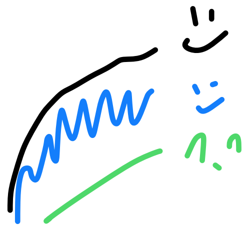
Data Structures
AVL Analysis

CS 225
Brad Solomon & G Carl Evans

September 27, 2023



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Review AVL trees

Prove that the AVL Tree speeds up all operations

1) Our rotations fix the imbalances in a AVL tree

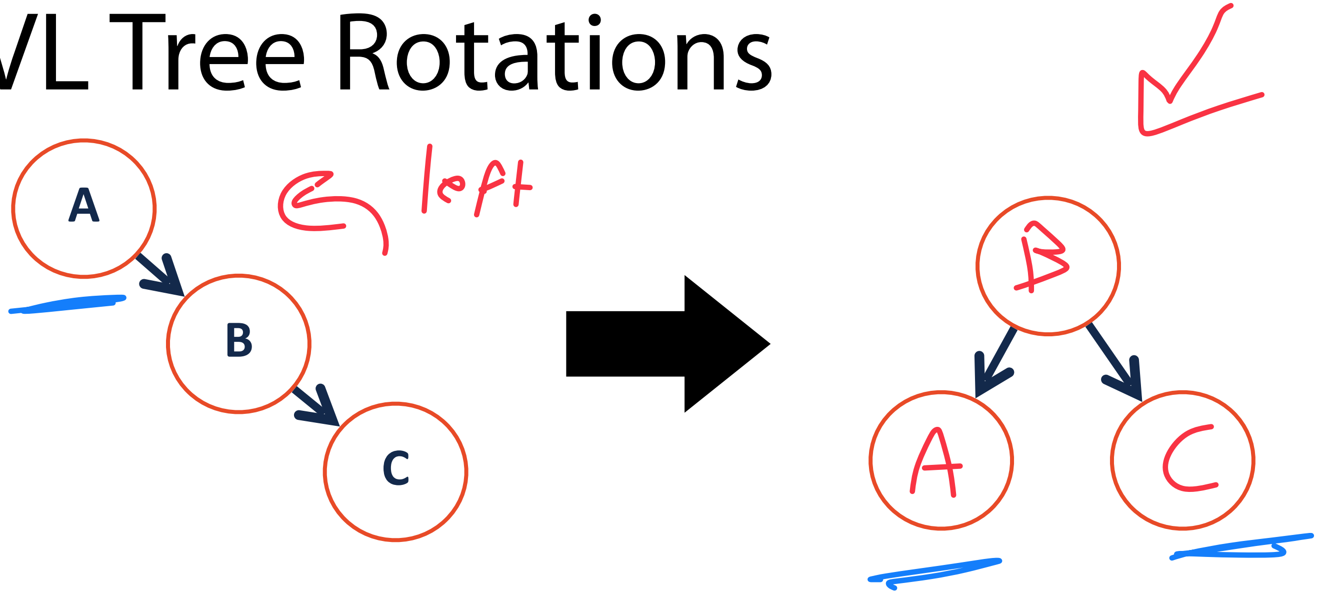
↳ Any operation that modifies our trees can be fixed

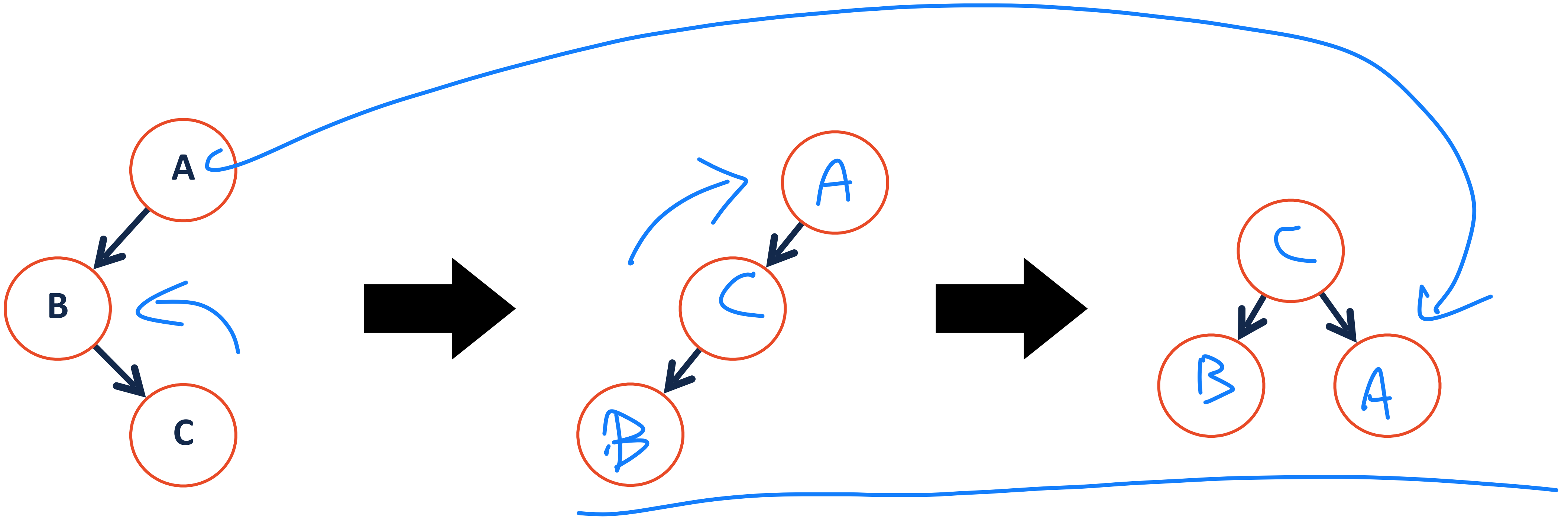
2) The balance of an AVL tree limits our height

AVL Tree Rotations

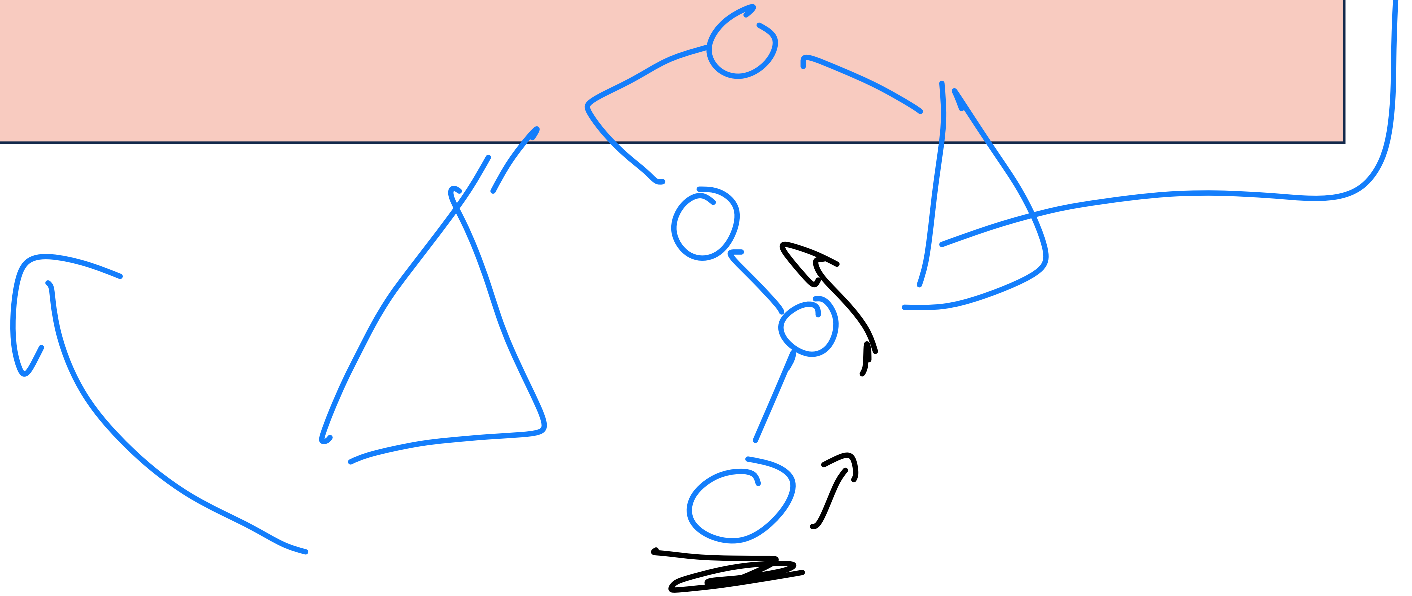
All rotations are O(1)

All rotations reduce subtree height by one





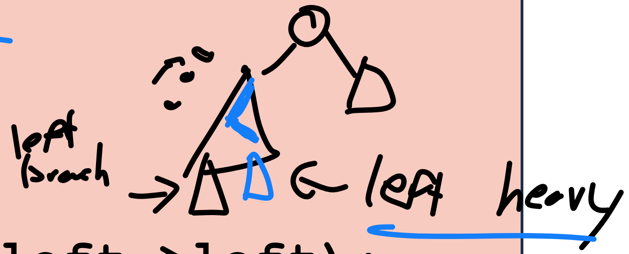
```
template <typename K, typename V>
void AVL<K, D>::_insert(const K & key, const V & data, TreeNode *& cur) {
  if (cur == NULL)           { cur = new TreeNode(key, data);   }
  else if (key < cur->key) { _insert( key, data, cur->left ); }
  else if (key > cur->key) { _insert( key, data, cur->right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



```
template <typename K, typename V>
void AVL<K, D>::_ensureBalance(TreeNode *& cur) {
  // Calculate the balance factor:
  int balance = height(cur->right) - height(cur->left);

  // Check if the node is current not in balance:
  if ( balance == -2 ) {
    int l_balance =
          height(cur->left->right) - height(cur->left->left);
    if ( l_balance == -1 ) { ______rotateRight()__________; }
    else                   { _____rotateLeftRight()_______; }
  } else if ( balance == 2 ) {
    int r_balance =
          height(cur->right->right) - height(cur->right->left);
    if( r_balance == 1 ) { ______rotateLeft()__________; }
    else                 { _____rotateRightLeft()_______; }
  }

  _updateHeight(cur);
};
```

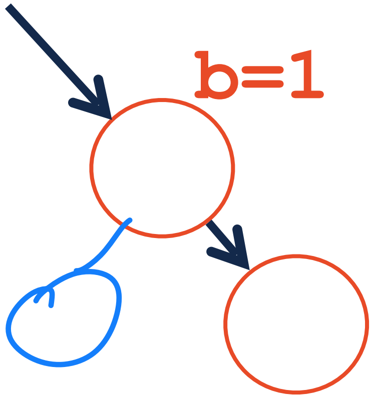


AVL Insertion

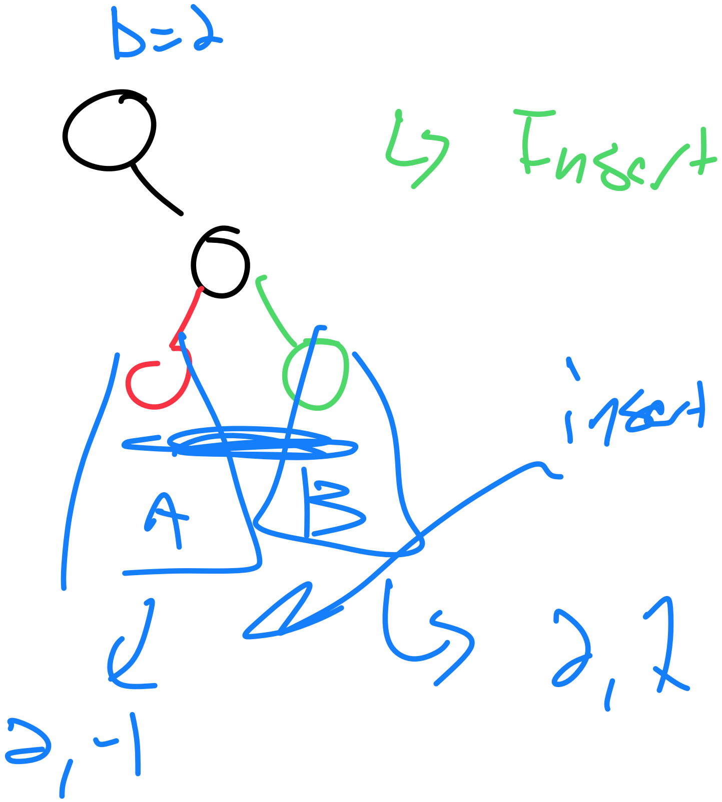
Given an AVL is balanced, insert can insert **at most** one imbalance

one node

↳ Insert can sometimes not change height



b=0



↳ Insert can cause an imbalance

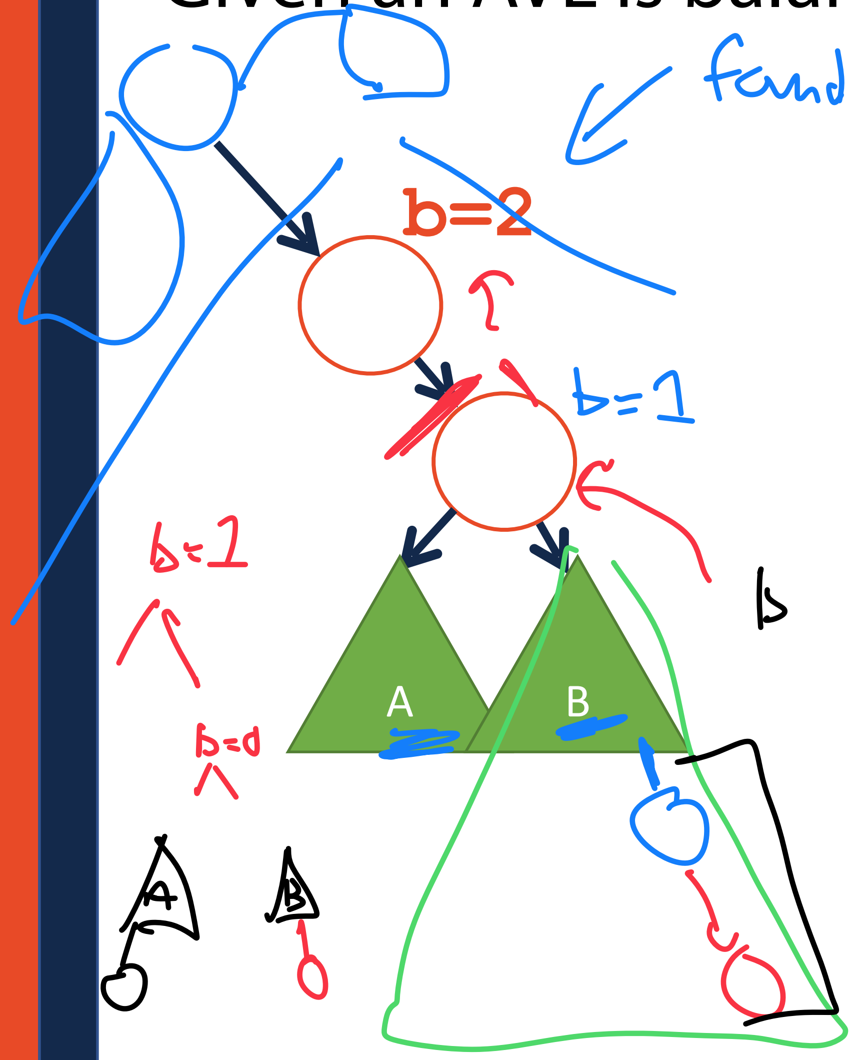
insert & height changes

AVL Insertion

Given an AVL is balanced, insert can insert **at most** one imbalance

imbalance

found an imbalance in the lowest height subtree



Claim: Insert in B produces 2, 1 balance pattern

↳ Imagine height(A) > height(B)

↳ Contradiction — can't insert into smaller subtree and increase of root

height(B) > height(A)

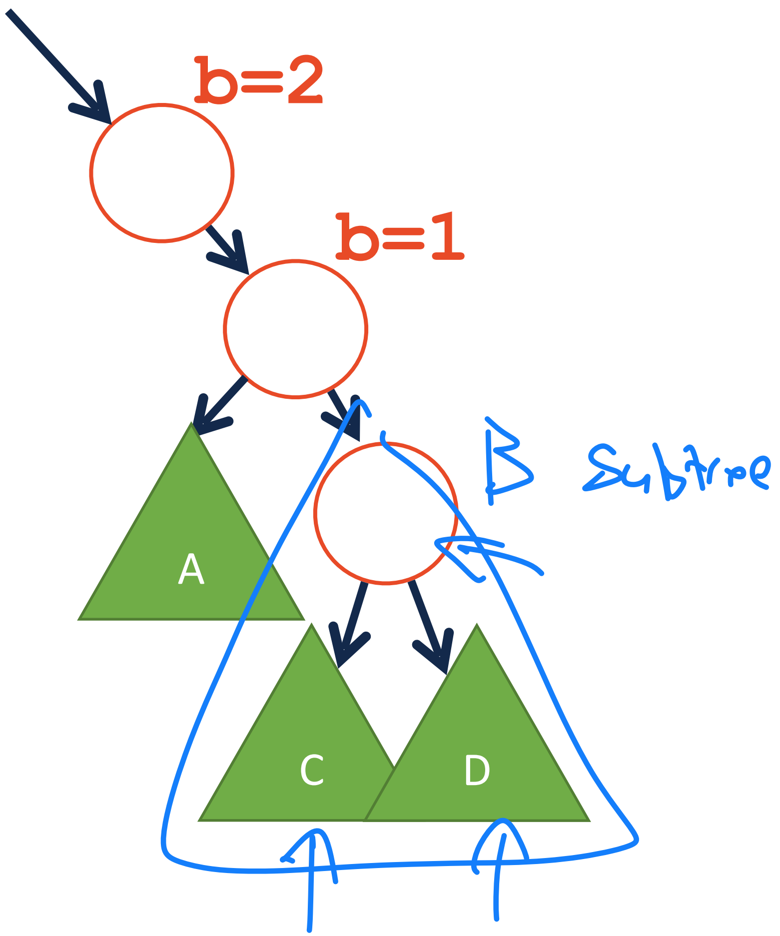
↳ Contradiction — if B was larger and insert increased height my imbalance root would be lower

height(A) = height(B)

AVL Insertion

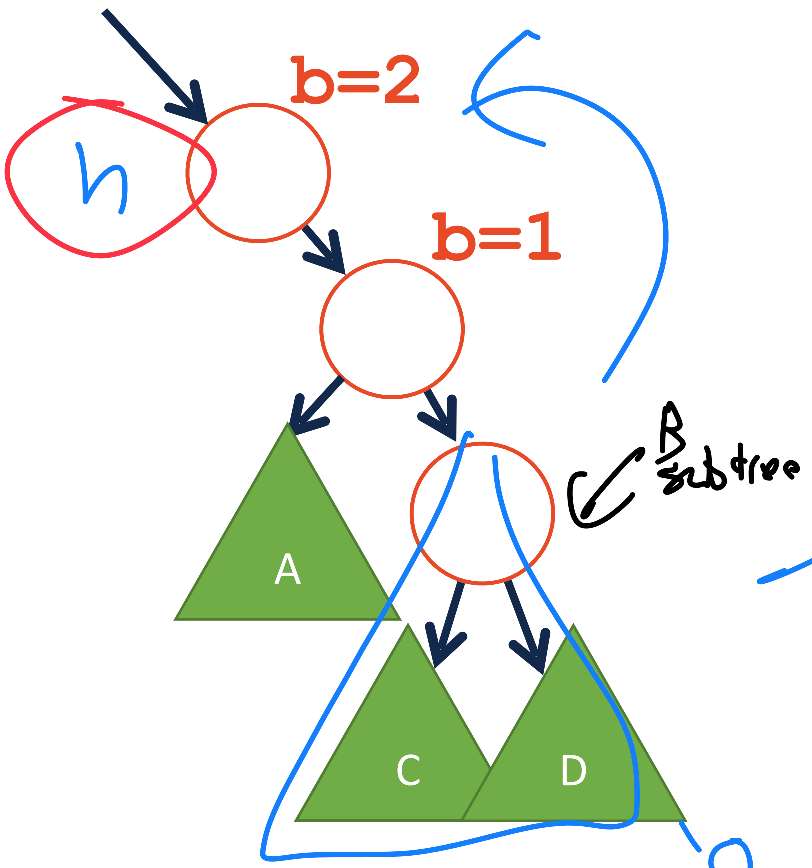
If we insert in B, I must have a balance pattern of **2, 1**

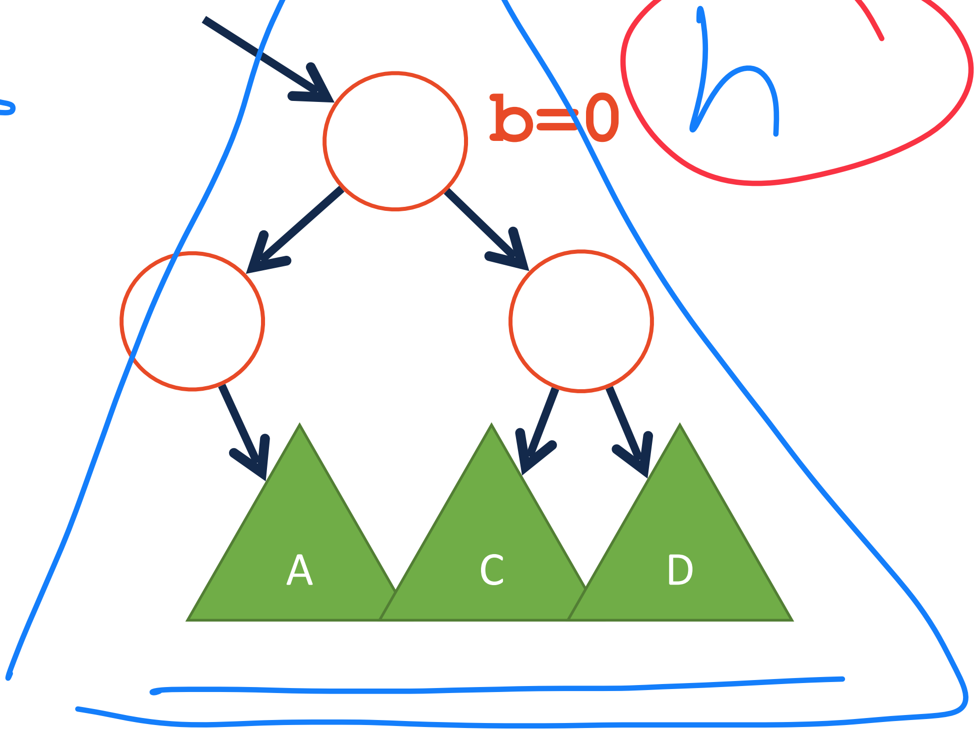
↳ left rotation



AVL Insertion

A **left** rotation fixes our imbalance in our local tree.

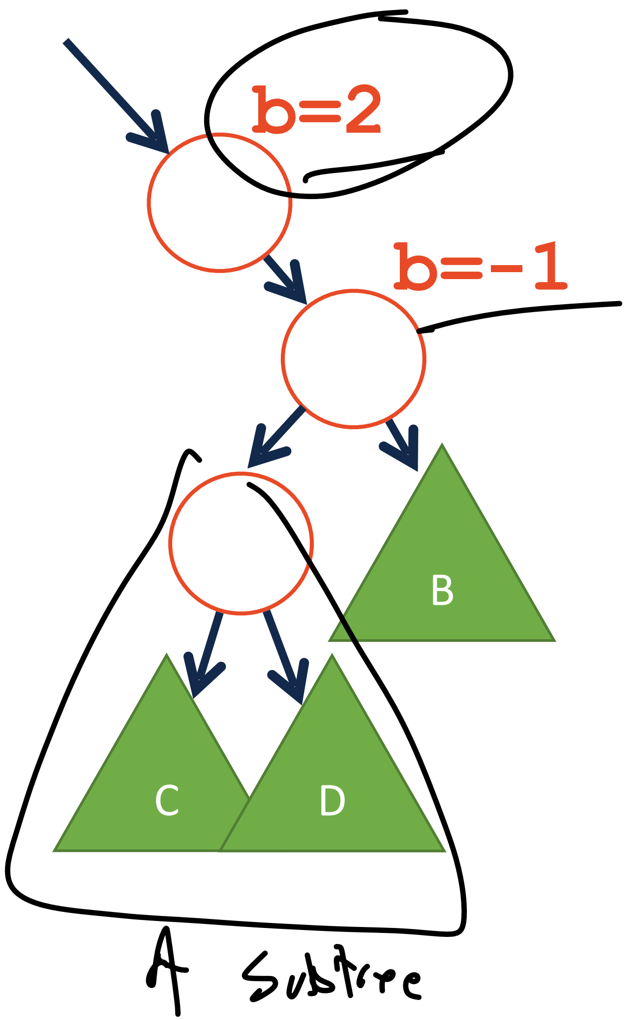




After rotation, subtree has **pre-insert height**. (Overall tree is balanced)

AVL Insertion

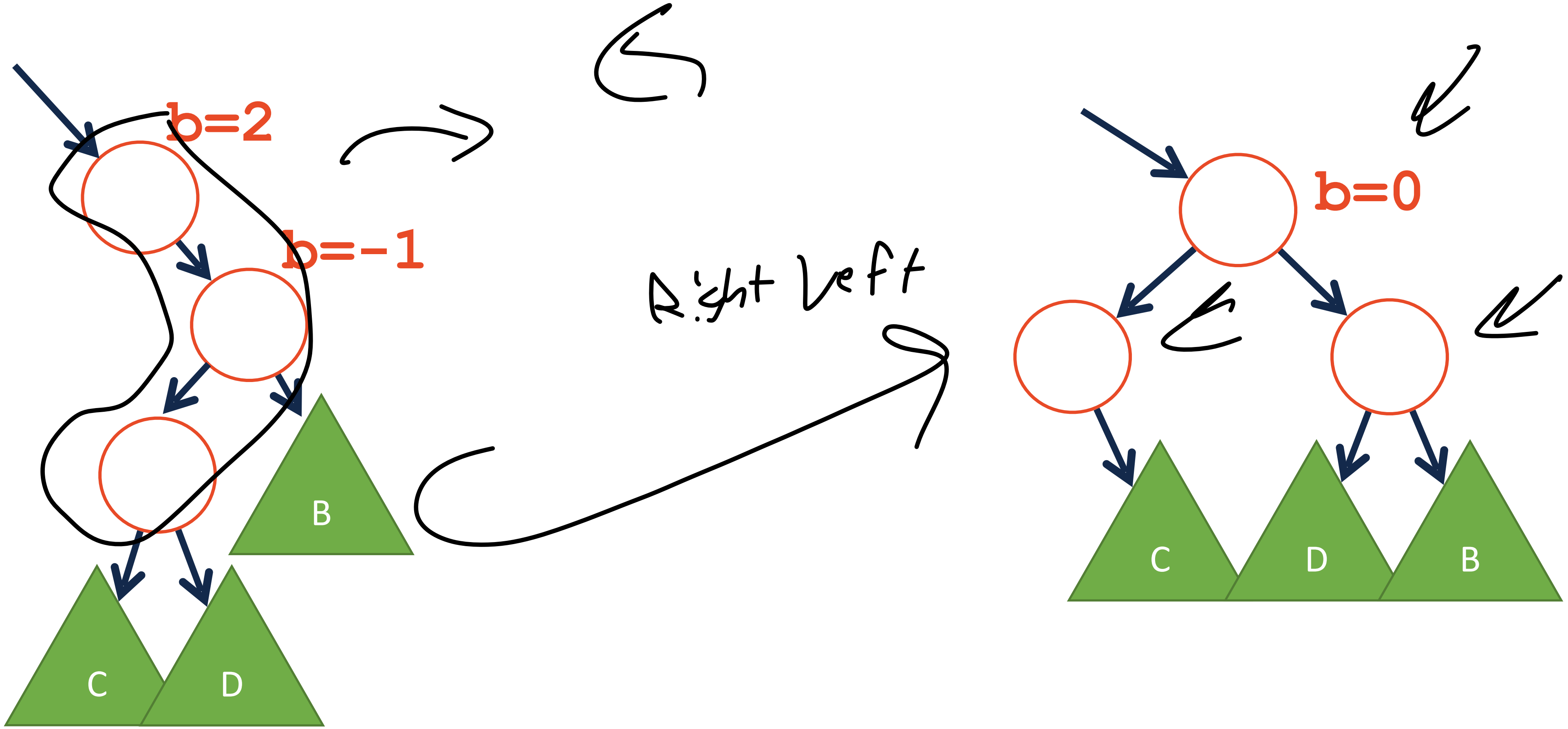
If we insert in A, I must have a balance pattern of **2, -1**



2, -1 must be pattern if insert into subtree A & height increases & original balance is 2

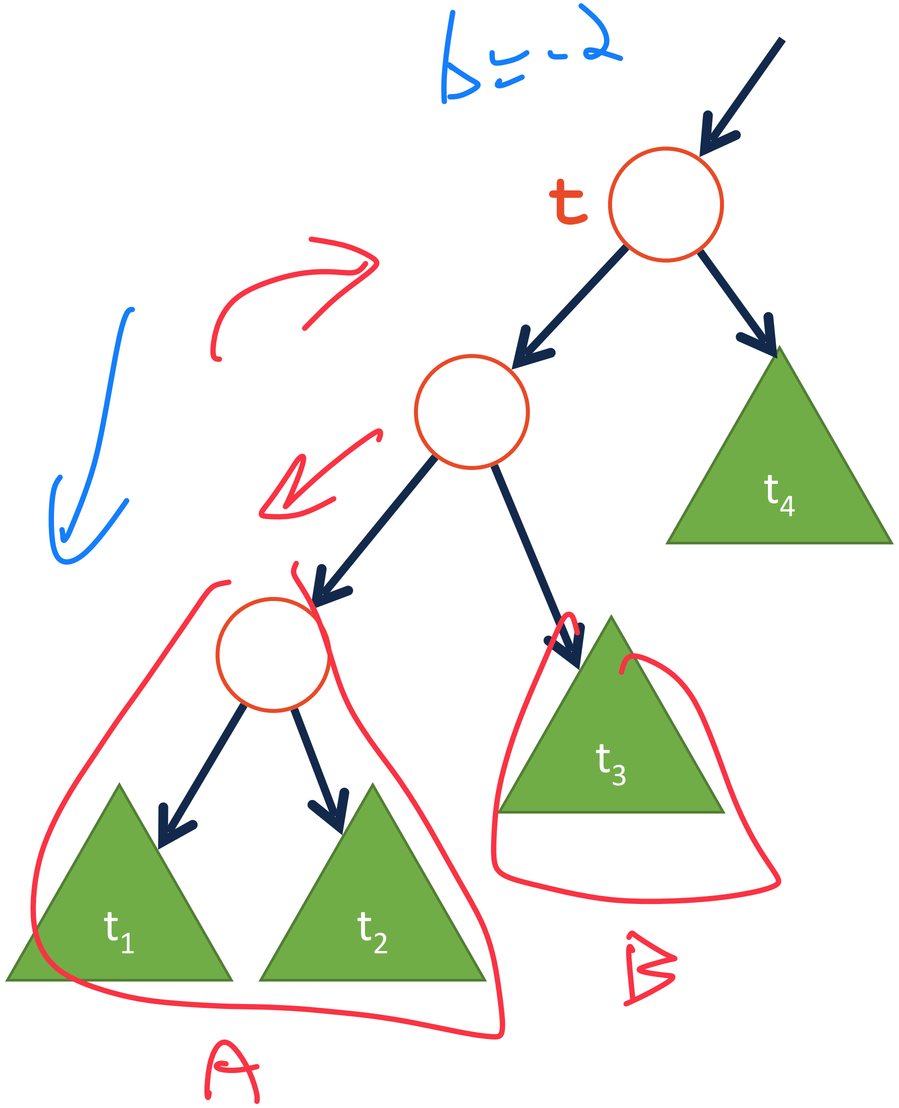
AVL Insertion

A **rightLeft** rotation fixes our imbalance in our local tree.



After rotation, subtree has **pre-insert height**. (Overall tree is balanced)

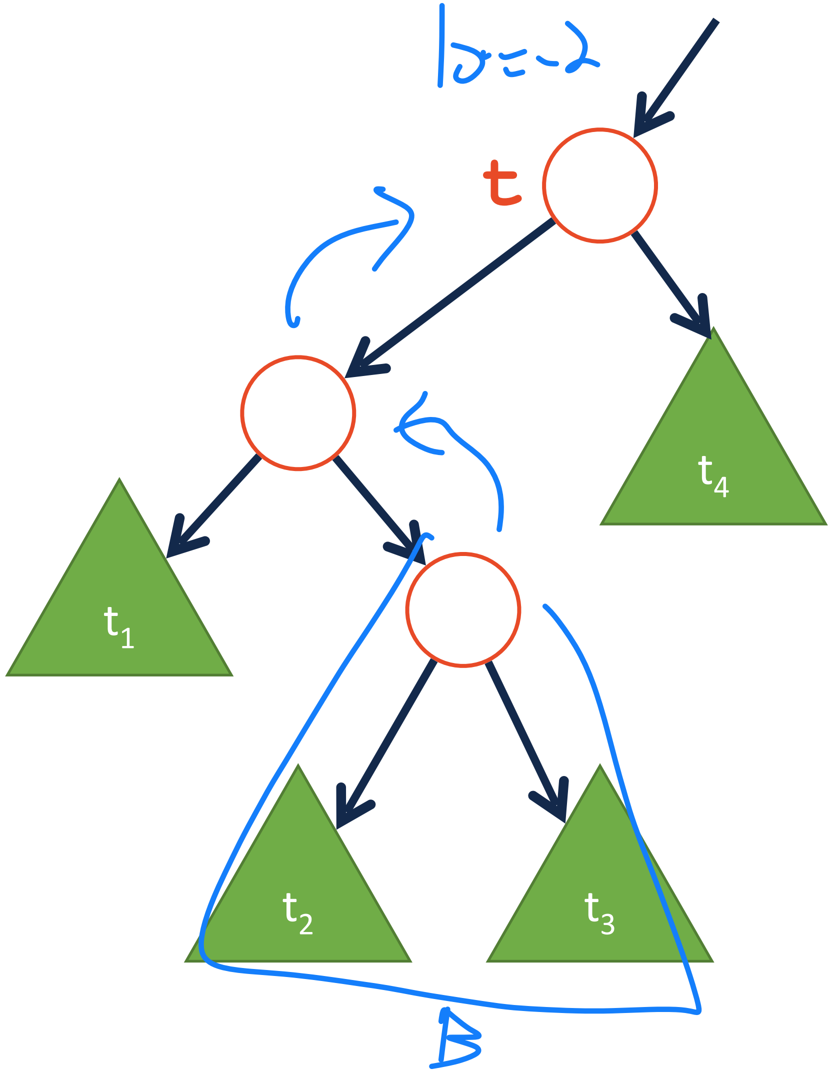
AVL Insertion



Theorem:
If an insertion occurred in subtrees $\mathbf{t_1}$ or $\mathbf{t_2}$ and an imbalance was first detected at **t**, then a ___right_________ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** __-2____ and the balance factor of **t->left** is __-1____.

AVL Insertion



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

We gauge this by noting the balance factor of **t is** -2 _______ and the balance factor of **t->left** is 1 _______.

AVL Insertion

We've seen every possible insert that can cause an imbalance

Insert increases height by at most: 1 (can be 0)

A rotation reduces the height of the subtree by: 1

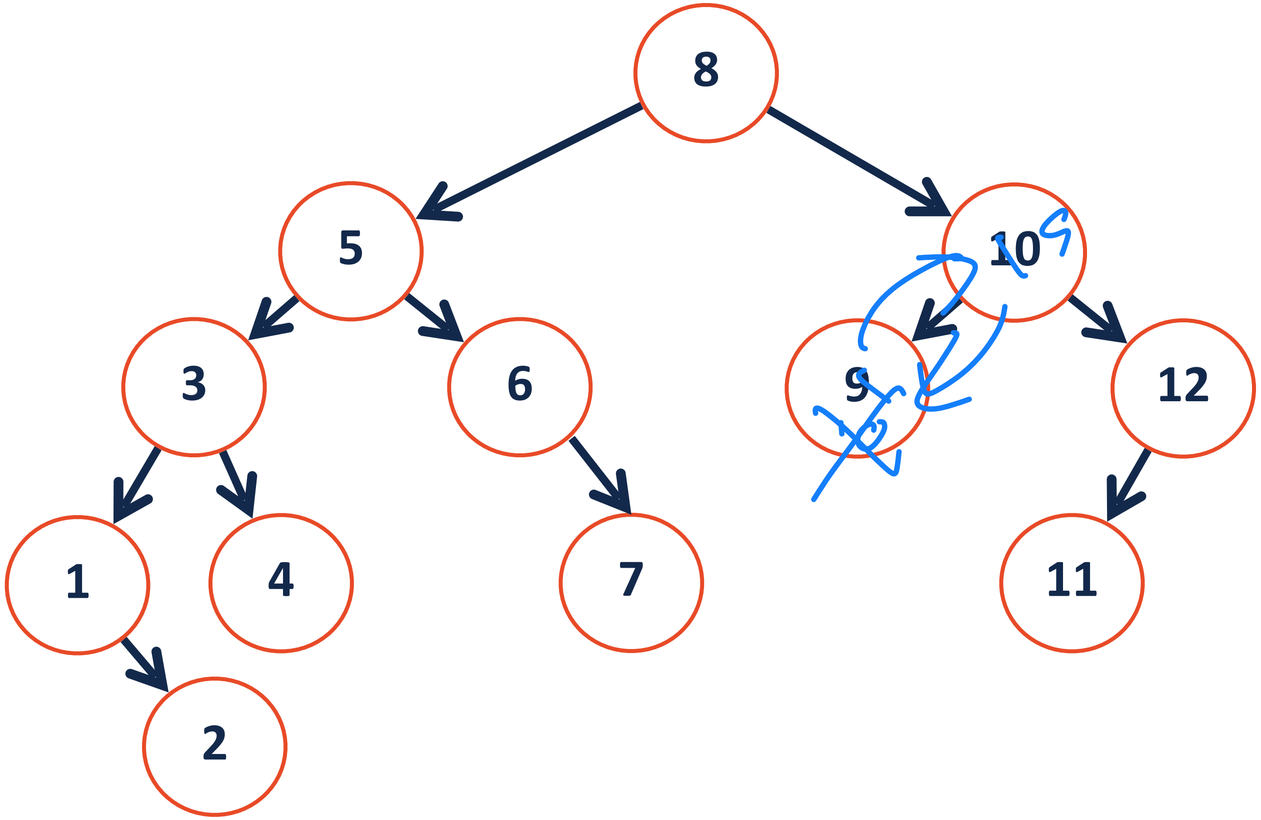
A single* rotation restores balance and corrects height!

double rotations "Right Left" "Left Right" counts as one rotation

O(1)

AVL Remove

`_remove(10)`

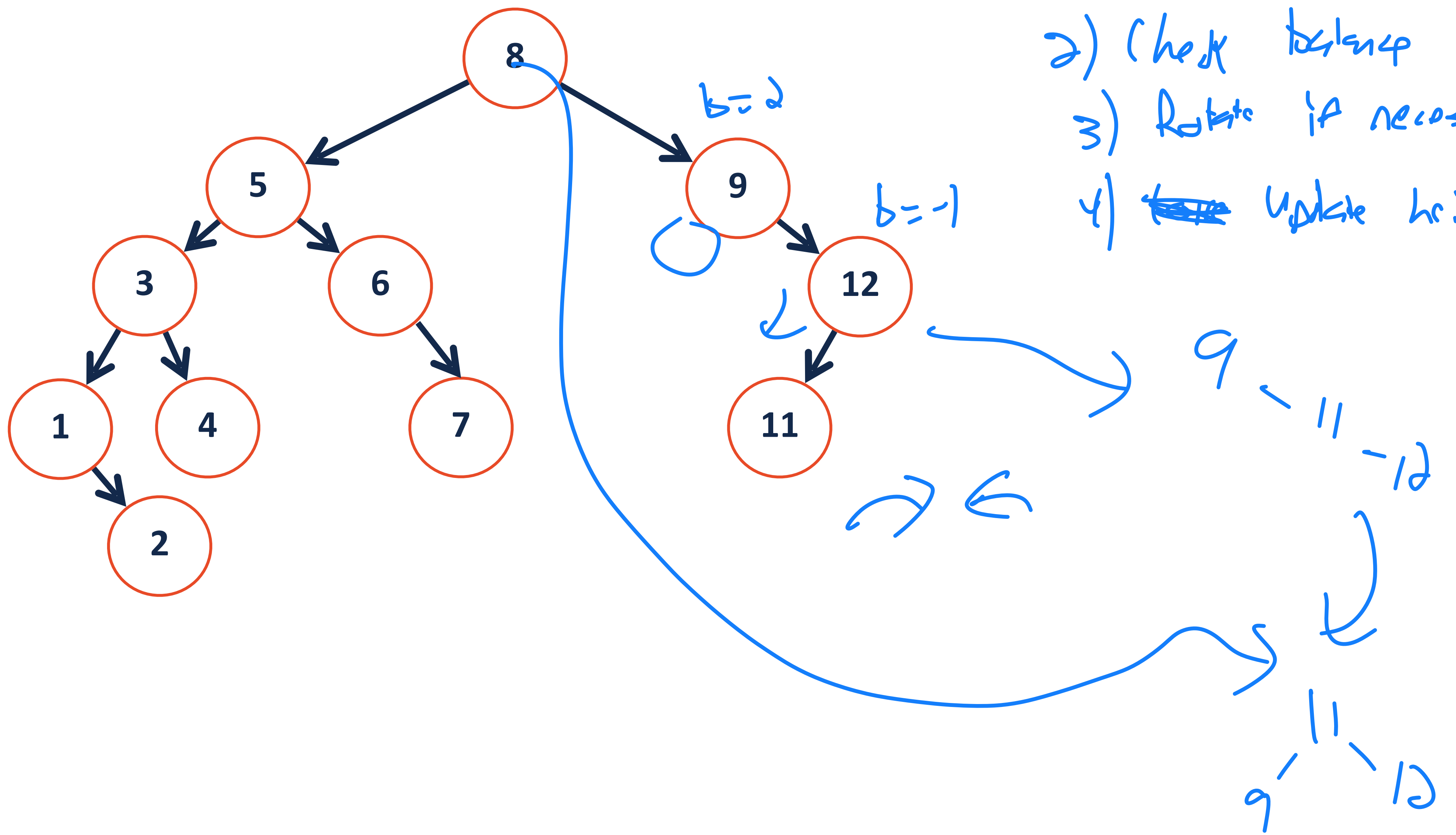


1) find (10)
→ IOP
↳ find(9)
↳
2) delete (10)

AVL Remove

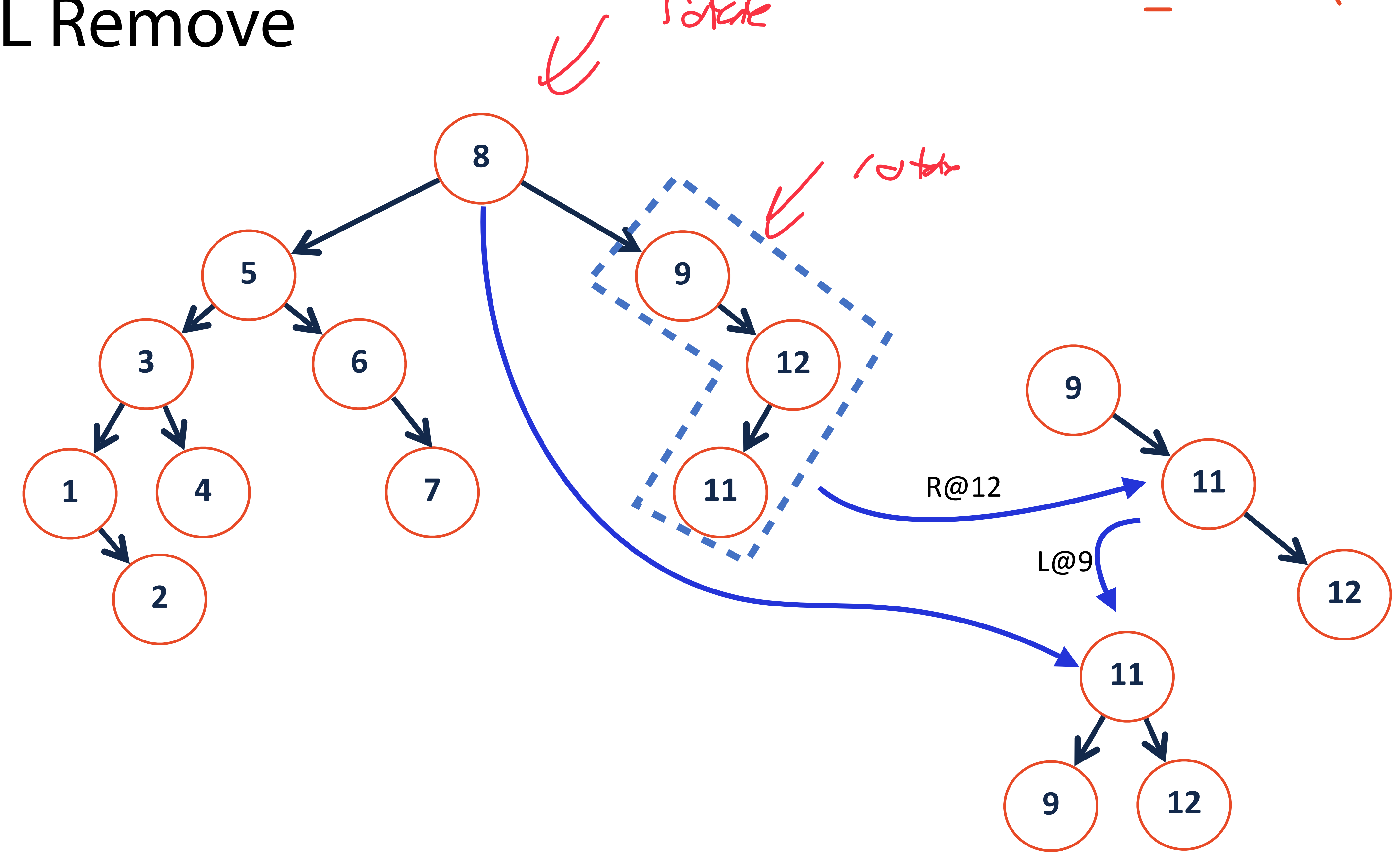
_remove(10)

1) Standard BST delete
2) Check balance
3) Rotate if necessary
4) Update height



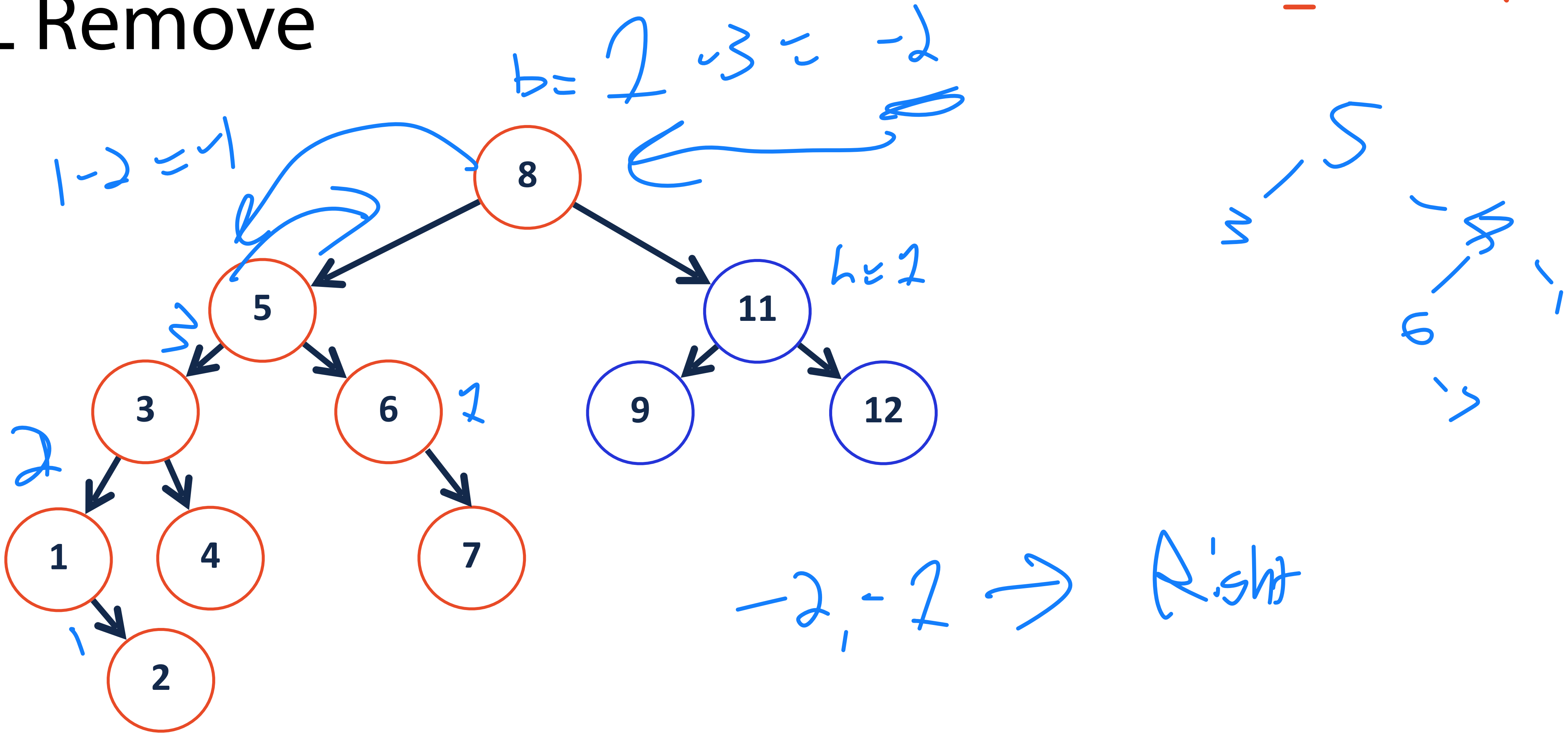
AVL Remove

`_remove(10)`



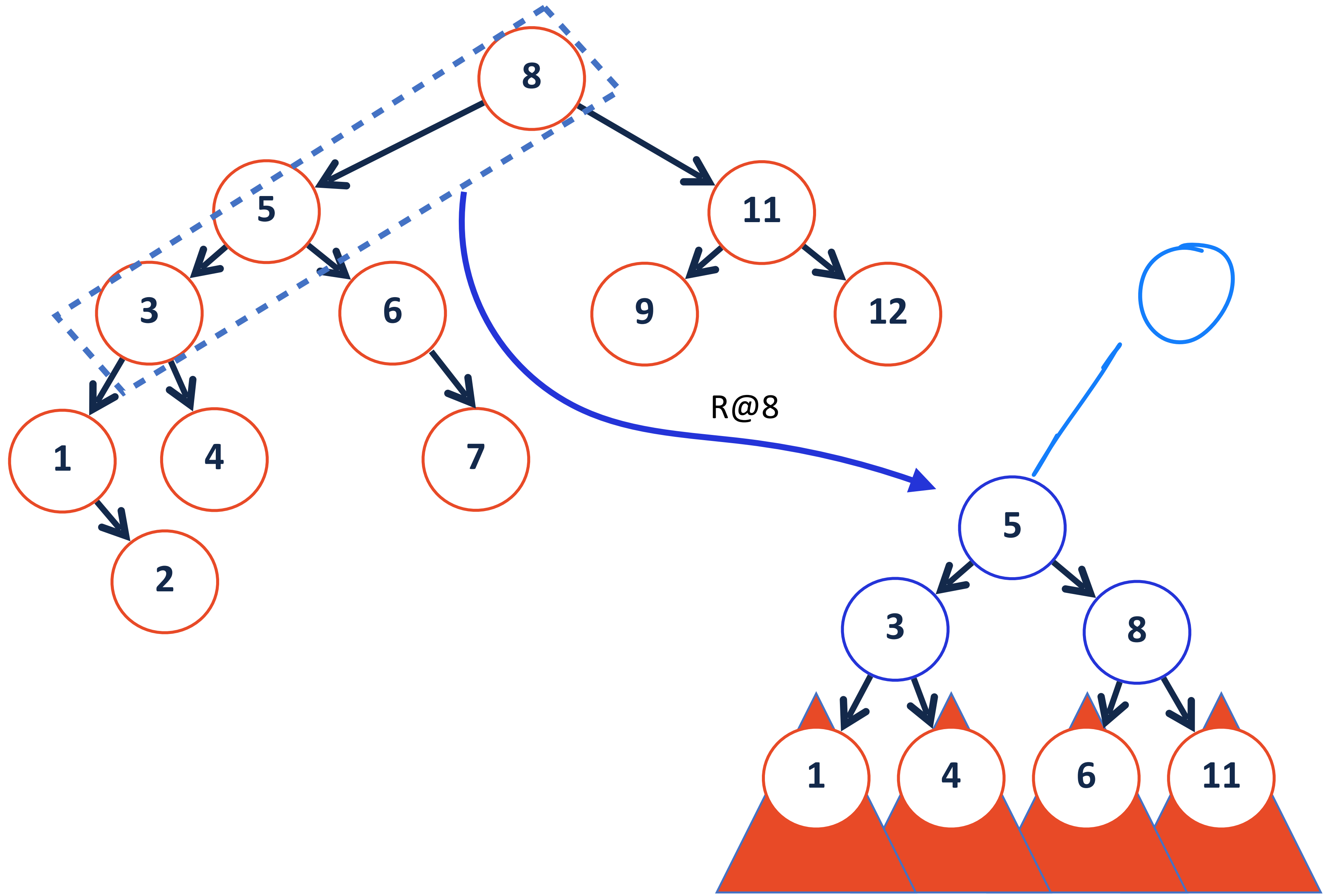
AVL Remove

`_remove(10)`

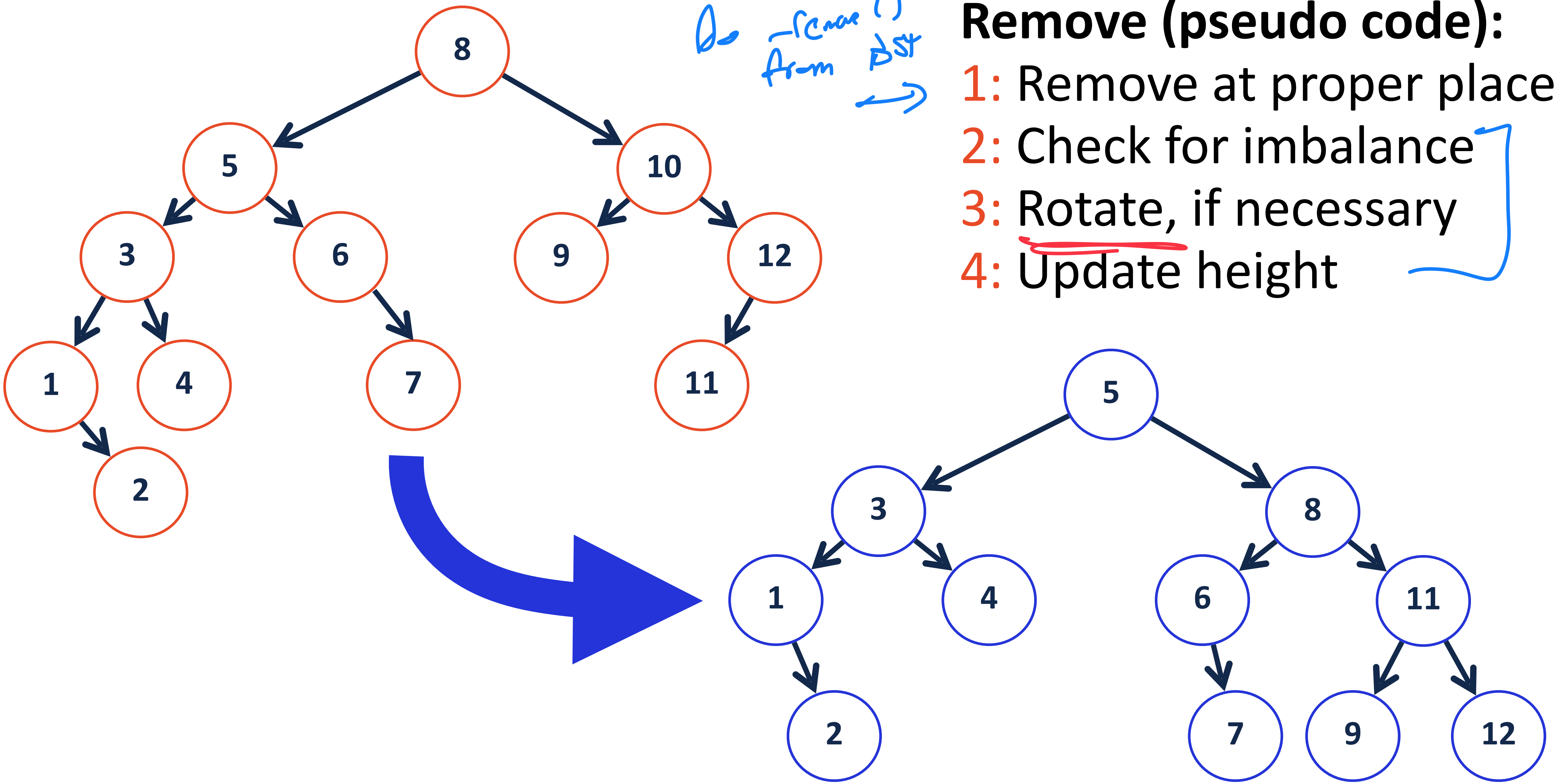


AVL Remove

`_remove(10)`



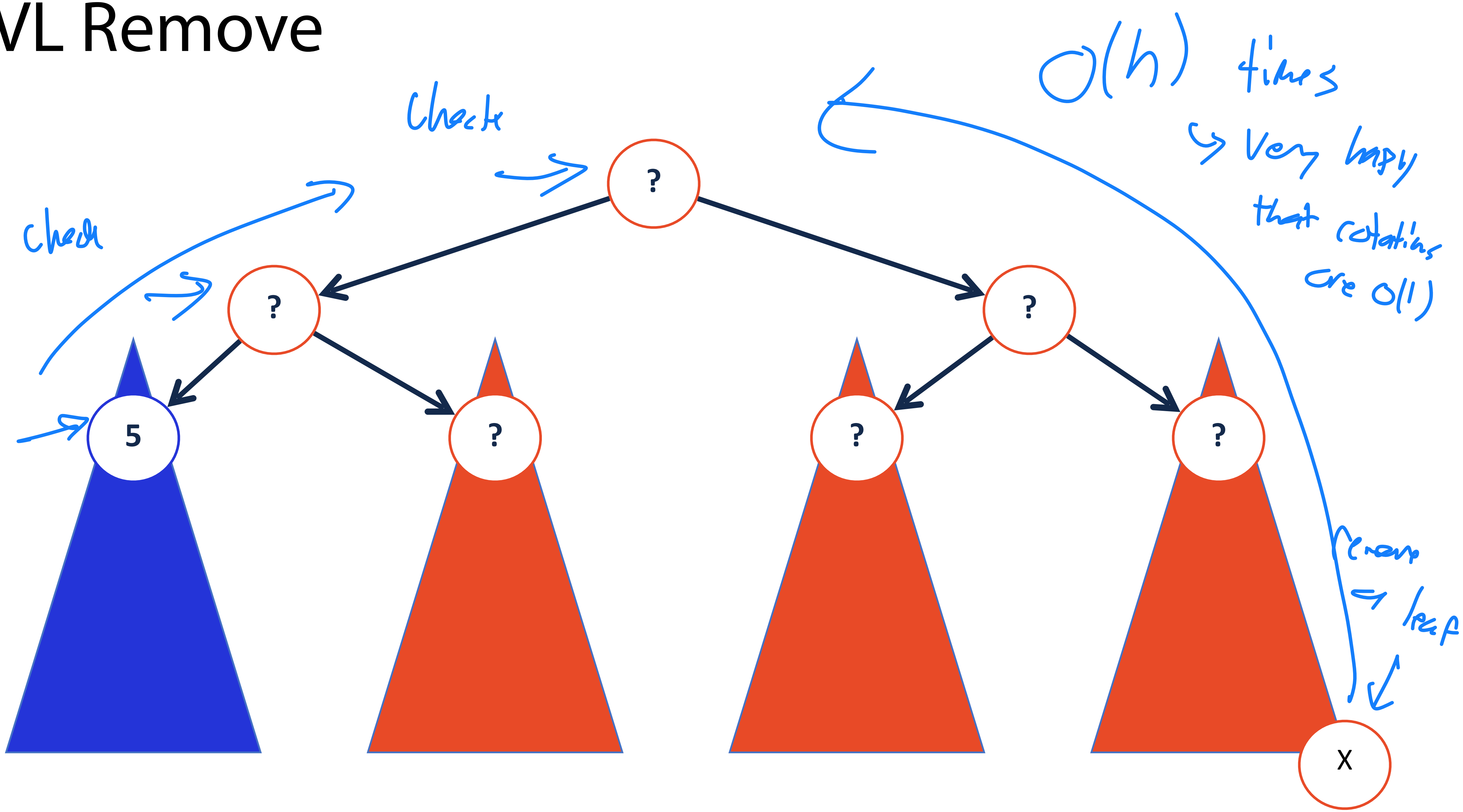
AVL Remove



Remove (pseudo code):

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

AVL Remove



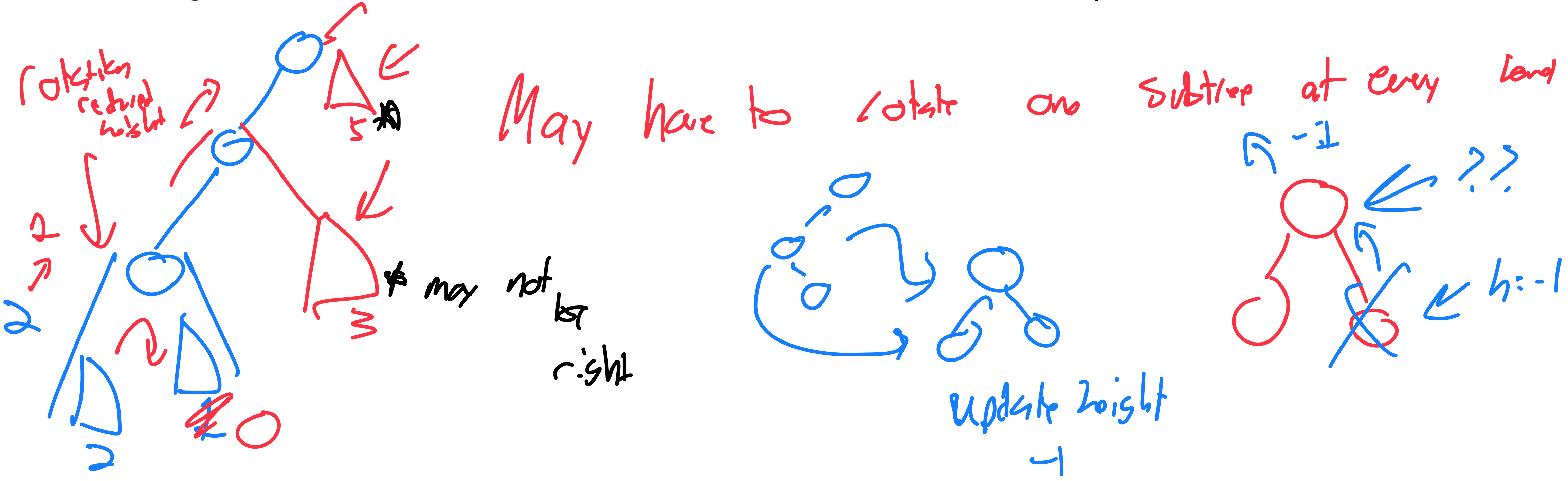
?
?
?
5
?
?
?
X
check
check
O(h) times
→ very happy that rotations are O(1)
remove → leaf

AVL Remove

An AVL remove step can reduce a subtree height by at most: 1

But a rotation ***reduces*** the height of a subtree by one!

We might have to perform a rotation at every level of the tree!



AVL Tree Analysis

For an AVL tree of height h:

Find runs in: $O(h)$.

Insert runs in: $O(h)$.

Remove runs in: $O(h)$.

Claim: The height of the AVL tree with n nodes is: $O(\log n)$.

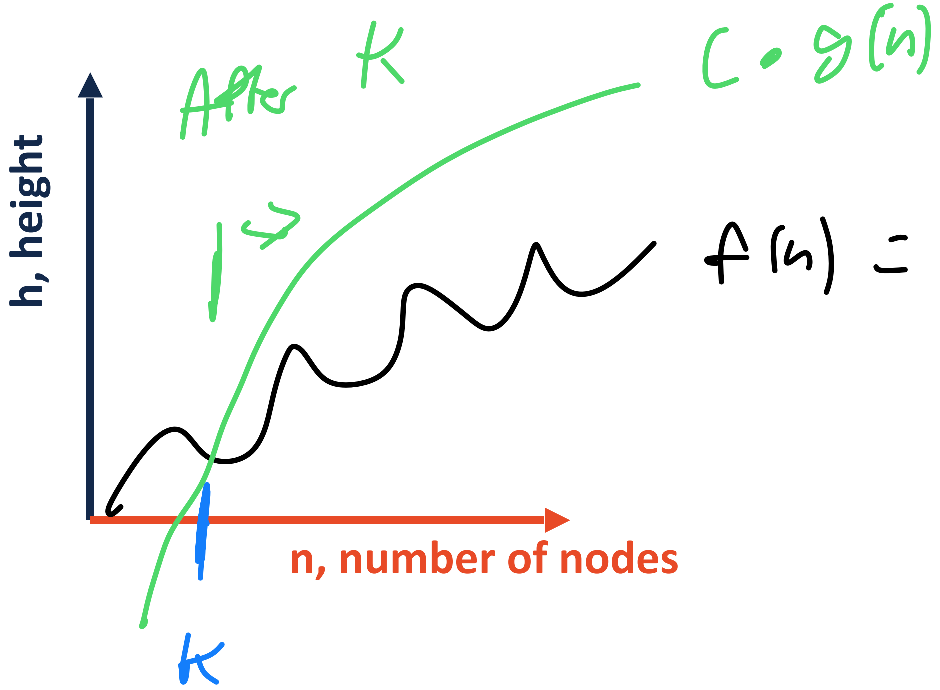
AVL Tree Analysis

Definition of big-O:

$$f(n) \text{ is } O(g(n)) \text{ iff } \exists c, k \text{ s.t. } f(n) \le cg(n)\ \forall n > k$$

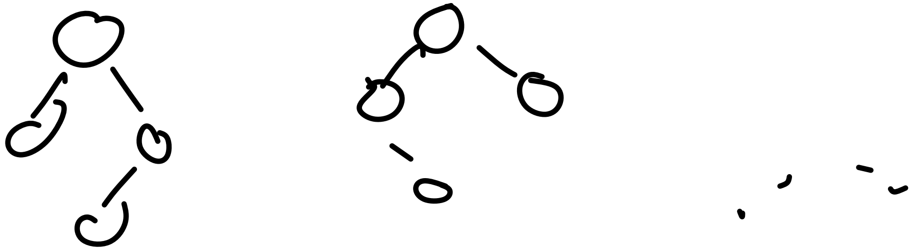
constant

...or, with pictures:

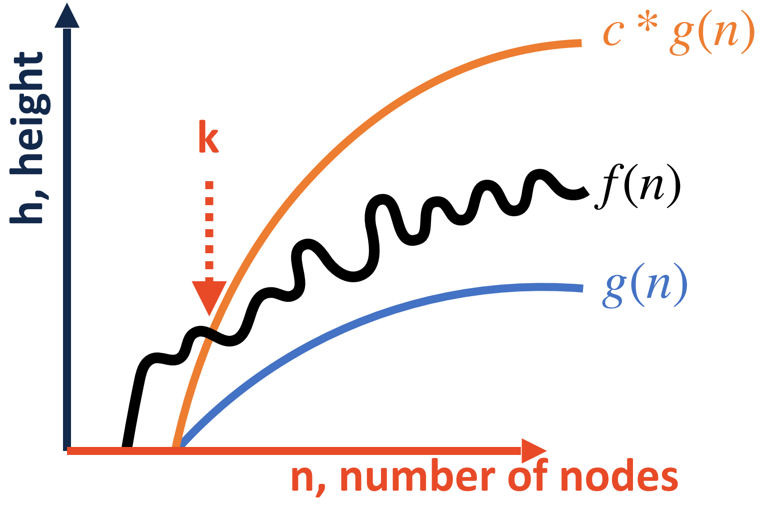


f(n) = Tree height given n nodes

∀ nodes

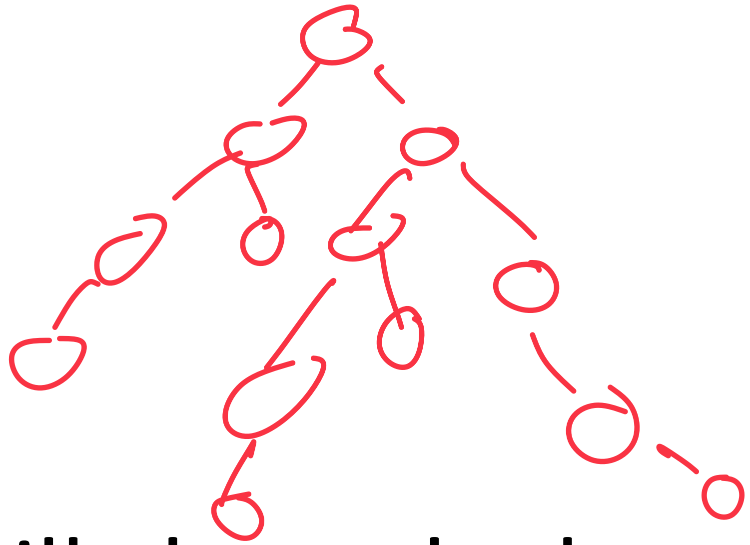


AVL Tree Analysis



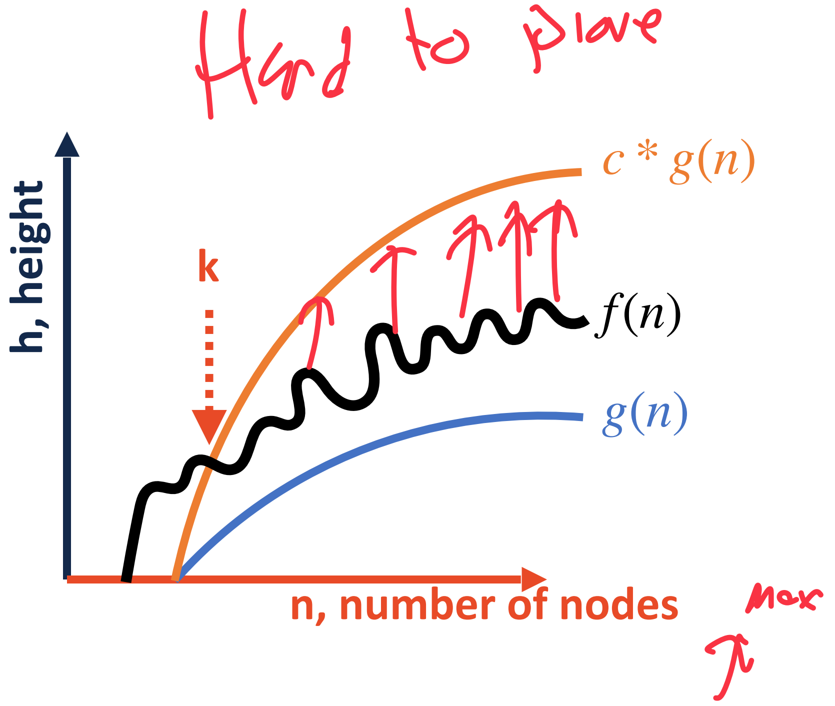
Max height for n nodes

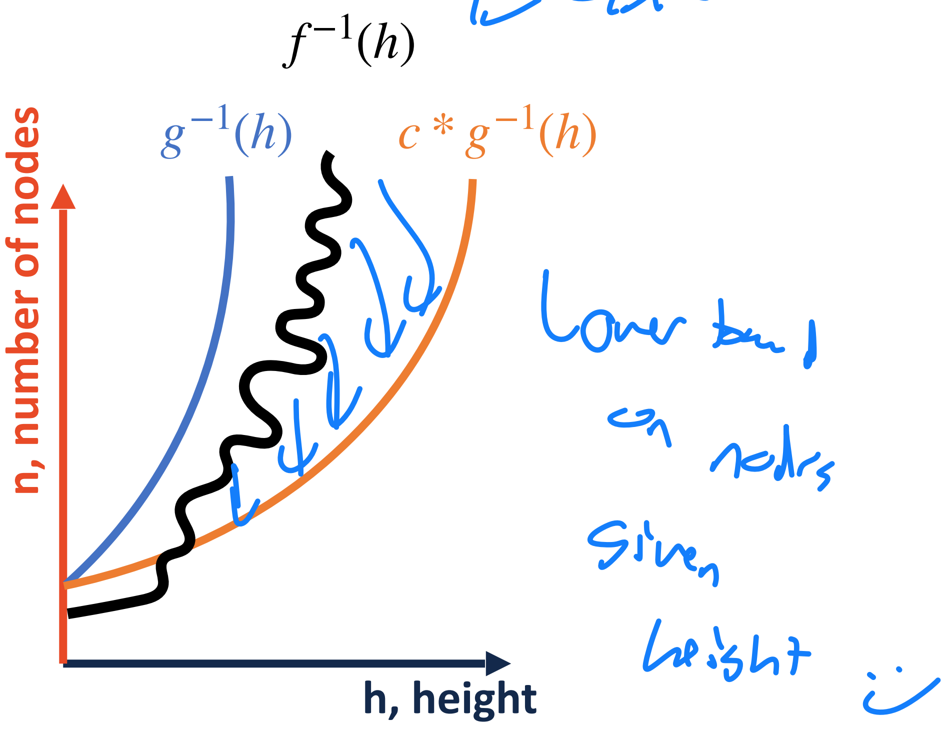
n = 10000...



The height of the tree, **f(n)**, will always be less than **c × g(n)** for all values where **n > k**.

AVL Tree Analysis





$f(n)$ = "Tree height given nodes"

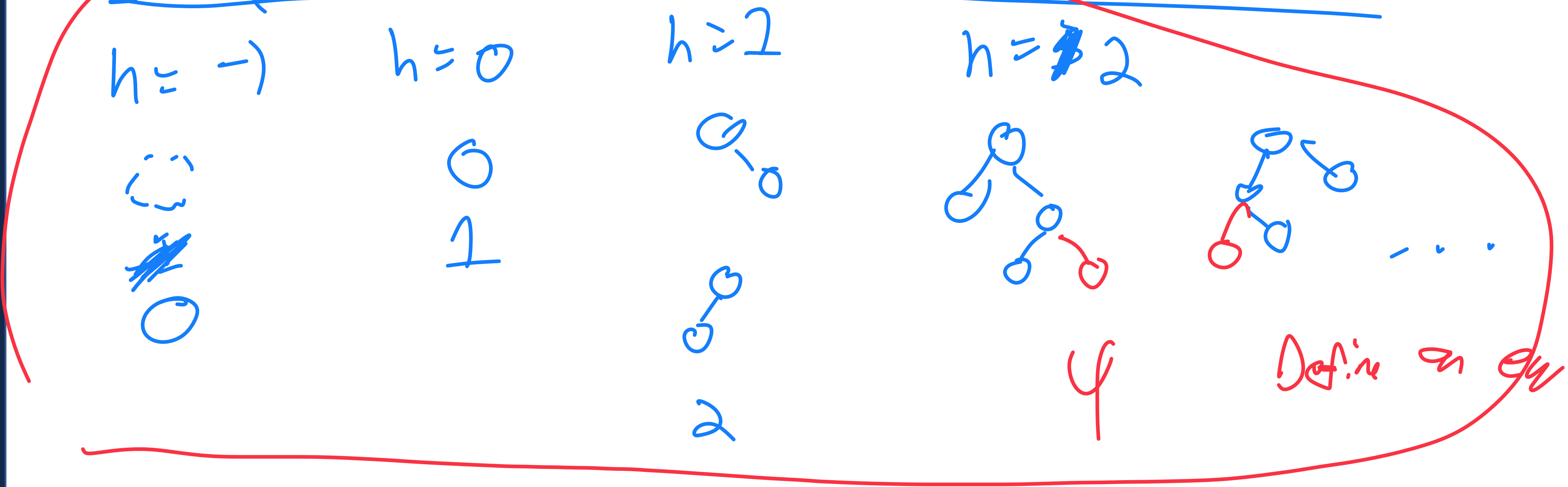
$f^{-1}(h)$ = "Nodes in tree given height"

The number of nodes in the tree, $\mathbf{f^{-1}(h)}$, will always be greater than $\mathbf{c \times g^{-1}(h)}$ for all values where $\mathbf{n > k}$.

Plan of Action

Since our goal is to find the lower bound on **n** given **h**, we can begin by defining a function given **h** which describes the smallest number of nodes in an AVL tree of height **h**:

$N(h)$ = minimum number of nodes in an AVL tree of height h



Simplify the Recurrence

$$N(h) = 1 + N(h-1) + N(h-2)$$

State a Theorem

Theorem: An AVL tree of height h has at least ___________.

Proof by Induction:

I. Consider an AVL tree and let **h** denote its height.

II. Base Case: ________________

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

III. Base Case: ________________

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

IV. Induction Case: ________________

Assume for all heights $i < h$, $N(i) \geq 2^{i/2}$. Prove that $N(h) \geq 2^{h/2}$

Prove a Theorem

V. Using a proof by induction, we have shown that:

…and inverting:

AVL Runtime Proof

An upper-bound on the height of an AVL tree is **O(lg(n))**:

N(h) := Minimum # of nodes in an AVL tree of height h

N(h) = 1 + **N(h-1)** + **N(h-2)**

$> 1 + 2^{(h-1)/2} + 2^{(h-2)/2}$

$> 2 \times 2^{(h-2)/2} = 2^{(h-2)/2+1} = 2^{h/2}$

Theorem #1:

Every AVL tree of height h has at least $2^{h/2}$ nodes.

AVL Runtime Proof

An upper-bound on the height of an AVL tree is **O(lg(n))**:

# of nodes (n) ≥ N(h) > $2^{h/2}$

n > $2^{h/2}$

lg(n) > h/2

2 × lg(n) > h

h < 2 × lg(n) ***, for h ≥ 1***

Proved: The maximum number of nodes in an AVL tree of height h is less than 2 × lg(n).