```java
package model;

import tests.TreeNode;
import tests.SLLNode;

public class TreeUtilities {

    /*
     * Assumptions:
     *  + lowerRank >= 1
     *  + upperRank <= size of tree
     *  + root != null
     */
    public SLLNode<Integer> getElementsOfRanks(TreeNode<Integer> root, int lowerRank, int upperRank) {
        SLLNode<Integer> sortedList = getSortedListFrom(root);
        SLLNode<Integer> result = null;
        SLLNode<Integer> currentResult = null;

        int i = 1;
        SLLNode<Integer> current = sortedList;
        while(current != null) {

            if(lowerRank <= i && i <= upperRank) {
                SLLNode<Integer> n = new SLLNode<>(current.getElement(), null);
                if(result == null) {
                    result = n;
                    currentResult = result;
                }
                else {
                    currentResult.setNext(n);
                    currentResult = currentResult.getNext();
                }
            }
            current = current.getNext();
            i ++;
        }

        return result;
    }
```

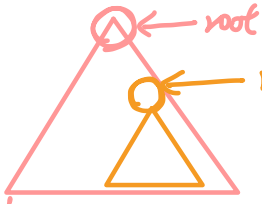
phase 1 ↓ traverse & sort

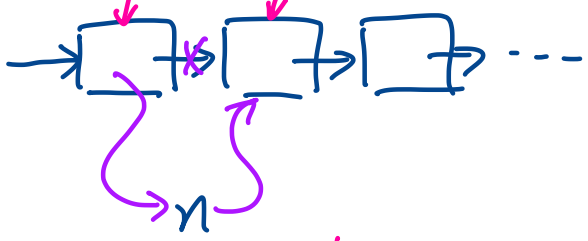
phase 2 ↓ extract nodes from indices i to j from sorted chain

when loop counter i is in range, extract the nodes.

```java
private SLLNode<Integer> getSortedListFrom(TreeNode<Integer> root) {
    SLLNode<Integer> start = new SLLNode<>(null, null);
    SLLNode<Integer> sorrtedList = getSortedListFromHelper(root, start);
    return sorrtedList;
}

private SLLNode<Integer> getSortedListFromHelper(TreeNode<Integer> root, SLLNode<Integer> current) {
    SLLNode<Integer> result = null;
    if(current.getElement() == null) {
        current.setElement(root.getElement());
        result = current;
    }
    else {
        result = current;
        boolean inserted = false;
        Integer rootElement = root.getElement();
        SLLNode<Integer> n = new SLLNode<>(rootElement, null);
        SLLNode<Integer> prev = null;
        while(current != null && !inserted) {
            Integer currentElement = current.getElement();
            if(rootElement > currentElement) {
                prev = current;
                current = current.getNext();
            }
            else {
                if(prev == null) { /* root element becomes the min in the list */
                    n.setNext(current);
                    result = n;
                }
                else {
                    n.setNext(prev.getNext());
                    prev.setNext(n);
                }
                inserted = true;
            }
        }
        if(!inserted) { /* root element becomes the max in the list */
            prev.setNext(n);
        }
    }
    SLLNode<TreeNode<Integer>> children = root.getChildren();
    SLLNode<TreeNode<Integer>> currentChild = children;
    while(currentChild != null) {
        result = getSortedListFromHelper(currentChild.getElement(), result);
        currentChild = currentChild.getNext();
    }
    return result;
}
```




n.getElement > prev.e
n.getElement ≤ current.e

```java
    public TreeNode<String> getStats(TreeNode<Integer> n) {
        TreeNode<Integer> rootOfSizes = getSizes(n);
        TreeNode<Integer> rootOfSums = getSums(n);
        return getStatsHelper(rootOfSizes, rootOfSums);
    }

    private TreeNode<String> getStatsHelper(TreeNode<Integer> currentSize, TreeNode<Integer> currentSum) {
        String stat = String.format("Number of descendants: %d; Sum of descendants: %d", currentSize.getElement(),
currentSum.getElement());
        TreeNode<String> result = new TreeNode<>(stat);

        if(currentSize.getChildren() != null) {
            SLLNode<TreeNode<Integer>> currentSizeChild = currentSize.getChildren();
            SLLNode<TreeNode<Integer>> currentSumChild = currentSum.getChildren();

            while(currentSizeChild != null) {
                TreeNode<String> statOfChild = getStatsHelper(currentSizeChild.getElement(),
currentSumChild.getElement());

                result.addChild(statOfChild);
                statOfChild.setParent(result);

                currentSizeChild = currentSizeChild.getNext();
                currentSumChild = currentSumChild.getNext();
            }
        }

        return result;
    }

    private TreeNode<Integer> getSizes(TreeNode<Integer> n) {
        TreeNode<Integer> result = new TreeNode<>(1);
        if(n.getChildren() != null) {
            SLLNode<TreeNode<Integer>> currentChild = n.getChildren();
            while(currentChild != null) {
                TreeNode<Integer> sizeOfChild = getSizes(currentChild.getElement());
                result.setElement(result.getElement() + sizeOfChild.getElement());
                result.addChild(sizeOfChild);
                sizeOfChild.setParent(result);
                currentChild = currentChild.getNext();
            }
        }
        return result;
    }

    private TreeNode<Integer> getSums(TreeNode<Integer> n) {
        TreeNode<Integer> result = new TreeNode<>(n.getElement());
        if(n.getChildren() != null) {
            SLLNode<TreeNode<Integer>> currentChild = n.getChildren();
            while(currentChild != null) {
                TreeNode<Integer> sumOfChild = getSums(currentChild.getElement());
                result.setElement(result.getElement() + sumOfChild.getElement());
                result.addChild(sumOfChild);
                sumOfChild.setParent(result);
                currentChild = currentChild.getNext();
            }
        }
        return result;
    }
}
```

pay attention to how these two references are used to traverse the two structurally identical trees at the same time.