CS 225

Data Structures

September 26 – AVL Trees

G Carl Evans

AVL Trees

Three issues for consideration:

- Rotations
- Maintaining Height
- Detecting Imbalance

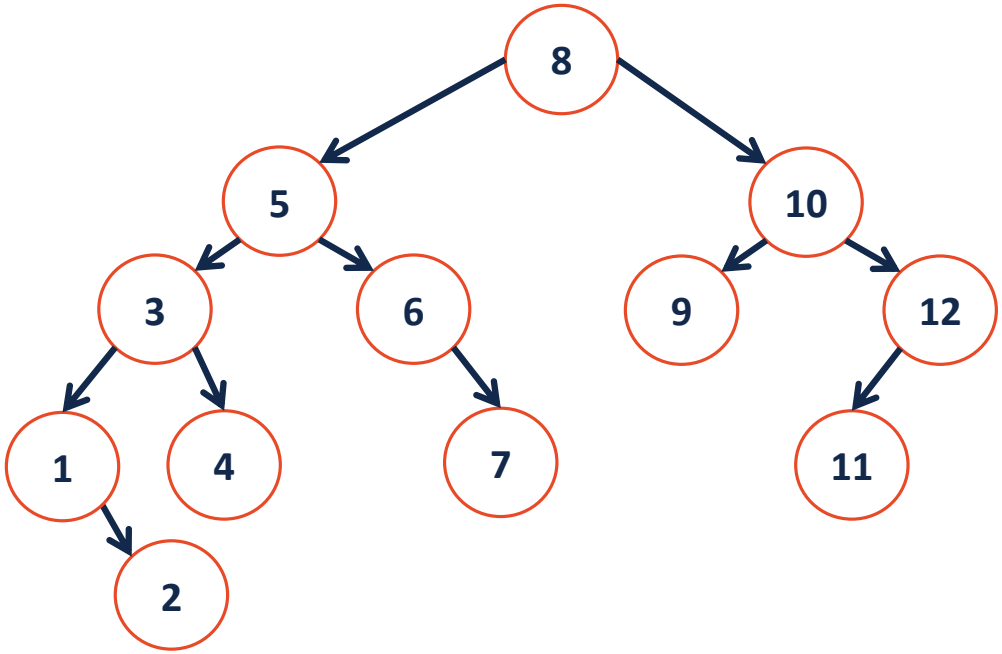
Insertion into an AVL Tree

`_insert(6.5)`

Insert (pseudo code):

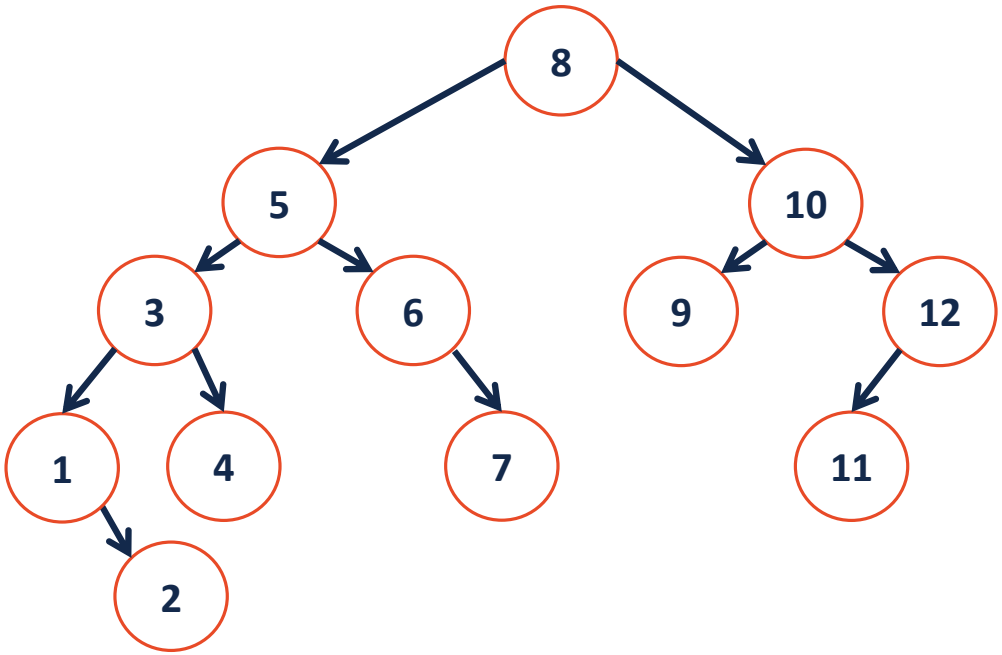
1: Insert at proper place
2: Check for imbalance
3: Rotate, if necessary
4: Update height

```
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

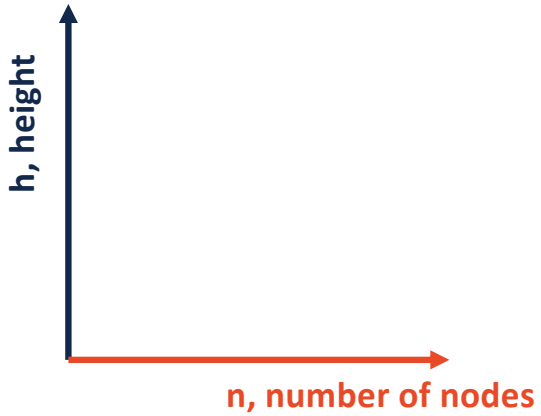
We know: insert, remove and find runs in: _____.

We will argue that: h is _____.

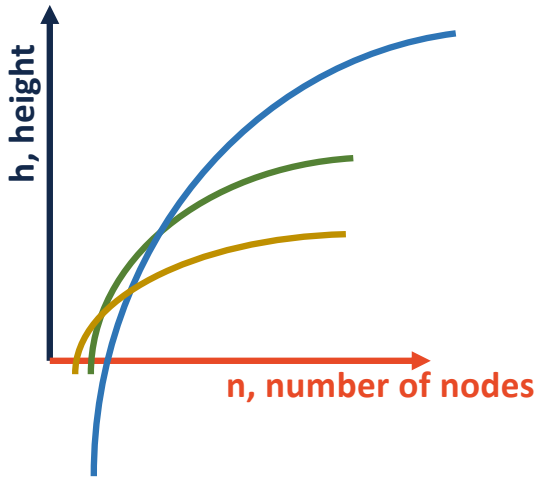
AVL Tree Analysis

Definition of big-O:

…or, with pictures:

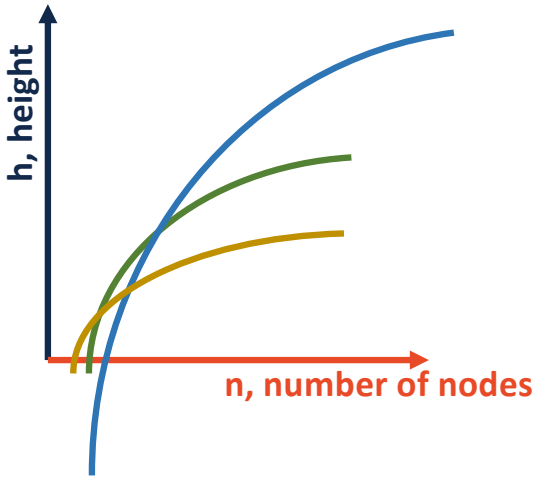


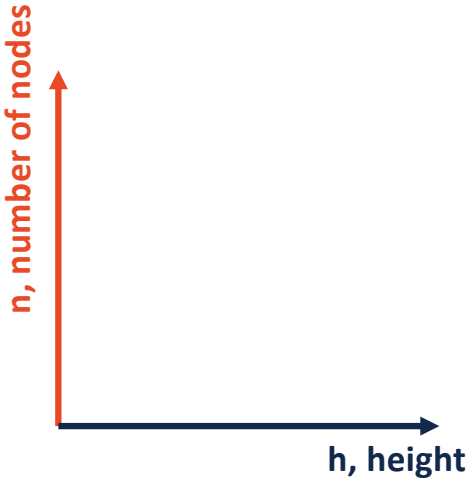
AVL Tree Analysis



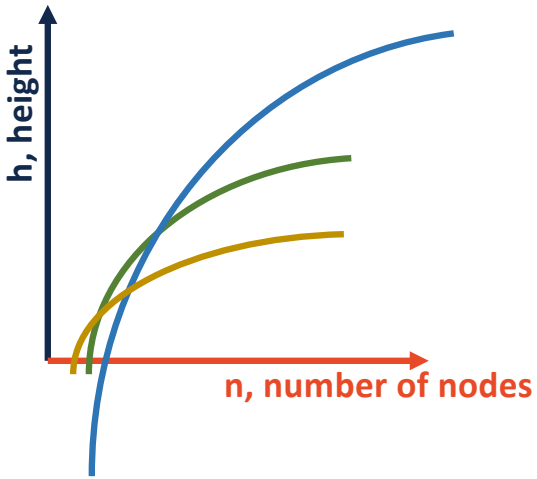
- The height of the tree, **f(n)**, will always be <u>less than</u> **c × g(n)** for all values where **n > k**.

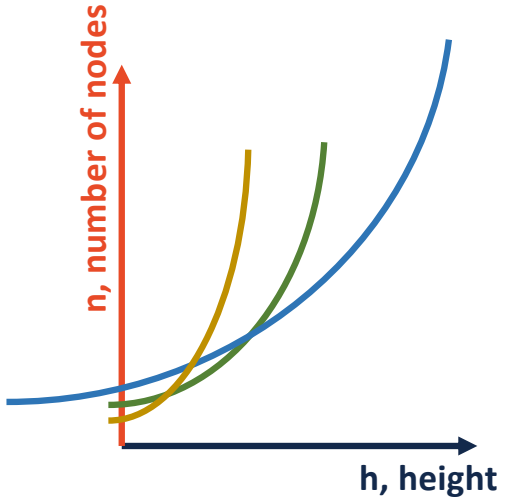
AVL Tree Analysis





AVL Tree Analysis





- The number of nodes in the tree, $\mathbf{f^{-1}(h)}$, will always be <u>greater than</u> $\mathbf{c \times g^{-1}(h)}$ 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**:

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:

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

II. Case: _____

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

Prove a Theorem

III. Case: _____

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

Prove a Theorem

By an Inductive Hypothesis (IH):

We will show that:

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

Prove a Theorem

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

…and inverting:

Summary of Balanced BST

Red-Black Trees

- Max height: 2 * lg(n)
- Constant number of rotations on insert, remove, and find

AVL Trees

- Max height: 1.44 * lg(n)
- Rotations:

Summary of Balanced BST

Pros:

- Running Time:

  - Improvement Over:

- Great for specific applications:

Summary of Balanced BST

Cons:

- Running Time:

- In-memory Requirement:

Red-Black Trees in C++

C++ provides us a balanced BST as part of the standard library:

```
std::map<K, V> map;
```

Red-Black Trees in C++

```
V & std::map<K, V>::operator[]( const K & )
```

Red-Black Trees in C++

```
V & std::map<K, V>::operator[]( const K & )
```

```
std::map<K, V>::erase( const K & )
```

Red-Black Trees in C++

```
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```

Every Data Structure So Far

	Unsorted Array	Sorted Array	Unsorted List	Sorted List	Binary Tree	BST	AVL
Find							
Insert							
Remove							
Traverse							