Announcements

1. MP1 Extra Credit due Tonight!

Exam 2 Review in Lab this week!

Prof. Solomon's OH are cancelled this week

Join Code: **225**

Warm-Up Question: If you are searching through an infinite tree, should you use Breadth-First Search or Depth-First Search?

Iterative Deepening Search/Binary Search Trees

Learning Objectives

1. Understand Iterative Deepening
2. Know the runtime of iterative deepening on a binary tree
3. Understand the Dictionary ADT
4. Compare runtimes of operations on BST with binary trees

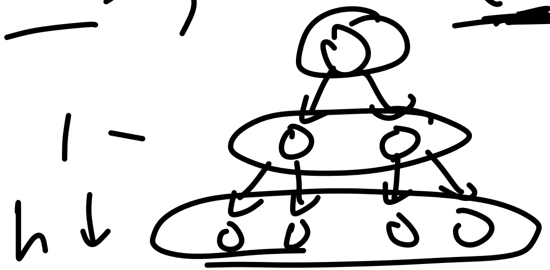
Tree Search

→ Guarantees a solution in ∞ tree

Breadth-First - in ∞ tree

Time   Space Complexity

Level Order Traversal, Queue, $O(n)$, $O(2^h)$



Q: root

Q: left right

Depth-First -

Pre-
Post-Order Traversals, Stack, $O(n)$, $O(h)$
In

Time   Space

80% BFS
20% DFS

In an ∞ tree, can't guarantee a solution

Iterative Deepening DFS

K - Current depth limit

m - depth goal node

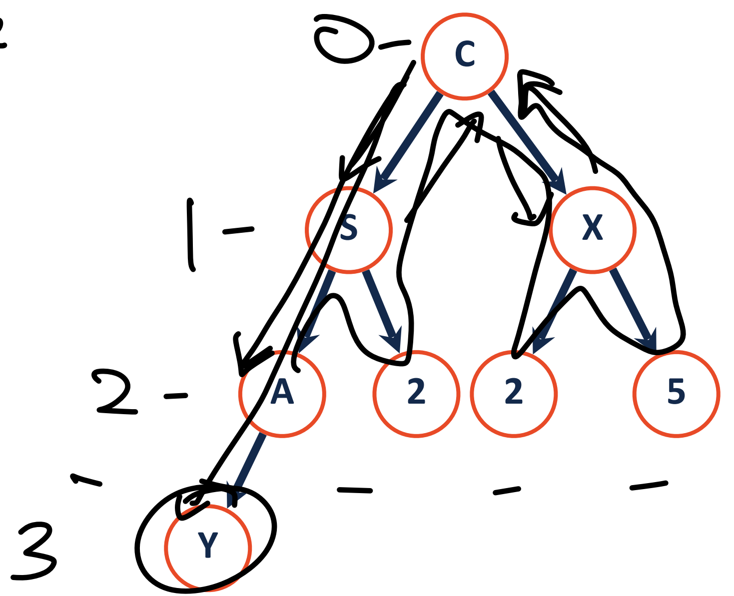
Time	Space
??	O(h)
	O(m)

Goal is found is

$K \geq m$

Ex. Goal → y

m → 3



K=3

Iterative Deepening Analysis

m - Goal depth

Level	0	1	2	...	m
Total Nodes	$2^0=1$	$1+2=3$, $2^0+2^1=3$	$1+2+4$, $2^0+2^1+2^2=7$	...	

1. Guarantees solution
2. Saves space complexity

$$\sum_{k=0}^{m} \sum_{j=0}^{k} 2^j = \sum_{k=0}^{m} 2^{k+1} - 1$$

$$= \sum_{k=0}^{m} 2^{k+1} - \sum_{k=0}^{m} 1$$

k: # of times DFS is run

j: For each level in DFS

$$= 2^{m+2} - 1 - m$$

$$\Rightarrow O(2^{m+2}) \Rightarrow O(4 \cdot 2^m) \Rightarrow O(n)$$

$$\boxed{\text{Time Complexity: } O(n)}$$

Dictionary

Stores key -> value pairs

What applications of dictionaries can you think of?

Applications:

Words -> Meanings / Definitions

Usernames -> Passwords

Names -> Phone Numbers

UIN -> Student Info

Flight No -> Flight Info.

Array Compact Set of Integers -> data at index

[0, 1, 2, 3...]

X [0, 2, 4]

Dictionary ADT

void Insert (Key , Value)

(Value, void) Remove (Key)

Value Find (Key)

Key musts be unique

Values don't need to be unique

```cpp
#pragma once


class Dictionary {
  public:
    void insert(const K & key, V & value);
    V remove(const K & key);
    iterator find(const K & key) const;
    iterator begin();
    iterator end();
    // ...

  private:
    // ...



};
```

What if my element isn't in my dictionary?

```
#pragma once


class Dictionary {
  public:
    void insert(const K & key, V & value);
    V remove(const K & key);
    V find(const K & key) const;
    iterator begin();
    iterator end();
    // ...

  private:
    // ...
         TreeNode * root_;



};
```

What if you are trying to find a key that doesn't exist?

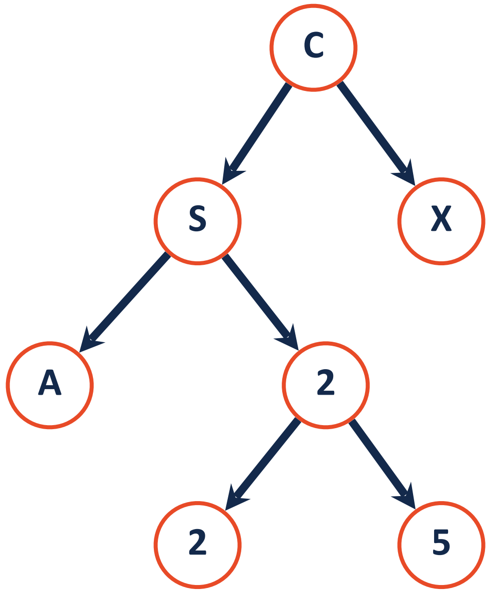
Binary Tree Definition

A *binary tree* T is either:

$$T = \emptyset$$

OR

$$T = (r, T_L, T_R)$$



Searching through Binary Trees

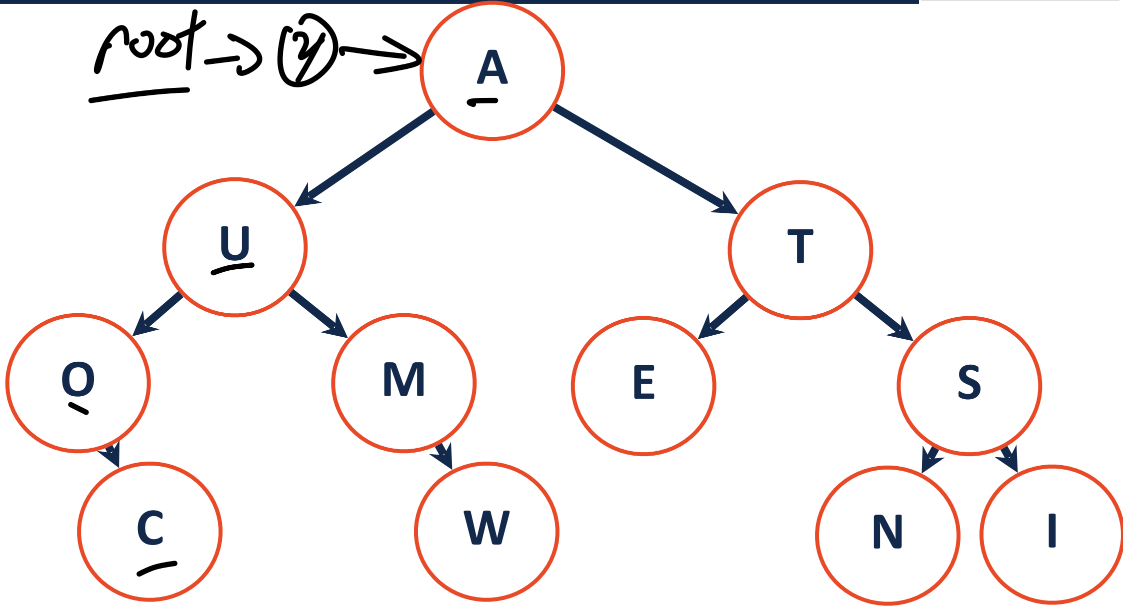
1:00

Join Code: 225

Insert (key, value)

Remove (key)

value Find (key)



Function	Runtime
Insert	O(1)
Remove	O(n)
Find	O(n)

Binary Search Tree Definition

A *binary search tree* T is either:

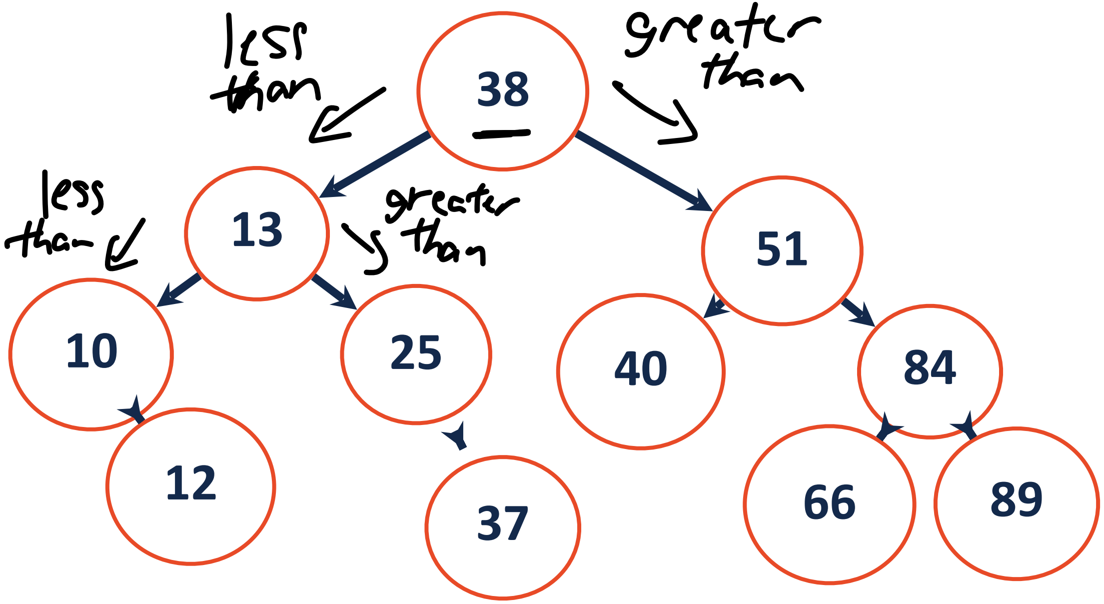
$$T = \emptyset$$

OR

$$T = (r, T_L, T_R)$$

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

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



Binary Search Trees (BSTs)

How do my functions change given the structure of the BST?

Insert: Insert 41

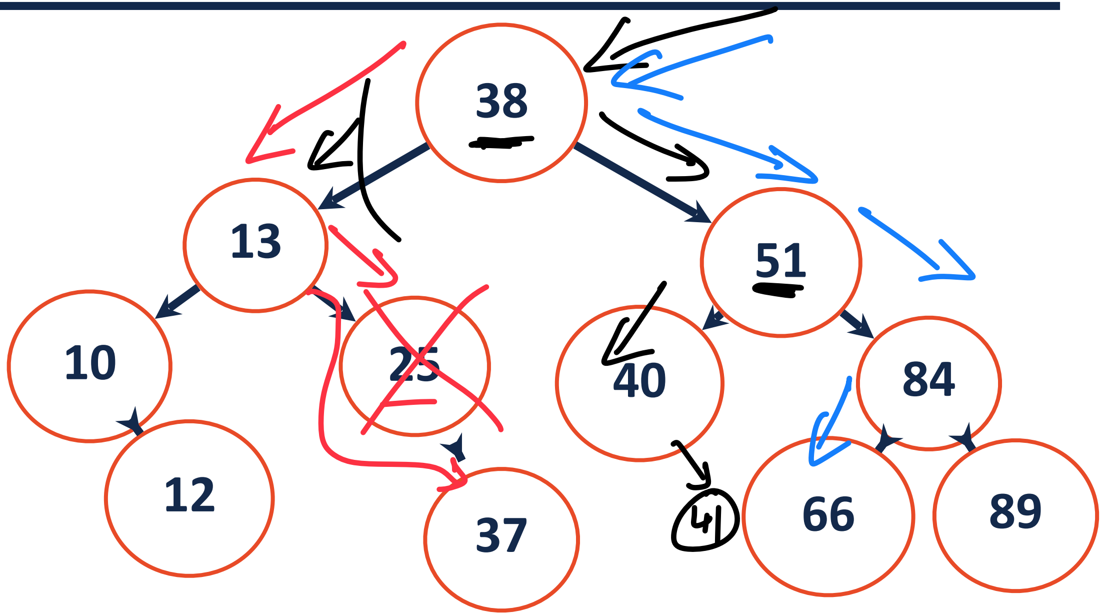
Find: Find(66)

-No traversal required

Removing: Remove(25)

Find(25)

Delete(25)

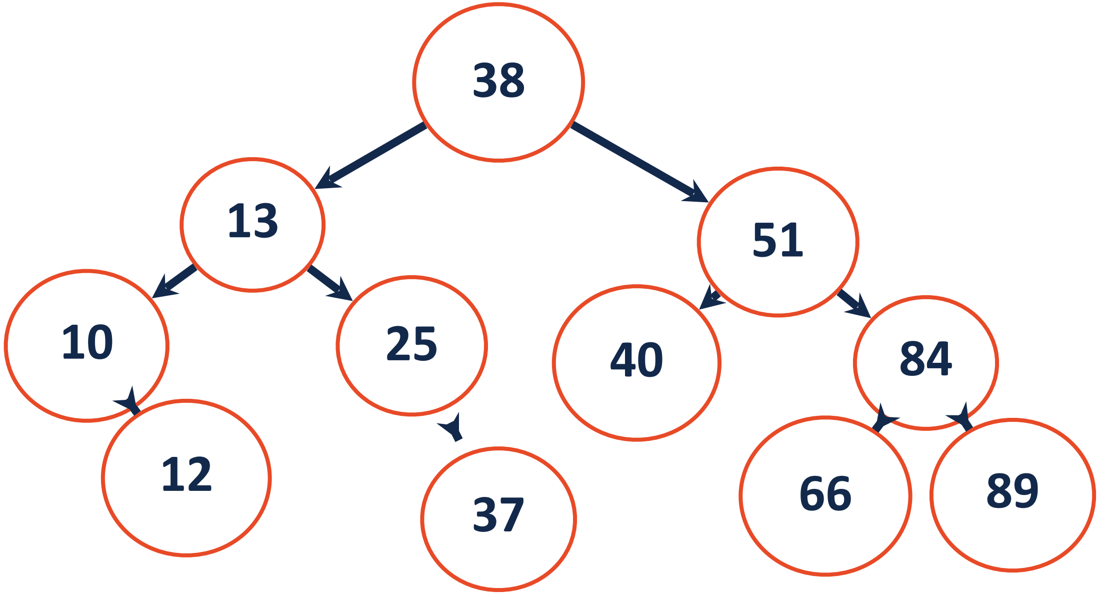


Join Code: **225**

Insert (key, value)

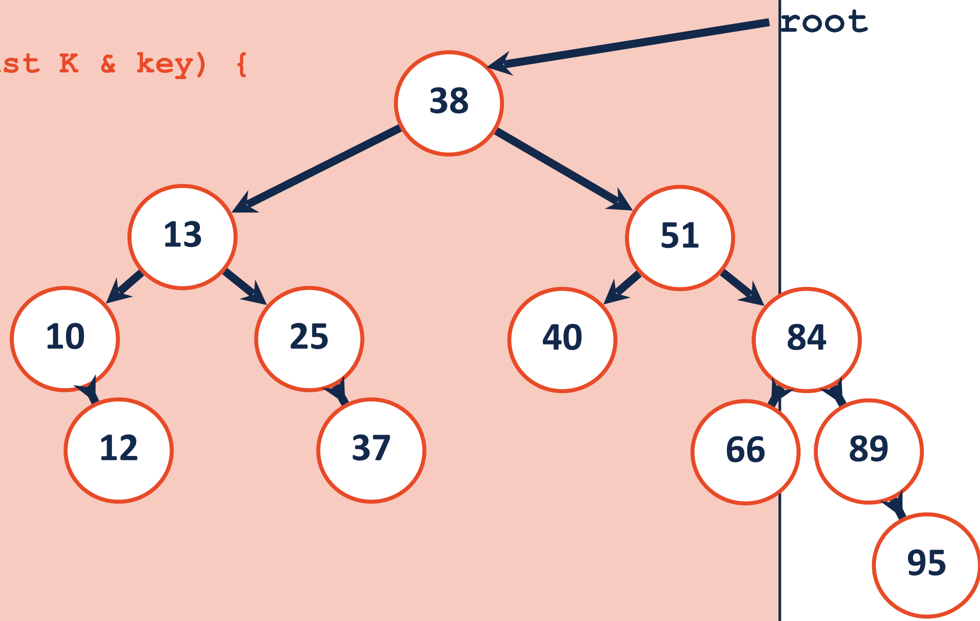
Remove (key)

value Find (key)



Function	Runtime
Insert	
Remove	
Find	

```
template<class K, class V>

TreeNode*& find(TreeNode *& root, const K & key) {























}
```



```
template<class K, class V>
TreeNode*& find(TreeNode *& root, const K & key) {

          if (root == NULL){
                    return root;
     }

          if (root->key == key){
                    return root;
     }

          if(key < root->key){
                    return find(root->left, key);
     }

     if(key > root->key){
          return find(root->right,key);
     }






}

```