CATALYST

[4] Data Structures: Trees

What will be covered?

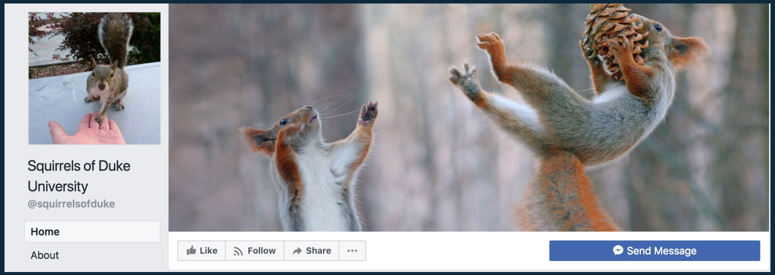
- **Definitions:** what's a tree?
- **Common strategies:** using tree properties, types of traversal
- **Beginner:** depth of tree, pre/in/post traversal with recursion
- **Medium:** validate BST, LCA in BST, LCA in Binary Tree, level order traversal, sorted array to BST, isSubTree()
- **Advanced:** Binary tree diameter, pre/in/post w/o recursion, construct b-tree, maxPathSum, Arithmetic

CATALYST

1

What's a Tree?

Hint: doesn't come with squirrels



Trees

A *tree* is an *undirected*, connected, acyclic graph

- Has v vertices and v-1 edges
- Any two vertices are connected by a unique path
- A leaf is a vertex of degree 1
- One node is designated as the root

- Each node has parent and/or children pointers
- A node's height is the length of its path to the root

- A forest has multiple distinct trees (a disjoint union)
- An n-ary tree has at most n children per node

CATALYST

Binary Tree

has nodes with at most 2 children (left & right)

- **Full**: every node has 0 or 2 children
  - # of nodes is at most 2^(h+1)-1
- **Complete**: every level, except possibly the last, is filled, and the last level's nodes are as far left as possible
  - # of internal nodes: floor(n/2)
- **Balanced**: has the minimum possible maximum depth
  - Height is ceil(lg(n+1))

- **Pre-order**: root, left, right
- **In-order**: left, root, right (returns sorted list)
- **Post-order**: left, right, root
- **Level-order**: breadth-first traversal, level by level

Binary Search Tree **is an ordered binary tree**

- Satisfies the **BST property**: <u>each node's value is greater than all keys stored in the left subtree and less than all keys stored in the right subtree</u>
- Designed to make searching ***faster***--each comparison allows operations to skip about half the tree (search in b-tree is O(n), BST is O(logn))

- **Search**: recursively search subtrees; takes O(h)
- **Insertion**: like search, but insert node when a leaf is reached; takes O(h)
- **Deletion**: more complicated; takes O(h)

2

Common Strategies

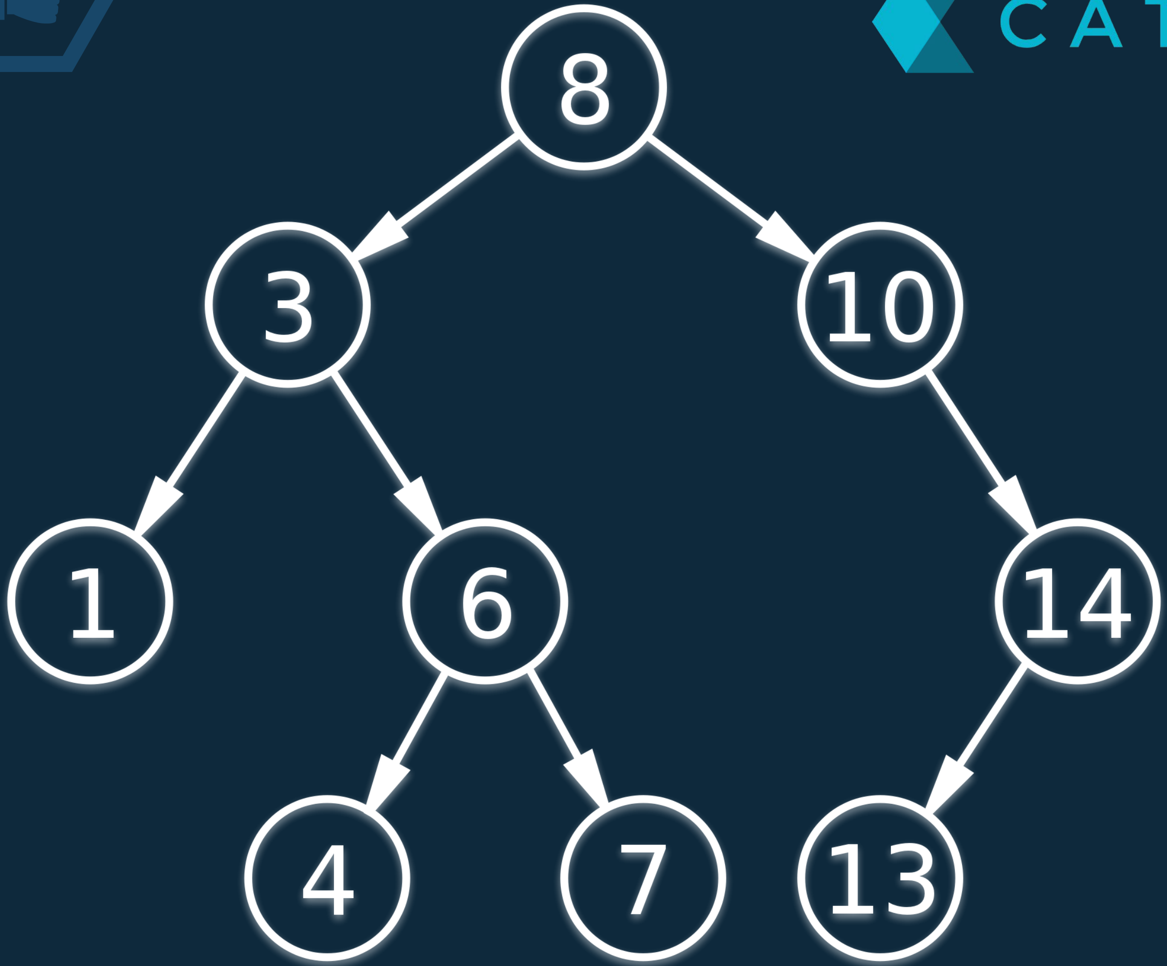
Hint: how do I go up or down a tree?

Tree traversal

Pre-order, In-order, Post-order, Level-order, DFS and BFS

- Types of tree traversal
- Iterative preorder, inorder, postorder traversal instead of recursion
- Finding depth of binary tree
- Example of level-order traversal
- A note on BFS / DFS

CATALYST



3

Interview Questions

How hard can it get?

Q1: Given a tree, give me it's pre, in, and post order traversals. Then, pick one type of traversal as input and write an algorithm to construct a binary search tree.

CATALYST

Q2: BST Validator

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

```
    2
   / \
  1   3
```

Binary tree `[2,1,3]`, return true.

Q3: kth Smallest Element in BST

Given a binary search tree, write a function 'kthSmallest' to find the **k**th smallest element in it.

For example:

Given the below binary tree,

```
    1
   / \
  2   3
```

Return 6.