Homework 3

Points 100

1. **Write functions to perform following tasks**
    a. **to count the number of nodes in a binary tree [5 points]**

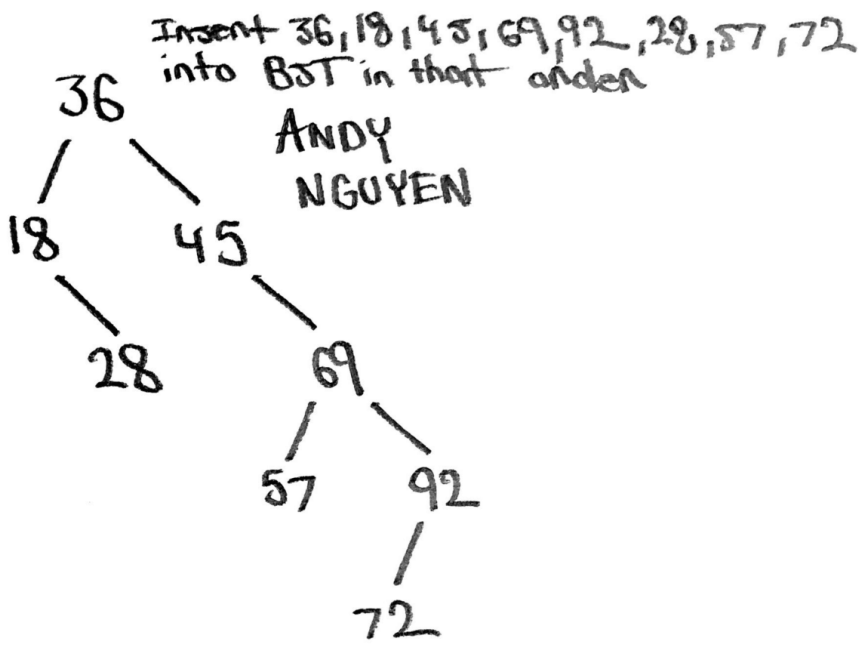
```
// created a function which will count number of nodes in binary tree recursively
// the function takes in the root node as a parameter
public int countOfNodes (Node rootNode) {

   if (rootNode == null) {

      return 0;

   }
   // continue recursively counting the children of the node
   // getLeftNode() and getRightNode() are functions which get the children of parent node
   return 1 + countOfNodes(rootNode.getLeftNode()) + countOfNodes(rootNode.getRightNode());
}
```

    b. **to count the number of leaves [5 points]**

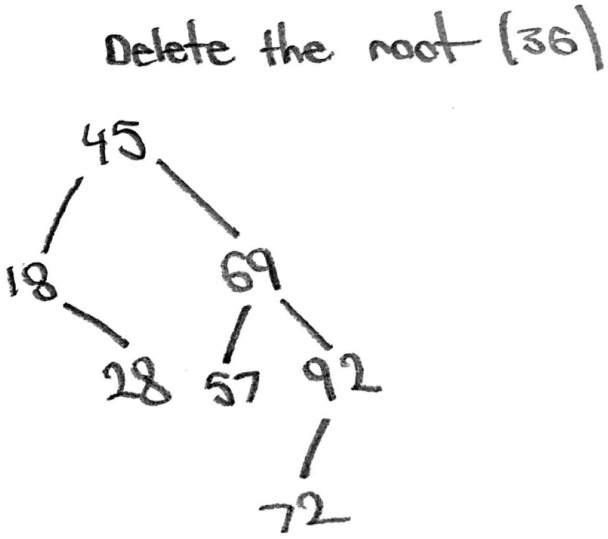
```
// created a function which will count number of leaf nodes in binary tree recursively
// the function takes in the root node as a parameter
public int countOfLeafNodes (Node rootNode) {
        if (rootNode == null) {
                return 0; // if the current node is null, end recursion.
        }
        if (rootNode.getLeftNode() == null && rootNode.getRightNode() == null) {
                return 1; // if both children are null, return 1 (adds to count of leaf nodes)
        }
        // continues recursively traversing to children nodes of the binary tree to count leaf nodes
        return countLeafNodes(rootNode.getLeftNode()) + countLeafNodes(rootNode.getRightNode());
}
```

2. BST

a. Show the result of inserting 36, 18, 45, 69, 92, 28, 57, 72 into an initially empty binary search tree. [10 Points]

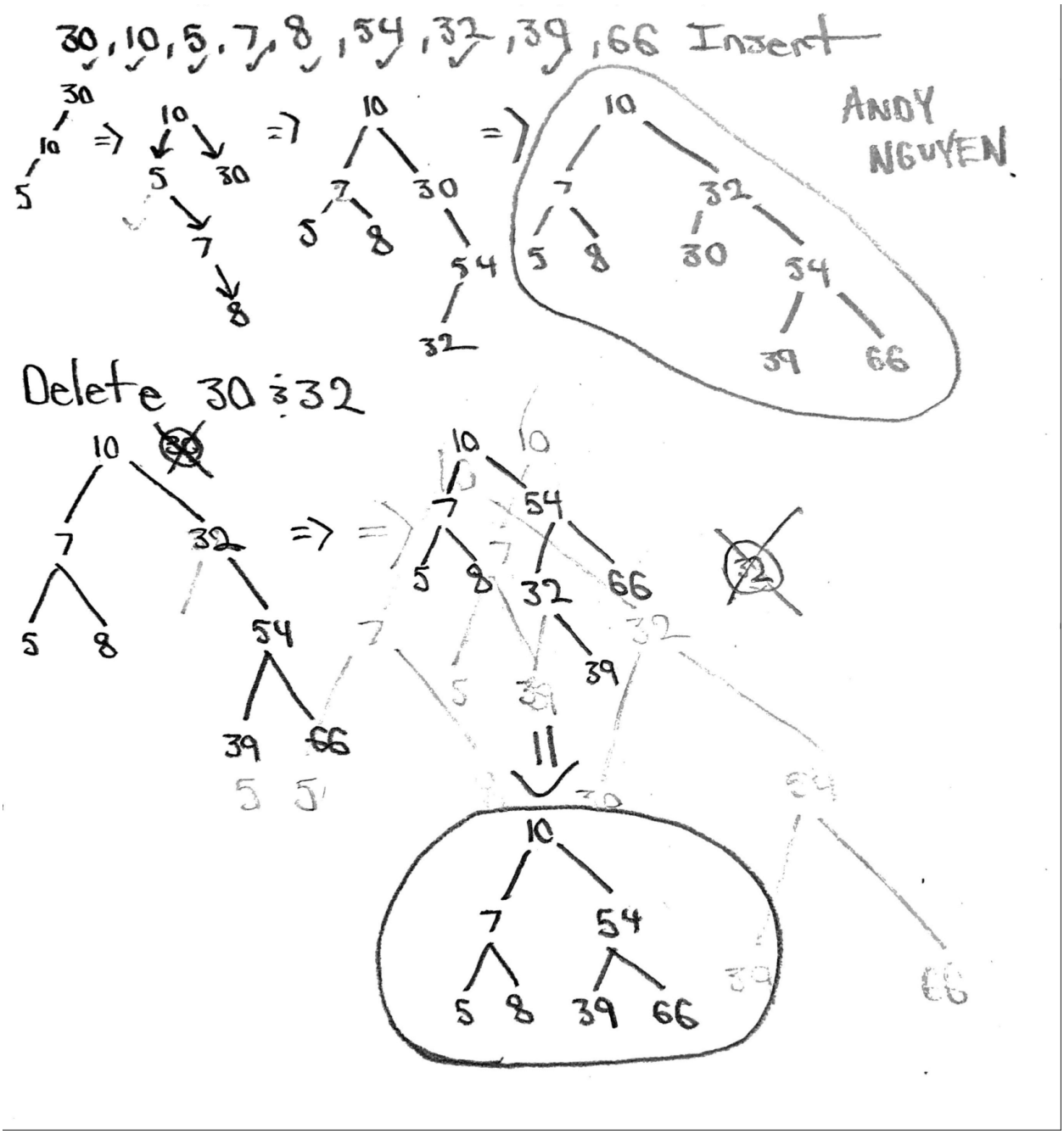


b. Show the result of deleting the root. [5 Points]

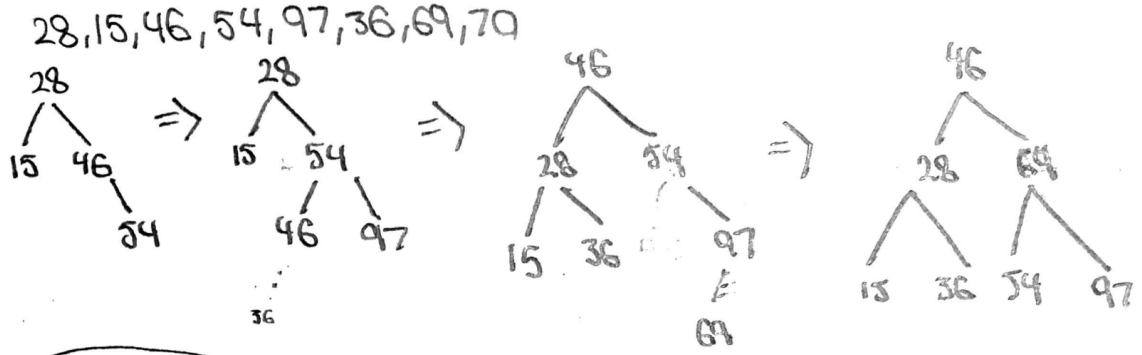


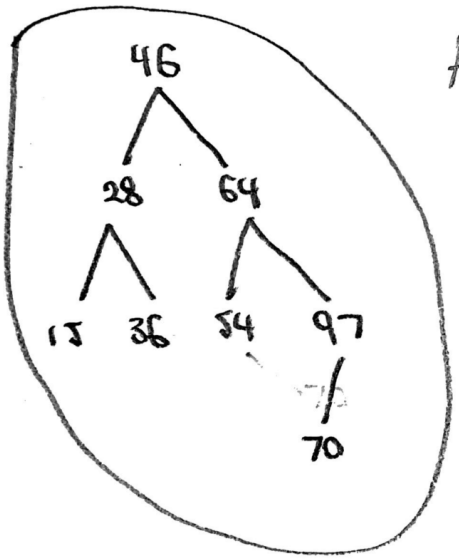
3. **AVL Tree**

   Show the result of inserting 30, 10, 5, 7, 8, 54, 32, 39, 66 and then deleting 30 and 32 from the AVL tree **[20 points]**



Show the result of inserting 28, 15, 46, 54, 97, 36, 69, 70 into an AVL Tree [10 Points]





ANDY NGUYEN

4. Splay Tree

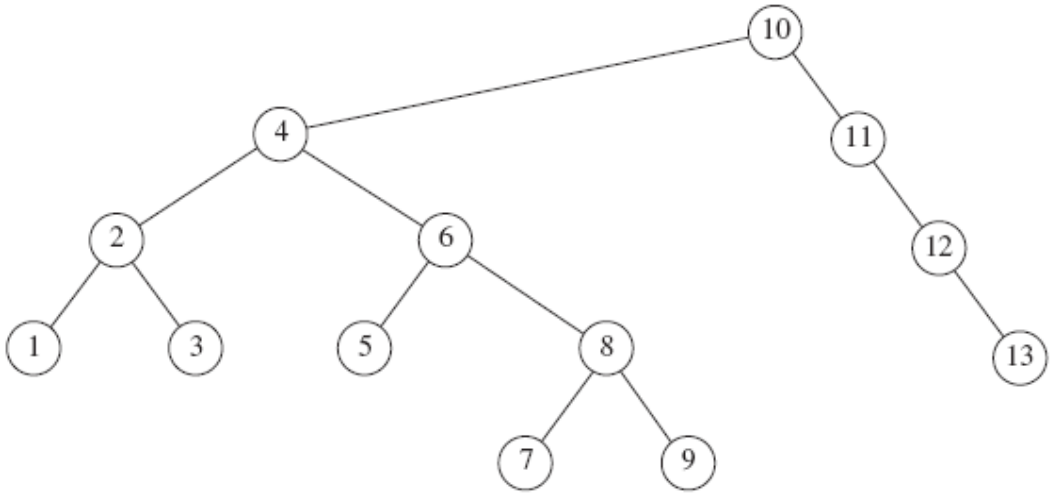
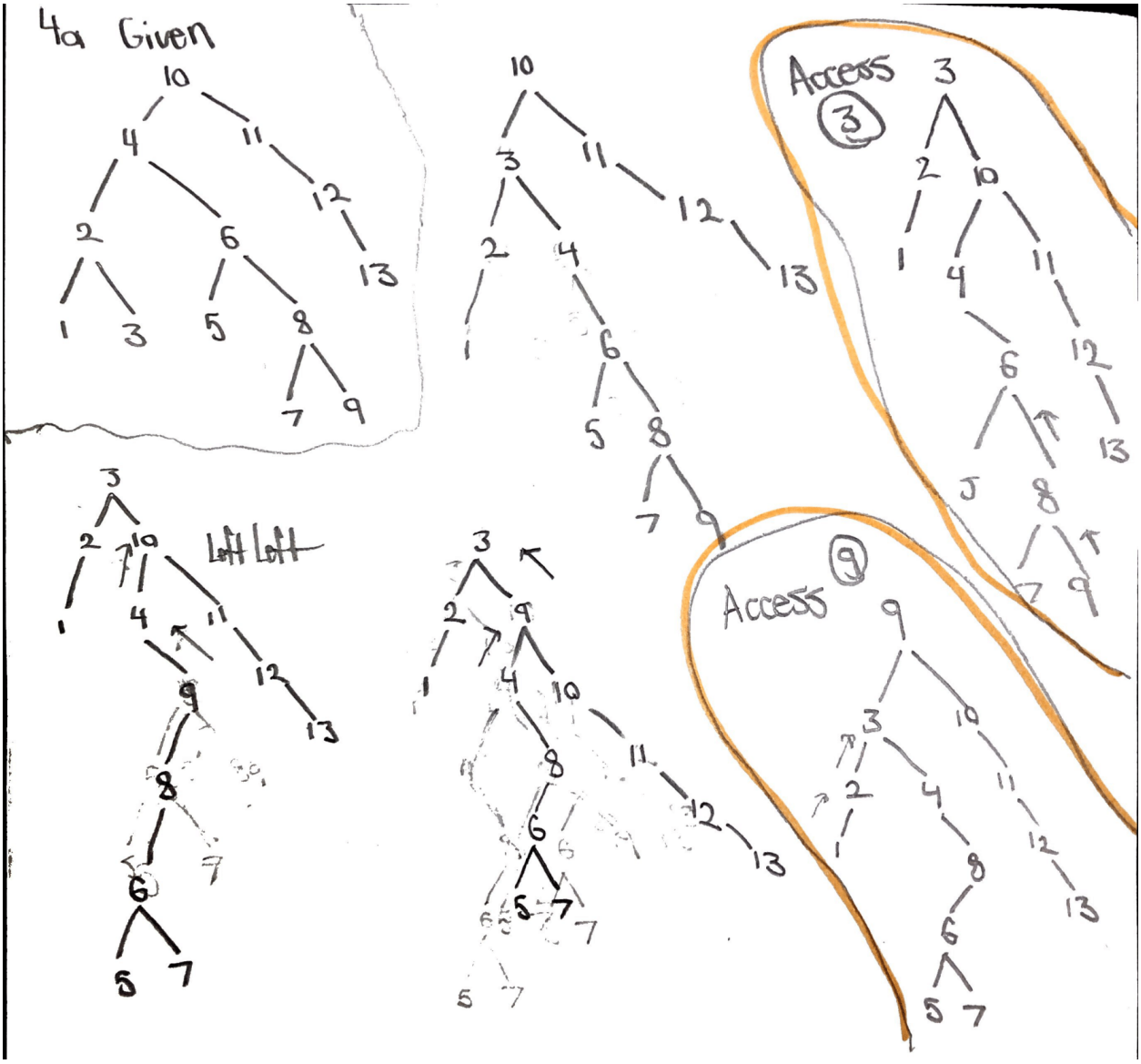
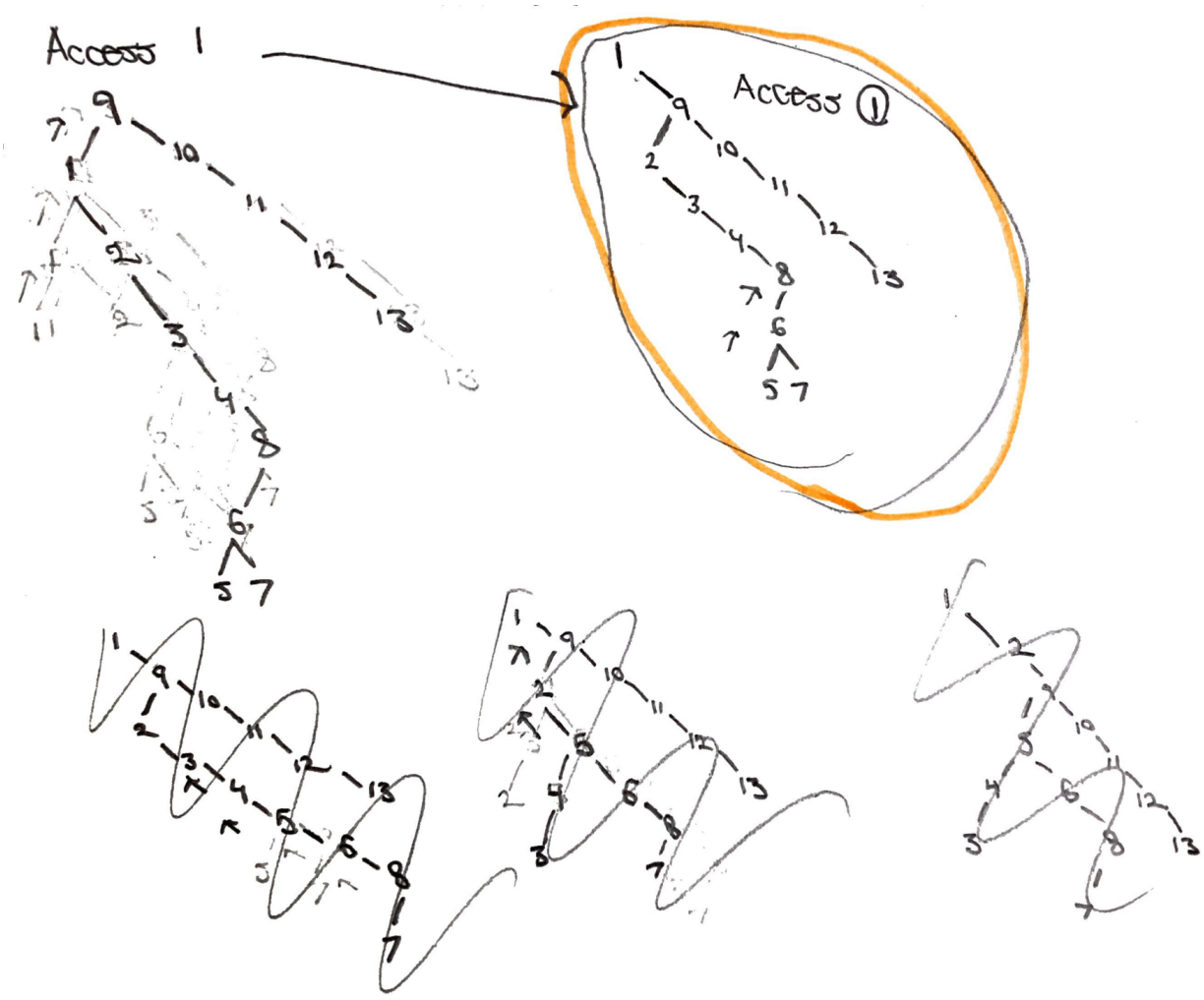
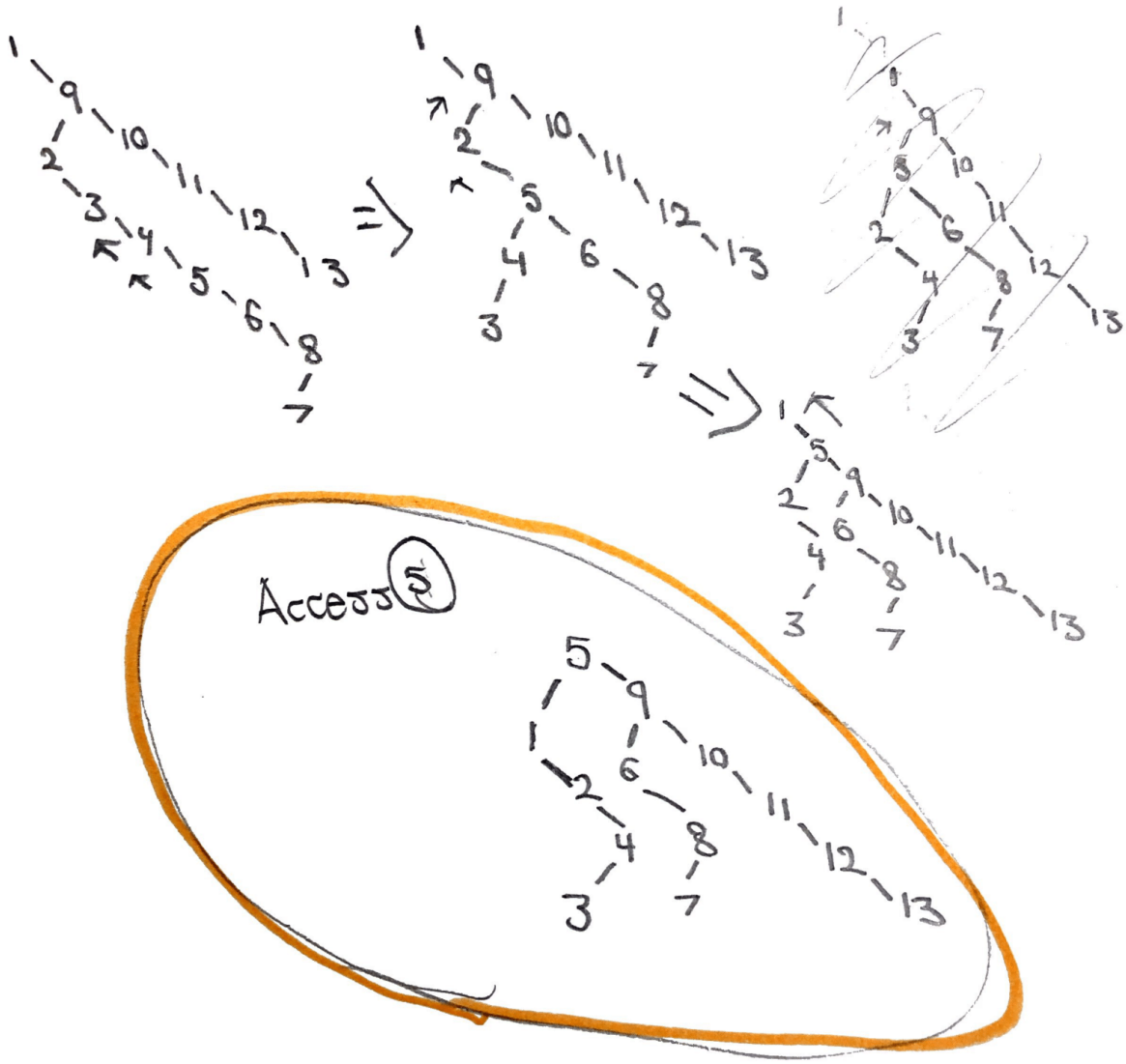


Figure 4.72 Tree for Exercise 4.27

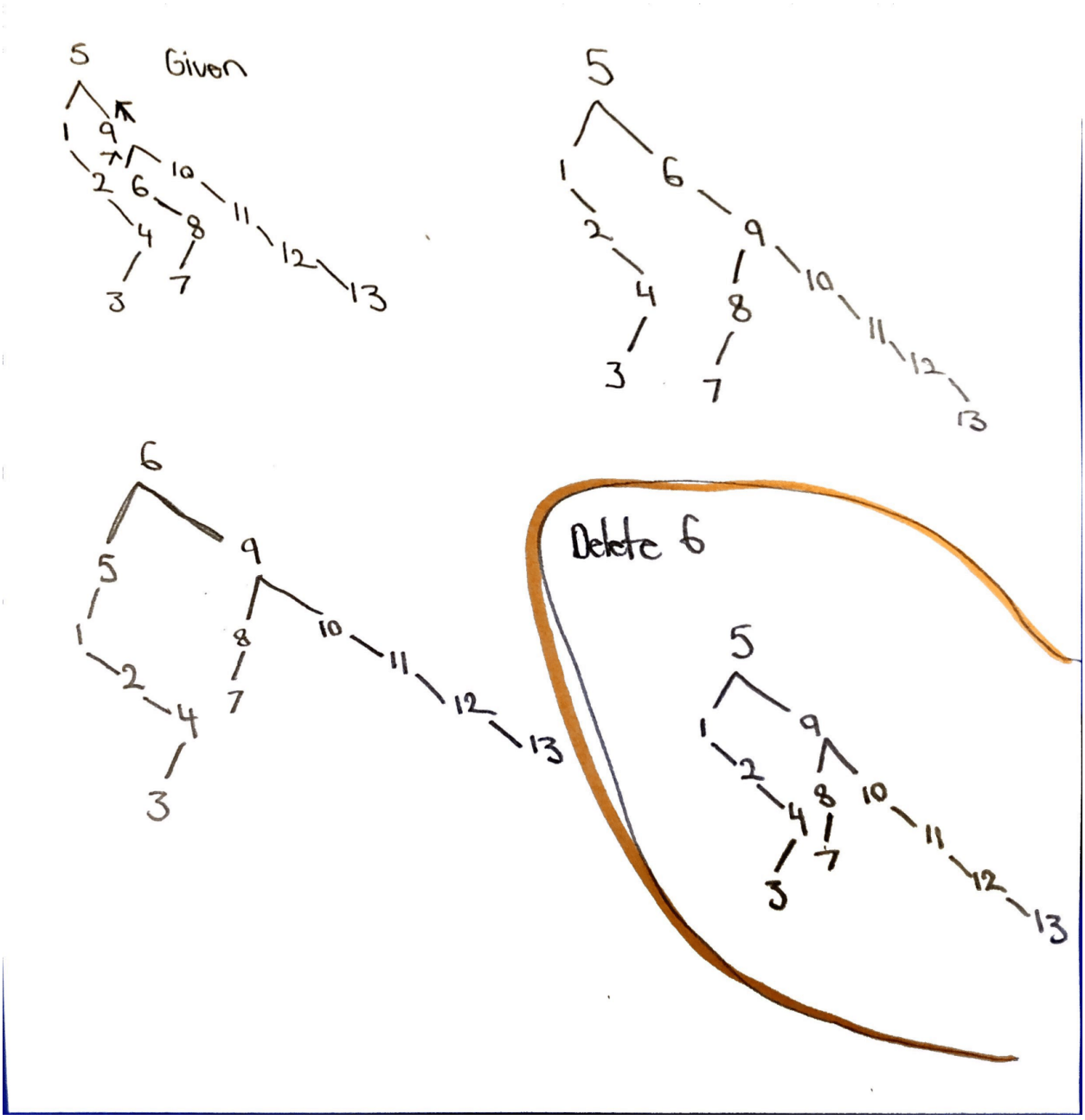
a. Show the result of accessing the keys 3, 9, 1, 5 in order in the splay tree [10 Points]


4a Given
Access 3
Access 9
Left Left


Access 1
Access ①


Access 5

b. Show the result of deleting key 6 after doing (a). **[5 Points]**



5. **Two binary trees are similar if they are both empty or both nonempty and have similar left and right subtree. Write a method to decide whether two binary tree are similar. [10 points]**

```
public boolean areTreesSimilar(Node rootNodeA, Node rootNodeB) {
    if (rootNodeA == null && rootNodeB == null) {
        return true; // if both trees are empty, trees are similar
    }
    if (rootNodeA != null && rootNodeB != null) { // non-empty so check children nodes and compare data
return (rootNodeA.getData() == rootNodeB.getData()) && areTreesSimilar(rootNodeA.getLeft(),
rootNodeB.getLeft()) && areTreesSimilar(rootNodeA.getRight(), rootNodeB.getNodeB());
    }
    return 0; // trees are not similar.
}
```

What is the running time of your method? [5 Points]

The running time of the method will be based upon the binary tree with less number of nodes. Assuming two binary trees have 'x' and 'y' number of nodes respectively and 'x' < 'y'. The running time complexity of the method is O(x).

6. **Compare and contrast AVL tree and splay tree** [5 Points]

Both AVL trees and Splay trees are similar in that they are self-balancing data structures, both are binary trees in which a node can have up to two children nodes, and each node has a smaller value in the left child node while the right child node has a greater value. Both trees use the general idea of rotations to maintain BST order.

AVL trees differ from splay tree in that the heights of two subtrees of any node only differ at most by 1. With the following balance condition, it is guaranteed that the worst-case time for searching/insertion/deletion is O(log n). It can also be said that the rebalancing of the tree can slow down insertion and deletion operations in order to maintain BST ordering. Worst case performance for AVL trees is more predictable.

Splay trees differ from AVL trees in that the most recently accessed element is moved to the root of the tree. This parameter results in more frequently accessed nodes being closer to the top of the tree resulting in improved search times. Splay trees zig-zag operations in addition to move nodes around in the tree to maintain BST ordering.

7. **Given a binary search tree and a value *k*, write a pseudo code to find a node in the binary search tree whose value is closest to *k*. [10 Points]**

```
public Node findClosestValue(Node rootNode, int k) {

    If the rootNode is equal to null, return null meaning the tree is empty

    Create a Node variable to store the node which contains the closest value to k // closest node
    Create a Node variable used to traverse the BST and set it equal to the root node // traversal node

    // traverse the BST
    While the traversal node is not equal to null
        If the traversal node's value is equal to k
            Return the traversal node
        If the absolute value of (traversal node's value – k) is less than the absolute value of the (closest value - k)
            Set the closest node variable equal to the traversal node
        If k is less than the traversal node's value
            Set the traversal node equal to the traversal node's left child
        Else
            Set the traversal node equal to the traversal node's right child
    Return the closest node

}
```