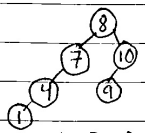
Trees

Binary search tree

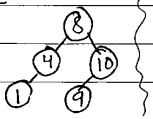
- insert 8, 7, 4, 10, 9, 1 into a bst.



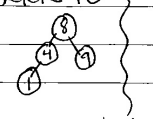
* if smaller go left and if save is a

- average height of a BST is $\Theta(\sqrt{n})$
- full tree height $\lceil \log_2(n) \rceil - 1$ ← celing
- degenerate is $n-1$

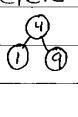
- delete 7



delete 10



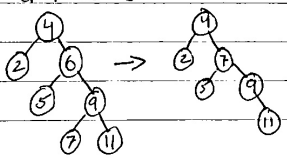
delete 8



- When we delete the root grab the [redacted] value from the [redacted] right subtree

smallest

- delete 6



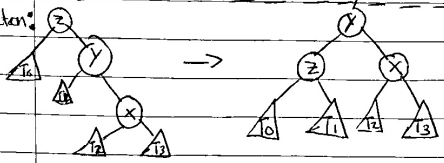
- The 7 is the smallest in the right sub tree so we lift that node up.
- For a balanced tree operations are $O(\log(n))$
- for degenerate costs of operations are $O(n)$
- If we allow deletes average tree height is $O(\sqrt{n})$

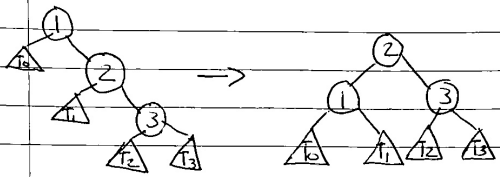
AVL Trees Adelson-Velskii and Landis

- a bst with a "balance" condition

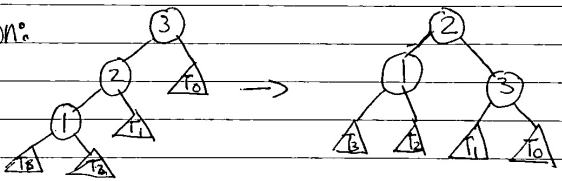
- Let y be the node where imbalance occurs.
- Let z be the parent of y
- Let x be the child of y that causes imbalance.

Left Rotation:



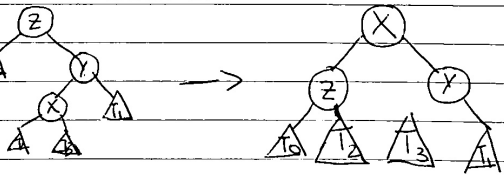


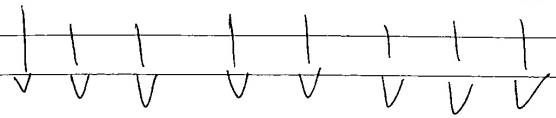
Right Rotation:

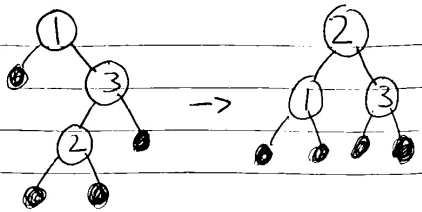


x is the middle

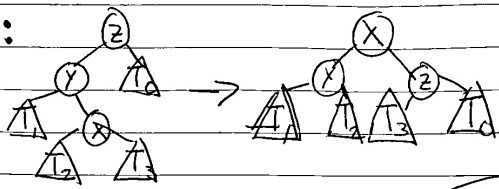
Right-Left:







Left-Right:



- With AVL tree with 1023 nodes should have a height around 13 or $1.44 * \log_2(1023) - 1.328$
- This tree works way better than without balance condition.

Splay Tree *still is a BST* Not strictly balanced like AVL

- Data locality can be important so when we search for a node or access it we Make it the root. (Done by rotations)
- Amortized cost guarantees that m consecutive operation is $O(m \log_2 n)$
- if m operations are $O(m \cdot f(n))$
- the amortized cost is $O(f(n))$
- Splaying happens when we search, insert, or delete
- During delete a node the nodes parent node is splayed to the root.

Zig Rotations:

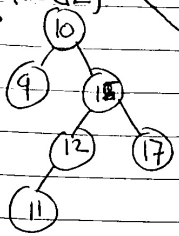
Case1: Search item is root node

Case2: Search item is child of root node
- left child
- right child

case 3 : node has parent & grandparent (Zig-Zig)

case 4 : node has parent & grandparent not symetrical (Zig-Zag)

find(12)



find(13)