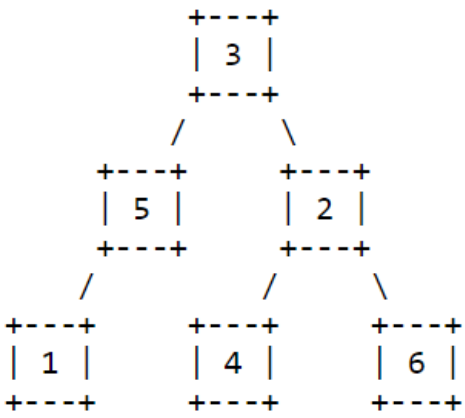
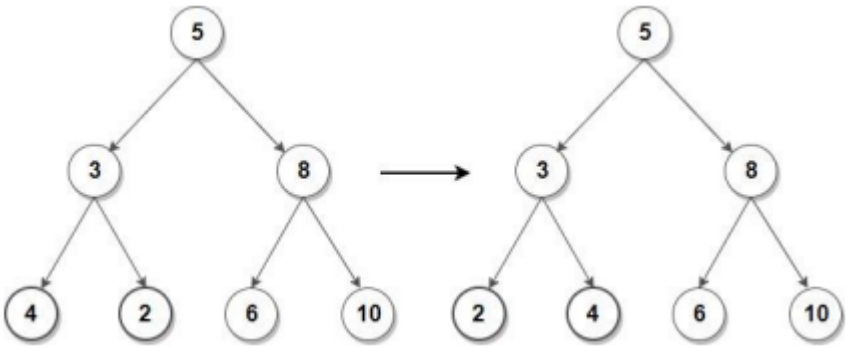
Problem 1: Traversal

Write the elements of the tree below in the order they would be seen by a pre-order, in-order, and post-order traversal.



Problem 2: More BSTs

Given a binary tree that is only one swap away from becoming a BST, convert it into a BST in a single traversal. For example, consider the binary tree shown on the left image below. Your code should swap2 and 4 to transform the tree into a BST. Write your code in BST.java in the function correctBST(). Your code should work on any given tree that is one step away from becoming a BST.



Hints:

- What traversal returns the nodes in sorted order?

- Keep track of the last visited node while traversing the tree to mark the BST violation

Problem3: Binary Trees

Review and understand the LinkedBinaryTree code provided to you. Make sure to understand all the functions as you will be asked to explain some individually. Once done, extend the BT code with the following functions

a- traversePreOrder() which traverses the tree in the order(root, left child, right child)
b- findElement(int x) which returns true if x is found in the Binary Tree, false otherwise
c- countOdd() which returns the number of odd elements in the tree
d- countEven() which returns the number of even elements in the tree
e- countInternalNodes() which counts the number of internal nodes in the tree
f- countExternalNodes() which counts the number of leaf nodes in the tree
g- calculateHeight() which returns the height of the tree starting from the Root element

Problem4: Priority Queue

Review and understand the Priority Queue code provided to you. Make sure to understand all the functions and solve the following:

a) In the main of PQTester.java, add a class "intComparator" that implements comparator and compares integers. Check the way "stringLengthComparator" is implemented as the logic is similar.

b) In the main of PQTester.java create PQ2 an instance of SortedPQ with an Integer key and Student value with intComparator. Insert the students' s1-s4 into PQ2 with their grades as keys then print the elements of the queue.

c) In SortedPQ.java add the boolean method exists () that takes in as input a key and a value. Your method should return true if the key already exists in the queue, and false otherwise.

d) Using PQ2 test your exists () method on both cases (false and true)