```
public class P176Q08 {
 // A
 public static void printPostOrderNegative(BinNode<Integer> root) {
  if (root == null) return;
  printPostOrderNegative(root.getLeft());
  printPostOrderNegative(root.getRight());
  if (root.getValue() < 0) System.out.print(root.getValue() + " ");
 }
 // B
 public static <T> void printAllLeft(BinNode<T> root) {
  if (root == null) return;
  if (root.hasLeft()) System.out.print(root.getLeft().getValue() + " ");
  printAllLeft(root.getLeft());
  printAllLeft(root.getRight());
 }
 // C
 public static void main(String[] args) {
  // ????????????
 }
}
public class P176Q09 {
 public static void update(BinNode<Character> root) {
  if (root == null) return;
  if (root.getValue() == 'z') root.setValue('a');
  else root.setValue((char)(root.getValue() + 1));
  update(root.getLeft());
  update(root.getRight());
 }
}
public class P176Q10 {
 public static <T> void printLeaves(BinNode<T> root) {
  if(root == null) return;
  printLeaves(root.getLeft());
  if (!root.hasLeft() && !root.hasRight()) System.out.print(root.getValue() + " ");
  printLeaves(root.getRight());
 }
}
public class P176Q11 {
 public static void printSpecificNodes(BinNode<Integer> root) {
  if (root == null) return;
  if (root.getValue()%2 == 0) {
   if(!root.hasLeft() || root.getLeft().getValue()%2 == 0) {
    if (!root.hasRight() || root.getRight().getValue()%2 == 0)
     System.out.print(root.getValue() + " ");
   }
  }
  printSpecificNodes(root.getLeft());
  printSpecificNodes(root.getRight());
 }
}
public class P176Q12 {
 public static int getCount10till100(BinNode<Integer> root) {
  if (root == null) return 0;
  if (root.getValue() >= 10 && root.getValue() < 100) return 1 + getCount10till100(root.getLeft()) + getCount10till100(root.getRight());
  return getCount10till100(root.getLeft()) + getCount10till100(root.getRight());
 }
}
public class P176Q13 {
 public static void printClosestSon(BinNode<Integer> root) {
  if (root == null || !root.hasLeft() && !root.hasRight()) return;
  if (!root.hasRight()) System.out.print(root.getLeft().getValue() + " ");
  else if (!root.hasLeft()) System.out.print(root.getRight().getValue() + " ");
  else if  (Math.abs(root.getValue()-root.getRight().getValue()) < Math.abs(root.getValue()-root.getLeft().getValue()))
   System.out.print(root.getRight().getValue() + " ");
  else System.out.print(root.getLeft().getValue() + " ");

  printClosestSon(root.getLeft());
  printClosestSon(root.getRight());
 }
}
public class P176Q14 {
 public static <T> int getLeafCount(BinNode<T> root) {
  if (root == null) return 0;
  if (!root.hasLeft() && !root.hasRight()) return 1;
  return getLeafCount(root.getLeft()) + getLeafCount(root.getRight());
 }
}
public class P176Q15 {
 public static int getBetweenCount(BinNode<Integer> root, int low, int high) {
  if (root == null) return 0;
  if (root.getValue() > low && root.getValue() < high)
   return 1 + getBetweenCount(root.getLeft(), low, high) + getBetweenCount(root.getRight(), low, high);
  return getBetweenCount(root.getLeft(), low, high) + getBetweenCount(root.getRight(), low, high);
 }
}
public class P176Q16 {
 public static int getParentsSum(BinNode<Integer> root) {
  if (root == null) return 0;
  if (root.hasLeft() && root.hasRight())
   return root.getValue() + getParentsSum(root.getLeft()) + getParentsSum(root.getRight());
  return getParentsSum(root.getLeft()) + getParentsSum(root.getRight());
 }
}
public class P177Q17 {
 public static int getParentsNoLeaves(BinNode<Integer> root) {
  if (root == null) return 0;
  if (root.hasLeft() && root.hasRight()) {
   if (root.getLeft().hasLeft() || root.getLeft().hasRight()) {
    if (root.getRight().hasLeft() || root.getRight().hasRight()) {
```

```
     return 1 + getParentsNoLeaves(root.getLeft()) + getParentsNoLeaves(root.getRight());
    }
   }
  }
  return getParentsNoLeaves(root.getLeft()) + getParentsNoLeaves(root.getRight());
 }
}
public class P177Q18 {
    public static <T> boolean isInTree(BinNode<T> root, T val) {
     if (root == null) return false;
     if (root.getValue() == val) return true;
     return isInTree(root.getLeft(), val) || isInTree(root.getRight(), val);
    }
    public static boolean isIncluded(BinNode<Integer> t1, BinNode<Integer> t2) {
 if (t2 == null) return true;
 return isInTree(t1, t2.getValue()) && isIncluded(t1, t2.getLeft()) && isIncluded(t1, t2.getRight());
    }
}
public class P177Q19 {
 public static int getTreeSum(BinNode<Integer> root) {
  if(root == null) return 0;
  return root.getValue() + getTreeSum(root.getLeft()) + getTreeSum(root.getRight());
 }
}

public class P177Q20 {
 public static <T> int getValCount(BinNode<T> root, Integer n) {
  if (root == null) return 0;
  if (root.getValue() == n) return 1 + getValCount(root.getLeft(), n) + getValCount(root.getRight(), n);
  return getValCount(root.getLeft(), n) + getValCount(root.getRight(), n);
 }
 public static <T> boolean isConsNTree(BinNode<T> root, Integer n) {
  if (root == null || !(root.getValue() instanceof Integer)) return false;
  if (n == 0) return true;
  if (getValCount(root, n) != 1) return false;
  return isConsNTree(root, n-1);
 }
}
public class P177Q21 {
 public static <T> int getHeight(BinNode<T> root) {
         if (root == null) return 0;
         if (!root.hasLeft() && !root.hasRight()) return 0;
         return 1 + Math.max(getHeight(root.getLeft()), getHeight(root.getRight()));
        }
 public static boolean isSymmetrical(BinNode<Integer> root) {
  if (root == null) return true;
  int cl = root.hasLeft() ? 1 : 0,
   cr = root.hasRight() ? 1 : 0;
  if (cl+getHeight(root.getLeft()) - (cr+getHeight(root.getRight())) > 1) return false;
  //if (Math.abs(cl+getHeight(root.getLeft()) - (cr+getHeight(root.getRight())) > 1) return false;
  return isSymmetrical(root.getLeft()) && isSymmetrical(root.getRight());
 }
}
public class P177Q22 {
 public static <T> boolean isBalanced(BinNode<T> root) {
  if (root == null) return true;
  if (root.hasLeft() ^ root.hasRight()) return false;
  return isBalanced(root.getLeft()) && isBalanced(root.getRight());
 }
}

public class P177Q23 {
 // A
 public static Integer getMaxValue(BinNode<Integer> root) {
  if (root == null) return null;
  if (!root.hasLeft() && !root.hasRight()) return root.getValue();
  if (!root.hasRight()) return Math.max(root.getValue(), getMaxValue(root.getLeft()));
  if (!root.hasLeft()) return Math.max(root.getValue(), getMaxValue(root.getRight()));
  return Math.max(Math.max(root.getValue(), getMaxValue(root.getRight())), getMaxValue(root.getLeft()));
 }
 // B
 public static Integer getMinValue(BinNode<Integer> root) {
  if (root == null) return null;
  if (!root.hasLeft() && !root.hasRight()) return root.getValue();
  if (!root.hasRight()) return Math.min(root.getValue(), getMinValue(root.getLeft()));
  if (!root.hasLeft()) return Math.min(root.getValue(), getMinValue(root.getRight()));
  return Math.min(Math.min(root.getValue(), getMinValue(root.getRight())), getMinValue(root.getLeft()));
 }
 // C
 public static void main(String[] args) {
  BinNode<Integer> root = new BinNode<Integer>(1);
  BinNode<Integer> a = new BinNode<Integer>(2);
  BinNode<Integer> b = new BinNode<Integer>(3);
  BinNode<Integer> c = new BinNode<Integer>(4);
  root.setLeft(a);
  a.setRight(b);
  root.setRight(c);
  System.out.println(getMinValue(root)-getMaxValue(root)); // abs diff?
 }
}
public class P177Q24 {
 // A
 public static <T> boolean hasOneChild(BinNode<T> node) {
  if (node == null) return false;
  return node.hasLeft() ^ node.hasRight();
 }
 // B
 public static <T> int getOnlyChildCount(BinNode<T> root) {
```

```
  if (root == null) return 0;
  int s = hasOneChild(root) ? 1 : 0;
  return s + getOnlyChildCount(root.getLeft()) + getOnlyChildCount(root.getRight());
 }
 // C
 public static int getOnlyGrandsonCount(BinNode<Integer> root) {
  if (root == null) return 0;
  int c = hasOneChild(root) && (hasOneChild(root.getLeft()) || hasOneChild(root.getRight())) ? 1 : 0;
  return c + getOnlyGrandsonCount(root.getLeft()) + getOnlyGrandsonCount(root.getRight());
 }
}
public class P177Q25 {
 public static boolean isDescendantPriv(BinNode<Character> root, char c) {
  if (root == null) return false;
  if (root.getValue() == c) return true;
  return isDescendantPriv(root.getLeft(), c) || isDescendantPriv(root.getRight(), c);
 }
 public static boolean isDescendant(BinNode<Character> t, char c1, char c2) {
  if (t == null) return false;
  if (t.getValue() == c1) return isDescendantPriv(t, c2);
  if (t.getValue() == c2) return isDescendantPriv(t, c1);
  return isDescendant(t.getLeft(), c1, c2) || isDescendant(t.getRight(), c1, c2);
 }
}
public class P177Q26 {
 public static <T> int getHeight(BinNode<T> root) {
     if (root == null) return 0;
     if (!root.hasLeft() && !root.hasRight()) return 0;
     return 1 + Math.max(getHeight(root.getLeft()), getHeight(root.getRight()));
    }
    public static <T> int getLeafCount(BinNode<T> root){
     if (root == null) return 0;
     if(!root.hasLeft() && !root.hasRight()) return 1;
     return getLeafCount(root.getLeft()) + getLeafCount(root.getRight());
    }
 public static <T> boolean isFull(BinNode<T> root) {
  return Math.pow(2, getHeight(root)) == getLeafCount(root);
 }
}
public class P177Q27 {
 public static <T> int getHeight(BinNode<T> root) {
     if (root == null) return 0;
     if (!root.hasLeft() && !root.hasRight()) return 0;
     return 1 + Math.max(getHeight(root.getLeft()), getHeight(root.getRight()));
    }
}
```