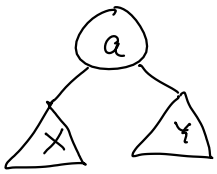
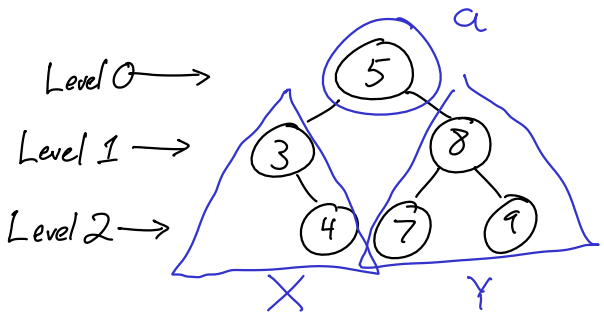
Binary Tree : a tree where each node has at most one left child and one right child

Traversals — visits everything (here: put in a list)



rec
- Pre-order: a, X, Y → 5, 3, 4, 8, 7, 9
- In-order: X, a, Y → 3, 4, 5, 7, 8, 9
- Post-order: X, Y, a → 4, 3, 7, 9, 8, 5

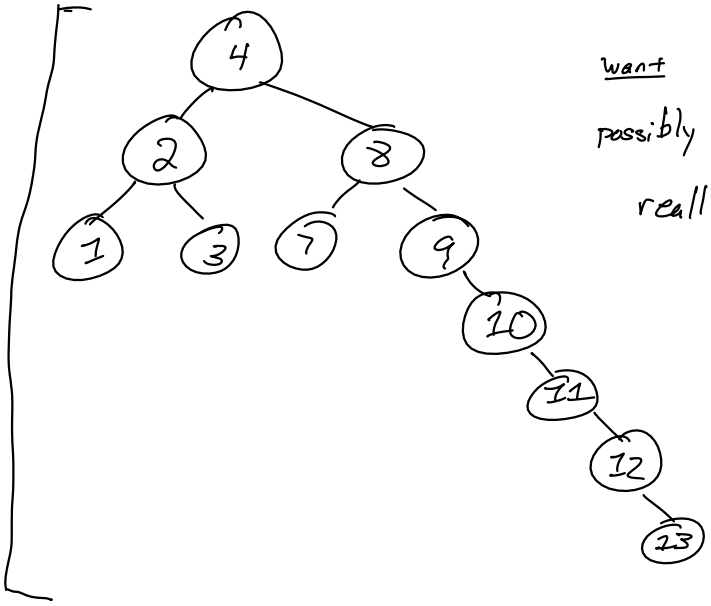
BFS
- Level-order → 5, 3, 8, 4, 7, 9



Binary Search Trees:

binary tree where all left descendants have lesser key than each node and all rights ..... greater .....

height



want $O(\log n)$

possibly $O(n)$

really $O(h)$

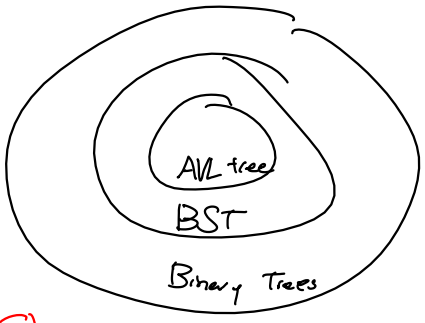
Dictionary (ADT)

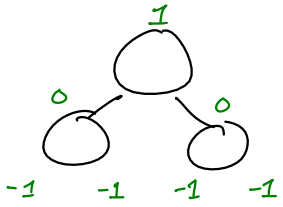
- get (K)
- insert (K, V)
- update (K, V)
- remove (K)

Node

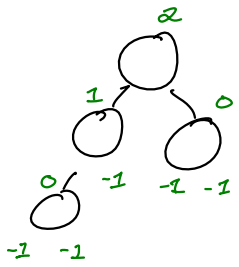
- key
- value
- left
- right
- height

AVL tree — every node has a left subtree and a right subtree whose heights differ by no more than one

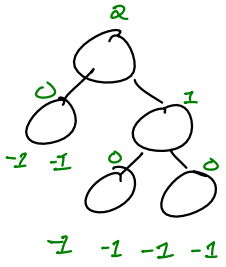




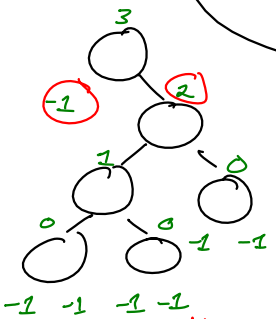
AVL



AVL

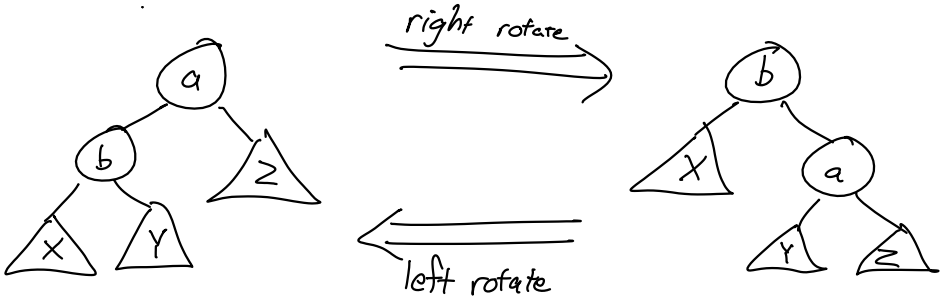


AVL



not AVL

Tree Rotation



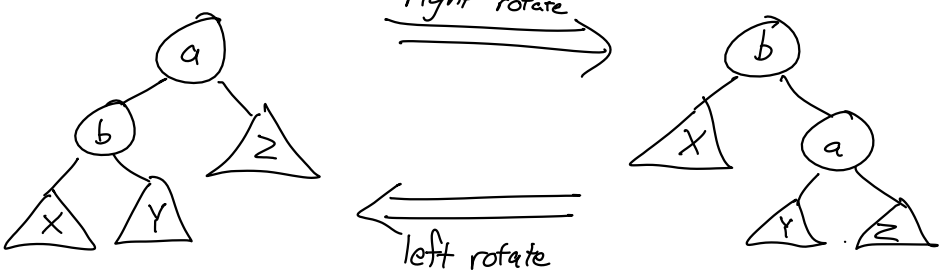
If I rotate right and the tree was a BST, is it still a BST?

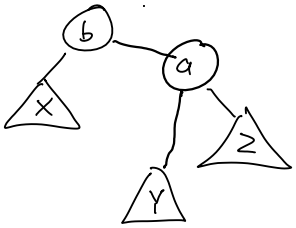
Know?

$b < a$
$b < Y < a$
$X < b$
$a < Z$

Need?

$X < b$
$b < a$
$a < Z$
$b < Y < a$



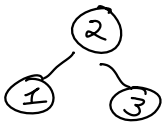



```
Function rightRotate(tree):
    oldroot ← tree
    tree ← tree→left
    oldroot→left ← tree→right→right
    Return tree
EndFunction
```

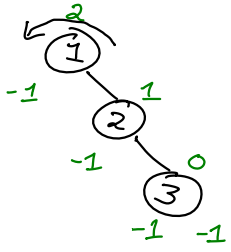
don't use this one

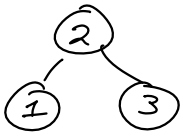
```
Function rightRotate(tree):
    a ← tree
    b ← a→left
    X ← b→left
    Y ← b→right
    Z ← a→right
    a→left ← Y
    b→right ← a
    Return b
```

2, 1, 3



1, 2, 3






```
insertInSubtree(key, value, node)
  if node is an empty tree:
        return new node(key, value)
  else if key < node->key:
      node->left <- insertInSubtree(key, value, node->left)
      node <- rebalance(node)
      recalculate Height(node)
      Return node
  else ....
```