Algorithms & Data Structures I: AVL Tree

Mikhail Anukhin

Today's Topics

- Binary Search Tree recall
- AVL Tree definition
- Why AVL tree is balanced
- Insert in AVL Tree
    - Rotations
- AVL tree sorting

Binary Search Tree (BST)

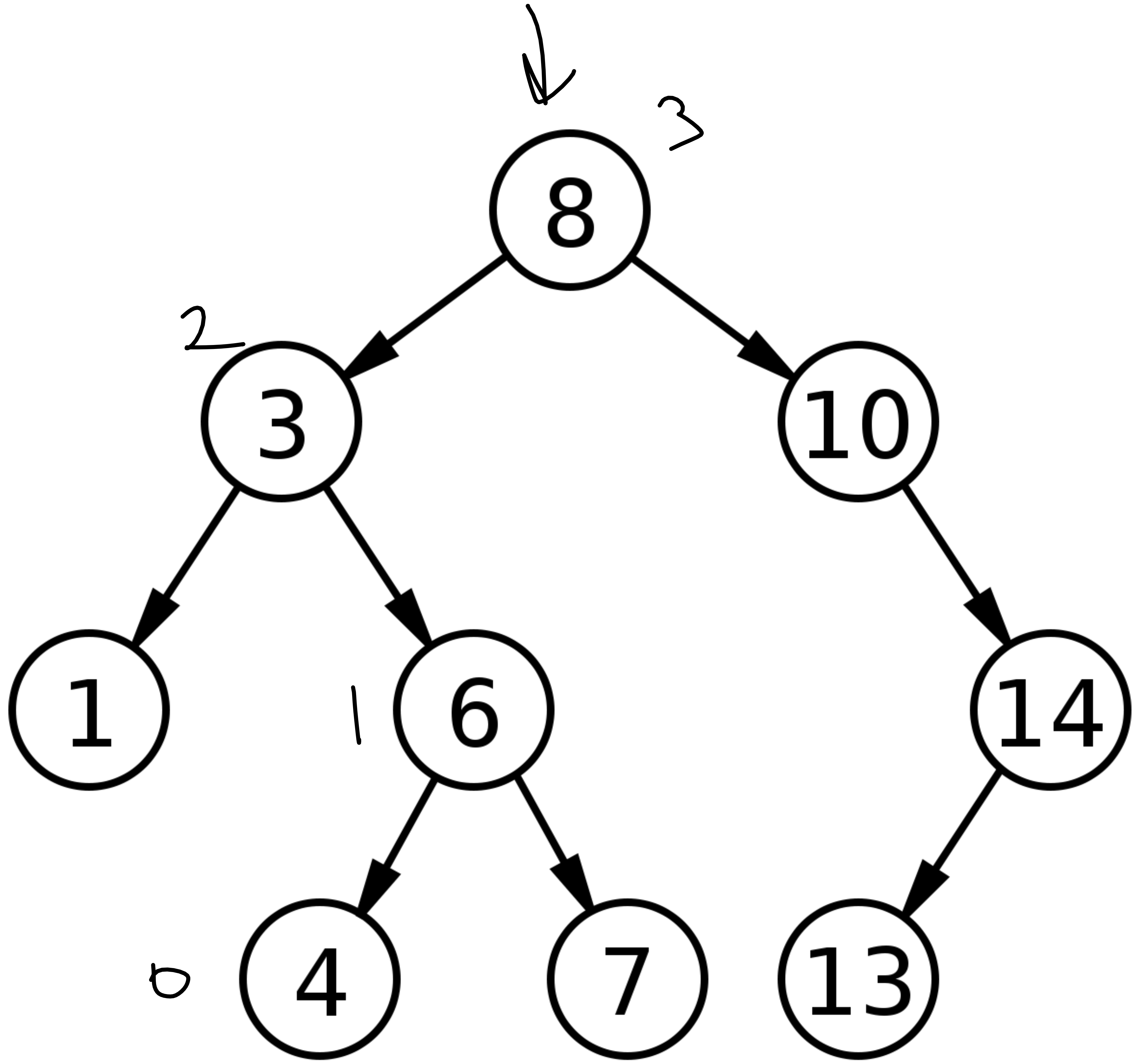
Properties

- Each node x in the binary tree has a key *key(x)*
- Nodes other than the root have a parent *p(x)*
- Nodes may have a left child *left(x)* and/or a right child *right(x)*. These are **pointers** unlike in a heap

Main Invariant

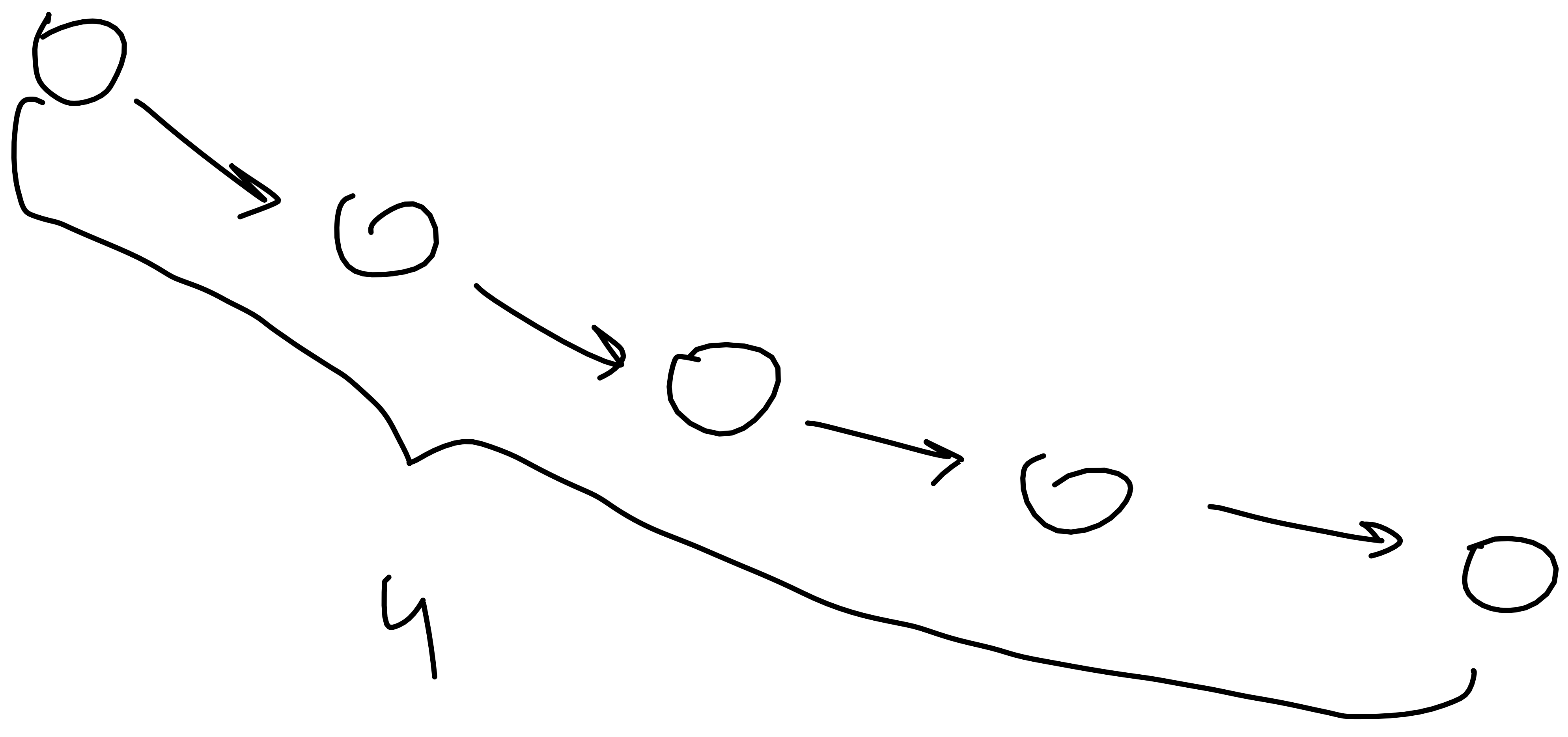
For any node x:

1. for all nodes y in the left subtree of x, *key(y)* ≤ *key(x)*
2. for all nodes y in the right subtree of x, *key(y)* ≥ *key(x)*



Height of a binary search tree

Consider a tree contains **n** elements. What is a height of a tree?

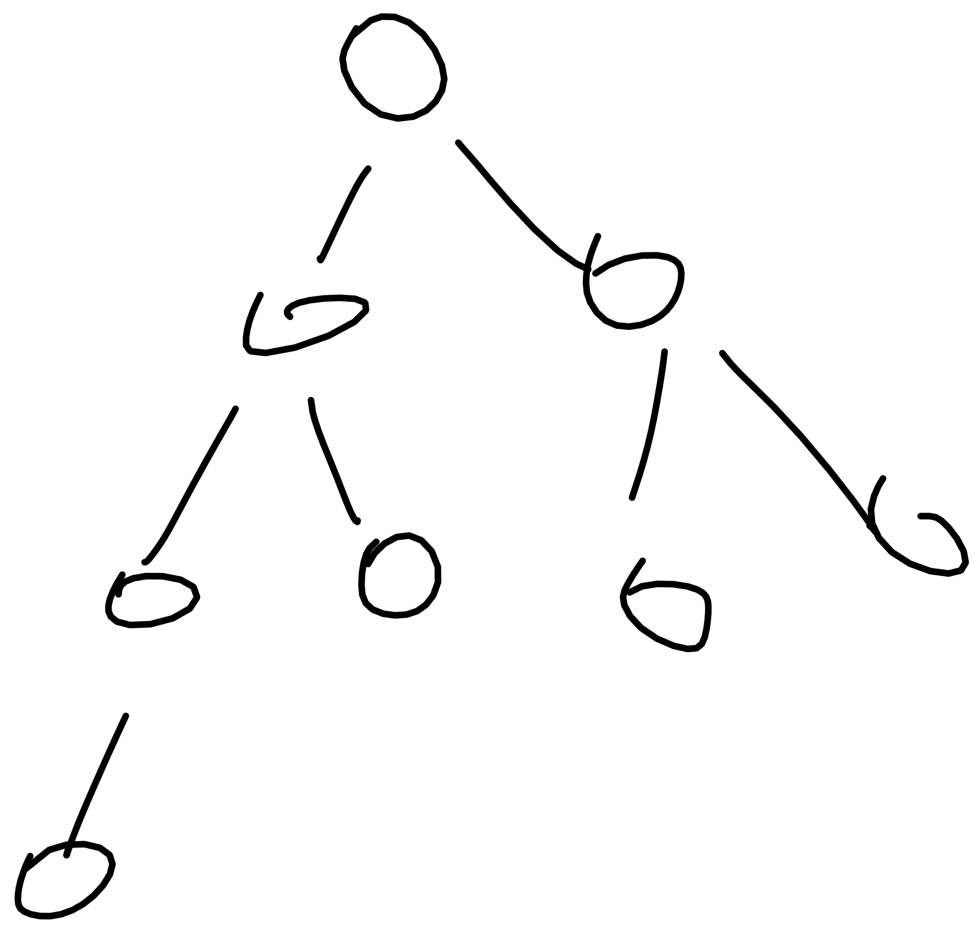


Balanced Binary Search Tree

Height of a tree is a length of the longest path from the root to a leaf

If the height of a tree is asymptotically O(log n), then a tree is **balanced**

log n + 1



BST Operations

Insertion, Find and Deletion operation is O(h). That can be O(n) in the worst case, and is O(logn) if a tree is balanced.

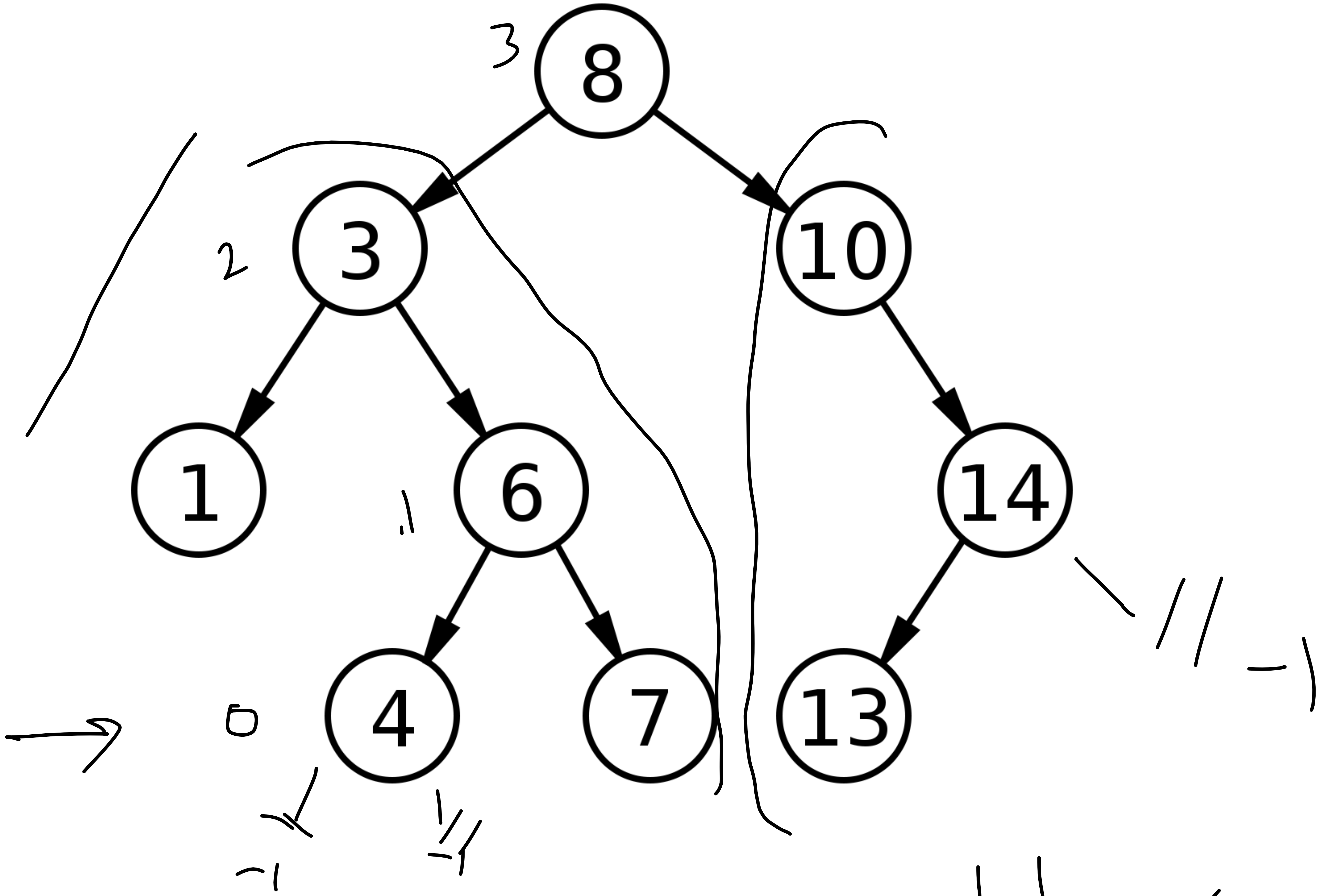
Insert(value): O(h)

Find(value): O(h)

Delete(value): O(h)

FindMax(): O(h)

How to find BST height?



$\rightarrow H(x) = \max(x \rightarrow right, x \rightarrow left) + 1$

if $x == nullptr \Rightarrow$ return -1

Mikhail Anukhin

How to find BST height?

```
Struct Node {
    Node* left;
    Node* right;
    T data;
    int heigt;
}
```

AVL Tree: Adelson-Velskii & Landis 1962

Properties

- Each node x in the binary tree has a key *key(x)*
- Nodes other than the root have a parent *p(x)*
- Nodes may have a left child *left(x)* and/or a right child *right(x)*. These are **pointers** unlike in a heap

Search Invariant

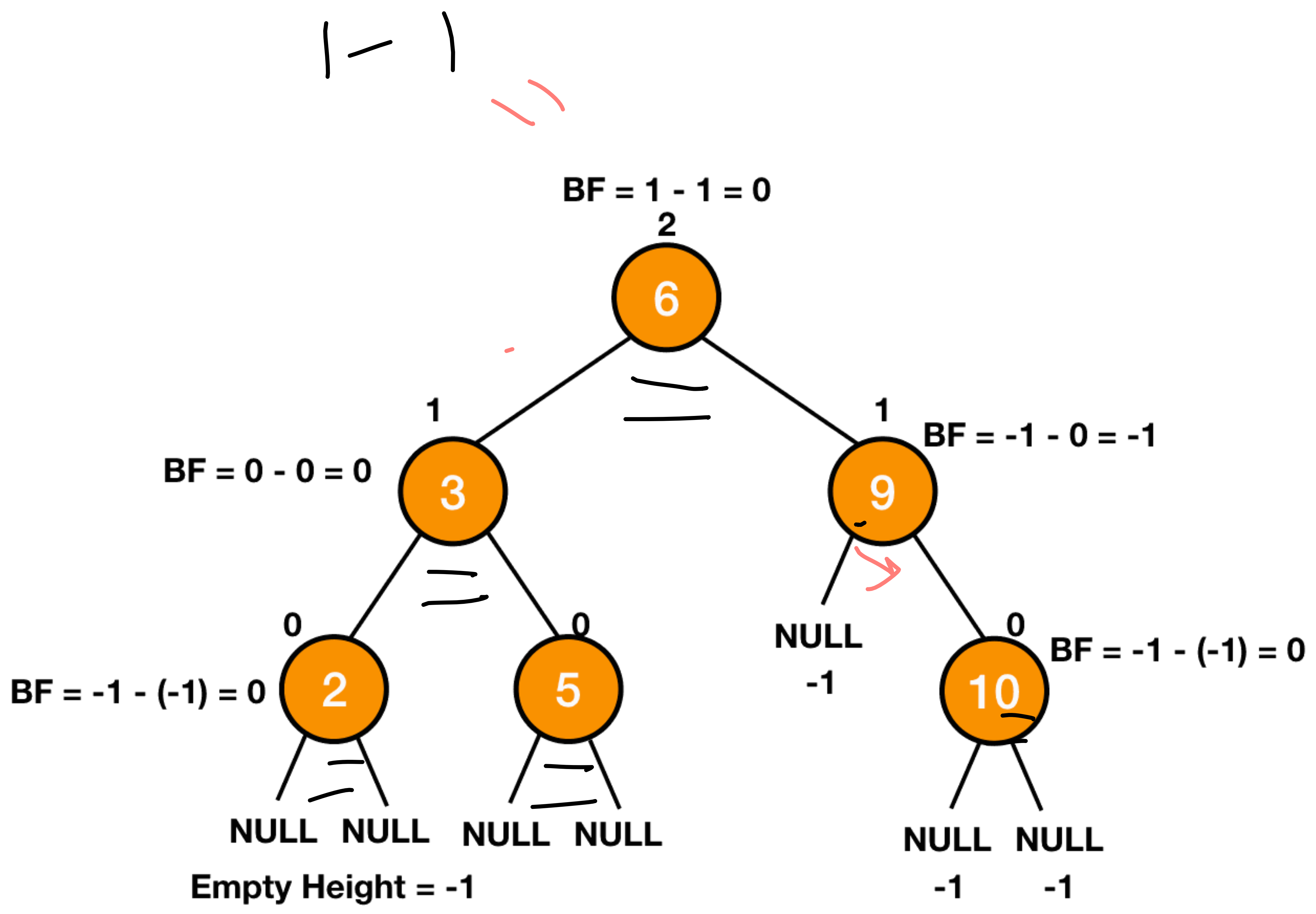
For any node x:

1. for all nodes y in the left subtree of x, *key(y)* ≤ *key(x)*
2. for all nodes y in the right subtree of x, *key(y)* ≥ *key(x)*

AVL Invariant

For any node x:

1. require heights of left & right children to differ by at most ±1

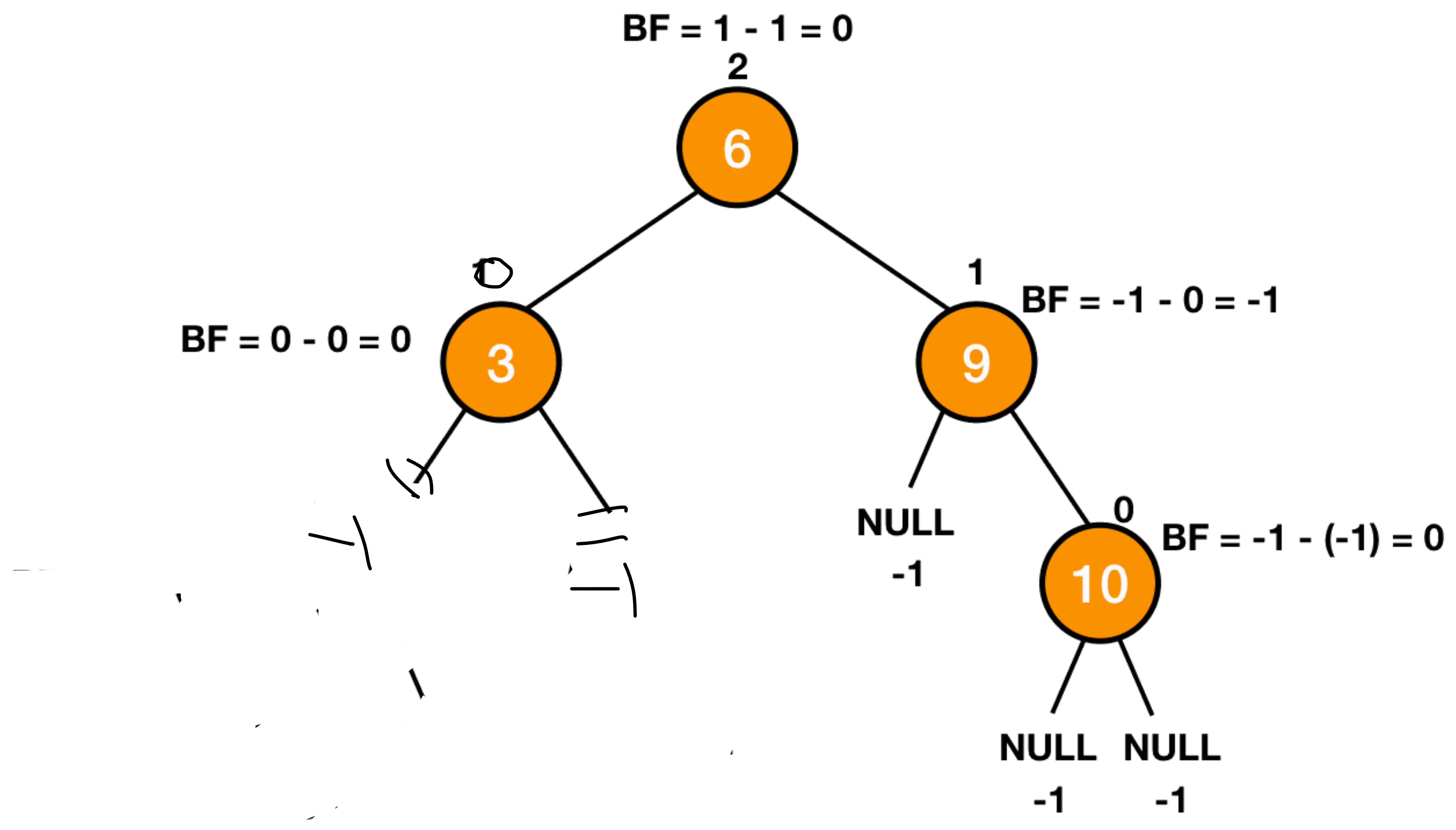


BF = Balance Factor

AVL Tree: Adelson-Velskii & Landis 1962

AVL Tree implementation properties

1. Treat nullptr tree as height - 1
2. Store height of node in every node



AVL Tree is balanced

Worst when every node differs by 1.

N_h = min # nodes in height-h AVL tree

n

$h < 2\lg(N_h)$

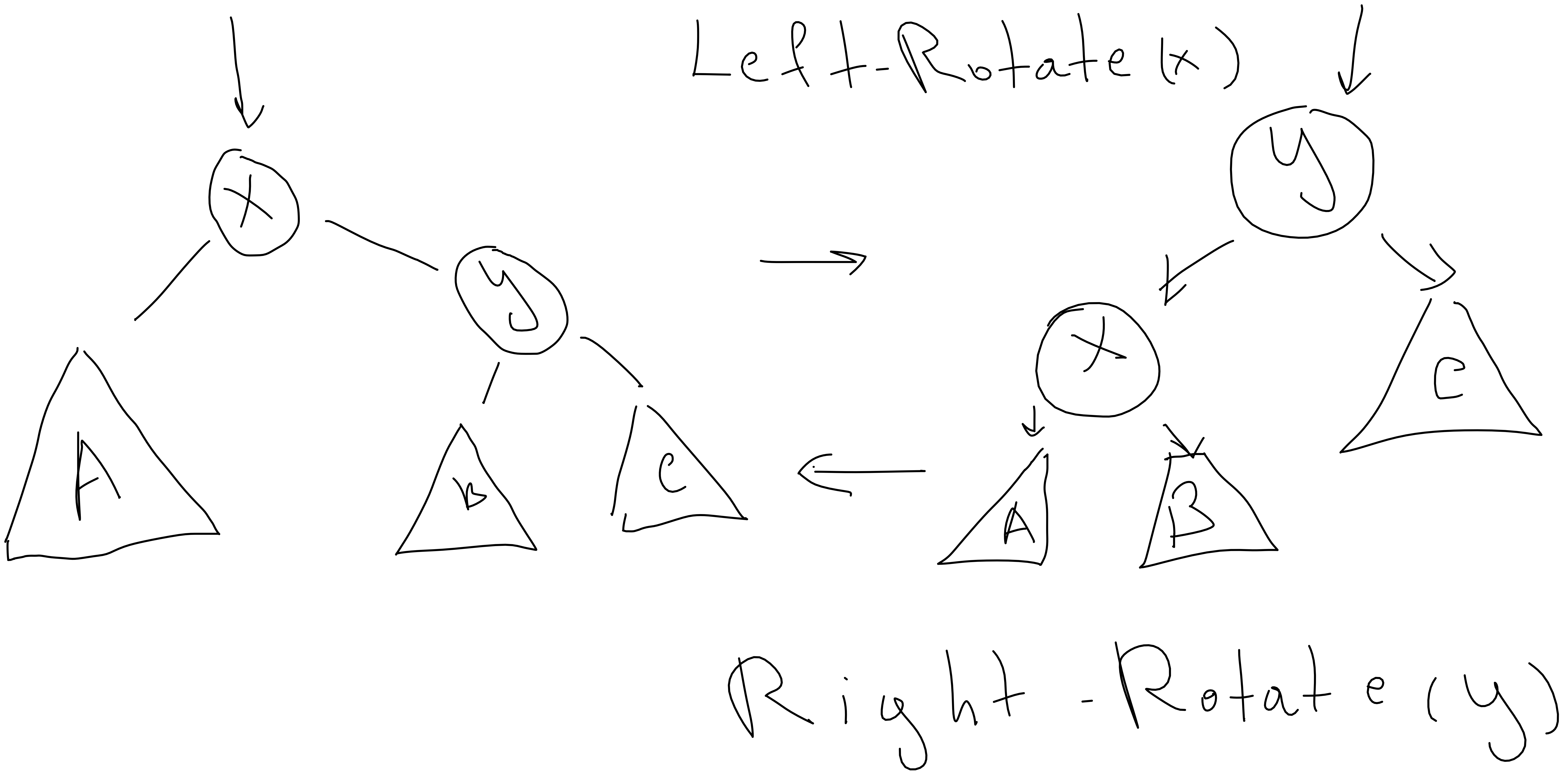
$N_h = N_{h-1} + N_{h-2} + 1$

$N_h > 2N_{h-2} \Rightarrow N_h > 2^{h/2} \Rightarrow$

Mikhail Anukhin

How to keep tree balanced? Rotate!

AVL Rotations

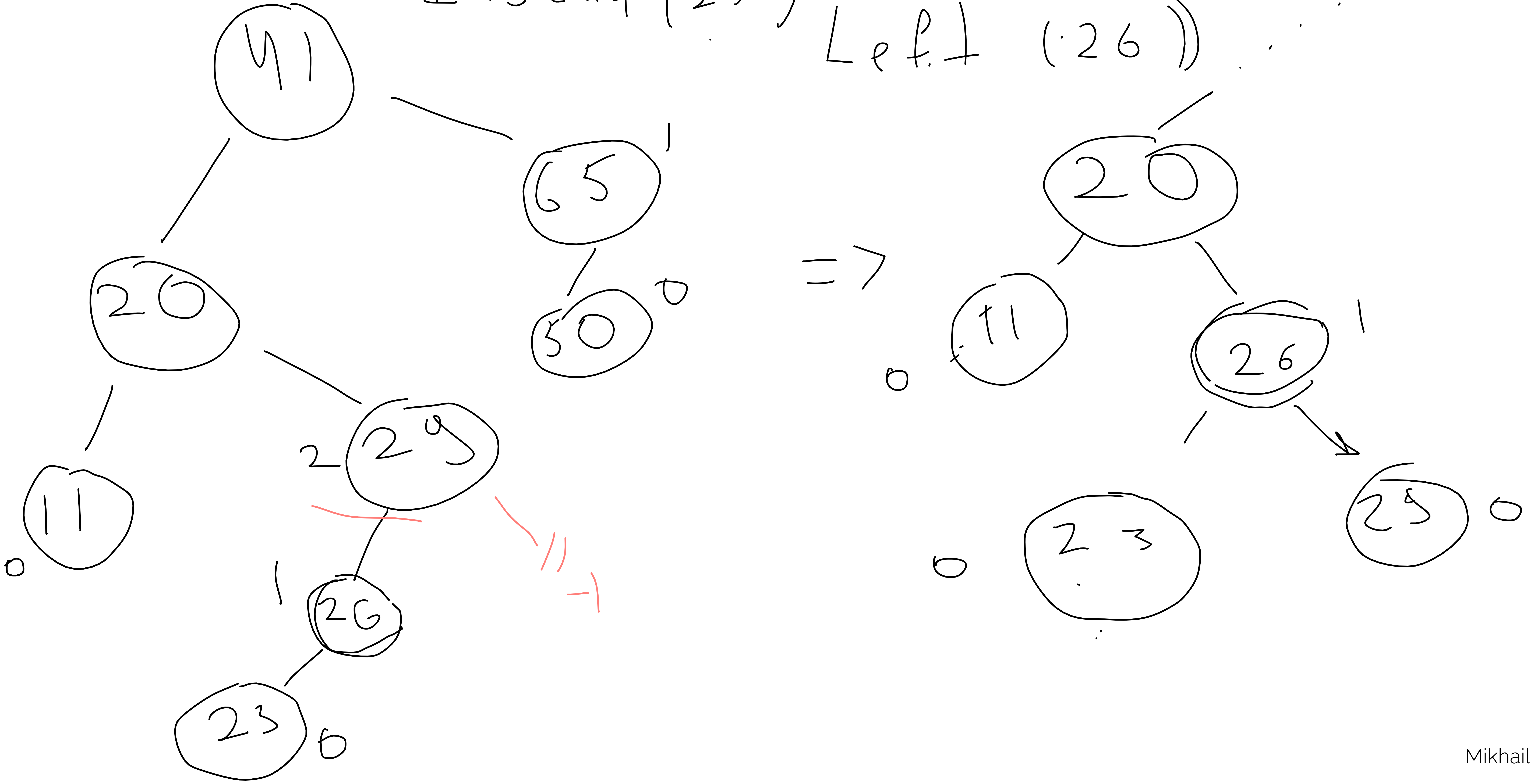


Mikhail Anukhin

AVL Insert Example

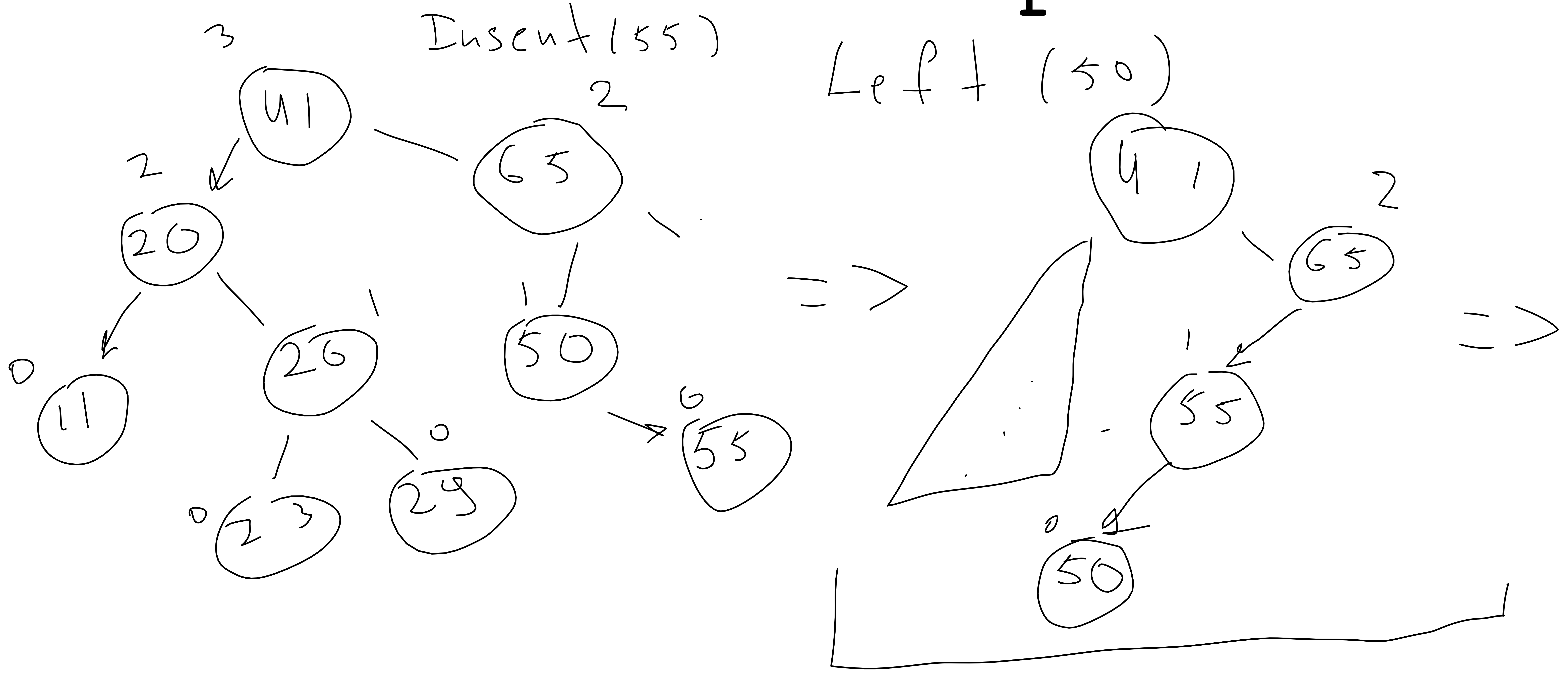
Insert(23)

Left(26)



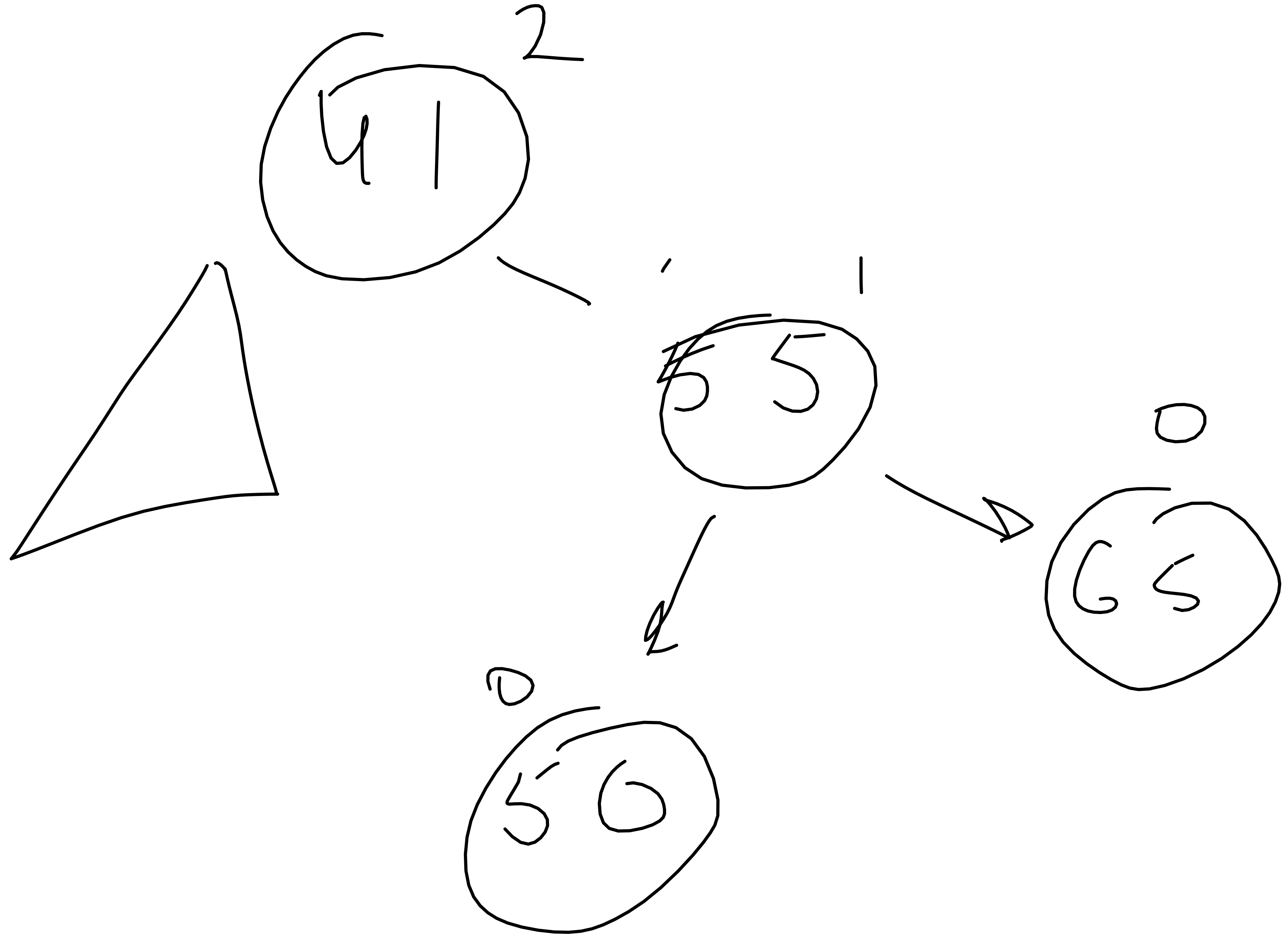
Mikhail Anukhin

One more example

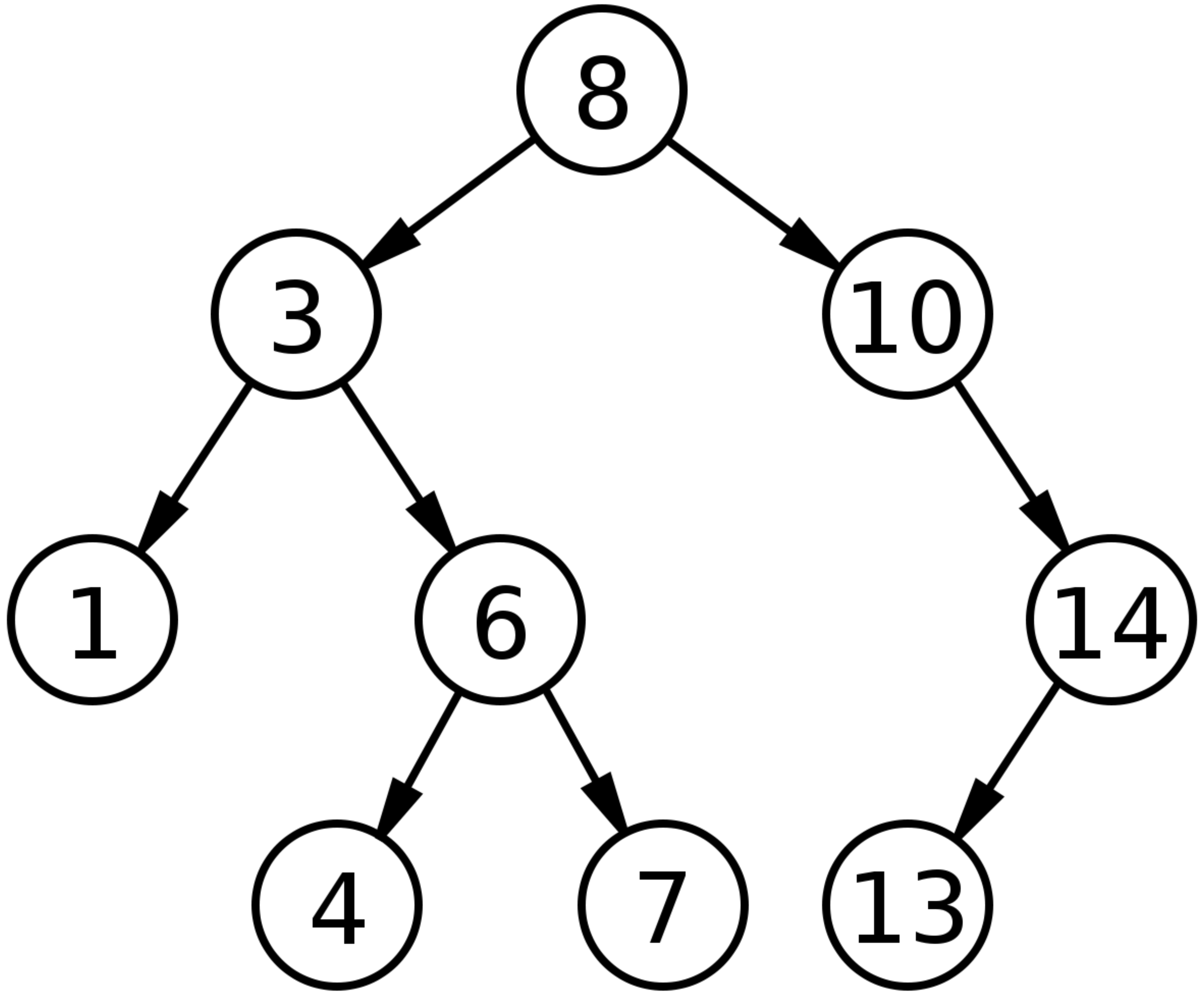


Mikhail Anukhin

=> Right 65



AVL Insert General case

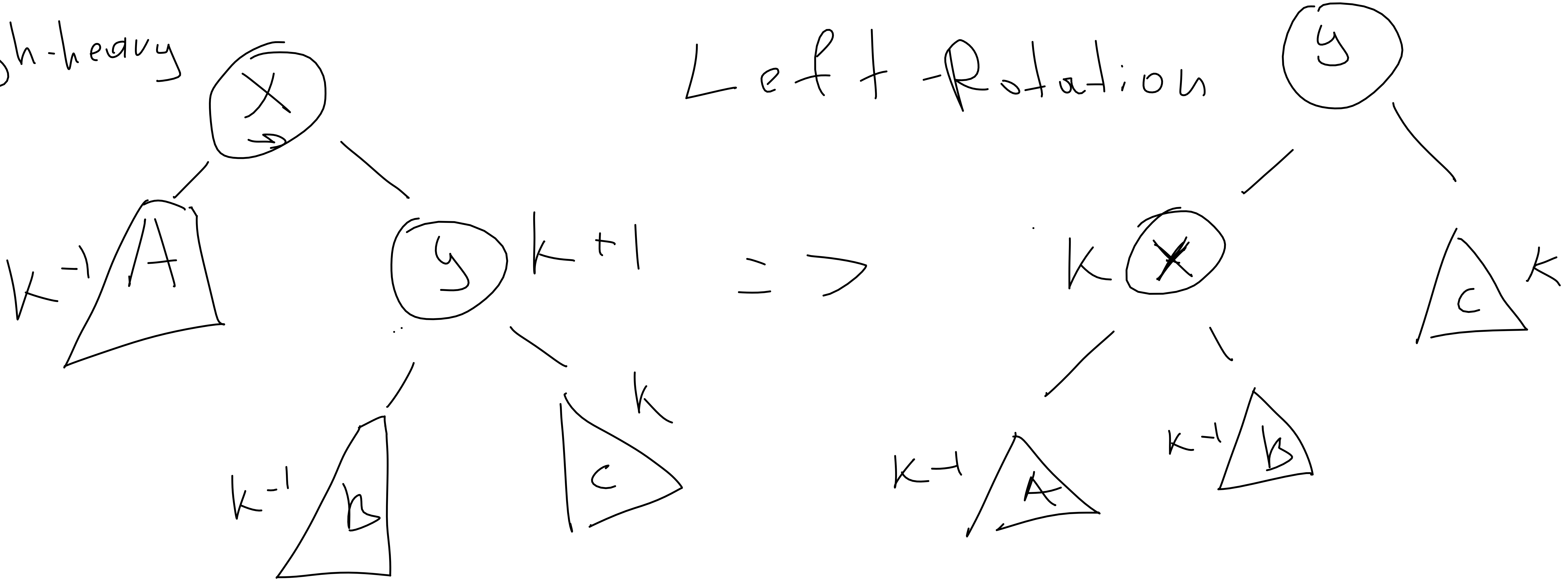


Mikhail Anukhin

- Node x is lowest violator
- x is right-heavy (left case is symmetric)

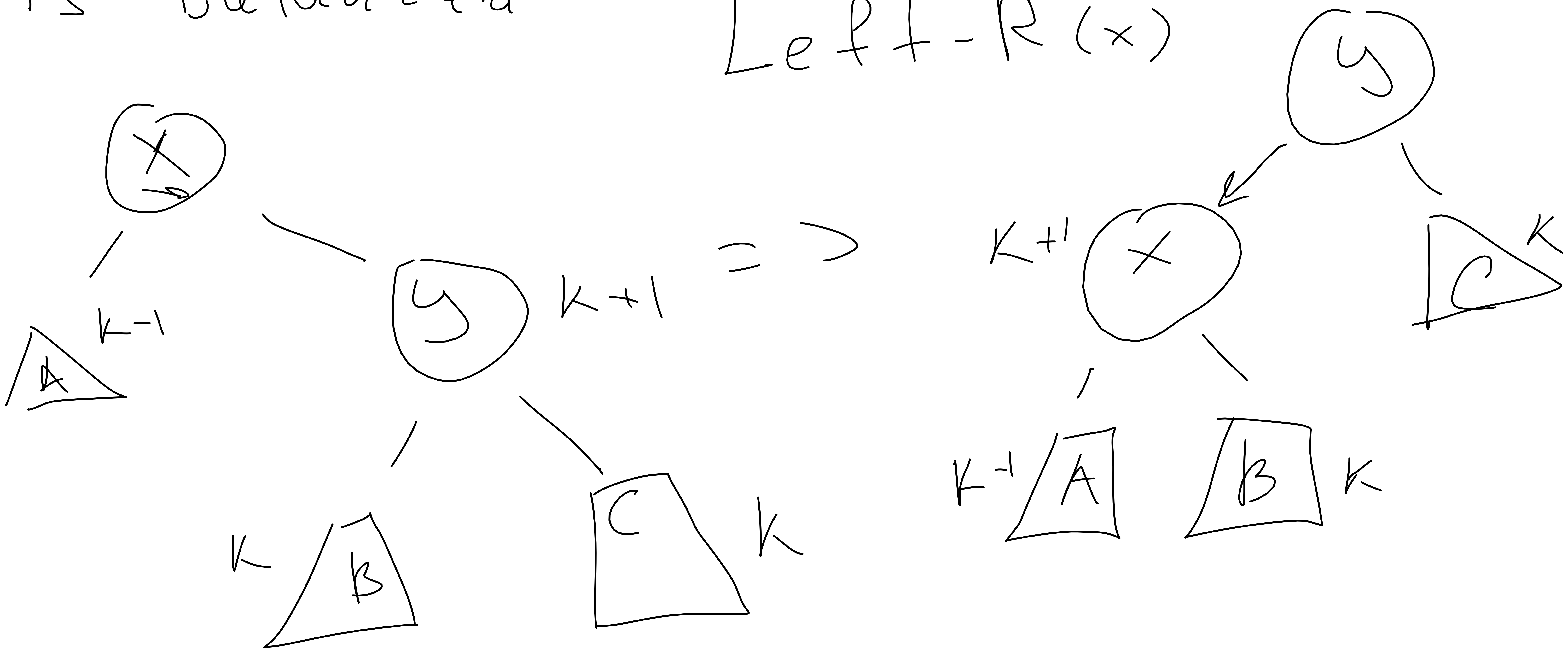
1) y is righ-heavy

Left-Rotation

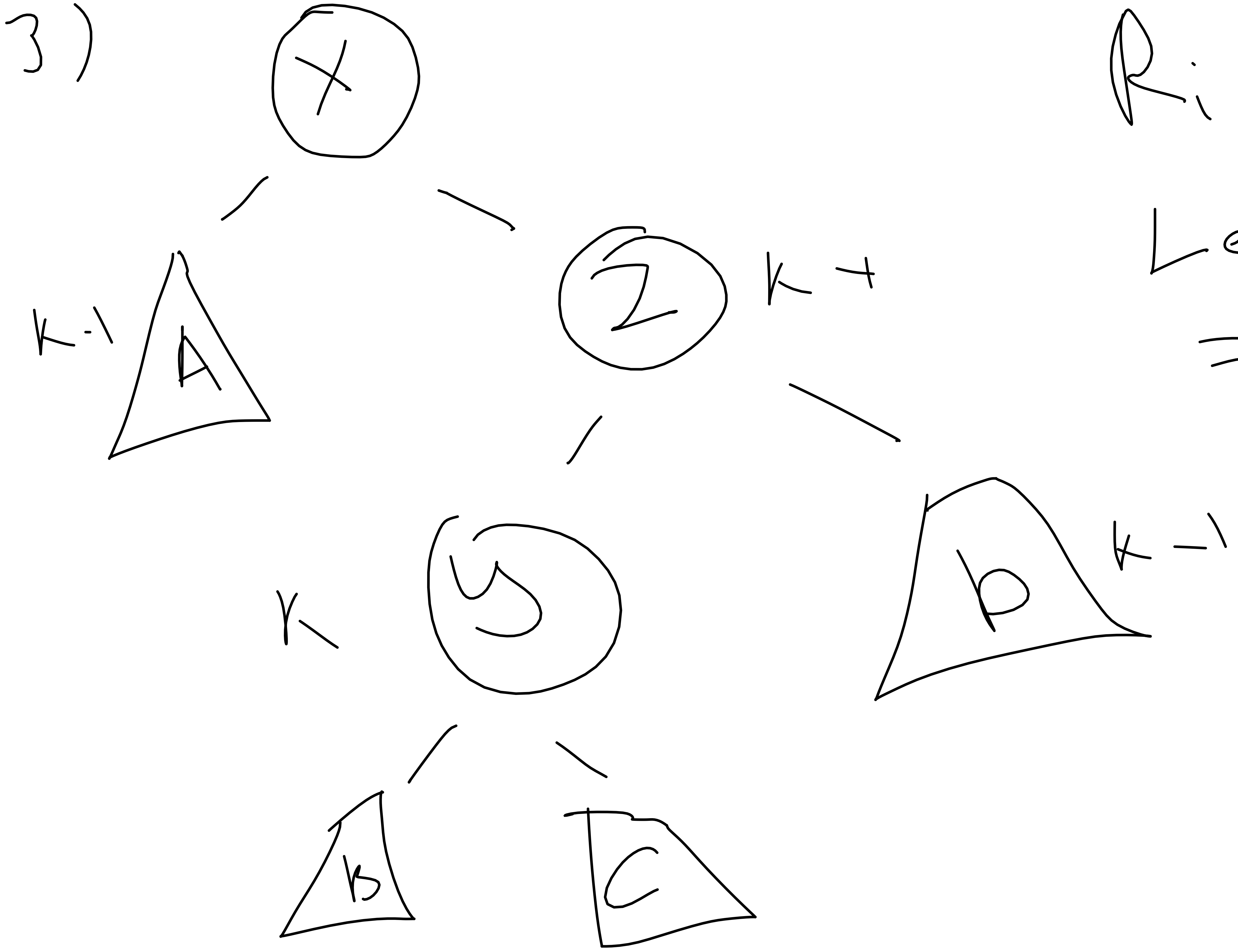


2) y is balanced

Left-R(x)



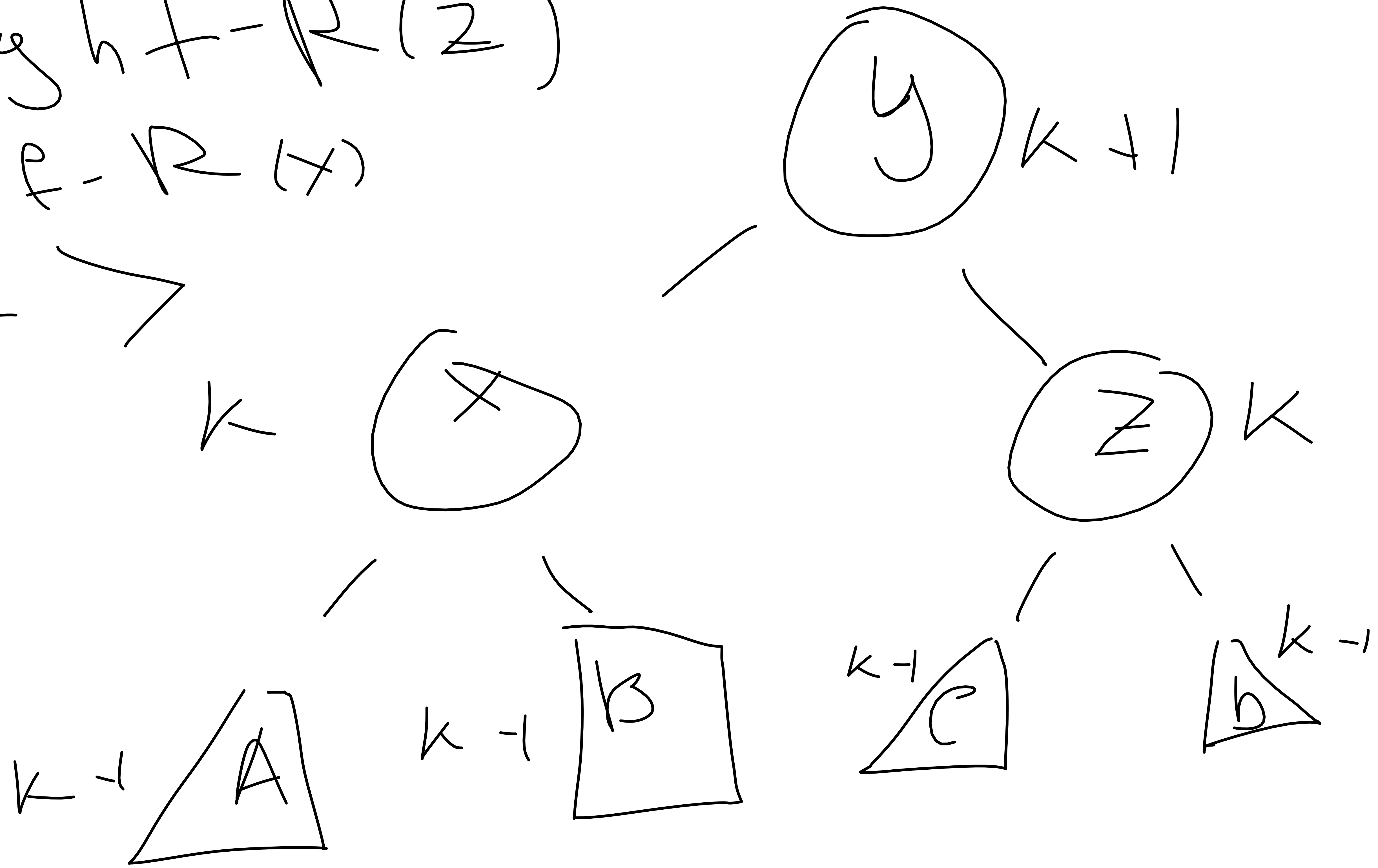
3)



Right-R(z)

Left-R(x)

=>



AVL Tree Sort

Sorting

1) Create AVL on n nodes $O(n \cdot \log n)$

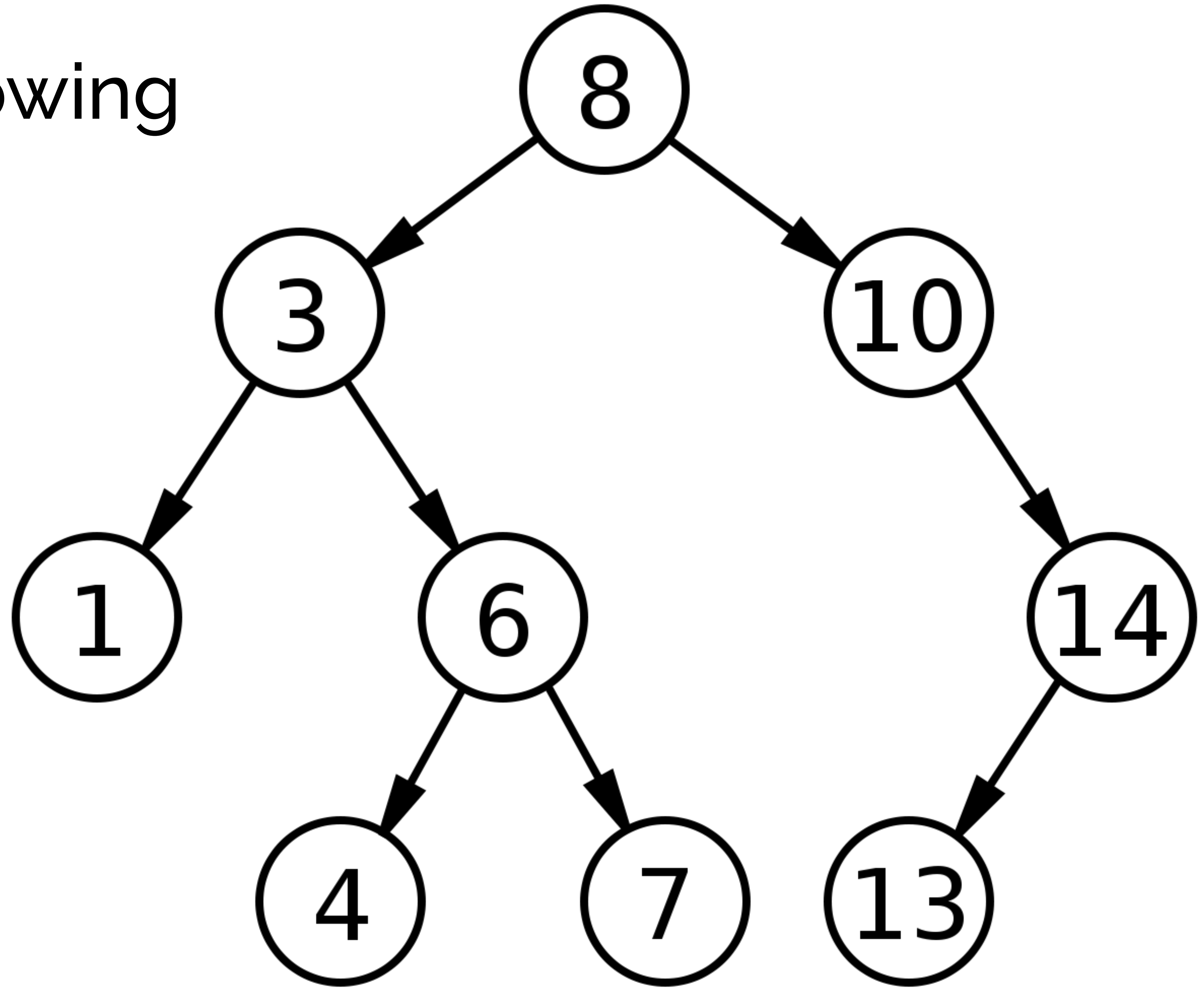
2) In order traversale $O(n)$

Time Complexity: $O(n \cdot \log n)$

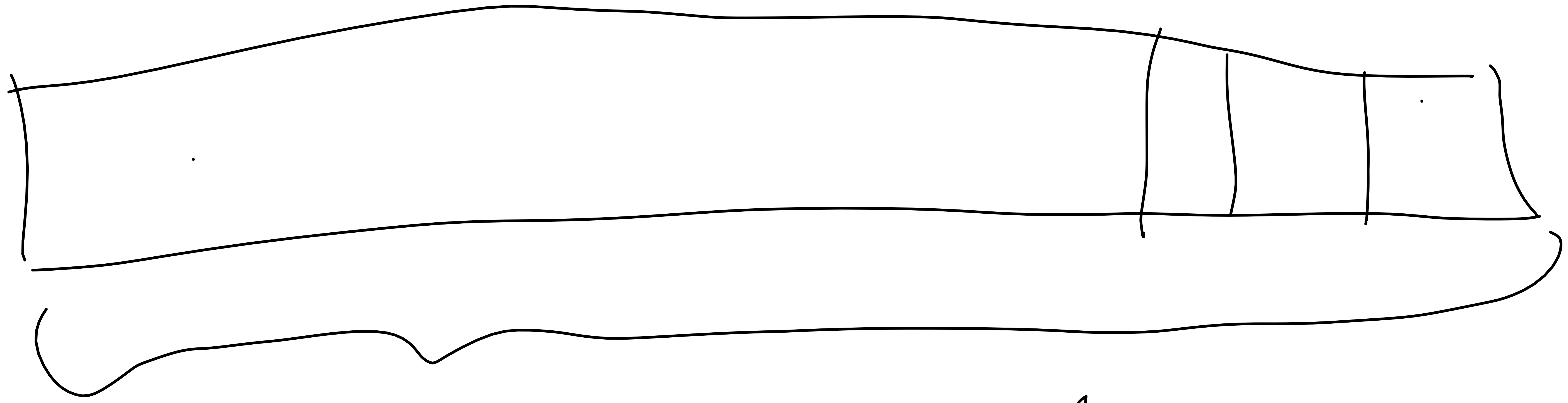
In-order Traversal

- **InOrderTraversal** - visit nodes in the following order: left subtree, right subtree, node
- We get the **sorted** order!

```
func inOrderTraversal(Node x):
   if x != nullptr:
      inOrderTraversal(x.left)
      print x.key
      inOrderTraversal(x.right)
```



Your questions!



n

$n \times (3+n)$

$\boxed{n}$

2 int → $n \times 4$

1 int

pointer

$O(1)$