Data Structures

Balanced Binary Search Trees

CS 225
Harsha Tirumala

October 1, 2025

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Announcements

Exam 2 - 10/01 to 10/03

MP_Stickers survey processed - we will make some changes!

Mp_Lists survey out Today

Exam Regrades - 1. Go over exam with a staff member
2. If unhappy, fill out regrade request form
(and mention staff member's name)

Learning Objectives

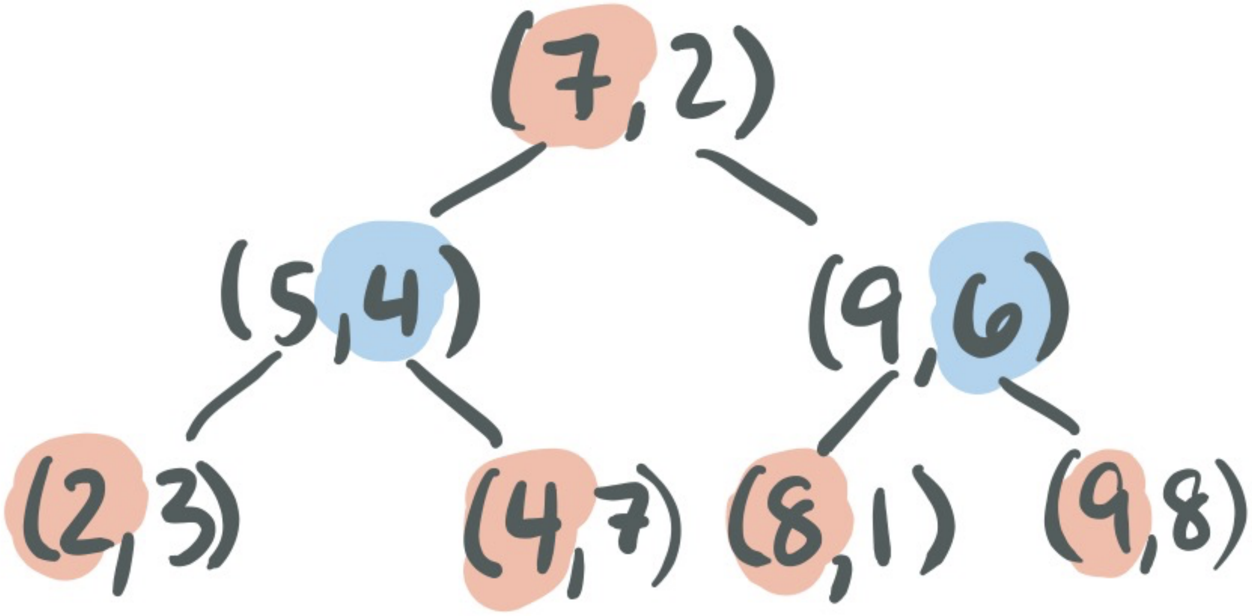
Review kd tree : Nearest Neighbor Search

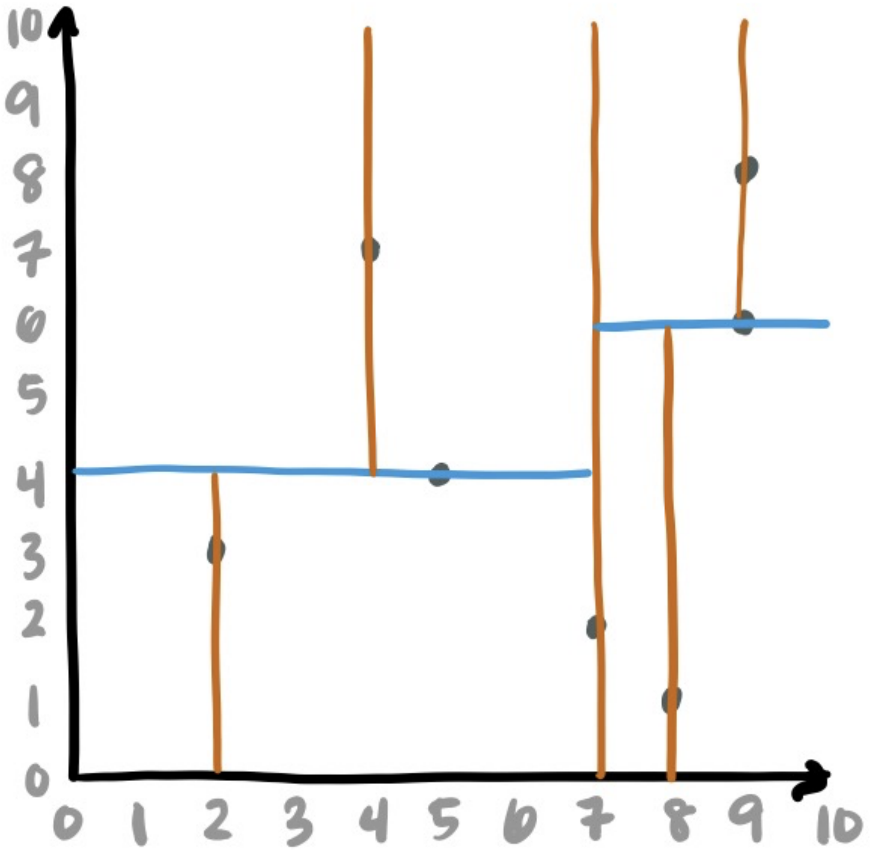
Briefly review BST in the context of height

BST : Challenges and Solutions

AVL Tree : self-balancing BST

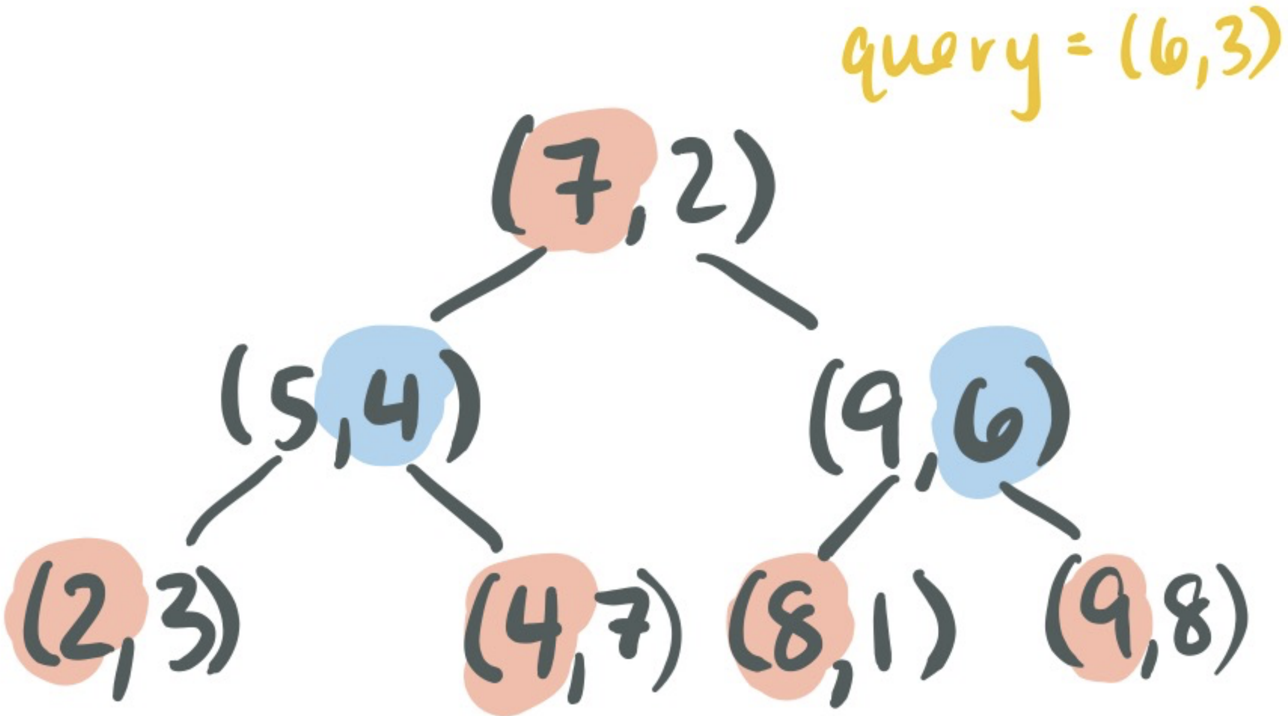
Nearest Neighbor: k-d tree

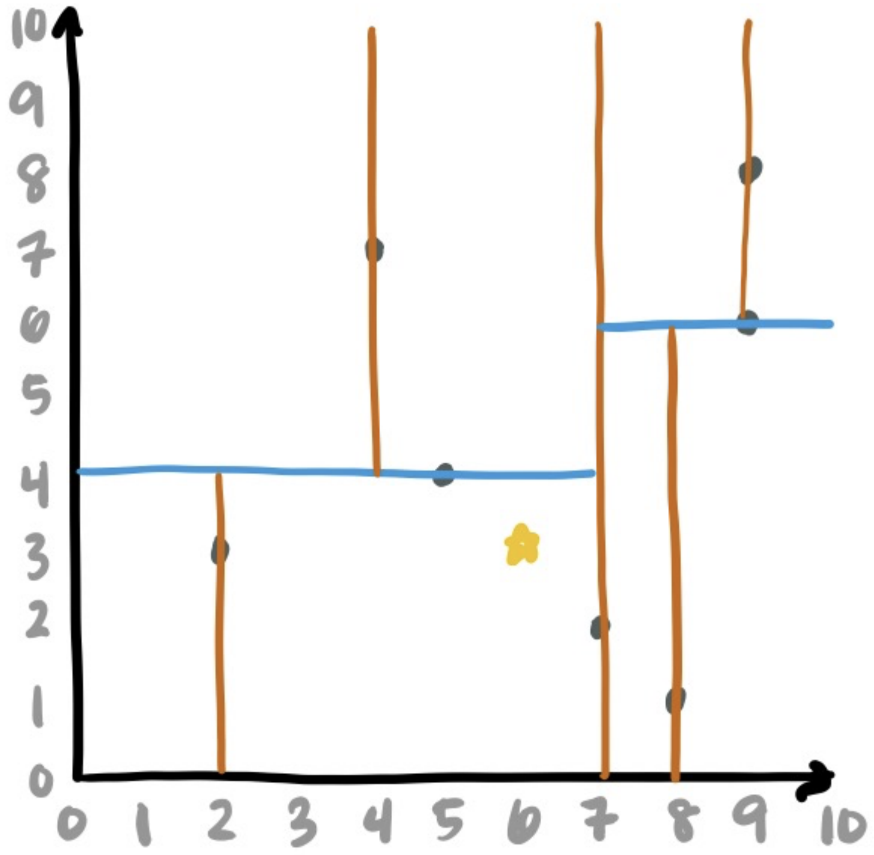




Nearest Neighbor: k-d tree

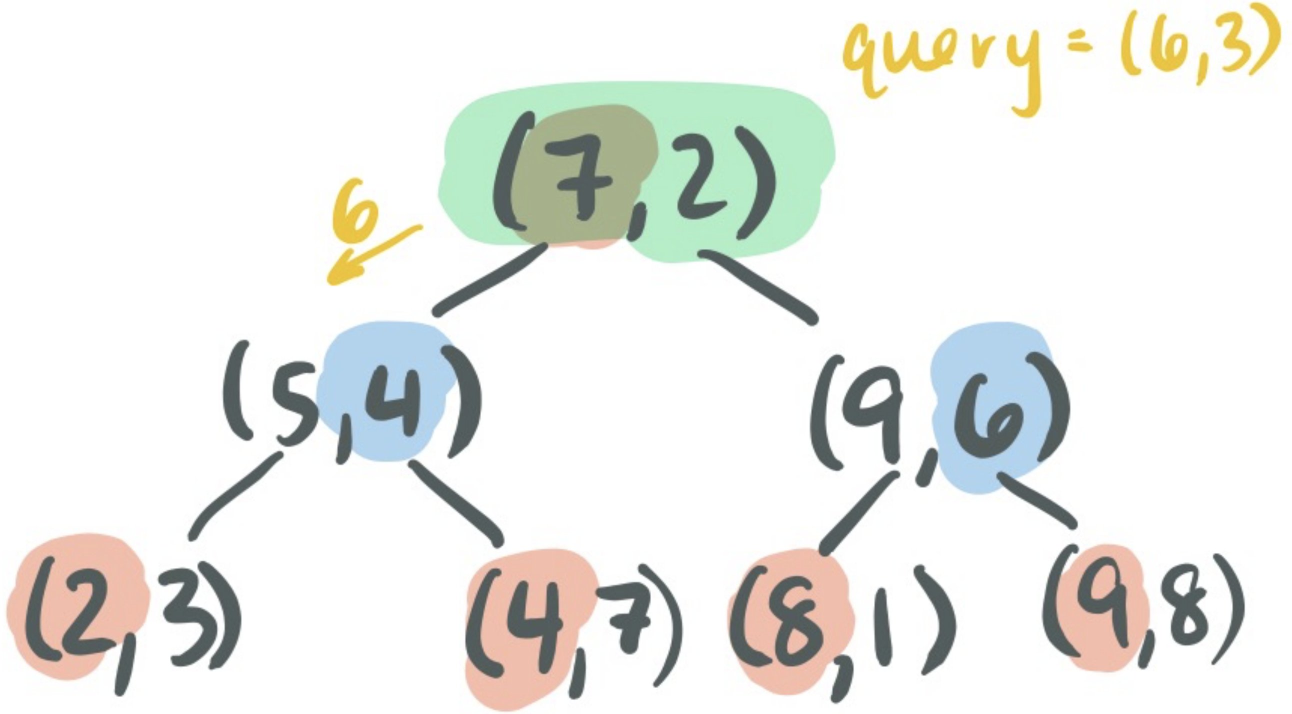
When querying a k-d tree, it acts like a BST* at first…

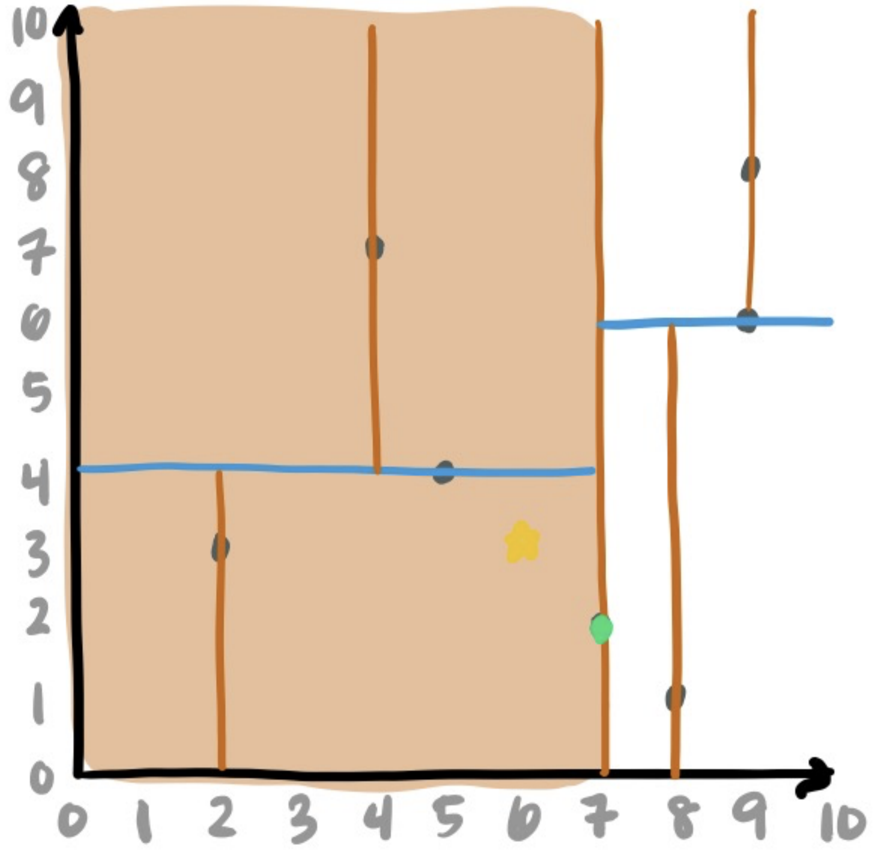




Nearest Neighbor: k-d tree

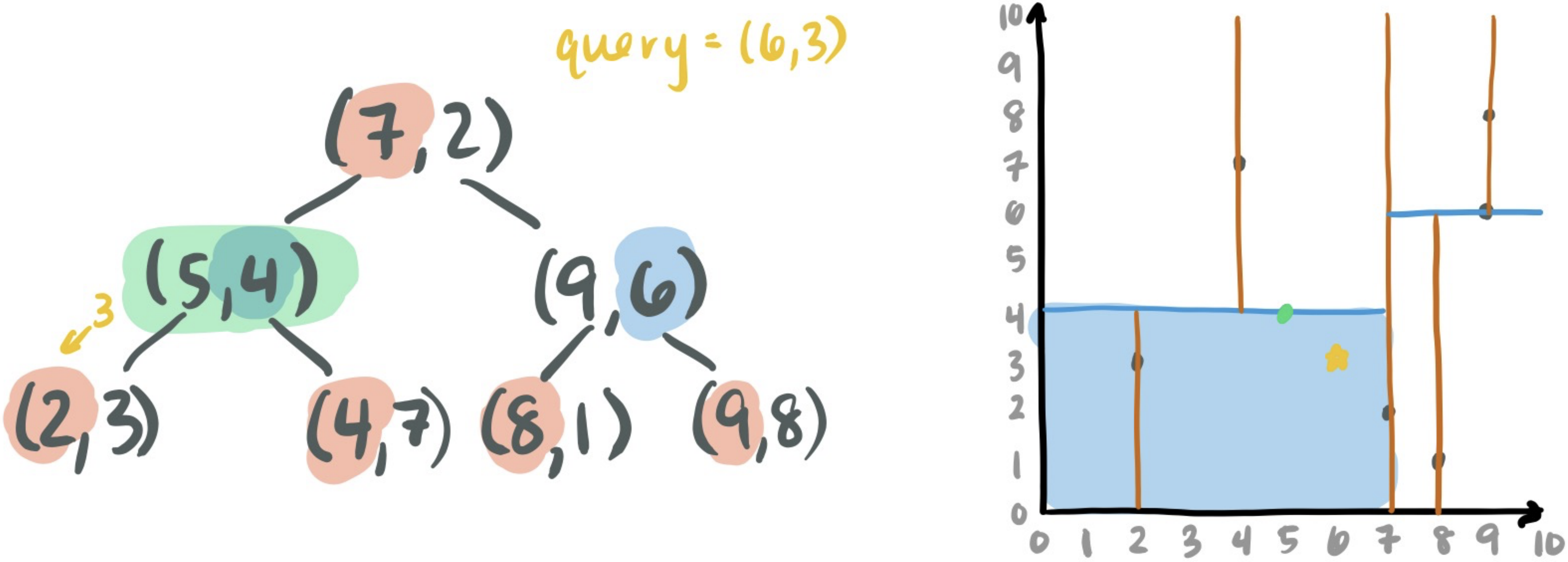
When querying a k-d tree, it acts like a BST* at first…





Nearest Neighbor: k-d tree

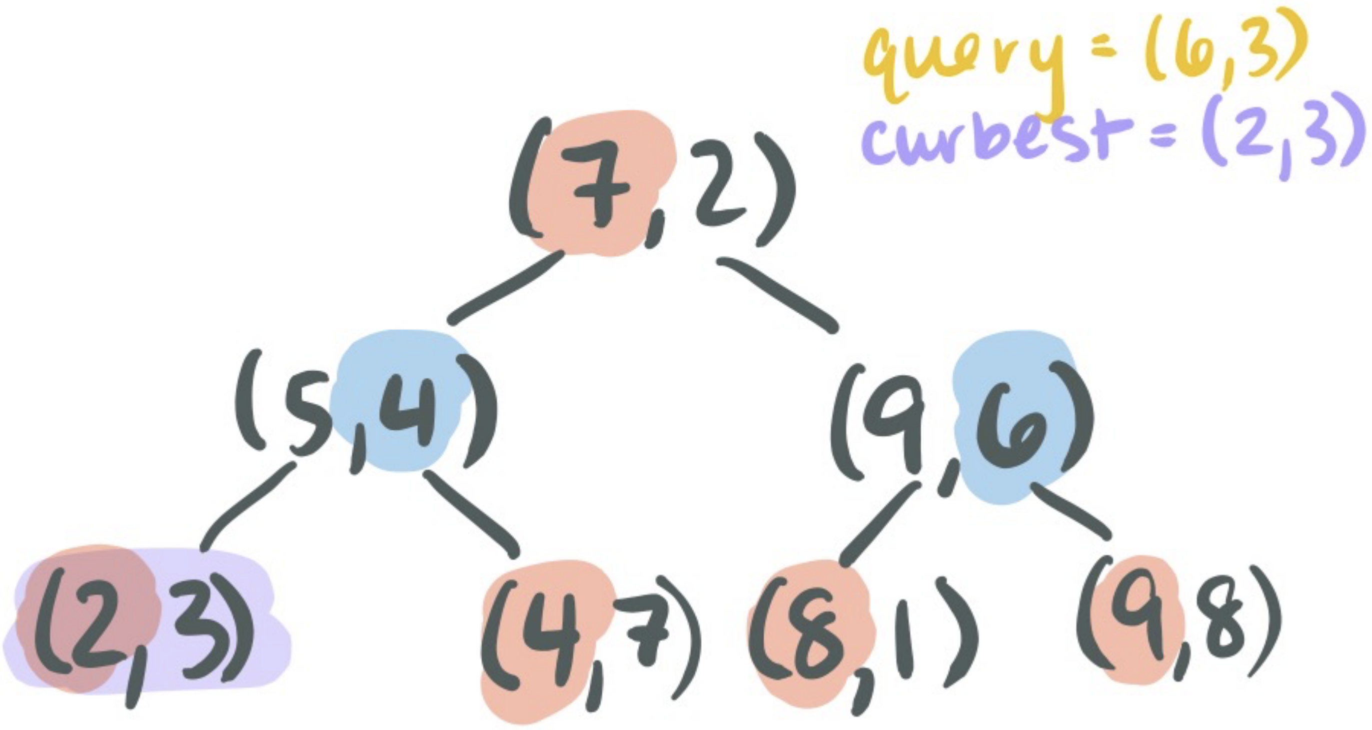
When querying a k-d tree, it acts like a BST* at first…

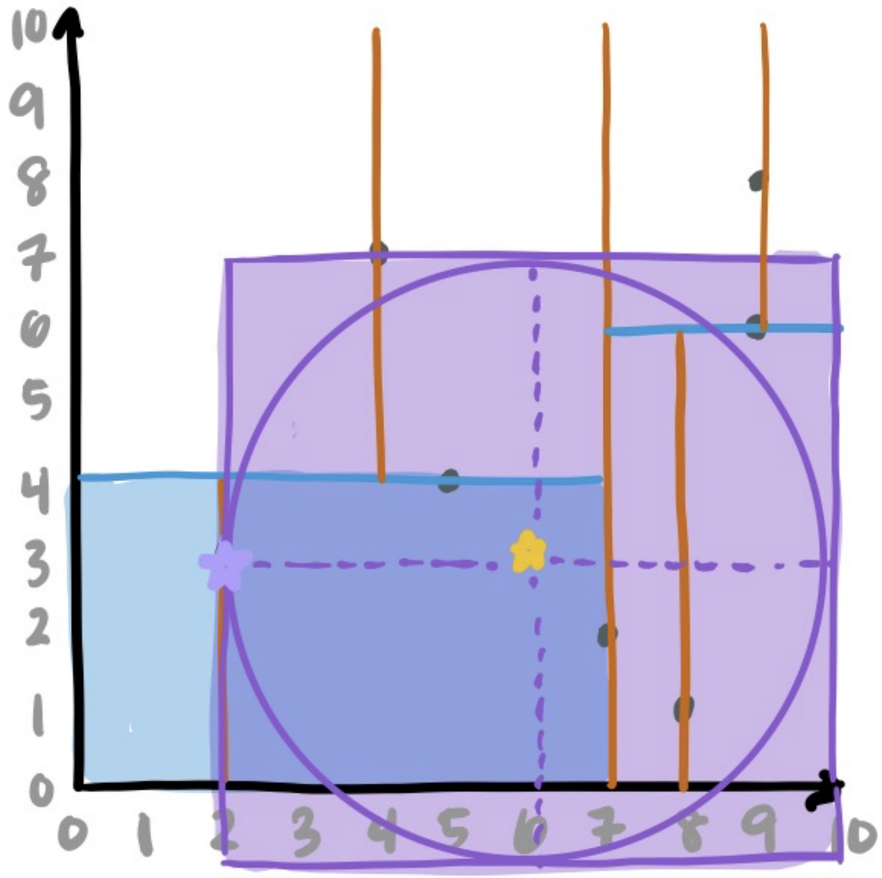


Nearest Neighbor: k-d tree

When querying a k-d tree, it acts like a BST* at first…

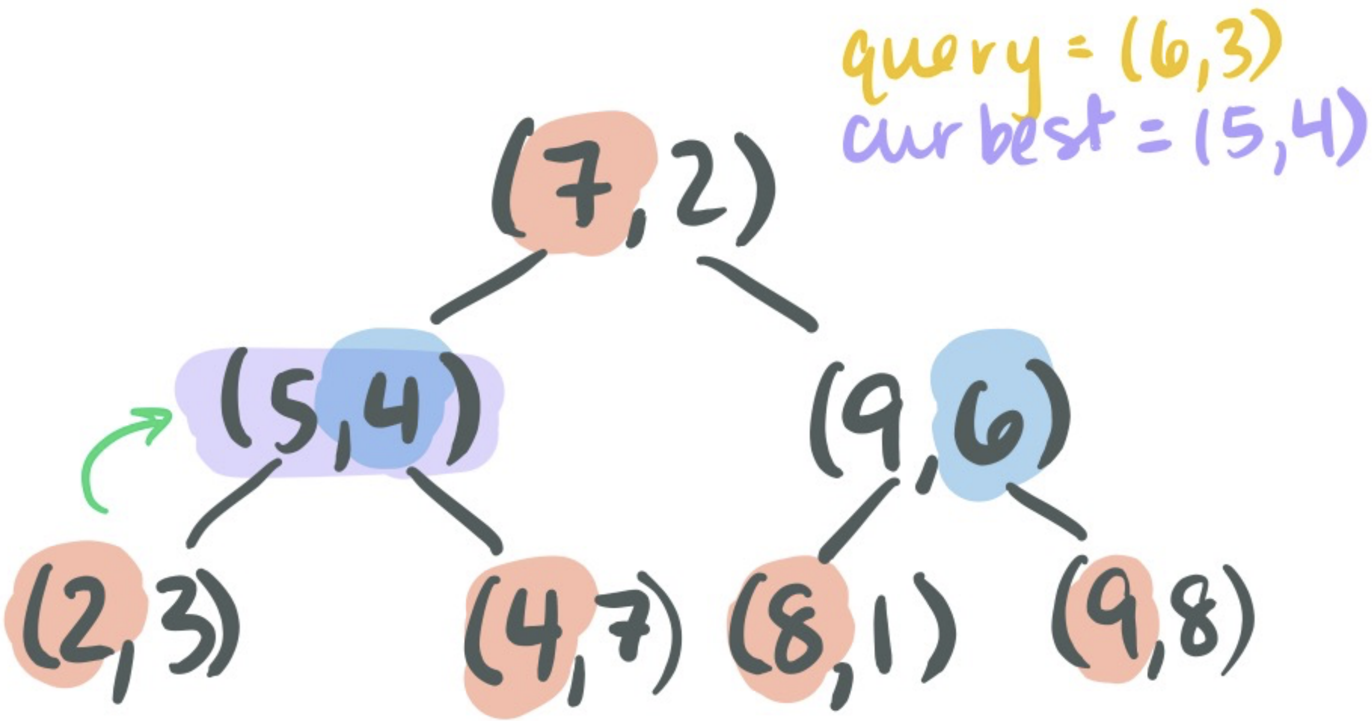
… But if we dont find exact match, have to find nearest neighbor

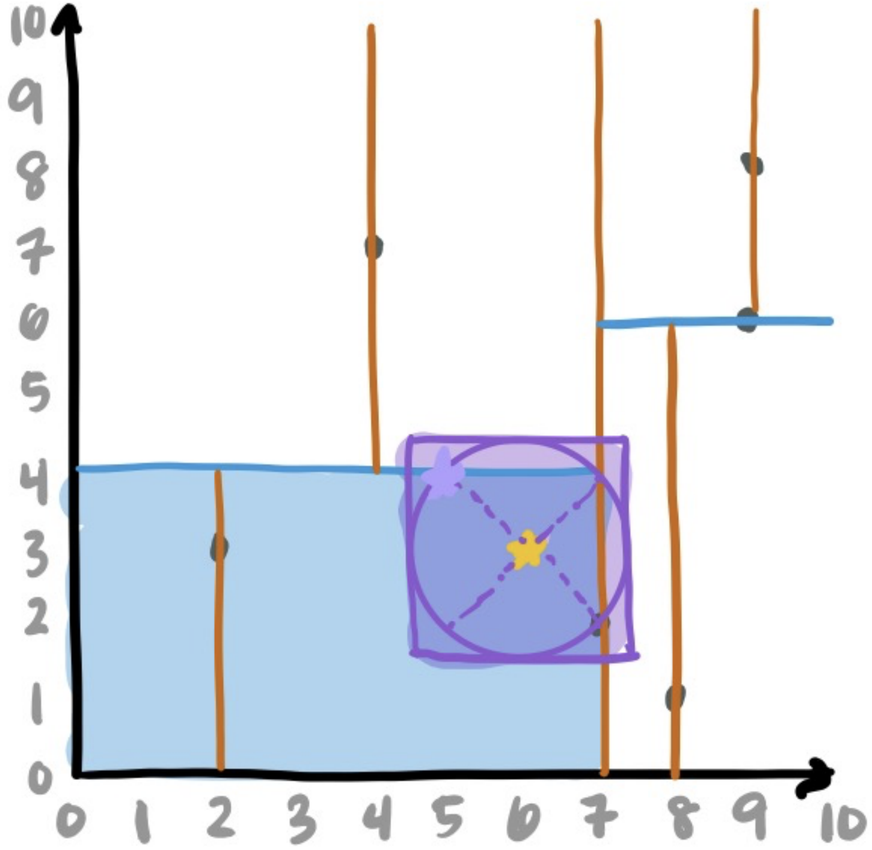




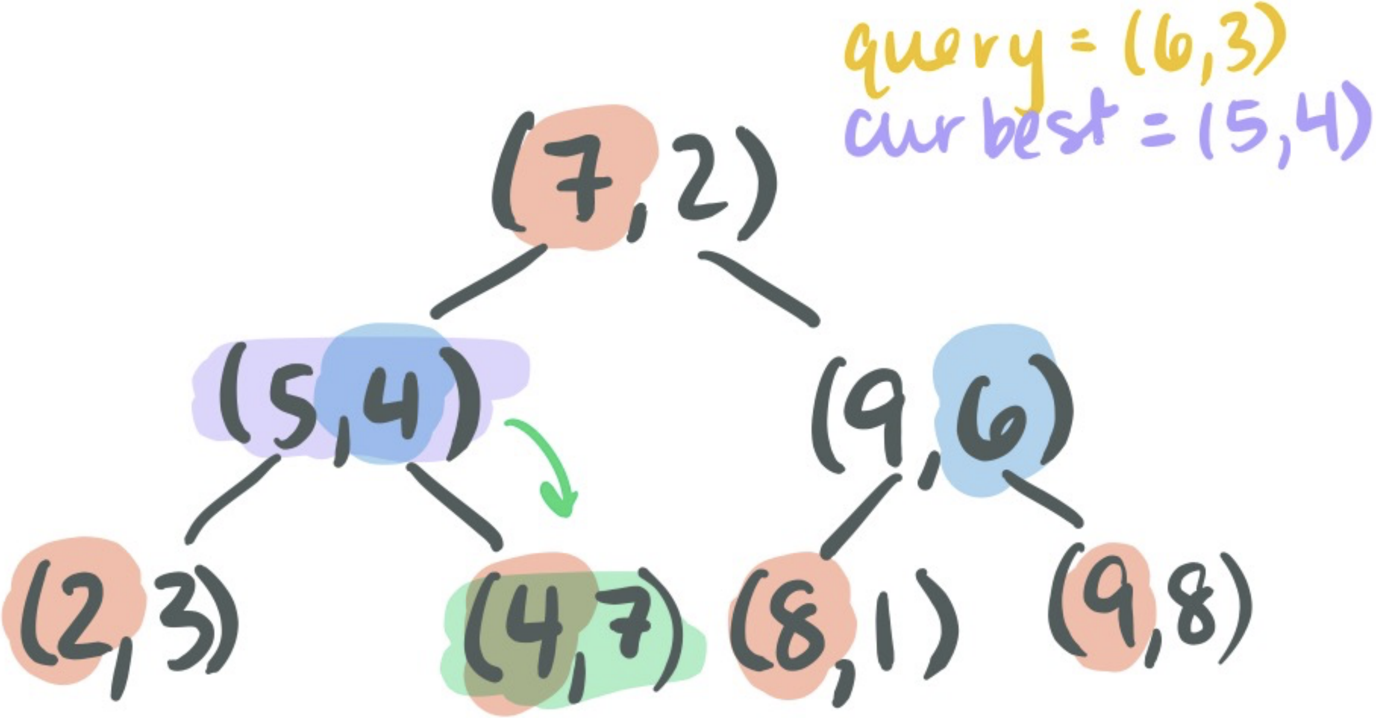
Nearest Neighbor: k-d tree

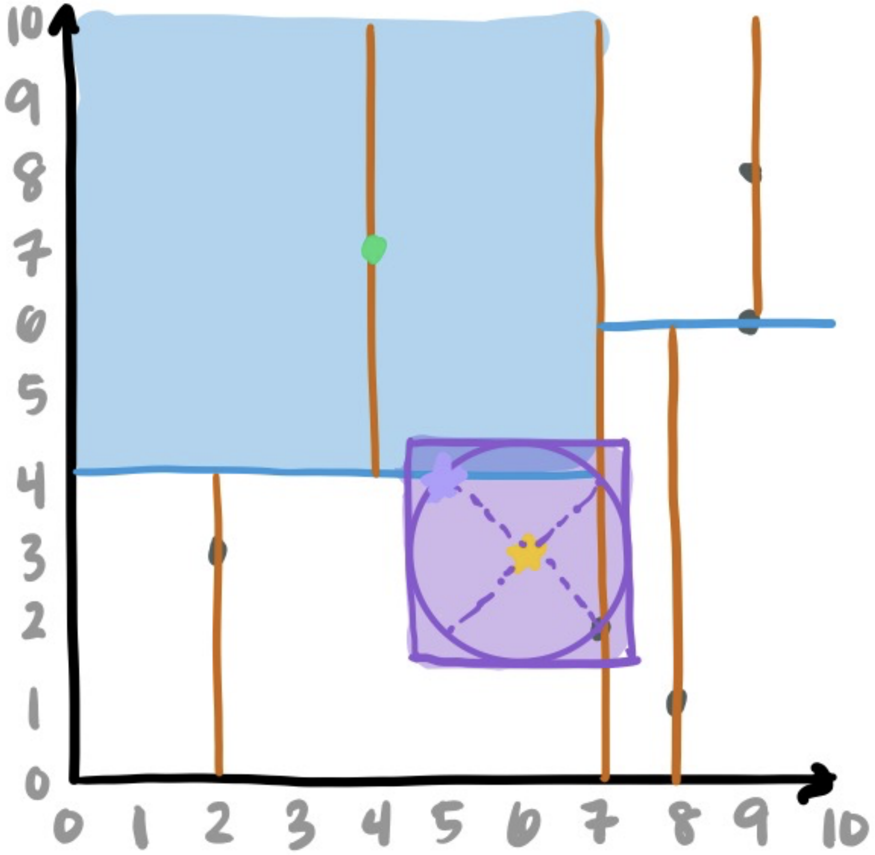
Backtracking: start recursing backwards -- store "best" possibility as you trace back





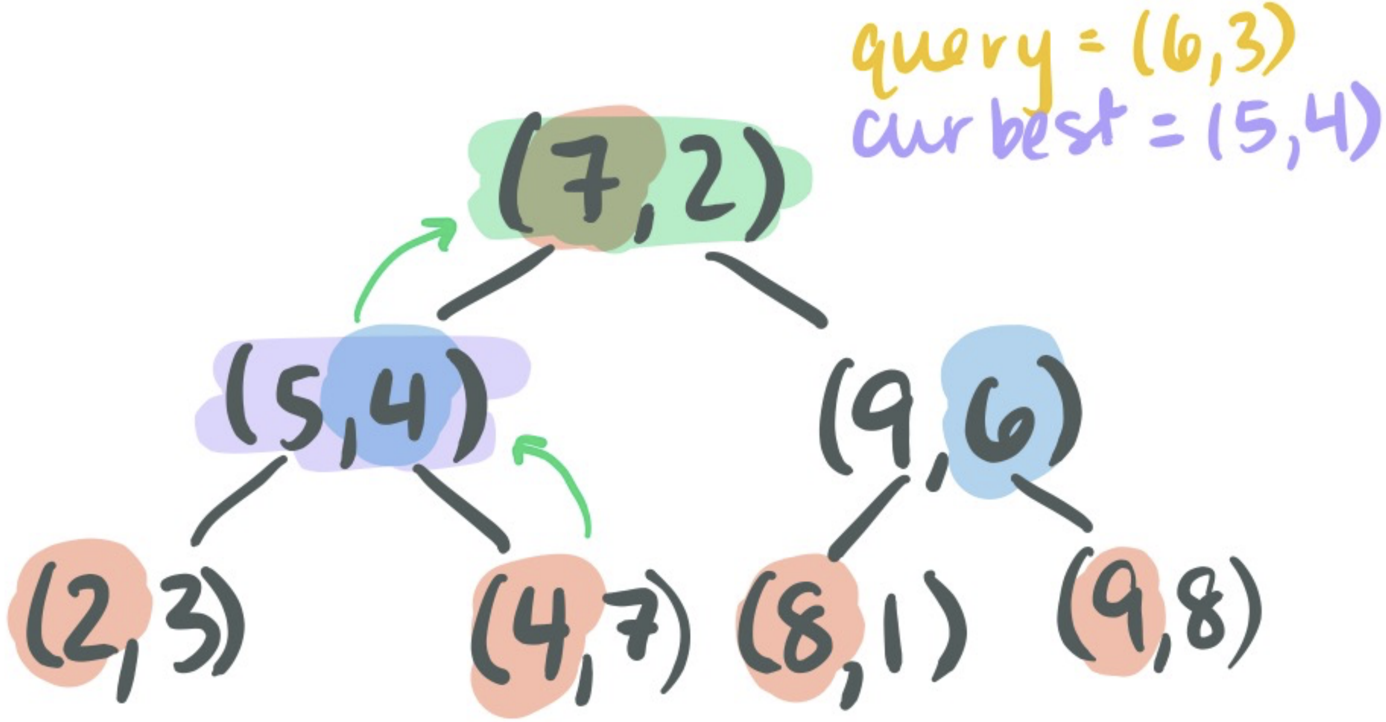
Nearest Neighbor: k-d tree

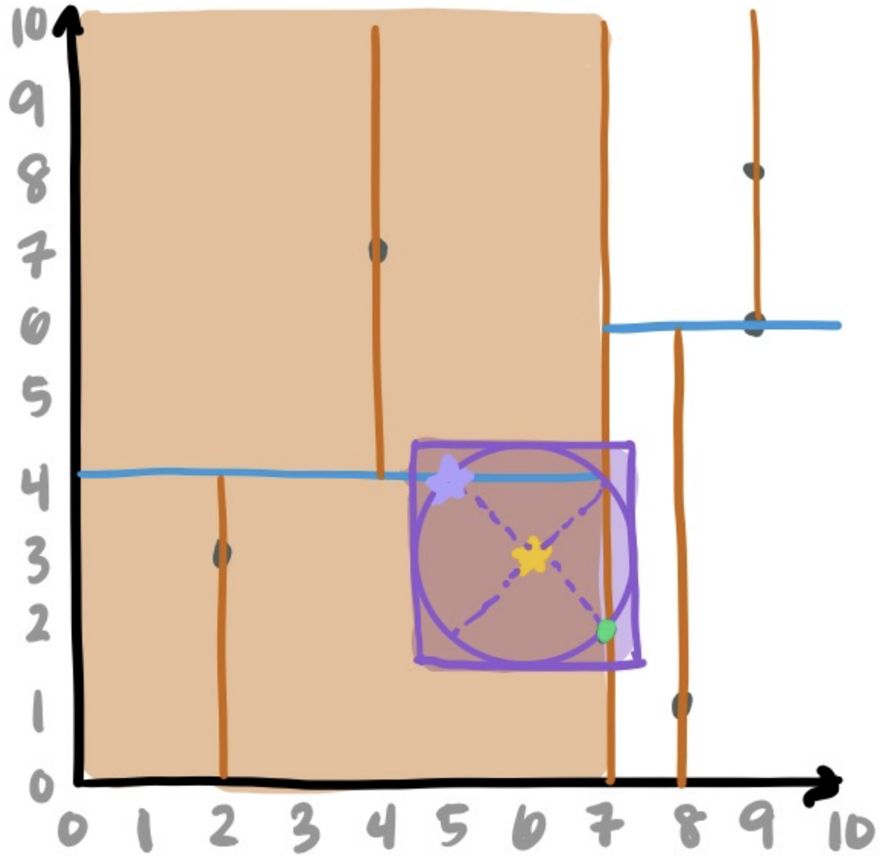




Nearest Neighbor: k-d tree

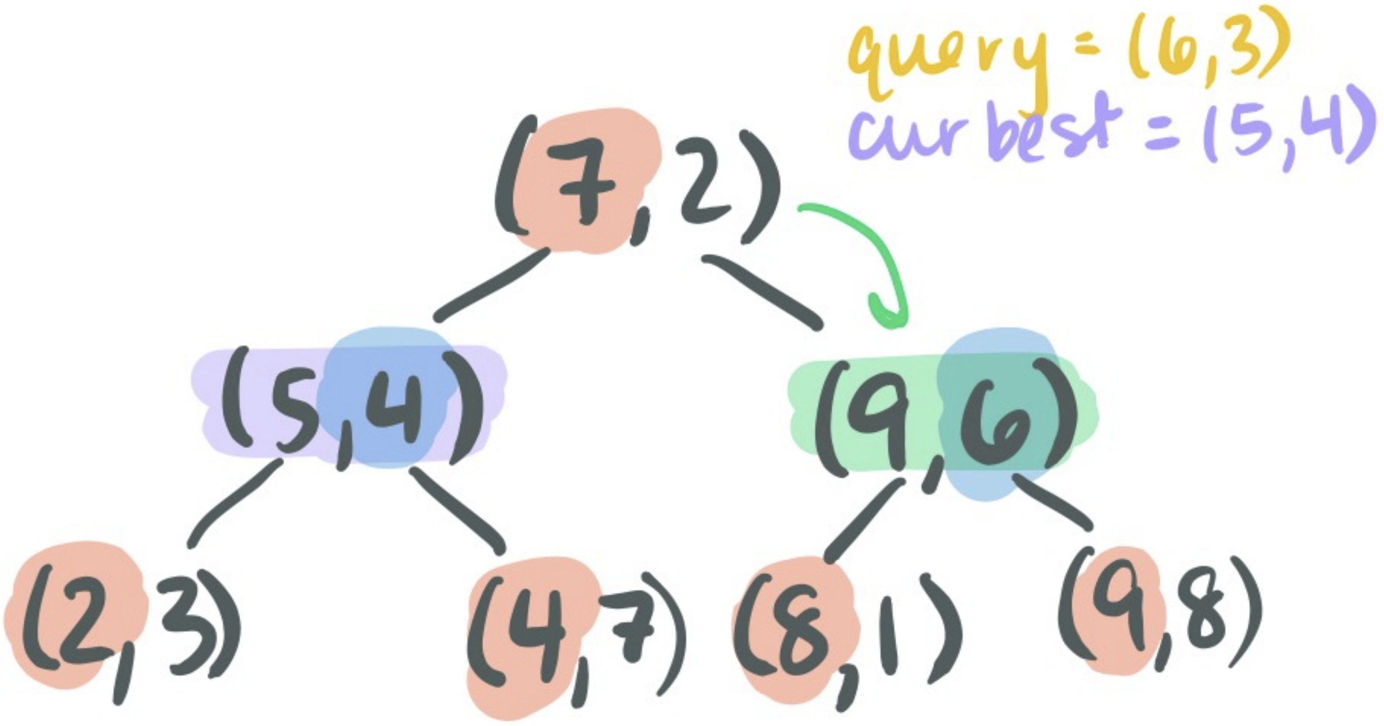
On ties, use smallerDimVal to determine which point remains curBest

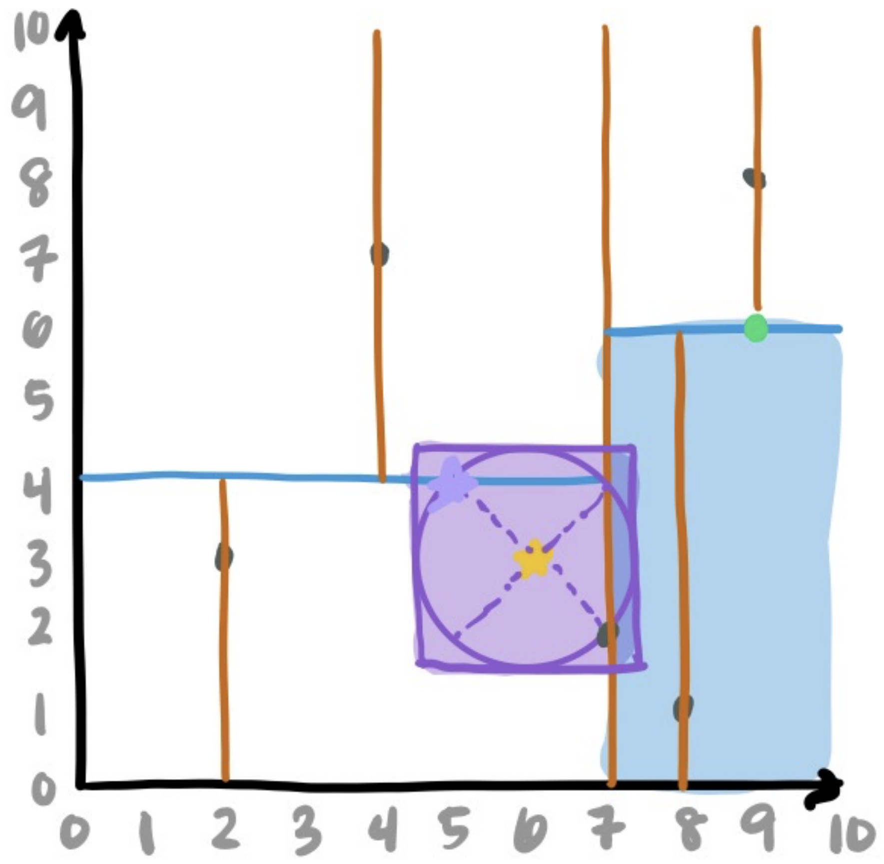




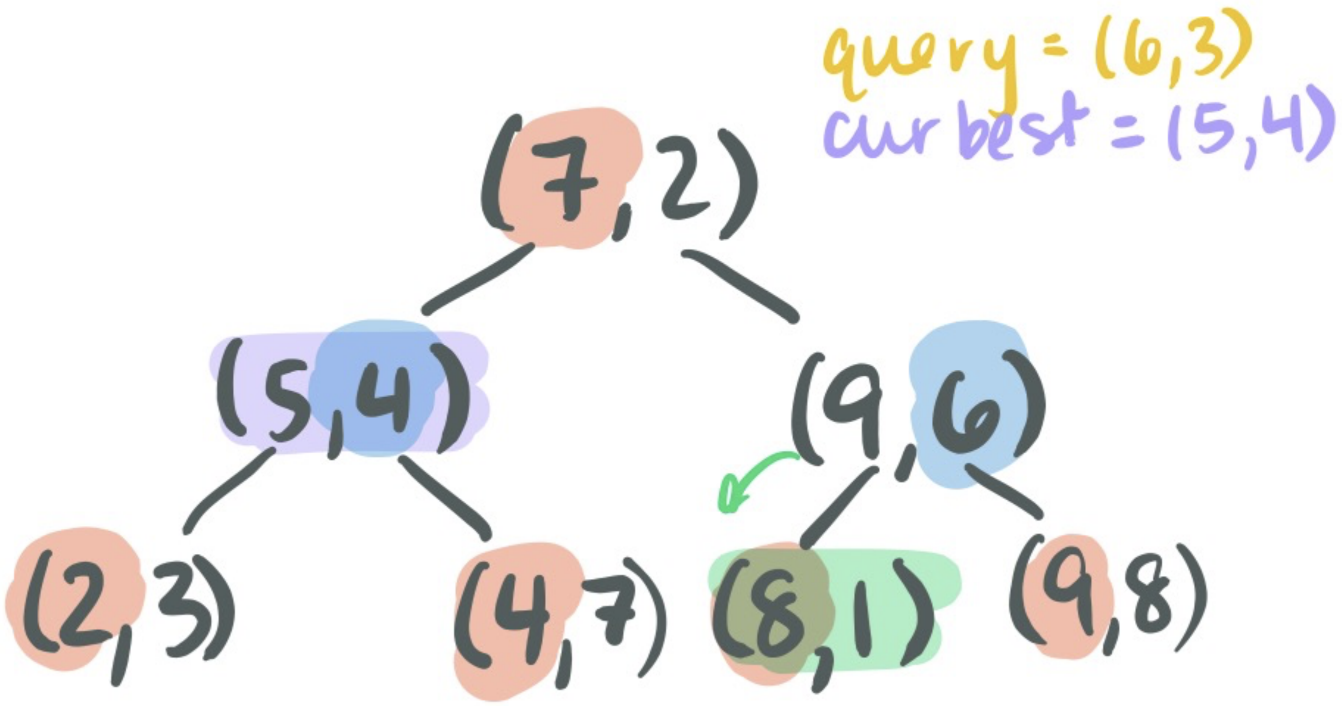
Nearest Neighbor: k-d tree

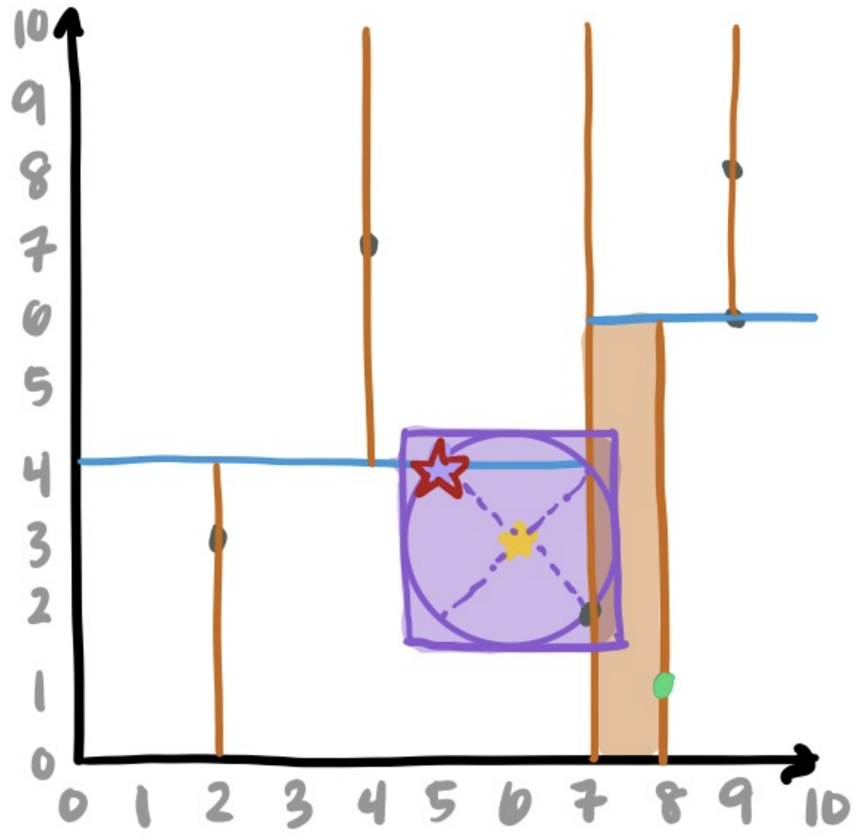
Why do we need to explore this subtree?





Nearest Neighbor: k-d tree

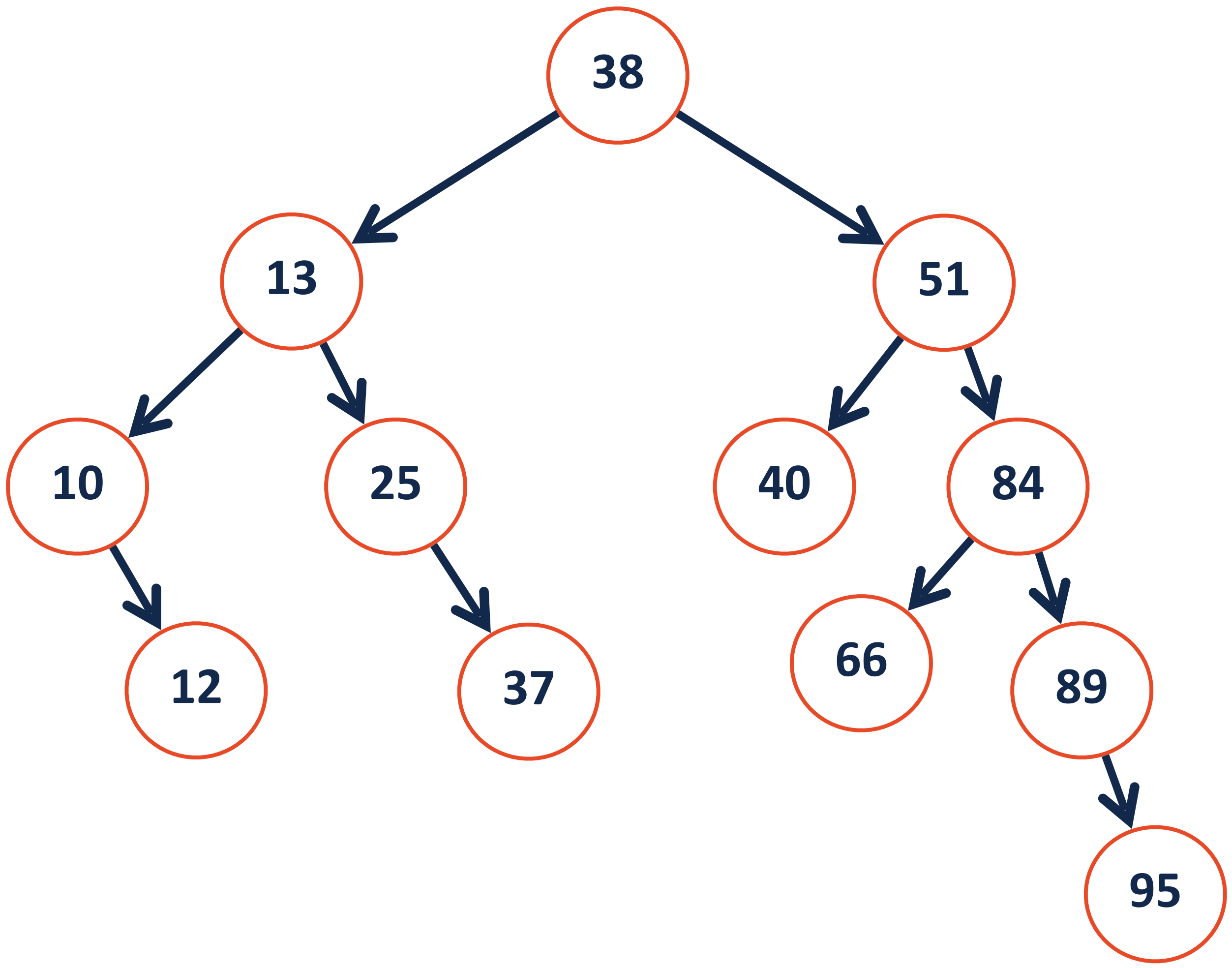




BEST: (5,4)

BST Analysis – Running Time

Operation	BST Worst Case
find	O(h)
insert	O(h)
remove	O(h)
traverse	O(n)



BST Analysis

Every operation on a BST depends on the **height** of the tree.

… how do we relate $O(h)$ to n, the size of our dataset?

Quiz

What is the range of number of nodes in a binary tree of height h ?

1. $(h, 2^h)$

2. $(h+1, 2^h)$

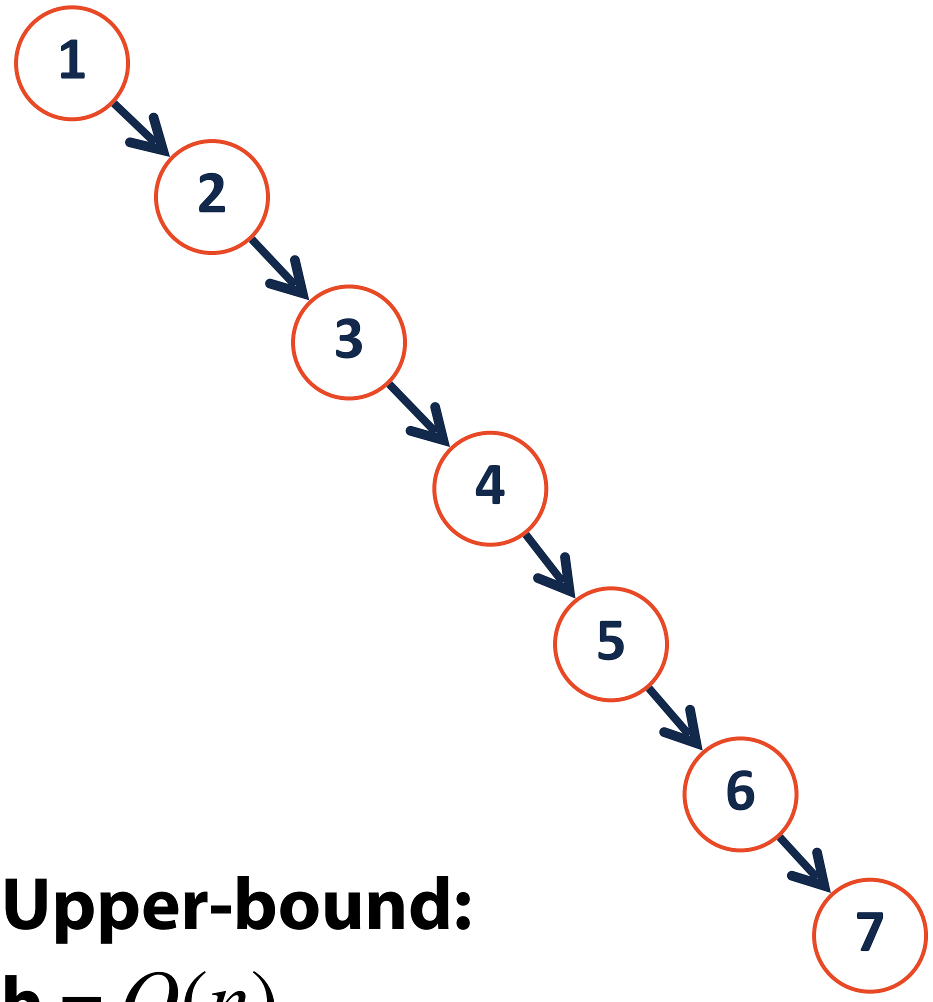
3. $(h+1, 2^{h+1})$

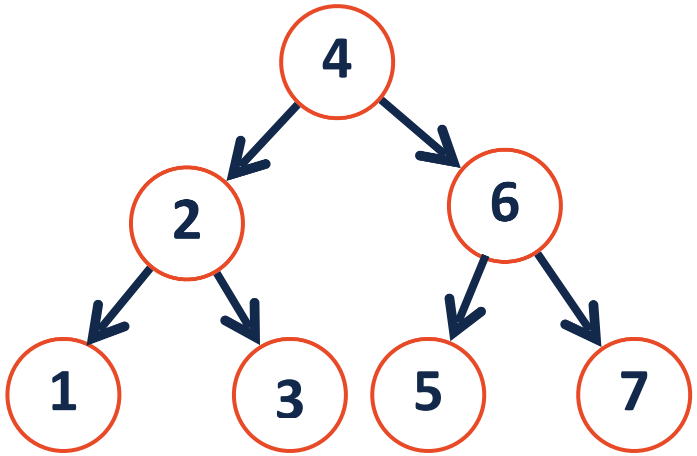
4. $(h+1, 2^{h+1} - 1)$

5. $(h+1, 2^h - 1)$

BST Analysis

A BST of n nodes has a height between:





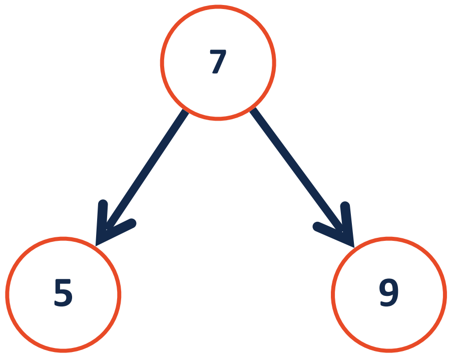
Lower-bound:
h = $\Omega(\log n)$

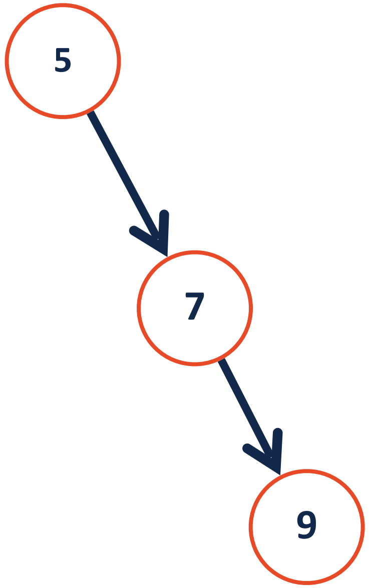
Upper-bound:
h = $O(n)$

Height-Balanced Tree

What tree is better?

b = 0





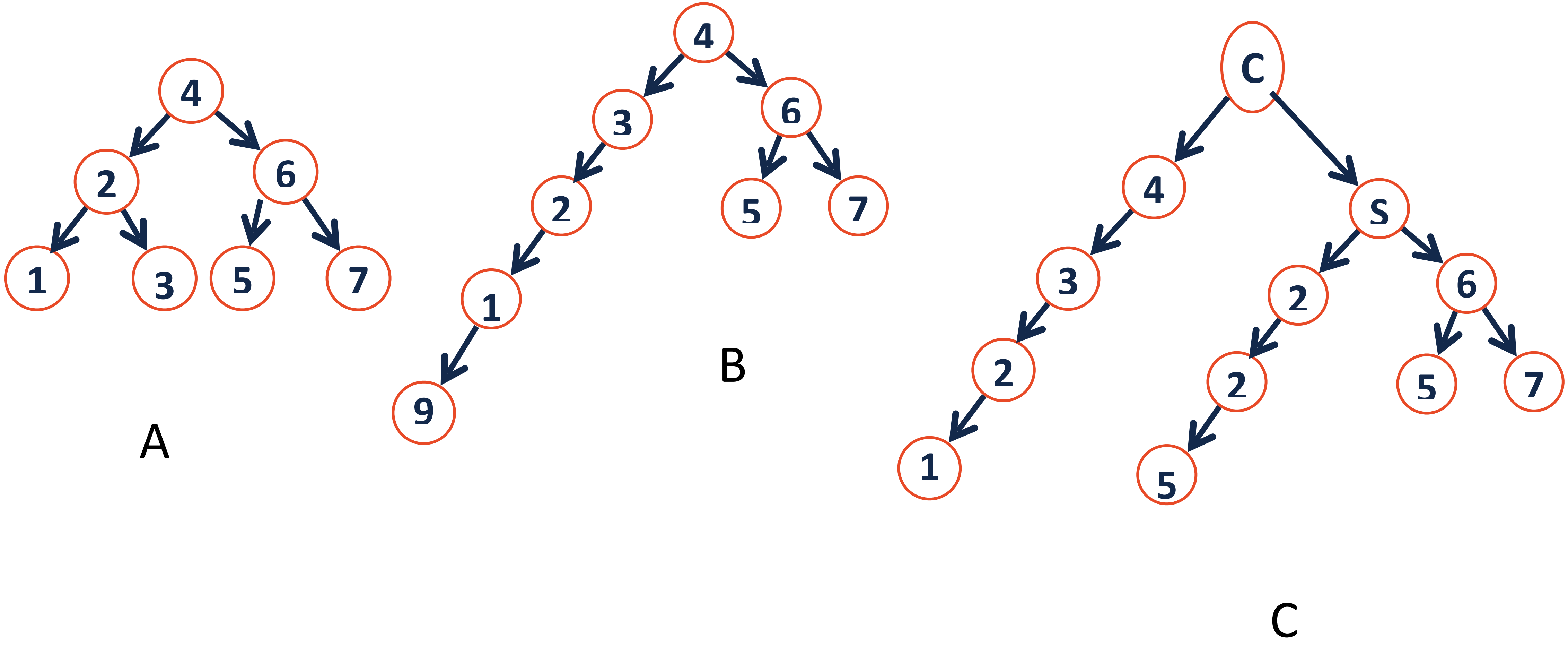
b = 2

Height balance: $b = height(T_R) - height(T_L)$

A tree is "balanced" if: |b| <= 1 **for every node**

Quiz

Which of the following binary trees are balanced?



Option A: Correcting bad insert order

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

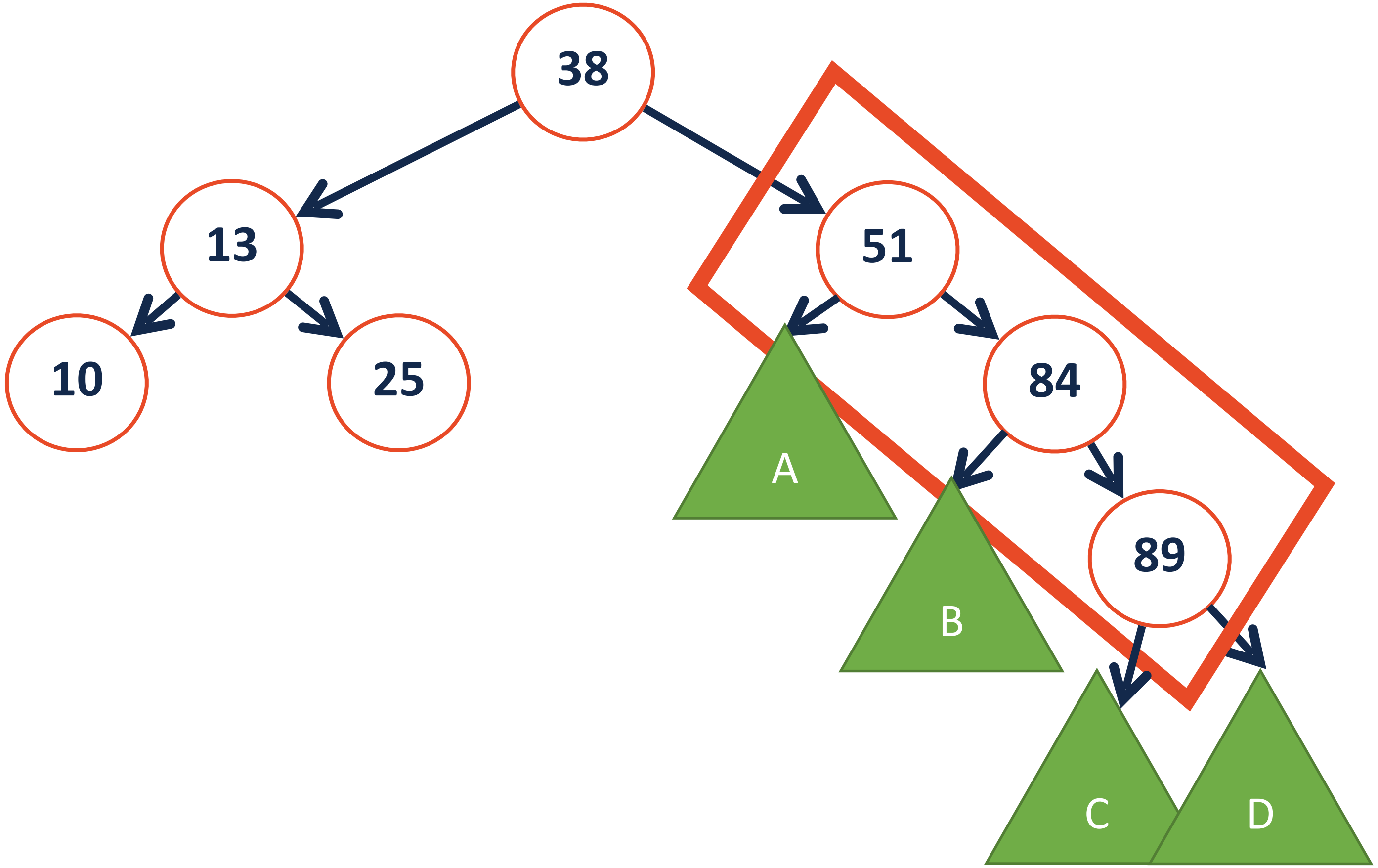
Insert Order: [1, 2, 3, 4, 5, 6, 7] BAD

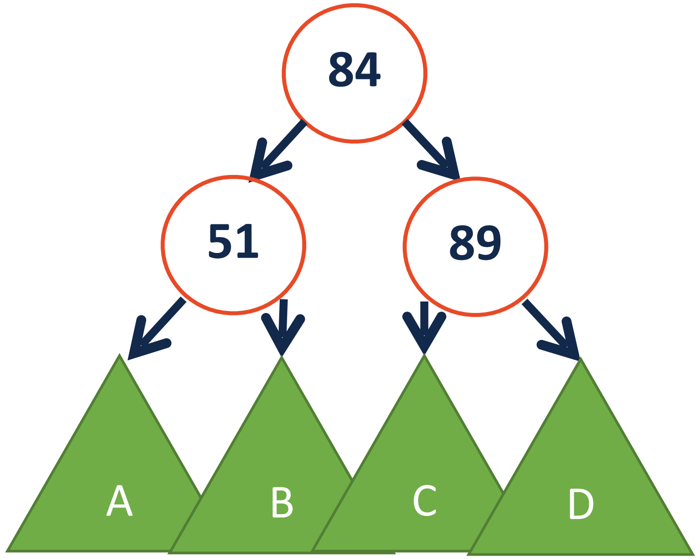
Insert Order: [4, 6, 2, 3, 7, 1, 5] GOOD

Random Order : Mostly GOOD, sometimes BAD - as you've seen in lab_BST

AVL-Tree: A self-balancing binary search tree

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





Adelson **V**elsky,
Evgenii **L**andis 1962

BST Rotations (The AVL Tree)

We can adjust the BST structure by performing **rotations**.

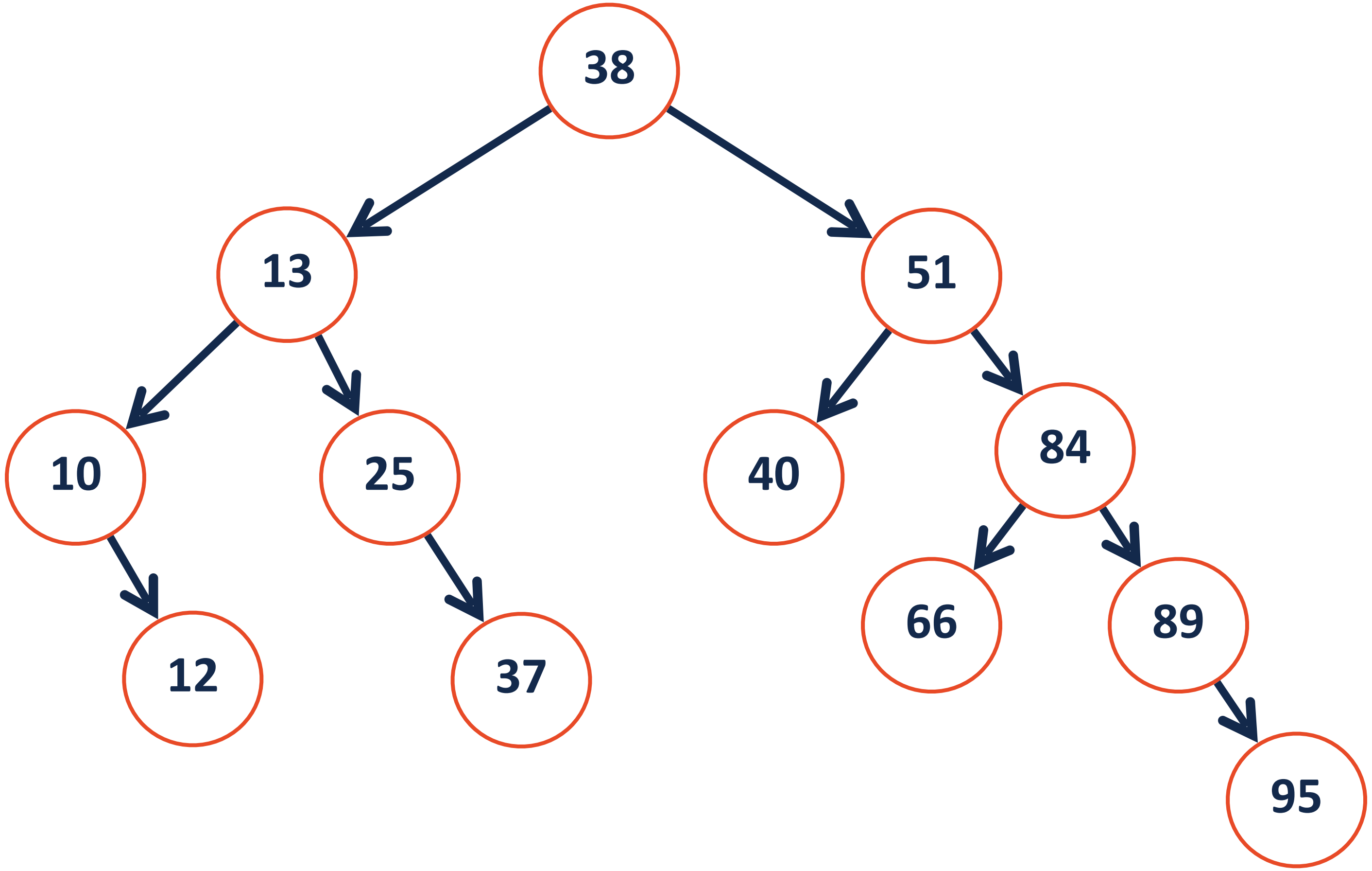
These rotations, when used correctly:

1. Modify the arrangement of nodes while preserving BST property

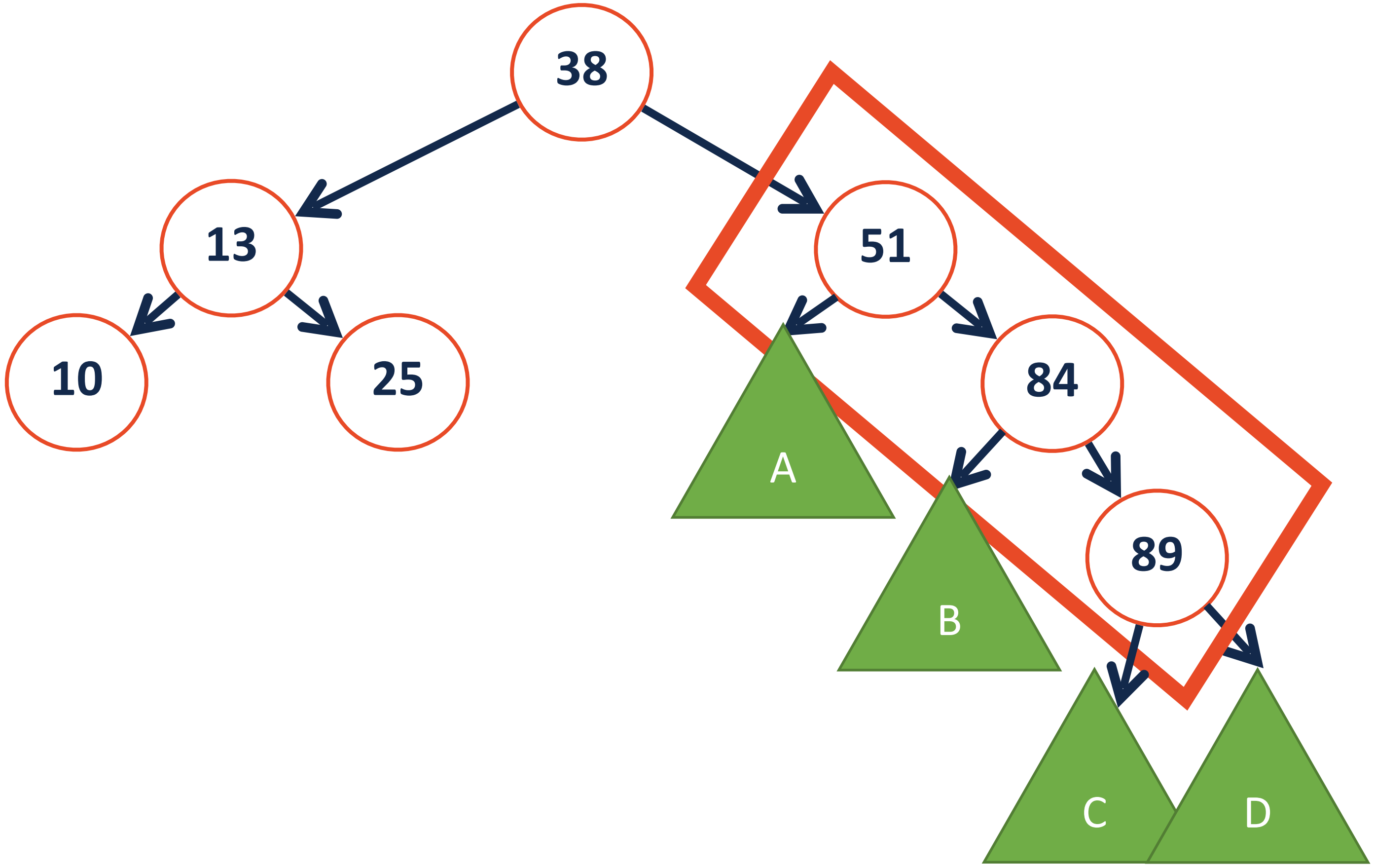
2. Reduce tree height by one

BST Rotations (The AVL Tree)

To begin, lets find the imbalance in the following tree:

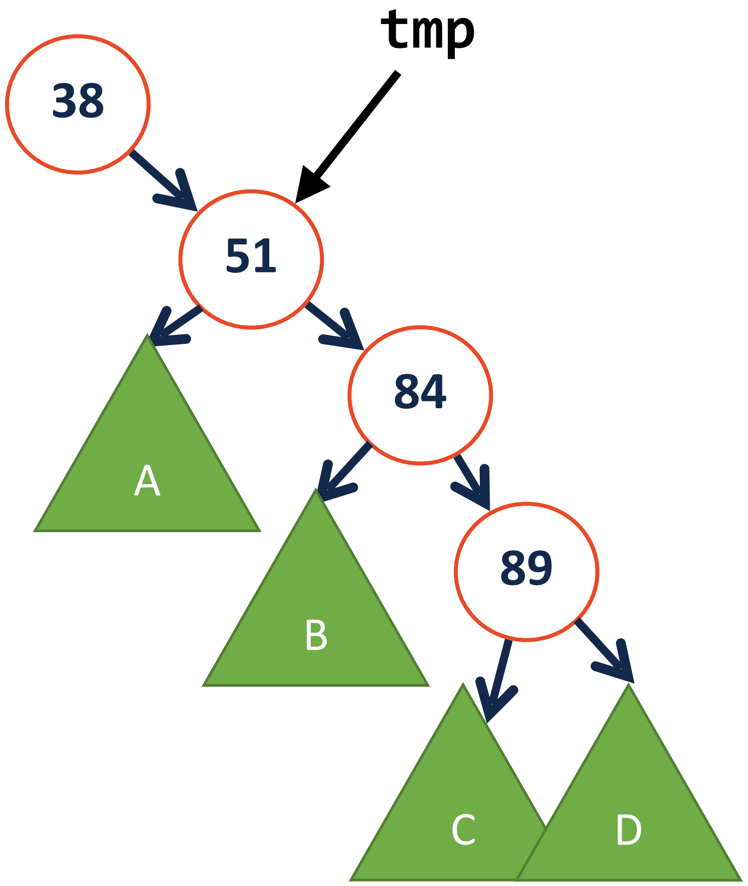


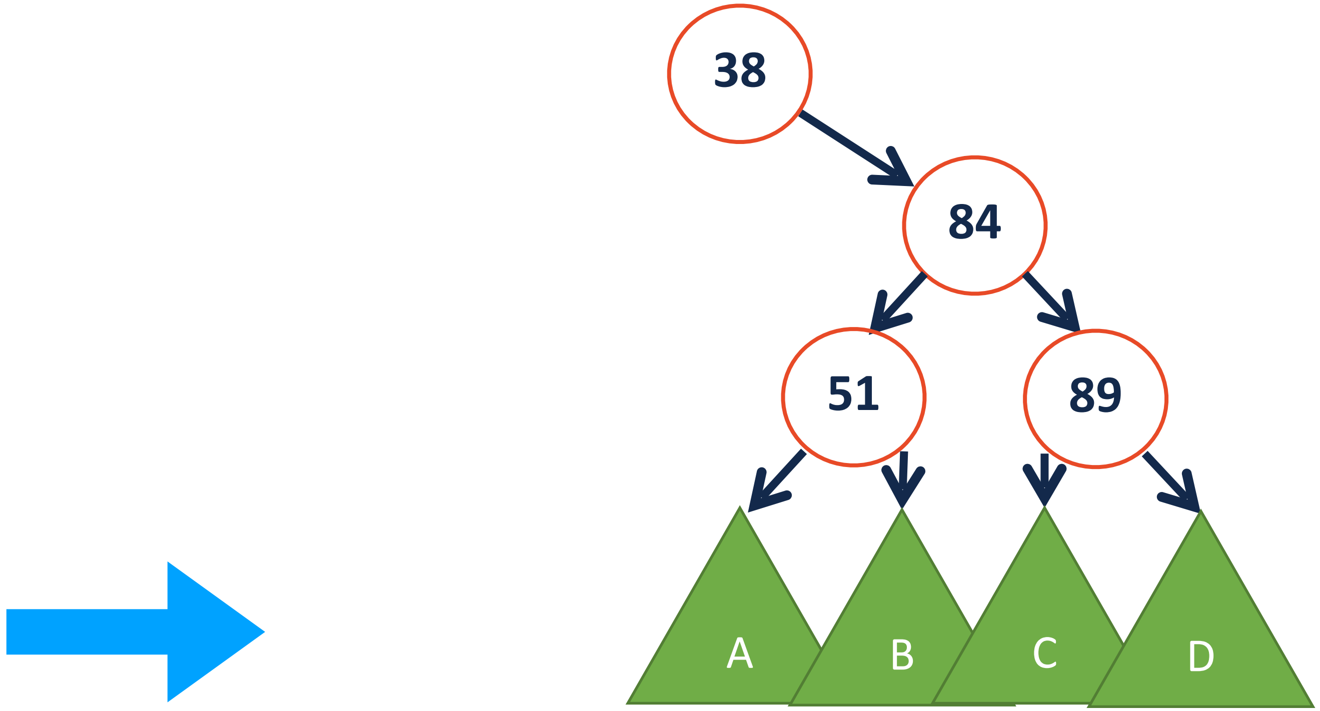
Left Rotation



Left Rotation

1) Create a tmp pointer to root

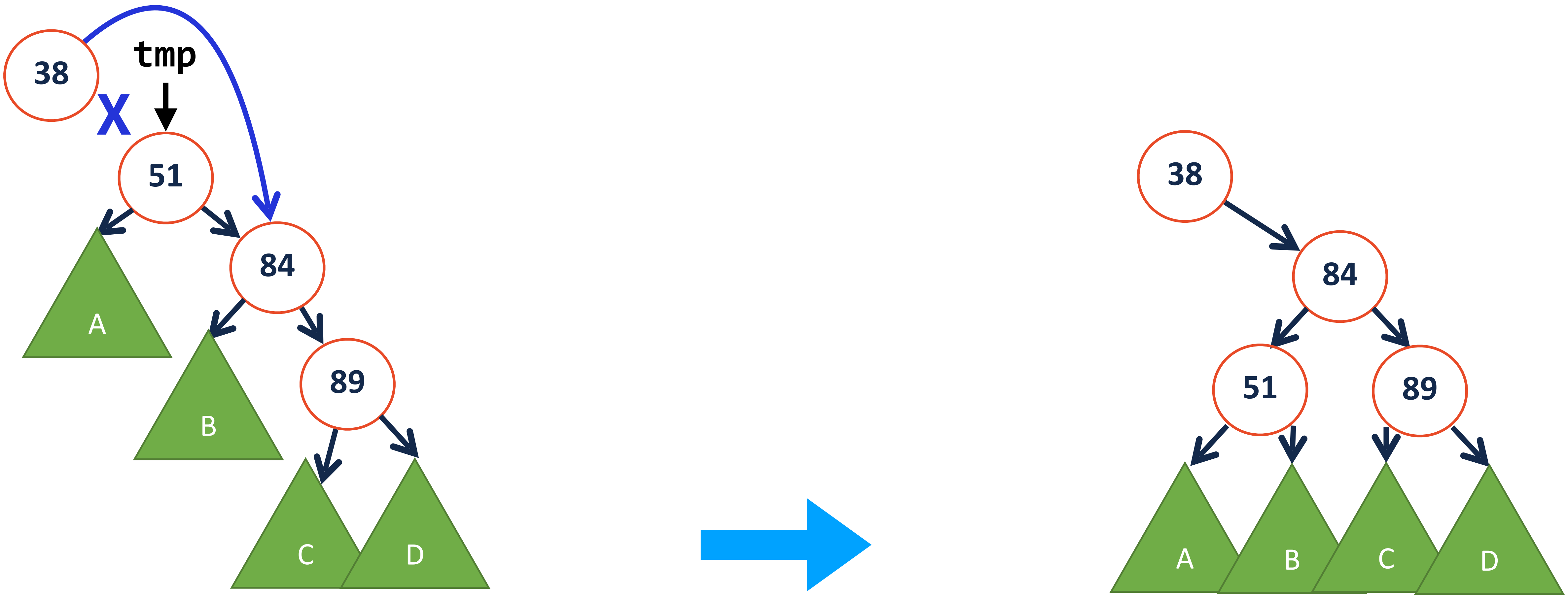




Left Rotation

1) Create a tmp pointer to root

2) Update root to point to mid

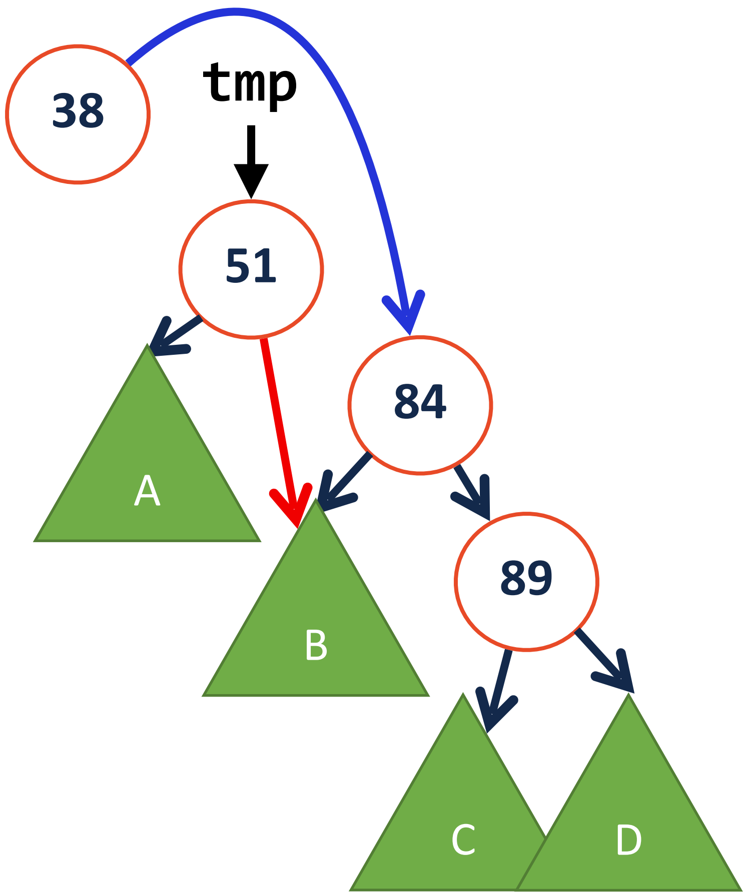


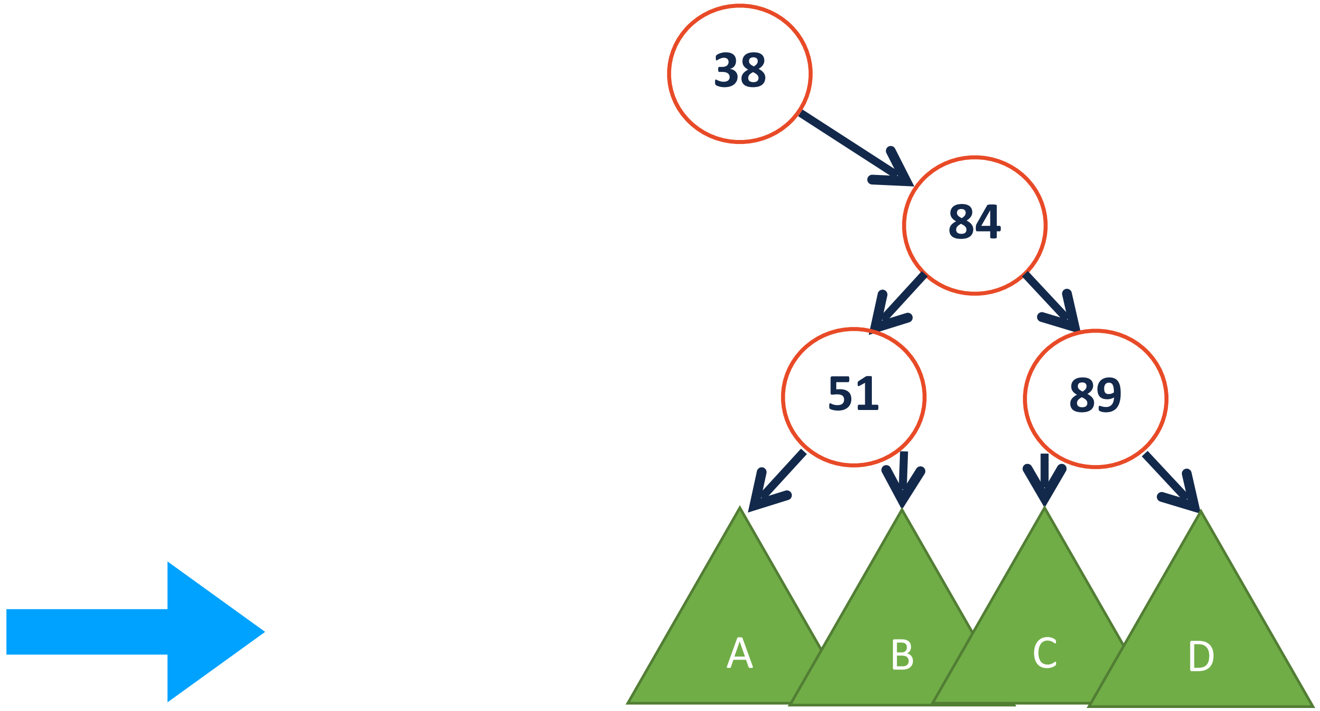
Left Rotation

1) Create a tmp pointer to root

2) Update root to point to mid

3) tmp->right = root->left

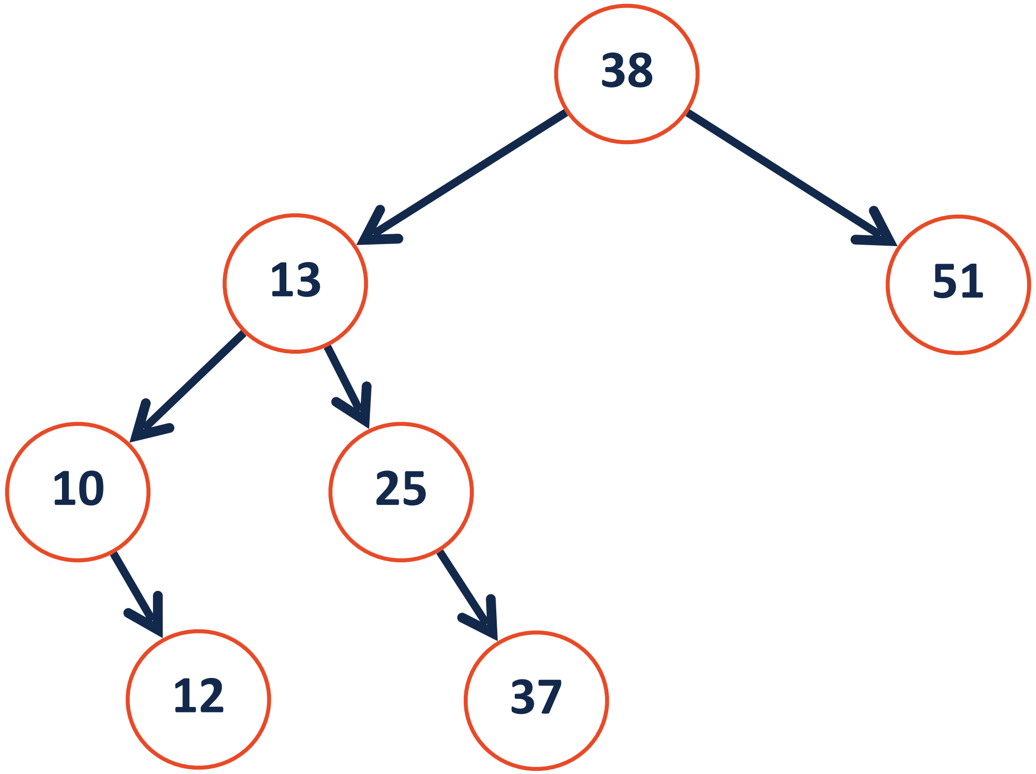




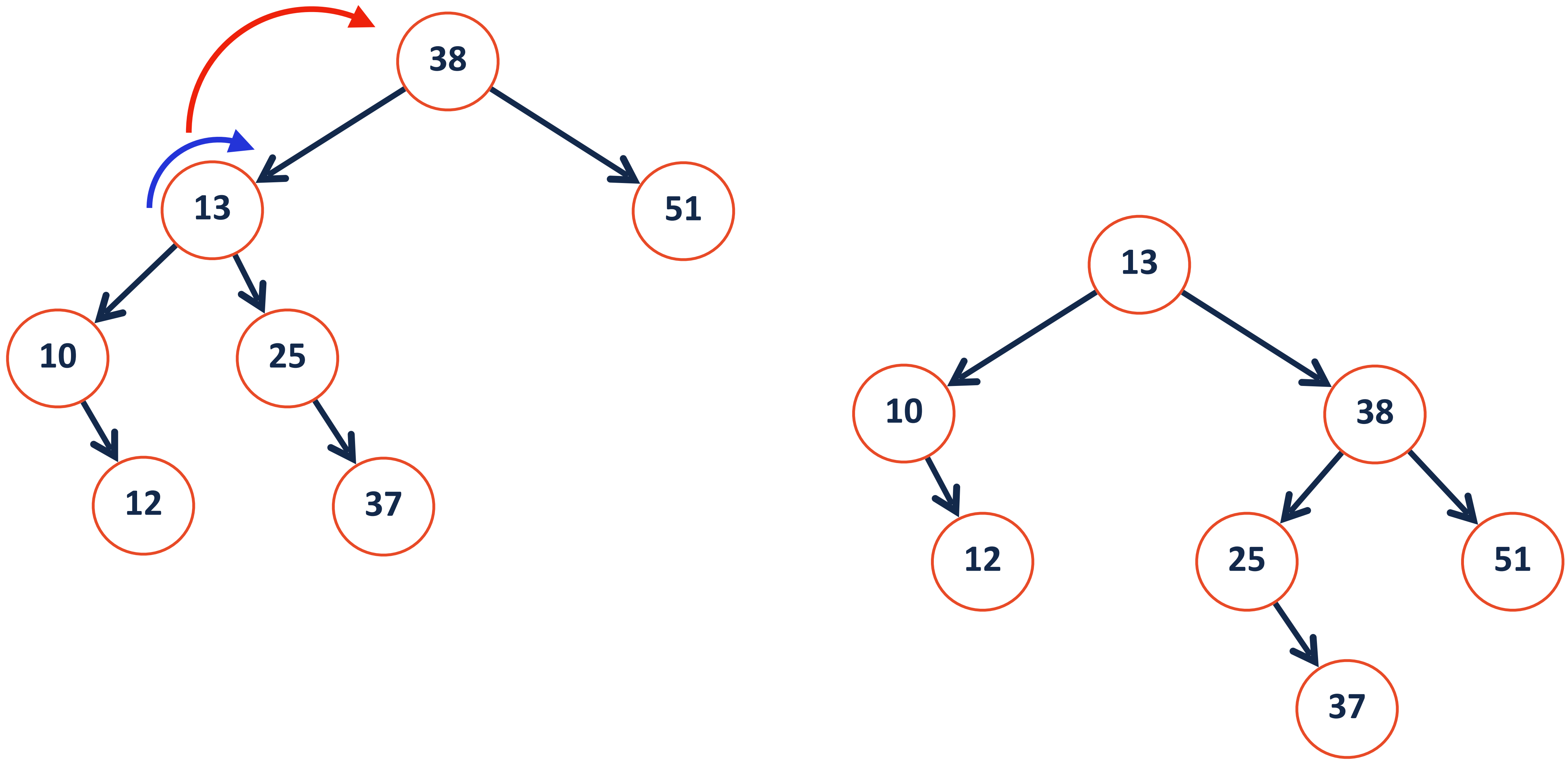
Left Rotation

1) Create a tmp pointer to root
2) Update root to point to mid
3) tmp->right = root->left
4) root->left = tmp

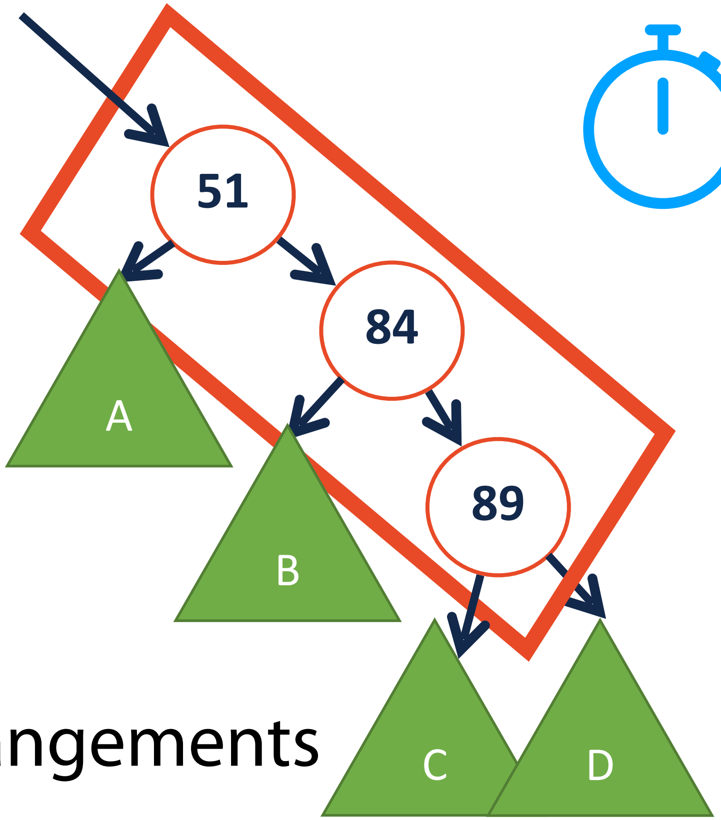
Right Rotation



Right Rotation



Coding AVL Rotations



Two ways of visualizing:

1) Think of an arrow 'rotating' around the center

2) Recognize that there's a concrete order for rearrangements

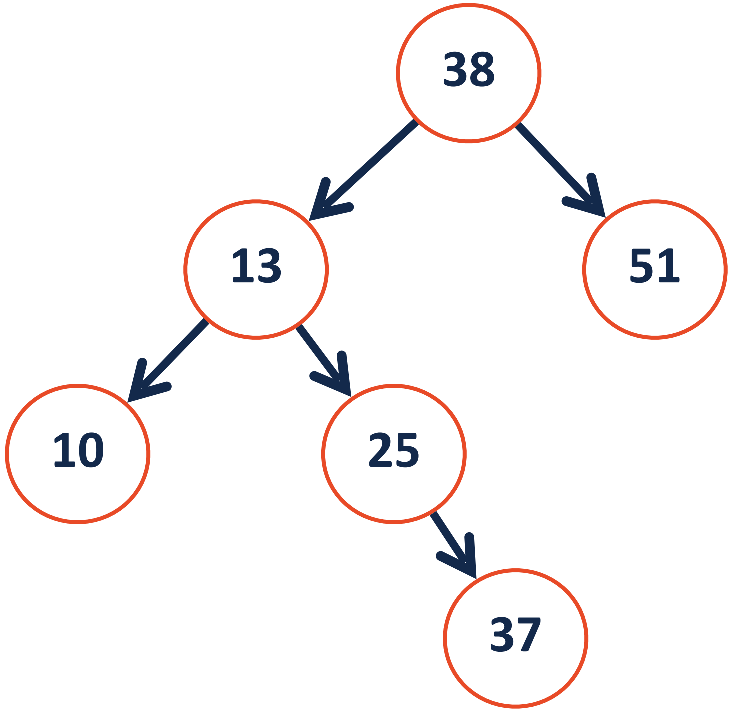
Ex: Unbalanced at current (root) node and need to *rotateLeft*?

Replace current (root) node with it's right child.

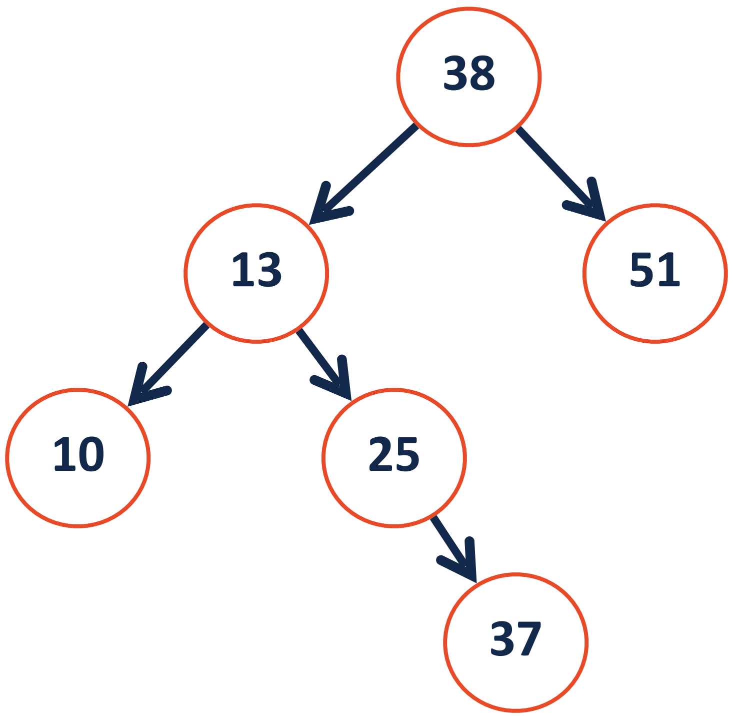
Set the right child's left child to be the current node's right

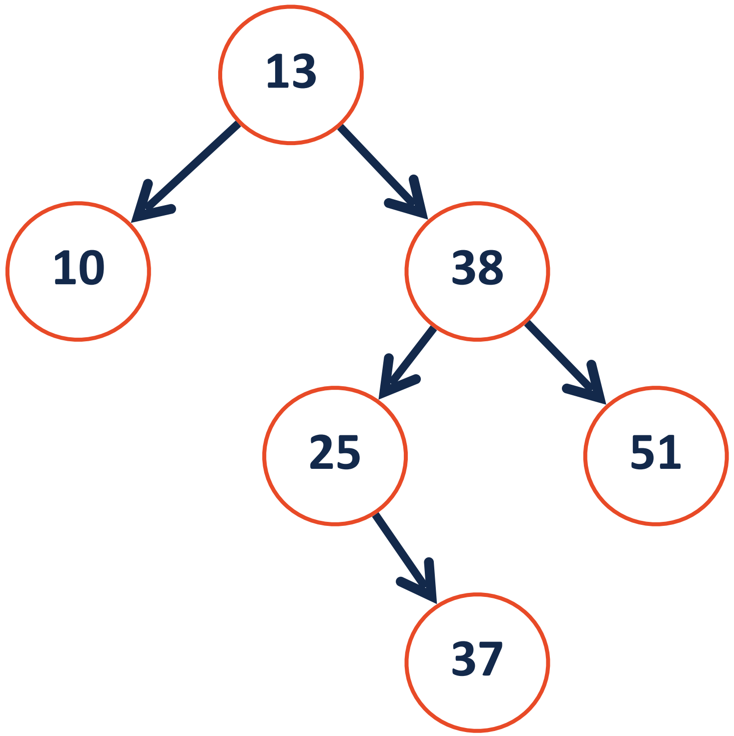
Make the current node the right child's left child

AVL Rotation Practice



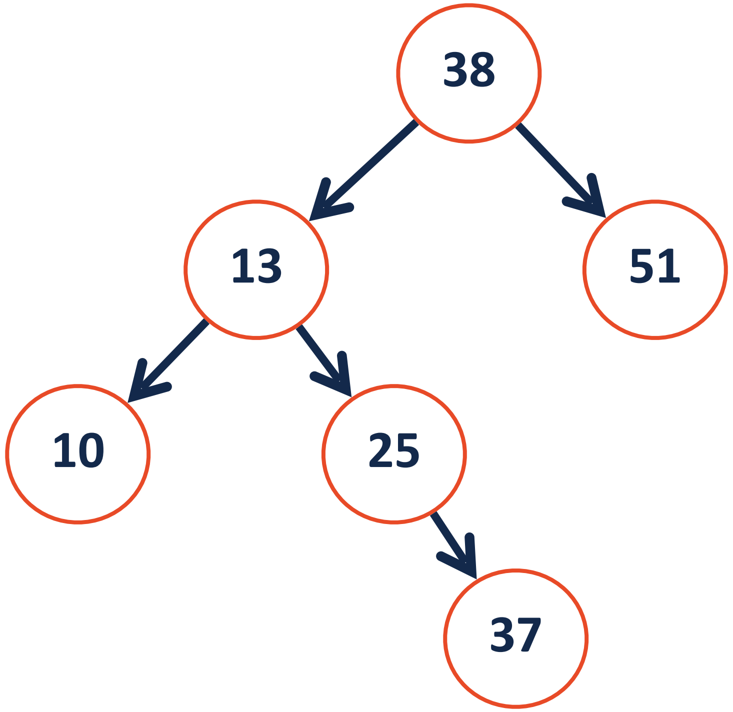
AVL Rotation Practice



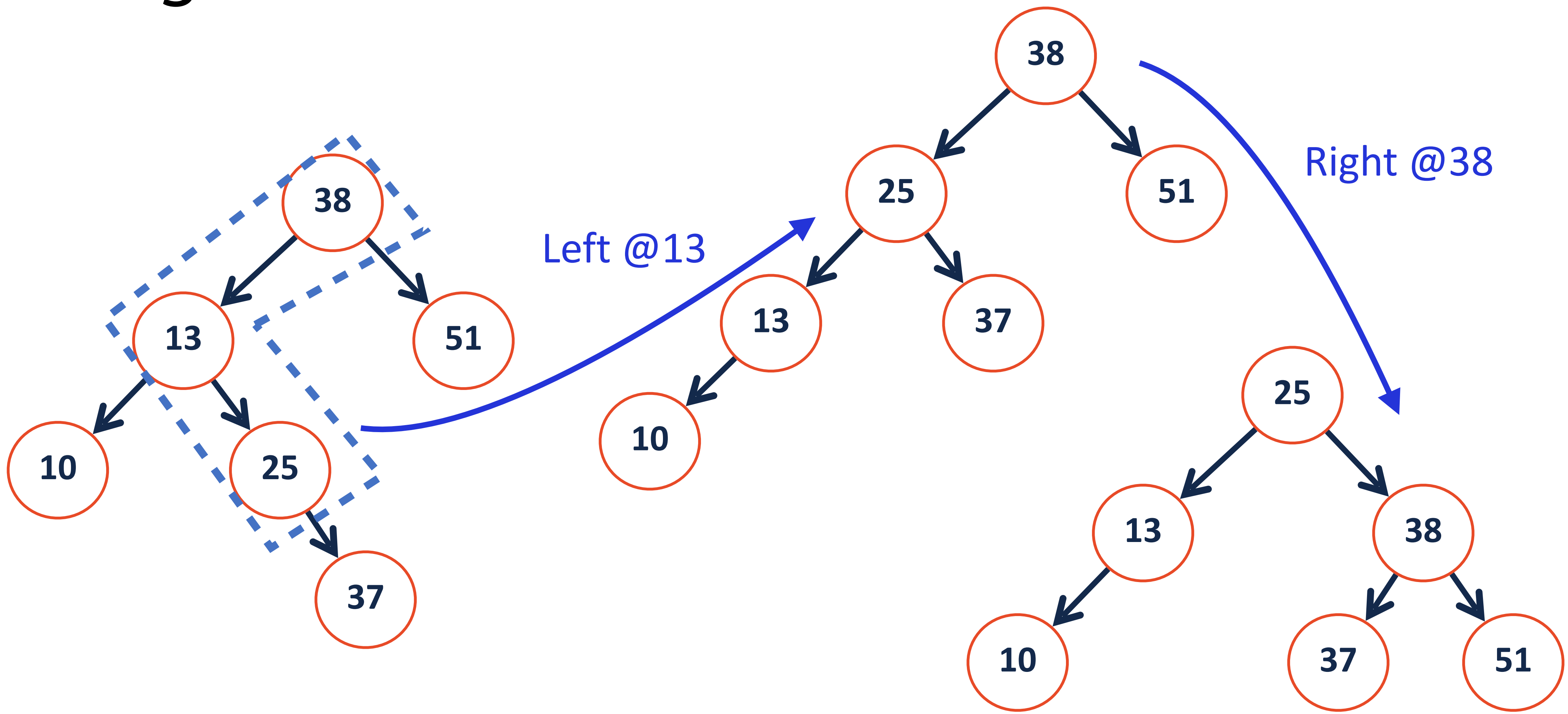


Somethings not quite right…

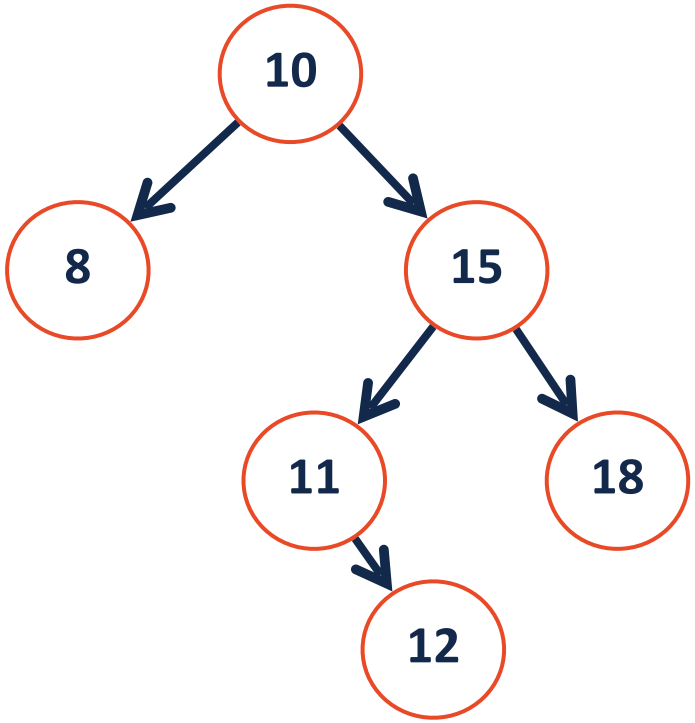
LeftRight Rotation



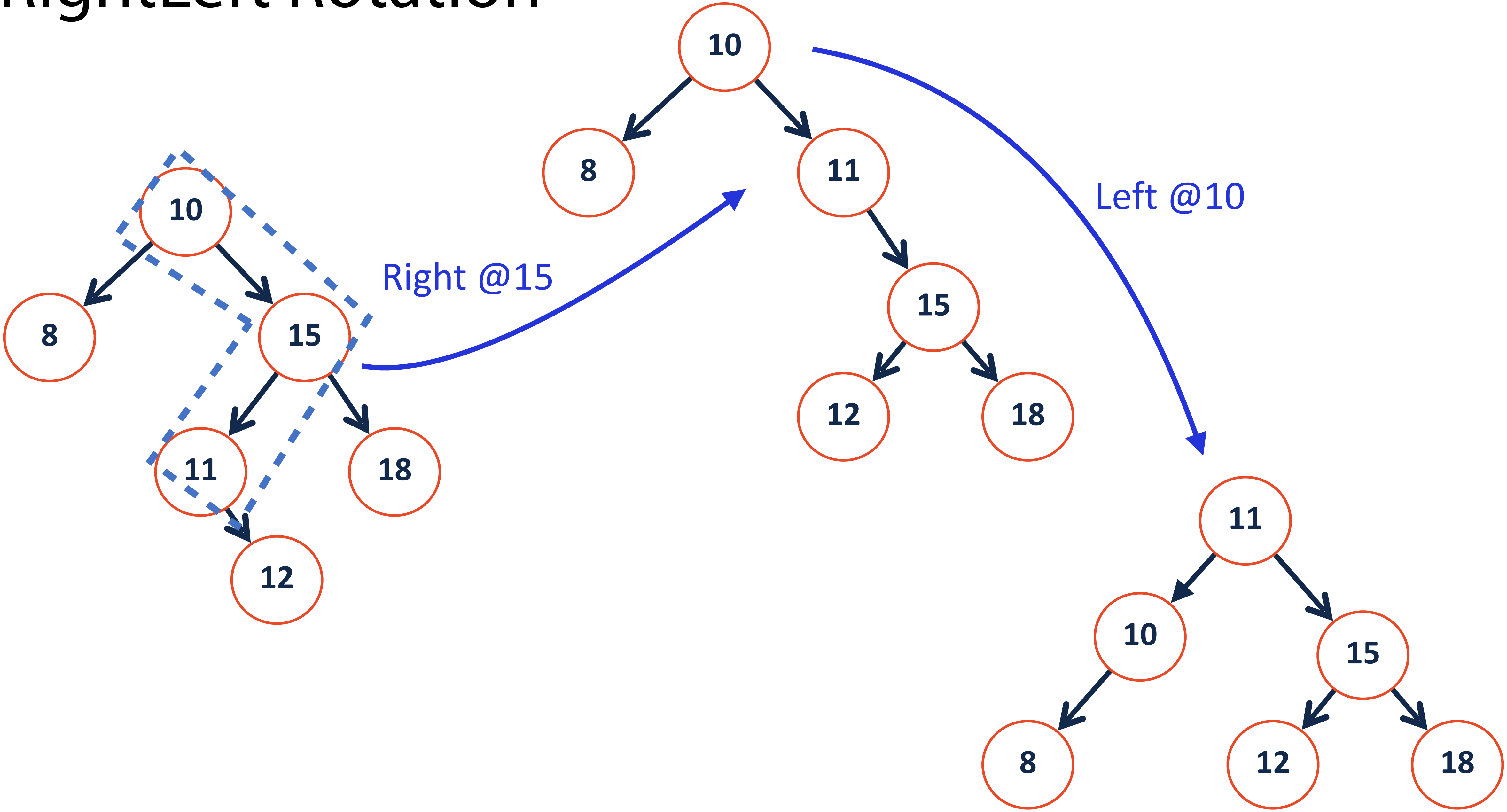
LeftRight Rotation



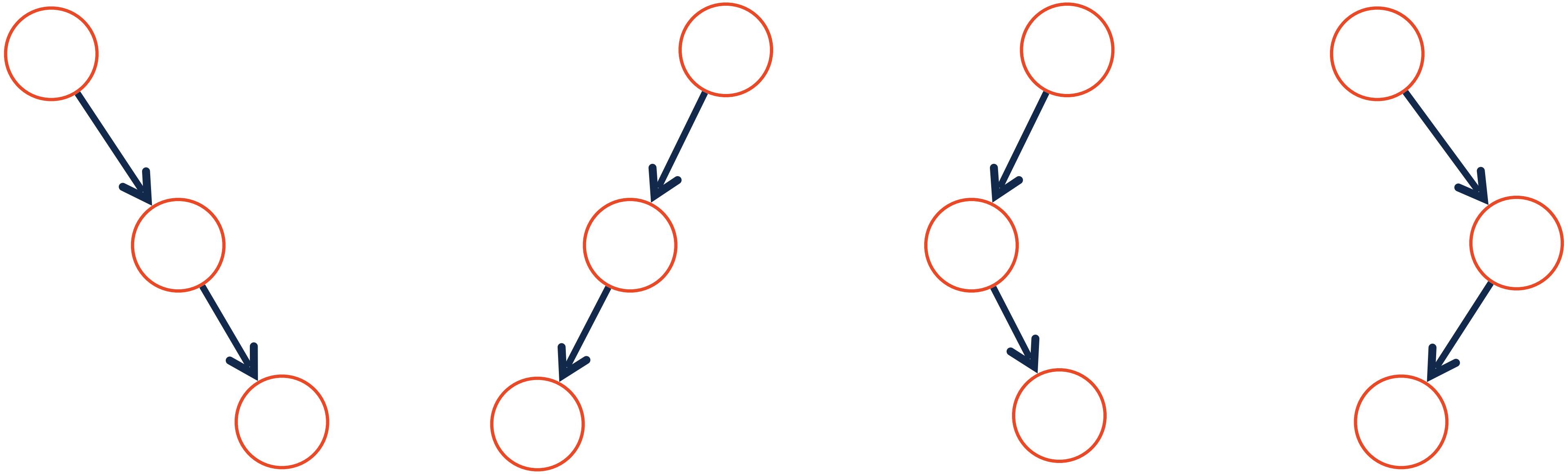
RightLeft Rotation



RightLeft Rotation



AVL Rotations



AVL Rotations

Four kinds of rotations: (L, R, LR, RL)

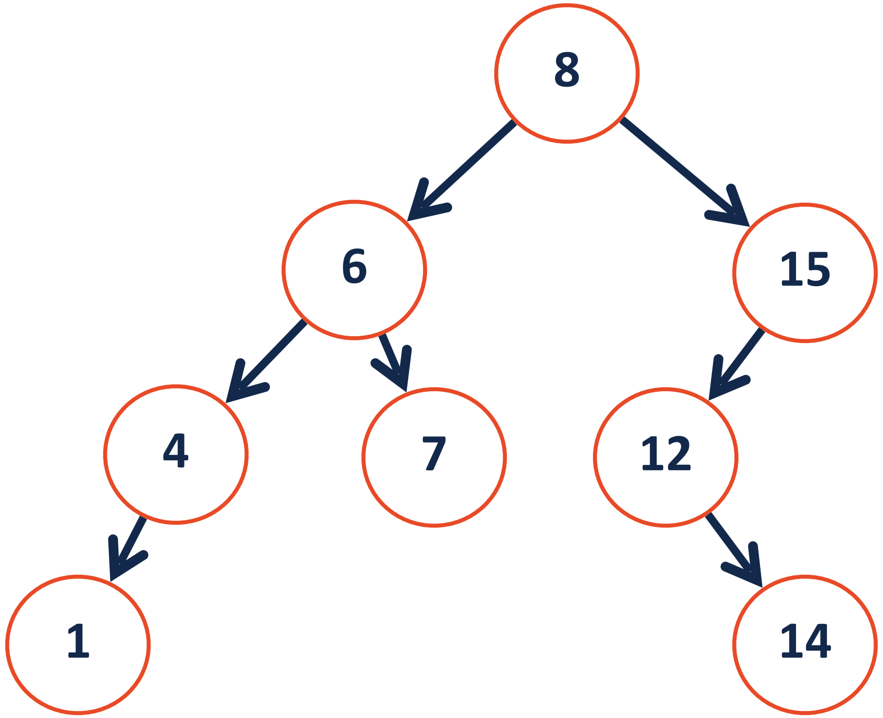
1. All rotations are local (subtrees are not impacted)

2. The running time of rotations are constant

3. The rotations maintain BST property

Goal:

AVL Rotation Practice



AVL vs BST ADT

The AVL tree is a modified binary search tree that rotates **when necessary**

How does the constraint on balance affect the core functions?

Find

Insert

Remove