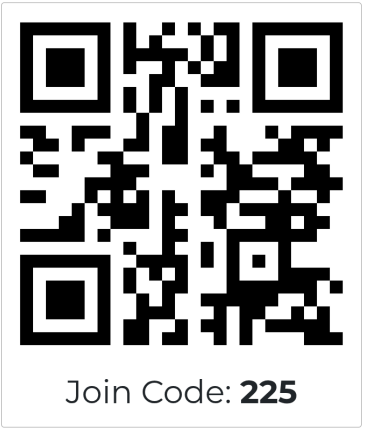
Announcements

1. Exam 2 Review in Lab!
   a. New FRQs
   b. New Practice Coding Questions
2. Exam 2 Next Week (M-W)
   a. Lab BST content removed from exam
   b. BST conceptually for FRQs and MCQs is fair game
3. Extra Credit Survey! (2pts due March 4th 11:59pm)

Extra Credit Survey!



Coding isn't about BSTs

Warm-Up Question: How would you explain the binary search tree property?

Put your answer in the discord!

Binary Search Tree Implementations

Learning Objectives

1. Implement the BST Find and Remove Algorithm Insert
2. Analyze the different runtimes of dictionary implementations

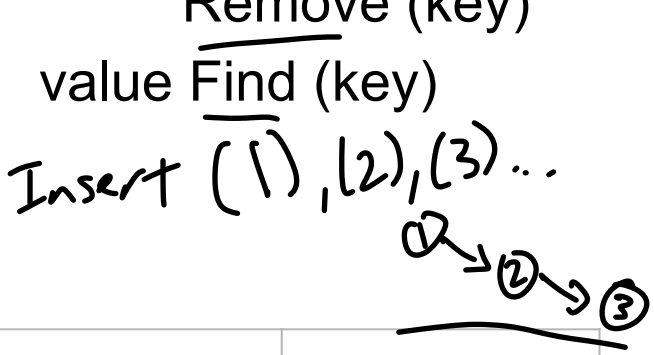
BST Runtimes

Insert (key, value)

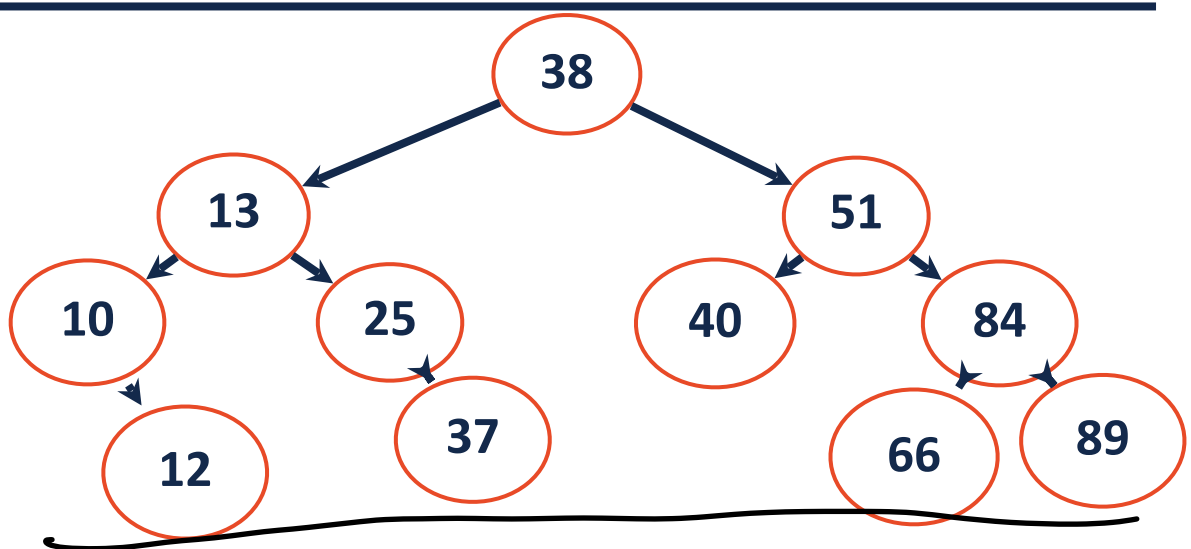
Remove (key)

value Find (key)

Insert (1), (2), (3) ...

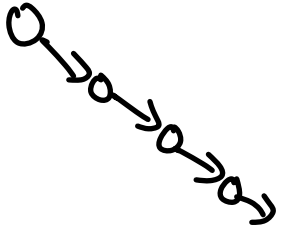


Function	Runtime
Insert	O(h), O(n)
Remove	O(h), O(n)
Find	O(h), O(n)

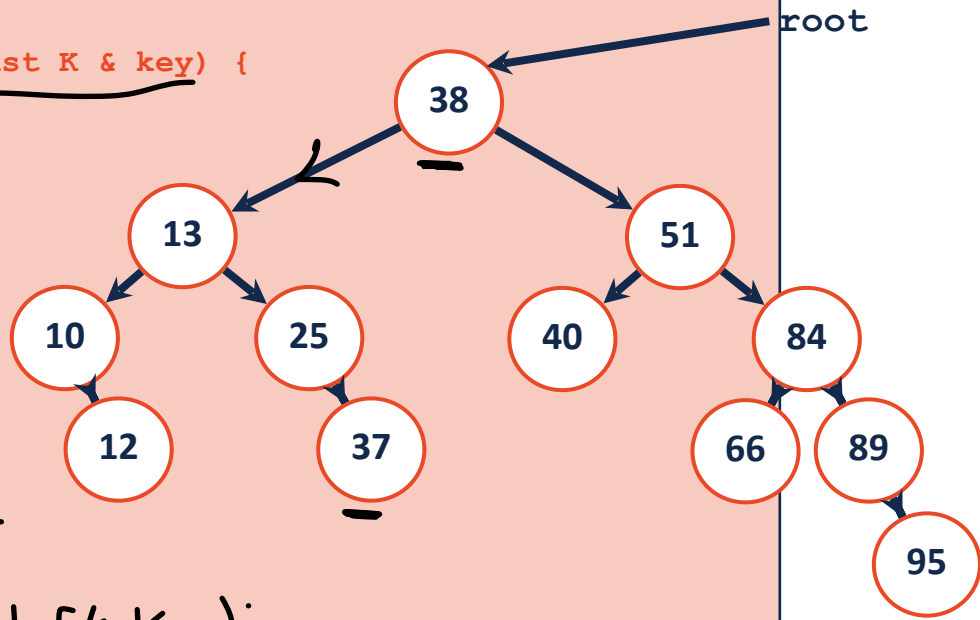


$\forall x, x \in T_L, x < r$

$\forall y, y \in T_R, y > r$

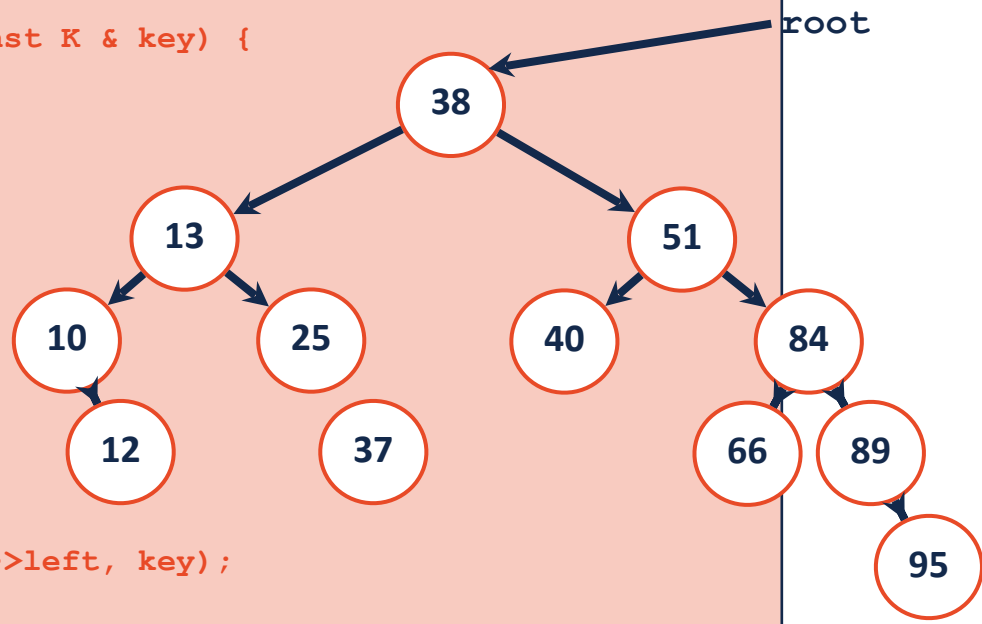


```
1  template<class K, class V>
2  TreeNode*& find(TreeNode *& root, const K & key) {
3      if(root == Null){
4          return root;
5      }
6      if(root->key == key){
7          return root;
8      }
9      if (root->key > key){
10         return find(root->left, key);
11     }
12     if (root->key < key){
13         return find(root->right, key);
14     }
15 }
```

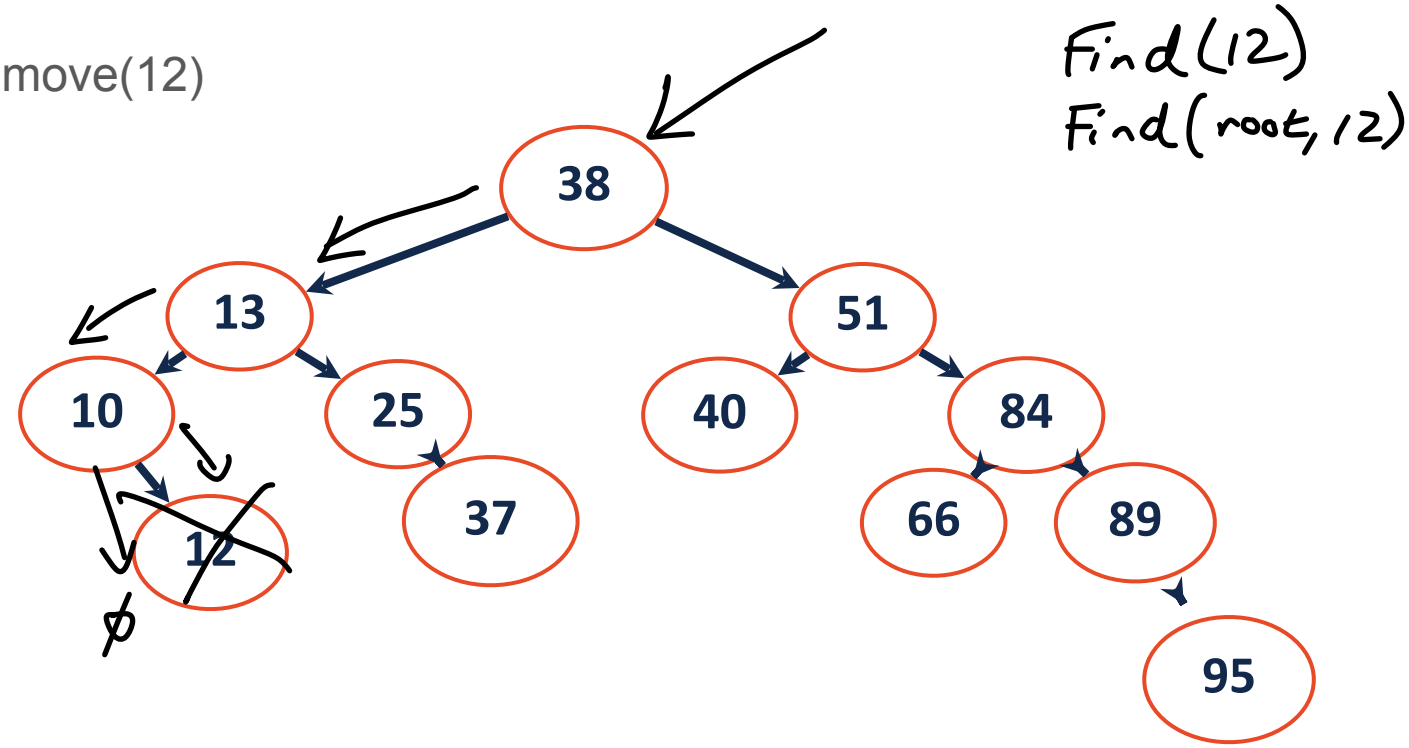


```
1  template<class K, class V>

2  TreeNode*& find(TreeNode *& root, const K & key) {
3
4          if (root == NULL){
5                  return root;
6      }
7
8          if (root->key == key){
9                  return root;
10     }
11
12
13
14
15         if(key < root->key){
16                 return find(root->left, key);
17     }
18
19     if(key > root->key){
20         return find(root->right,key);
21     }
22
23
24
25
26
```



Remove(12)



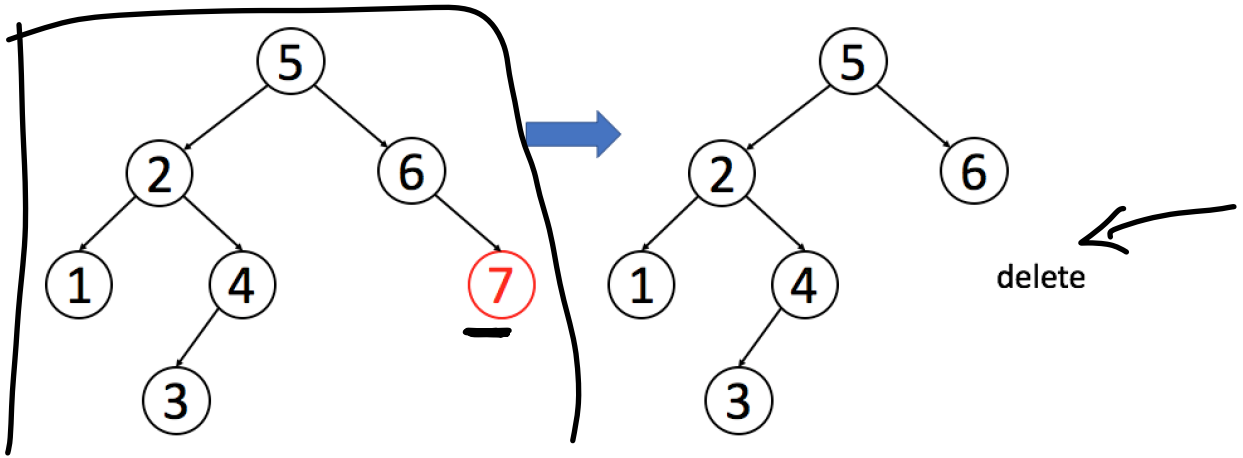
Remove

Case 1 (No child):

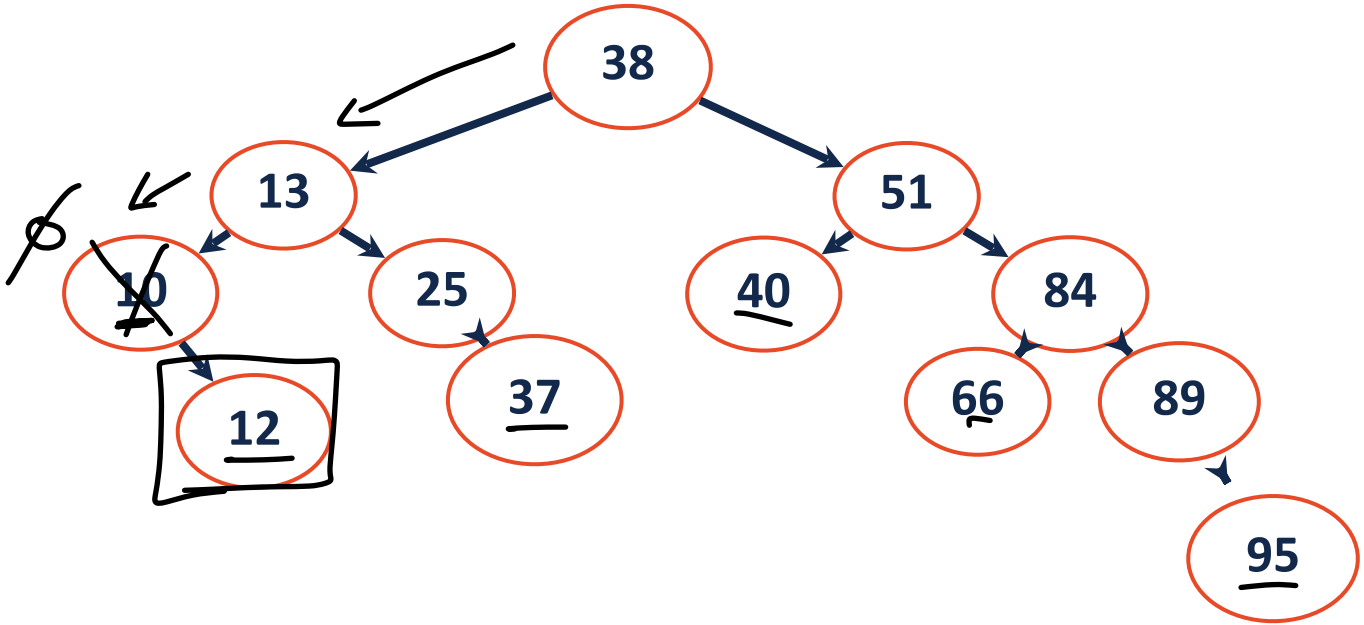
1. Find the target node (passed as a reference to the parent's pointer)
2. Delete the node
3. Set child pointer of parent to NULL

Case 1: No Child

Delete 7



Remove(10)

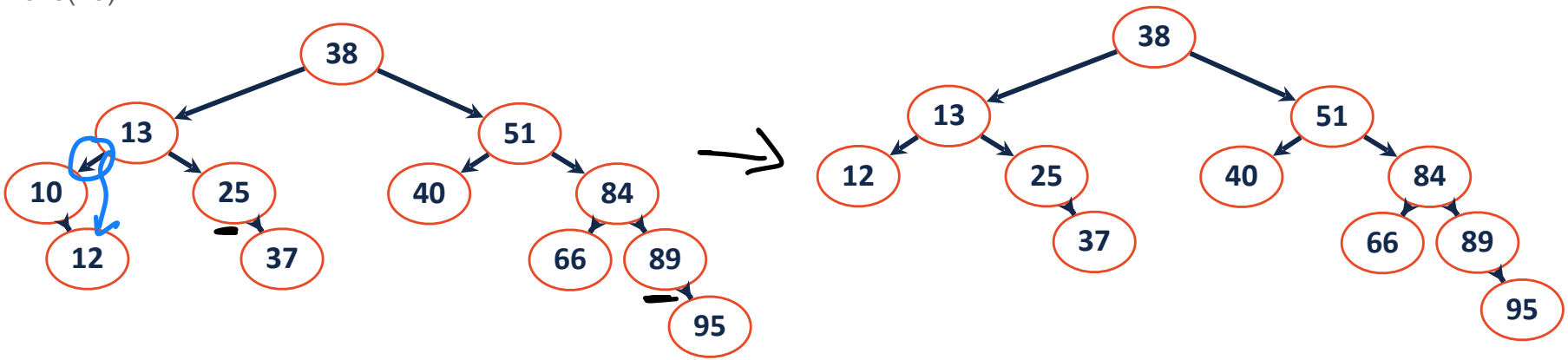


Remove

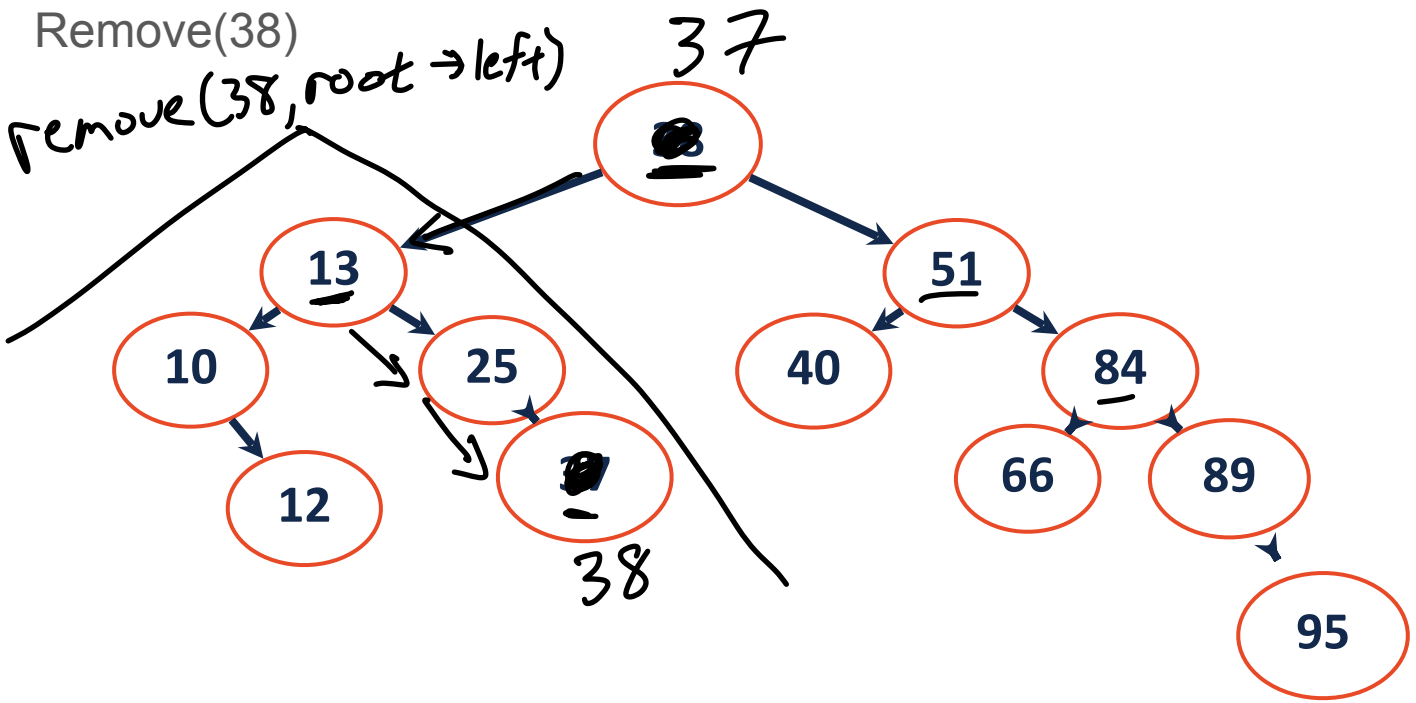
Case 2 (1 child):

1. Find the target node (passed as a reference to the parent's pointer)
2. Make a temporary pointer to the target
3. Set the parent's node to the target's child
4. Delete the target

Remove(10)

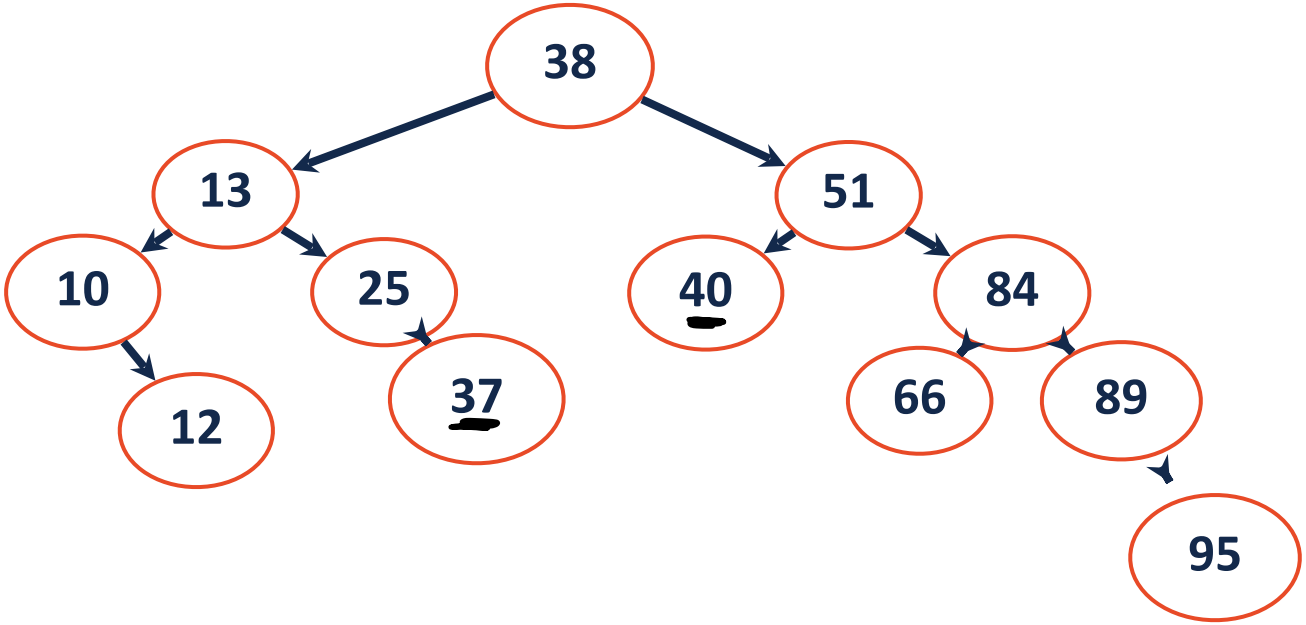


Remove(38)



Join Code: **225**

Remove(38) - What node(s) could take 38's place?



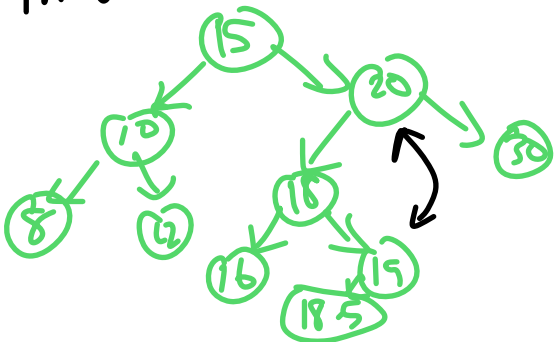
Remove

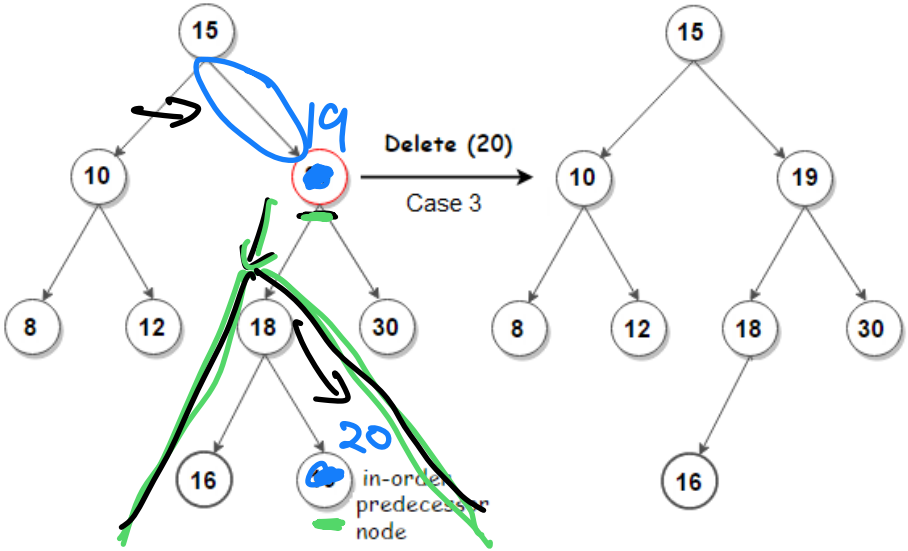
Case 1: No children
Case 2: 1 child
Case 3: root = find(target)
remove(root→left) ← swap IOP

Case 3 (2 children):

1. Find the target node (passed as a reference to the parent's pointer)
2. Find the target's In Order Predecessor (IOP) → The node preceding the target in an inorder traversal
3. Swap the target with the IOP
4. Recursively call on the target's new ~~location~~ subtree

IOS (In order successor)
The node following the target in an inorder traversal





Dictionary ADT

Insert (key, value)

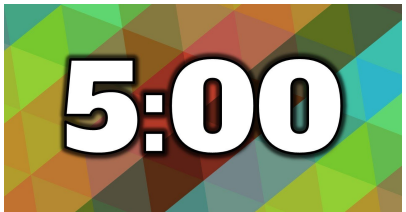
Remove (key)

value Find (key)

Operation	BST	Binary Tree	Sorted Array	Unsorted Array	Sorted Linked-List	Unsorted Linked-List
find	O(h), O(n)	O(n)	O(log n)	O(n)	O(n)	O(n)
insert	O(h), O(n)	O(1)	O(n)	O(n), O(1)*	O(n)	O(1)
delete	O(h), O(n)	O(n)	O(n)	O(n)	O(n)	O(n)
traverse	O(n)					

n: # of elements in my data structure

Fill out the Big O for each of these functions!

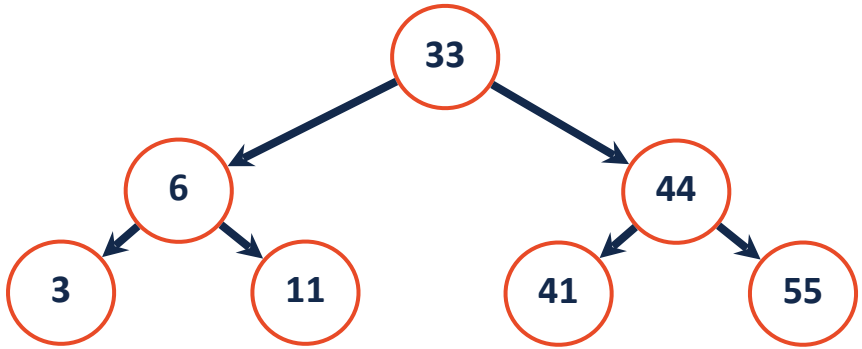


Closest Element

BSTs are useful structures for range-based and nearest-neighbor searches.

Q: Consider points in 1D: **p = {p_1, p_2, …, p_n}.**

…what point is closest to 16?



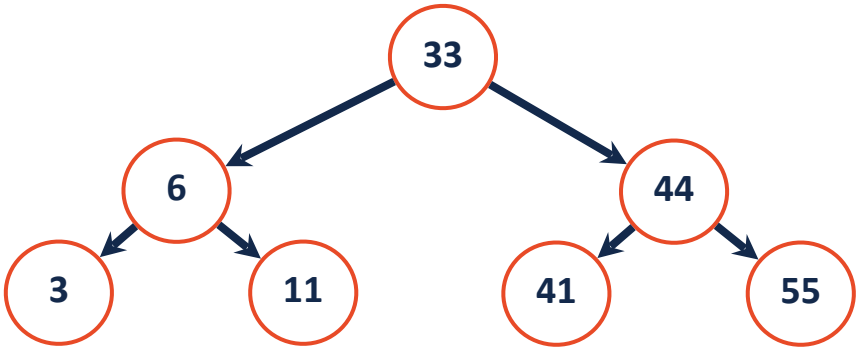
Ex:

Range-based Searches

Balanced BSTs are useful structures for range-based and nearest-neighbor searches.

Q: Consider points in 1D: **$p = \{p_1, p_2, \ldots, p_n\}$.**

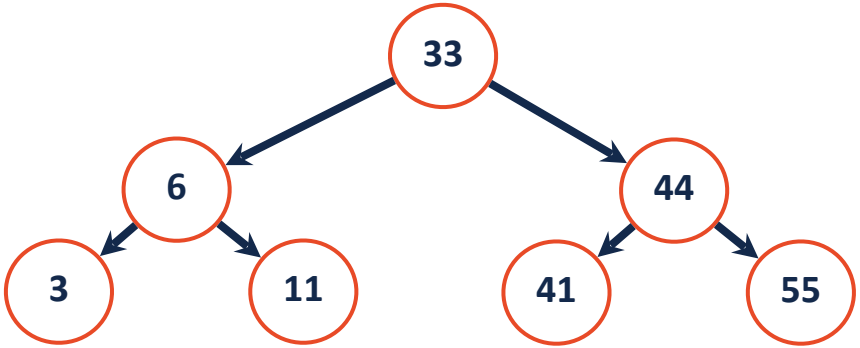
…what points fall in [11, 42]?



Ex:

Range-based Searches

Query: [11,42]



Red-Black Trees in C++ (map)

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

- Returns an iterator pointing to the first element in the container whose key is not considered to go before *k* (i.e., either it is equivalent or goes after).

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

- Returns an iterator pointing to the first element in the container whose key is considered to go after *k*.

Example

Given a set of points:

lower_bound(12)?

lower_bound(18)?

lower_bound(14)?

upper_bound(40)?

upper_bound(44)?

upper_bound(45)?