n = 0 \\ T(n-1) + 2(n-1) & \text{for } n > 0 \end{cases}$$

(i) 4 points Given $k > 0$, assume that $T(n) = n(n-1)$ for $n = 0, 1, 2, ..., k-1$. Under those assumptions, prove $T(k) = k(k-1)$.

By definition, $T(k) = T(k-1) + 2(k-1)$. Using the assumption about $T(k-1)$, write this as $T(k) = (k-1)(k-2) + 2(k-1)$. The righthand side can be simplified to $k(k-1)$.

(ii) 2 points What is the tightest Big-O complexity class for $T(n)$? e.g. $O(n)$, $O(n \log n)$, $O(n^2)$
$O(n^2)$

(iii) 2 points Is it accurate to say that $T(n)$ is $O(n^4)$? Explain.
Yes. Being $O(n^2)$ means that n^2 is essentially an upper bound on $T(n)$. Since n^4 is essentially larger than n^2, that implies n^4 is also essentially an upper bound on $T(n)$. This means that $T(n)$ belongs to $O(n^4)$.

4. Data Structures (25 points)

(a) Linked Lists (6 points)

(i) 3 points For each of the following operations, indicate the worst-case run time for a list of size n. Your answer should be written in Big-O notation.

Operation	Time Complexity
Finding the size of a singly linked list without a `size` field	$O(n)$
Finding the size of a doubly linked list without a `size` field	$O(n)$
Getting the last element of a doubly linked list	$O(1)$
Searching for a value in a sorted doubly linked list	$O(n)$
Getting an element at a particular index of singly linked list	$O(n)$

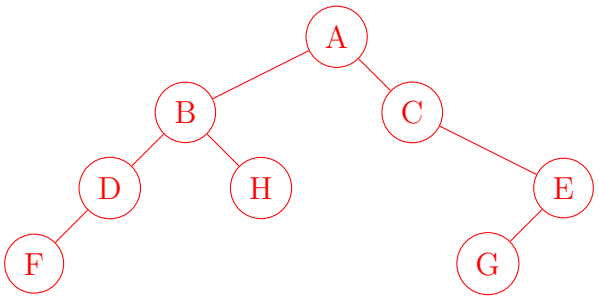
(ii) 1 point Describe in one sentence the difference between a doubly and singly linked list.
Singly linked lists only have a head field and forward pointers, whereas doubly linked lists also have a tail field and backward pointers.

(iii) 2 points Describe a situation in which a doubly linked list is a better choice than an `ArrayList`.
You frequently want to add elements to or remove elements from the beginning of the list.
OR
You frequently want to remove and/or insert elements as you iterate through the list.

(b) Trees (6 points)

(i) 3 points Given the following tree traversals, draw the tree.

Pre-Order	A B D F H C E G
In-Order	F D B H A C G E



(ii) 3 points Consider class `TreeNode` defined below.

```
public class TreeNode<E> {
    public TreeNode<E> left= null;
    public TreeNode<E> right= null;
    public E value;

    public TreeNode(E value) {
        this.value= value;
    }
}
```

Complete procedure `printPostOrder` below according to its specification.

```
/** Print tree on the console, one element per line, in post-order. */
public static <E> void printPostOrder(TreeNode<E> tree) {


    if (tree == null)
        return;
    printPostOrder(tree.left);
    printPostOrder(tree.right);
    System.out.println(tree.value);


}
```

(c) Binary Search Trees (7 points)

Consider class `BSTNode`, defined below, representing a binary search tree.

```
public class BSTNode {
    public int value;
    public BSTNode left= null;
    public BSTNode right= null;

    public BSTNode(int value) { this.value= value; }
}
```

(i) 3 points Complete method `smallest` below according to its specifications.

```
/** Return the smallest value in the tree with root n.
  * Precondition: n is not null
  * Takes time O(d) for a tree of maximum depth d */
public static int smallest(BSTNode n) {

    while (n.left != null)
        n= n.left;
    return n.value;
```

OR

```
    if (n.left == null)
        return n.value;
    return smallest(n.left);

}
```

(ii) 4 points Complete method `depth` according to its specification.

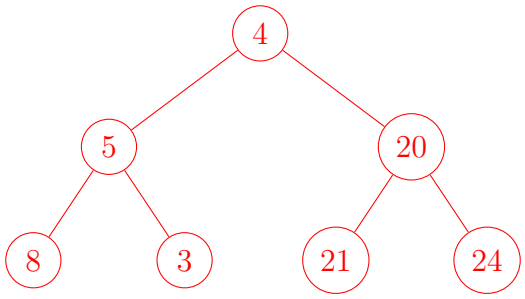
```
/** Return the depth of v in the tree with root n.
  * If v is not in n, return -1.
  * Takes time O(d) for a tree of maximum depth d */
public static int depth(int v, BSTNode n) {

    int d= 0;
    while (n != null && n.value != v) {
        d= d + 1;
        n= v < n.value ? n.left : n.right;
    }
    return n == null ? -1 : d;

}
```

(d) Heaps (6 points)

Given the following array of `int`s, answer parts (i) and (ii) below.

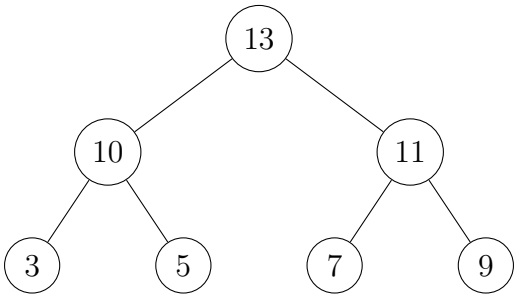
0	1	2	3	4	5	6
4	5	20	8	3	21	24

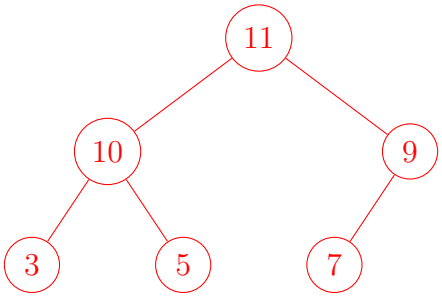
(i) 2 points Represent the array in tree form like one does with heaps.



(ii) 1 point Is the tree a valid **min**-heap? Explain why or why not.
No, because 3 is a child of 5, and all children in a **min**-heap should be greater than their parents.

(iii) 3 points For the **max**-heap given below, draw the tree resulting after polling once from the heap.





5. Graphs (20 points)

(a) Search (8 points)

In the game "Six degrees of Kevin Bacon", one tries to guess the shortest path between actor Kevin Bacon and another actor via the movies in which they have acted. For example, Mindy Kaling acted in "License to Wed" with Roxanne Hart, who acted in "The Little Sister" with Kevin Bacon, so Mindy Kaling's Kevin Bacon number is 2.

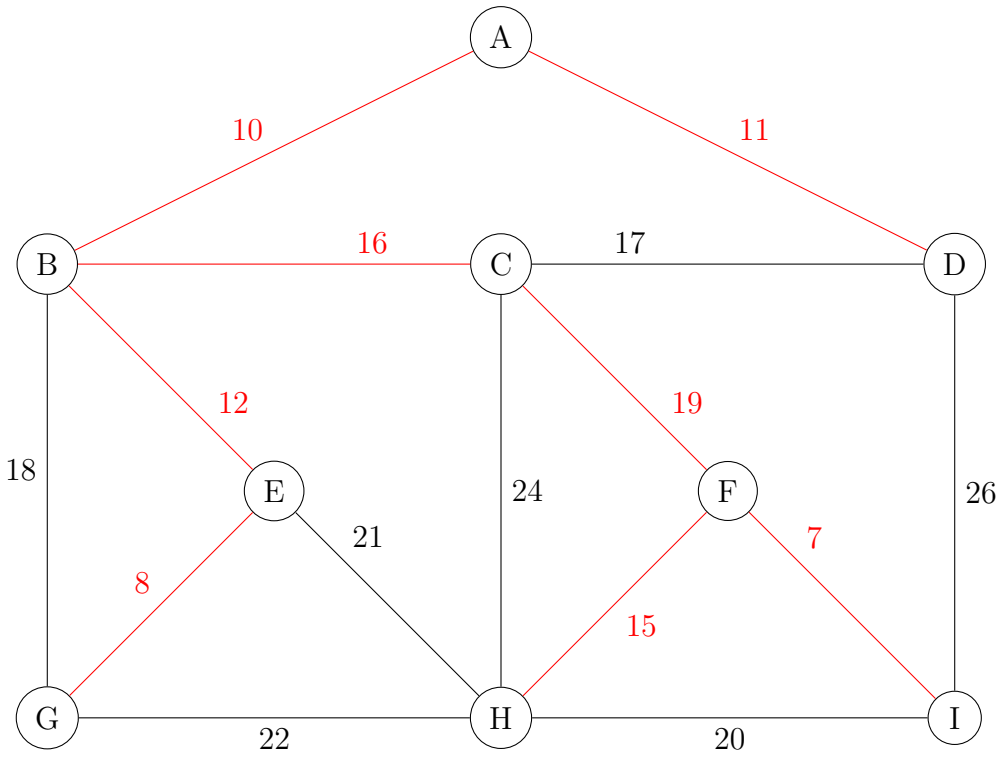
Complete the following code to calculate an actor's Kevin Bacon number. Your code must take time proportional to the number of *actors* in the worst case. This is a version of an algorithm you already know, so we do not give the invariant that explains local variables work and depths.

```
public class Actor {
    public String name; // not null
    public Set<Actor> coactors; // not null
    public boolean isKevinBacon() { return name.equals("Kevin Bacon"); }
}

/** Return actor's Kevin Bacon number, or null if no such number exists.
  * Precondition: actor is not null.
  * Requirement: worst-case time is proportional to the number of actors. */
public Integer KevinBaconNumber(Actor actor) {
    Queue<Actor> work= new LinkedList<Actor>();
    Map<Actor,Integer> depths= new HashMap<Actor,Integer>();
    work.add(actor);
    depths.put(actor, 0);
    while (!work.isEmpty()) {
        Actor a= work.poll();


        int depth= depths.get(a);
        if (a.isKevinBacon()) return depth;
        for (Actor c : a.coactors) {
            if (!depths.containsKey(c)) {
                depths.put(c, depth + 1);
                work.add(c);
            }
            // no need to handle else-case because we're using BFS
            // but it's OK if they do, so long as they don't add unnecessary work
        }


    }
    return null;
}
```

(b) Spanning Trees (4 points)

Given the graph below, calculate the minimum spanning tree using *either* Prim or Kruskal. Circle the weight of the edges that belong to the minimum spanning tree. In addition,

- If using Prim, start with node $v_0 = A$ in the tree. In the empty space below, write the order in which vertices are added to the tree.
- If using Kruskal, write down the weight of edges in the order they are added to the forest.



Prim: A, B, D, E, G, C, F, I, H

Kruskal: 7, 8, 10, 11, 12

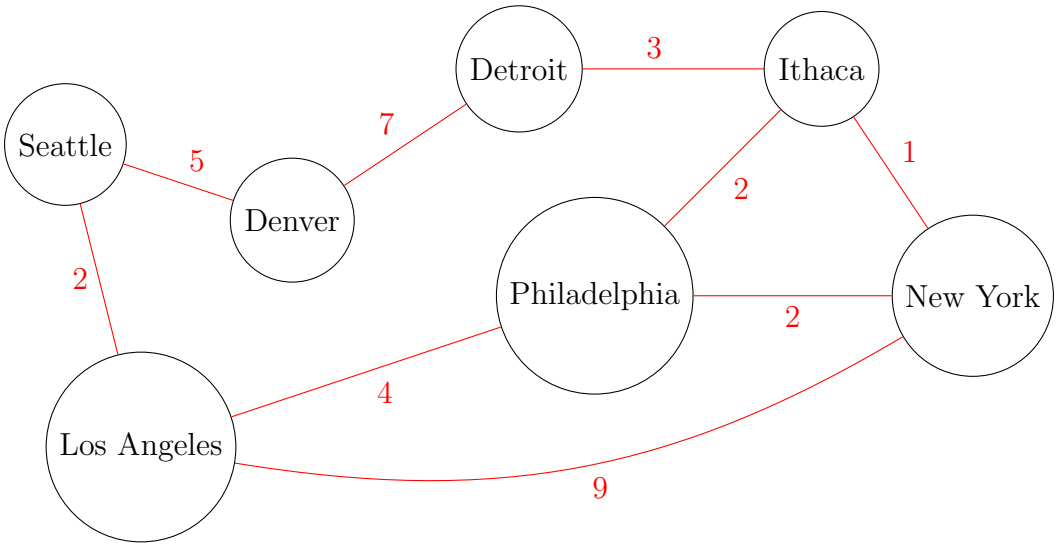
(c) Shortest Path (8 points)

You are a travel agent. You have to provide your client the fastest air trip from Ithaca to Seattle. For doing so, you are provided with the following table showing the flight times from one city to another:

From / To	Ithaca	Detroit	New York	Philadelphia	Denver	Los Angeles	Seattle
Ithaca	-	3	1	2	-	-	-
Detroit	3	-	-	-	7	-	-
New York	1	-	-	2	-	9	-
Philadelphia	2	-	2	-	-	4	-
Denver	-	7	-	-	-	-	5
Los Angeles	-	-	9	4	-	-	2
Seattle	-	-	-	-	5	2	-

(i) 1 point Explain why one can quickly look at this table and tell that the corresponding graph is undirected.
Yes, because it is symmetric, meaning equal to its own transpose.

(ii) 2 points Draw the graph on the following figure and label the edges with their weights.



(iii) 5 points Use Dijkstra's algorithm to find the shortest flight time from Ithaca to Seattle. Write down the nodes in the order you add them to the settled set. For simplicity, ignore any time spent on the ground.

Ithaca New-York Philadelphia Detroit Los-Angeles Seattle Denver

6. Concurrency (6 points)

(a) Shared Memory (3 points)

Assume a program has two threads, which execute the code in the table below. They reference shared variable `public int x`, which initially has value `0`.

Thread A	Thread B

`for (int i= 0; i < 10; i++) {`
`  x= x + 1;`
`}`
	`for (int i= 0; i < 10; i++) {`
	`  x= x - 1;`
	`}`

Do you know for sure what value `x` will have when the program terminates? If so, write the value `x` will have. If not, explain why you can't guarantee what the value is.
No. If both threads read `x` as x, and Thread A writes $x+1$ to `x`, then Thread B will write $x-1$ to `x` even though x is outdated, which effectively misses one of the increments by Thread A.

(b) Synchronization (3 points)

You and your partner are working on a program that processes a large amount of data. To speed things up, your partner makes the program multithreaded. Unfortunately, after changing your code to be multithreaded, the program seems to take even longer! In fact, you haven't been able to finish processing a single set of data when running with more than one thread. A portion of the code is below.

```
public class DataObject {
    /** Precondition: one != two */
    public static update(DataObject one, DataObject two) {
        synchronized(one) {
            synchronized(two) {
                // do updating ...
            }
        }
    }
}
```

Why isn't the program finishing when there are multiple threads? Note that there are multiple possible explanations; just provide one.
Deadlock: another thread could be synchronizing on `one` and `two` but in the opposite order.