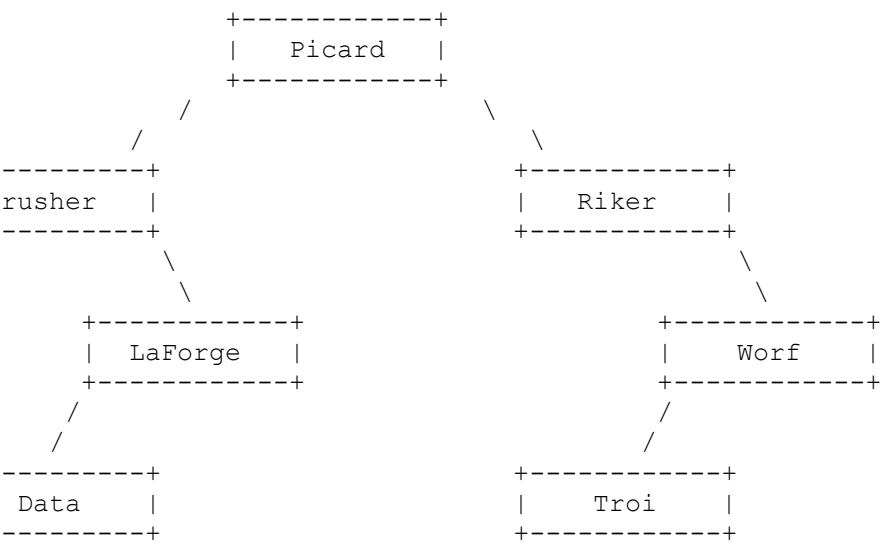
CSE143 Key to Final, Spring 2021

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

2. 

3.

Method Call	Contents of Set Returned
mystery(grid, 1)	[1, 2, 3, 4, 5]
mystery(grid, 2)	[4, 6, 8, 9]
mystery(grid, 4)	[4, 5, 6, 8]

4. One possible solution appears below.

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

5. Binary Trees, 10 points. One possible solution appears below.

```
public int oddPathSum() {
    return oddPathSum(overallRoot, 0);
}

public int oddPathSum(IntTreeNode root, int sum) {
    if (root == null) {
        return 0;
    } else {
        sum += root.data;
        int result = oddPathSum(root.left, sum) +
                     oddPathSum(root.right, sum);
        if (sum % 2 != 0) {
            result++;
        }
        return result;
    }
}
```

6. 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;
}
```

7. One possible solution appears below.

```
public class ClockTime implements Comparable<ClockTime> {
    private int hours;
    private int minutes;
    private String amPm;

    public ClockTime(int hours, int minutes, String amPm) {
        this.hours = hours;
        this.minutes = minutes;
        this.amPm = amPm;
    }

    public int compareTo(ClockTime other) {
        if (!amPm.equals(other.amPm)) {
            return amPm.compareTo(other.amPm);
        } else if (hours != other.hours) {
            return hours % 12 - other.hours % 12;
        } else {
            return minutes - other.minutes;
        }
    }

    public int getHours() {
        return hours;
    }

    public int getMinutes() {
        return minutes;
    }

    public String getAmPm() {
        return amPm;
    }

    public String toString() {
        String result = hours + ":";
        if (minutes < 10) {
            result += "0" + minutes;
        } else {
            result += minutes;
        }
        result += " " + amPm;
        return result;
    }
}
```

8. 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;
}
```

9. One possible solution appears below.

```
public LinkedIntList removeAlternatePairs() {
    LinkedIntList result = new LinkedIntList();
    if (front != null && front.next != null) {
        result.front = front;
        front = front.next.next;
        ListNode current1 = result.front.next;
        ListNode current2 = front;
        while (current2 != null && current2.next != null &&
               current2.next.next != null &&
               current2.next.next.next != null) {
            current1.next = current2.next.next;
            current2.next.next = current2.next.next.next.next;
            current2 = current2.next.next;
            current1 = current1.next.next;
        }
        current1.next = null;
    }
    return result;
}
```