Algorithms & Data Structures I: Binary Search Tree

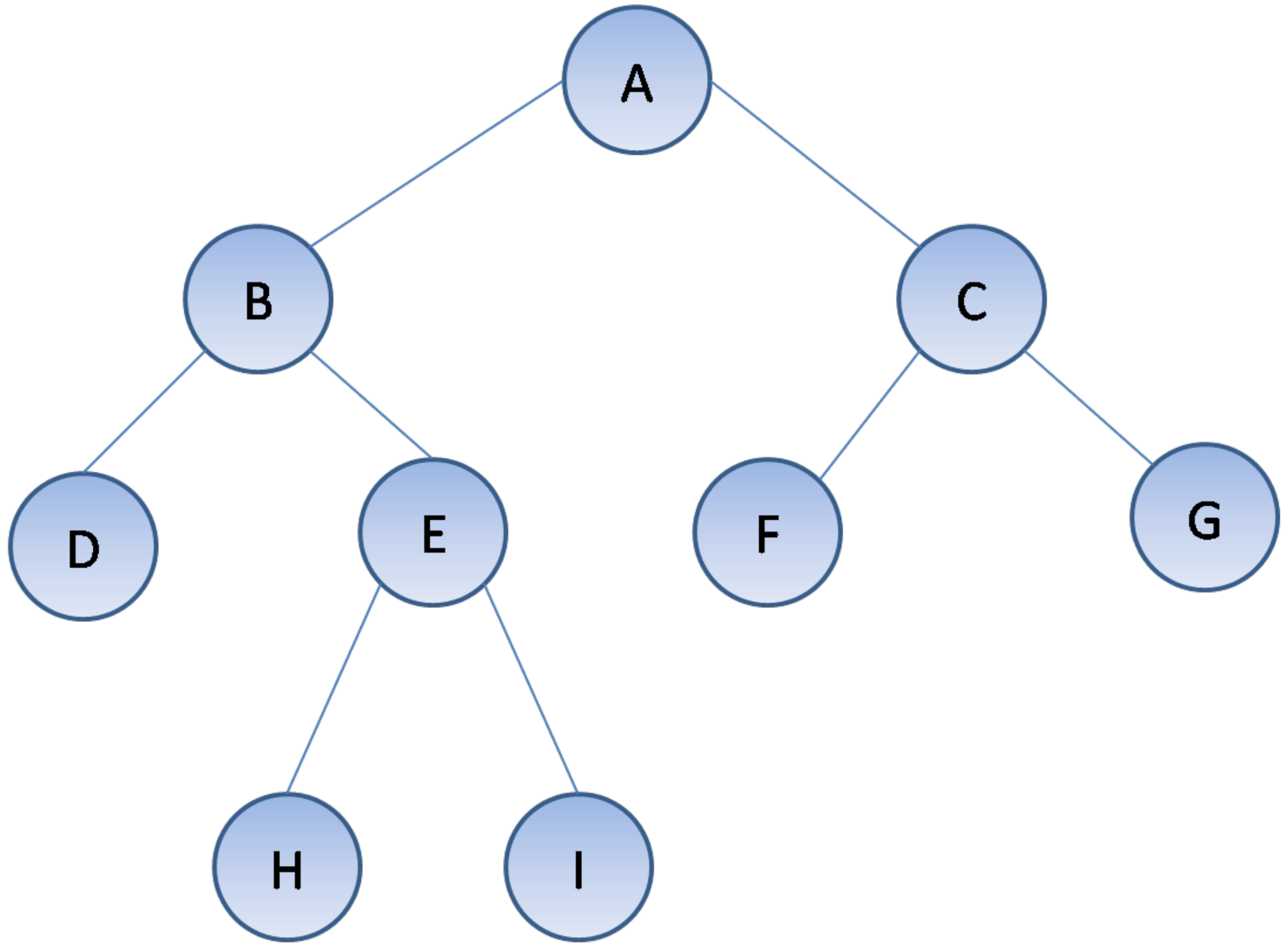
Mikhail Anukhin

Today’s Topics

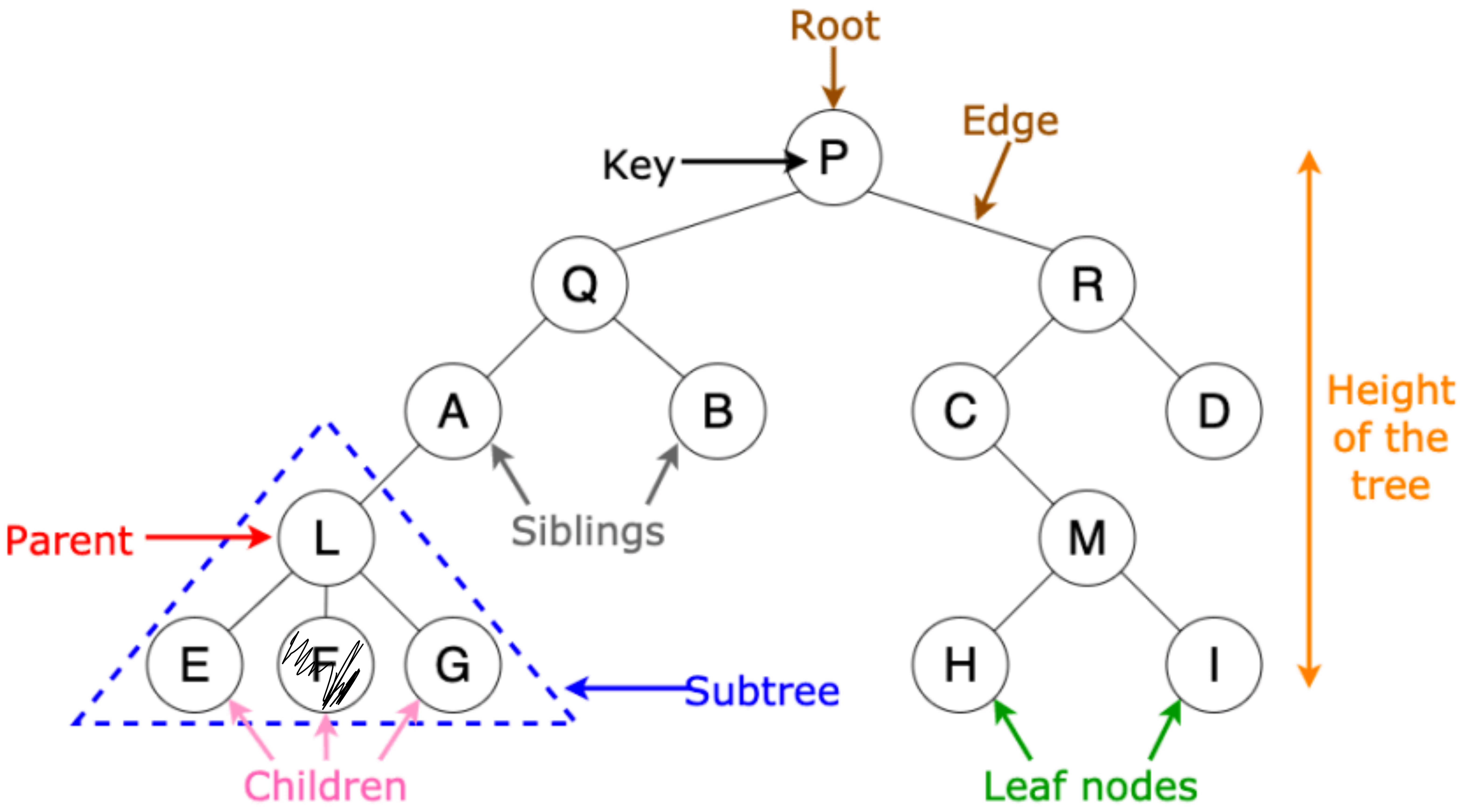
- Applications
- Binary Search Tree definition
- Operations: Insert, Find, Delete, FindMax
- Tree Traversals

Binary Tree Aplications

- Used in almost every **high-bandwidth** router to store router tables
- Used in almost any 3D video game to determine which objects need to be rendered.
- Although most **databases** use a form of B-tree to store data on the drive.
- Constructed by **compilers** and (implicit) **calculators** to parse expressions.



Binary Tree Terms

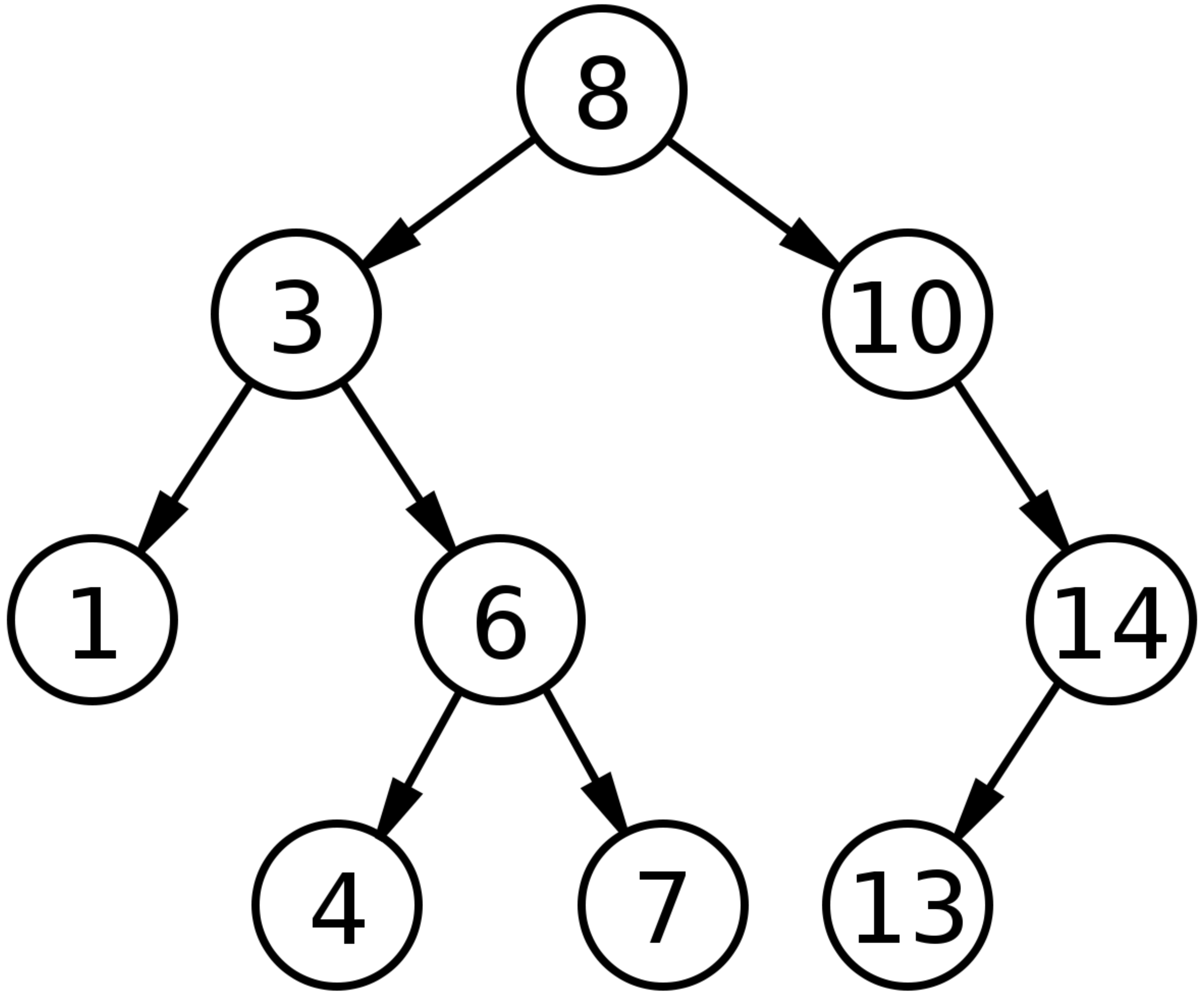


Mikhail Anukhin

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)*

Mikhail Anukhin

Binary Sea~~t~~rch Tree Operations

Find(value): check if there is a node with the value

FindMax(): return the maximum value a tree stores

Insert(value): insert a new node with the value in a tree

Delete(value): delete a node with the value from a tree

Find Idea

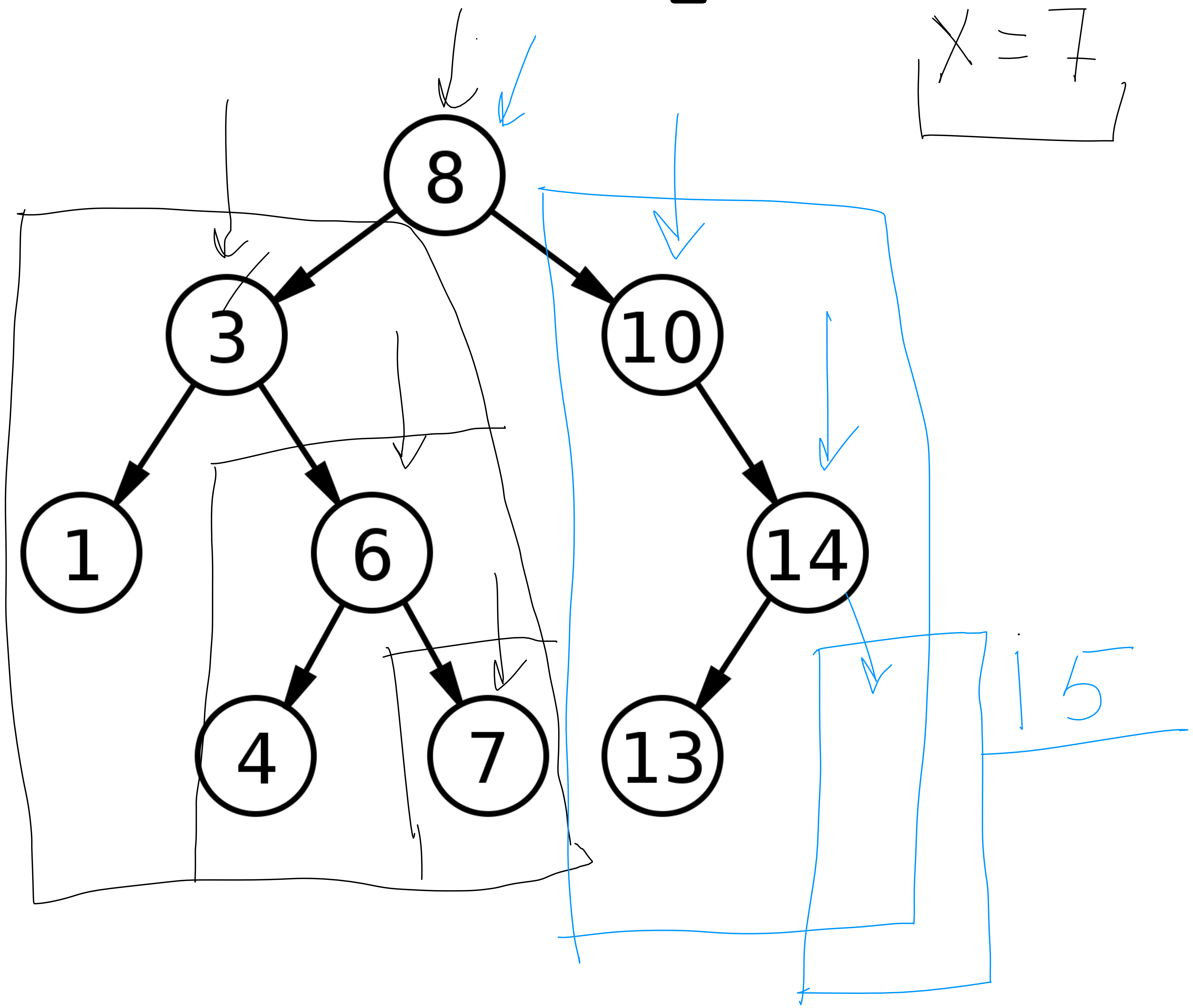
Find(value): check if there is a node with the value

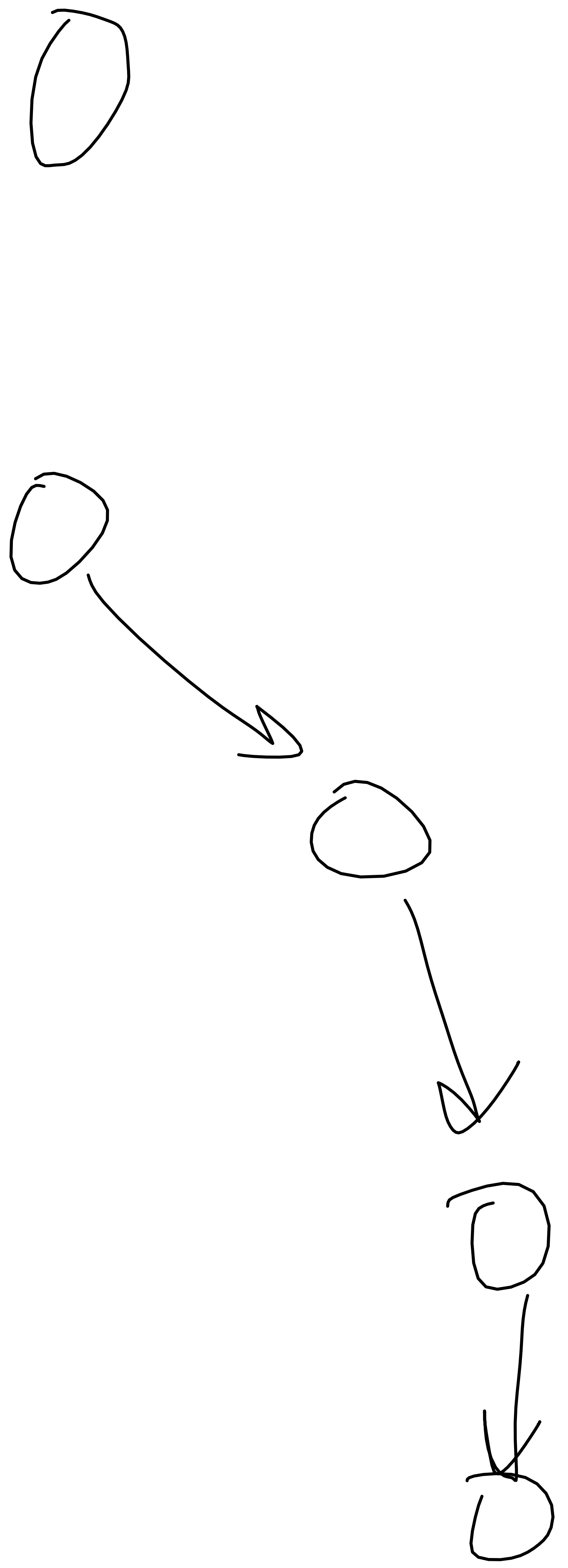
Idea: Follow left and right pointers until you find it or hit false

Find Example

Node {
  Node* left;
  Node* right;
  T data;
}

Tree {
  Node* root
}





Mikhail Anukhin

Find Pseudocode

O(h)

```
Node* search (Node* x, T k):
        if x == null or k == x->data:
            return x;
        if k < x->k:
            return search(x->left, k)
        else:
            return search(x->right, k)
```

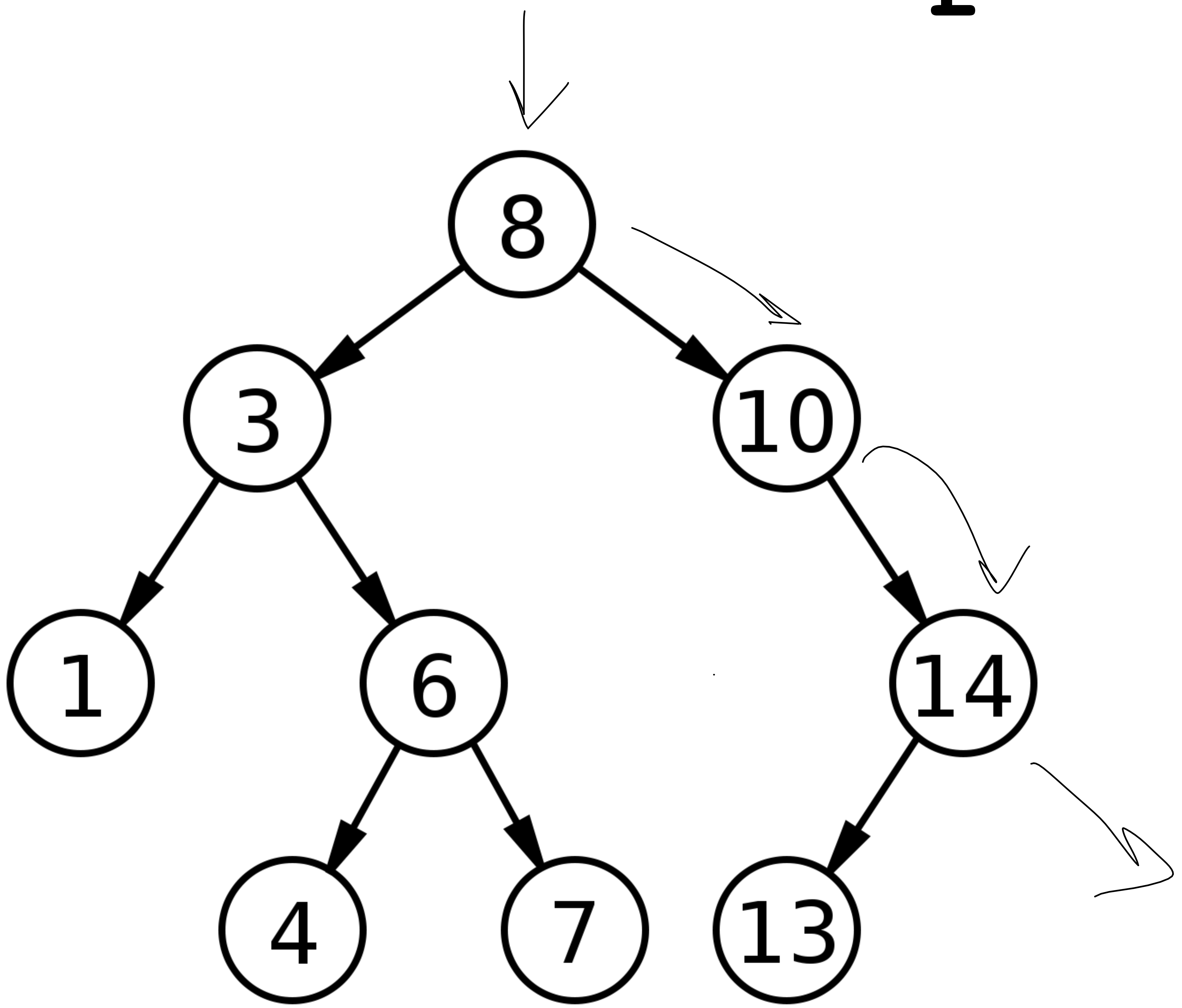
Mikhail Anukhin

FindMax Idea

FindMax: return the maximum value a tree stores

Idea: Go to right pointer until you can

FindMax Example

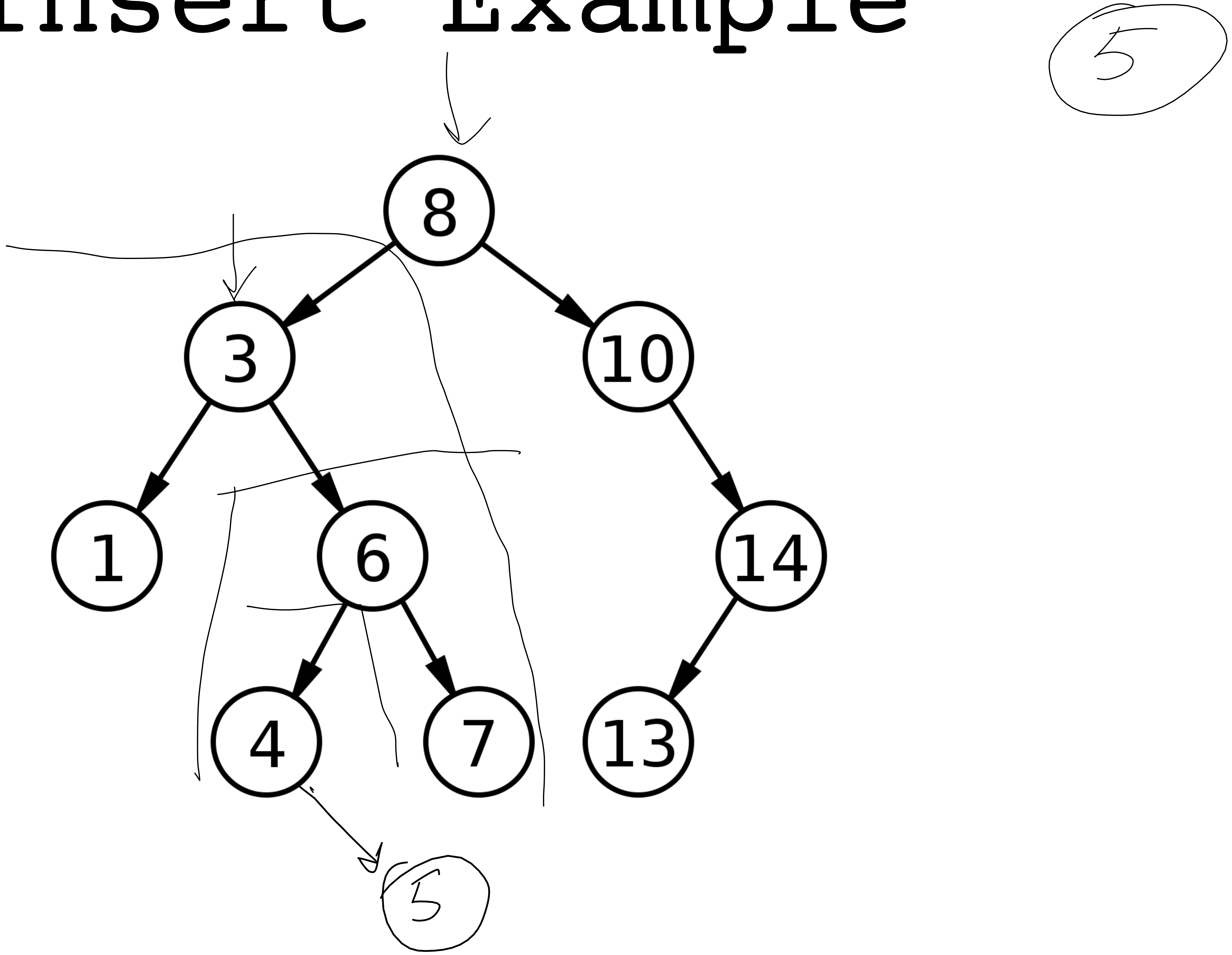


Insert Idea

Insert(value): insert a new node with the value in a tree

Idea: Follow left and right pointers till you find the position

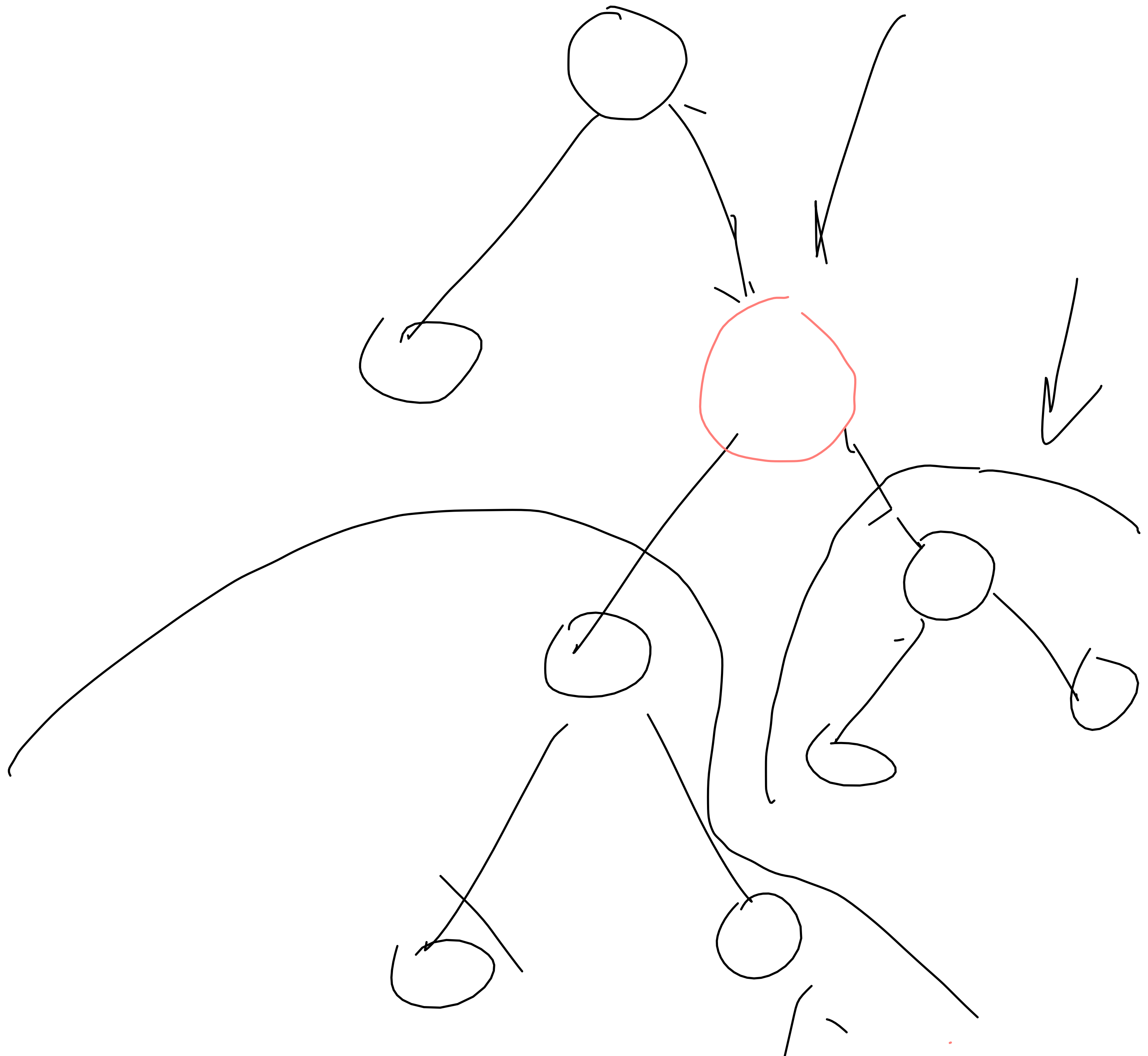
Insert Example



Deletion Idea

Delete(value): delete a node with the value from a tree

Idea: Find a node with the value and swapped with the one such that swap doesn't brake the main invariant.
Delete consist of 4 steps:
1) Find a node to delete
2) Find a node we can swap with. (A node with the next value in sorted order)
3) Swap
4) Discard a node to delete after swap



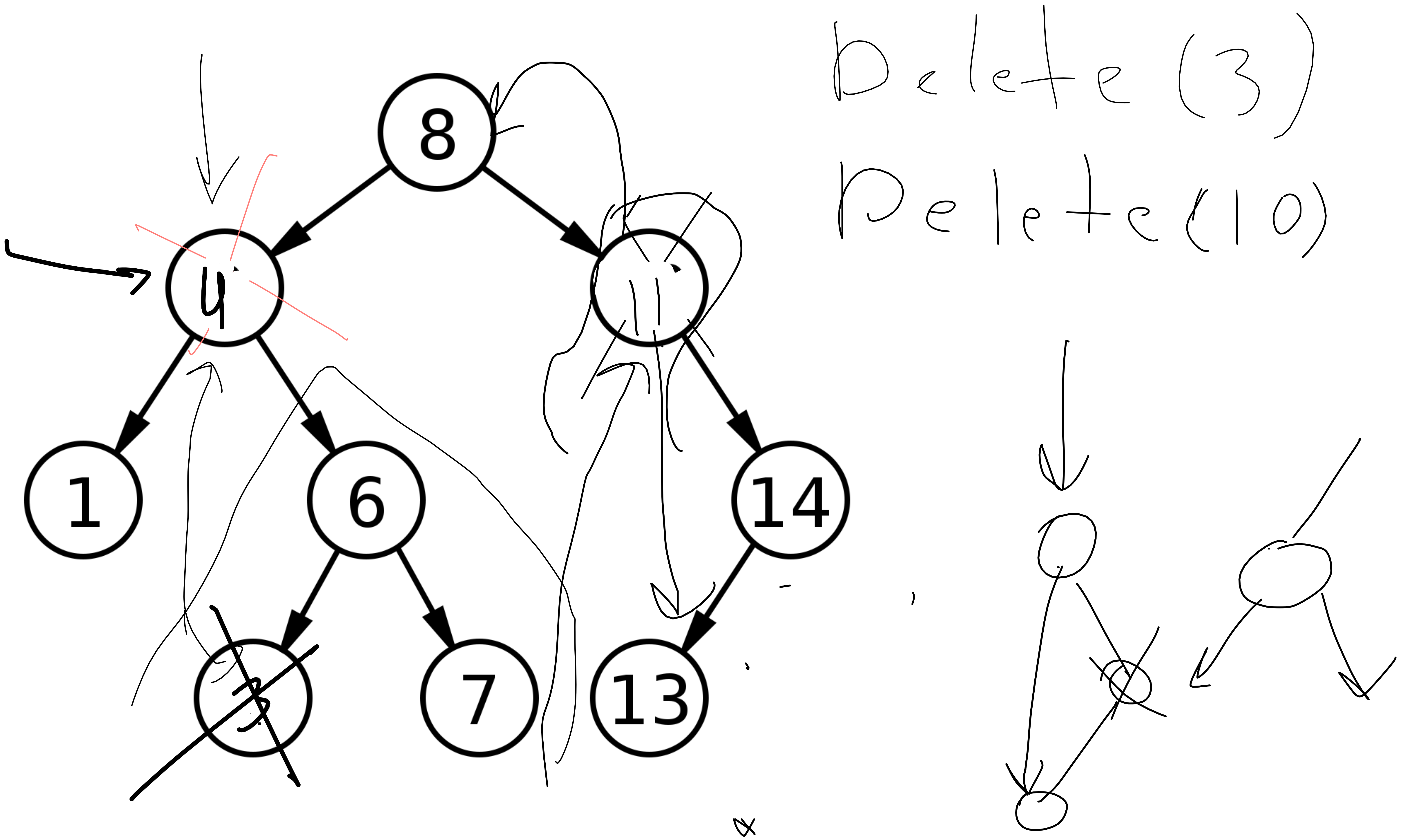
1 max from left subtree

2 min from right subtree

Deletion Example

1) a leaf

2)



Delete(3)

Delete(10)

Tree Traversals

We want to visit all nodes in a tree. How can we do it?

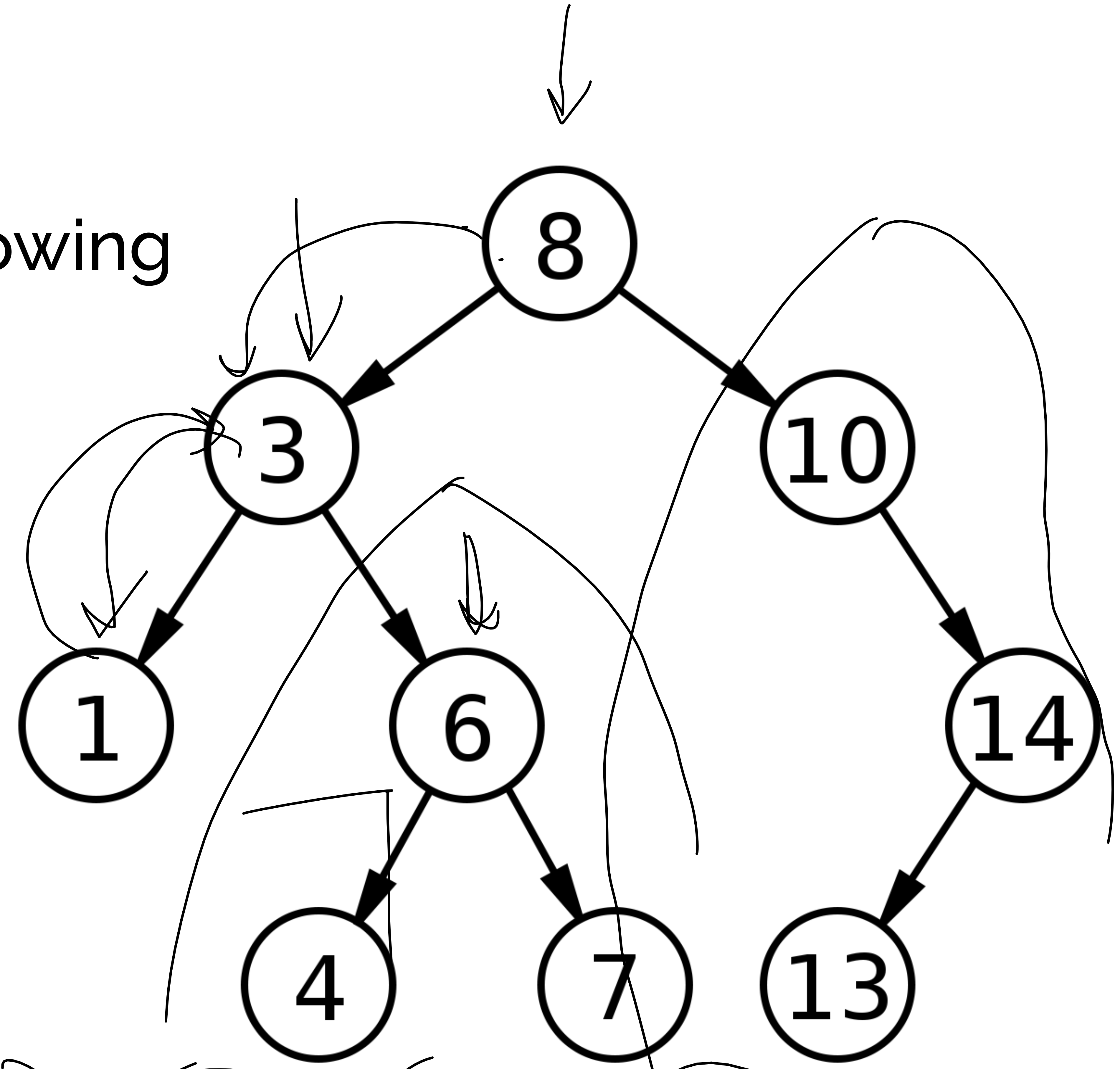
- **InOrderTraversal** - visit nodes in the following order: left subtree, node right subtree, ~~node~~
- **PreOrderTraversal** - visit nodes in the following order: node, left subtree, right subtree
- **PostOrderTraversal** - visit nodes in the following order: left subtree, right subtree, node

Mikhail Anukhin

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)
```

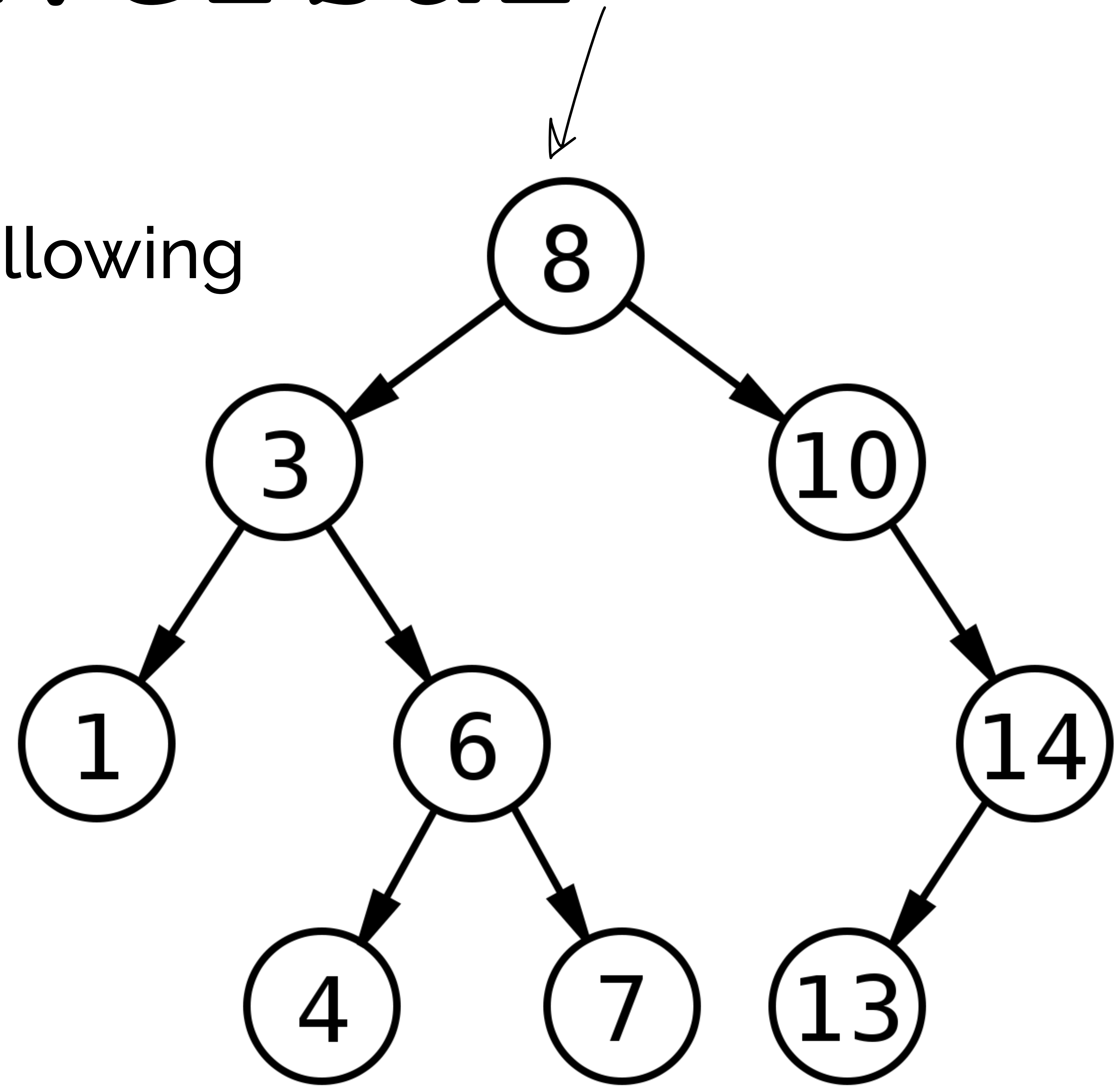


1 3 4 6 7 8 10 13 14

Pre-order Traversal

- **PreOrderTraversal** - visit nodes in the following order: node, left subtree, right subtree

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



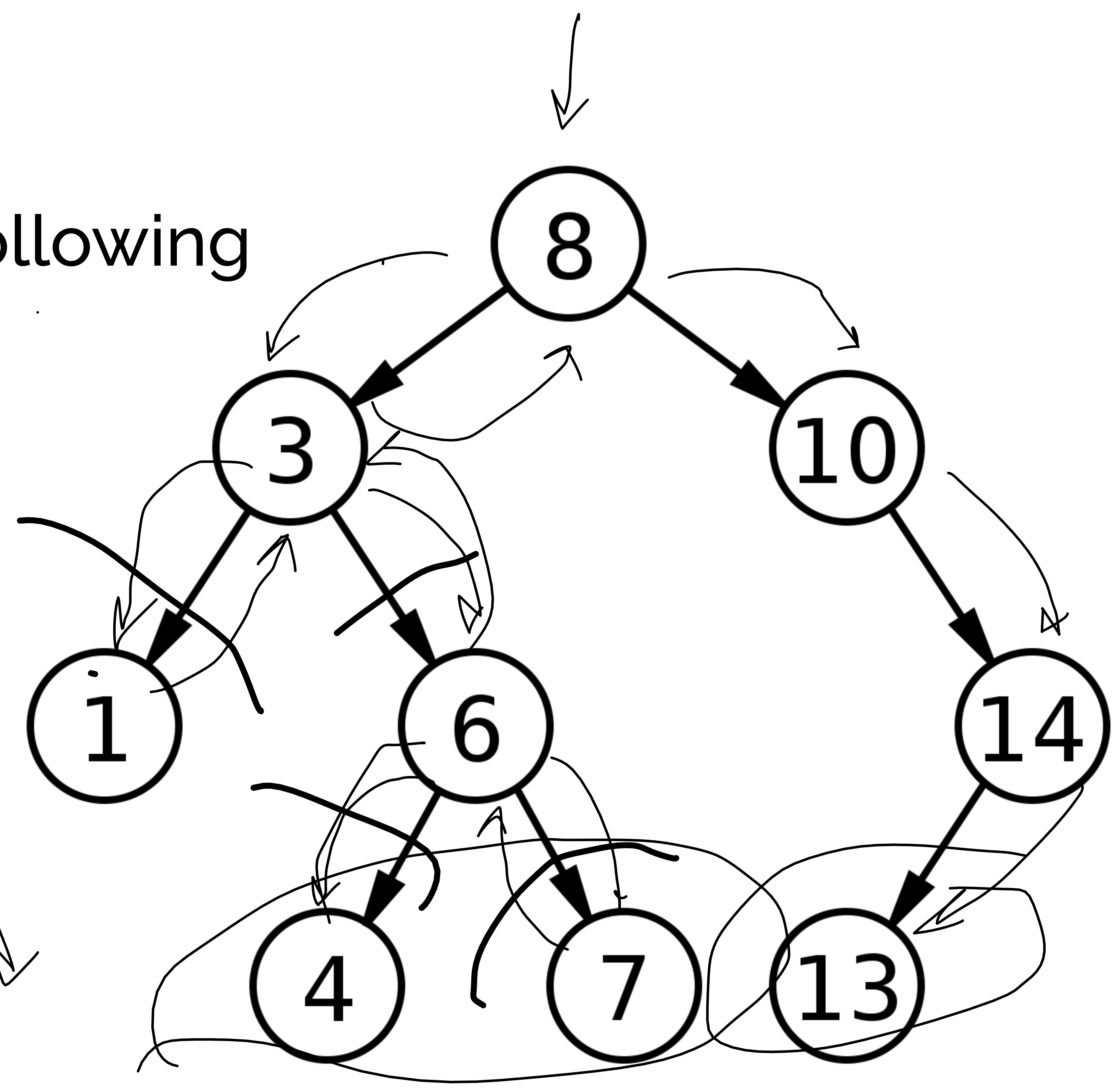
8 3 1 6 4 7 10 14 13

Mikhail Anukhin

Post-order Traversal

- **PostOrderTraversal** - visit nodes in the following order: left subtree, right subtree, node

```
func postorderTraversal(x : Node)
   if x != null
      postorderTraversal(x.left)
      postorderTraversal(x.right)
      print x.key
```

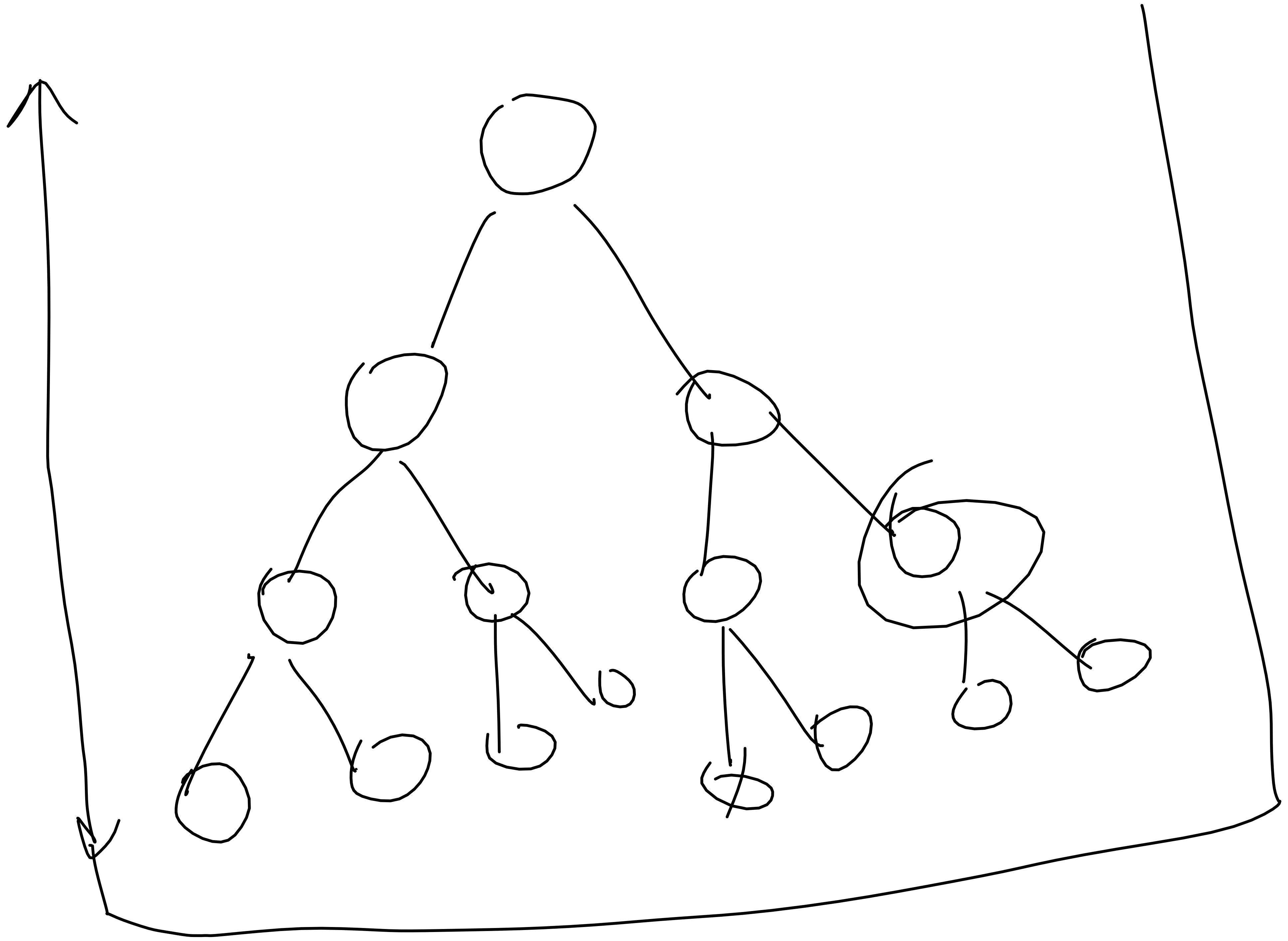


1 4 7 6 3 13 14 10 8

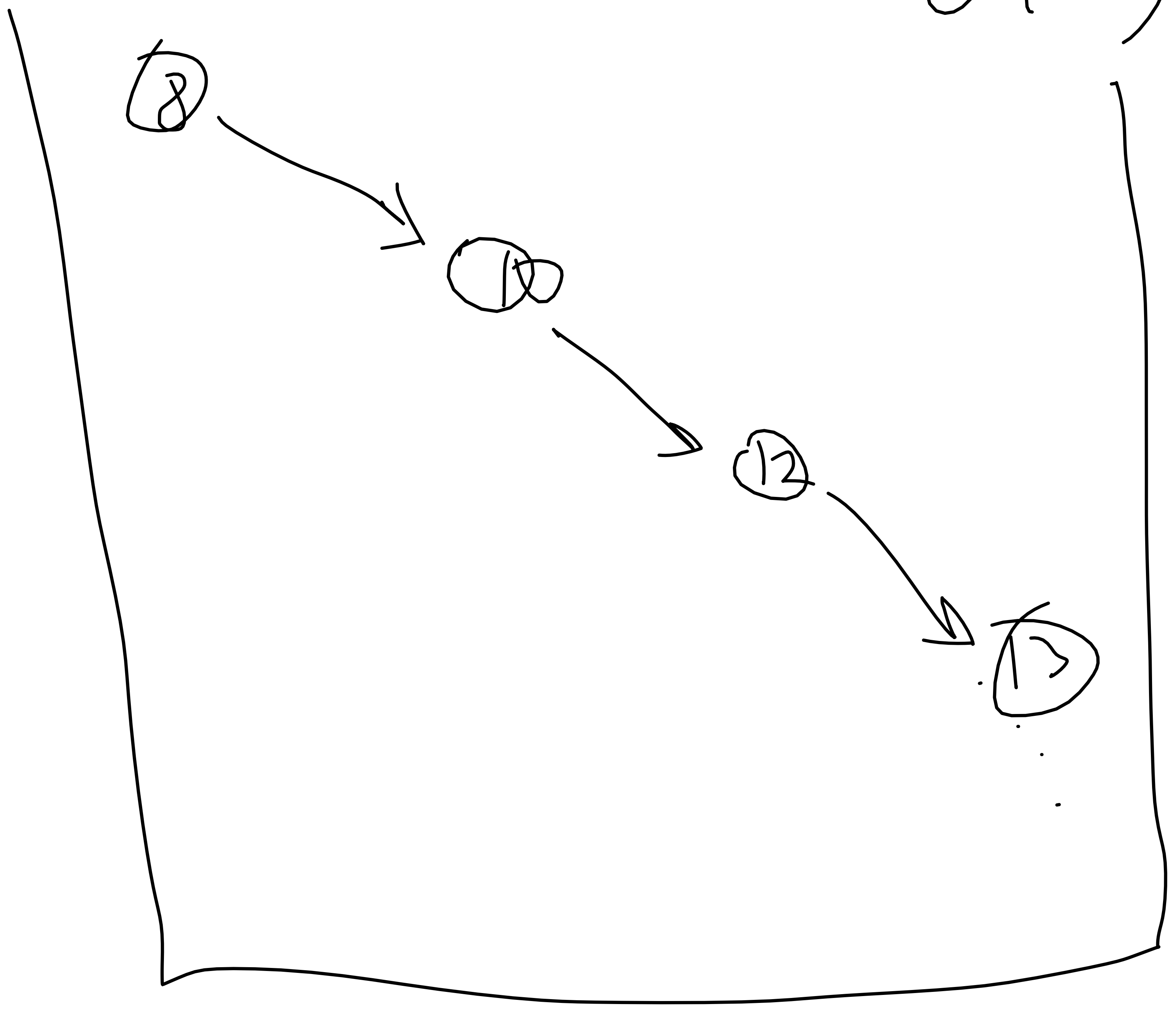
Height of a binary search tree

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

$h \sim O(\log n)$



$h \sim O(n)$



8 10 12 13

Height of a binary search tree

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

Summary

Insertion, Find and Deletion operation is quicker in trees when compared to arrays and linked lists. Of course, if a height is O(log n)

Balanced BSTs to the rescue in the next lecture!

Insert(value): O(h)

Find(value): O(h)

Delete(value): O(h)

FindMax(): O(h)

Your questions!