CSE143X Key to Final, Fall 2020

1. Preorder traversal 7, 2, 3, 8, 4, 1, 9, 6, 0, 5
   Inorder traversal 3, 8, 2, 4, 1, 7, 6, 9, 5, 0
   Postorder traversal 8, 3, 1, 4, 2, 6, 5, 0, 9, 7

2. One possible solution appears below.

```
public int digitMatch(int x, int y) {
    if (x < 0 || y < 0) {
        throw new IllegalArgumentException();
    }
    if (x < 10 || y < 10) {
        if (x % 10 == y % 10) {
            return 1;
        } else {
            return 0;
        }
    } else if (x % 10 == y % 10) {
        return 1 + digitMatch(x / 10, y / 10);
    } else {
        return digitMatch(x / 10, y / 10);
    }
}
```

3.

Statement	Output
var1.method2();	Spoon 2
var2.method2();	Bowl 2
var3.method2();	Bowl 2
var4.method2();	Pot 2
var5.method2();	compiler error
var6.method2();	Fork 2/Pot 2
var1.method1();	compiler error
var2.method1();	Bowl 1
var3.method1();	compiler error
var1.method3();	Pot 3/Spoon 2
var2.method3();	Pot 3/Bowl 2
var3.method3();	Pot 3/Bowl 2
var4.method3();	Pot 3/Pot 2
((Spoon)var1).method1();	Spoon 1
((Bowl)var3).method1();	Bowl 1
((Fork)var3).method3();	Pot 3/Bowl 2
((Fork)var5).method1();	compiler error
((Spoon)var5).method1();	runtime error
((Fork)var6).method2();	Fork 2/Pot 2
((Bowl)var6).method3();	runtime error

4. One possible solution appears below.

```
public static void alternatingReverse(Stack<Integer> s) {
    if (s.size() % 2 != 0) {
        throw new IllegalArgumentException();
    }
    Queue<Integer> q = new LinkedList<>();
    int oldSize = s.size();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    for (int i = 0; i < oldSize / 2; i++) {
        s.push(q.remove());
        q.add(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(q.remove());
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

5. One possible solution appears below.

```
public String acronymFor(List<String> words) {
    String acronym = "";
    for (String next : words) {
        acronym += next.toUpperCase().charAt(0);
    }
    return acronym;
}
```

6. One possible solution appears below.

```
public boolean hasPathSum(int sum) {
    return hasPathSum(overallRoot, sum);
}

private boolean hasPathSum(IntTreeNode root, int sum) {
    if (root == null) {
        return false;
    }
    if (root.data == sum) {
        return true;
    }
    int sum2 = sum - root.data;
    return hasPathSum(root.left, sum2) || hasPathSum(root.right, sum2);
}
```

7. One possible solution appears below.

```
public Map<String, Set<List<String>>> acronyms(Set<List<String>> lists) {
    Map<String, Set<List<String>>> result = new TreeMap<>();
    for (List<String> words : lists) {
        String acronym = acronymFor(words);
        if (!result.containsKey(acronym)) {
            result.put(acronym, new HashSet<>());
        }
        result.get(acronym).add(words);
    }
    return result;
}
```

8. One possible solution appears below.

```
public class ItemOrder implements Comparable<ItemOrder> {
    private String item;
    private int quantity;
    private int pricePer;

    public ItemOrder(String item, int quantity, int pricePer) {
        this.item = item;
        this.quantity = quantity;
        this.pricePer = pricePer;
    }

    public int getPrice() {
        return quantity * pricePer;
    }

    public String toString() {
        return "item #" +  item + ": " + quantity + "@" +
            moneyString(pricePer) + " = " + moneyString(getPrice());
    }

    private String moneyString(int cents) {
        return "$" + cents / 100 + "." + cents % 100 / 10 + cents % 10;
    }

    public int compareTo(ItemOrder other) {
        if (!item.equals(other.item)) {
            return item.compareTo(other.item);
        } else {
            return getPrice() - other.getPrice();
        }
    }
}
```

9. One possible solution appears below.

```
public void makeFull() {
    overallRoot = makeFull(overallRoot, 1);
}

private IntTreeNode makeFull(IntTreeNode root, int level) {
    if (root != null) {
        if (root.left == null && root.right != null) {
            root = new IntTreeNode(-level, root, root.right);
            root.left.right = null;
        } else if (root.left != null && root.right == null) {
            root = new IntTreeNode(-level, root.left, root);
            root.right.left = null;
        }
        root.left = makeFull(root.left, level + 1);
        root.right = makeFull(root.right, level + 1);
    }
    return root;
}
```

10. One possible solution appears below.

```
public void switchEvens(LinkedIntList other) {
    if (front != null && other.front != null) {
        ListNode temp = other.front;
        other.front = front;
        front = temp;
        temp = front.next;
        front.next = other.front.next;
        other.front.next = temp;
        ListNode curr1 = front.next;
        ListNode curr2 = other.front.next;
        while (curr1 != null && curr2 != null && curr1.next != null &&
               curr2.next != null) {
            temp = curr2.next;
            curr2.next = curr1.next;
            curr1.next = temp;
            temp = curr2.next.next;
            curr2.next.next = curr1.next.next;
            curr1.next.next = temp;
            curr1 = curr1.next.next;
            curr2 = curr2.next.next;
        }
    }
}
```