CS 225

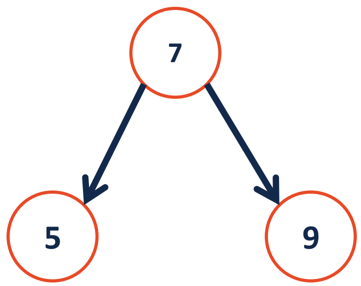
Data Structures

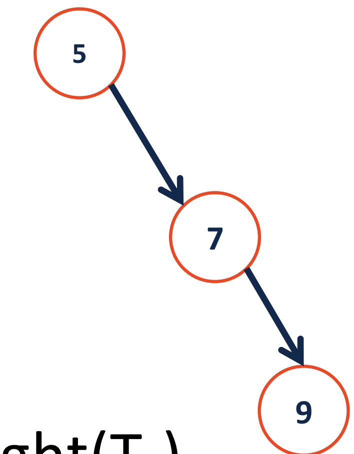
February 17 – BST Balance

G Carl Evans

Height-Balanced Tree

What tree makes you happier?





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

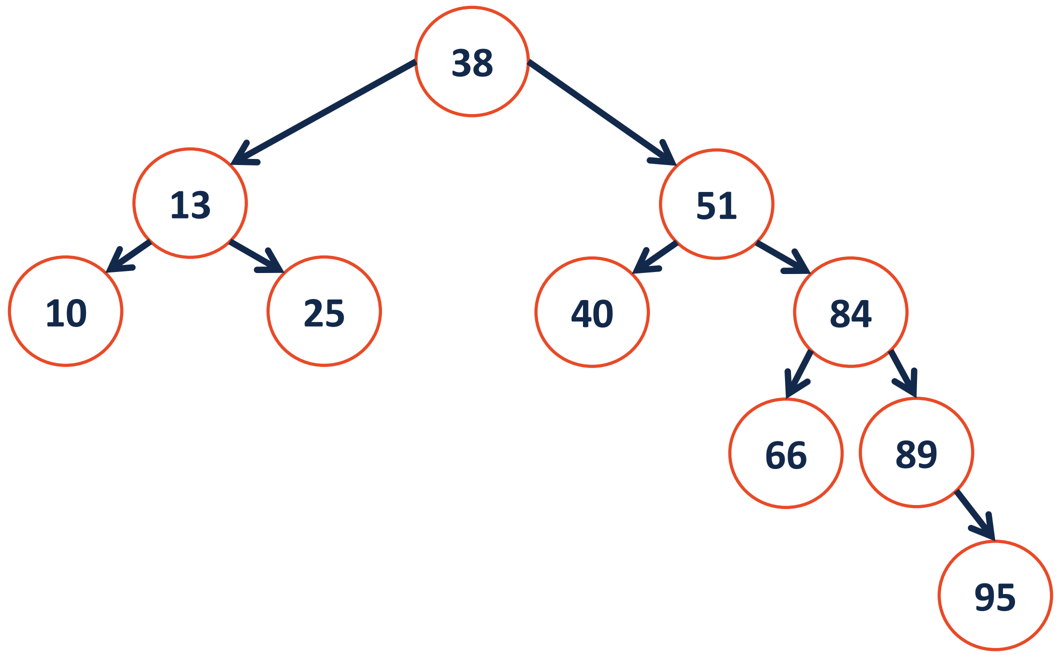
A tree is height balanced if:

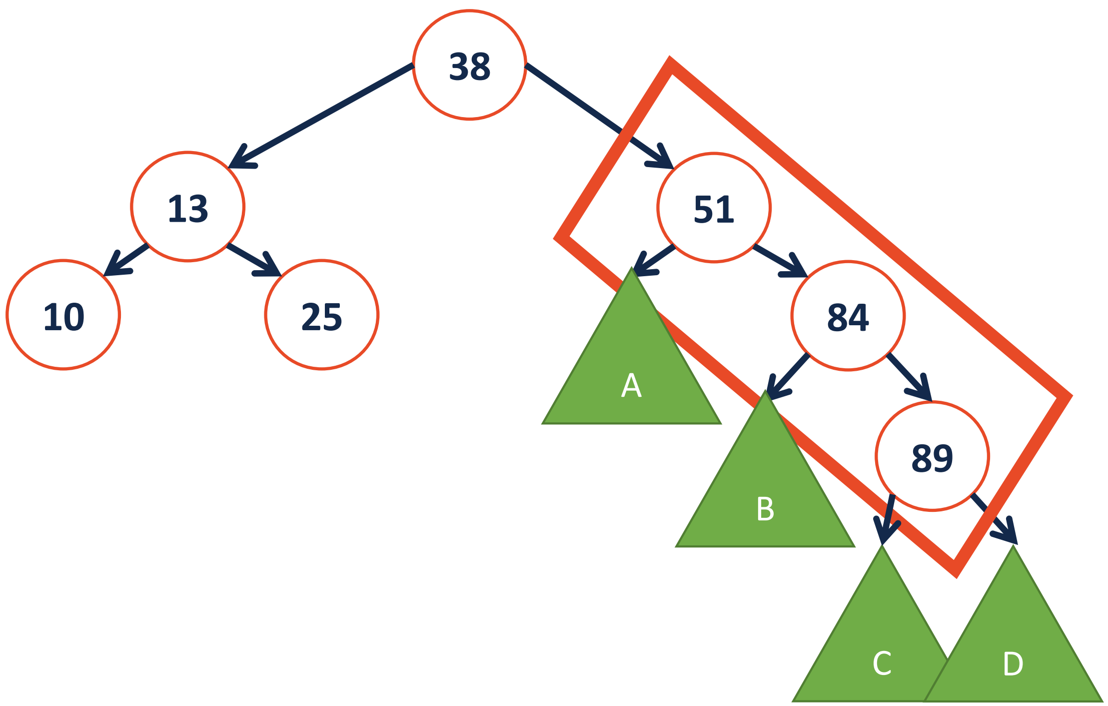
BST Rotation

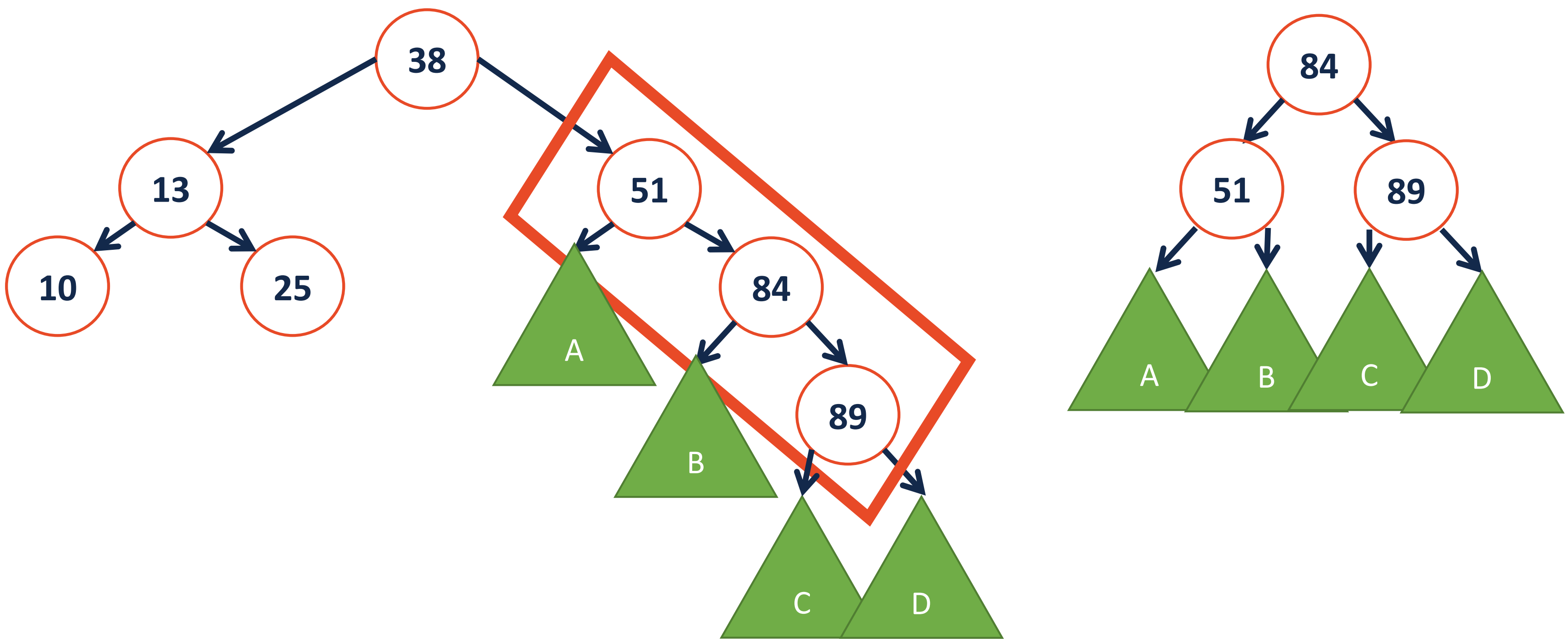
We will perform a rotation that maintains two properties:

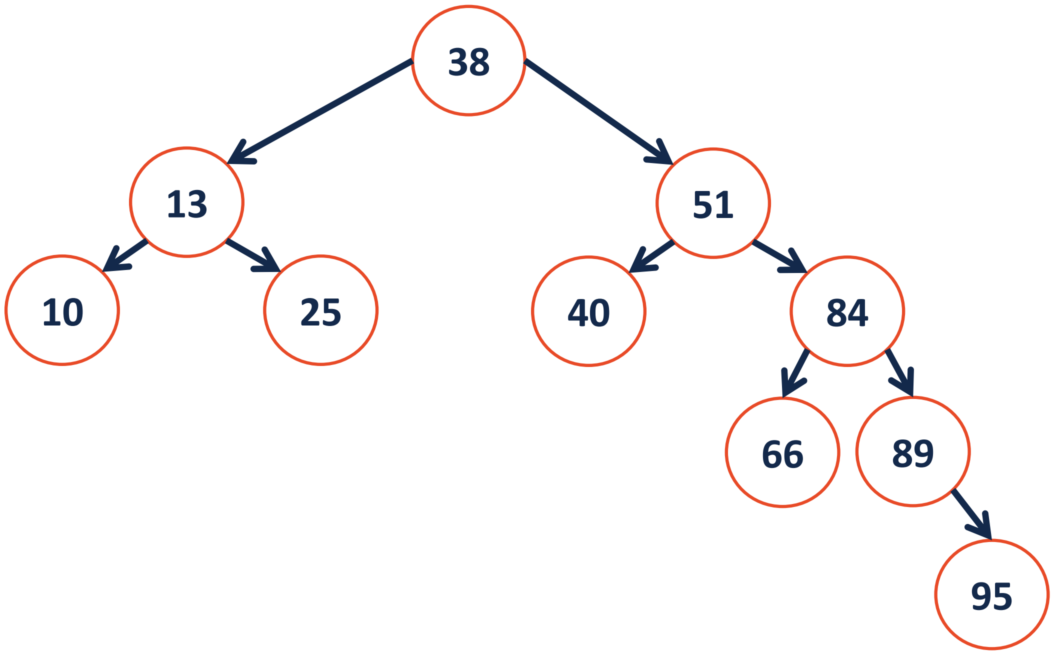
1.

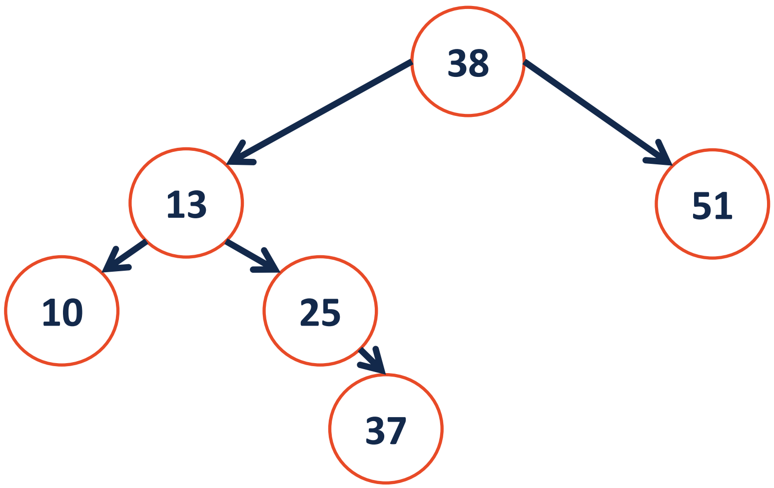
2.

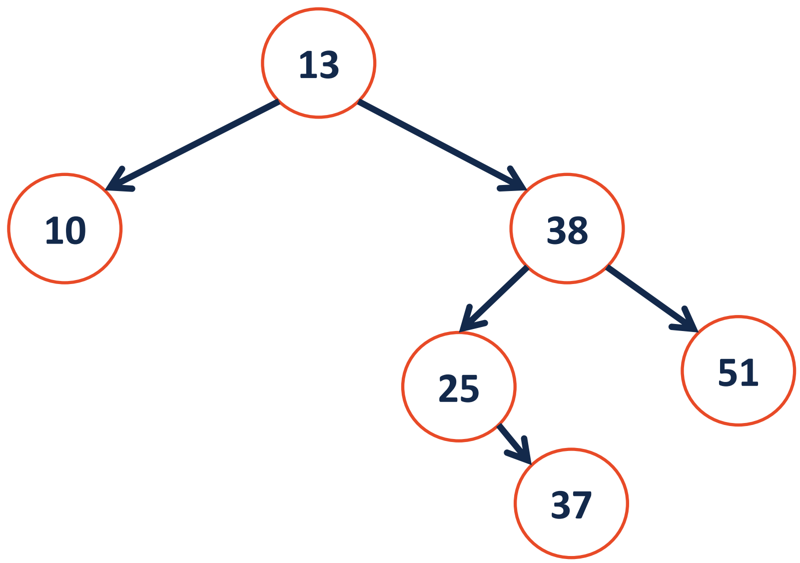

38
13
51
10
25
40
84
66
89
95

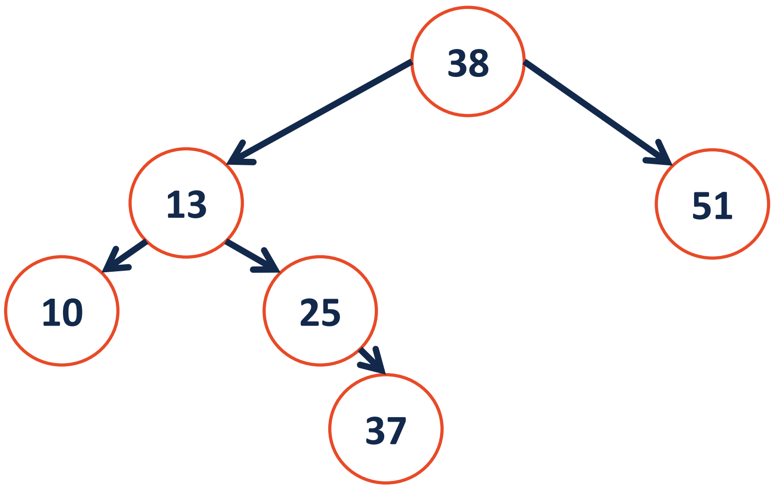

38
13
51
10
25
84
89
A
B
C
D


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


38
13
51
10
25
40
84
66
89
95


38
13
51
10
25
37


13
10
38
25
51
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:

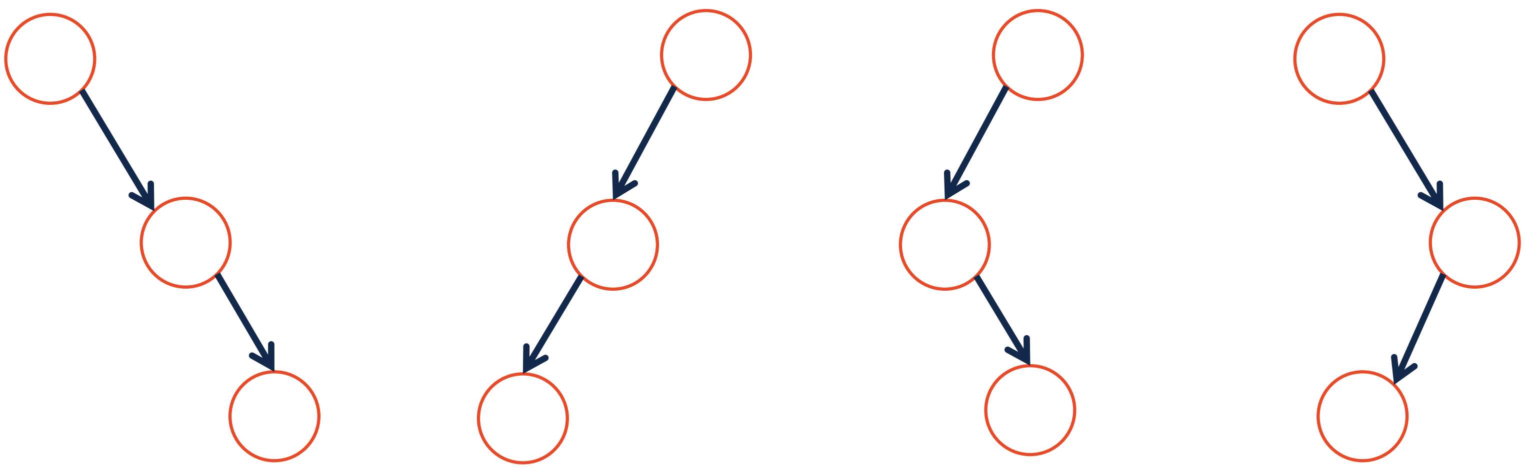
AVL Trees

Three issues for consideration:

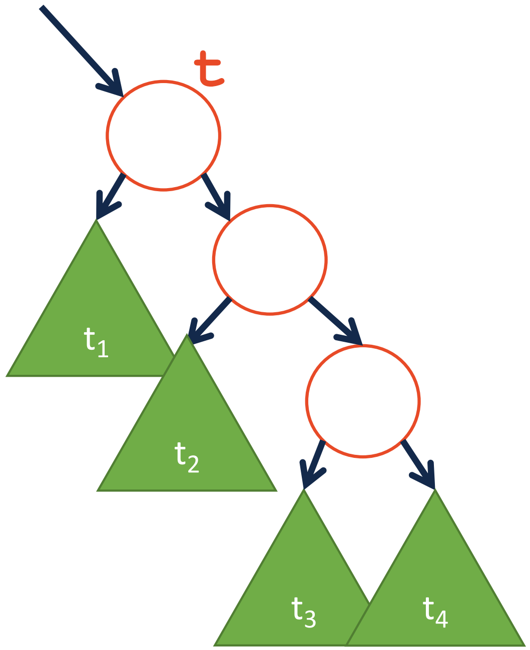
- Rotations
- Maintaining Height
- Detecting Imbalance

AVL Tree Rotations

Four templates for rotations:



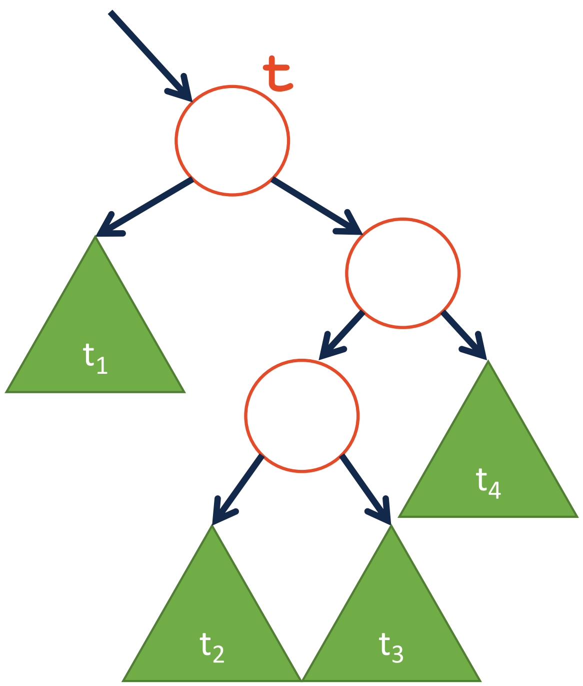
Finding the Rotation on Insert



Theorem:
If an insertion occurred in subtrees **t_3** or **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 ______.

Finding the Rotation on Insert

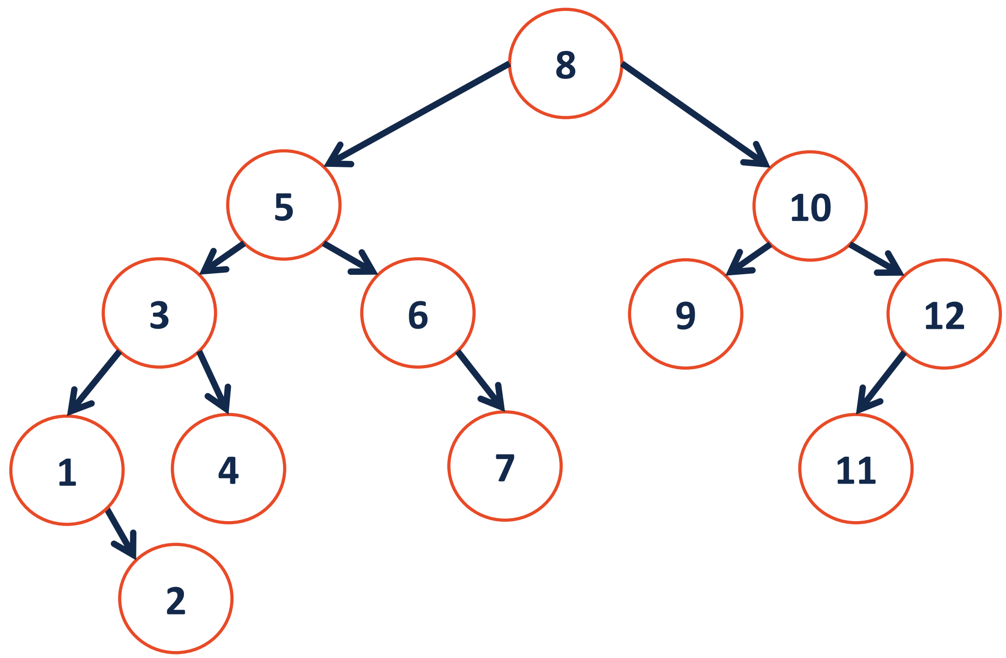


Theorem:
If an insertion occurred in subtrees $\mathbf{t_2}$ or $\mathbf{t_3}$ and a subtree was detected at **t**, then a ___________ rotation about **t** restores the balance of the tree.

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

Insertion into an AVL Tree

`_insert(6.5)`




```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

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

```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```