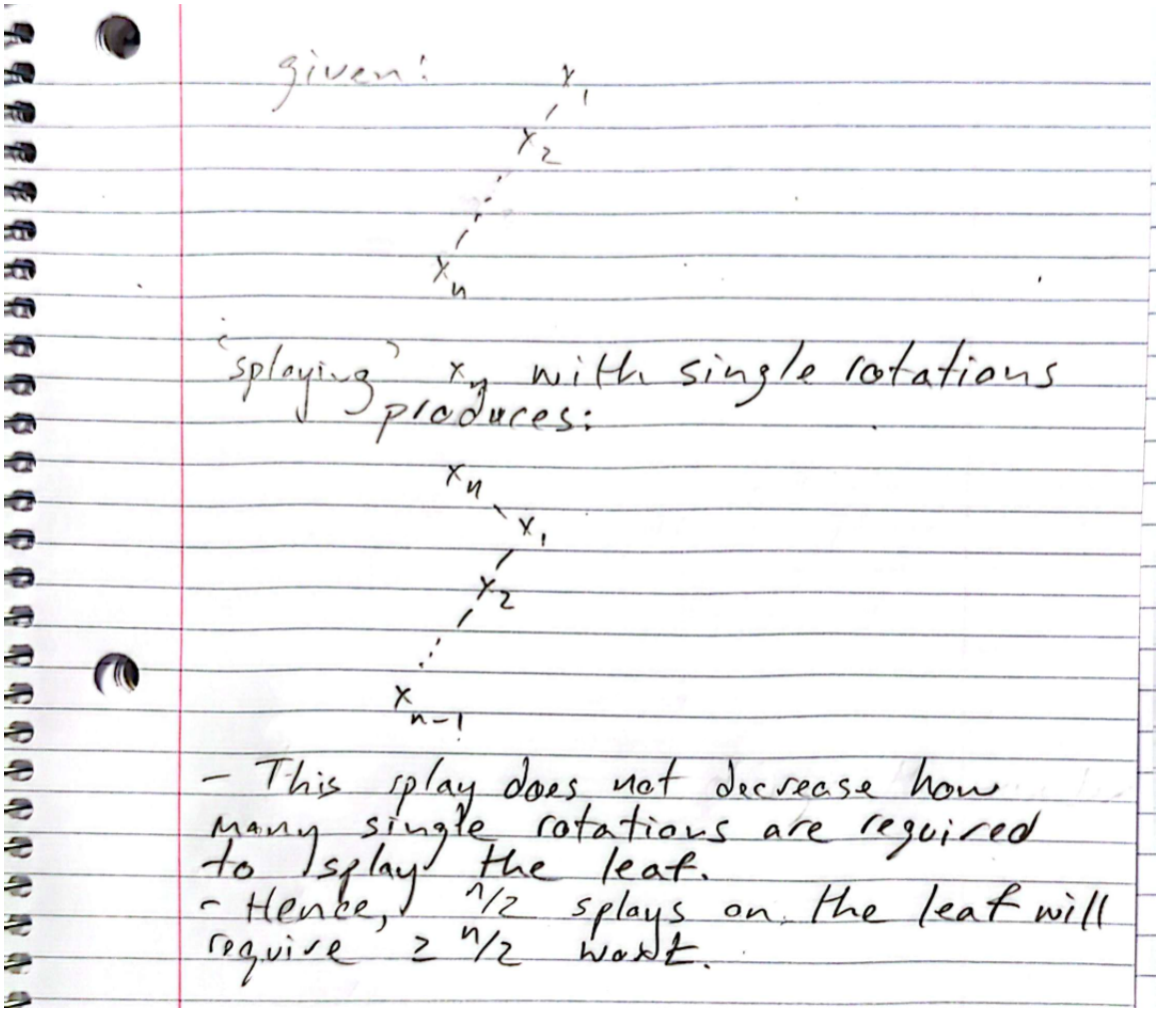
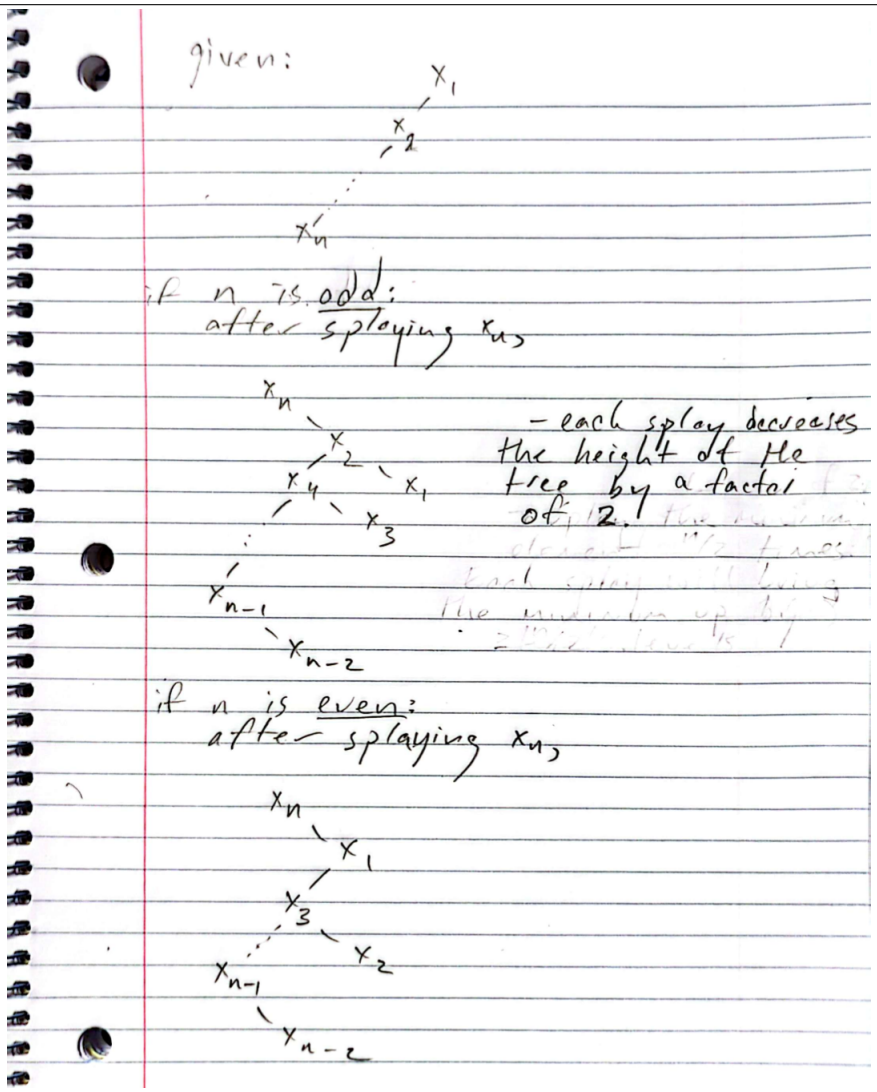
Contact: David Zhang – dzhang [at] cs [dot] toronto [dot] edu
All questions are taken from Karger's 6.854 Advanced Algorithms Homework (link):

1. Some Splaying Counterexamples

   (a) In class, we stated that single rotations "don't work" for splay trees. To demonstrate this, consider a degenerate n-node "linked list shaped" binary tree where each node's right child is empty. Suppose the (only) leaf is splayed to the root by single rotations. Show the structure of the tree after this splay. To generalize this example, argue that there is a sequence of $n/2$ splays that each take at least $n/2$ work.



(b) Now, for the same starting tree, show the structure after splaying the leaf with (zig-zig) double rotations. Explain how this splay has made much more progress than single rotations in "improving" the tree.



It is more accurate to say that the specific splay decreases the height of the tree rather than 'each' splay, although a subsequent splay will be cheaper than if we had used single rotations.

(c) To quantify this, compute the splay tree potential change for each single and double rotation of the linked list described above.

Let $s(x)$ denote the size of the subtree rooted at x. Let $r'(x)$ denote the rank of node x following a **single rotation** and let $r(x)$ denote its original rank. Suppose that x_i is the parent of x_n. Then, the difference in potential is

$$\begin{aligned} r'(x_n) - r(x_n) + r'(x_i) - r(x_i) &= \log(s(x_n)+1) - \log s(x_n) + \log(s(x_i)-1) - \log s(x_i) \\ &= \log(s(x_n)+1) - \log s(x_n) + \log s(x_n) - \log(s(x_n)+1) \\ &= 0 \end{aligned}$$

Let $r''(x)$ denote the rank of node x following a **double rotation** and suppose that x_{i-1} is its grandparent. Note that only the ranks of x_n and x_{i-1} change. Thus, the difference in potential in this case is

$$\begin{aligned} r''(x_n) - r(x_n) + r''(x_{i-1}) - r(x_{i-1}) &= \log(s(x_n)+2) - \log s(x_n) + 0 - (\log s(x_n) + 2) \\ &= -\log s(x_n) \end{aligned}$$

Hence, double rotations will simplify the data structure and single rotations will not.

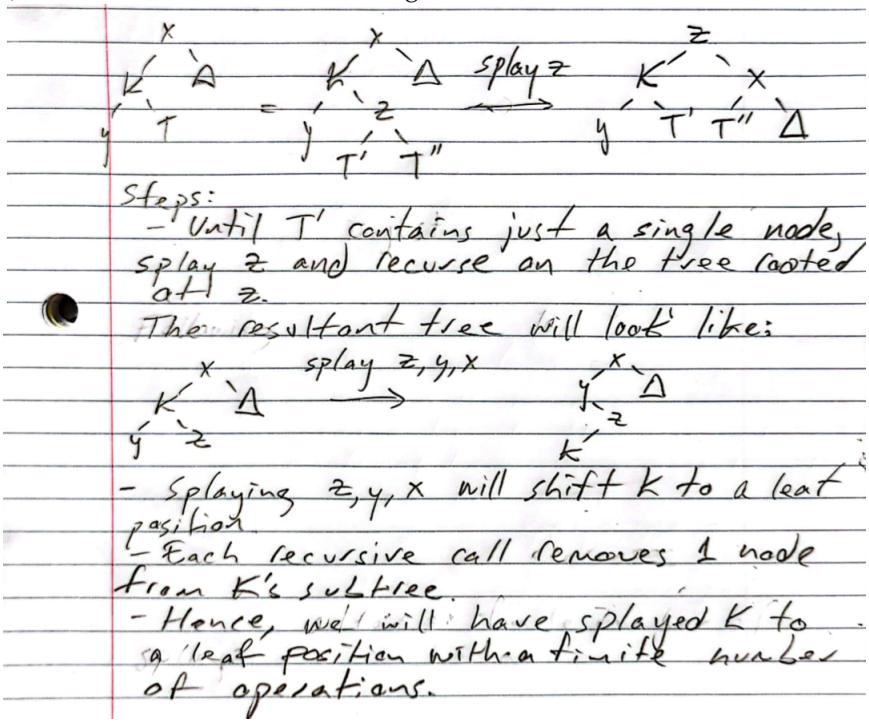
(d) Given the theorem about access times in splay trees, it is tempting to conjecture that splaying does not create trees in which it would take a long time to find an item. Show that this conjecture is false by showing that for large enough n, it is possible to restructure any binary tree on n nodes into any other binary tree on n nodes by a sequence of splay operations. Conclude that it is possible to make a sequence of requests that cause the splay tree to achieve any desired shape.

Hint: Start by showing how you can use splay operations to make a specified node into a leaf; then recurse.

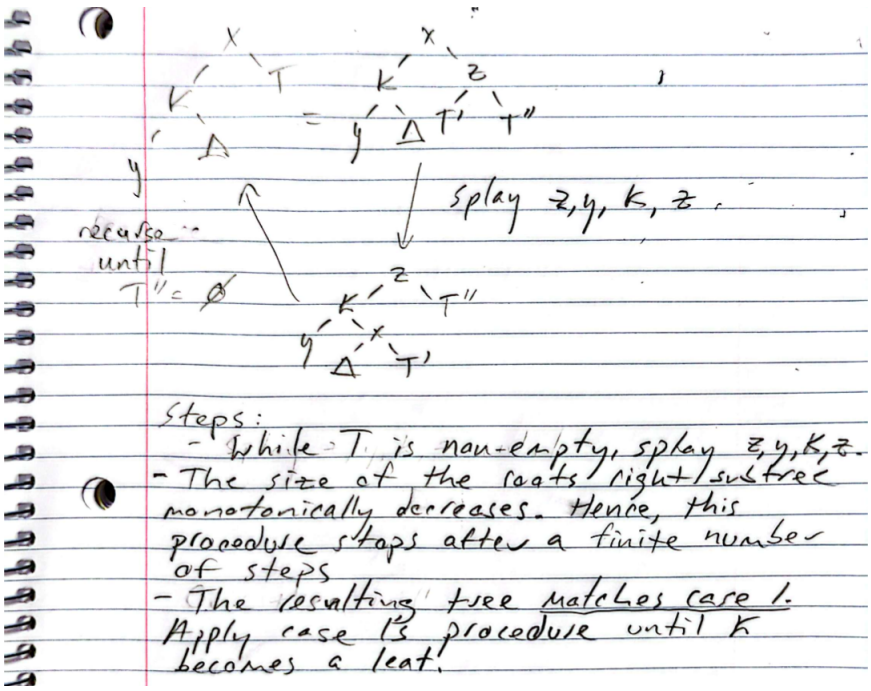
The first n for which the claim is true is $n = 4$. One can verify that given any binary search tree on 4 nodes, some sequence of spays will transform it into one of the other 13 possibilities.

Let the claim hold for $n \geq 4$. Suppose that we have an arbitrary binary search tree on $n+1$ nodes, and let our desired tree (also on $n+1$ nodes) be arbitrary. Pick a leaf K in our desired binary search tree. In the non-trivial case, K will not be a leaf in our given binary search tree – we will demonstrate a procedure to force K to a leaf position. The special cases of the procedure are as follows:

- If K is the minimum or maximum element in the tree, then splaying its successor (or predecessor) will push it to a leaf position.
- If K is the second smallest element in the tree, then its left node must exist, and it must be the minimum. In particular, splaying the minimum element or any of its ancestors will induce a series of zig-zig right rotations and maybe a final single right rotation. First, splay K's parent, x. Every right rotation (single or double) leaves K as the left child of its parent. Let Δ denote unimportant but present subtrees (including empty subtrees). After splaying x to the root, we will be in one of 2 cases (or their flipped variant): Either K has an element in its right subtree,

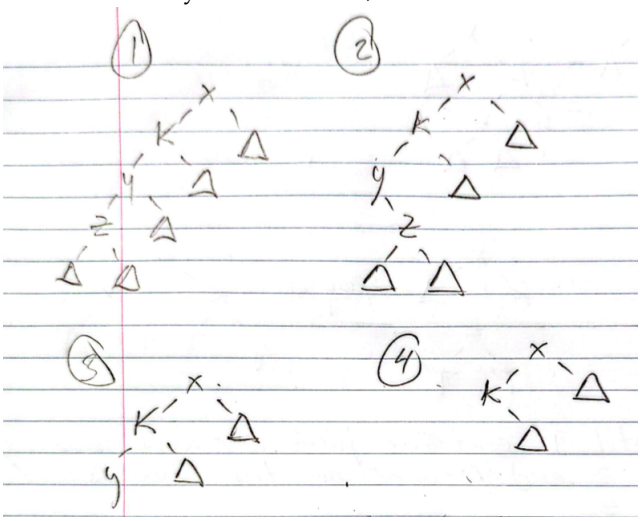


or x has an element in its right subtree. At least one of these cases must be true since $n + 1 \geq 5$.



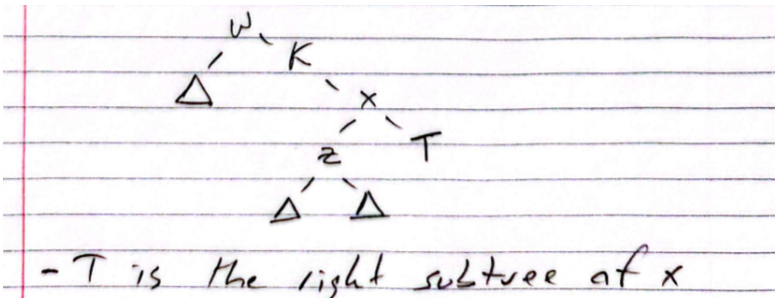
Both cases splay K to a leaf position.

If K is any other element, consider the case where it is the left child of its parent.



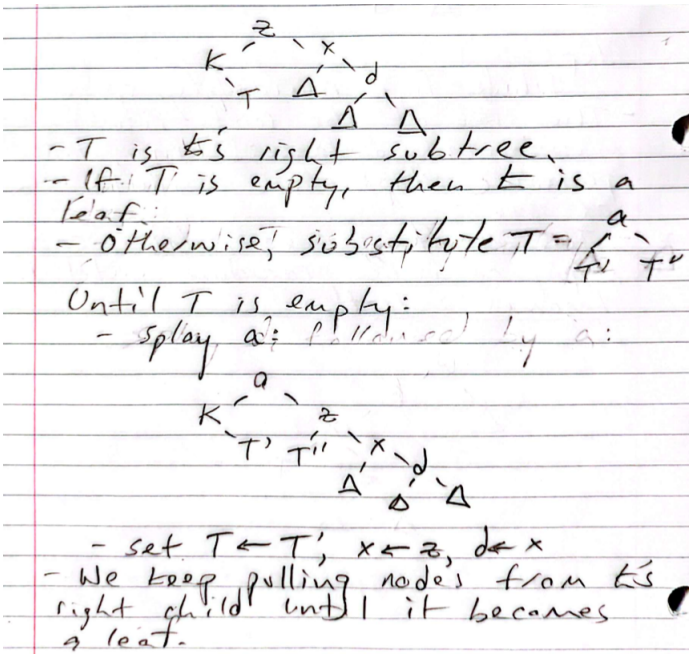
If K is the root, then splay its non-empty child to obtain (1)-(4) or their flipped variant. We can deal with the flipped versions of (1)-(4) symmetrically. Notice that

- (2) can be transformed into (1) by splaying z, K and then x. This sequence of splays works regardless of if x has a (left or right) parent or not.
- (3) can be transformed into (1) as well. Notice that y is not the minimum element (otherwise K would be the second-smallest element), so we cannot be in the leftmost path of the tree. Thus, x either has a left parent or a right parent. If x has a left parent, then splay x, K, y to obtain a flipped variant of (1). On the other hand, if it has a right parent then splay y, K, x to obtain an instance of (1). As it turns out, this sequence of splays works regardless of if x has a grandparent.
- (4) is slightly more interesting. If K's right subtree contains only a single node, then splay it to the root. Then, K will be a leaf. Otherwise, x must have a parent since K is not the minimum element. If x's parent is $w < x$, then splay K followed by w. If z is K's left child, then the resulting tree will be:

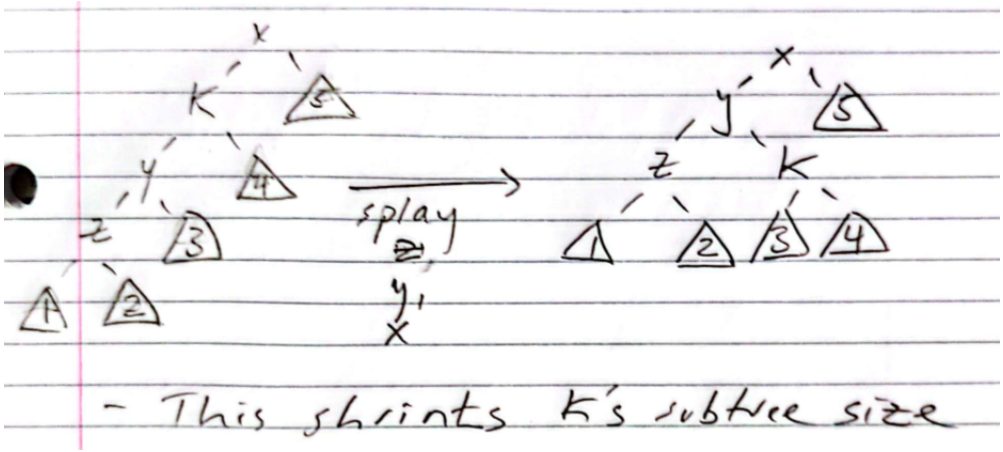


If T is empty, then we have (2). If T is not empty, then we have (1). Again, w's parent (or lack thereof) has no bearing on the result.

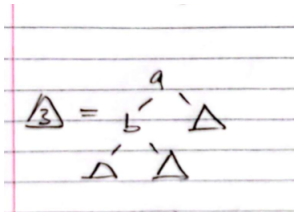
Conversely, if x's parent is $d > x$, then splay K followed by z. The resulting tree is:



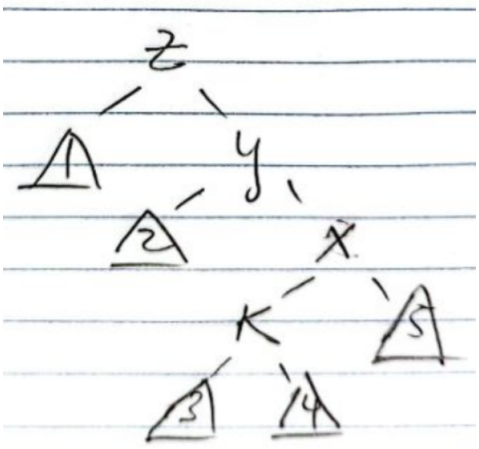
Therefore, we can either directly splay K to a leaf position, or convert any tree into an instance of (1) or its inverse. If in (1), we can strictly decrease K's subtree size then eventually it will empty, only containing K. Hence, we will have made K a leaf. We will do precisely this by splaying z, y and x.



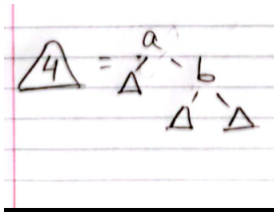
a) If K is a leaf following the procedure, then we are done.



b) If _____ then splay z:

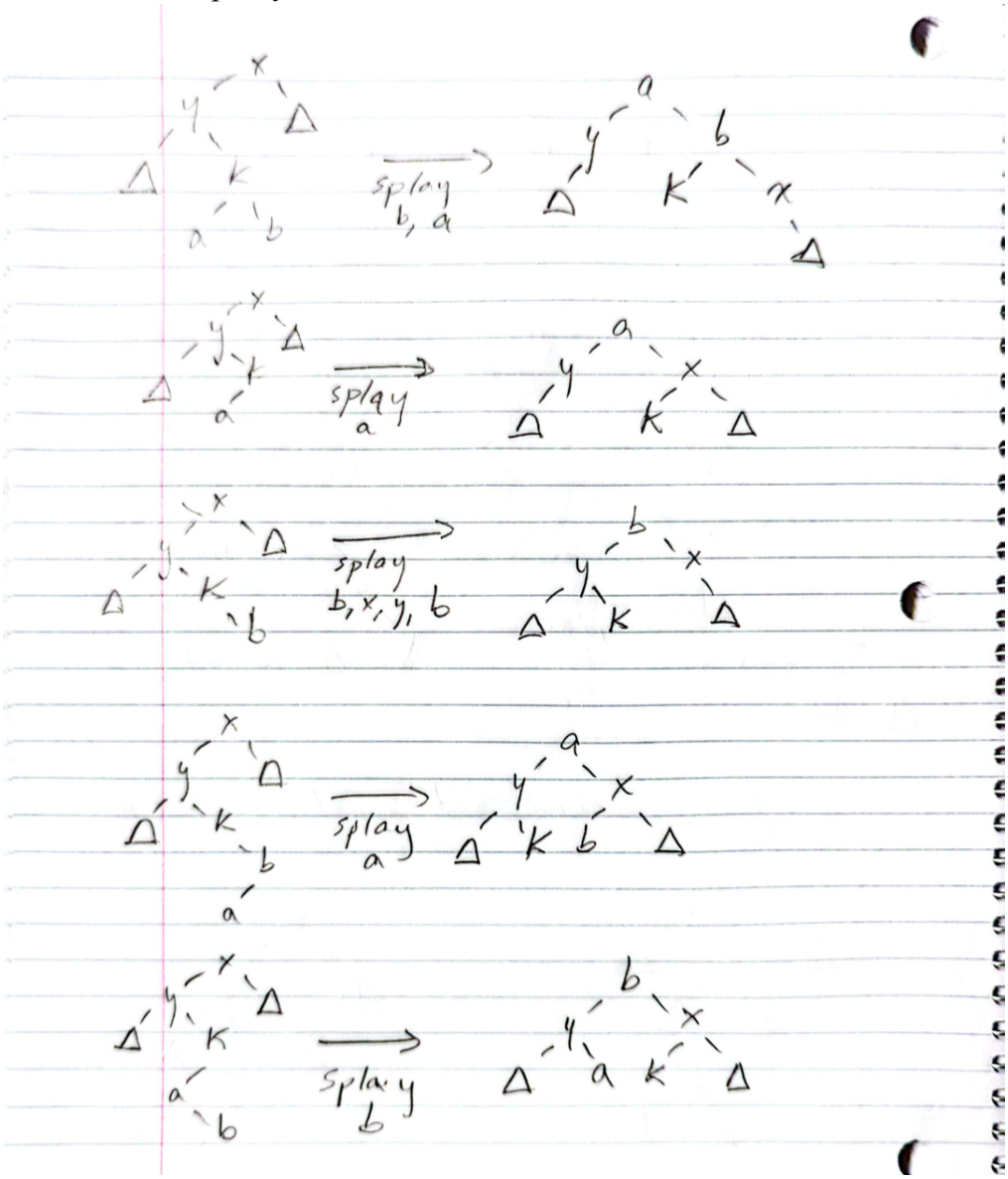


and recursively apply the procedure on x's subtree to further shrink K.



c) If _____ then recursively apply the symmetric procedure on y's subtree.

All other cases require y's subtree to be as follows:



In each figure, we present a sequence of splay operations that sink K to a leaf position. Hence, any node can be made into a leaf by a sequence of splay operations.

Now that K is a leaf, recurse on the remaining n nodes to produce the desired tree. K is not in any path that subsequent splays take, so it remains a leaf. Further, K must be in the same position in our constructed tree and our desired tree; otherwise, there is some other node for which K is in one subtree in the constructed tree, and in the other subtree in the desired tree. This is a contradiction since K is either less than the node or greater than it. Hence, we have produced our desired tree using only splays.