CSE 143 Sample Final Exam #1

1. **Inheritance and Polymorphism**. Consider the following classes (`System.out.println` has been abbreviated as `S.o.pln`):

```
public class Cup extends Box {
    public void method1() {
        S.o.pln("Cup 1");
    }

    public void method2() {
        S.o.pln("Cup 2");
        super.method2();
    }
}

public class Pill {
    public void method2() {
        S.o.pln("Pill 2");
    }
}

public class Jar extends Box {
    public void method1() {
        S.o.pln("Jar 1");
    }

    public void method2() {
        S.o.pln("Jar 2");
    }
}

public class Box extends Pill {
    public void method2() {
        S.o.pln("Box 2");
    }

    public void method3() {
        method2();
        S.o.pln("Box 3");
    }
}
```

The following variables are defined:

```
Box var1 = new Box();
Pill var2 = new Jar();
Box var3 = new Cup();
Box var4 = new Jar();
Object var5 = new Box();
Pill var6 = new Pill();
```

In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a / b / c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, fill in the right-hand column with the phrase "error" to indicate this.

Statement	**Output**
`var1.method2();`	_____
`var2.method2();`	_____
`var3.method2();`	_____
`var4.method2();`	_____
`var5.method2();`	_____
`var6.method2();`	_____
`var1.method3();`	_____
`var2.method3();`	_____
`var3.method3();`	_____
`var4.method3();`	_____
`((Cup) var1).method1();`	_____
`((Jar) var2).method1();`	_____
`((Cup) var3).method1();`	_____
`((Cup) var4).method1();`	_____
`((Jar) var4).method2();`	_____
`((Box) var5).method2();`	_____
`((Pill) var5).method3();`	_____
`((Jar) var2).method3();`	_____
`((Cup) var3).method3();`	_____
`((Cup) var5).method3();`	_____

2. **Inheritance and Comparable Programming**. You have been asked to extend a pre-existing class `Point` that represents 2-D (x, y) coordinates. The `Point` class includes the following constructors and methods:

Constructor/Method	Description
`public Point()`	constructs the point (0, 0)
`public Point(int x, int y)`	constructs a point with the given x/y coordinates
`public void setLocation(int x, int y)`	sets the coordinates to the given values
`public int getX()`	returns the x-coordinate
`public int getY()`	returns the y-coordinate
`public String toString()`	returns a `String` in standard "(x, y)" notation
`public double distanceFromOrigin()`	returns the distance from the origin (0, 0), computed as the square root of $(x^2 + y^2)$

You are to **define a new class called `Point3D` that extends this class through inheritance**. It should behave like a `Point` except that it should be a 3-dimensional point that keeps track of a z-coordinate. You should provide the same methods as the superclass, as well as the following new behavior.

Constructor/Method	Description
`public Point3D()`	constructs the point (0, 0, 0)
`public Point3D(int x, int y, int z)`	constructs a point with given x/y/z coordinates
`public void setLocation(int x, int y, int z)`	sets coordinates to the given values
`public int getZ()`	returns the z-coordinate

Some of the existing behaviors from `Point` should behave differently on `Point3D` objects:

- When the original 2-parameter version of the `setLocation` is called, the 3-D point's x/y coordinates should be set as specified, and the z-coordinate should be set to 0.
- When a 3-D point is printed with `toString`, it should be returned in an "(x, y, z)" format that shows all three coordinates.
- A 3-D point's distance from the origin is computed using all three coordinates; it is equal to the square root of $(x^2 + y^2 + z^2)$.

You must also **make `Point3D` objects comparable to each other using the `Comparable` interface**. 3-D points are compared by x-coordinate, then by y-coordinate, then by z-coordinate. In other words, a `Point3D` object with a smaller x-coordinate is considered to be "less than" one with a larger x-coordinate. If two `Point3D` objects have the same x-coordinate, the one with the lower y-coordinate is considered "less." If they have the same x and y-coordinates, the one with the lower z-coordinate is considered "less." If the two points have the same x, y, and z-coordinates, they are considered to be "equal."

3. **Linked List Programming**. Write a method `reorder` that could be added to the `LinkedIntList` class from lecture and section. The method rearranges a list of integers into sorted order assuming that the list is already sorted by absolute value. Suppose a `LinkedIntList` variable `list` stores the following values:

```
[0, -3, 3, -5, 7, -9, -10, 10, -11, -11, -11, 12, -15]
```

Notice that the values are in sorted order if you ignore their signs. The call `list.reorder();` should reorder the values into sorted, non-decreasing order (including sign).

```
[-15, -11, -11, -11, -10, -9, -5, -3, 0, 3, 7, 10, 12]
```

Because the list is sorted by absolute value, you can solve this problem very efficiently. Your solution is required to run in O(*N*) time where *N* is the length of the list. Recall the `LinkedIntList` and node classes:

```
public class LinkedIntList {
    private ListNode front;

    methods
}

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

You may not call any methods of your linked list class to solve this problem, you may not construct any new nodes, and you may not use any auxiliary data structure to solve this problem (such as an array, `ArrayList`, `Queue`, `String`, etc). You also may not change any `data` fields of the nodes. You must solve this problem by rearranging the links of the list.

4. **Searching and Sorting**.

(a) Suppose we are performing a **binary search** on a sorted array called `numbers` initialized as follows:

```
// index          0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
int[] numbers = {-9, -6, -2, -1,  0,  1,  3,  4,  5,  7,  9, 10, 12, 19, 23, 26};

// search for the value 1
int index = binarySearch(numbers, 1);
```

Write the indexes of the elements that would be examined by the binary search (the `mid` values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in lecture and section.

- Indexes examined: _____
- Value Returned: _____

(b) Write the state of the elements of the array below after each of the first 3 passes of the outermost loop of the selection sort algorithm.

```
int[] numbers = {37, 29, 19, 48, 23, 55, 74, 12};
selectionSort(numbers);
```

(c) Trace the complete execution of the merge sort algorithm when called on the array below, similarly to the example trace of merge sort shown in the lecture slides. Show the sub-arrays that are created by the algorithm and show the merging of sub-arrays into larger sorted arrays.

```
int[] numbers = {37, 29, 19, 48, 23, 55, 74, 12};
mergeSort(numbers);
```

5. **Binary Search Trees**.

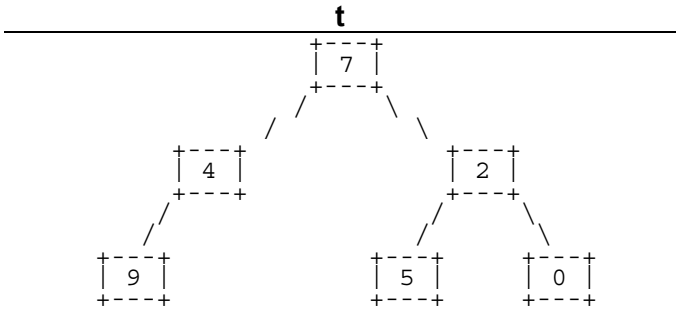
(a) Write the binary search tree that would result if these elements were added to an empty tree in this order:

- Kirk, Spock, Scotty, McCoy, Chekov, Uhuru, Sulu, Khaaaan!

(b) Write the elements of your tree above in the order they would be visited by each kind of traversal:

- Pre-order: _____
- In-order: _____
- Post-order: _____

6. **Binary Tree Programming**. Write a method `inOrderList` that could be added to the `IntTree` class from lecture and section. The method returns a list containing the sequence of values obtained from an in-order traversal of your binary tree of integers. For example, if a variable `t` refers to the following tree:



Then the call `t.inOrderList()` should return the following list:

```
[9, 4, 7, 5, 2, 0]
```

If the tree is empty, your method should return an empty list.

You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the `IntTree` class. Your method should not change the structure or contents of the tree.

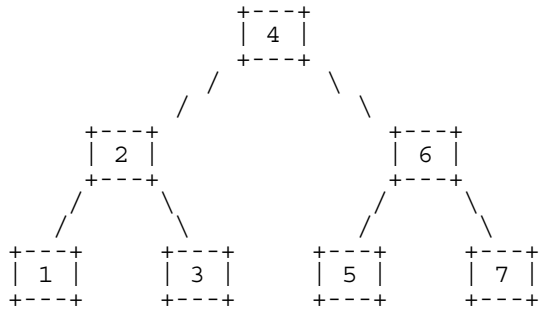
Recall the `IntTree` and `IntTreeNode` classes as shown in lecture and section:

```
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

    public IntTreeNode(int data) { ... }
    public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {...}
}

public class IntTree {
    private IntTreeNode overallRoot;

    methods
}
```

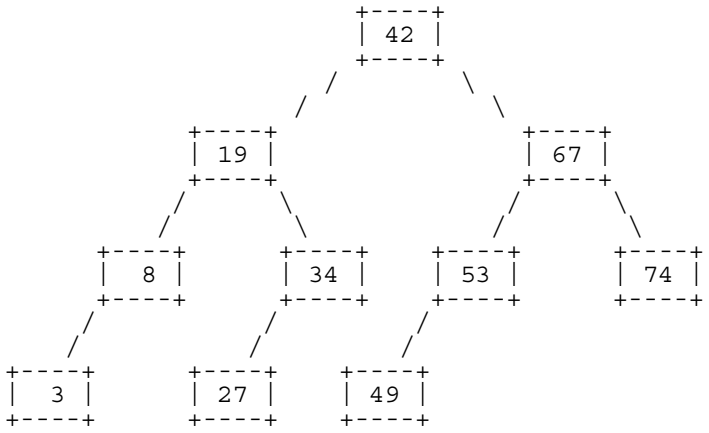
7. **Binary Tree Programming**. Write a method `construct` that could be added to the `IntTree` class from lecture and section. The method accepts a sorted array of integers as a parameter and constructs a balanced binary search tree containing those integers. The tree should be constructed so that for every node, either the left/right subtrees have the same number of nodes, or the left subtree has one more node than the right.

   For example, if you have an `IntTree` variable called `t` and an array called `a` storing values `[1, 2, 3, 4, 5, 6, 7]`, and call of `t.construct(a);` is made, `t` should store the tree below at left. If the array stores `[3, 8, 19, 27, 34, 42, 49, 53, 67, 74]`, the tree below at right is constructed.

First Example
[1, 2, 3, 4, 5, 6, 7]



Second Example
[3, 8, 19, 27, 34, 42, 49, 53, 67, 74]



   Notice that when it is not possible to have left and right subtrees of equal size, the extra value always ends up in the left subtree, as in the overall tree which has 5 nodes in the left subtree and 4 in the right.

   The new tree should replace any old tree. For full credit, your solution must run in O(*N*) time, where *N* is the number of elements in the array. You may assume that the values in the array appear in sorted order.

   You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class nor create any data structures such as arrays, lists, etc. You also may not alter the array that you are passed.