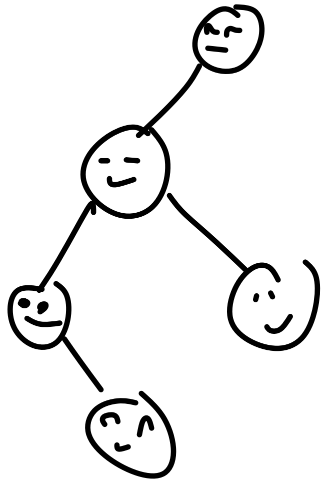
Algorithms and Data Structures for Data Science
Binary Search Tree

CS 277
Brad Solomon

March 4, 2024



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

why is this useful?

Review understanding of Binary Trees

Huffman Tree

Introduce the dictionary ADT

Extend ADT to Binary Search Trees

Practice recursion in the context of trees

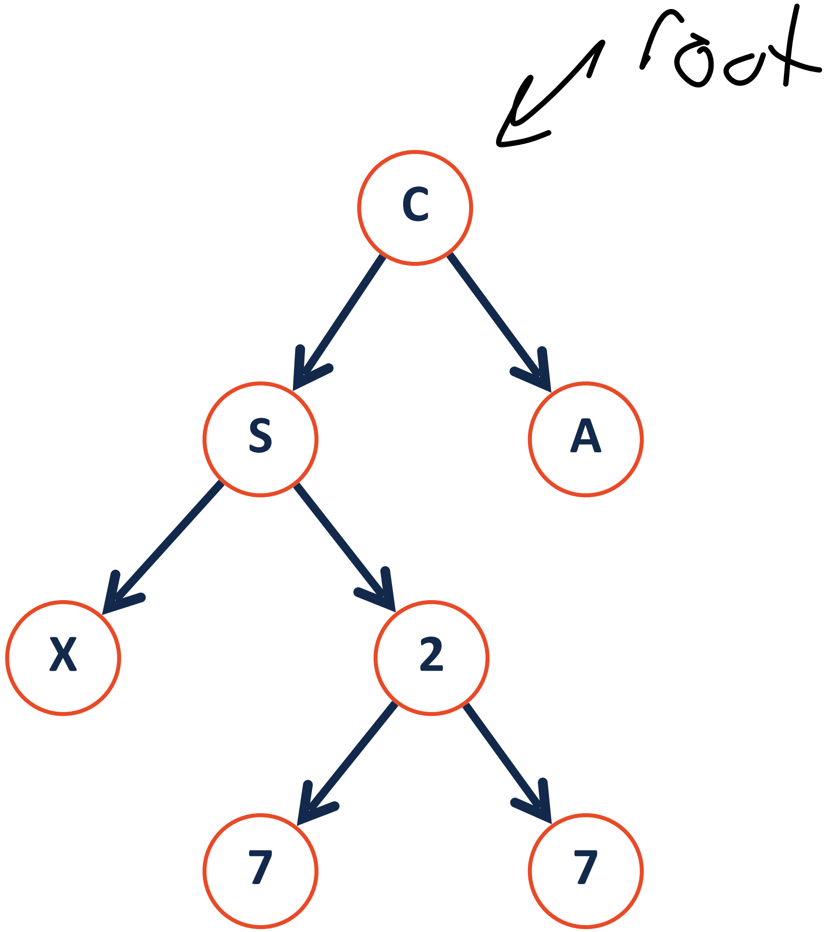
Binary Tree Recursion

A **binary tree** is a tree T such that:

$$T = None$$

or

$$T = treeNode(val, T_L, T_R)$$




```
class treeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

```
class binaryTree:
    def __init__(self):
        self.root = None
```

Tree ADT

Constructor: Build a new (empty) tree

Insert: Add an object into tree

Remove: Remove a specific object from tree

Traverse: Visit every node in tree (all objects)

Search: Find a specific object in the tree

Binary Tree Traversal

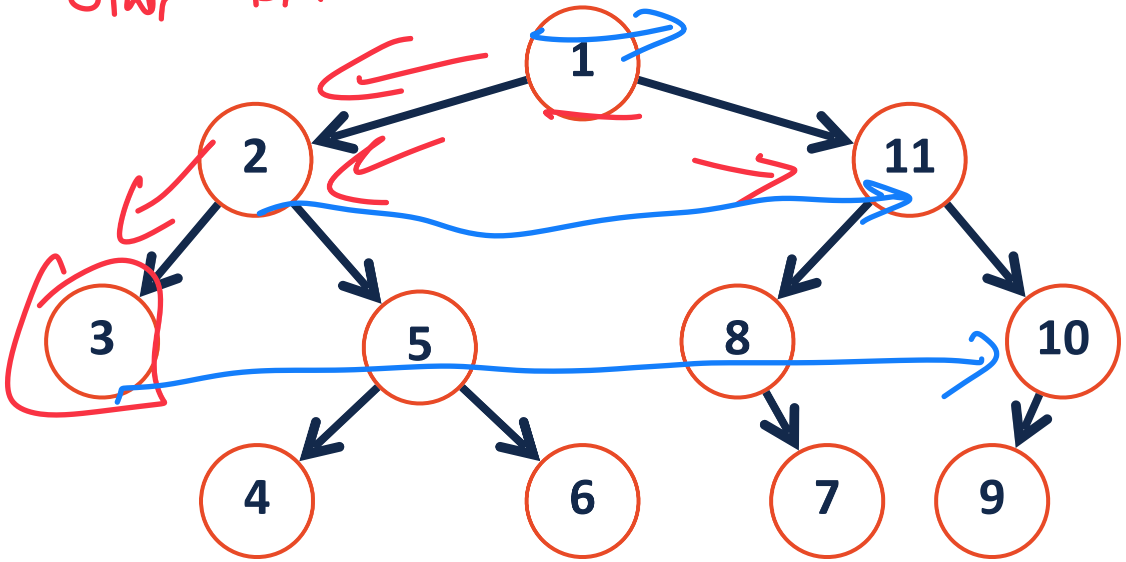
Last class we implemented traversals using recursion, stacks, and queues.

What implementations led to a **depth first search traversal**?

↳ pre, in, post order

↳ Descend down one branch as much as possible (to a leaf) before looking at other branches

↳ Stack



Which lead to **breadth first search?**

↳ clear a level before looking deeper

↳ Queue

1 2 11 3 5 8 10

Binary Tree Utility

This week we will deep dive into useful implementations of binary trees

Binary Search Tree: An efficient implementation of a dictionary

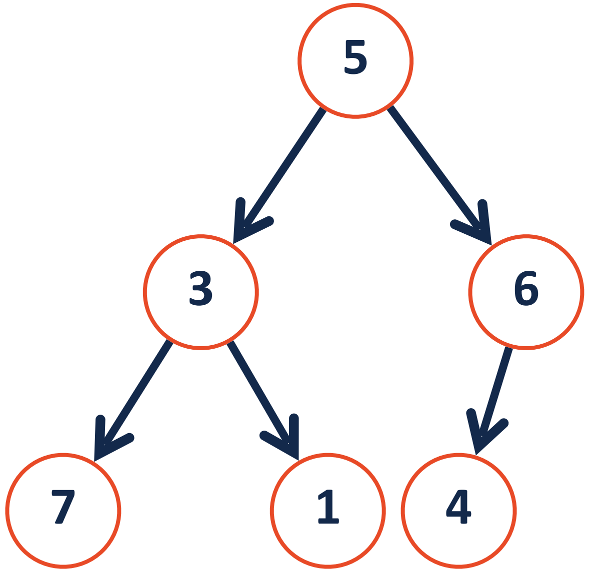
Huffman Tree: A binary tree used to define an optimal text encoding

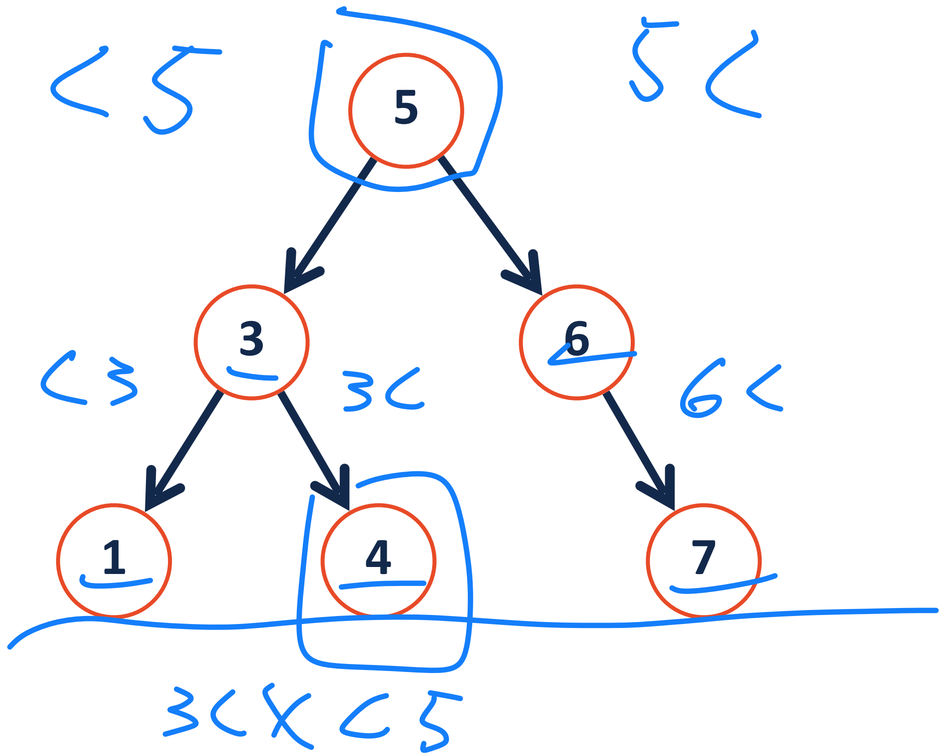
↳ information theory!!

Improved search on a binary tree

5	3	6	7	1	4

1	3	4	5	6	7





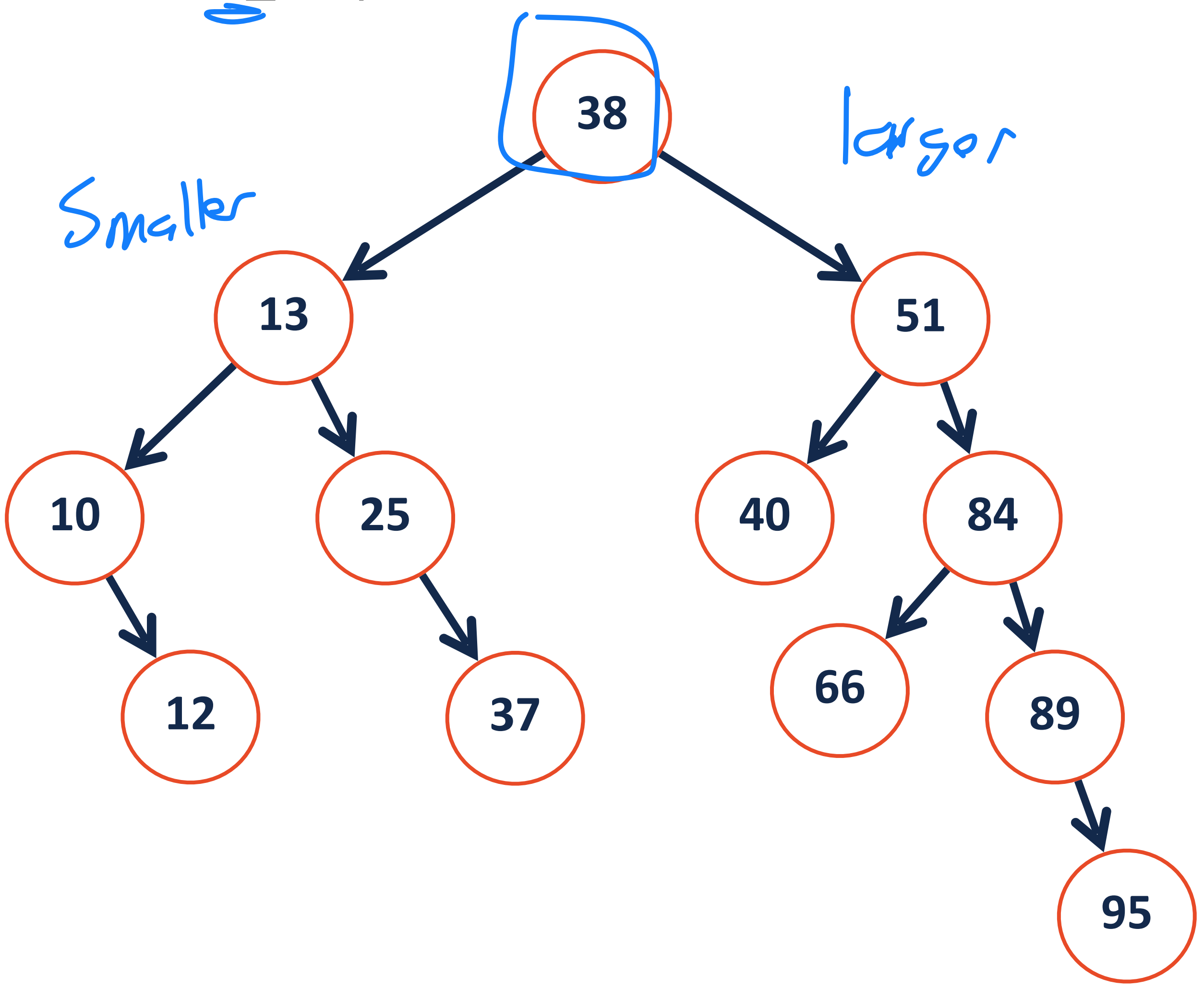
Binary Search Tree (BST)

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

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

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

Added constraint!



Dictionary ADT

Data is often organized into key/value pairs:

Key → Value

Word → Definition

Course Number → Lecture/Lab Schedule

Node → Edges

Flight Number → Arrival Information

URL → HTML Page

Average Image Color → File Location of Image

Dictionaries in Python

```
# The dictionary data structure
d = {}

# Change Value / Insert
d[key] = value
d[k2] = v2
d[key] = v3

# Remove value
d.pop(k2)

# Get Value
print(d[key])
```

Bye

x = { (2: "Hi") }

d[key] = value   d[2] = "Hi"

1) If key not in dictionary, (Insert)
add k, v pair

2) If key is in dictionary,
(Change value)   d[2] = "Bye"

x.pop(2) → will remove
(2: "Hi")

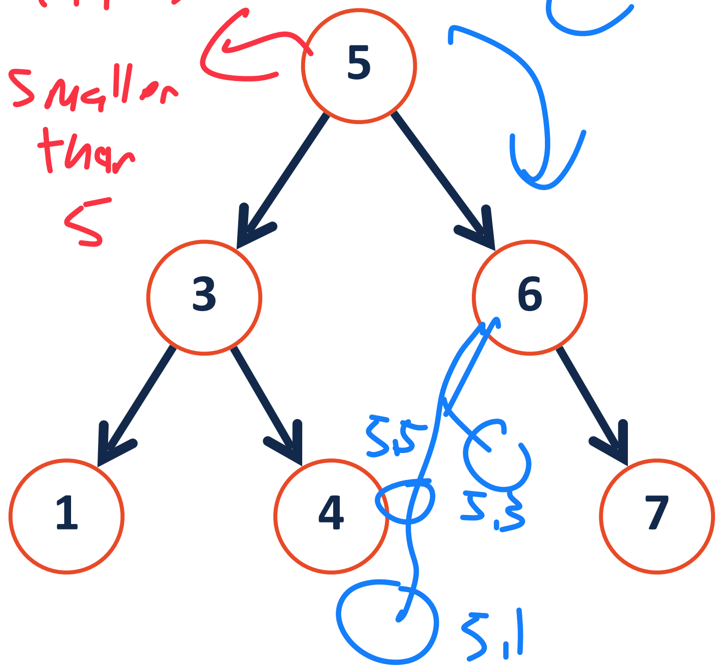
key

Dictionary as a Binary Search Tree

```
class bstNode:
    def __init__(self, key, val, left=None, right=None):
        self.key = key
        self.val = val
        self.left = left
        self.right = right
```

— val now has key + val

left is smaller than 5



1) tree has structure!

↳ More efficient (?)

2) Tree search has meaning

↳ Look up key to get (value)

Key	5	3	6	7	1	4
Value	A	B	C	D	E	F

Binary **Search** Tree ADT — **what changed?**

Constructor: Build a new (empty) tree

Insert: Add an object into tree

Old insert: Parent, direction, value

tree has structure! (Sorted)

↳ A new insert!

Remove: Remove a specific object from tree → A new remove

Traverse: Visit every node in tree (all objects)

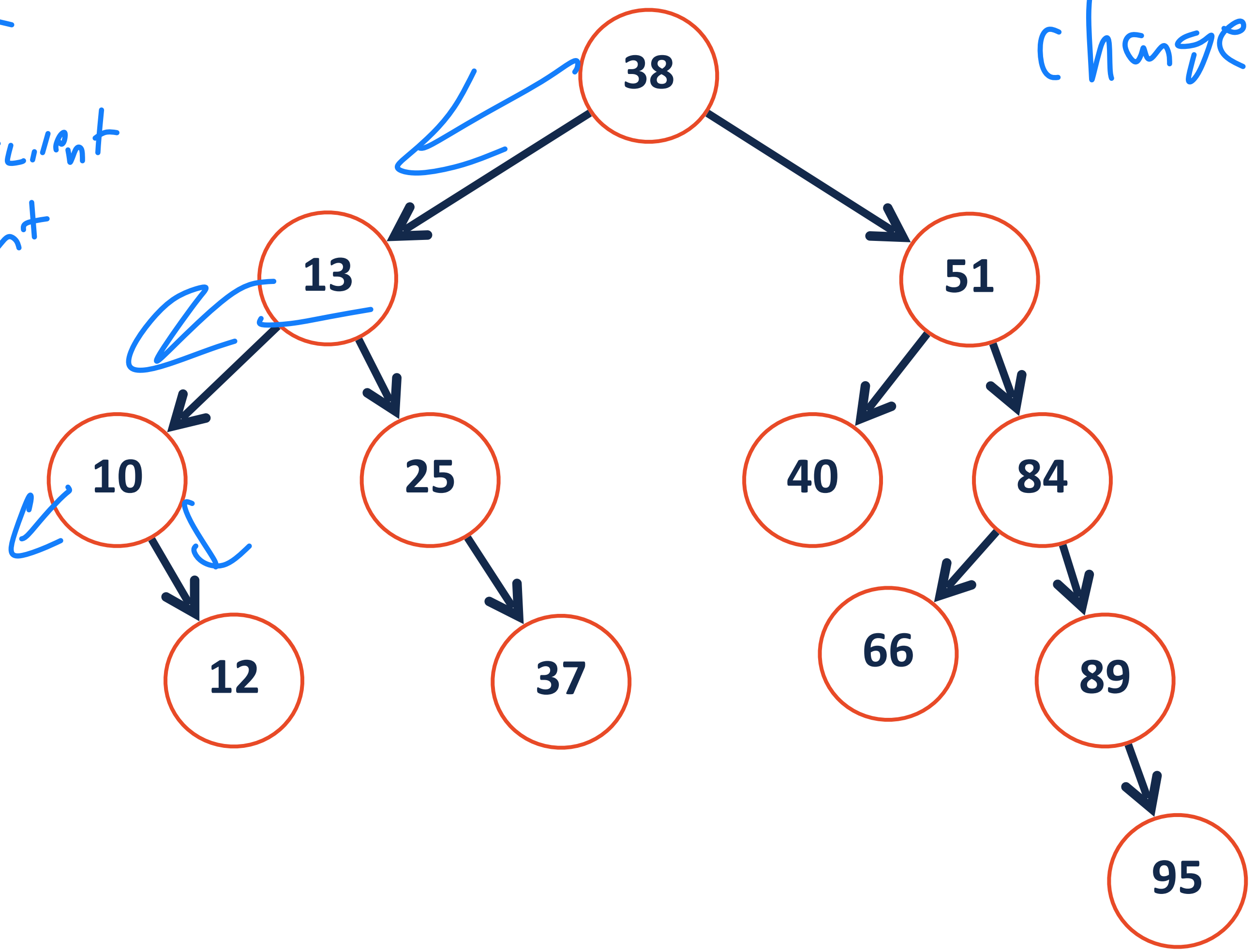
Search: Find a specific object in the tree → Sorted order!

Old search: Do a traversal

BST In-Order Traversal

No traversal doesn't change

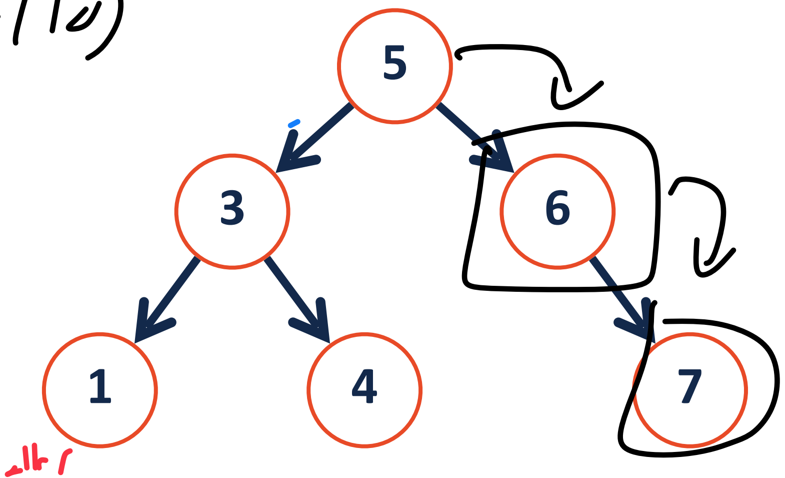
1) Recurse left
2) Process mid/current
3) Recurse right



10 12 13 25 37 38 40 51 66 84 89 95

BST Insert

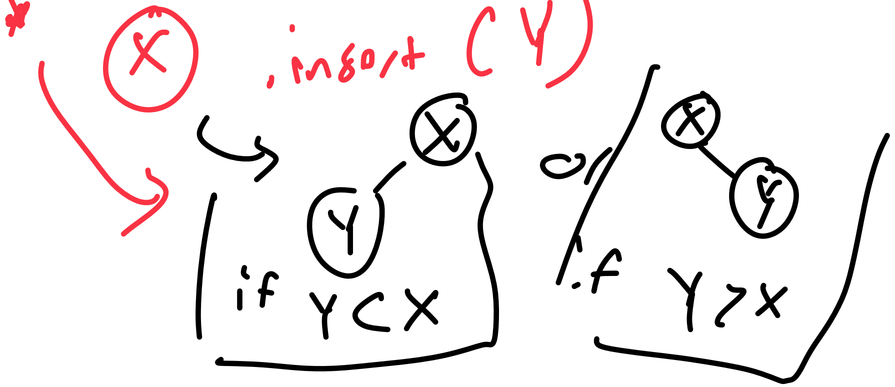
Insert(10)



Base Case:

Tree of size 0 → make new bst Node set to root

Tree of size 1 → Check if insert larger or smaller Set new Node as child*



Recursive Step:

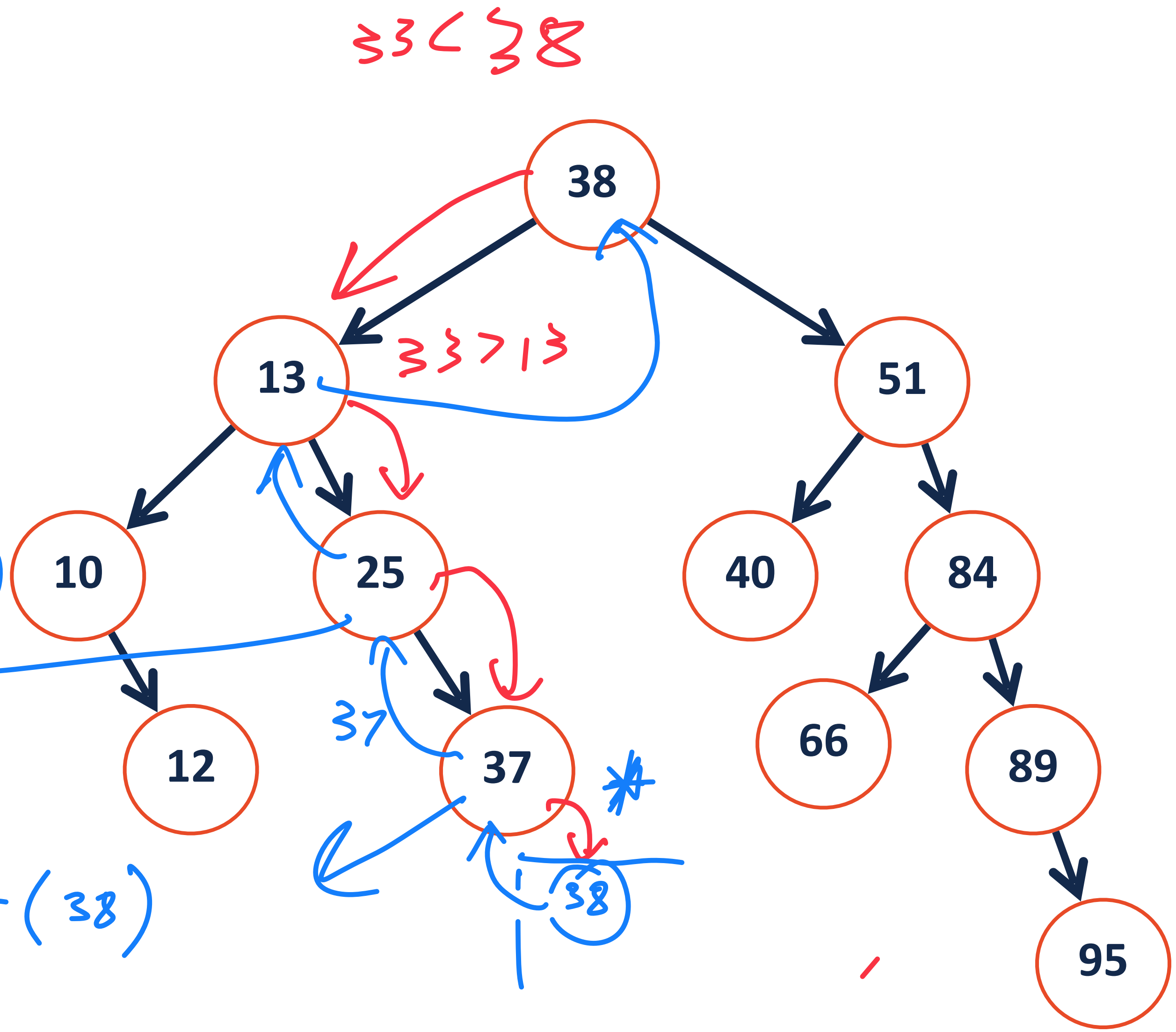
Check if key is larger or smaller than root node

↳ recurse insert to appropriate child

Combining: ??

BST Insert

insert(33)



@25 node.right = insert(38)

@37 node.right = insert(38)

BST Insert

Insert (2)

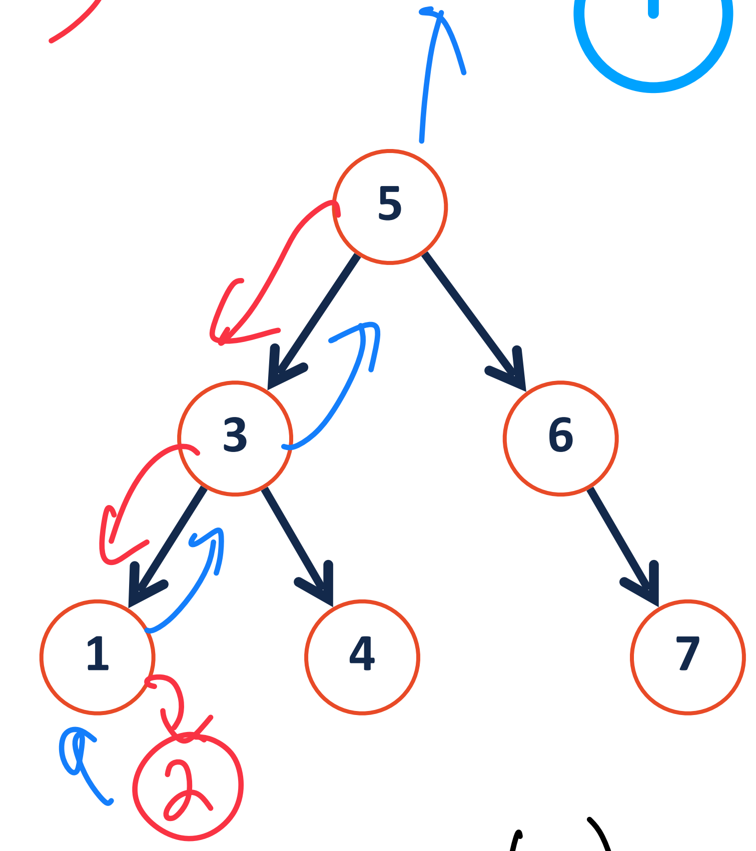
```
# inside class bst
def insert(self, key, val):
    self.root = self.insert_helper(self.root, key, val)

def insert_helper(self, node, key, val):
    if node == None:
        return bstNode(key, val)

    if key < node.key: # look left
        node.left = self.insert_helper(node.left, key, val)
    else: # look right
        node.right = self.insert_helper(node.right, key, val)

    return node
```

Base case

Recursion

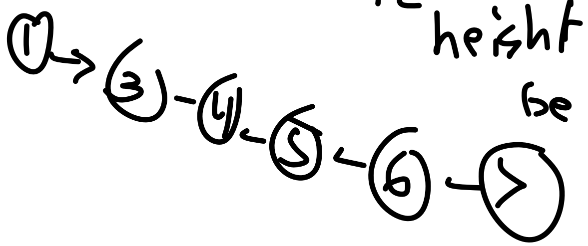
leaf

combining



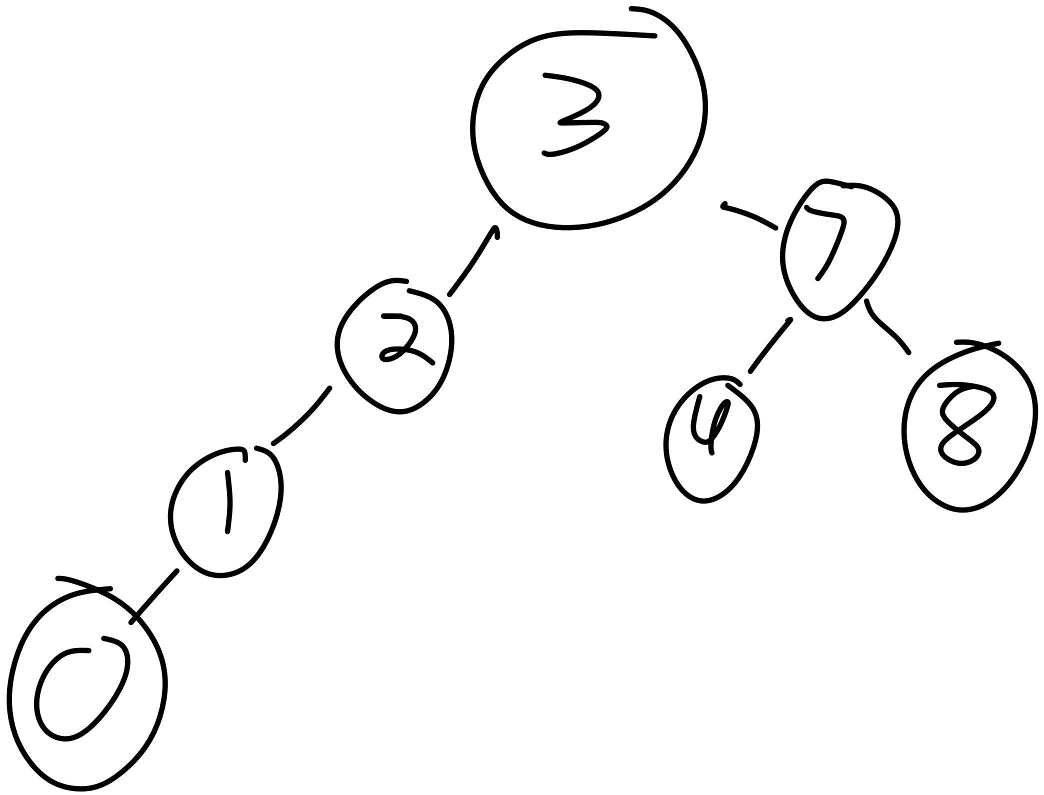
Big O : O(n)

b/c height can be n.



BST Insert

What binary would be formed by inserting the following sequence of integers: `[3, 7, 2, 1, 4, 8, 0]`

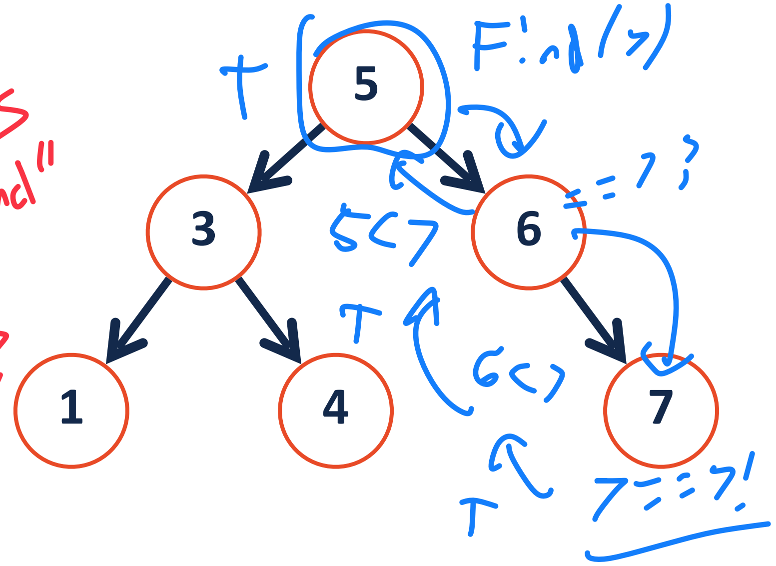


BST Find

Base Case:

Tree is None | Empty → return False / None / −1

what is "Not found" in context of problem?



Recursive Step:

If Key being searched is smaller than Node.key
↳ Recurse Left
else
↳ Recurse Right

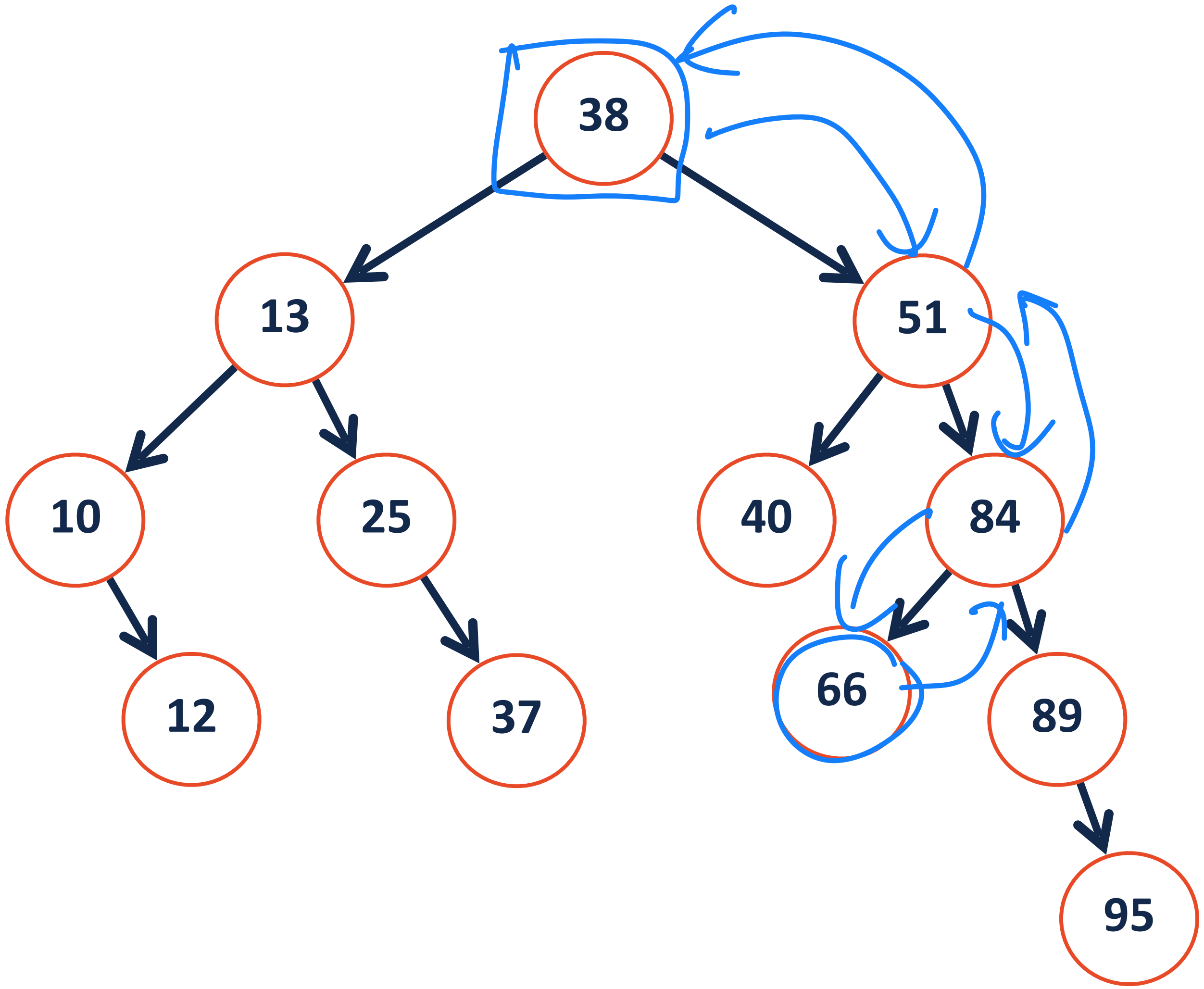
before recursing,
check if Node.Key == key
↳ return True!

Combining:

Return the value from recursion call

BST Find

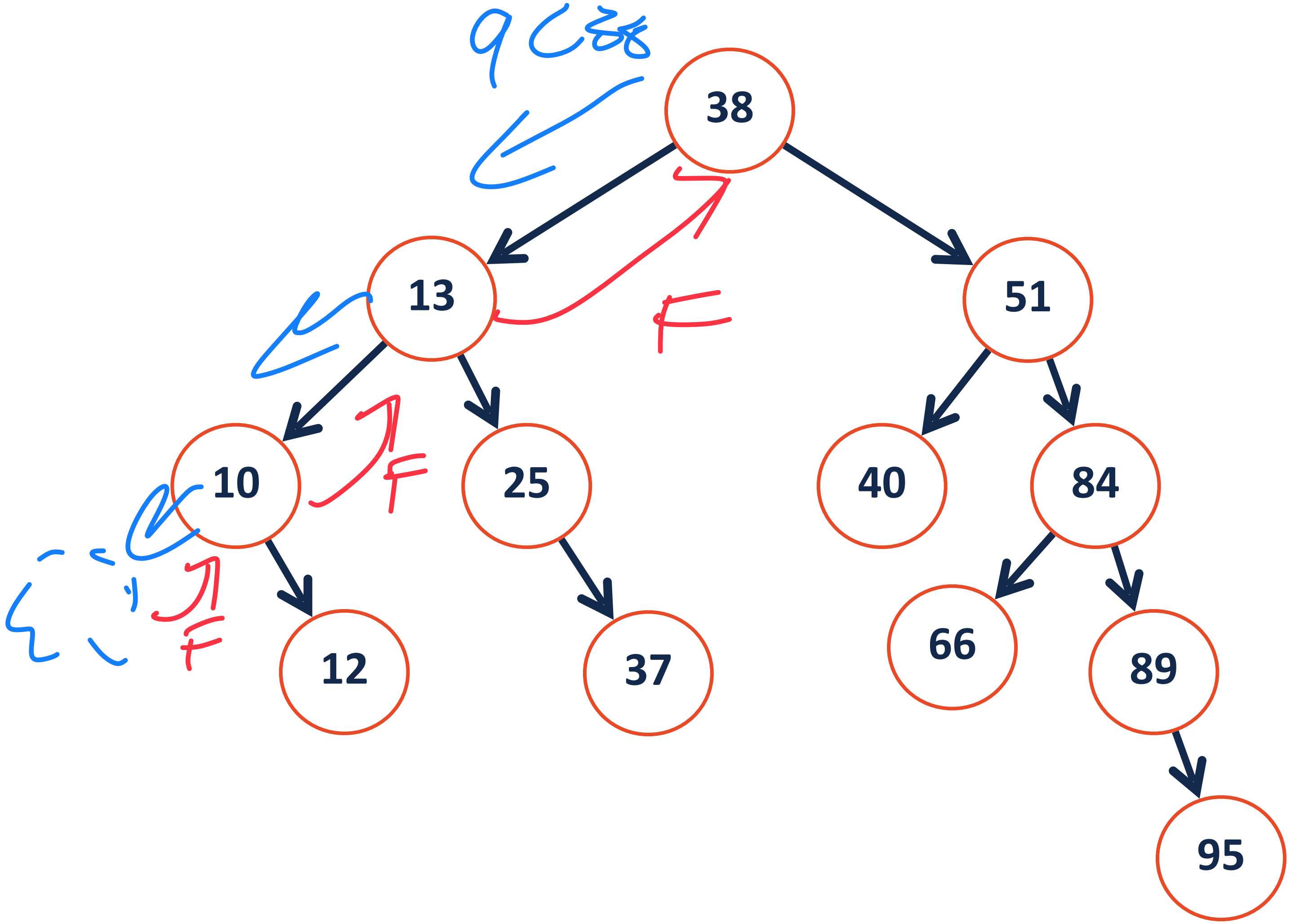
`find(66)`



BST Find

`find(9)`

If we ever reach a Node = None,
value doesn't exist!
↳ False!

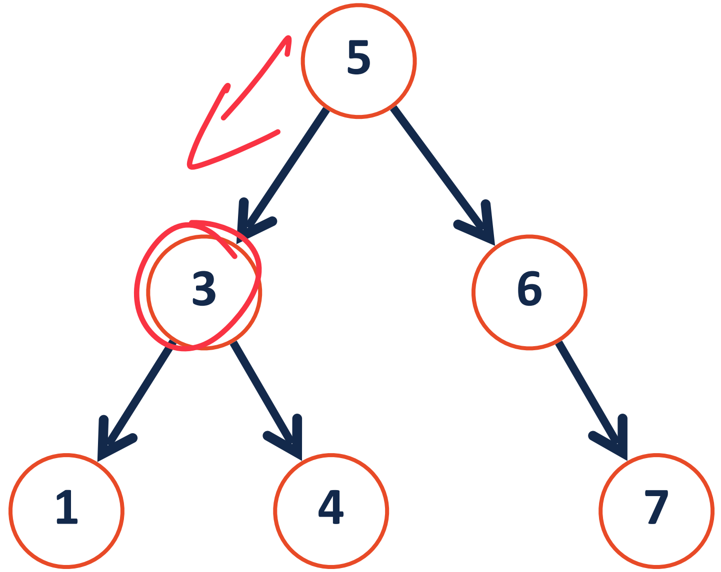


BST Find

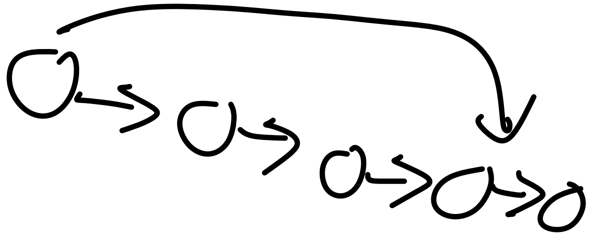
My implementation

Find(3)

```
class bst:
    def find(self, key):
        n = self.find_helper(self.root, key)
        if n:
            return n.val
        else:
            return None

    def find_helper(self, node, key):
        nkey = node.key
        if nkey > key:
            if node.left:
                return self.find_helper(node.left, key)
            else:
                return None
        elif nkey < key:
            if node.right:
                return self.find_helper(node.right, key)
            else:
                return None
        else:
            return node
```



Big O: O(n)



BST Remove

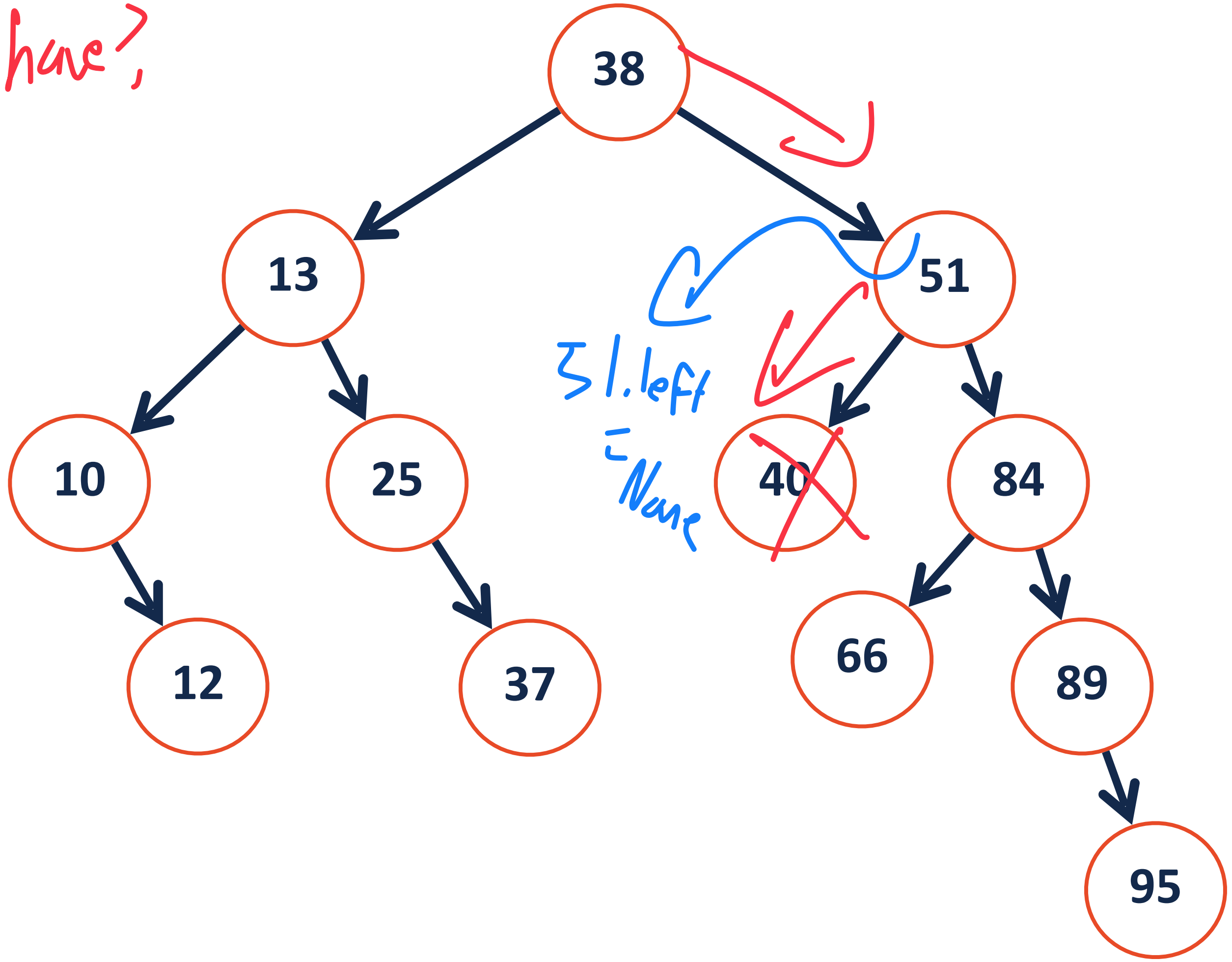
remove(40)

1) Find (40)

How many children does 40 have?

0-child removal

↳ Set parent (at correct child) to None



BST Remove

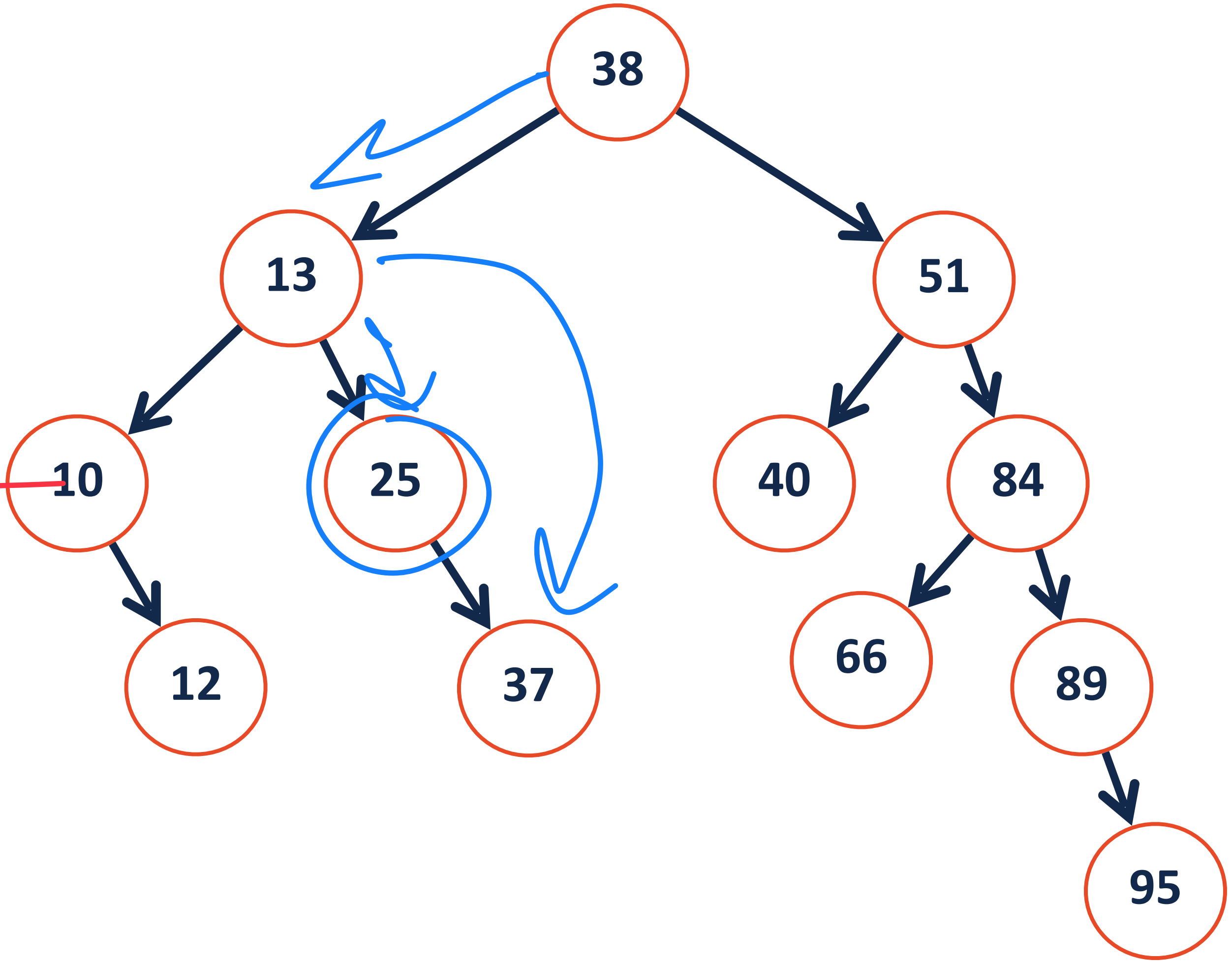
remove(25)

1) Find(25)

2) # children?

Linked list removal *

parent.direction = removed nodes child



BST Remove

remove(13)

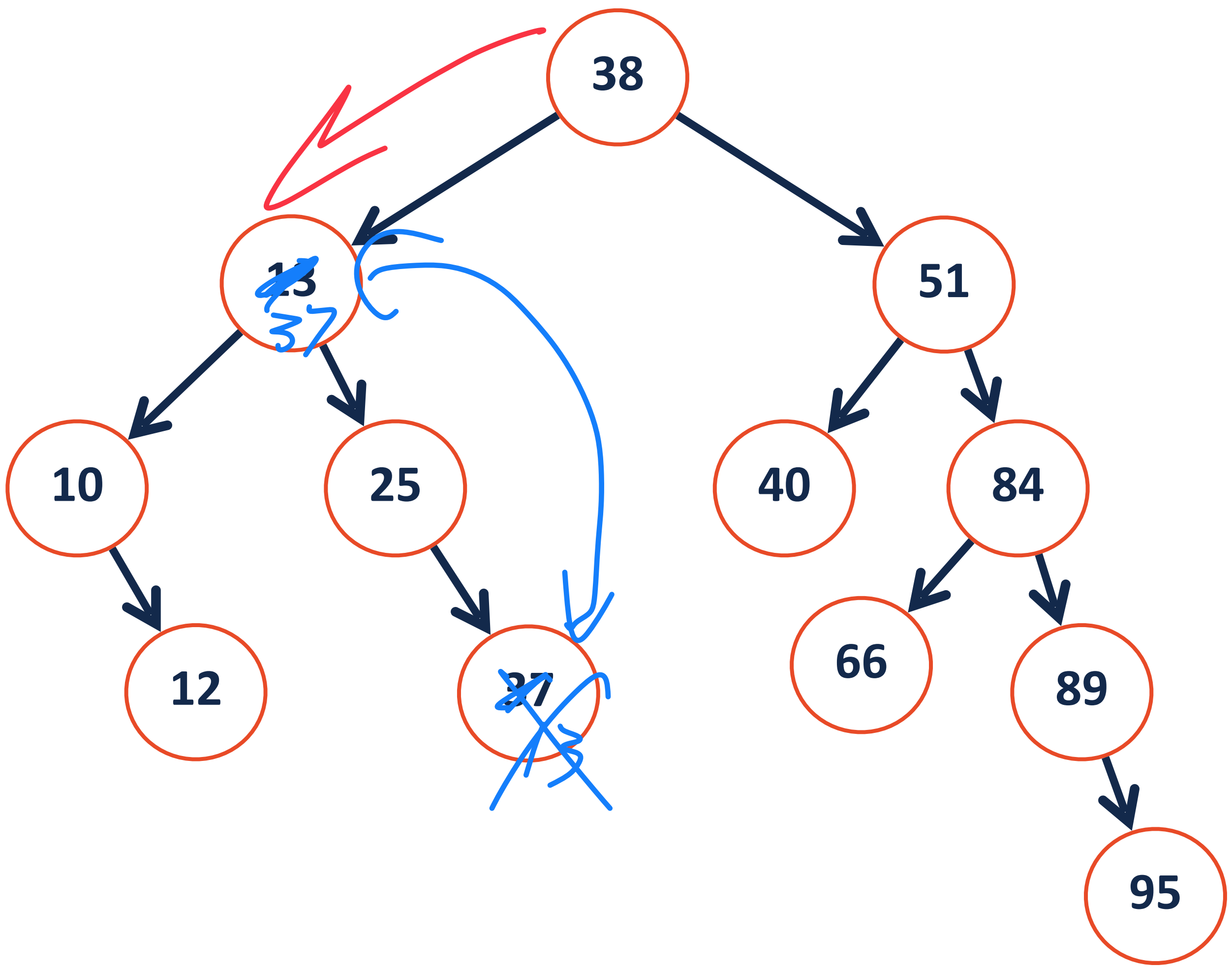
1) Find (13)

2) # children?

↳ 2 children

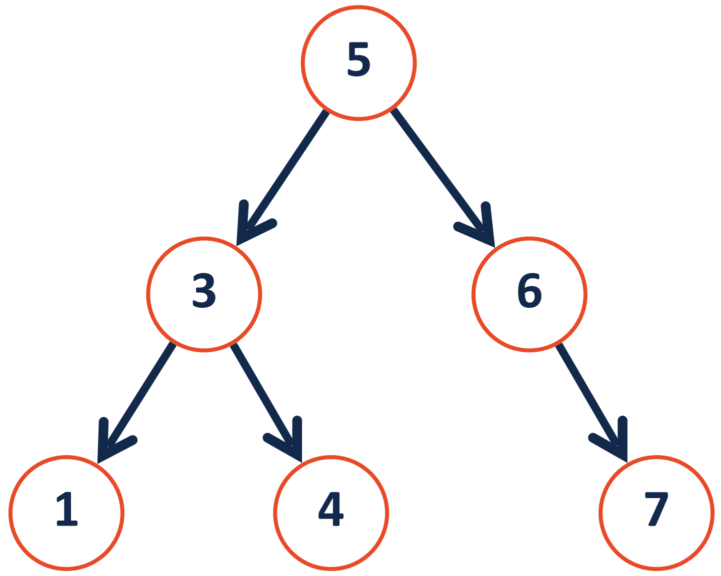
Before swap w/ leaf was ok!

But now... It can break BSTree property



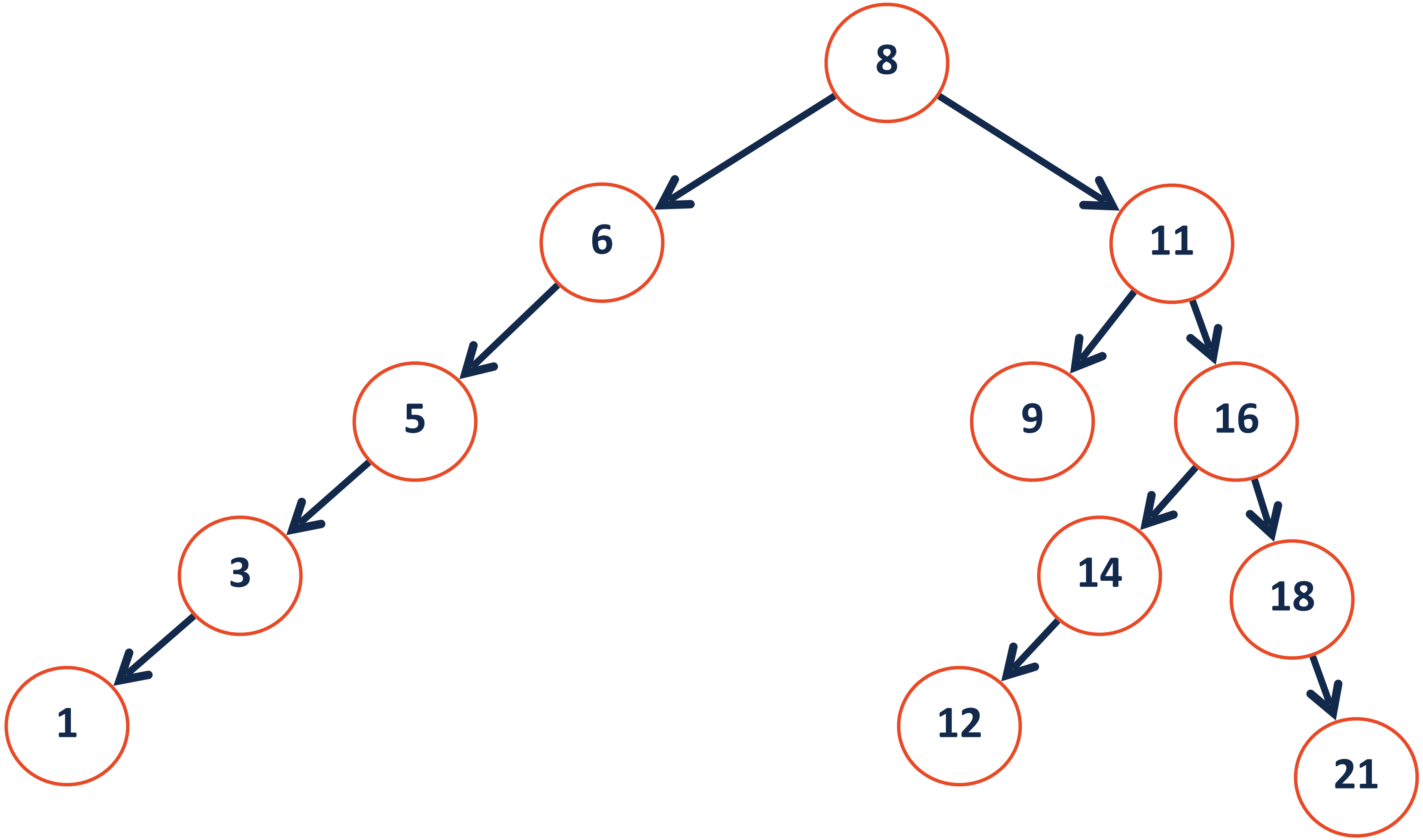
BST Remove

```
def remove(self, key):
    self.root = self.remove_helper(self.root, key)

def remove_helper(self, node, key):
```



BST Remove

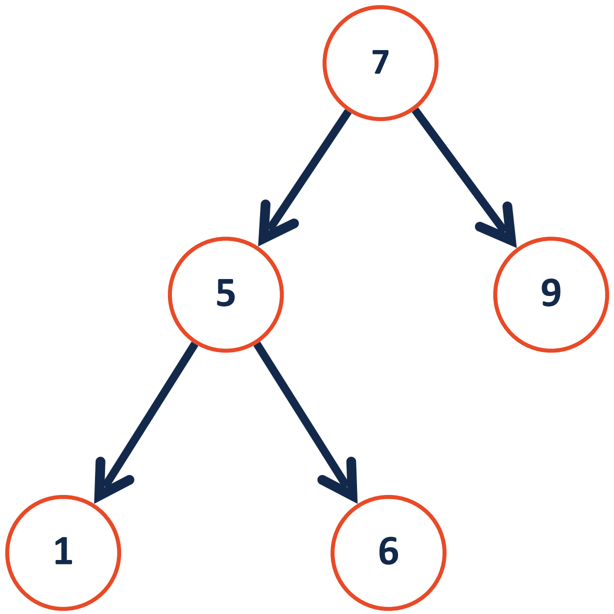
What will the tree structure look like if we remove node 16 using IOS?

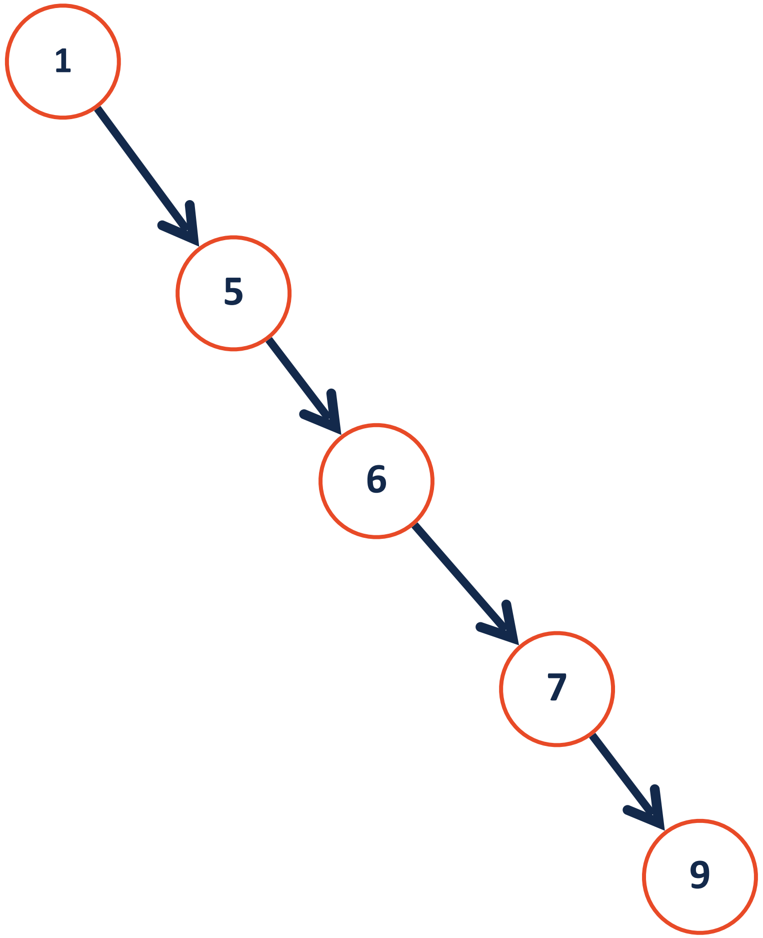


BST Analysis – Running Time

Operation	BST Worst Case
find	
insert	
delete	
traverse	

Limiting the height of a tree





Option A: Correcting bad insert order

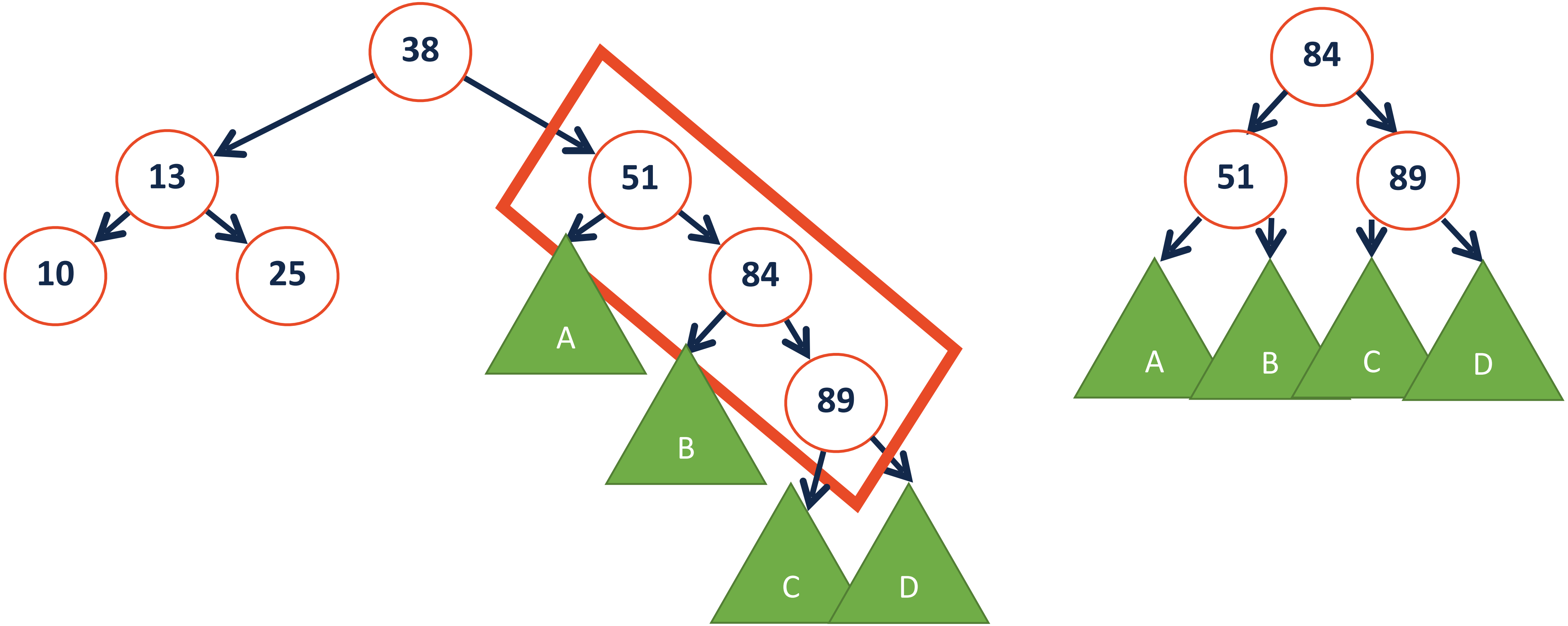
The height of a BST depends on the order in which the data was inserted

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

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

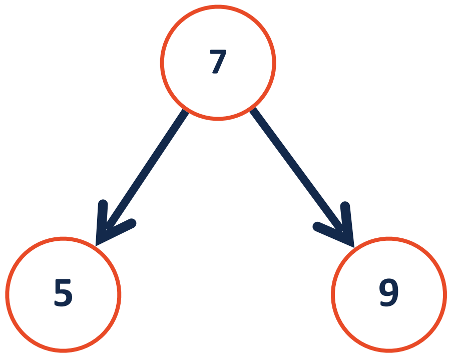
AVL-Tree: A self-balancing binary search tree

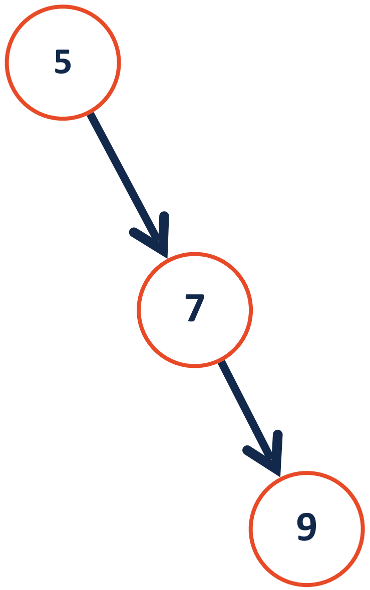
Rather than fixing an insertion order, just correct the tree as needed!



Height-Balanced Tree

What tree is better?





How would you describe this mathematically?