CSE143 Section #20 Problems

1. Binary Tree Traversals, 6 points. Consider the following tree.

```
                    +---+
                    | 2 |
                    +---+
                  /       \
                /           \
          +---+               +---+
          | 0 |               | 3 |
          +---+               +---+
         /     \                  \
        /       \                  \
     +---+     +---+               +---+
     | 5 |     | 7 |               | 4 |
     +---+     +---+               +---+
      /             \             /     \
     /               \           /       \
  +---+               +---+     +---+     +---+
  | 1 |               | 6 |     | 9 |     | 8 |
  +---+               +---+     +---+     +---+
```

Fill in each of the traversals below:

Preorder traversal ____

Inorder traversal ____

Postorder traversal ____

2. Binary Search Tree, 4 points. Draw a picture below of the binary search tree that would result from inserting the following words into an empty binary search tree in the following order: France, Canada, Italy, USA, Germany, England, Japan. Assume the search tree uses alphabetical ordering to compare words.

3. Collections Mystery, 5 points. Consider the following method:

```
public List<Integer> mystery(int[][] data, int n) {
    List<Integer> result = new LinkedList<Integer>();
    for (int i = 0; i < data.length; i++) {
        if (data[i][n] % n != 0) {
            result.add(data[i][n]);
        }
    }
    return result;
}
```

Suppose that a variable called grid has been declared as follows:

```
int[][] grid = {{25, 72, 13, 12, 24, 18}, {16, 14, 4, 23, 15, 3},
                {13, 29, 42, 52, 89, 18}, {25, 31, 35, 22, 66, 61}};
```

which means it will store the following 4-by-6 grid of values:

```
25      72      13      12      24      18
16      14      4       23      15      3
13      29      42      52      89      18
25      31      35      22      66      61
```

For each call below, indicate what value is returned. If the method call results in an exception being thrown, write "exception" instead.

Method Call	Contents of List Returned
mystery(grid, 2)	_________________________________
mystery(grid, 3)	_________________________________
mystery(grid, 5)	_________________________________

4. Collections Programming, 5 points. Write a method called removeSorted that takes a set of Point objects as a parameter and that removes the points from the set whose x/y coordinates are in sorted order (points whose x-value is less than or equal to its y-value), returning a set of the removed Point objects. For example, if a set called points contains the following values:

```
[[x=6,y=3], [x=9,y=7], [x=1,y=3], [x=7,y=7], [x=3,y=2], [x=4,y=3],
 [x=3,y=4], [x=-3,y=8], [x=108,y=15], [x=125,y=482], [x=1,y=42]]
```

then the call removeSorted(points) should leave the set storing:

```
[[x=6,y=3], [x=9,y=7], [x=3,y=2], [x=4,y=3], [x=108,y=15]]
```

and the set returned should store the following values:

```
[[x=1,y=3], [x=7,y=7], [x=3,y=4], [x=-3,y=8], [x=125,y=482],
 [x=1,y=42]]
```

The values in the returned set can appear in any order.

You may construct iterators and the set to be returned, but you are not allowed to construct other structured objects (no string, set, list, etc.).

5. Binary Trees, 10 points. Write a method called printLevel that takes an integer n as a parameter and that prints the values at level n from left to right. By definition, the overall root is at level 1, it's children are at level 2, and so on. For example, if a variable t stores this tree:

```
                    +----+
                    | 12 |
                    +----+
                  /        \
           +----+            +----+
           | 19 |            | 93 |
           +----+            +----+
          /      \                 \
     +----+      +----+            +----+
     | 11 |      | 14 |            | 15 |
     +----+      +----+            +----+
```

then the call:

```
    t.printLevel(3);
    t.printLevel(5);
```

would produce the output:

```
    nodes at level 3 = 11 14 15
    nodes at level 5 =
```

Notice that if there are no values at the level (e.g., level 5), your method should produce no output after the equals sign. You must exactly reproduce the format of this output. Your method should throw an IllegalArgumentException if passed a value for level that is less than 1.

You are writing a public method for a binary tree class defined as follows:

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    <constructors>
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class. You may not construct any extra data structures to solve this problem.

6. Collections Programming, 10 points. Write a method called byUnits that takes two maps as parameters and that returns a third map. The first parameter will be a map whose keys are course numbers and whose values are the units for that course, as in the following:

```
{CHEM237=4, CHEM241=3, CSE142=4, CSE143=5, CSE190=1, DANCE102=3,
 EE215=4, ENGL101=5, ENGL115=2, HIST101=3, MATH124=5, MATH125=5,
 MKTG301=4, PHIL100=5, PHYS121=4, PSYCH101=5}
```

The second parameter will be a map whose keys are student numbers (a string because of possible leading zeros) and whose values are sets of course numbers that the student with that ID is enrolled in, as in:

```
{0615694=[CHEM237, CHEM241, DANCE102, ENGL101], 1009195=[CSE143,
 EE215], 1021012=[MATH124, MKTG301, PSYCH101], 1035859=[CSE143, EE215,
 PHYS121], 1437611=[CSE142, ENGL115, HIST101, MATH125],
 1556919=[CHEM237, CHEM241, CSE142, PHIL100], 2199600=[CSE142, CSE190,
 MATH125, PHYS121], 2381328=[CSE142, CSE190, DANCE102, ENGL101],
 2440382=[CSE142, PSYCH101]}
```

The method computes the total units each student is taking. For example, the student 1556919 is taking CHEM237, CHEM241, CSE142, and PHIL100. The courses add up to 16 units (4 + 3 + 4 + 5). The map returned by the method should indicate for each number of units, the set of student IDs for students taking that number of units. Thus, there would be an entry for 16 that would include in its set the student ID 1556919. Assuming the two maps above are stored in variables called units and enrollments, respectively, then the call byUnits(units, enrollments) should return the following map:

```
{9=[1009195, 2440382], 13=[1035859, 2381328], 14=[1021012, 1437611,
 2199600], 15=[0615694], 16=[1556919]}
```

As in this example, the keys of the map returned by your method should appear in increasing numerical order. Your method should construct the new map and each of the sets contained in the map and can construct iterators but should otherwise not construct any other structured objects (no string, set, list, etc.). It should also not modify the two parameters and it should be reasonably efficient.

7. Comparable class, 20 points. Define a class called AdmissionsEntry that keeps track of information for an admissions candidate, how that candidate is rated by reviewers (real numbers between 0.0 and 5.0), and whether or not a candidate will be discussed. The class has the following public methods:

AdmissionsEntry(id)	constructs an AdmissionsEntry object with given ID
rate(rating)	records a rating for the candidate
flag()	indicates that the candidate should be discussed
getRating()	returns the average rating (0.0 if no ratings)
toString()	returns a String with ID and average rating

Below is an example for a candidate that has been reviewed four times:

```
AdmissionsEntry entry = new AdmissionsEntry("2222222");
entry.rate(3.75);
entry.rate(3.65);
entry.rate(3.8);
entry.rate(3.75);
entry.flag();
```

After these calls, the call entry.getRating() would return 3.7375 (the average of the ratings). The toString method should return a string composed of the ID, a colon, and the average rating rounded to 2 digits after the decimal point ("2222222: 3.74" for this example). If there are no ratings, then getRating and toString should indicate a rating of 0.0.

Each AdmissionsEntry object should keep track of whether that candidate should be discussed by the admissions committee. Any candidate who receives an individual score of 4.0 or higher should be discussed even if their average rating is below 4.0. Notice also that the flag method can be called, as in the example above, in which case the candidate will be discussed even if none of the ratings are 4.0 or higher.

The AdmissionsEntry class should implement the Comparable<E> interface. Define the method so that when sorted, a list of entries will have students to be discussed appearing first followed by students not to be discussed. Within those groups, students with higher average ratings should appear earlier in the list. Students with the same discussion status and the same average rating should appear in increasing order by ID. Recall that values considered "less" appear earlier in a sorted list.

8. Binary Trees, 20 points. Write a method called limitLeaves that takes an integer n as a parameter and that removes nodes from a tree to guarantee that all leaf nodes store values that are greater than n. For example, suppose a variable t stores a reference to the following tree:

```
                    +----+
                    | 13 |
                    +----+
                  /        \
           +----+            +----+
           | 18 |            | 10 |
           +----+            +----+
          /                 /      \
     +----+           +----+        +----+
     | 82 |           | 17 |        | 23 |
     +----+           +----+        +----+
    /      \                \             \
+----+    +----+           +----+        +----+
| 92 |    |  8 |           | 12 |        | 20 |
+----+    +----+           +----+        +----+
```

And we make the following call:

```
t.limitLeaves(20);
```

Then your method must guarantee that all leaf node values are greater than 20. So it must remove the leaves that store 8, 12, and 20. But notice that this creates a new leaf that stores 17 when its child is removed. This new leaf must also be removed. Thus, we end up with this tree:

```
                              +----+
                              | 13 |
                              +----+
                            /        \
                     +----+           +----+
                     | 18 |           | 10 |
                     +----+           +----+
                    /                       \
               +----+                         +----+
               | 82 |                         | 23 |
               +----+                         +----+
              /
          +----+
          | 92 |
          +----+
```

Notice that the nodes storing 13, 18, and 10 remain even though those values are not greater than 20 because they are branch nodes. Also notice that you might be required to remove nodes at many levels. For example, if the node storing 23 instead had stored the value 14, then we would have removed it as well, which would have led us to remove the node storing 10.

Your method should remove the minimum number of nodes that satisfies the constraint that all leaf nodes store values greater than the given n.

You are writing a public method for a binary tree class defined as follows:

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    <constructors>
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. You are not allowed to change the data fields of the existing nodes in the tree (what we called "morphing" in assignments 7 and 8), you are not allowed to construct new nodes or additional data structures, and your solution must run in O(n) time where n is the number of nodes in the tree.

9. Linked Lists, 20 points. Write a method of the LinkedIntList class called removeSmaller that removes the smaller value in each successive pair of numbers in the list, constructing and returning another LinkedIntList that contains the values removed. For example, suppose that a variable called list stores the following values:

```
[100, 68, 102, 138, 20, 105, -128, 100]
  |   |    |    |    |   |     |    |
  +---+    +----+    +---+     +----+
  pair      pair     pair       pair
```

As indicated, this list has four pairs. If the following call is made:

```
LinkedIntList result = list.removeSmaller();
```

then after the call, list and result would store the following values:

```
list:   [100, 138, 105, 100]
result: [68, 102, 20, -128]
```

Notice that for the first pair, the smaller value 68 has been moved to the second list while the larger value 100 has been retained in the original list. Similarly for the second pair, the value 102 has been moved to the second list while the value 138 has been retained in the original list.

If the two values in a pair are equal, you should remove the first one in the pair and retain the second one. If there is an odd number of values in the original list, the final value is retained in the original list. For example, if the list had instead stored these values:

```
[87, 145, 189, 146, 140, 61, 66]
```

then after the call list and result would store the following values:

```
list:   [145, 189, 140, 66]
result: [87, 146, 61]
```

You are writing a public method for a linked list class defined as follows:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

You are writing a method that will become part of the LinkedIntList class. You will need to call the zero-argument LinkedIntList constructor to construct the list to be returned and you may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. You are allowed to define your own variables of type ListNode, but you may not construct any new nodes, you may not use any auxiliary data structure to solve this problem (no array, ArrayList, stack, queue, String, etc), and your solution must run in O(n) time where n is the number of nodes in the list. You also may not change any data fields of the nodes. You MUST solve this problem by rearranging the links of the list.