Problem Set 6

Binary Search Tree (BST)

---

1. Given the following sequence of integers:

    ```
    10, 17, 3, 90, 22, 7, 40, 15
    ```

    1. Starting with an empty binary search tree, insert this sequence of integers one at a time into this tree. Show the final tree. Assume that the tree will not keep any duplicates. This means when a new item is attempted to be inserted, it will not be inserted if it already exists in the tree.
    2. How many item-to-item comparisons in all did it take to build this tree? (Assume one comparison for equality check, and another to branch left or right.)

---

2. For the tree built in the above problem:
    1. What is the worst case number of comparisons for a successful search in this tree? For an unsuccessful (failed) search? (Assume one comparison for equality check, and another to branch left or right.)
    2. What is the average case number of comparisons for a successful search in this tree?
    3. From this tree, delete 17: find the node (y) that has the smallest value in its right subtree, write y's value over 17, and delete y. How much work in all (locating 17, then locating y , then deleting y) did it take to complete the deletion? Assume the following (a) you are using two pointers to navigate down the tree, a tracking pointer, and a lagging pointer, (b) 1 unit of work for an equality comparison between target and tree item, one for an inequality check to branch left or right, and 1 unit of work for a pointer assignment.

---

3. Given the following BST node class:

    ```
    public class BSTNode<T extends Comparable<T>> {
        T data;
        BSTNode<T> left, right;
        public BSTNode(T data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
         public String toString() {
            return data.toString();
        }
    }
    ```

    Consider the following method to insert an item into a BST that does not allow duplicate keys:

    ```
    public class BST<T extends Comparable<T>> {
         BSTNode<T> root;
         int size;
         ...
         public void insert(T target)
         throws IllegalArgumentException {

                 BSTNode ptr=root, prev=null;
                 int c=0;
                 while (ptr != null) {
                         c = target.compareTo(ptr.data);
                         if (c == 0) {
                                 throw new IllegalArgumentException("Duplicate key");
                         }
                         prev = ptr;
                         ptr = c < 0 ? ptr.left : ptr.right;
                 }
                 BSTNode tmp = new BSTNode(target);
                 size++;
                 if (root == null) {
                         root = tmp;
                         return;
                 }
    ```

```
            if (c < 0) {
                    prev.left = tmp;
            } else {
                    prev.right = tmp;
            }
    }
```

Write a recursive version of this method, using a helper method if necessary.

---

4. * With the same **BSTNode** class as in the previous problem, write a method to count all entries in the tree whose keys are in a given range of values. Your implementation should make as few data comparisons as possible.

```
// Accumulates, in a given array list, all entries in a BST whose keys are in a given range,
// including both ends of the range - i.e. all entries x such that min <= x <= max.
// The accumulation array list, result, will be filled with node data entries that make the cut.
// The array list is already created (initially empty) when this method is first called.
public static <T extends Comparable<T>>
void keysInRange(BSTNode<T> root, T min, T max, ArrayList<T> result) {
    /* COMPLETE THIS METHOD */

 }
```

---

5. With the same **BSTNode** class as in the previous problem, write a method that would take a BST with keys arranged in ascending order, and "reverse" it so all the keys are in descending order. For example:

```
          25                          25
        /    \                      /    \
      10      40      --->        40      10
     /  \    /  \                /  \    /  \
    2   20 30    45            45   30  20   2
       /    \                      /      \
      15     35                   35       15
```

The modification is done in the input tree itself, NO new tree is created.

```
public static <T extends Comparable<T>>
void reverseKeys(BSTNode<T> root) {
    /* COMPLETE THIS METHOD */
```

---

6. * A binary search tree may be modified as follows: in every node, store the number of nodes in its *right subtree*. This modification is useful to answer the question: what is the **k-th largest element** in the binary search tree? (k=1 refers to the largest element, k=2 refers to the second largest element, etc.)

You are given the following enhanced binary search tree node implementation:

```
public class BSTNode<T extends Comparable<T>> {
   T data;
   BSTNode<T> left, right;
   int rightSize;  // number of entries in right subtree
   ...
}
```

Implement the following *recursive* method to find the **k-th largest** entry in a BST:

```
public static <T extends Comparable<T>> T kthLargest(BSTNode<T> root, int k) {
   /* COMPLETE THIS METHOD */
}
```