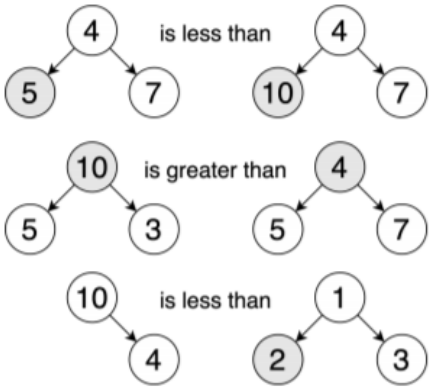
This question is about comparing binary trees.

To compare two binary tree nodes, we first compare their left subtrees, then compare their values (with compareTo), then compare their right subtrees. If all of these are equal, then the nodes are equal. Otherwise the first non-equal comparison in this order is the result of the comparison. To compare the subtrees, you should recurse and compare them in the same way as described here.

Regarding nulls (e.g. missing left/right children), a null subtree should compare equal to another null, or less than any non-null subtree.

Implement the BinaryTreeComparator class in BinaryTreeComparator.java. This has a single method compare which takes two trees x and y then returns −1, 0, +1 if x < y, x = y, or x > y (respectively). State the worst case time and space complexity (in Big-O notation) in the method's Javadoc.

Here are some examples of trees and how they compare. The highlighted node(s) are those which determine the comparison result. Take note of the last case where the null left subtree is less than the non-null left subtree.



Hints and notes:

• You may add private helper methods if needed, but you should not use any instance variables.

• You can assume that values stored within nodes are not null and implement Comparable (so you can use the compareTo method).

• You cannot assume the node itself will be non-null. Nulls should be compared as described above.

• You should not perform unnecessary comparisons. For example, if the left subtrees differ, you should not compare the right subtrees.

• Although the Comparator interface only requires the return value to be negative/zero/positive, here you must return exactly −1, 0, or +1.