Balanced Binary Search Trees

Sorted order is given by an inorder traversal.
Changes in their parent/child relationship:

- `y` becomes the parent of `z`.

Operations performed:

- `rotateLeft(y)`
- `rotateRight(y)`

Relationships:

- `y` is between `x` and `z`.
- `x` is between `z` and `y`.

Notes:

- Move "weight" left.
- One subtree reattaches.
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
Midterm Topics 100 pt exam

Review Hws & practice problems.
Bring one 8½×11 crib sheet

Divide-and-Conquer Algorithms

15-20 Should be able to design with good hints
use
Should be able to analyze (master method)