CS 225

#10: *Binary Search Trees*

Brad Solomon

Traversal vs. Search:

- **Traversal** visits every node in the tree exactly once.
- **Search** finds one (or more) element(s) in the tree.

Breadth First Traversal + Search:

Depth First Traversal + Search:

Runtime Analysis on a Binary Tree:

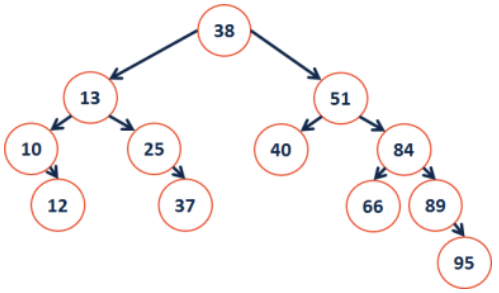
- Find an element: Best case? Worst case?
- Insertion of a sorted list of elements?
  Best case? Worst case?
- Running time bound by

Dictionary ADT

Dictionary.h

```
3
4  class Dictionary {
5    public:
6
7
8
9
10
11
12
13   private:
14
15
16 };
```

A Searchable Binary Tree?



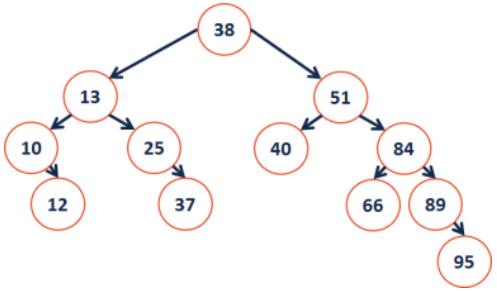
Binary Search Tree Property:

Finding an element in a BST:

BST.hpp

```
template <typename K, typename V>
_____________________________________ find(const K & key) {



}

template <typename K, typename V>
_____________________________________________________ _find
    (TreeNode *& root, const K & key) {









}
```

Inserting an element into a BST:

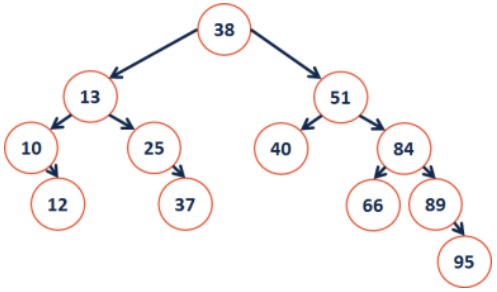


BST.hpp

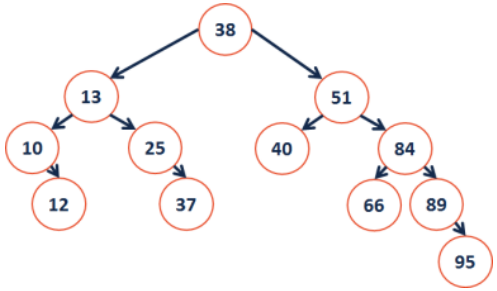
```
template <typename K, typename V>
void BST<K, V>::_insert(TreeNode *& root, K key, V value)
{




}
```

Running time? _____ **Bound by?** _____

What if we did <u>not</u> pass a pointer by reference?



Removing an element from a BST:



`_remove(40)`

`_remove(25)`

`_remove(10)`

`_remove(13)`

One-child Remove	Two-child remove

BinaryTree.hpp

```
template <class K, class V>
void BST<K,V>::_remove(TreeNode *& root, const K & key) {




}
```

Running time? _____ **Bound by?** _____