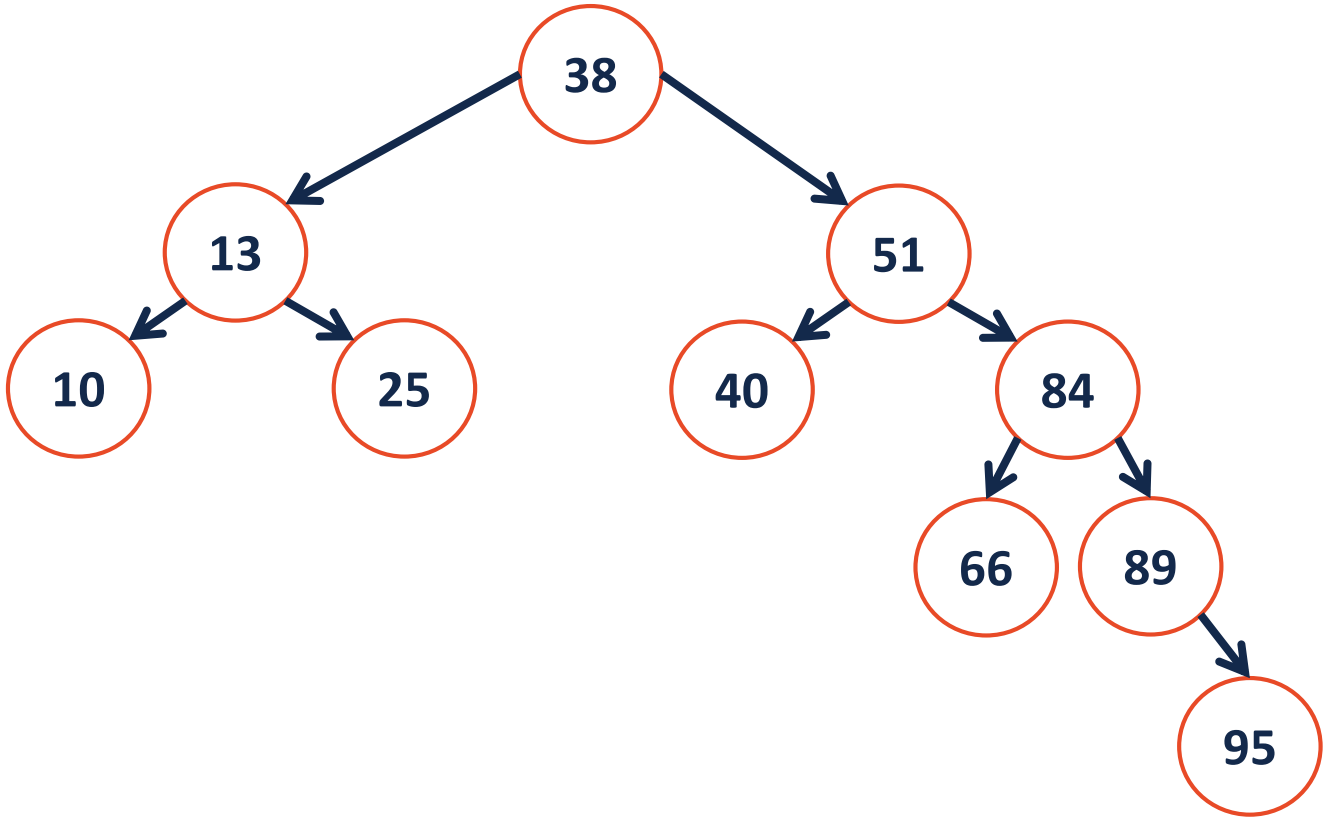
CS 225

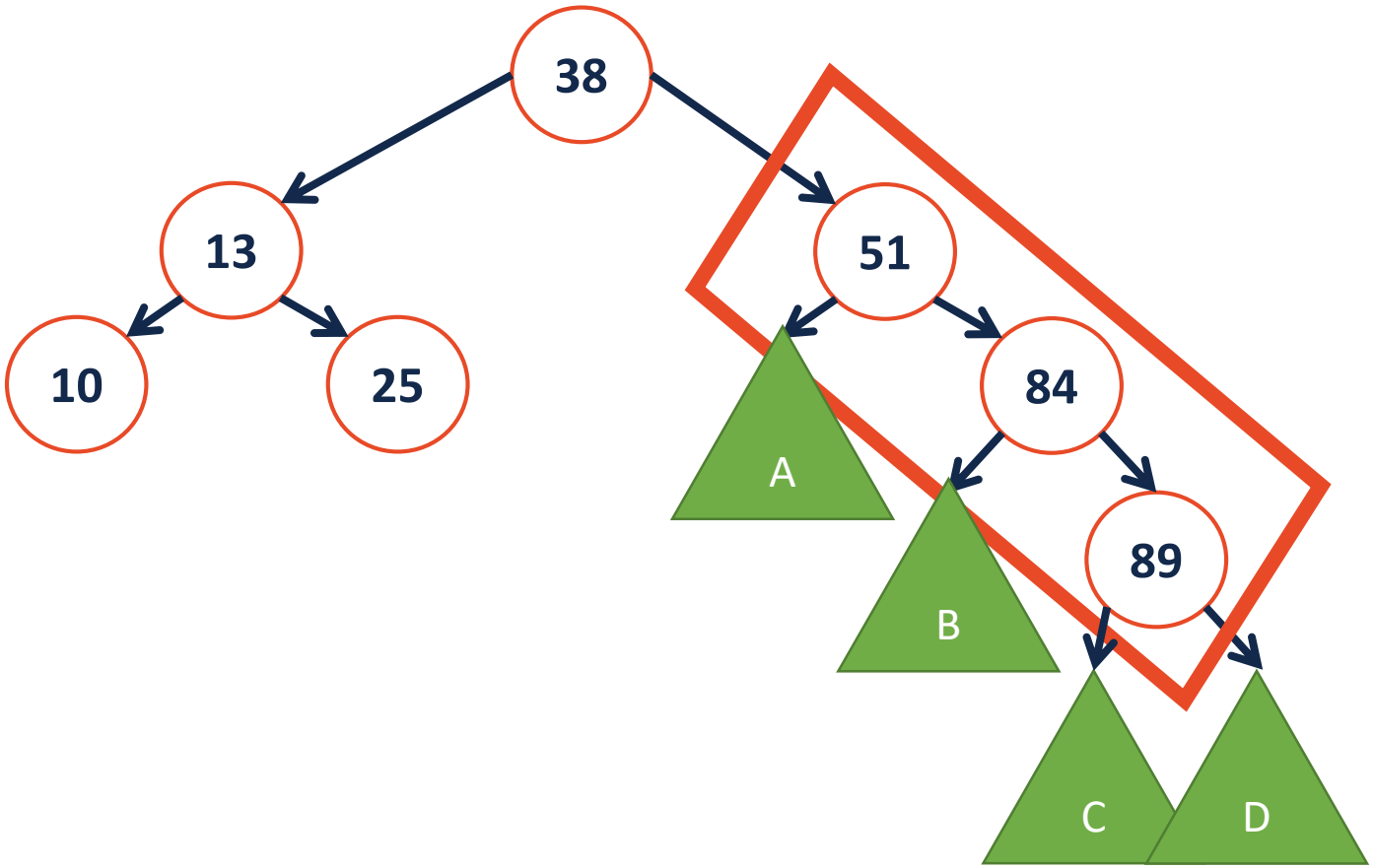
Data Structures

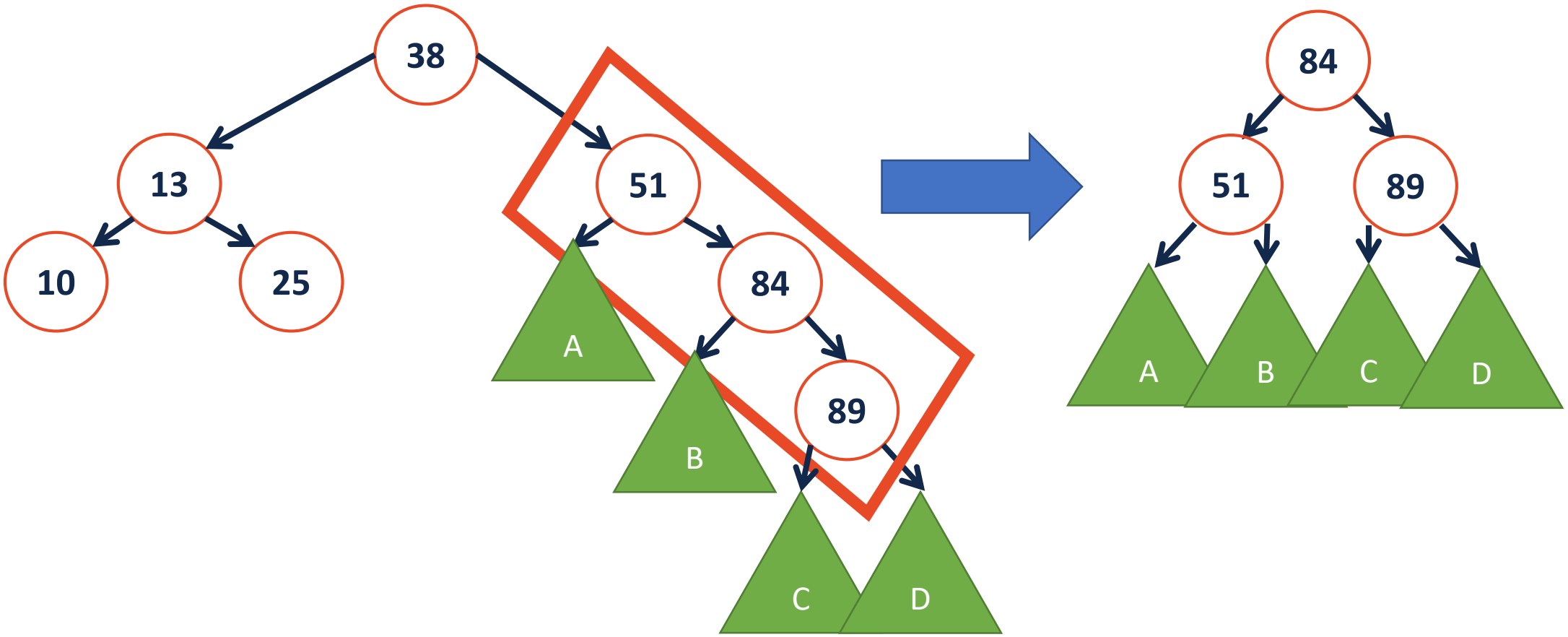
Feb. 27 – AVL Trees

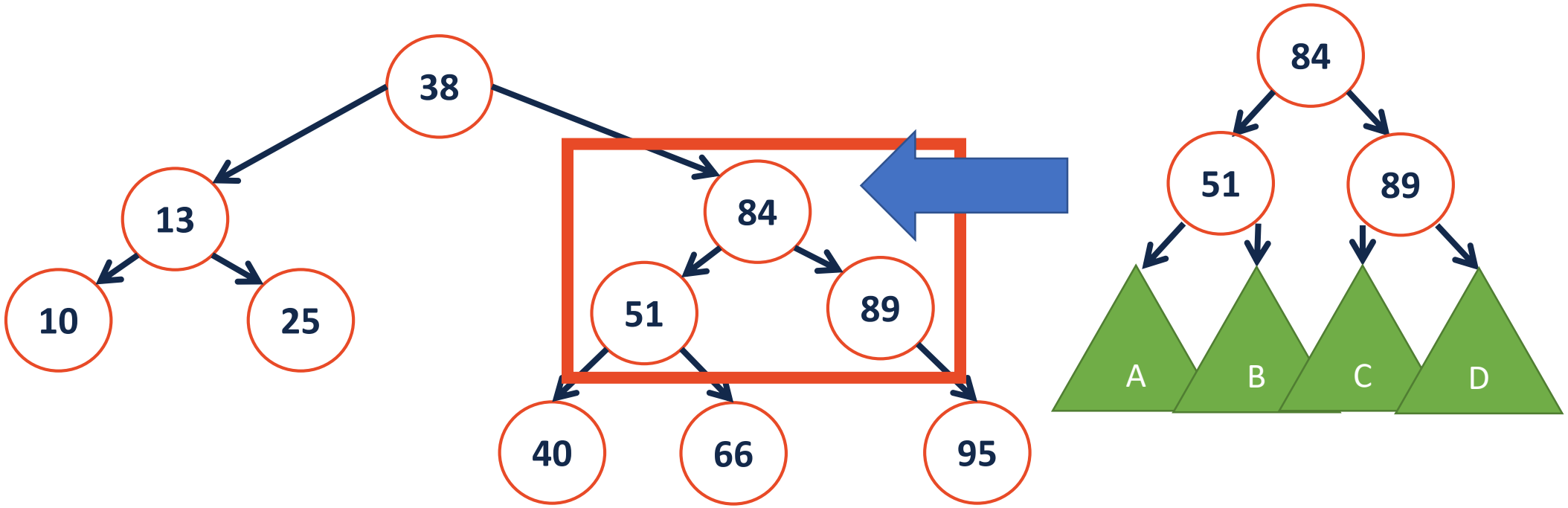
Wade Fagen-Ulmschneider

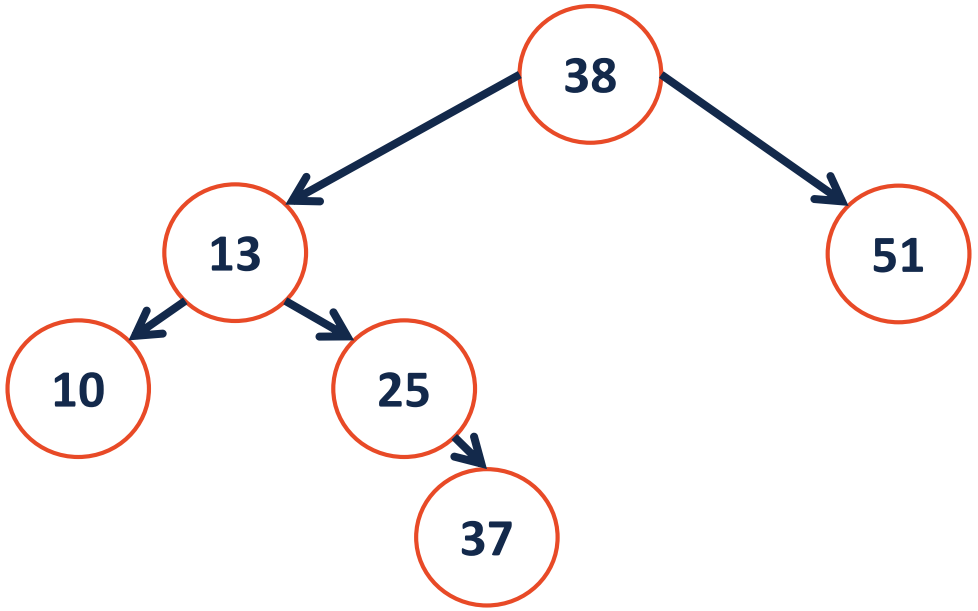
Left Rotation

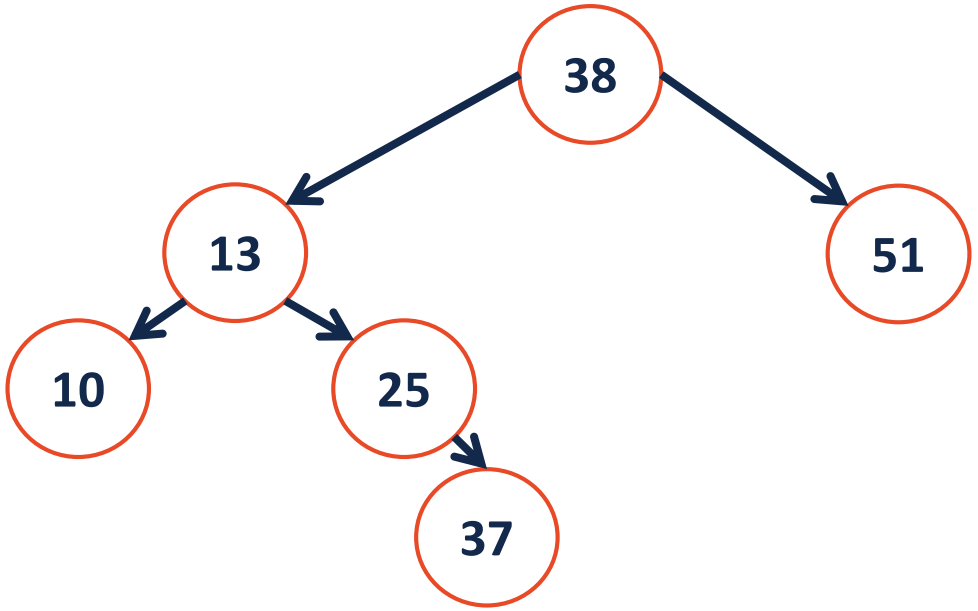



38
13
51
10
25
84
A
89
B
C
D


38
13
51
10
25
84
A
89
B
C
D
84
51
89
A
B
C
D


38
13
10
25
84
51
89
40
66
95
84
51
89
A
B
C
D


38
13
51
10
25
37


38
13
51
10
25
37

BST Rotation Summary

- Four kinds of rotations (L, R, LR, RL)
- All rotations are local (subtrees are not impacted)
- All rotations are constant time: O(1)
- BST property maintained

GOAL:

We call these trees:

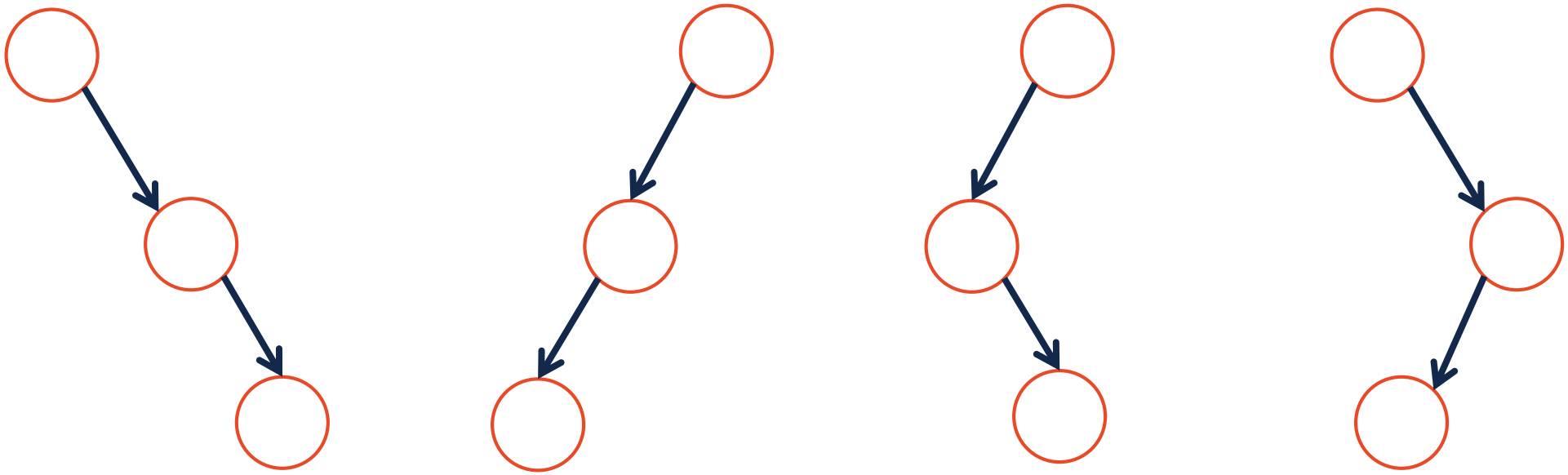
AVL Trees

Three issues for consideration:

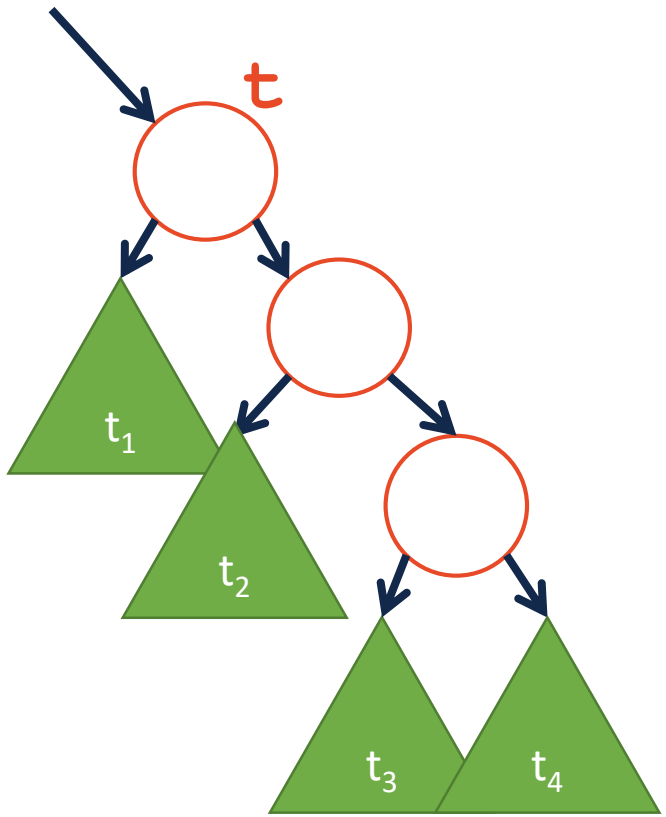
- Rotations
- Maintaining Height
- Detecting Imbalance

AVL Tree Rotations

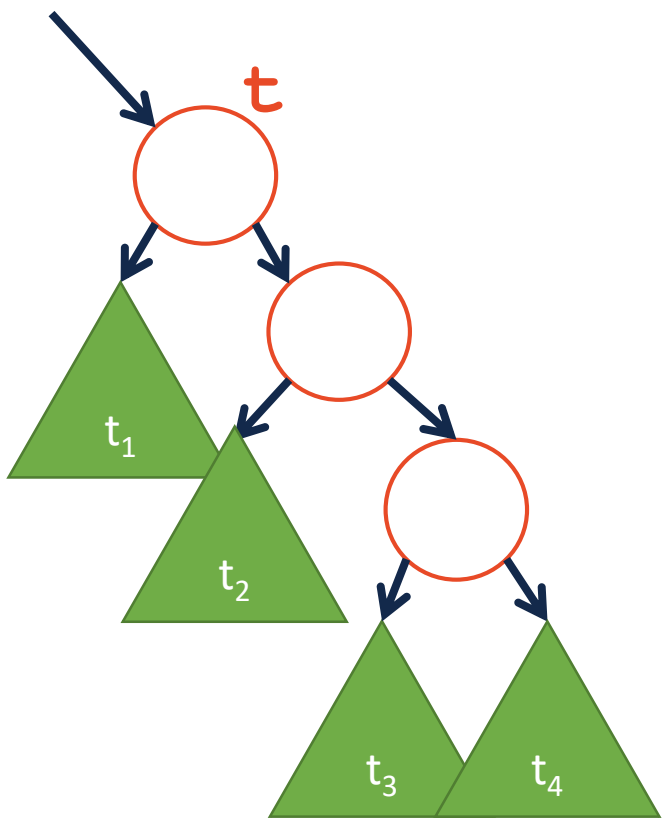
Four templates for rotations:



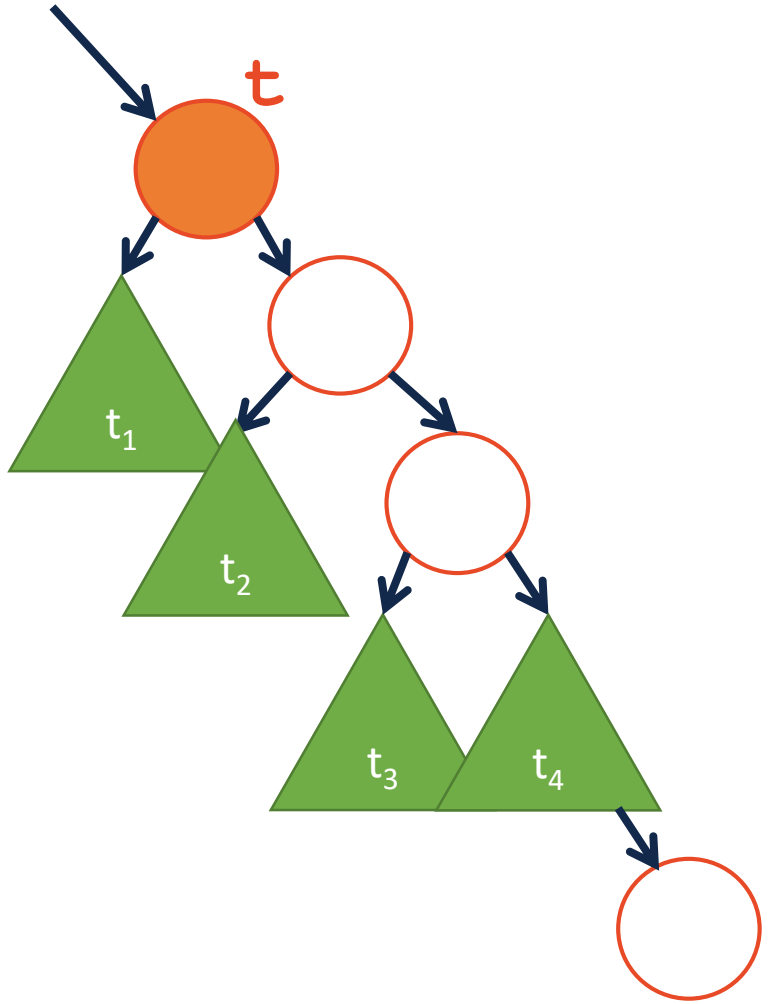
Finding the Rotation



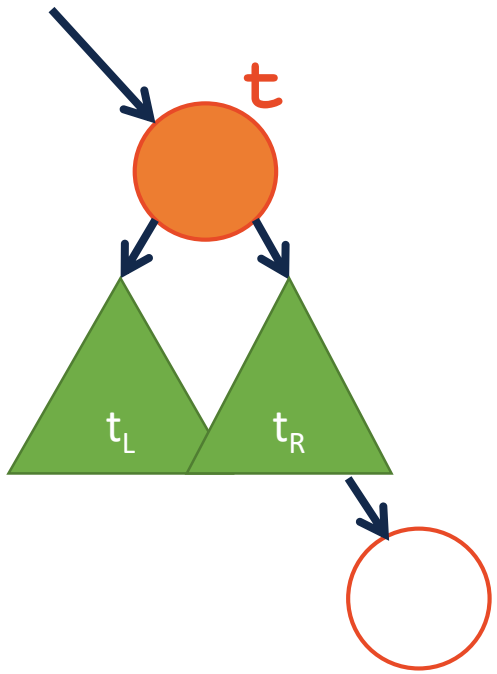
Finding the Rotation



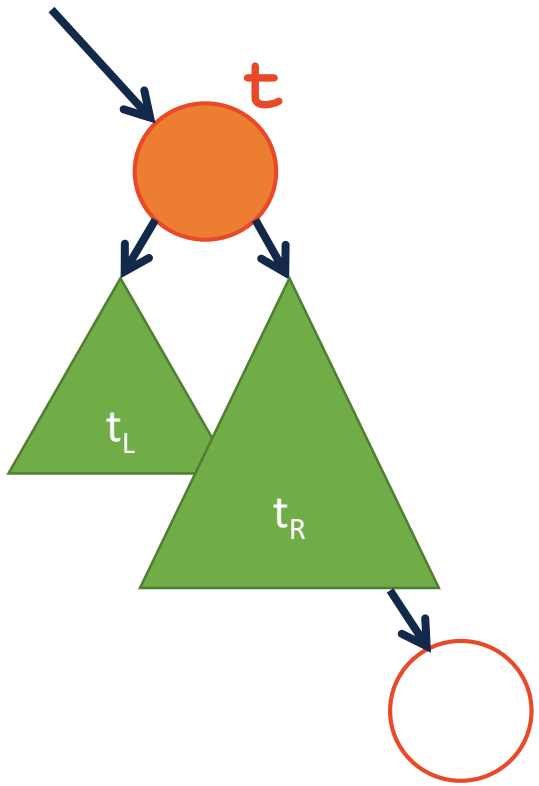
If an insertion occurred in subtrees $\mathbf{t_3}$ or $\mathbf{t_4}$ and a subtree was detected at **t:**



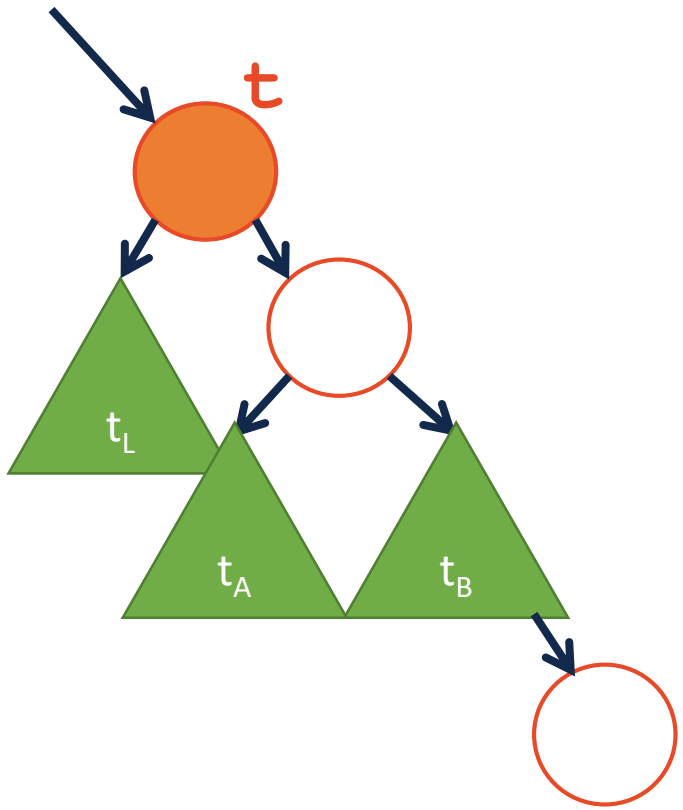
Finding the Rotation



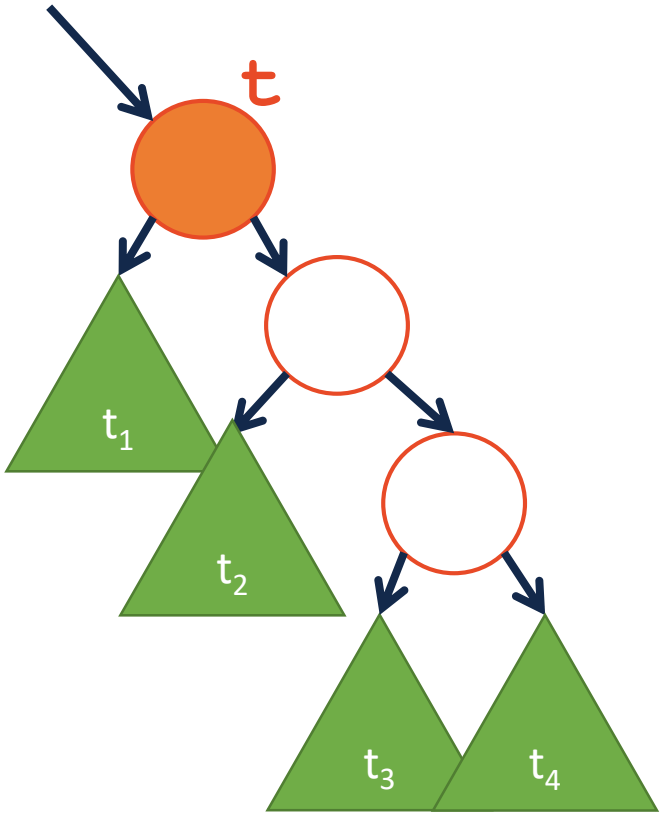
Finding the Rotation



Finding the Rotation



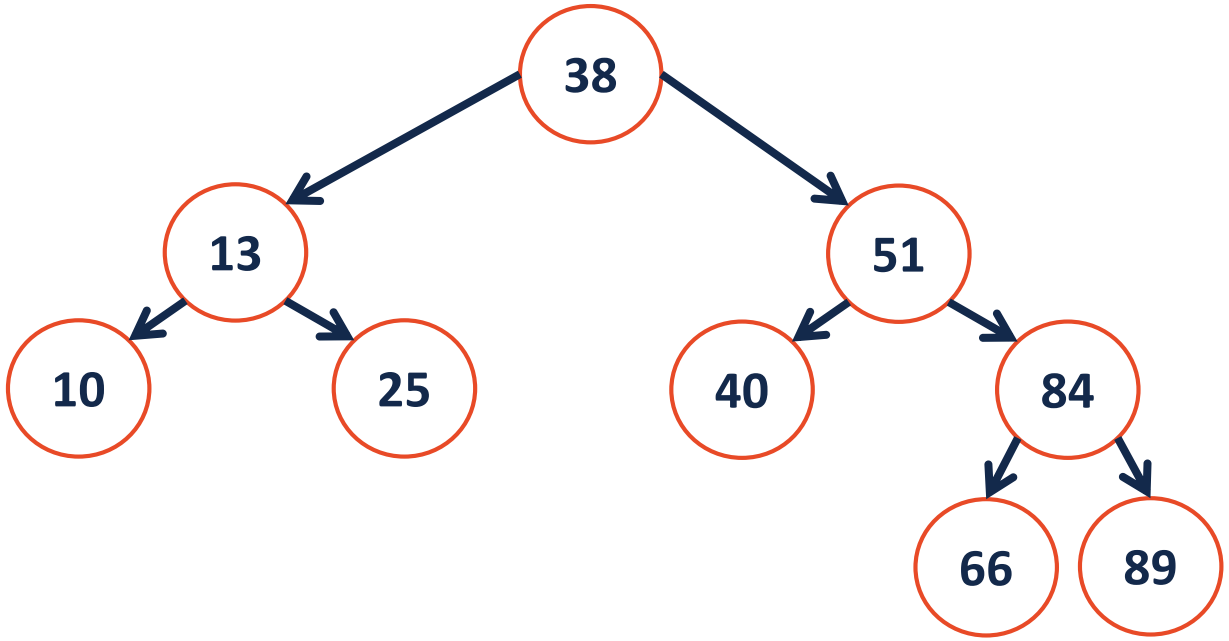
Finding the Rotation



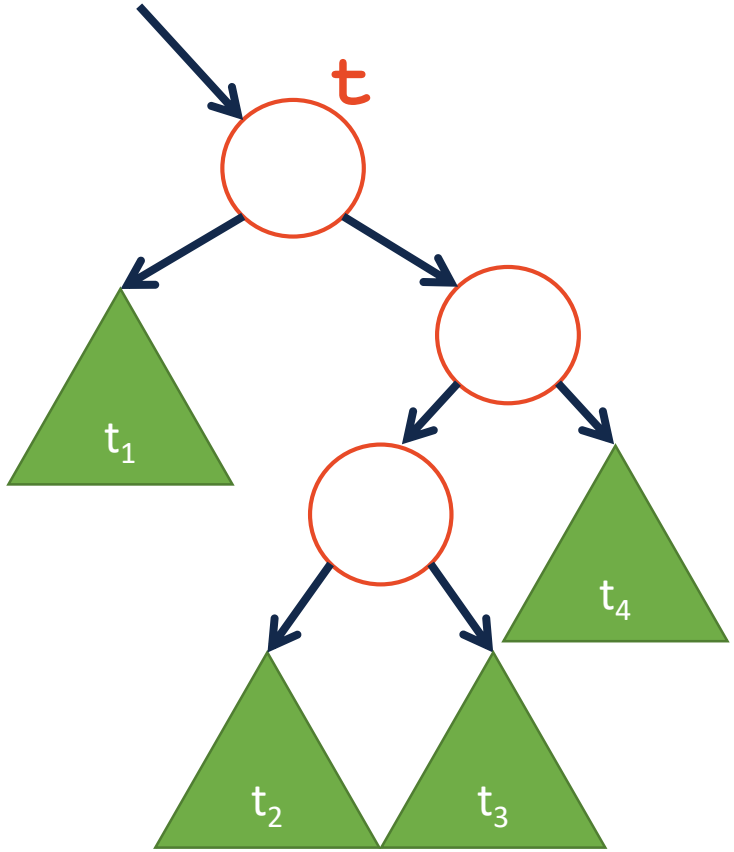
Theorem:
If an insertion occurred in subtrees $\mathbf{t_3}$ or $\mathbf{t_4}$ and a subtree was detected at **t**, then a _____ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t->right** is _____.

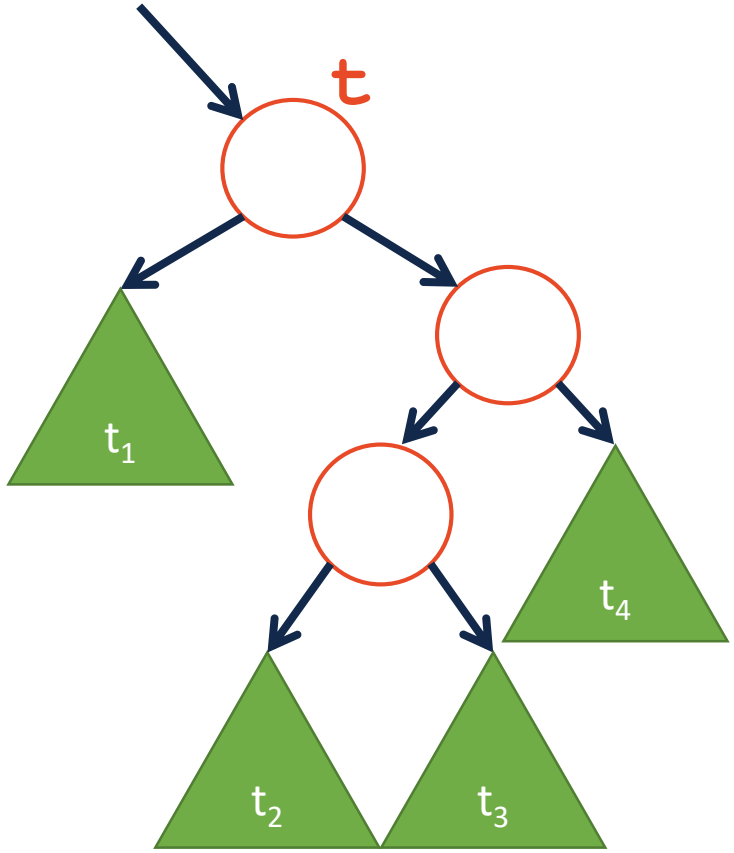
Example:



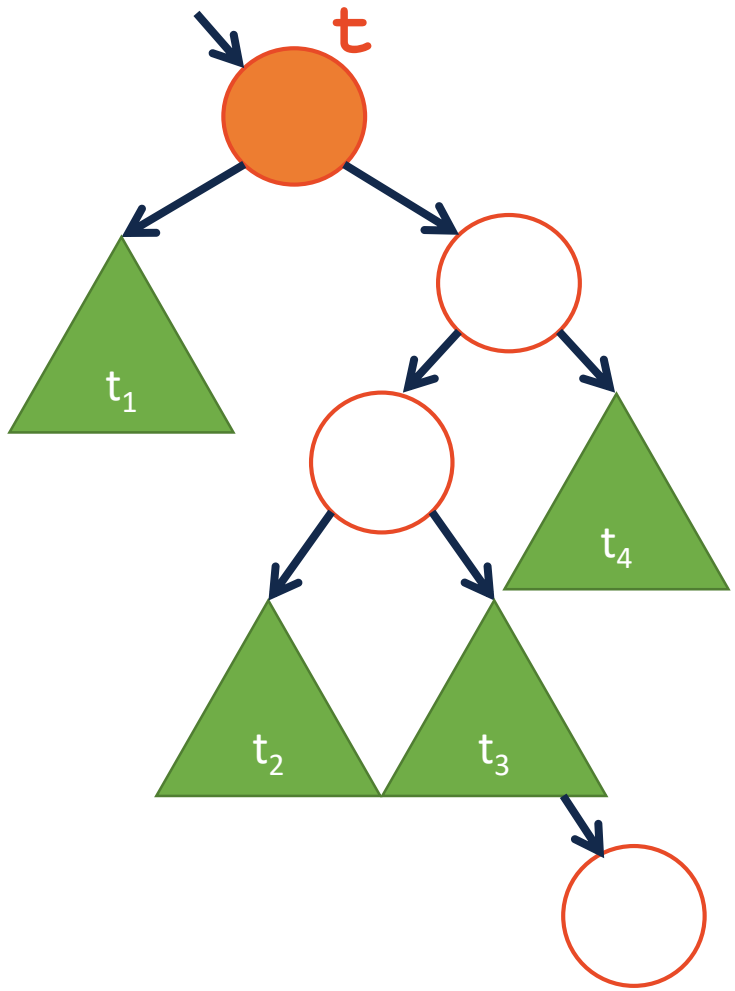
Finding the Rotation



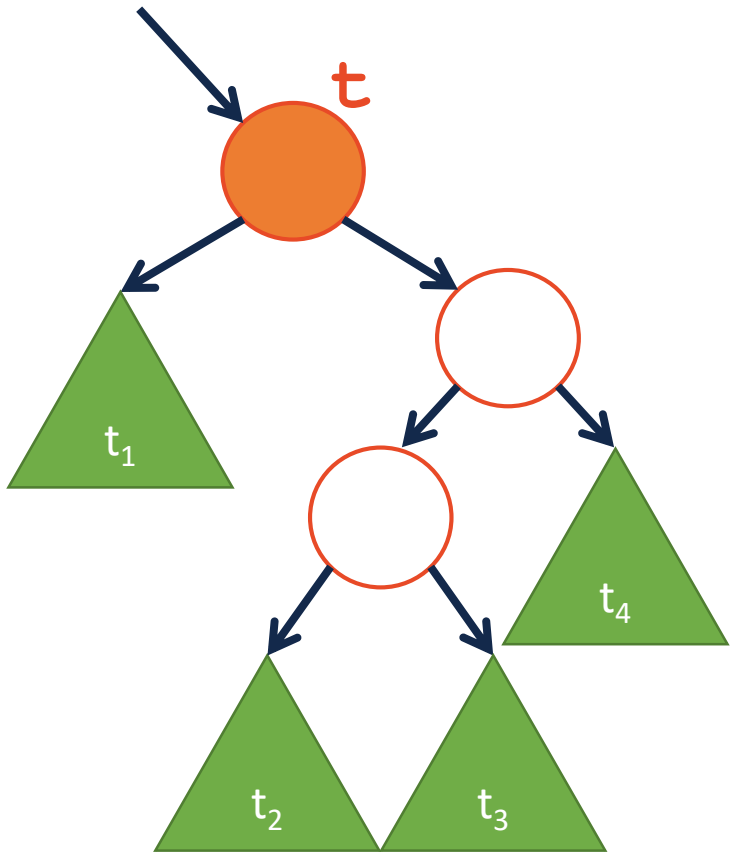
Finding the Rotation



Theorem:
If an insertion occurred in subtrees $\mathbf{t_2}$ or $\mathbf{t_3}$ and a subtree was detected at **t**:



Finding the Rotation



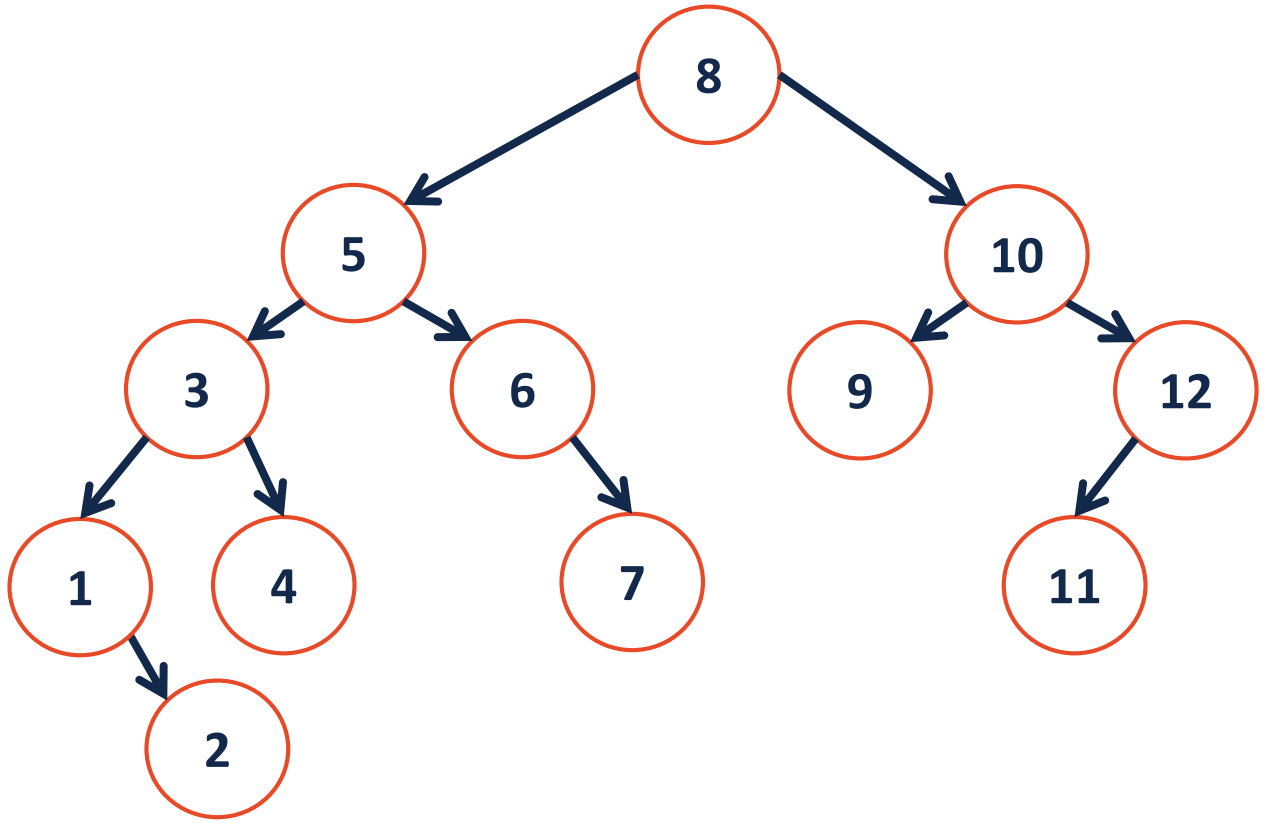
Theorem:
If an insertion occurred in subtrees $\mathbf{t_2}$ or $\mathbf{t_3}$ and a subtree was detected at **t**, then a _____ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t->right** is _____.

Insertion into an AVL Tree

_insert(6.5)

```
1  struct TreeNode {
2    T key;
3    unsigned height;
4    TreeNode *left;
5    TreeNode *right;
6  };
```

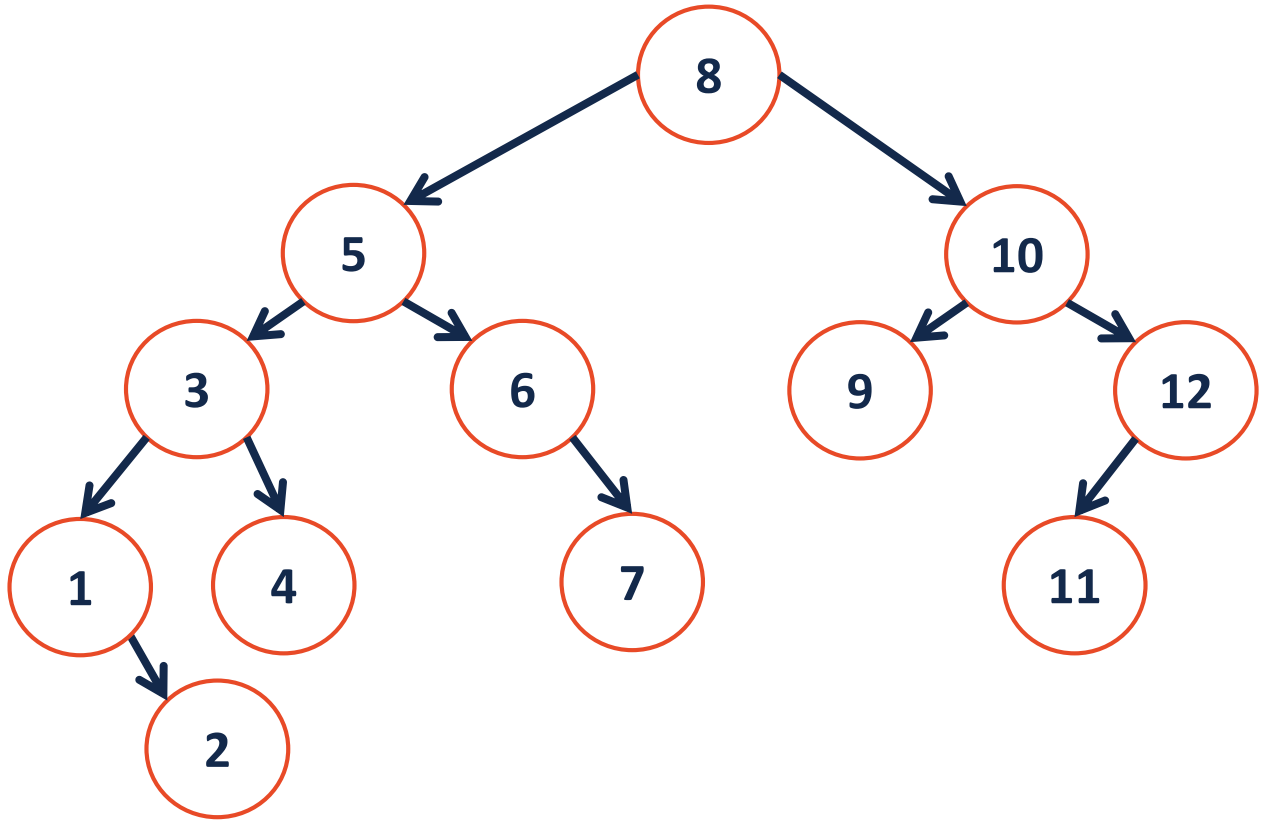


Insertion into an AVL Tree

_insert(6.5)

Insert (pseudo code):

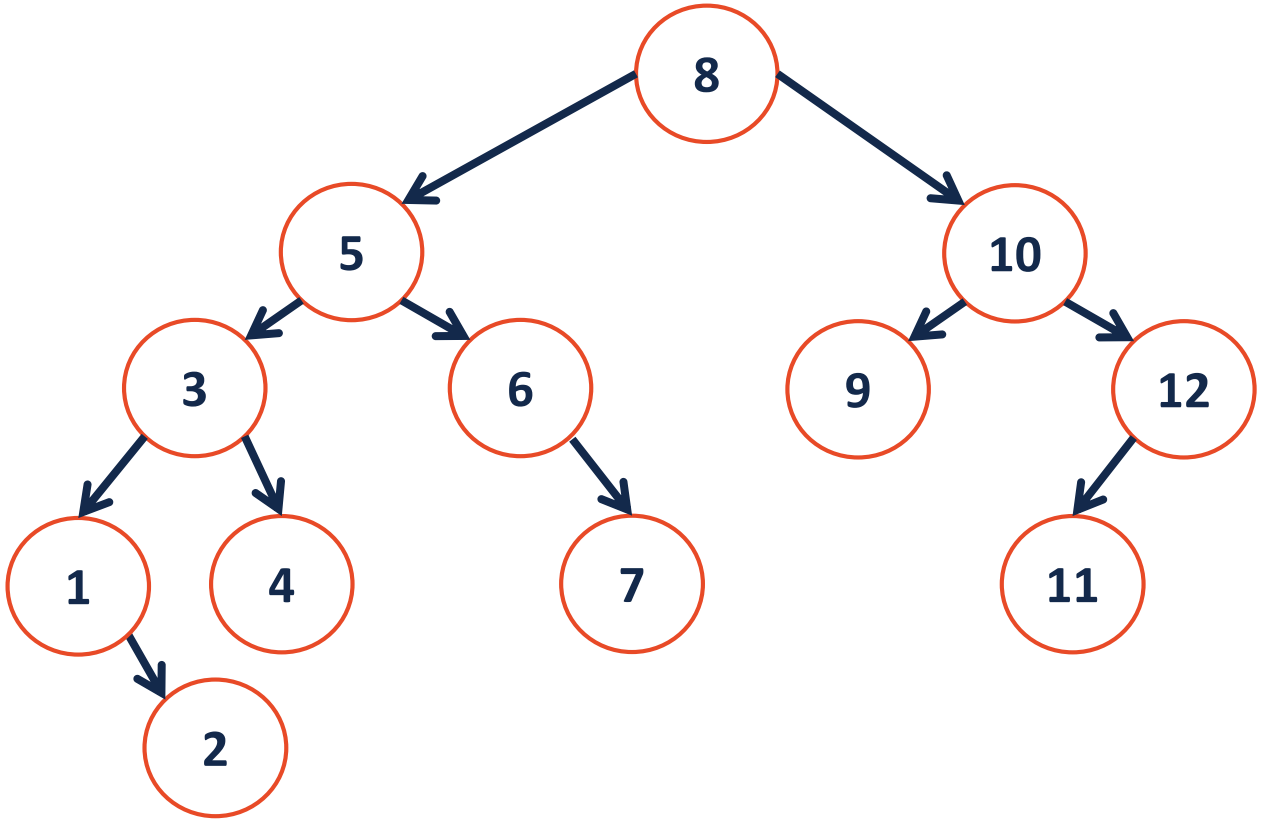
1: Insert at proper place
2: Check for imbalance
3: Rotate, if necessary
4: Update height




```
1  struct TreeNode {
2    T key;
3    unsigned height;
4    TreeNode *left;
5    TreeNode *right;
6  };
```

```
151 template <typename K, typename V>
152 void AVL<K, D>::_insert(const K & key, const V & data, TreeNode
    *& cur) {
153   if (cur == NULL)           { cur = new TreeNode(key, data);   }
157   else if (key < cur->key) { _insert( key, data, cur->left ); }
160   else if (key > cur->key) { _insert( key, data, cur->right );}
166   _ensureBalance(cur);
167 }
```

```
119 template <typename K, typename V>
120 void AVL<K, D>::_ensureBalance(TreeNode *& cur) {
121   // Calculate the balance factor:
122   int balance = height(cur->right) - height(cur->left);
123
124   // Check if the node is current not in balance:
125   if ( balance == -2 ) {
126     int l_balance =
            height(cur->left->right) - height(cur->left->left);
127     if ( l_balance == -1 ) { _____________________________________; }
128     else                   { _____________________________________; }
129   } else if ( balance == 2 ) {
130     int r_balance =
            height(cur->right->right) - height(cur->right->left);
131     if( r_balance == 1 ) { _____________________________________; }
132     else                 { _____________________________________; }
133   }
134
135   _updateHeight(cur);
136 };
```


8
5
10
3
6
9
12
1
4
7
11
2

AVL Tree Analysis

We know: insert, remove and find runs in: _____.

We will argue that: h = _____.

AVL Tree Analysis

Definition of big-O:

…or, with pictures: