

## Welcome to Lab Trees!

Course Website: <https://courses.engr.illinois.edu/cs225/sp2019/>

### Overview

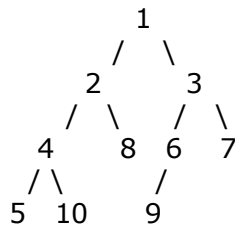
In this week's lab, you will work with binary trees, learn about tree traversals and implement them using iterators, thinking recursively while working with helper functions.

### Tree Traversals

Tree traversals refer to the process of **visiting each node in the tree data structure exactly once**. The order in which the nodes are visited determines the type of traversal:

- **Inorder:** left subtree -> current node -> right subtree (useful for checking if the tree is a Binary Search Tree)
- **Preorder:** current node -> left subtree -> right subtree (useful for copying the tree)
- **Postorder:** left subtree -> right subtree -> current node (useful for deleting the tree)

**Exercise 1.1:** What will an Inorder traversal of the following tree print out if we start at the root (node 1)??



**Exercise 1.2:** Write the printInOrder() function. The function should print the values of the nodes in in-order traversal to the standard out.

```

tree.h
1 #pragma once
2 class Tree{
3     public:
4         struct Node {
5             int value;
6             Node* left;
7             Node* right;
8             Node(int value = 0, Node left = NULL,
9                 Node right = NULL):
10                value(value), left(left), right(right) {}
11        };
12        Node* root;
13        int findLargest();
14        void printInOrder(const Node* subRoot) const;
15
16    private:
17        int findLargest(const Node* subRoot) const;
18};
  
```

```

tree.cpp
1 #include "tree.h"
2 void Tree::printInOrder(const Node* subRoot) const
3 {
4     //YOUR CODE HERE
5
6
7
8
9
10
11 }
  
```

**Exercise 1.3:** What is the worst case running time (in big O) of the following operations, assuming you start at the root node?

- Printing the value of the root node
- Printing the value of any leaf node
- Finding the node with the largest value

## Using a Helper Function

When using recursive functions, we would like the externally used function call to directly call the functions on their object without needing to know/provide information about subproblems and parameters. This is why we use helper functions: they allow us to pass in parameters, separately handle the subproblems, and make our code much more user-friendly and readable.

**Exercise 2:** Now that we know the runtime of finding the largest value in a binary tree from the previous exercise, write a function that will find the largest node value. In the code below, `findLargest()` is a public member function of the `Tree` class introduced in **Exercise 1**. Assume the values are non-negative.

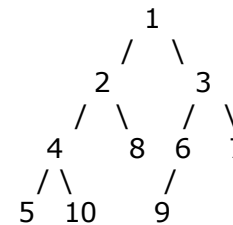
```
tree.cpp (CONTINUED)
1 int Tree::findLargest() {
2
3
4
5
6
7
8
9 }
10
11 int Tree::findLargest(const Node* subRoot) const{
12
13
14
15
16
17
18
19
20
21
22 }
```

## Iterators

In the previous exercise, we saw how to traverse trees using recursion. Another way to do this is by using a temporary data structure such as a **stack** to keep track of where we are as we traverse the tree. You will use this method when implementing tree traversal iterator classes in the coding part of this lab.

**Exercise 3.1:** Fill in the blanks in this pseudo code implementation of Inorder traversal using a stack as a placeholder for the traversal.

- 1) Create an empty stack `S`.
- 2) Initialize current node as root
- 3) Until `current = ____`, push current node to `S` and set `current = _____`
- 4) If `current = NULL` and `S` is not empty then
  - a) Print the top item in the stack
  - b) Set `current = _____`
  - c) Pop the top item from stack
  - d) Go to step 3
- 5) If `current = ____` and \_\_\_\_\_ then we are done



**Exercise 3.2:** Suppose we run this pseudo code on the tree above, what does the stack `S` look like when `current` is pointing to node `9`?  
`S` =

