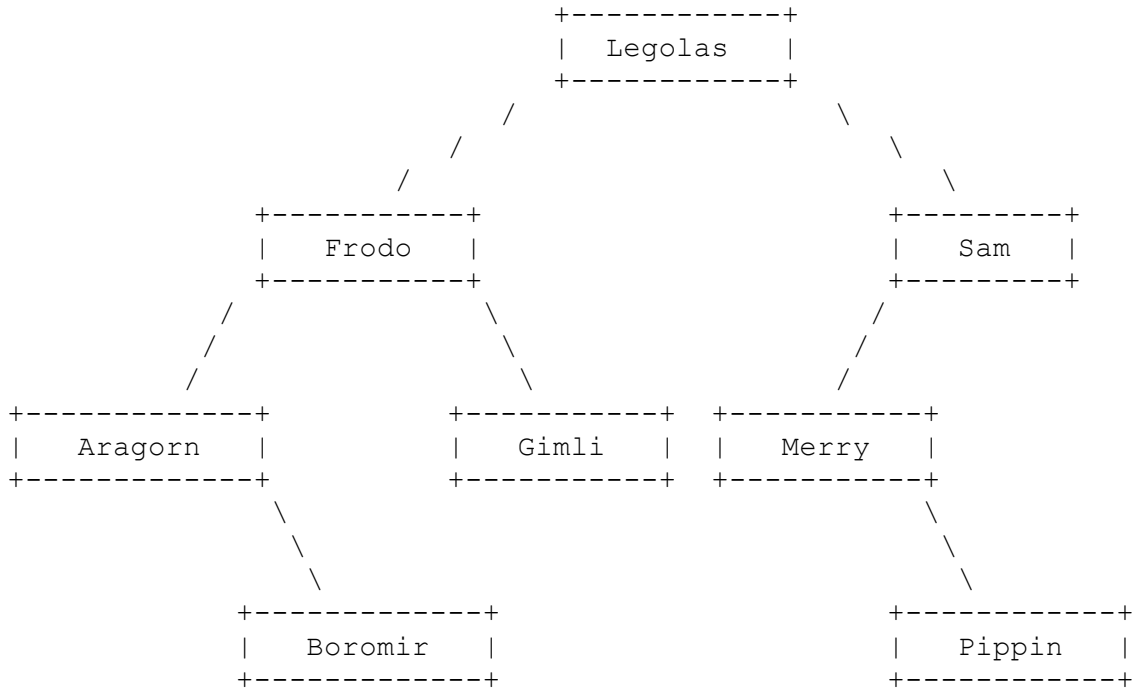
Key

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

2.



3.

Two-Dimensional Array	Contents of List Returned
[[1, 2, 3], [4, 5, 6]]	[8, 17]
[[3, 4], [1, 2, 3], [], [5, 6]]	[4, 8, 0, 6]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]	[8, 17, 26]

4.

Statement	Output
var1.method1();	Peak 1/Cliff 3/Peak 3
var2.method1();	Peak 1/Gorge 3
var3.method1();	Peak 1/Hill 3
var4.method1();	Peak 1/Gorge 3
var5.method1();	Peak 1/Peak 3
var6.method1();	error
var1.method2();	error
var2.method2();	Gorge 2
var3.method2();	error
var1.method3();	Cliff 3/Peak 3
var2.method3();	Gorge 3
var3.method3();	Hill 3
((Gorge)var6).method1();	error
((Cliff)var3).method2();	error
((Gorge)var4).method2();	Gorge 2
((Gorge)var3).method2();	error
((Hill)var3).method2();	Hill 2
((Gorge)var1).method1();	error
((Cliff)var4).method3();	Gorge 3
((Peak)var6).method3();	Cliff 3/Peak 3

5. One possible solution appears below.

```
public String toString() {
   return toString(overallRoot);
}

private String toString(IntTreeNode root) {
   if (root == null)
      return "empty";
   else if (root.left == null && root.right == null)
      return "" + root.data;
   else
      return "(" + root.data + ", " + toString(root.left)
   + ", " + toString(root.right) + ")";
}
```

6. One possible solution appears below.

```
public static Map<Point, Integer> sumStrings(Map<String, Point> data) {
    Map<Point, Integer> result = new HashMap<Point, Integer>();
    for (String s : data.keySet()) {
        Point p = data.get(s);
        if (!result.containsKey(p)) {
            result.put(p, s.length());
        } else {
            result.put(p, result.get(p) + s.length());
        }
    }
    return result;
}
```

7. Two possible solutions appear below.

```
public class TimeSpan implements Comparable<TimeSpan> {
    private int totalSeconds;

    public TimeSpan(int hours, int minutes, int seconds) {
        if (hours < 0 || minutes < 0 || seconds < 0) {
            throw new IllegalArgumentException();
        }
        totalSeconds = seconds + 60 * (minutes + 60 * hours);
    }

    public int hours() {
        return totalSeconds / 3600;
    }

    public int minutes() {
        return totalSeconds % 3600 / 60;
    }

    public int seconds() {
        return totalSeconds % 60;
    }

    public int totalSeconds() {
        return totalSeconds;
    }
```

```
    public String toString() {
        return hours() + ":" + minutes() / 10 + minutes() % 10 + ":" +
            seconds() / 10 + seconds() % 10;
    }

    public TimeSpan add(TimeSpan other) {
        TimeSpan result = new TimeSpan(0, 0, 0);
        result.totalSeconds = totalSeconds + other.totalSeconds;
        return result;
    }

    public int compareTo(TimeSpan other) {
        return this.totalSeconds - other.totalSeconds;
    }
}

public class TimeSpan implements Comparable<TimeSpan> {
    private int hours, minutes, seconds;

    public TimeSpan(int hours, int minutes, int seconds) {
        if (hours < 0 || minutes < 0 || seconds < 0) {
            throw new IllegalArgumentException();
        }
        int total = seconds + 60 * (minutes + 60 * hours);
        this.seconds = total % 60;
        this.minutes = total / 60 % 60;
        this.hours = total / 3600;
    }

    public int hours() {
        return hours;
    }

    public int minutes() {
        return minutes;
    }

    public int seconds() {
        return seconds;
    }

    public int totalSeconds() {
        return seconds + 60 * (minutes + 60 * hours);
    }

    public String toString() {
        return hours + ":" + minutes / 10 + minutes % 10 + ":" +
            seconds / 10 + seconds % 10;
    }

    public TimeSpan add(TimeSpan other) {
        return new TimeSpan(hours + other.hours, minutes + other.minutes,
                            seconds + other.seconds);
    }

    public int compareTo(TimeSpan other) {
        if (hours != other.hours)
            return hours - other.hours;
        else if (minutes != other.minutes)
            return minutes - other.minutes;
        else
            return seconds - other.seconds;
    }
}
```

8. One possible solution appears below.

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

private IntTreeNode tighten(IntTreeNode current) {
    if (current != null) {
        current.left = tighten(current.left);
        current.right = tighten(current.right);
        if (current.left == null && current.right != null) {
            current = current.left;
        } else if (current.left != null && current.right == null) {
            current = current.right;
        }
    }
    return current;
}
```

9. One possible solution appears below.

```
public void rearrange() {
    if (front != null && front.next != null) {
        ListNode current = front.next;
        while (current != null && current.next != null) {
            ListNode temp = current.next;
            current.next = current.next.next;
            temp.next = front;
            front = temp;
            current = current.next;
        }
    }
}
```