CMSC132 Fall 2018 Binary Search Trees Worksheet

The following Java class definition for a binary search tree will be used to answer the questions that follow. We use null to represent an empty tree. For example, an empty BinarySearchTree has a null root, and a leaf node has null left and right fields. Feel free to add any constructors or auxiliary methods you considered necessary.

```
public class BinarySearchTree <K extends Comparable<K>, V> {
       private class Node {
              private K key;
              private V data;
              private Node left, right;
              public Node(K key, V data) {
                 this.key = key;
                 this.data = data;
              }
       }

       private Node root;
}
```

1. Implement a **recursive** method called **getKeysSorted** that will return an ArrayList with the keys of the tree in sorted order.

```
public ArrayList<K> getKeysSorted()
```

2. Implement a **recursive** method called **getLevel** that will return an integer representing the level where the node is found in the tree. For this problem you can assume the root is at level 1. The method will return -1 if the node is not in the tree.

```
public int getLevel(K target)
```

3. Implement a **recursive** method called **getPath** that returns an ArrayList with the keys of nodes visited while searching for the node in the tree with a key that corresponds to **target**.

```
public ArrayList<K> getPath(K target)
```

4. Implement a method called **getTreeInRange** that returns a binary search tree with the keys and values present in the range defined by the **start** and **end** parameters. Your implementation should not traverse the whole tree if it can be avoided. For example, if we have a tree with a root value of (100, "house"), finding a tree in the range 10, 90 should not traverse the right subtree. The new tree can have any shape.

```
public BinarySearchTree<K, V> getTreeInRange(K start, K end)
```