CSE 143 Sample Final Exam #5 Key

1.

Statement	Output
`var1.one();`	`Blue 1 / Green 1`
`var1.two();`	`error`
`var1.three();`	`Green 3`
`var2.one();`	`Blue 1 / Green 1 / Red 1`
`var2.two();`	`error`
`var2.three();`	`Red 2 / Yellow 2 / Yellow 3`
`var3.two();`	`error`
`var3.three();`	`Yellow 2 / Yellow 3`
`var4.one();`	`error`
`((Blue) var1).one();`	`Blue 1 / Green 1`
`((Yellow) var1).two();`	`error`
`((Red) var2).three();`	`Red 2 / Yellow 2 / Yellow 3`
`((Yellow) var2).two();`	`Red 2 / Yellow 2`
`((Green) var4).three();`	`Green 3`
`((Yellow) var4).one();`	`error`

2.

```java
public class StudentTicket extends Ticket implements Comparable<StudentTicket> {
    private boolean honors;

    public StudentTicket(double price, boolean honors) {
        super(price, 14);
        this.honors = honors;
    }

    public double getPrice() {
        double price = super.getPrice() / 2;
        if (honors) {
            price = Math.max(0, price - 5.00);
        }
        return price;
    }

    public boolean isHonorStudent() {
        return honors;
    }

    public void setPromotionCode(String code) {
        super.setPromotionCode(code + " (student)");
    }

    public int compareTo(StudentTicket other) {
        if (getPrice() != other.getPrice()) {
            return (int) Math.signum(getPrice() - other.getPrice());
        } else {
            return getPromotionCode().compareTo(other.getPromotionCode());
        }
    }
}
```

3. Two solutions are shown.

```
// for loop solution
public void trimEnds(int k) {
    if (front != null) {
        // count size of list
        int size = 0;
        ListNode current = front;
        while (current != null) {
            current = current.next;
            size++;
        }

        if (size < 2 * k) {
            throw new IllegalArgumentException();
        } else if (size == 2 * k) {
            front = null;
        } else {
            // remove k from front
            for (int i = 0; i < k; i++) {
                front = front.next;
            }

            // move past middle part
            current = front;
            for (int i = 0; i < size - 2*k - 1; i++) {
                current = current.next;
            }
            current.next = null;  // remove k from back
        }
    }
}


// while loop solution
public void trimEnds(int k) {
    // count size of list
    int size = 0;
    ListNode current = front;
    while (current != null) {
        current = current.next;
        size++;
    }

    if (size < 2 * k) {
         throw new IllegalArgumentException();
    } else if (size == 2 * k) {
         front = null;
    } else if (k > 0) {
        // remove k from front
        int count = 0;
        while (count < k) {
            front = front.next;
            count++;
        }

        // move past middle part
        current = front;
        while (count < size - k - 1) {
            current = current.next;
            count++;
        }

        // remove k from back
        current.next = null;
    }
}
```

```
// inchworm solution (two node pointers) with size method
public void trimEnds(int k) {
    int size = getSize(front);
    if (size < 2 * k) {
        throw new IllegalArgumentException();
    } else if (size == 2 * k) {
        front = null;
    } else {
        ListNode current = front;   // old front
        for (int i = 1; i <= k; i++) {
            front = front.next;
        }

        ListNode current2 = front;  // new front
        while (current2 != null && current2.next != null) {
            current = current.next;
            current2 = current2.next;
        }
        current.next = null;
    }
}

private int getSize(ListNode node) {
    if (node == null) { return 0; }
    else { return 1 + getSize(node.next); }
}
```

```
// inchworm solution without size method - uglier
public void trimEnds(int k) {
    if (k <= 0) return;

    ListNode current = front;   // old front
    for (int i = 1; i <= k; i++) {
        if (front == null) { throw new IllegalArgumentException(); }
        front = front.next;
    }

    ListNode current2 = front;  // new front
    int count = 1;
    while (current2 != null && current2.next != null) {
        if (current == null) { throw new IllegalArgumentException(); }
        current = current.next;
        current2 = current2.next;
        count++;
    }

    if (count < k) {
        throw new IllegalArgumentException();
    } else if (count == k) {
        front = null;
    } else {
        current.next = null;
    }
}
```

4.

(a) Indexes examined: 7, 11, 9, 8 Value returned: -10

(b) `{22, 88, 44, 33, 77, 66, 11, 55}`
{**11**, 88, 44, 33, 77, 66, **22**, 55}
{11, **22**, 44, 33, 77, 66, **88**, 55}
{11, 22, **33**, **44**, 77, 66, 88, 55}

(c)

```
{22,  88,   44,  33,   77,  66,  11,  55}
{22,  88,   44,  33}  {77,  66,  11,  55}       split
{22,  88}  {44,  33}  {77,  66} {11,  55}       split
{22} {88}  {44} {33}  {77} {66} {11} {55}       split
{22,  88}  {33,  44}  {66,  77} {11,  55}       merge
{22,  33,   44,  88}  {11,  55,  66,  77}       merge
{11,  22,   33,  44,   55,  66,  77,  88}       merge
```

5. (a)

```
                          +------+
                          | Hill |
                          +------+
                       /            \
                    /                  \
          +-------+                    +------+
          | Barbs |                    | Stat |
          +-------+                    +------+
          /      \                   /
         /        \                 /
 +-------+      +--------+       +-------+
 | Amund |      | Dudley |       | Lopez |
 +-------+      +--------+       +-------+
                  /      \              \
                 /        \              \
           +--------+  +------+      +------+
           | Dragic |  | Frye |      | Rich |
           +--------+  +------+      +------+
                                      /
                                     /
                                 +------+
                                 | Nash |
                                 +------+
```

(b)

Pre-order: Hill, Barbs, Amund, Dudley, Dragic, Frye, Stat, Lopez, Rich, Nash

In-order: Amund, Barbs, Dragic, Dudley, Frye, Hill, Lopez, Nash, Rich, Stat

Post-order: Amund, Dragic, Frye, Dudley, Barbs, Nash, Rich, Lopez, Stat, Hill

6.

```
public void flip() {
    overallRoot = flip(overallRoot);
}

private IntTreeNode flip(IntTreeNode node) {
    if (node == null) {
        return null;
    } else {
        IntTreeNode temp = node.left;
        node.left = flip(node.right);
        node.right = flip(temp);
        return node;
    }
}

// alternative private method for above solution
private IntTreeNode flip(IntTreeNode node) {
    if (node != null) {
        IntTreeNode temp = flip(node.left);
        node.left = flip(node.right);
        node.right = temp;
    }
    return node;
}



// evil non x=change(x) solution
public void flip() {
    flip(overallRoot);
}

private void flip(IntTreeNode node) {
    if (node != null) {
        IntTreeNode temp = node.left;
        node.left = node.right;
        node.right = temp;
        flip(node.left);
        flip(node.right);
    }
}



// "we forgot to disallow creating new nodes" solution
public void flip() {
    overallRoot = flip(overallRoot);
}

private IntTreeNode flip(IntTreeNode node) {
    if (node == null) {
        return null;
    } else {
        return new IntTreeNode(node.data, flip(node.right), flip(node.left));
    }
}
```

7. Two solutions are shown.

```
// "change start to end when you find it" solution
public boolean hasPath(int start, int end) {
    return hasPath(overallRoot, start, end);
}

private boolean hasPath(IntTreeNode node, int start, int end) {
    if (node == null) {
        return false;
    } else {
        if (node.data == start) {
            start = end;   // remember that we have seen start by setting it to end
        }
        return (node.data == start && node.data == end) ||
               hasPath(node.left, start, end) ||
               hasPath(node.right, start, end);
    }
}



// "boolean flag for seeing the start value" solution
public boolean hasPath(int start, int end) {
    return hasPath(overallRoot, start, false, end);
}

private boolean hasPath(IntTreeNode node, int start, boolean seenStart, int end) {
    if (node == null) {
        return false;
    } else {
        seenStart = seenStart || node.data == start;
        boolean seenEnd = seenStart && node.data == end;
        return (seenStart && seenEnd) ||
               hasPath(node.left, start, seenStart, end) ||
               hasPath(node.right, start, seenStart, end);
    }
}



// "two helper methods" solution
public boolean hasPath(int start, int end) {
    return hasPath(overallRoot, start, end);
}

private boolean hasPath(IntTreeNode node, int start, int end) {
    if (node == null) {
        return false;
    } else if (node.data == start) {
        return contains(node, end);
    } else {
        return hasPath(node.left, start, end) || hasPath(node.right, start, end);
    }
}

private boolean contains(IntTreeNode node, int end) {
    if (node == null) {
        return false;
    } else if (node.data == end) {
        return true;
    } else {
        return contains(node.left, end) || contains(node.right, end);
    }
}
```