Balanced Binary Search Trees

sorted order is
given by an
inorder traversal
rotateLeft(y)

between z, y

t + t change in their parent/child relation

rotateRight(y)

between x, y
Requires parent is not null

BSTNode liftUp(BSTNode y) {
    BSTNode parent = y.parent;
    if (y.isLeftChild())
        rotateRight(parent);
    else
        rotateLeft(parent);
    return parent;
}
BSTNode liftUp(BSTNode y) {
    BSTNode parent = y.parent;
    if (y.isLeftChild())
        rotateRight(parent);
    else
        rotateLeft(parent);
    return parent;
}
We can use rotations to help keep a binary search tree balanced while maintaining InOrder property. Completely structural (no new comparisons)
Difference between different data structures that are balanced binary search trees is how you decide when and where to rotate.

Red-black trees (after full break)
Guarantees height $\leq 2 \log_2(n+1)$
Book includes

Splay tree - amortized
rotates each element
accessed/insert to root

AVL trees - red-black trees
are a better choice