Data Structures

Binary Search Trees 2

CS 225
Brad Solomon

September 23, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Exam 2 (10/02 — 10/04)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be released on PL

Topics covered can be found on website

Registration started September 19

https://courses.engr.illinois.edu/cs225/fa2024/exams/

Learning Objectives

Build conceptual and coding understanding of BST

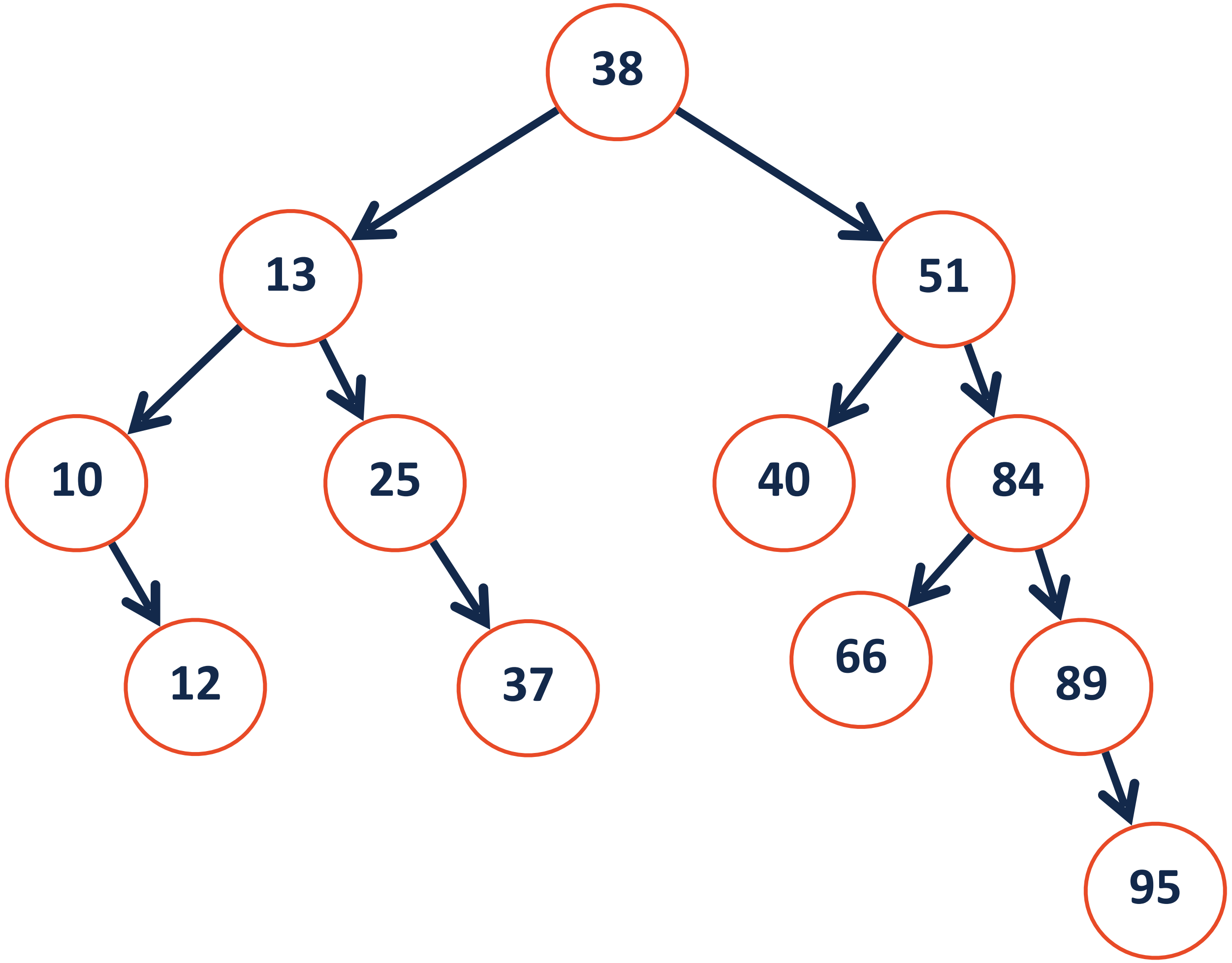
Discuss pros and cons of BST (and possible improvements)

Binary Search Tree (BST)

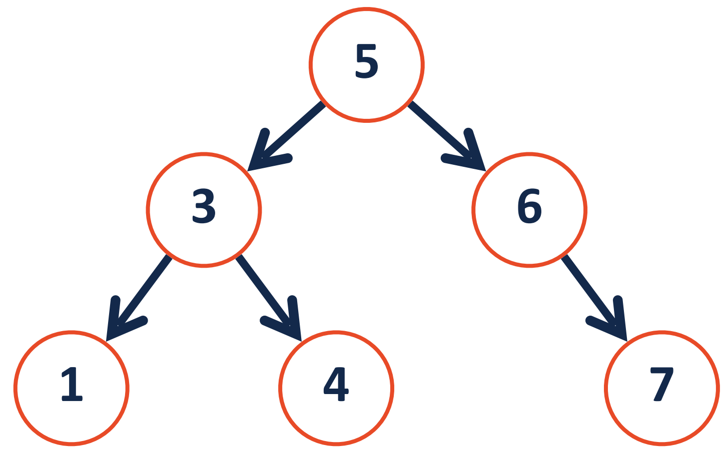
A **BST** is a binary tree $T = TreeNode(val, T_L, T_r)$ such that:

$\forall n \in T_L, n.val < T.val$

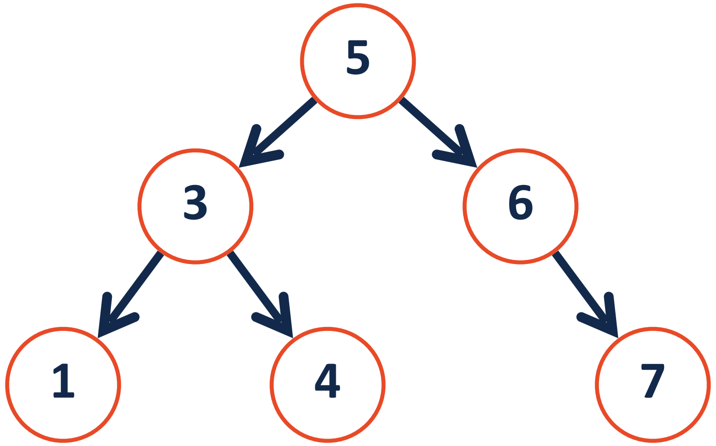
$\forall n \in T_R, n.val > T.val$



```
template<typename K, typename V>
TreeNode *& _find(TreeNode *& root, const K & key) {

// Base Case
if(root == nullptr || root->key == key){
     return root;
}

// Recursive Step ("Combining step" is 'return')
if (root->key > key){
     return _find(root->left, key);
}

return _find(root->right, key);

}
```

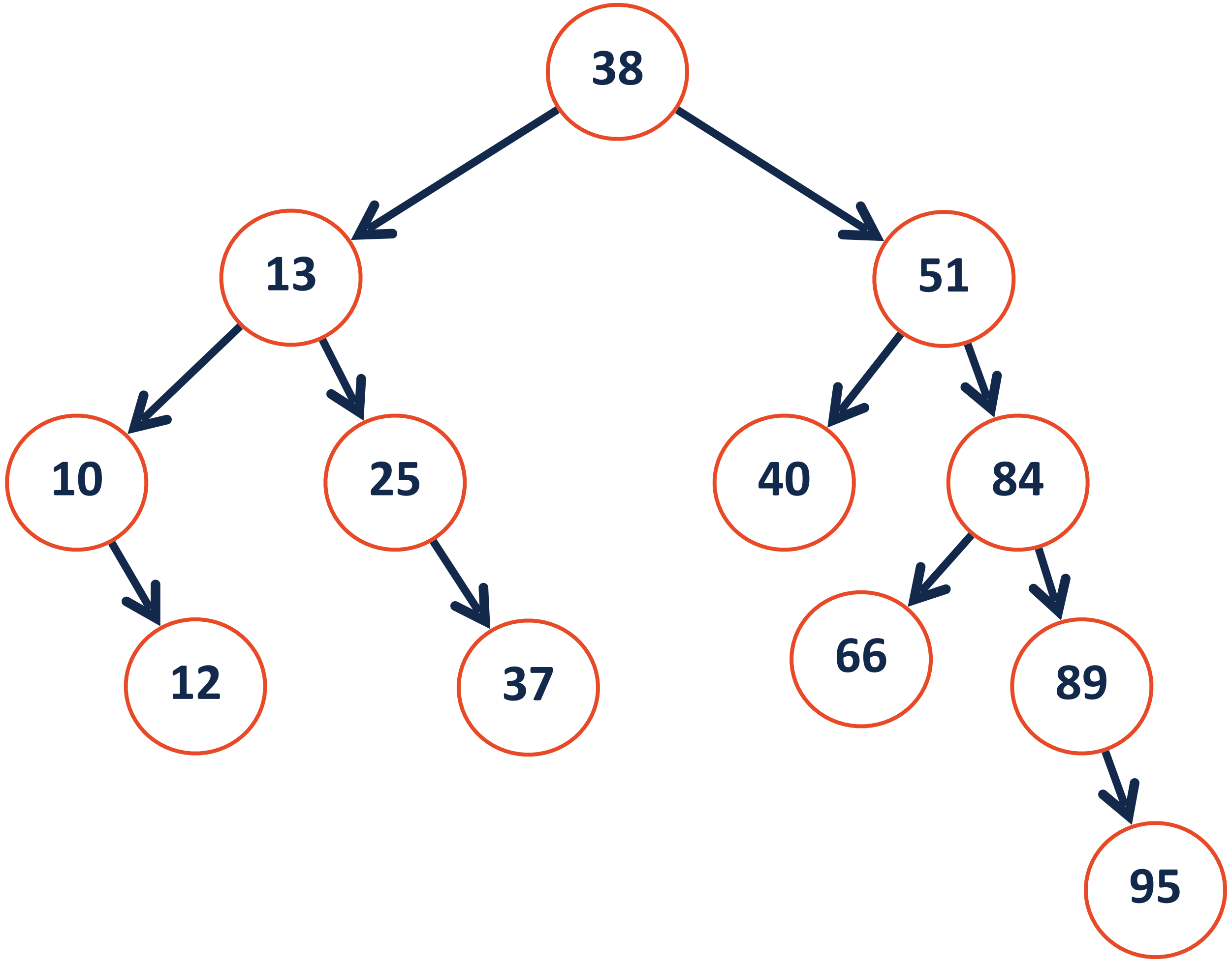


```
template<typename K, typename V>

void _insert(const K & key, const V & val) {


  TreeNode *& tmp = _find(root, key);


  tmp = new treeNode(key, val);


}
```



BST Remove

`remove(40)`

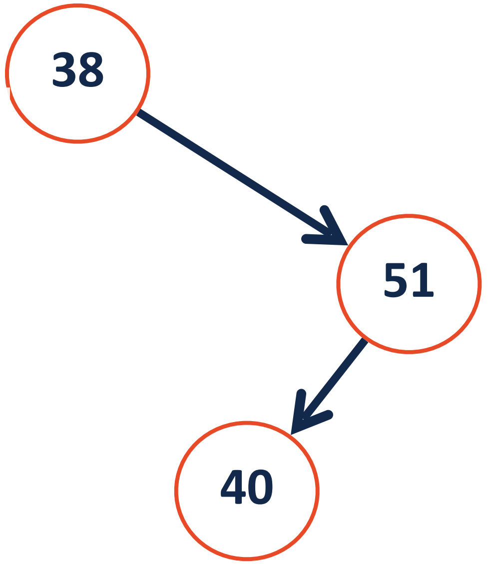


BST Remove

remove(40)

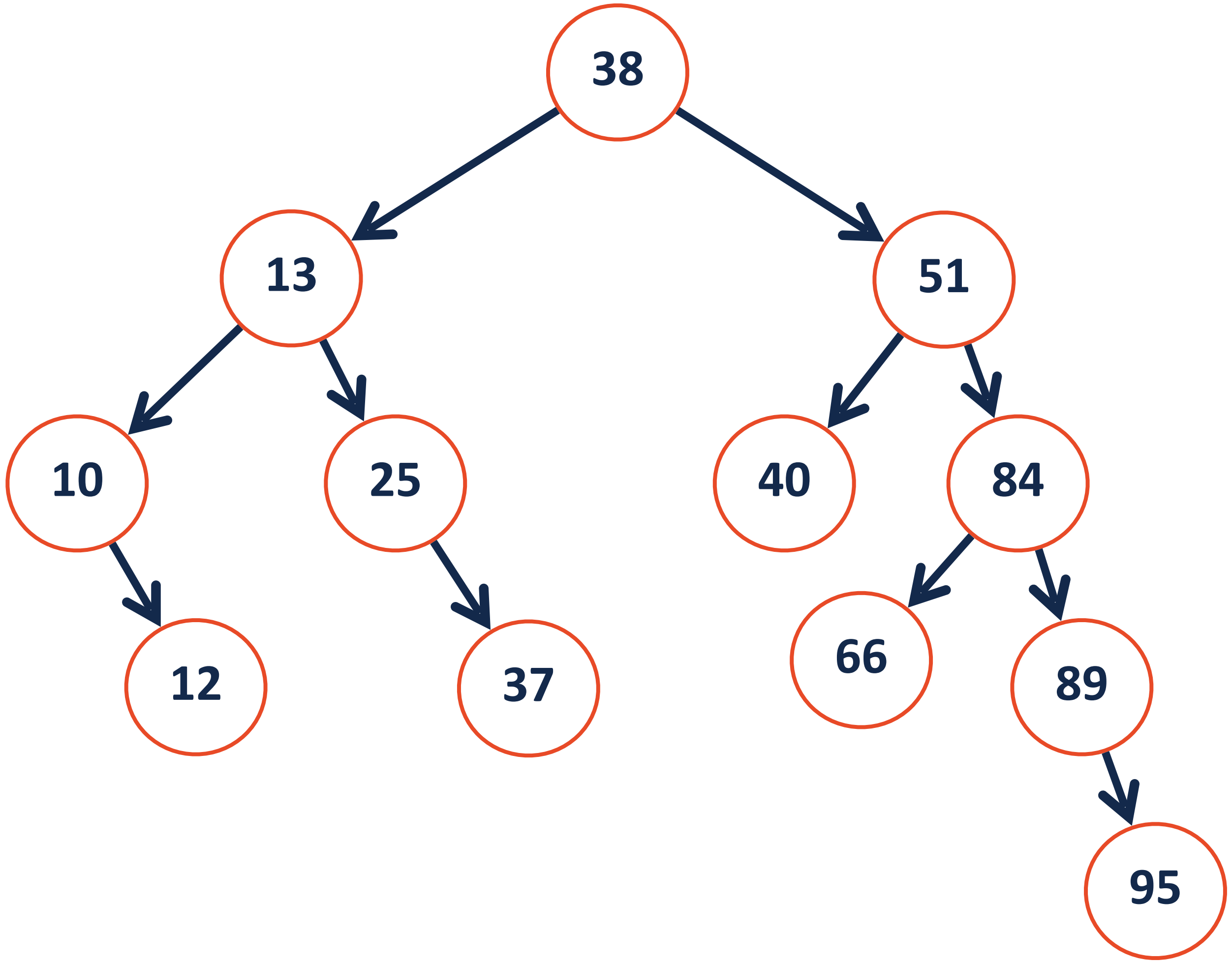
0-Child Case

```
TreeNode *& t = _find(root, 40);
delete t;
t = nullptr;
```



BST Remove

`remove(25)`

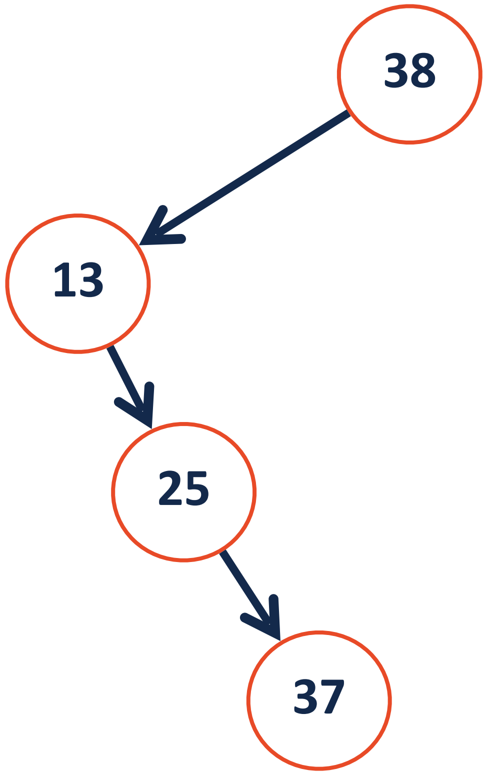


BST Remove

`remove(25)`

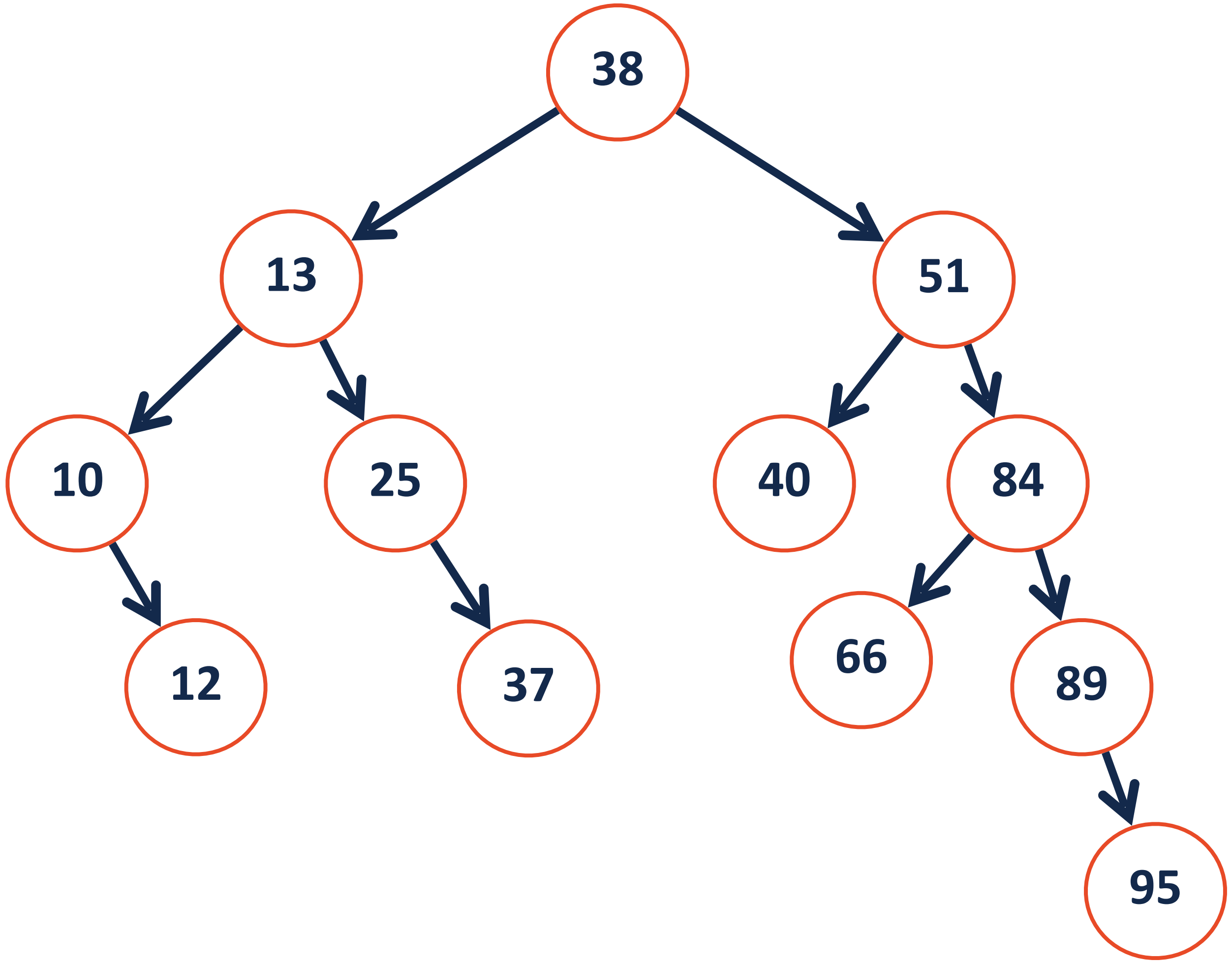
1-Child Case

```
TreeNode *& t = _find(root, 25);
TreeNode * tmp = t;
t = t->right;
delete tmp;
```



BST Remove

`remove(13)`



BST In-Order ________

In-Order Predecessor

Rightmost left child

IOP(38) =

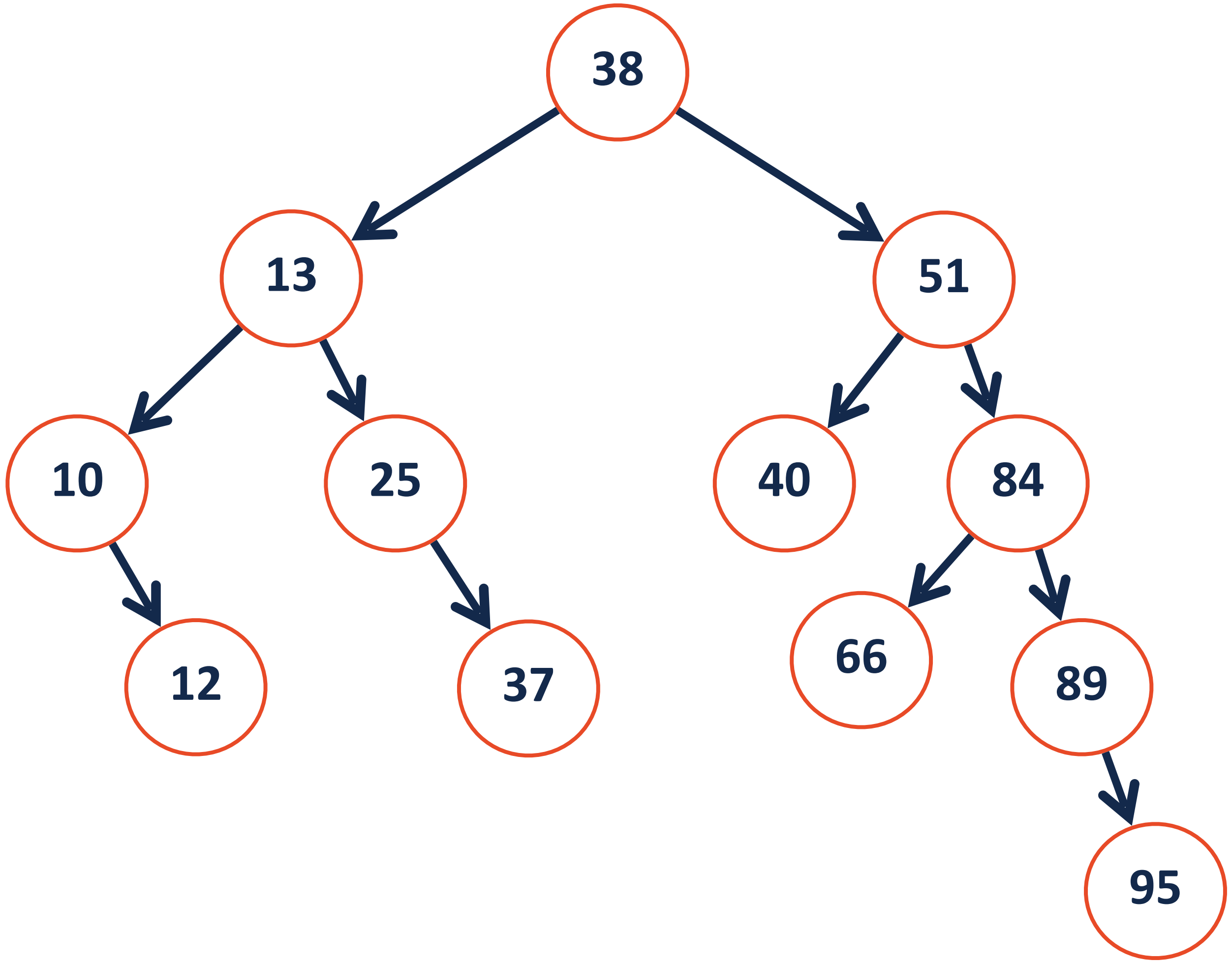
IOP(84) =

In-Order Successor

Leftmost right child

IOS(38) =

IOS(84) =

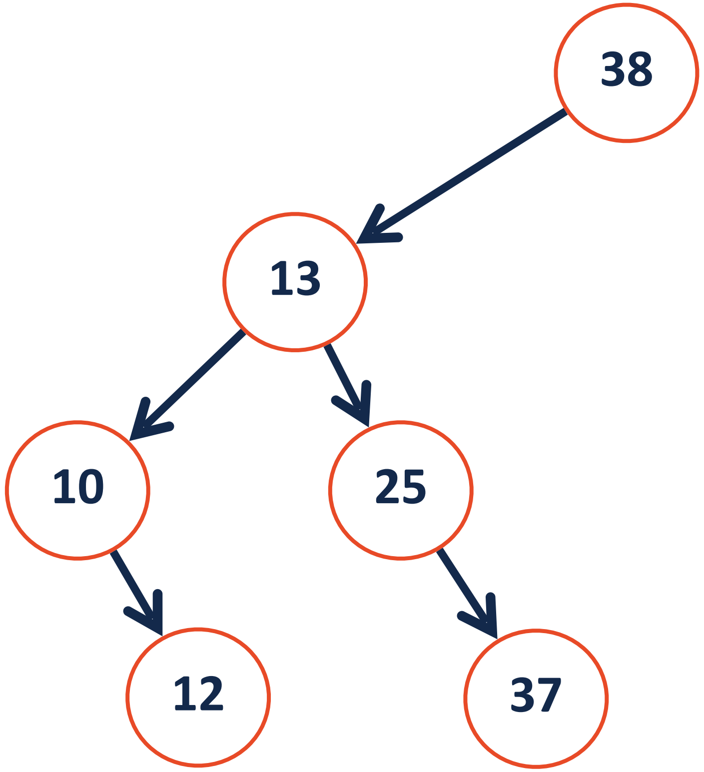


BST Remove

remove(13)

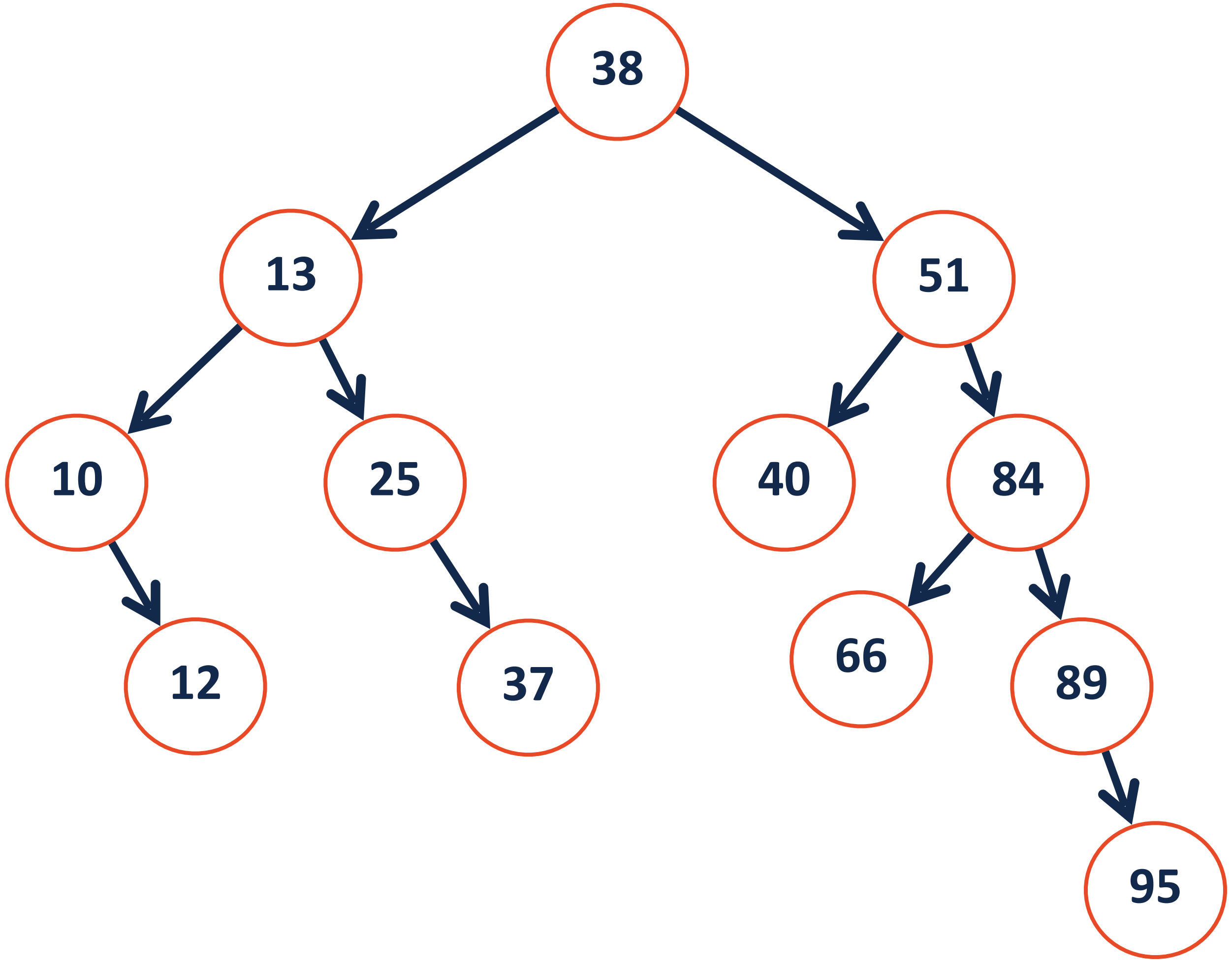
2-Child Case

```
TreeNode *& t = _find(root, 13);
TreeNode * IOP = getIOP(t);
swap(t, iop);
remove(13); //starting from t
```



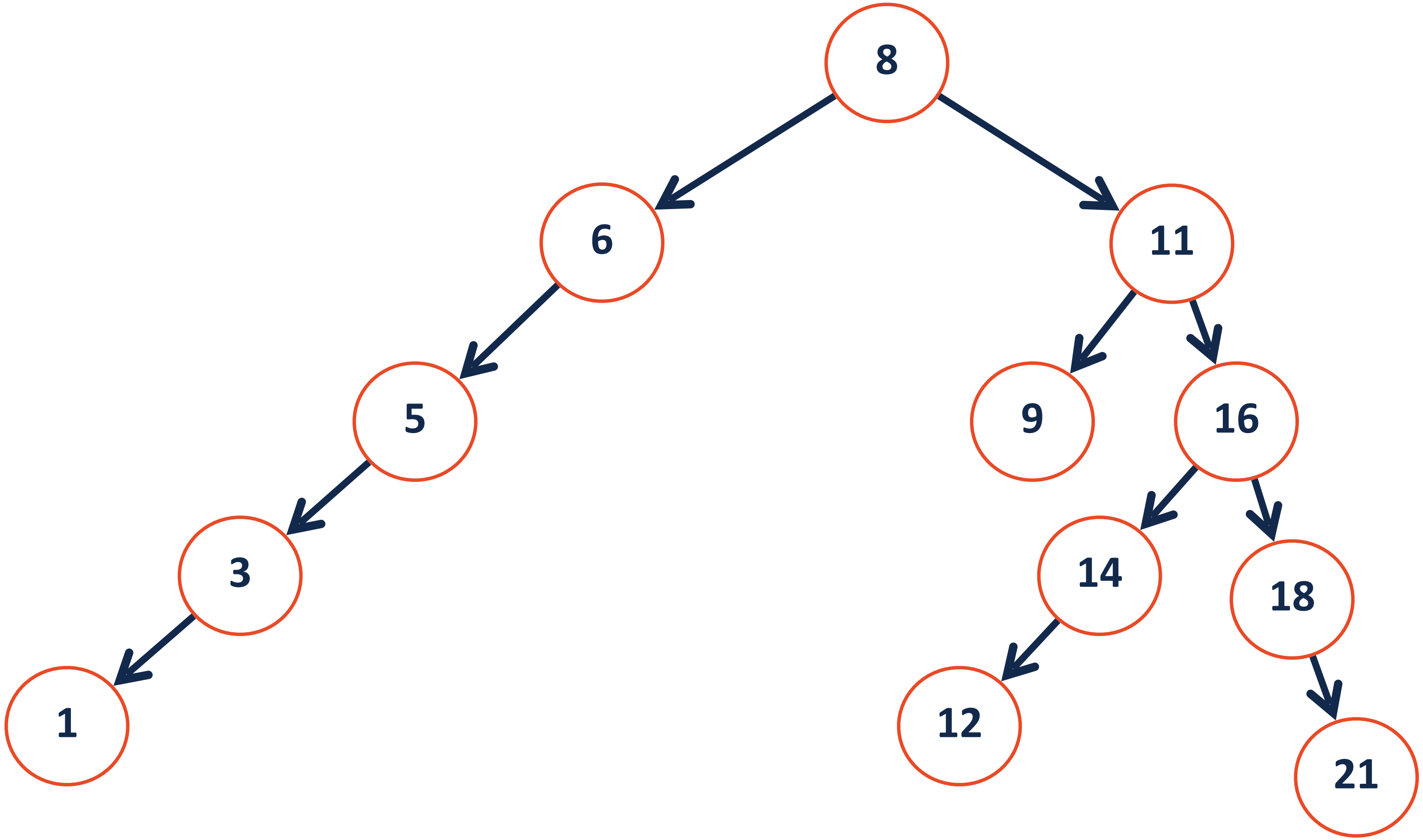
BST Remove

`remove(51)`

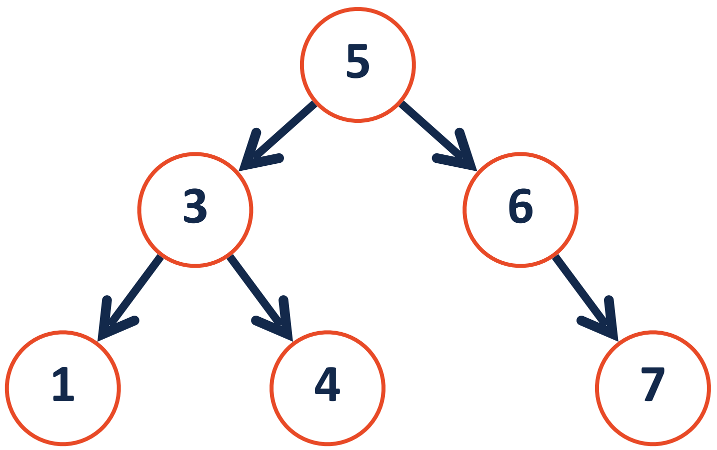


BST Remove

What will the tree structure look like if we remove node 16 using IOS?



```
template<typename K, typename V>

void _remove(TreeNode *& root, const K & key) {
















}
```



BST Analysis – Running Time

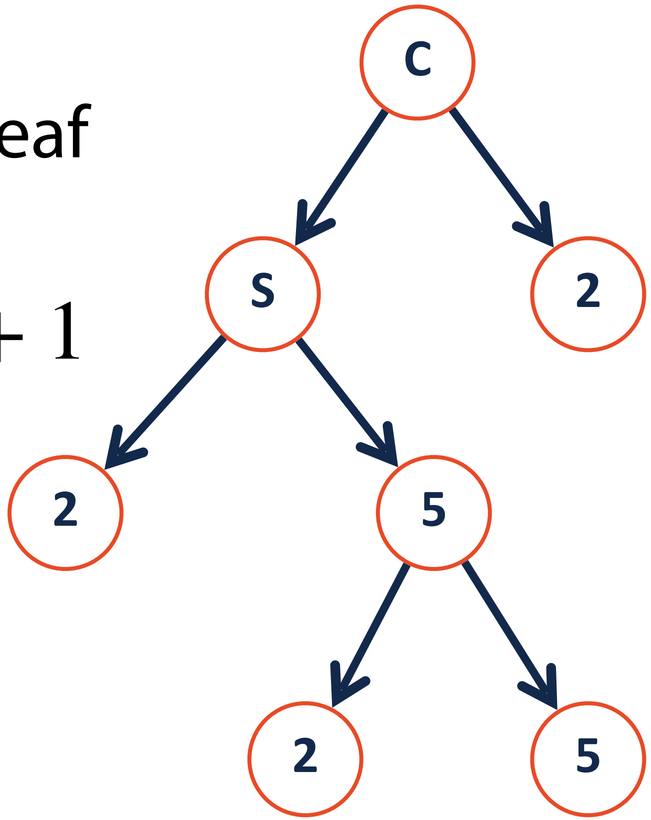
Operation	BST Worst Case
find	
insert	
remove	
traverse	

Binary Tree Height

Height: The length of the longest path from root to leaf

$$Height(root) = max\left(\ Height(T_L), Height(T_R)\right) + 1$$

Given this recursion, what is base case?



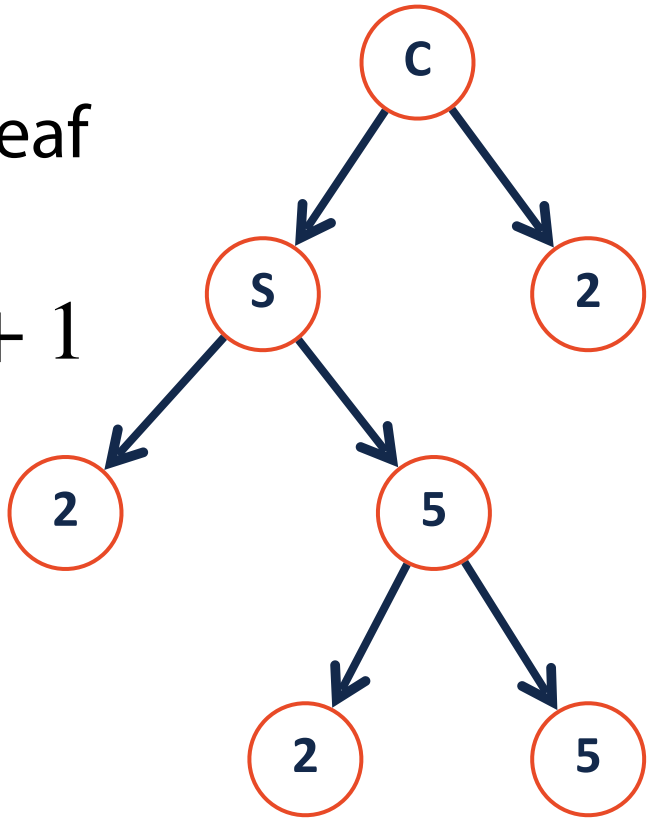
Binary Tree Height

Height: The length of the longest path from root to leaf

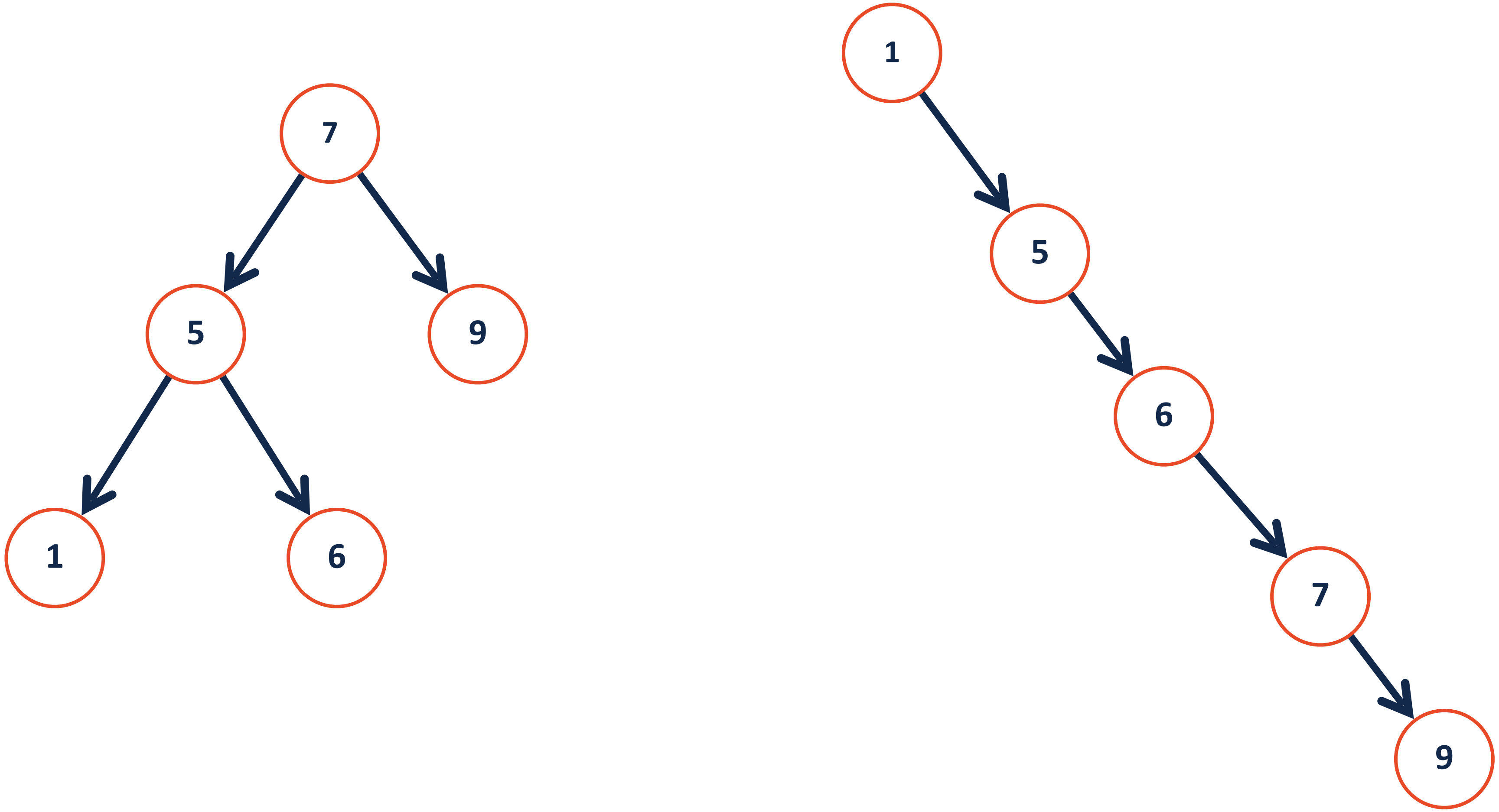
$$Height(root) = max\left(\ Height(T_L), Height(T_R)\right) + 1$$

Given this recursion, what is base case?

$$Height(\varnothing) = -1$$



Limiting the height of a tree



Option A: Correcting bad insert order

The height of a BST depends on the order in which the data was inserted

Insert Order: [1, 3, 2, 4, 5, 6, 7]

Insert Order: [4, 2, 3, 6, 7, 1, 5]

AVL-Tree: A self-balancing binary search tree

Rather than fixing an insertion order, just correct the tree as needed!