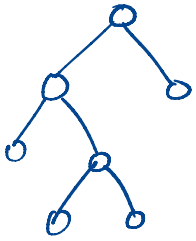
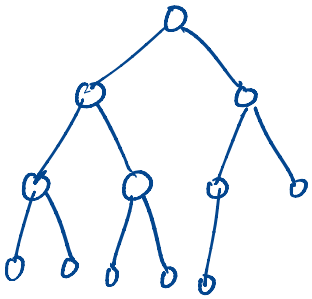
AVL Trees - I

- Full and complete binary tree



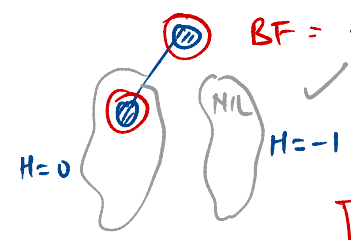
↳ Full binary tree: is a binary tree where every node has either two children or none.

↳ Complete binary tree: is a binary tree where every level except possibly the last is completely filled and the nodes in the last level are as far left as possible.



↳ Balanced binary tree: left and right subtrees of every node differ in height by no more than 1

# of edges from topmost node to the farthest node in the subtree.

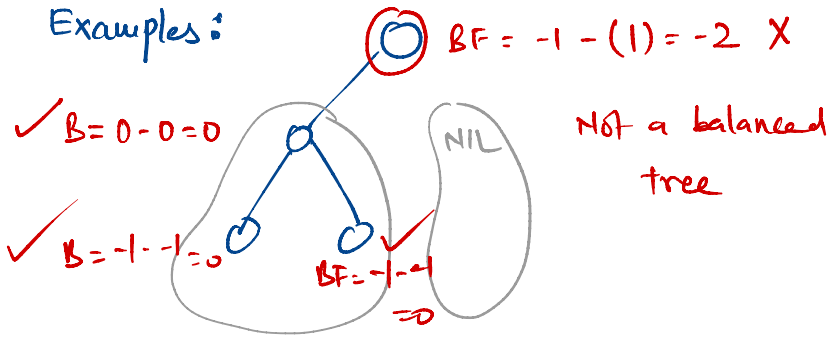


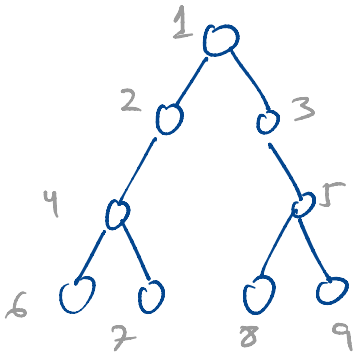
Balance Factor $BF(x) = Height(x.right) - Height(x.left)$

→ for every node

$BF \in \{-1, 0, 1\}$ → Permissible for balanced node

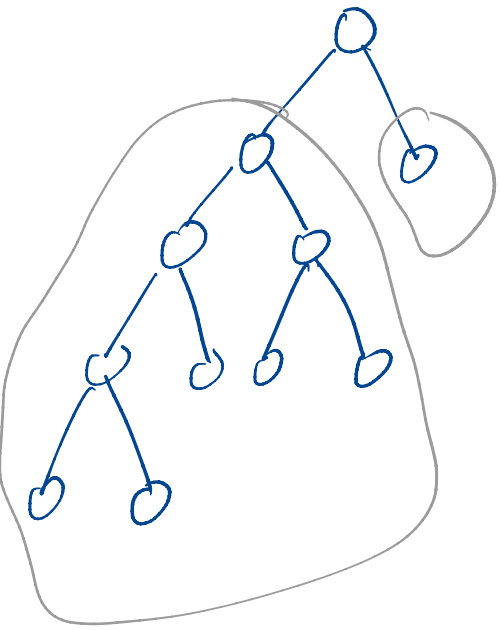
Examples:





Imbalance at nodes: 2, 3

a full binary tree that's not balanced



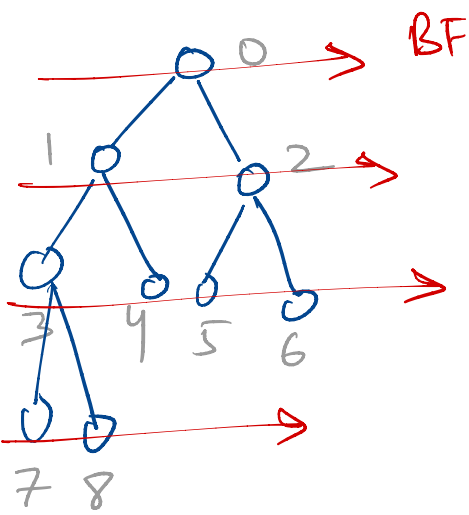
→ Array representation of a complete binary tree

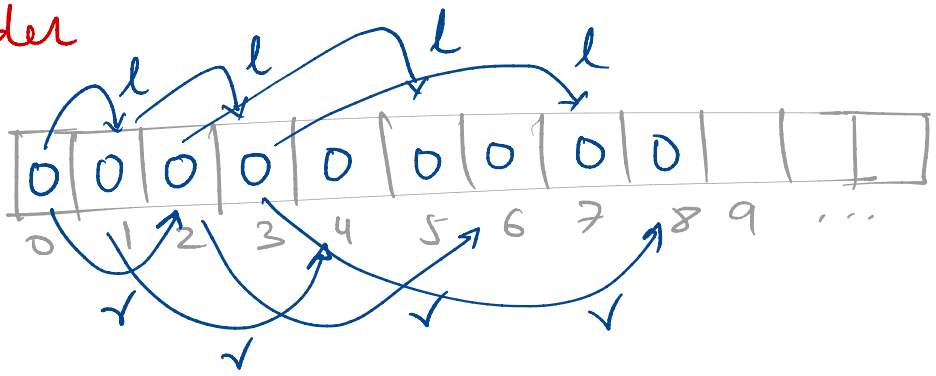
↳ Binary trees can be stored in breadth-first order as an implicit data structure in an array, and if the tree is complete then this method wastes no space.

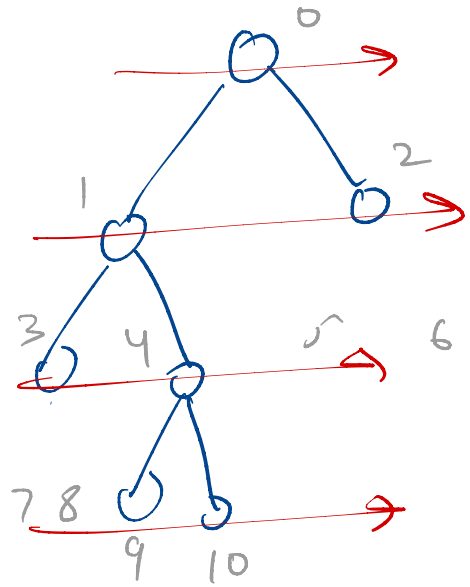
↳ If a node in the binary tree has an index i in the array then its left child has an index of $(2i+1)$ & right child has a index of $(2i+2)$

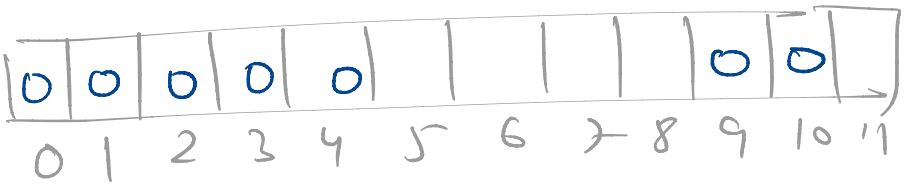
→ The parent of a node at index i is found at an index $\left\lfloor \frac{i-1}{2} \right\rfloor$ → floor

→ Root is stored at index 0.









→ This is a compact storage with better locality of reference, however it is expensive to grow and wastes space proportional to $(2^h - n)$ for a tree of height h & nodes n.

→ Tree operations are faster if height of the tree is small.

AVL trees

Goal: To maintain logarithmic height. this is the smallest we can make it. i.e., $h = \log_2 n$, n: # of nodes

↳ Therefore we somehow need to balance the tree.

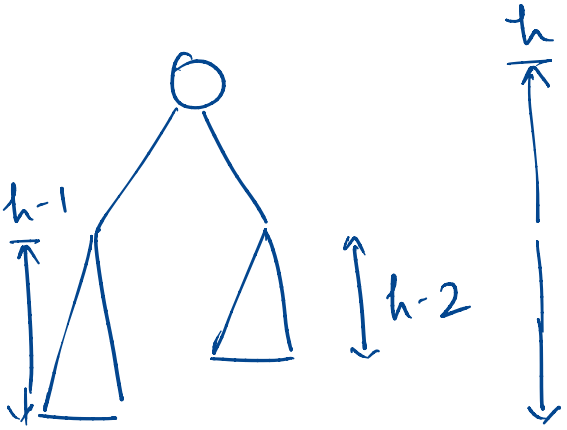
↳ Solution proposed by G.M Adelson - Velskii and E.M. Landis in 1962

AVL

↳ Average search achieved in $1.44 \log_2 n$ comparisons (pretty close to $\log_2 n$)

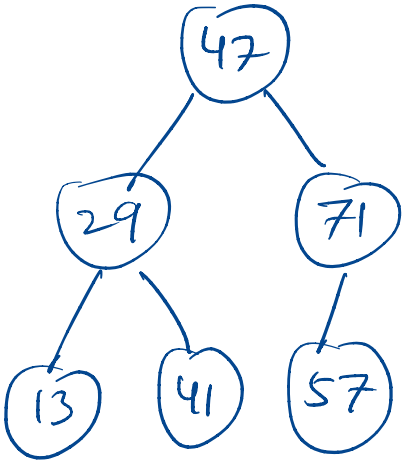
Definition: An AVL tree is a BST with following properties:

1. The subtrees of every node differ in height by at most one.
2. Every subtree is an AVL tree

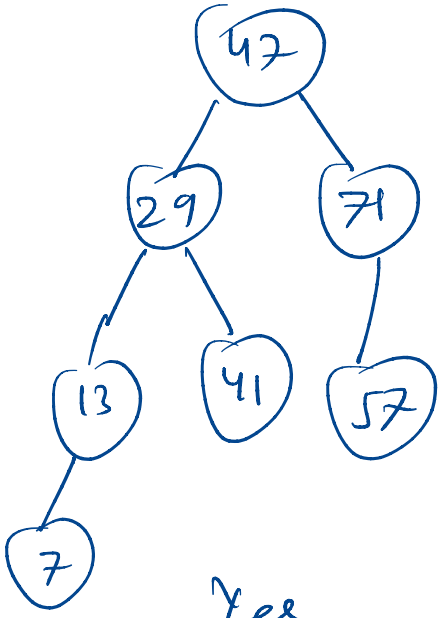


← Balance requirement in an AVL tree

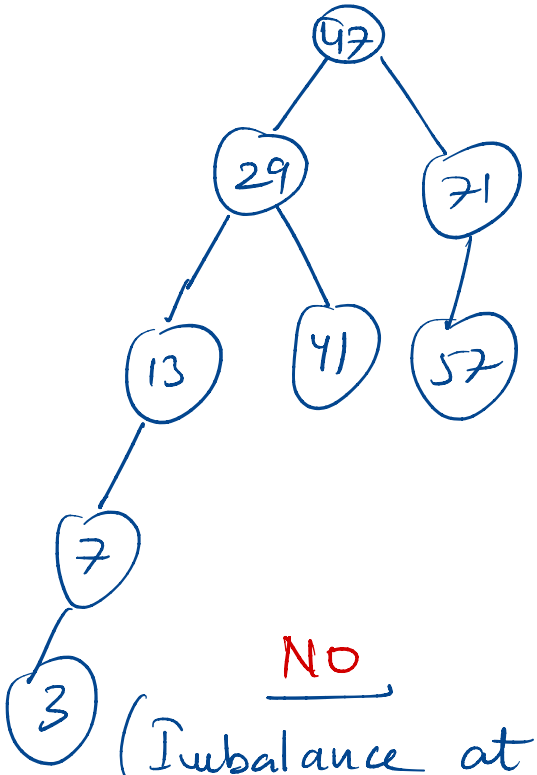
Examples of AVL trees :



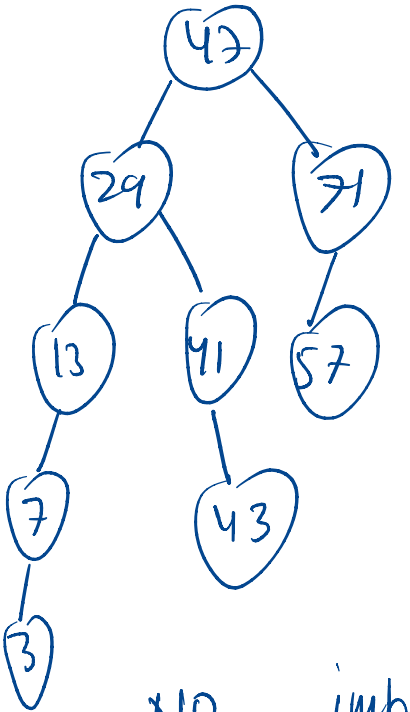
Yes



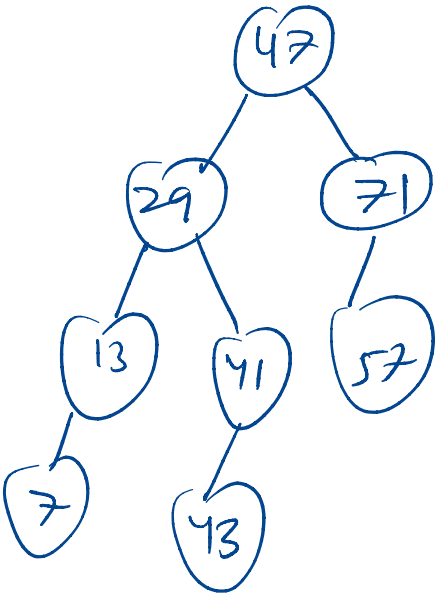
Yes



NO (Imbalance at 13, 29, 47)



NO imbalance: 13, 47



NO

Not a BST!

→ An AVL tree needs to be rebalanced after each deletion and insertion to keep the tree balanced

↳ This is done by using rotations.

↳ Insertion / deletion handled the same way as in a BST

↳ Uses a single rotation or a double rotation to keep the balance.

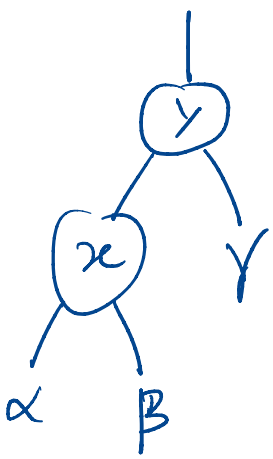
→ Left heavy tree : if $BF(x) < 0$

Right " " : if $BF(x) > 0$

Balanced node : if $BF(x) = 0$

Rotations: A local operation that preserves the BST property

↳ left rotation, right rotation



Left-rotate (T, x)

Right rotate (T, y)