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

Review understanding of Binary Trees

Introduce the dictionary ADT

Extend ADT to Binary Search Trees

Practice recursion in the context of trees

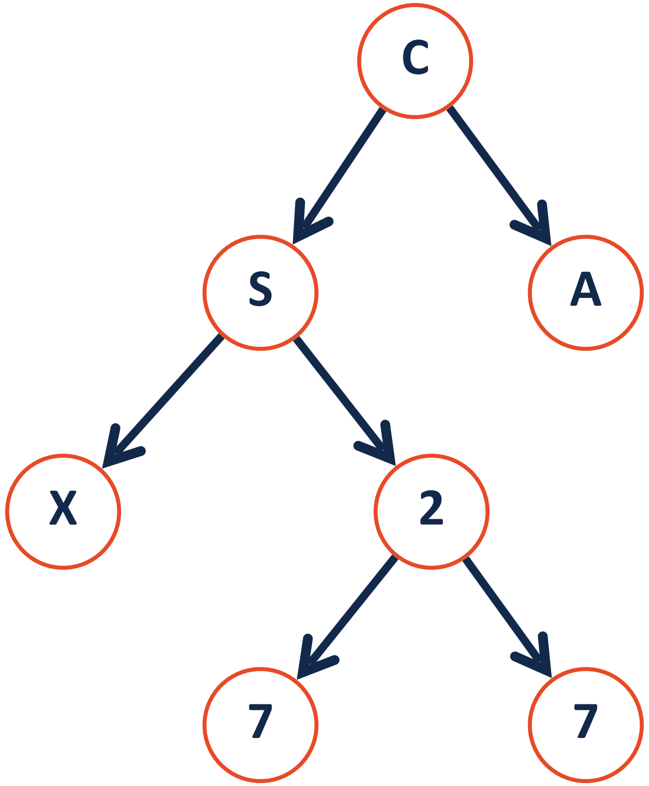
Binary Tree Recursion

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)

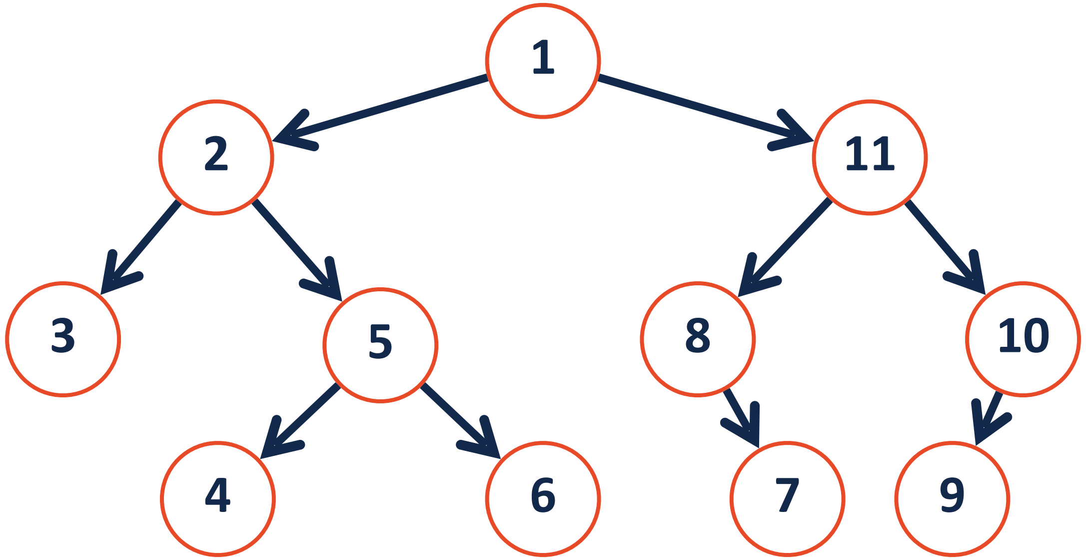
Search: Find a specific object in the tree

Binary Tree Traversal

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

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

Which lead to **breadth first search?**



Binary Tree Utility

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

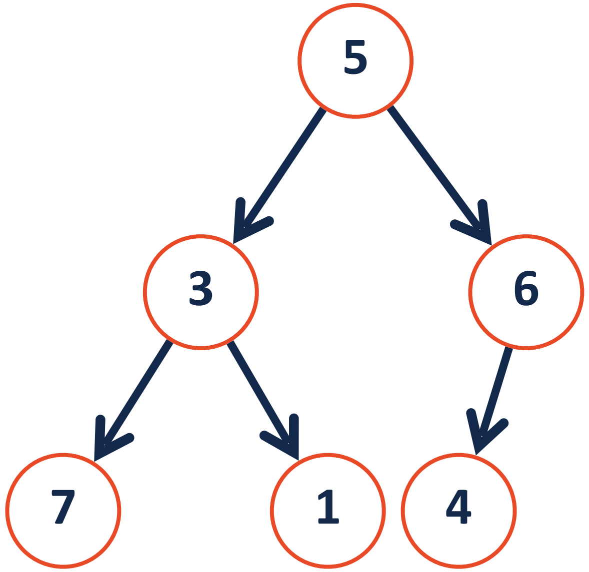
Binary Search Tree: An efficient implementation of a dictionary

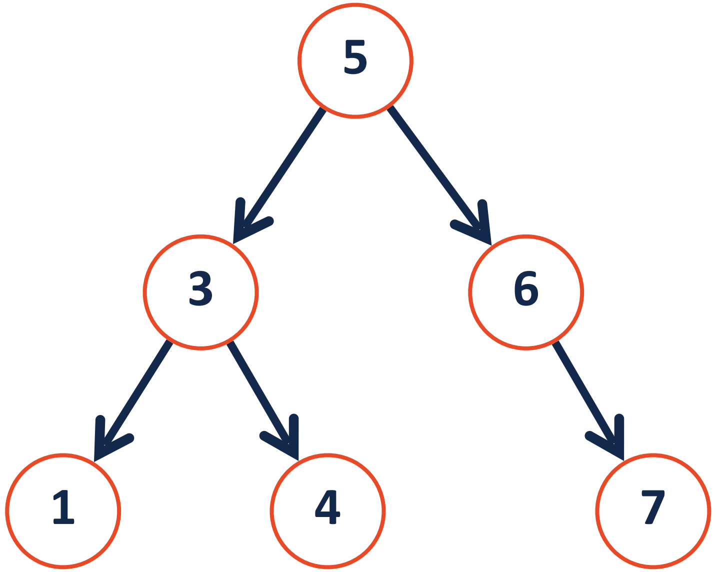
Huffman Tree: A binary tree used to define an optimal text encoding

Improved search on a binary tree

5	3	6	7	1	4

1	3	4	5	6	7



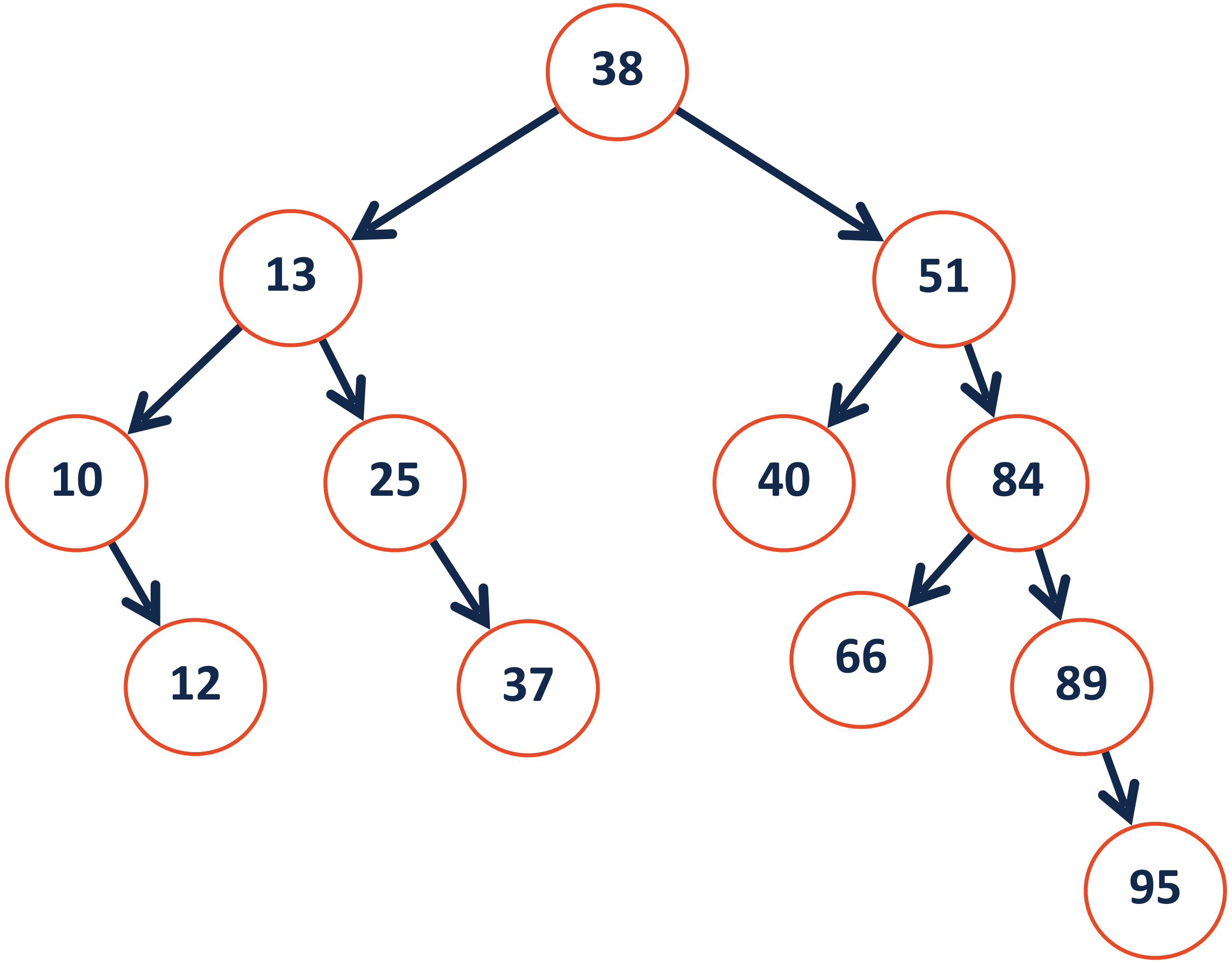


Binary Search Tree (BST)

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

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

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



Dictionary ADT

Data is often organized into key/value pairs:

Word ➔ Definition

Course Number ➔ Lecture/Lab Schedule

Node ➔ Edges

Flight Number ➔ Arrival Information

URL ➔ HTML Page

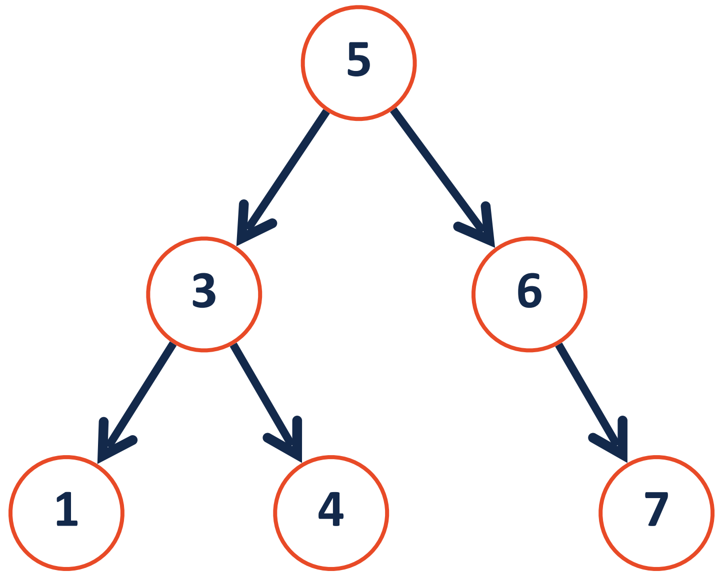
Average Image Color ➔ File Location of Image

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])
```

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
```



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

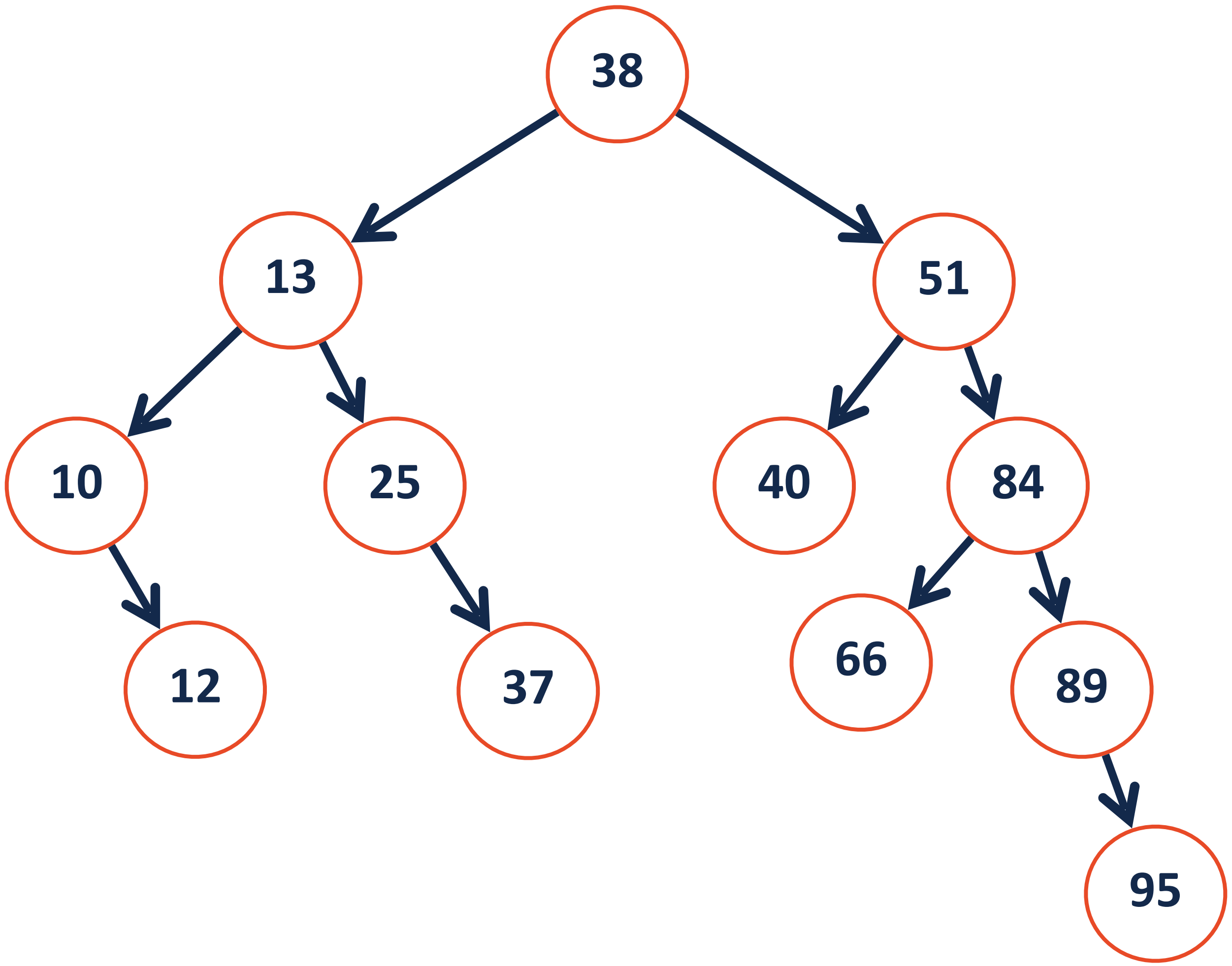
Insert: Add an object into tree

Remove: Remove a specific object from tree

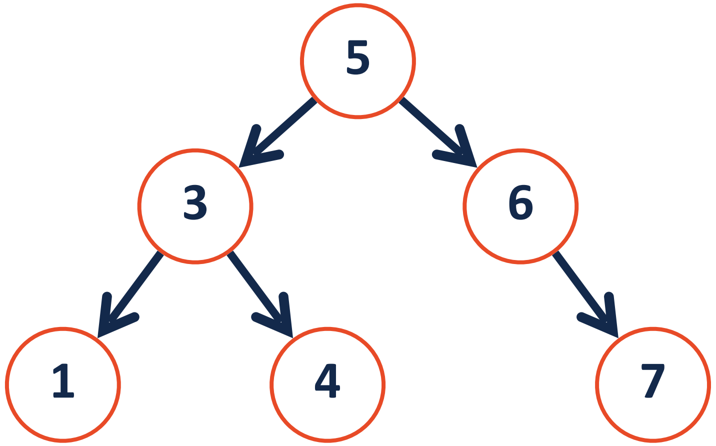
Traverse: Visit every node in tree (all objects)

Search: Find a specific object in the tree

BST In-Order Traversal



BST Insert



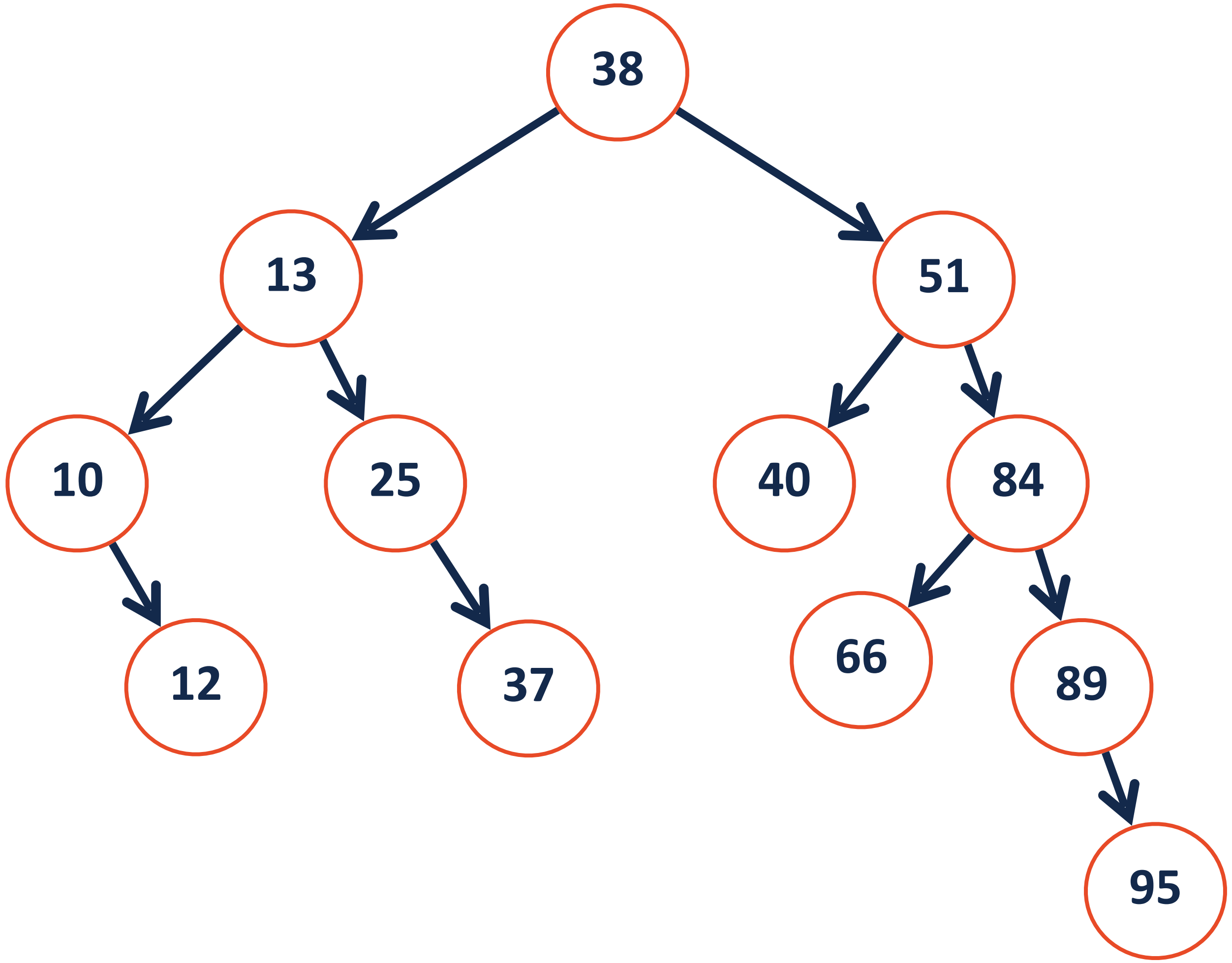
Base Case:

Recursive Step:

Combining:

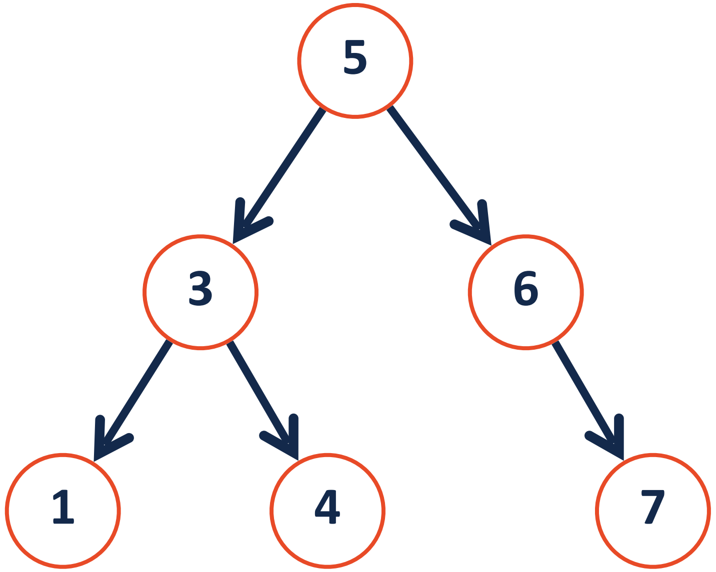
BST Insert

`insert(33)`



BST Insert

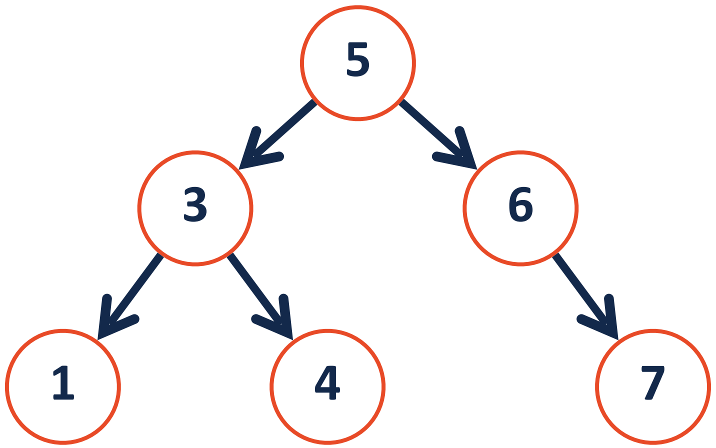
```
# inside class bst
def insert(self, key, val):
    self.root = self.insert_helper(self.root, key, val)

def insert_helper(self, node, key, val):
```



BST Insert

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

BST Find



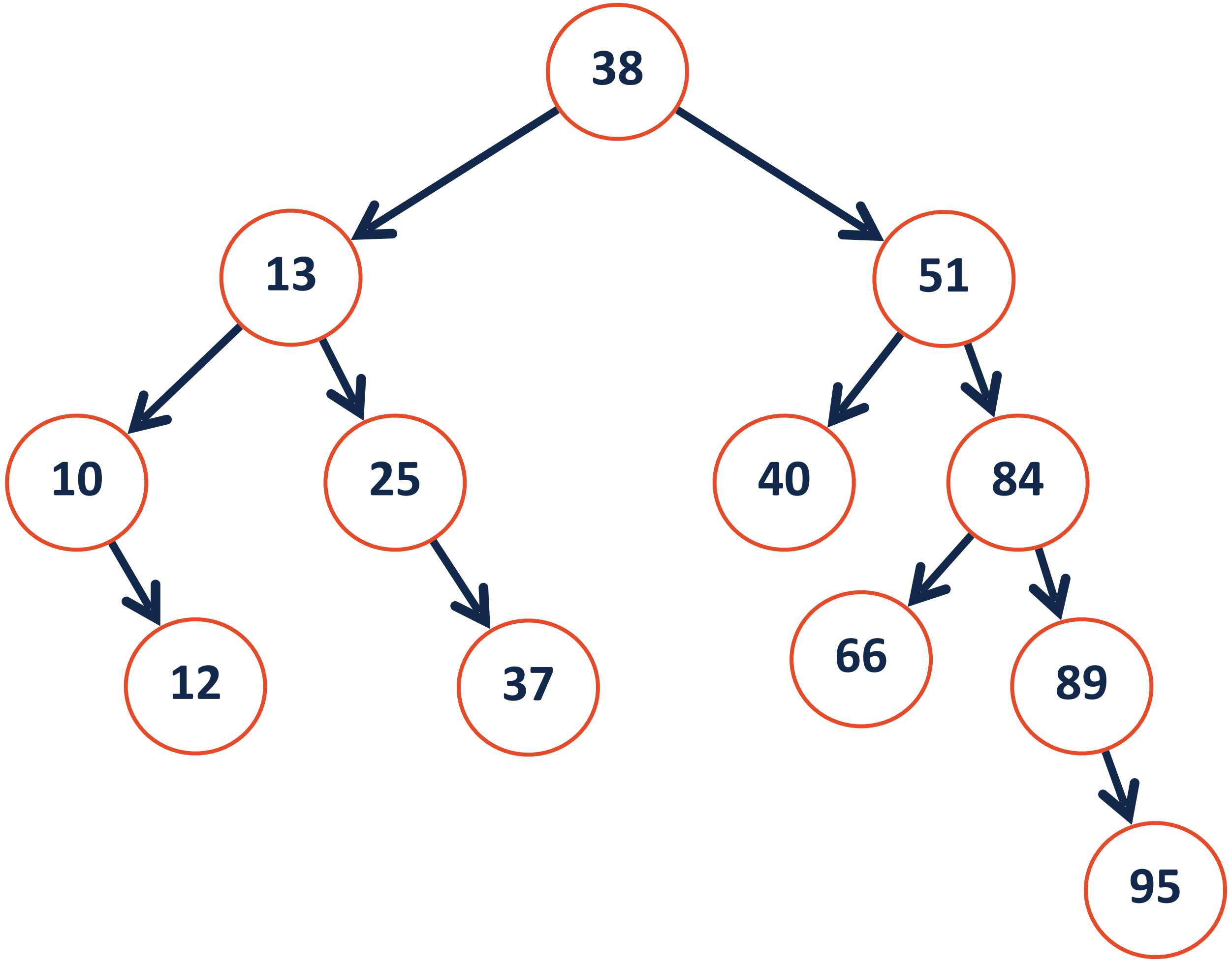
Base Case:

Recursive Step:

Combining:

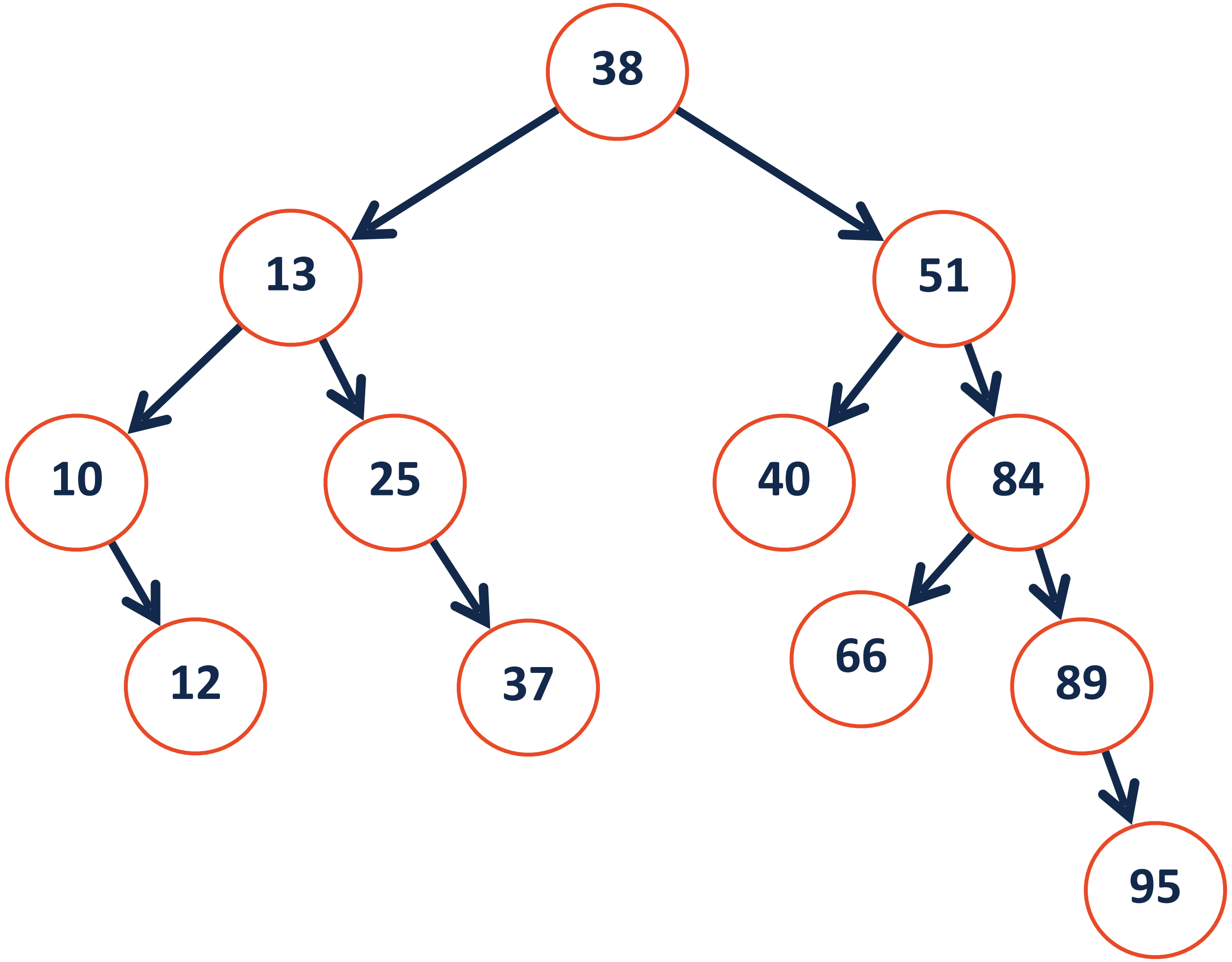
BST Find

`find(66)`



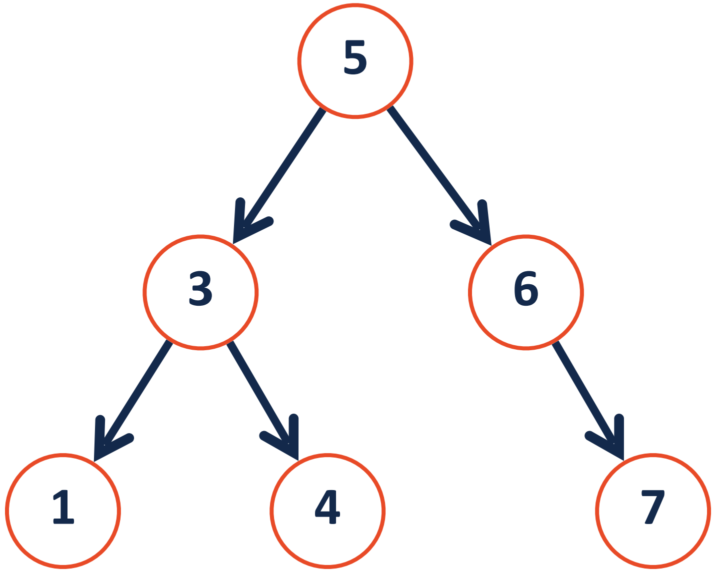
BST Find

`find(9)`



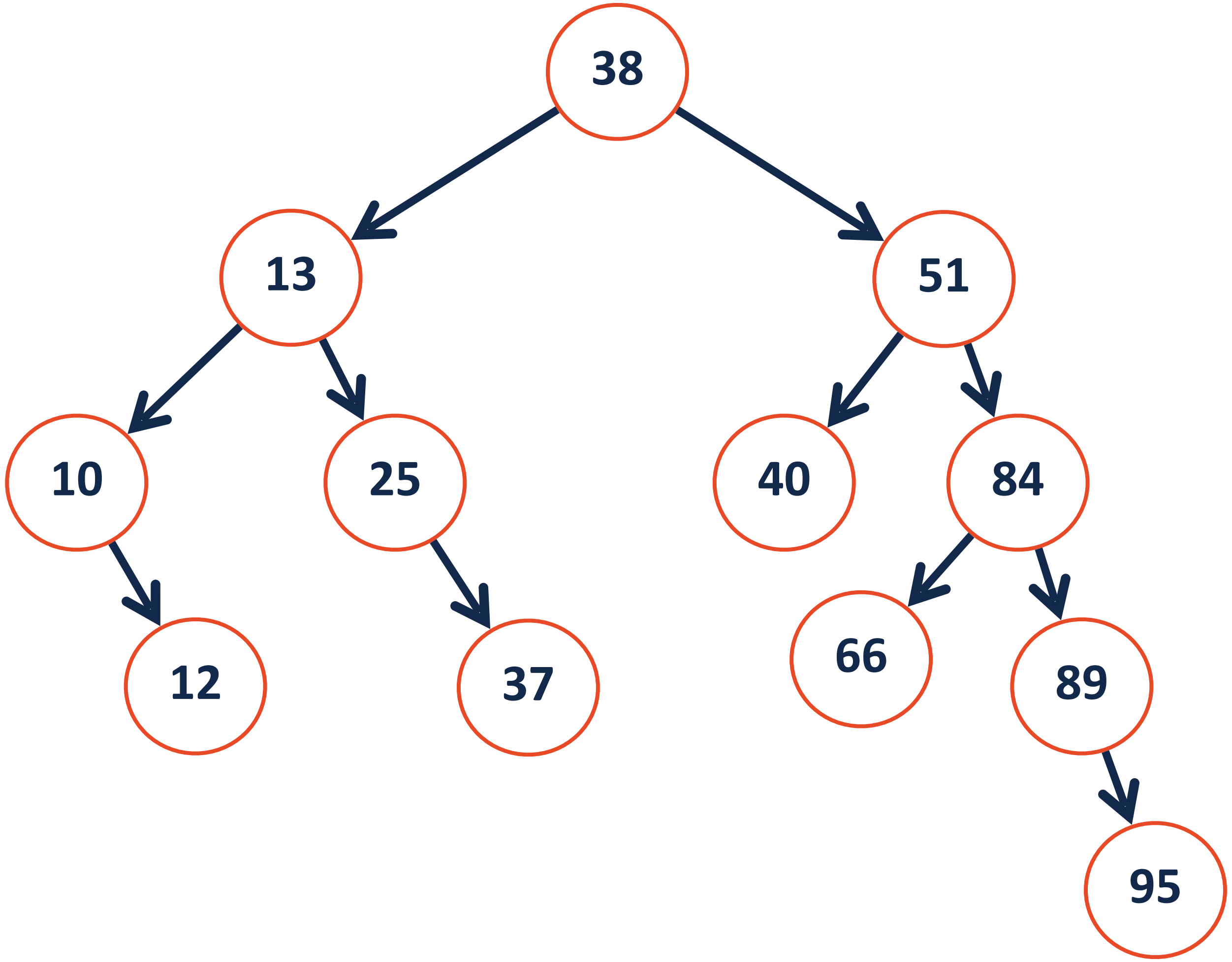
BST Find

```
#inside class bst
def find(self, key):






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



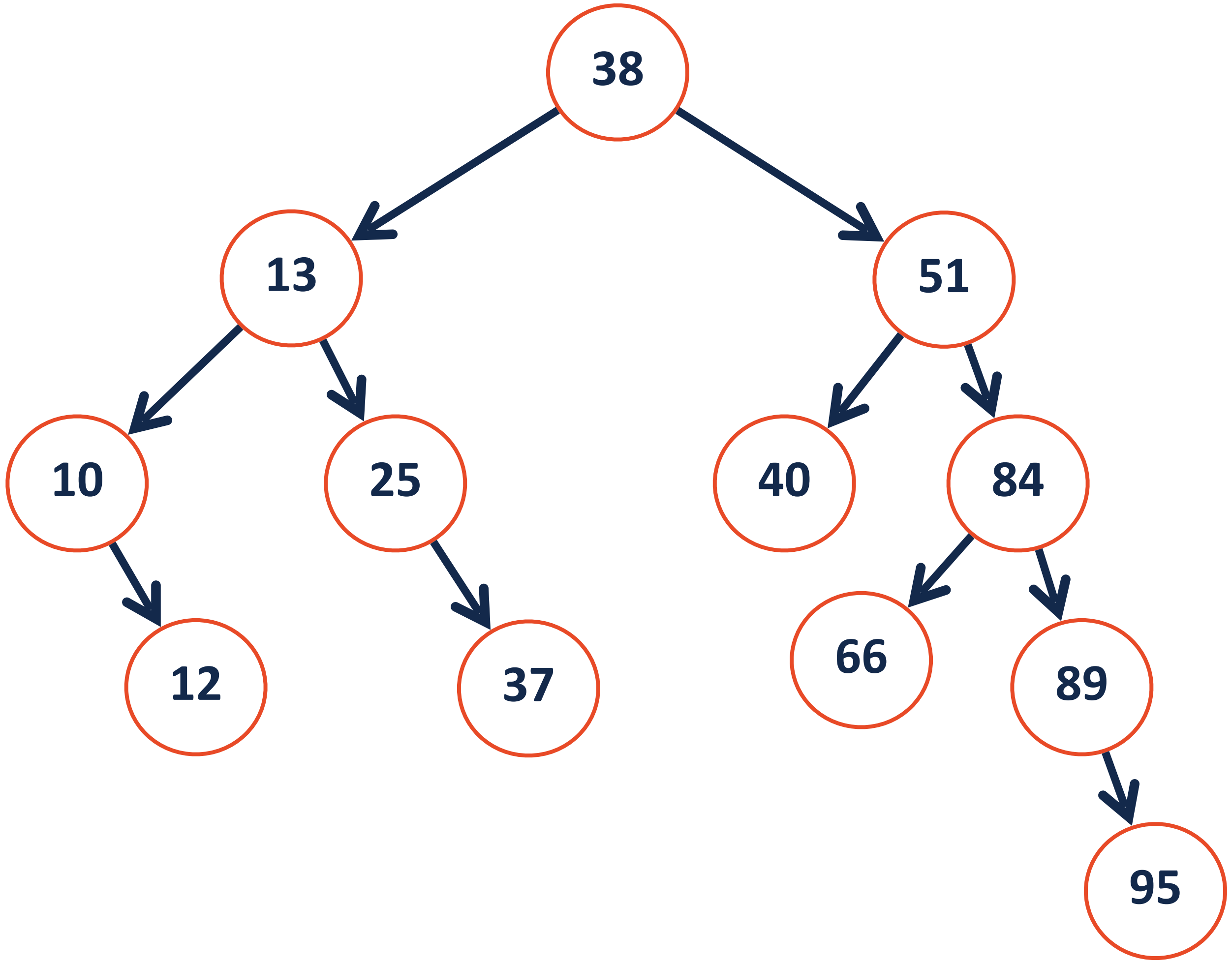
BST Remove

`remove(40)`



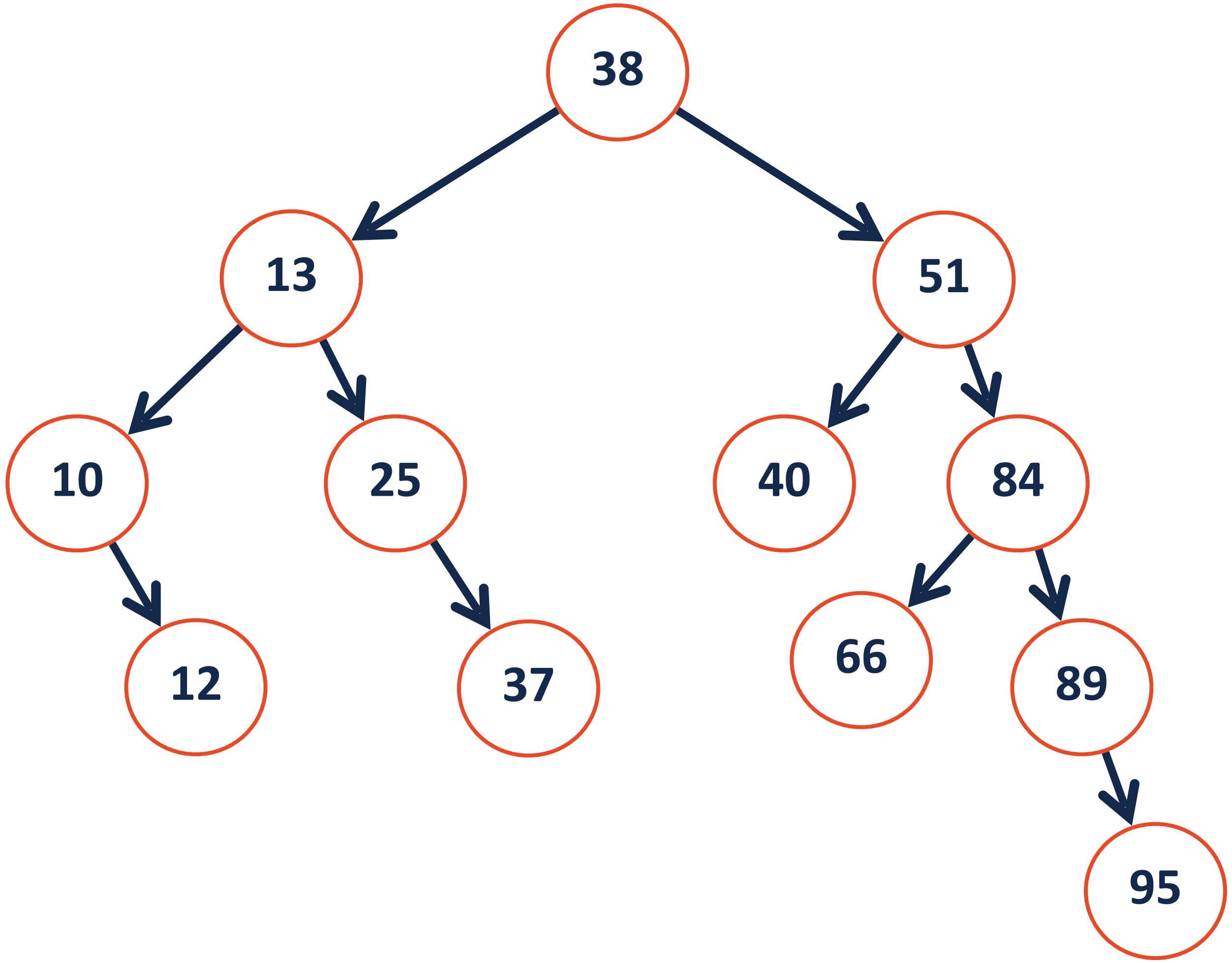
BST Remove

`remove(25)`



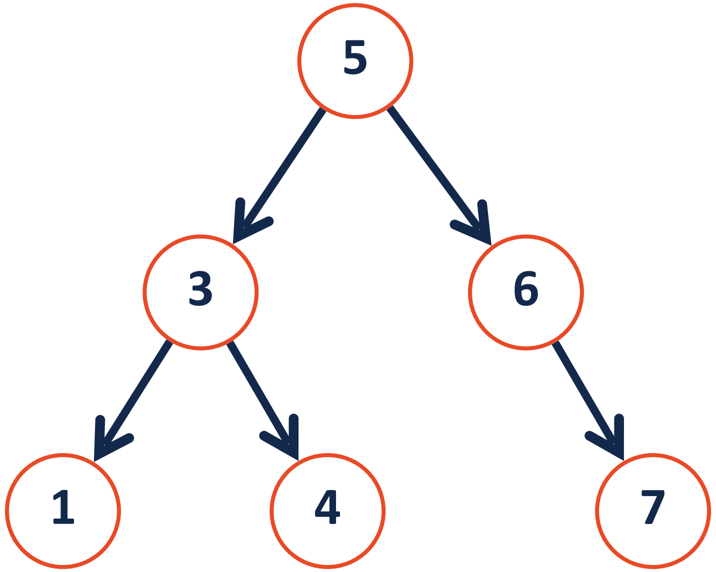
BST Remove

`remove(13)`



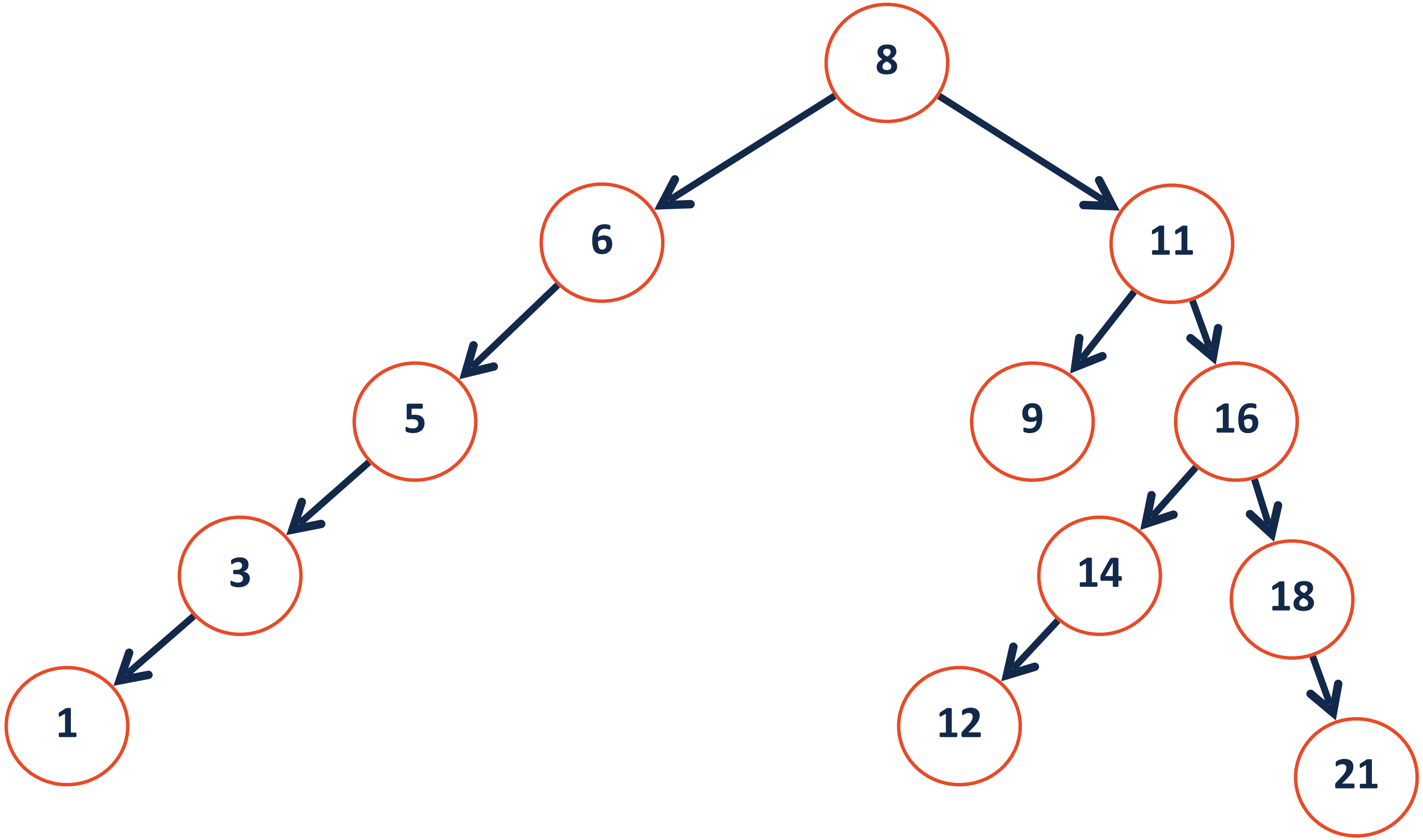
BST Remove

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

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



BST Remove

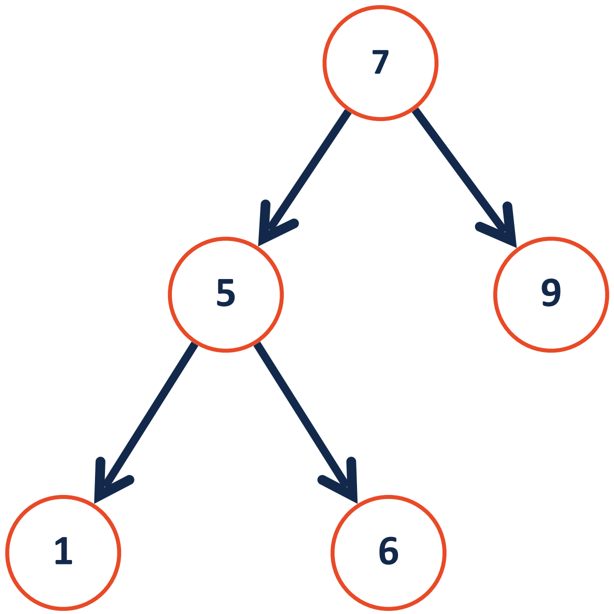
What will the tree structure look like if we remove node 16 using IOS?

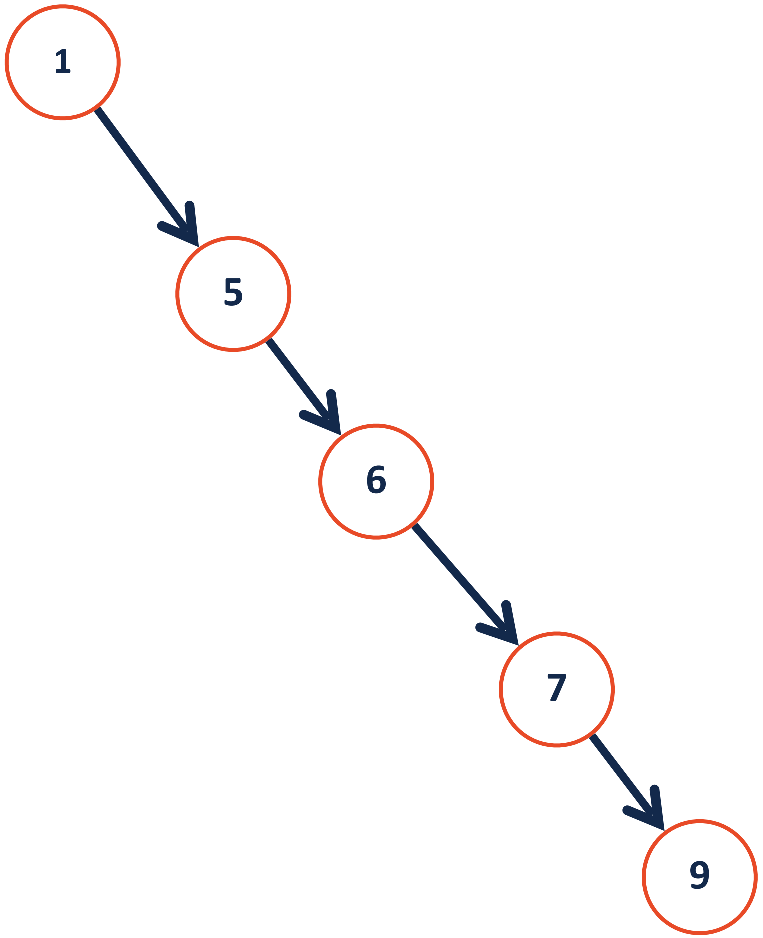


BST Analysis – Running Time

Operation	BST Worst Case
find	
insert	
delete	
traverse	

Limiting the height of a tree





Option A: Correcting bad insert order

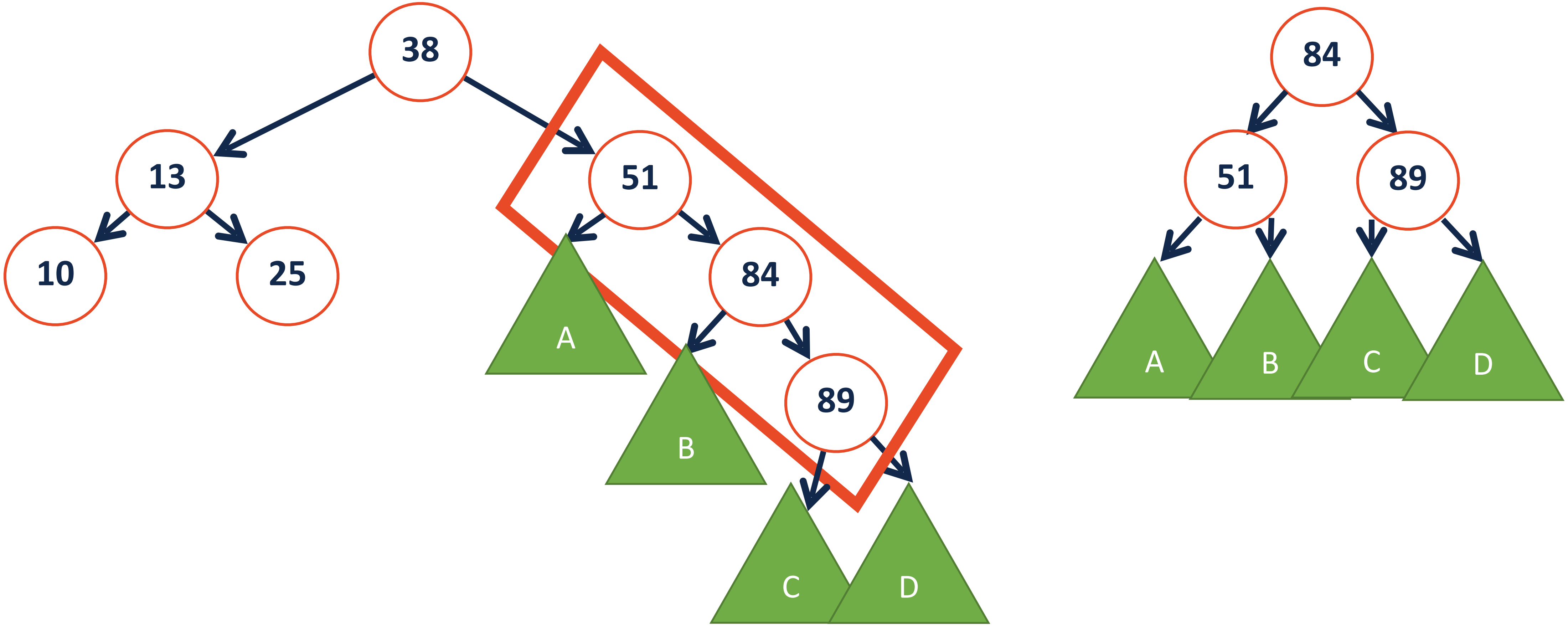
The height of a BST depends on the order in which the data was inserted

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

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

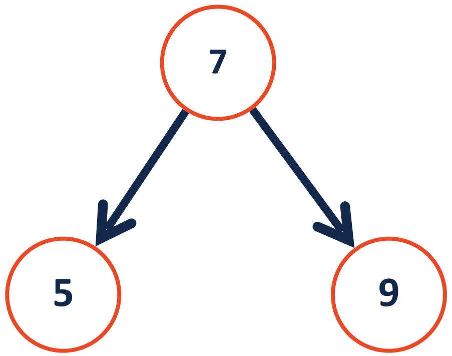
AVL-Tree: A self-balancing binary search tree

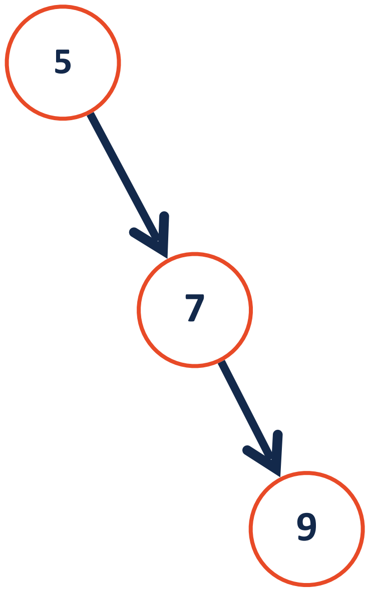
Rather than fixing an insertion order, just correct the tree as needed!



Height-Balanced Tree

What tree is better?





How would you describe this mathematically?