1 Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals of the following binary search tree. For all traversals, process child nodes left to right.

```
        10
       /  \
      3    12
     / \     \
    1   7     13
                \
                 15
```

```
DFS Pre-order: 10, 3, 1, 7, 12, 13, 15
DFS In-order: 1, 3, 7, 10, 12, 13, 15
DFS Post-order: 1, 7, 3, 15, 13, 12, 10
BFS: 10, 3, 12, 1, 7, 13, 15
```

2 Is This a BST?

(a) The following code should check if a given binary tree is a BST. However, for some trees, it returns the wrong answer. Give an example of a binary tree for which `brokenIsBST` fails.

```java
public static boolean brokenIsBST(TreeNode T) {
    if (T == null) {
        return true;
    } else if (T.left != null && T.left.val > T.val) {
        return false;
    } else if (T.right != null && T.right.val < T.val) {
        return false;
    } else {
        return brokenIsBST(T.left) && brokenIsBST(T.right);
    }
}
```

Here is an example of a binary tree for which `brokenIsBST` fails:

```
        10
       /  \
      5    15
     / \
    3   12
```

The method fails for some binary trees that are not BSTs because it only checks that the value at a node is greater than its left child and less than its right child, not that its value is greater than every node in the left subtree and less than every node in the right subtree. Above is an example of a tree for which it fails.

It is important to note that the method does indeed return true for every binary tree that actually is a BST (it correctly identifies proper BSTs).

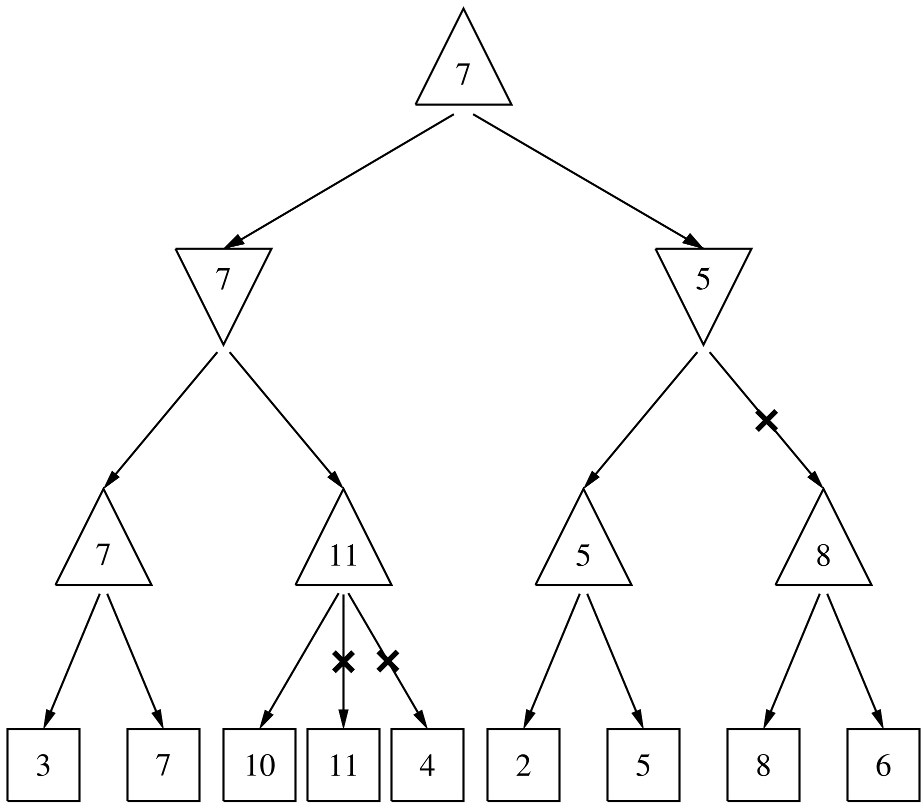
(b) Now, write `isBST` that fixes the error encountered in part (a).
Hint: You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful.

```
public static boolean isBST(TreeNode T) {
    return isBSTHelper(T, Integer.MIN_VALUE, Integer.MAX_VALUE);
}

public static boolean isBSTHelper(TreeNode T, int min, int max) {
    if (T == null) {
        return true;
    } else if (T.val < min || T.val > max) {
        return false;
    } else {
        return isBSTHelper(T.left, min, T.val)
               && isBSTHelper(T.right, T.val, max);
    }
}
```

3 Shall we play a game?

In the partial game tree below, we represent maximizing nodes as △; minimizing nodes as ▽; and nodes with static values as □. Determine the values for the nodes that would result from the minimax algorithm without pruning (write them inside the nodes), and then cross out branches that would not be traversed (would be pruned) as a result of alpha-beta pruning. Assume we evaluate children of a node from left to right.



4 Extra Binary Tree Practice: Sum Paths

Define a root-to-leaf path as a sequence of nodes from the root of a tree to one of its leaves. Write a method `printSumPaths(TreeNode T, int k)` that prints out all root-to-leaf paths whose values sum to k. For example, if T is the binary tree in the diagram below and k is 13, then the program will print out `10 2 1` on one line and `10 4 -1` on another.

```
      10
     /  \
    2    4
   / \    \
  5   1   -1
```

(a) Provide your solution by filling in the code below:

```java
public static void printSumPaths(TreeNode T, int k) {
    if (T != null) {
        sumPaths(T, k, "");
    }
}

public static void sumPaths(TreeNode T, int k, String path) {
    if (T.left == null && T.right == null && k == T.val) {
        System.out.println(path + T.val);
    } else {
        path += T.val + " ";
        if (T.left != null) {
            sumPaths(T.left, k - T.val, path);
        }
        if (T.right != null) {
            sumPaths(T.right, k - T.val, path);
        }
    }
}
```

(b) What is the worst case runtime of `printSumPaths` in terms of N, the number of nodes in the tree? What is the worst case runtime in terms of h, the height of the tree?

In the worst case, the height of the tree is N and we perform a string contatenation at each level. If we assume that all nodes in the tree have values bounded by some constant, then at level l we perform a string concatenation of a string of length l (the length of the path from the root to that node) and a string whose length is bounded by some constant. Since string concatenation is linear, we get a runtime of $1+2+\ldots+N = \Theta(N^2)$.

The worst case runtime in terms of h is when we have a complete binary tree. In this case, there are 2^h leaves, all at the bottom level of the tree. Each string concatenation on this level takes $\Theta(h)$ time (again assuming that the values in the tree are bounded by some constant). Thus the total runtime is $\Theta(h2^h)$, since there are at most 2^h non-leaf nodes and the string concatenation for these nodes takes $O(h)$ time.