(ii) preorder traversal f c b a e d h g i

(iii) postorder traversal a b d e c g i h f

Do you notice an interesting property of the in-order traversal? What is it?

(b) Let a binary search tree be defined by the following class:

(Next Page !)

```
public class IntTree {
    private IntTreeNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntTreeNode {
        public int data;
        public IntTreeNode left;
        public IntTreeNode right;

        // constructors omitted for clarity
    }
}
```

In class, we saw how to search for an element in a binary search tree. This question will demonstrate that binary search trees are more powerful. Describe an algorithm that calculates the k'th smallest element in the tree. (k, the input, is a number in $\{1, 2, \cdots, n\}$, where n is the number of nodes in the tree). You may find it helpful to modify the definition of IntTreeNode in order to accomplish this. Your algorithm should run in $O(h)$ time, where h is the height of the tree. An $O(n)$ solution will be given partial credit.

4. AVL Tree Implementation

(Next page !)

Write pseudocode for the AVL tree methods Balance, RotateLeft, and RotateRight. Assume that the rest of the data structure is implemented as in the Java code here `https://courses.cs.washington.edu/courses/cse373/18su/files/homework/AVLTree.java`.

You may use the skeleton of Balance from that code as a guide. Note you are not required to write your solution in Java, pseudocode is sufficient.

5. Hashing

Let the capacity of the hash table be 10 and the hash function be $h(x) = x$. Insert elements

$$42, 102, 12, 33, 25, 14, 62$$

(Next page !)

to a hash table

(a) that uses linear probing

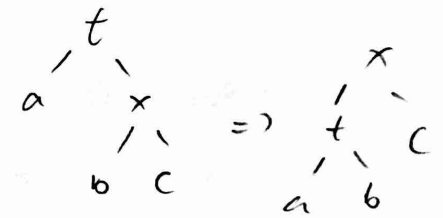
(b) that uses quadratic probing

Write down the total number of collisions and the hash table after all insertions in both cases. Why is the secondary clustering in quadratic probing less problematic than the primary clustering in linear probing (i.e. why are there fewer collisions)?

4 ✓ @param t: the root of the subtree to be balanced and we should return the "t". we set a private method, balance to check if the nodes are balanced.

```
private Node balanced (Node t) {
    if the "t" is equal to null (i.e it's empty).
    then return it (t).
  { if (the height of the t's left side is larger than
      the height of the t's right side over 1) {
  // we consider the double rotate in this case.
    if ( height of t's left left side smaller than
        height of t's left right side )
    rotateLeft (t.left) ; } // Rotate Left Right case.
    rotateRight (t); // Rotate for left left case
  } else if ( height of t's right - height of t's left > 1 ) {
    if ( height of t's right right side smaller than
        height of t's right left side ) {
      rotateRight (t.right); } // Right Right case
      rotateLeft (t) ;          // Right Left case.
  } return t ;
```

then we implement rotateLeft method:

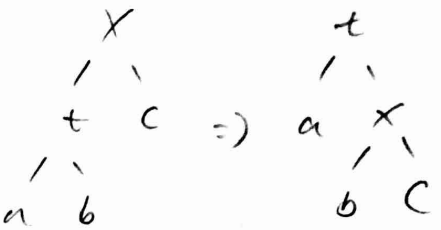



```
private Node rotateLeft (Node t) {
    Node x = t.right;
    Node b = x.left;
    t.right = b;   // Then adjust the height after rotated
    t.height = 1 + Math.max (height (t.left), height (t.right));
    x.height = 1 + Math.max (height (x.left), height (x.right));

    return x;

}
```

then we implement rotateRight(Node x) {




```
    Node t = x.left;
    Node b = t.right;
    x.left = b; // adjust the height after rotated
     x.height = if Math.max(height(x.left), height(x.right));
     t.height = if Math.max(height(t.left), height(t.right));
      return t;
}
```