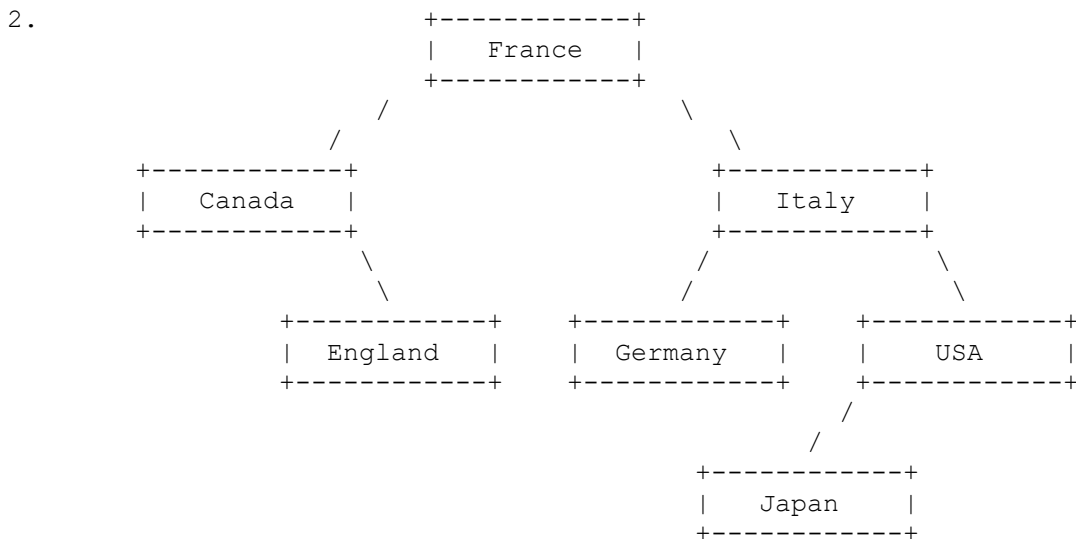


Solution to CSE143X Section #20 Problems

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



3.

Method Call	Contents of List Returned
mystery(grid, 2)	[13, 35]
mystery(grid, 3)	[23, 52, 22]
mystery(grid, 5)	[18, 3, 18, 61]

4. One possible solution appears below.

```

public Set<Point> removeSorted(Set<Point> data) {
    Set<Point> result = new HashSet<Point>();
    Iterator<Point> i = data.iterator();
    while (i.hasNext()) {
        Point p = i.next();
        if (p.getX() <= p.getY()) {
            result.add(p);
            i.remove();
        }
    }
    return result;
}

```

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

```

public void printLevel(int target) {
    if (target < 1) {
        throw new IllegalArgumentException();
    }
    System.out.print("nodes at level " + target + " =");
    printLevel(overallRoot, target, 1);
    System.out.println();
}

```

```

private void printLevel(IntTreeNode root, int target, int level) {
    if (root != null) {
        if (level == target) {
            System.out.print(" " + root.data);
        } else {
            printLevel(root.left, target, level + 1);
            printLevel(root.right, target, level + 1);
        }
    }
}

```

6. One possible solution appears below.

```

public Map<Integer, Set<String>> byUnits(Map<String, Integer> units,
    Map<String, Set<String>> enrollments) {
    Map<Integer, Set<String>> result = new TreeMap<Integer, Set<String>>();
    for (String student : enrollments.keySet()) {
        int sum = 0;
        for (String course : enrollments.get(student)) {
            sum += units.get(course);
        }
        if (!result.containsKey(sum)) {
            result.put(sum, new TreeSet<String>());
        }
        result.get(sum).add(student);
    }
    return result;
}

```

7. One possible solution appears below.

```

public class AdmissionsEntry implements Comparable<AdmissionsEntry> {
    private String ID;
    private int ratings;
    private double total;
    private boolean discuss;

    public AdmissionsEntry(String ID) {
        this.ID = ID;
        this.ratings = 0;
        this.total = 0.0;
        this.discuss = false;
    }

    public void rate(double rating) {
        ratings++;
        total += rating;
        if (rating >= 4) {
            discuss = true;
        }
    }

    public void flag() {
        discuss = true;
    }

    public double getRating() {
        if (ratings == 0) {
            return 0.0;
        } else {
            return total / ratings;
        }
    }
}

```

```

public String toString() {
    return ID + ": " + Math.round(100 * getRating()) / 100.0;
}

public int compareTo(AdmissionsEntry other) {
    if (discuss && !other.discuss) {
        return -1;
    } else if (!discuss && other.discuss) {
        return 1;
    } else if (getRating() > other.getRating()) {
        return -1;
    } else if (getRating() < other.getRating()) {
        return 1;
    } else {
        return ID.compareTo(other.ID);
    }
}
}

```

8. One possible solution appears below.

```

public void limitLeaves(int min) {
    overallRoot = limitLeaves(overallRoot, min);
}

private IntTreeNode limitLeaves(IntTreeNode root, int min) {
    if (root != null) {
        root.left = limitLeaves(root.left, min);
        root.right = limitLeaves(root.right, min);
        if (root.left == null && root.right == null && root.data <= min) {
            root = null;
        }
    }
    return root;
}

```

9. One possible solution appears below.

```

public LinkedIntList extractSmaller() {
    LinkedIntList other = new LinkedIntList();
    if (front != null && front.next != null) {
        if (front.data <= front.next.data) {
            other.front = front;
            front = front.next;
        } else {
            other.front = front.next;
            front.next = front.next.next;
        }
    }
    ListNode current1 = front;
    ListNode current2 = other.front;
    while (current1.next != null && current1.next.next != null) {
        if (current1.next.data <= current1.next.next.data) {
            current2.next = current1.next;
            current1.next = current1.next.next;
        } else {
            current2.next = current1.next.next;
            current1.next.next = current1.next.next.next;
        }
        current1 = current1.next;
        current2 = current2.next;
    }
    current2.next = null;
}
return other;
}

```