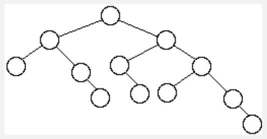
AVL Trees

An AVL tree is a BST + a height balance property:

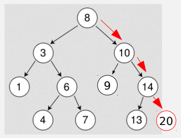
- for every node, the height of the node's left subtree is no more than one different from the height of the node's right subtree



The height balance property ensures that the height of an AVL tree with n nodes is O(log n).

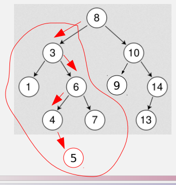
Insert

- structural property dictates that insertion only occurs at a node with fewer than 2 children
- ordering property dictates where



insert 20

no height-balance violations – we're done!



insert 5

height-balance property violated – uh oh!

Operations on AVL Trees

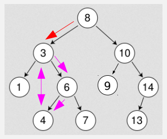
An AVL tree is a BST, so the find operation is no different.

For insert and remove:

- insert/remove as dictated by the (BST) structural and ordering rules
- fix up the broken balance property as needed

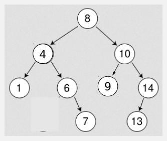
Remove

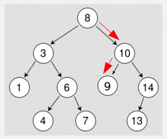
- structural property dictates that removal only occurs at a node with fewer than 2 children
  - may need to swap desired element with next larger/smaller in order to satisfy the structural property



remove 3

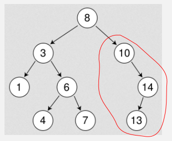
swap with 4 and remove
no height-balance violations – we're done!





remove 9

height-balance property violated – uh oh!

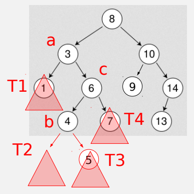


Restructuring

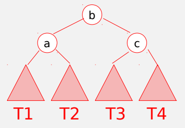
Both insertion and deletion may break the height balance property.

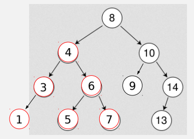
Restore it by performing one or more *restructuring operations* (or *rotations*).

Restructuring



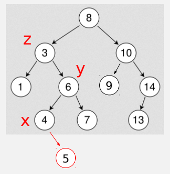
rearrange as shown:





height balance property restored!

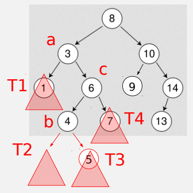
Restructuring



let *z* be the first unbalanced node (working up the tree from the point of insertion/deletion)

let *y* be z's tallest child

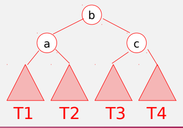
let *x* be y's tallest child



relabel x, y, z as a, b, c according to their correct sorted order

label the other subtree children of a, b, c as T1, T2, T3, T4 according to their correct sorted order

rearrange as shown:



Restructuring

How many restructuring operations are needed?

Observation.

- restructuring reduces the height of a subtree

Insertion –

- insertion increases the height of a subtree, so one restructuring is sufficient to shorten it and restore balance

Removal –

- removal decreases the height of a subtree, so one restructuring may only result in pushing the imbalance higher up the tree
- O(log n) restructurings may be required

Running Time

- initial BST insert/remove – O(log n)
- number of nodes to check for balance – O(log n)
- time to perform a balance check – O(1) if height info is stored for each node
- time to perform one restructuring – O(1)
- number of restructurings performed – 1 for insertion, O(log n) for removal
- time to update stored balance information – O(log n) nodes affected, O(1) per

Total time: O(log n) for insert/remove