CS 225

Data Structures

March 17 – AVL Analysis

Brad Solomon

Informal Early Feedback Reminder

CS 225 All SP21: Data Structures (Evans, C)

Dashboard / My courses / CS 225 All SP21

- Announcements
- Lab Attendance
- Informal Early Feedback
- Lab Informal Early Feedback

Final Project Team Formation Survey

What is your time zone in Coordinated Universal Time (UTC)? Paste in your browser for UTC: https://www.timeanddate.com/time/map/

- ☐ UTC -11:00
- ☐ UTC -10:00 (US Hawaii)
- ☐ UTC -9:00 (US Alaska)
- ☐ UTC -8:00 (US Pacific, British Columbia, Baja)
- ☐ UTC -7:00 (US Mountain, Alberta, W. Mexico)
- ☐ UTC -6:00 (US Central, E. Mexico, Manitoba)
- ☐ UTC -5:00 (US Eastern, Colombia, Quebec)

What is your gender? [Make a selection]

Please indicate the racial/ethnic group with which you most identify: [Make a selection]

Please check the times that you are in class, at work or practice and are **busy and unavailable** for group work:

(You may select entire rows or columns by clicking the column/row headers)

By default, students can retake Team Maker surveys up until it is closed (midnight of the 'End Date'). If you need to change your schedule after submitting this survey, you will be allowed to retake the survey to update your schedule.

Make Busy	Mon	Tue	Wed	Thu	Fri	Sat	Sun
8:00am	☐	☐	☐	☐	☐	☐	☐
9:00am	☐	☐	☐	☐	☐	☐	☐
10:00am	☐	☐	☐	☐	☐	☐	☐
11:00am	☐	☐	☐	☐	☐	☐	☐
12:00pm	☐	☐	☐	☐	☐	☐	☐
1:00pm	☐	☐	☐	☐	☐	☐	☐
2:00pm	☐	☐	☐	☐	☐	☐	☐
3:00pm	☐	☐	☐	☐	☐	☐	☐
4:00pm	☐	☐	☐	☐	☐	☐	☐
5:00pm	☐	☐	☐	☐	☐	☐	☐
6:00pm	☐	☐	☐	☐	☐	☐	☐
7:00pm	☐	☐	☐	☐	☐	☐	☐
8:00pm	☐	☐	☐	☐	☐	☐	☐
9:00pm	☐	☐	☐	☐	☐	☐	☐

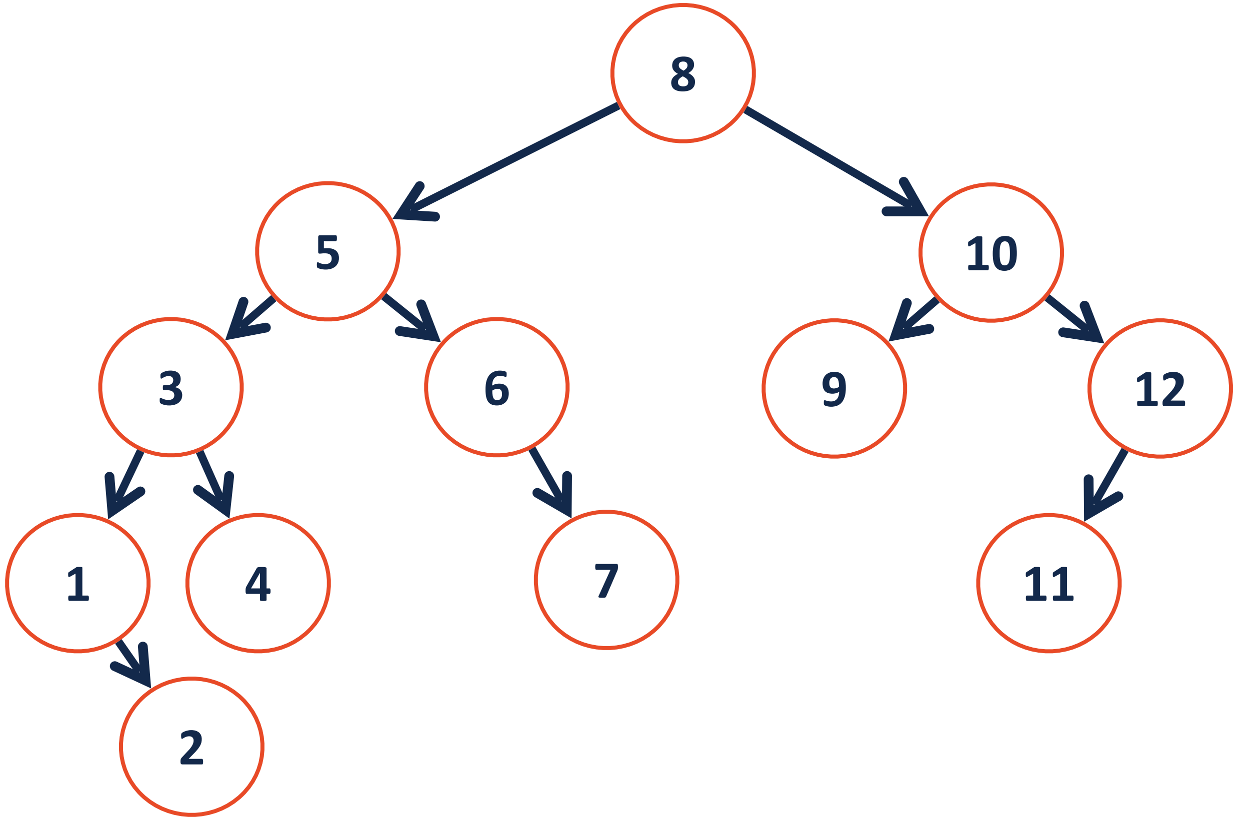
If you have formed a team already, what is your team's UUID? (Be sure to submit identical IDs!)

Learning Objectives

- Review AVL trees
- Formalize code for _insert and generalize to _find and _remove
- Quantify efficiency of AVL tree operations as a factor of h
- Develop strategies for formalizing h as a mathematical expression

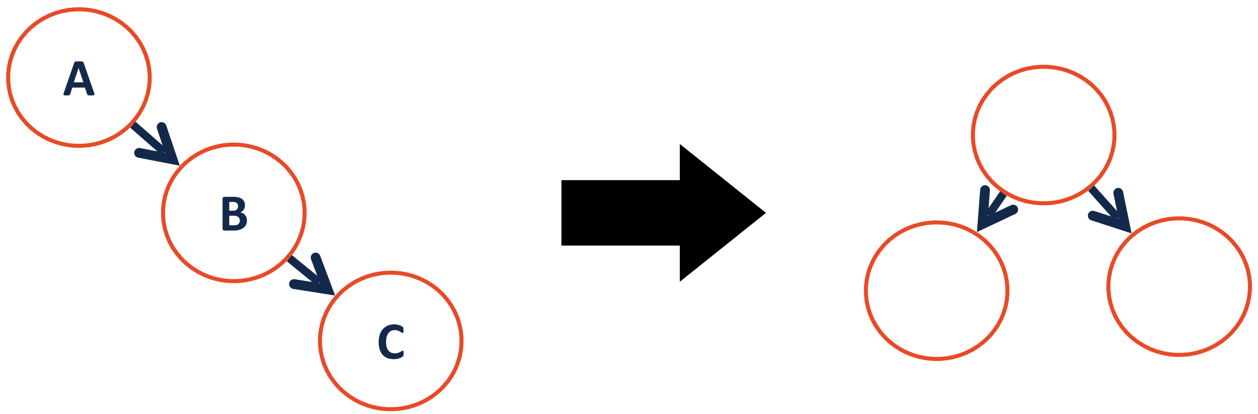
AVL TreeNode

AVL is a BST that maintains balance



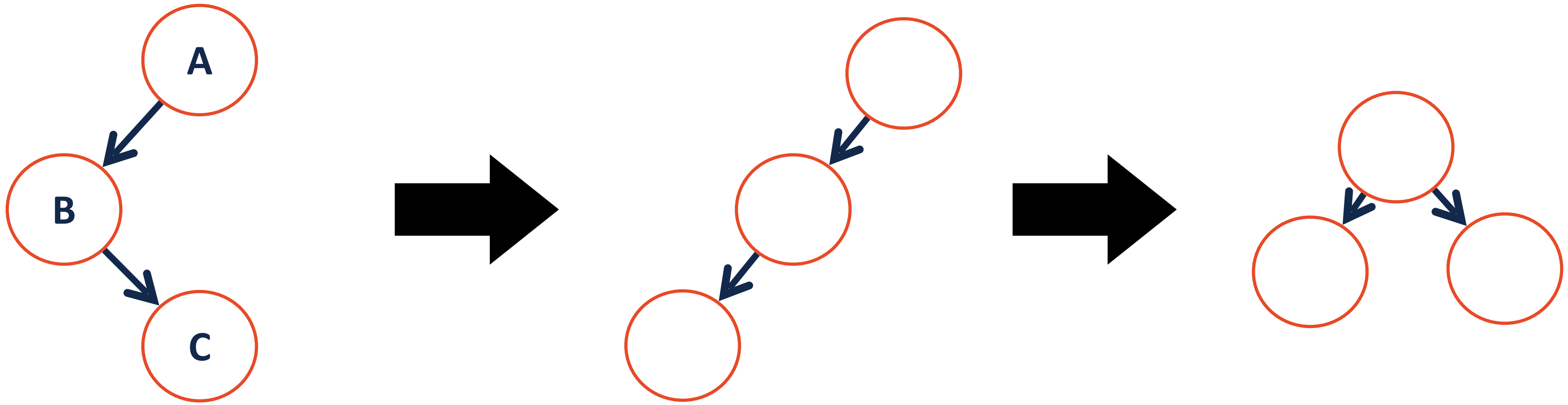

```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

AVL Tree Rotations



All rotations are O(1)

All rotations reduce subtree height by one

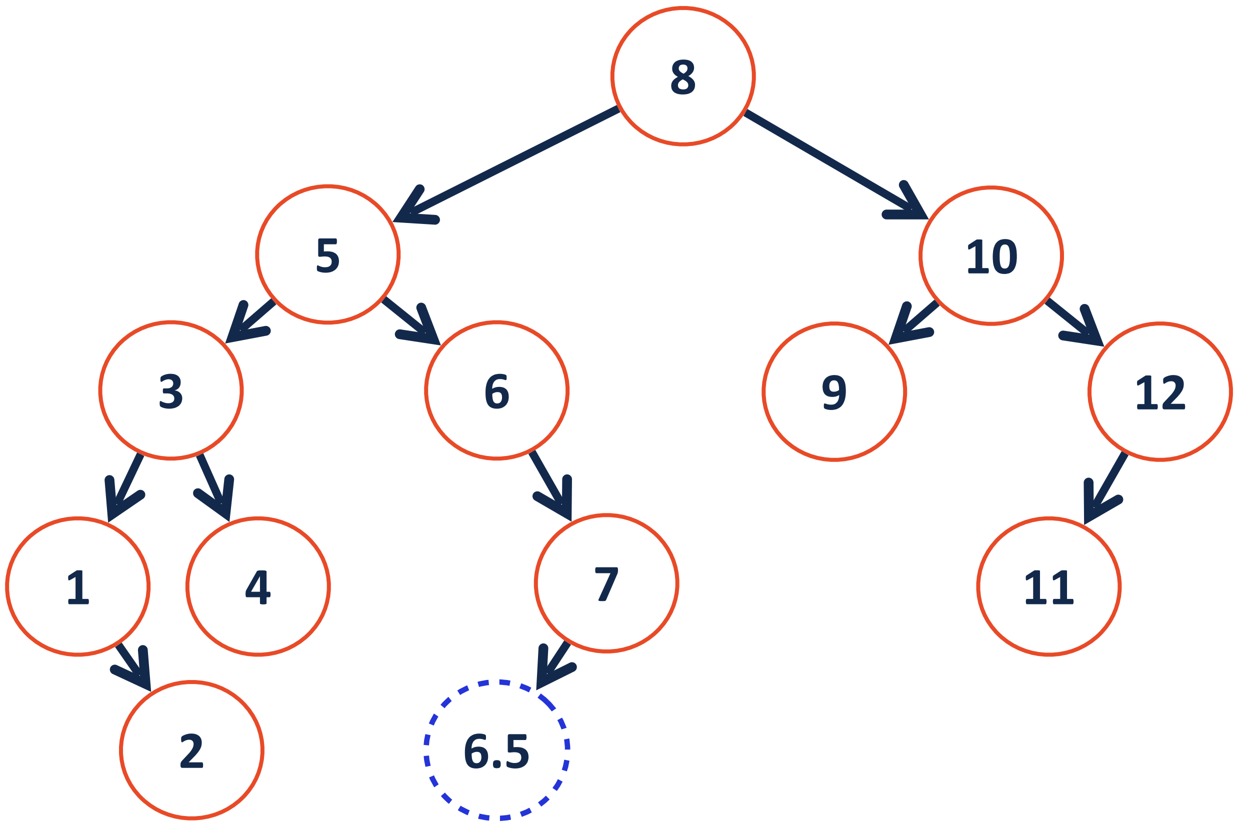


Insertion into an AVL Tree

`_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;
};
```



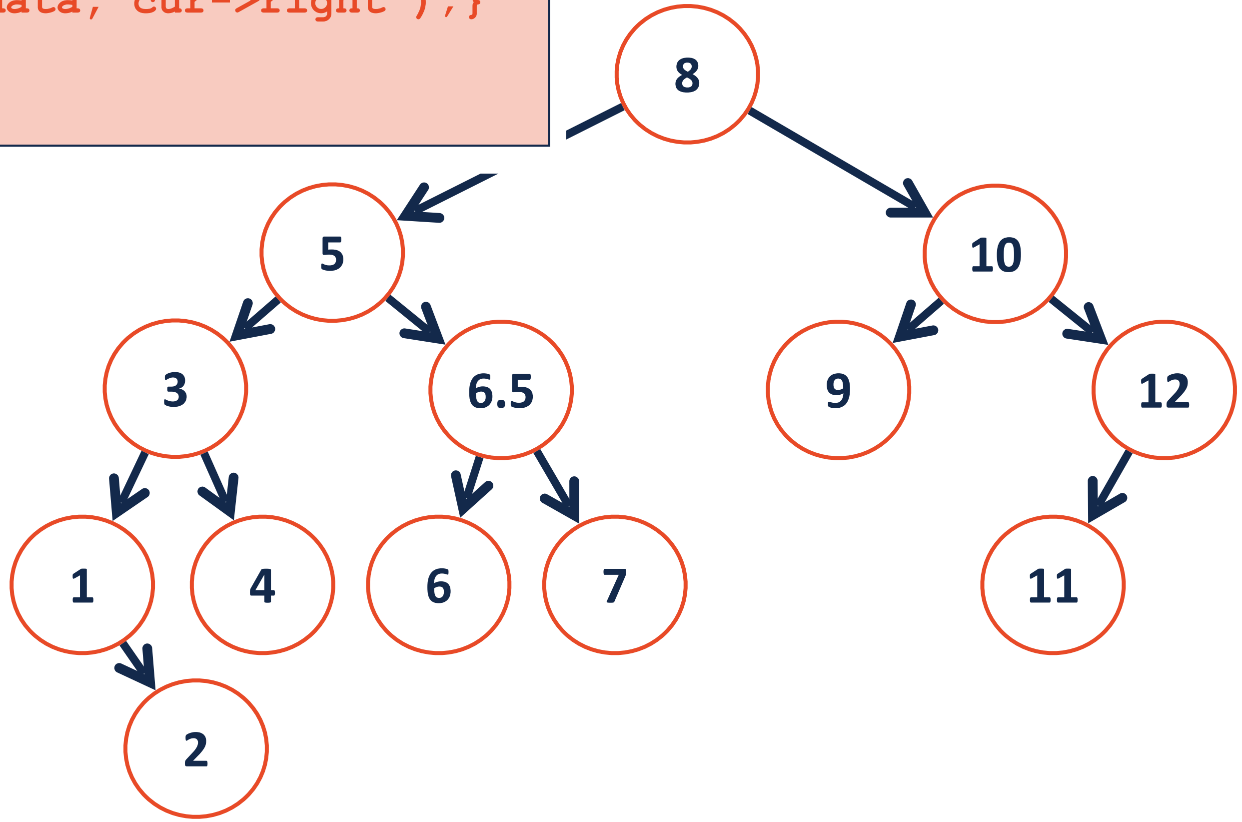
Insertion into an AVL Tree

`_insert(6.5)`

```
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);
}
```

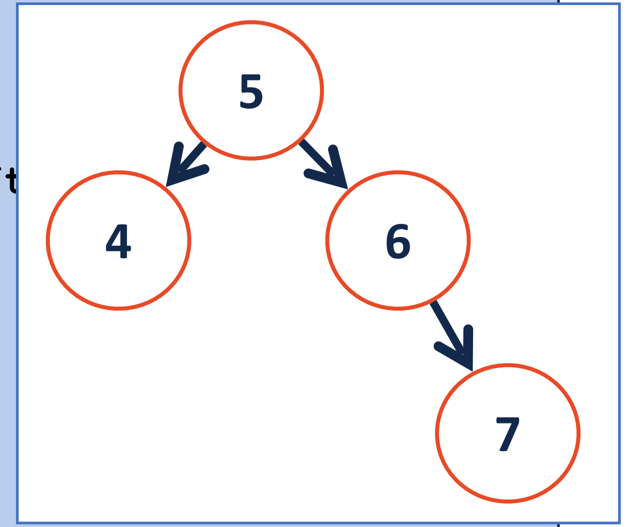
amethyst_cat2: can we call ensurebalance multiple times for one insert?

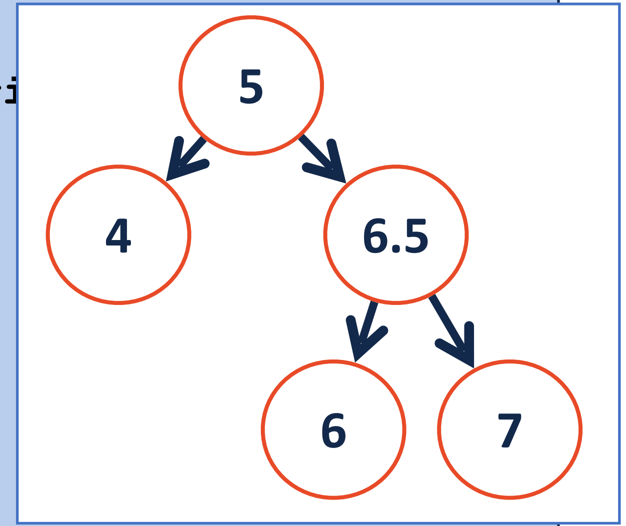
```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```



```
template <class T> void AVLTree<T>::_insert(const T & x, treeNode<T> * & t ) {
  if( t == NULL ) {
    t = new TreeNode<T>( x, 0, NULL, NULL);
  }

  else if( x < t->key ) {
    _insert( x, t->left );
    int balance = height(t->right) - height(t->left);
    int leftBalance = height(t->left->right) - height(t->left->left);
    if ( balance == -2 ) {
      if ( leftBalance == -1 ) { rotate_____________( t ); }
      else                     { rotate_____________( t ); }
    }
  }

  else if( x > t->key ) {
    _insert( x, t->right );
    int balance = height(t->right) - height(t->left);
    int rightBalance = height(t->right->right) - height(t->right->left);
    if( balance == 2 ) {
      if( rightBalance == 1 ) { rotate_____________( t ); }
      else                    { rotate_____________( t ); }
    }
  }

  t->height = 1 + max(height(t->left), height(t->right));
}
```

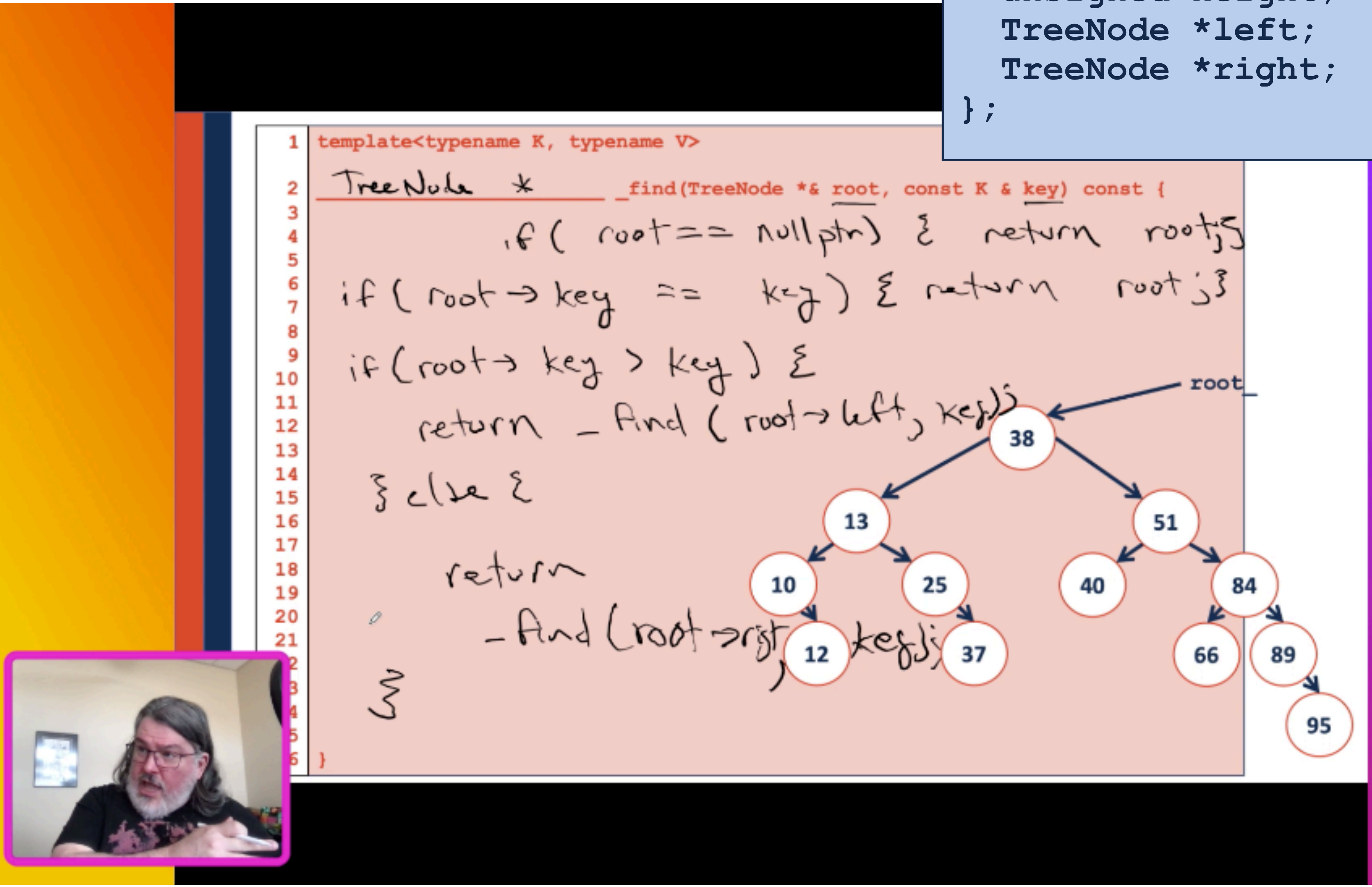
```
template <class T> void AVLTree<T>::_insert(const T & x, treeNode<T> * & t ) {
  if( t == NULL ) {
    t = new TreeNode<T>( x, 0, NULL, NULL);
  }

  else if( x < t->key ) {
    _insert( x, t->left );
    int balance = height(t->right) - height(t->left);
    int leftBalance = height(t->left->right) - height(t->left
    if ( balance == -2 ) {
      if ( leftBalance == -1 ) { rotate___Right___( t ); }
      else                     { rotate_LeftRight_( t ); }
    }
  }

  else if( x > t->key ) {
    _insert( x, t->right );
    int balance = height(t->right) - height(t->left);
    int rightBalance = height(t->right->right) - height(t->ri
    if( balance == 2 ) {
      if( rightBalance == 1 ) { rotate___Left___( t ); }
      else                    { rotate_RightLeft_( t ); }
    }
  }

  t->height = 1 + max(height(t->left), height(t->right));
}
```





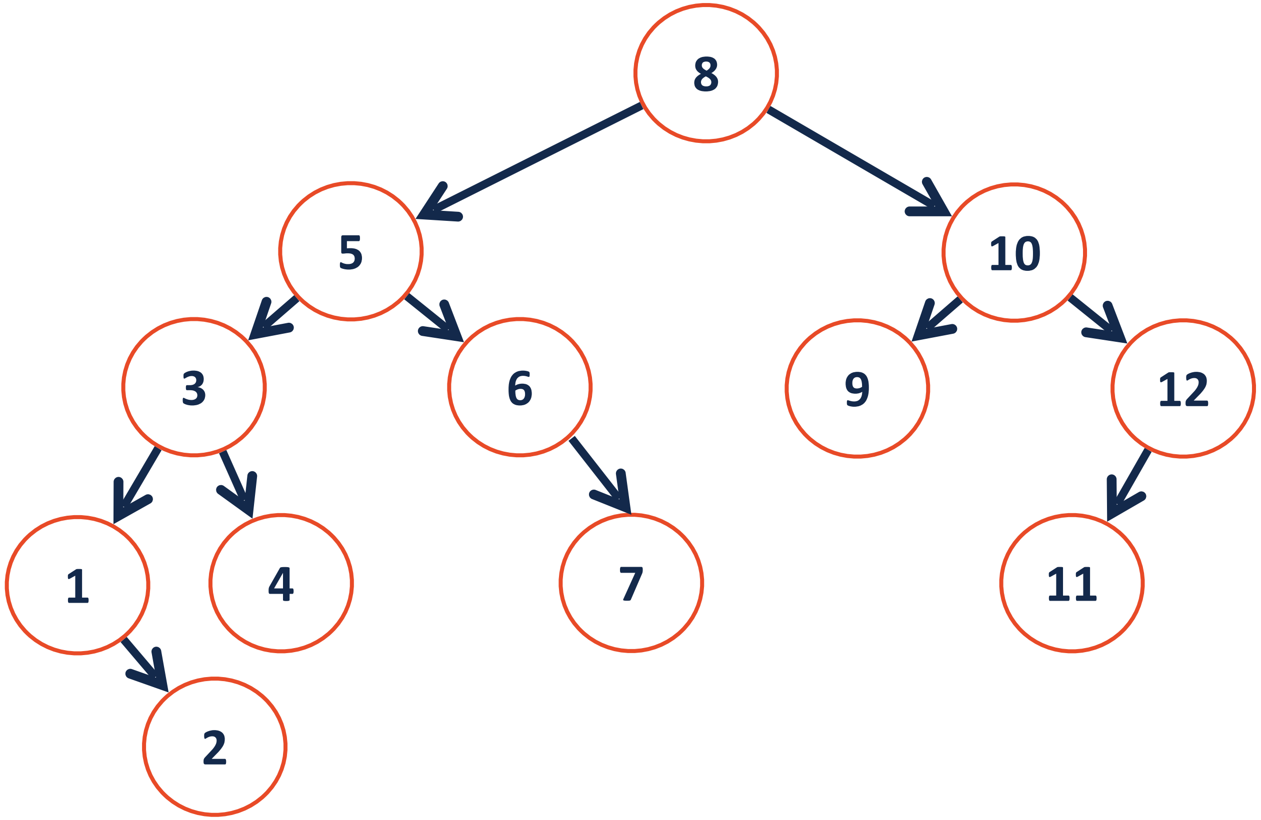
Find in an AVL Tree

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



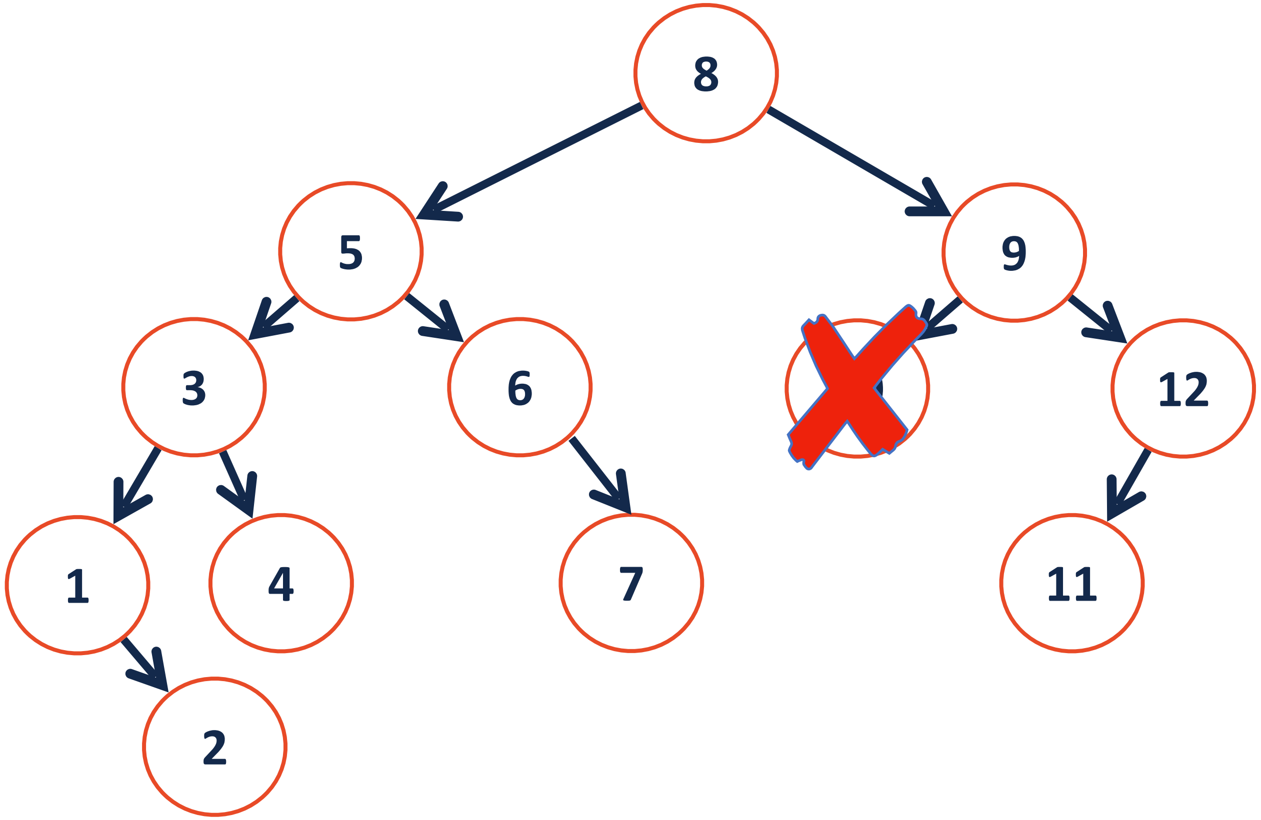
Remove from an AVL Tree

`_remove(10)`



Remove from an AVL Tree

`_remove(10)`

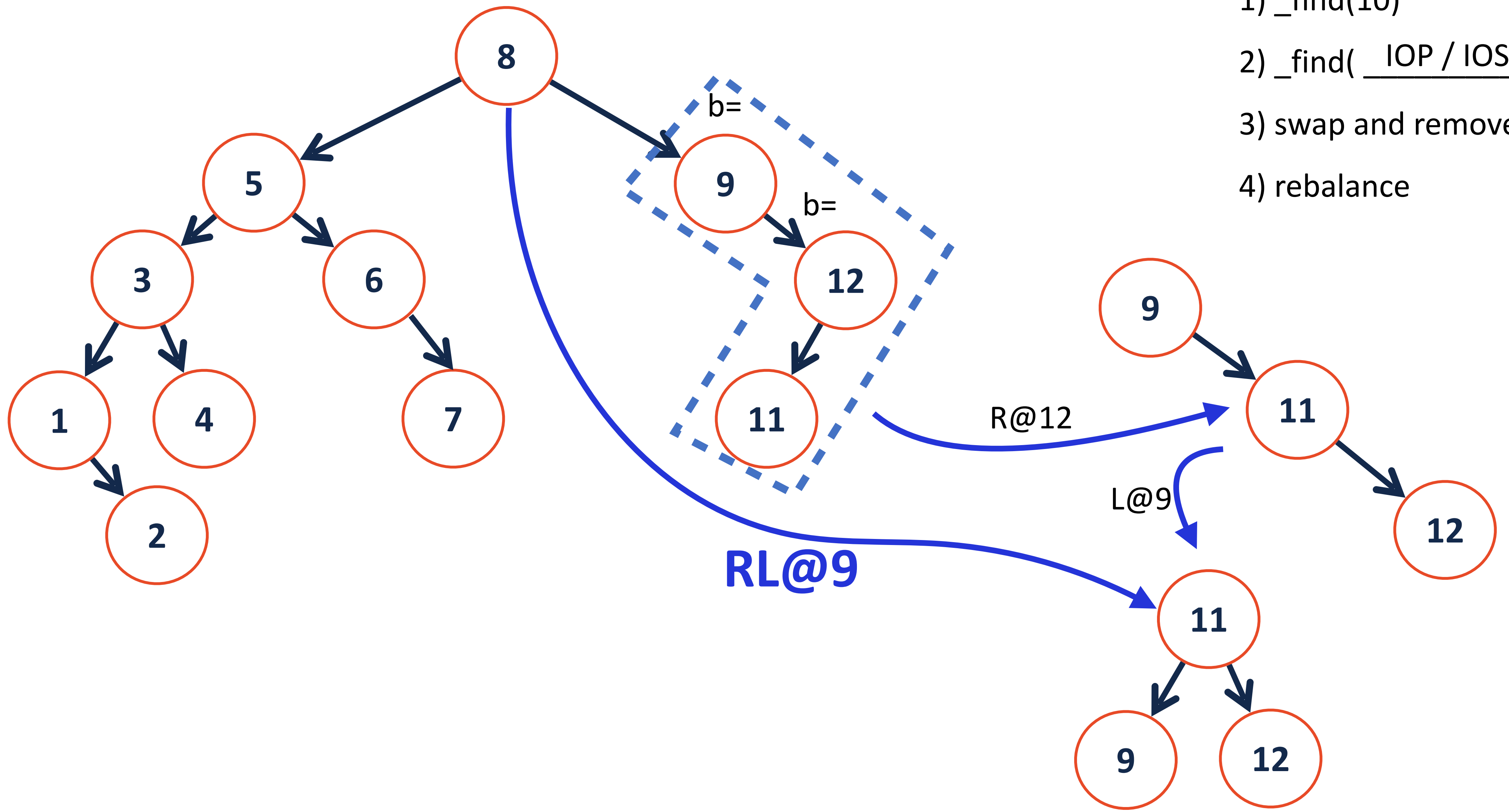


1) _find(10)
2) _find(IOP / IOS)
3) swap and remove

Remove from an AVL Tree

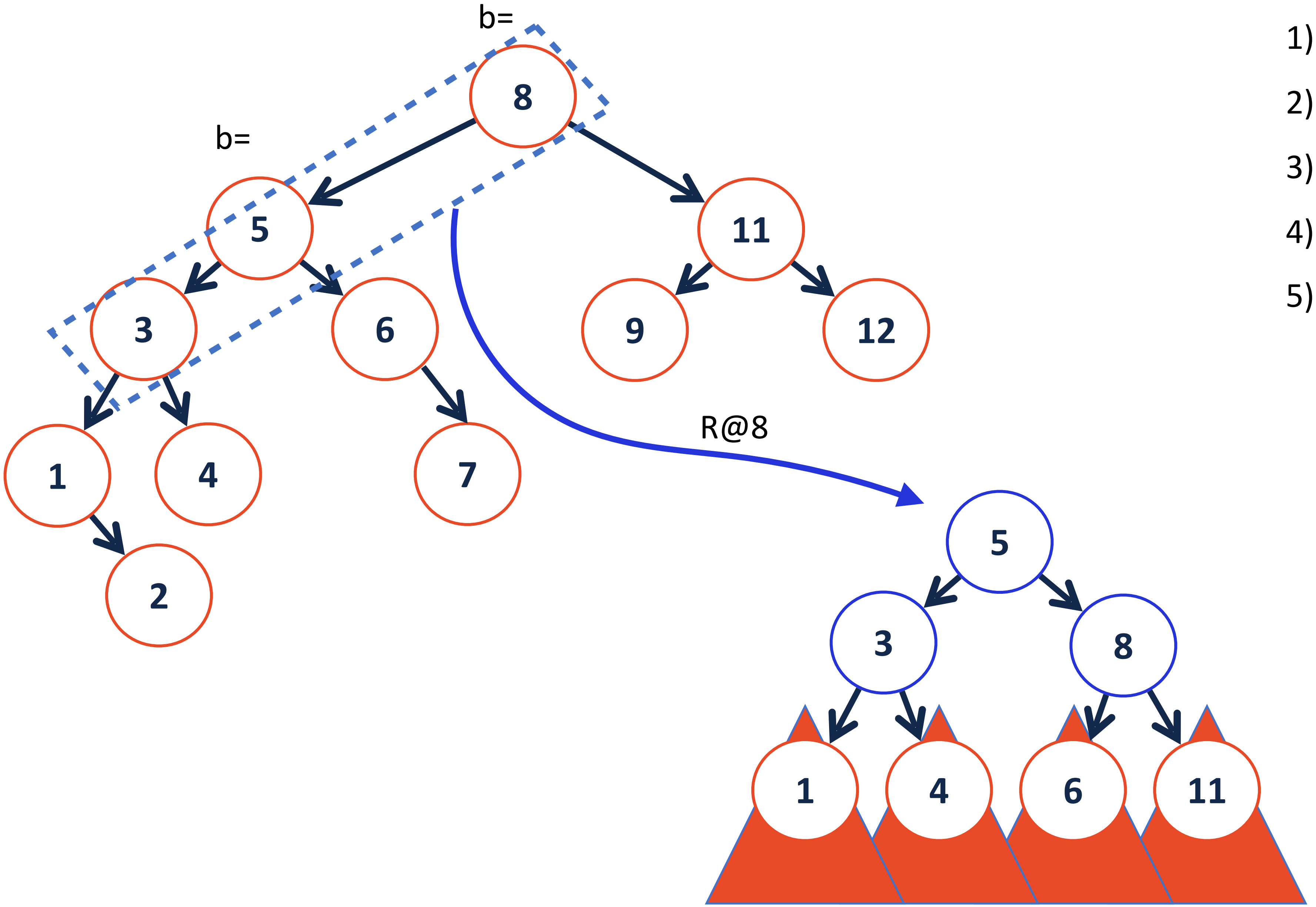
`_remove(10)`

1) _find(10)
2) _find(__IOP / IOS__)
3) swap and remove
4) rebalance



Remove from an AVL Tree

_remove(10)



1) _find(10)

2) _find(IOP / IOS)

3) swap and remove

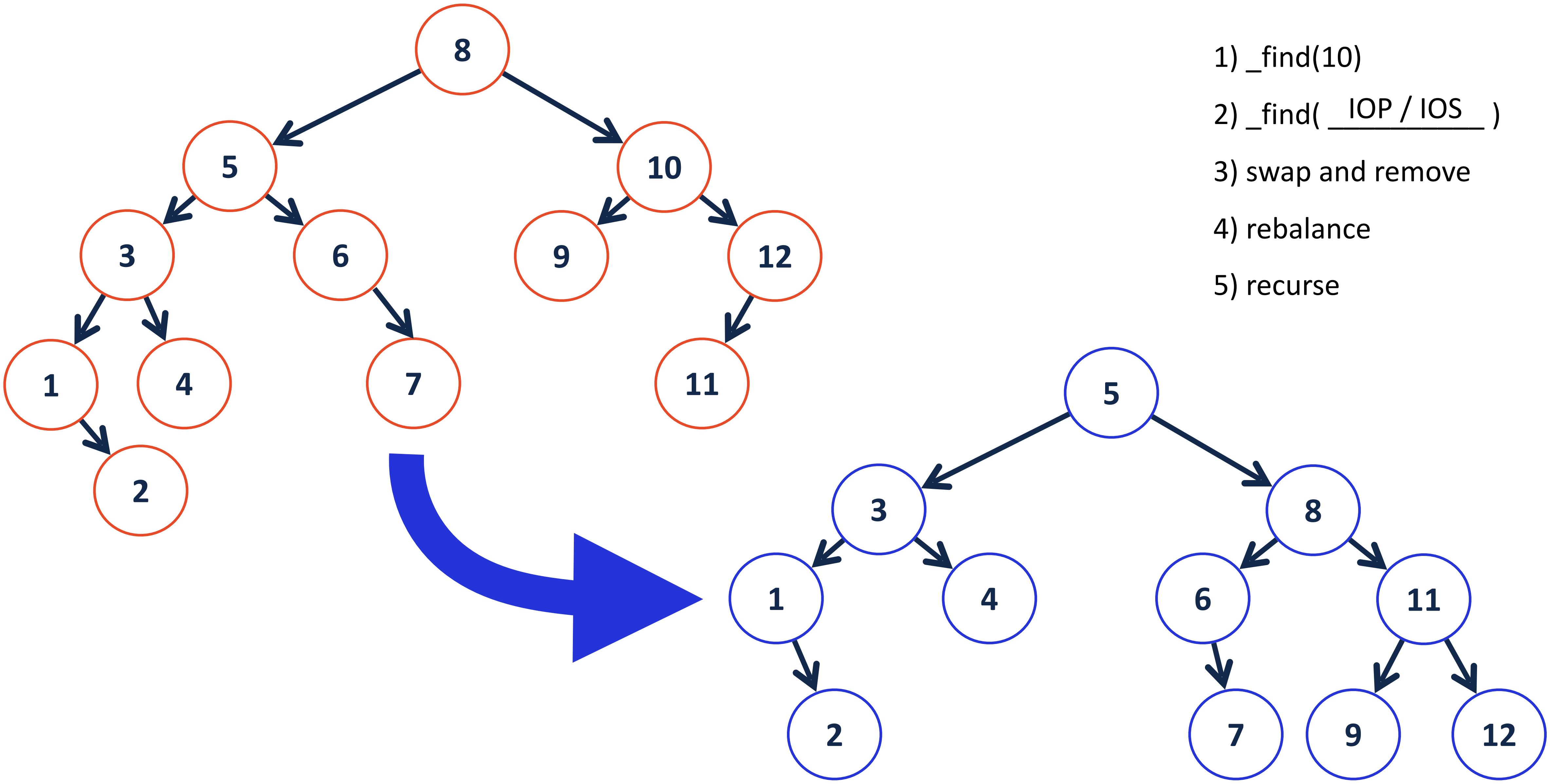
4) rebalance

5) recurse

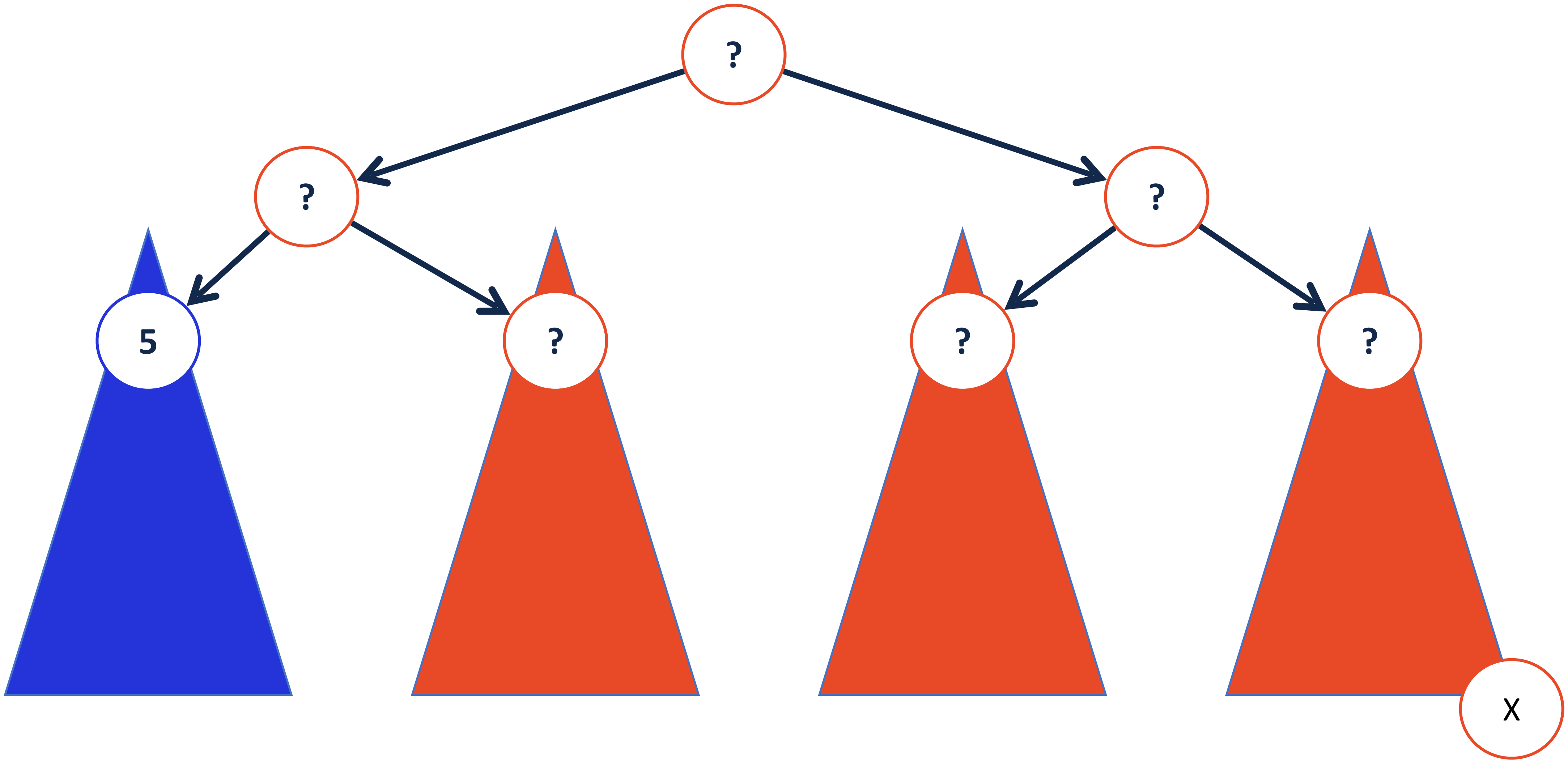
Remove from an AVL Tree

`_remove(10)`

1) _find(10)
2) _find(IOP / IOS)
3) swap and remove
4) rebalance
5) recurse



Remove from an AVL Tree



AVL Tree Analysis

For AVL tree of height h, we know:

find runs in: ____________.

insert runs in: ____________.

remove runs in: ____________.

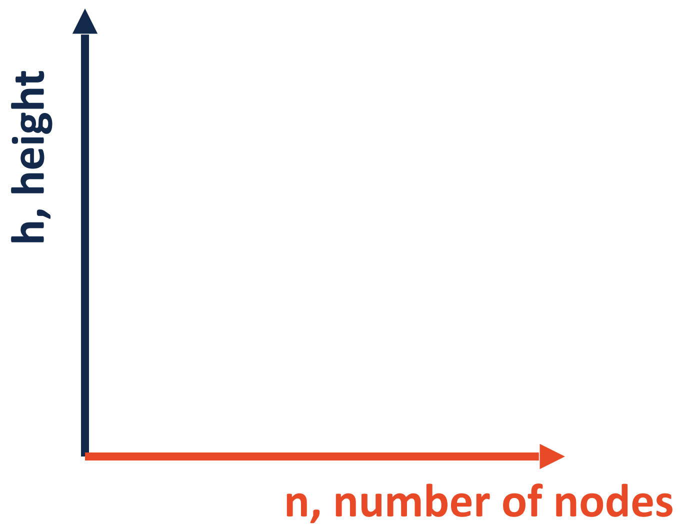
We will argue that: h is ___________.

AVL Tree Analysis

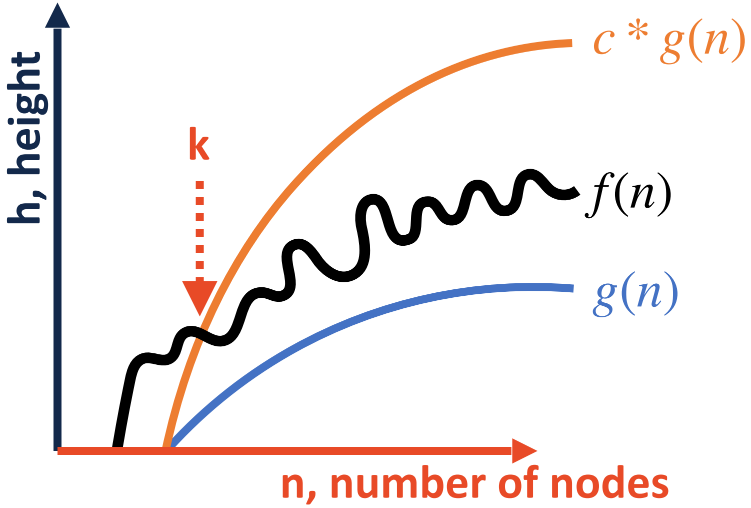
Definition of big-O:

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

…or, with pictures:

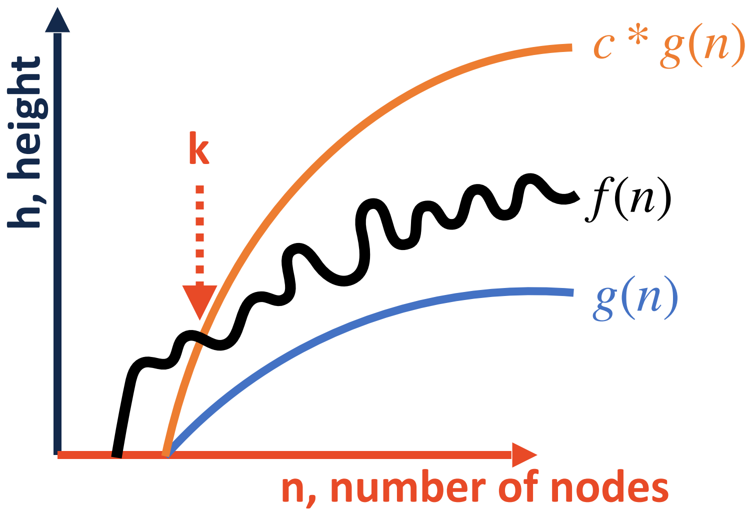


AVL Tree Analysis

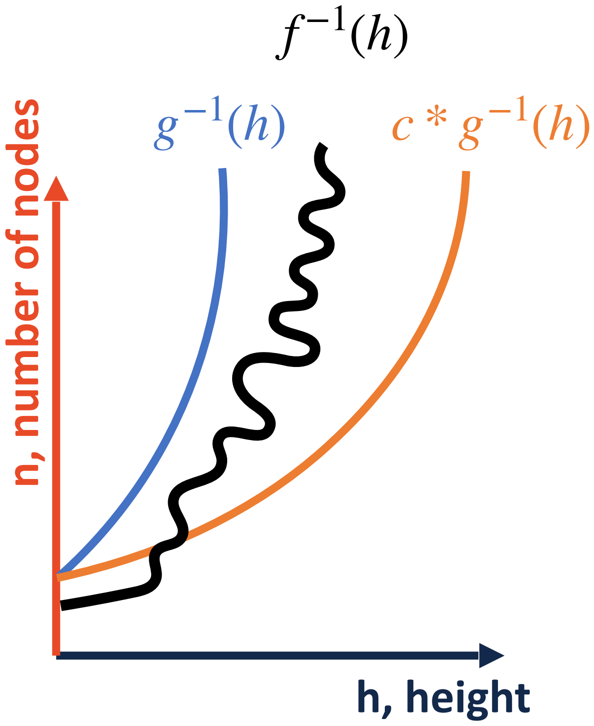


The height of the tree, **f(n)**, will always be <u>less than</u> **c × g(n)** for all values where **n > k**.

AVL Tree Analysis



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



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

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

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

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

$$N(h) \geq N(h) - 1$$

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: ________________

If for all heights $i < h, N(i) \geq 2^{i/2}$

then we must show for height h that $N(h) \geq 2^{h/2}$

Prove a Theorem

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

…and inverting: