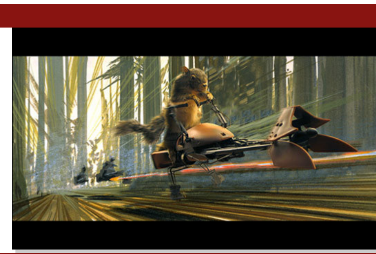
Lecture 18:
Tree Traversals

PIC 10B
Todd Wittman

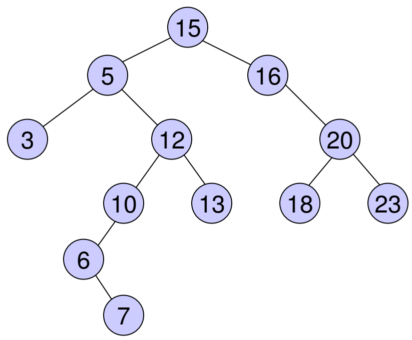


Tree Traversals

- It's unclear how we should print a tree.
- Top to bottom? Left to right?
- A tree traversal is a specific order in which to trace the nodes of a tree.
- There are 3 common tree traversals.
  1. in-order: left, root, right
  2. pre-order: root, left, right
  3. post-order: left, right, root
- This order is applied recursively.
- So for in-order, we must print each subtree's left branch before we print its root.
- Note "pre" and "post" refer to when we visit the root.

Tree Traversal Example

- Let's do an example first...



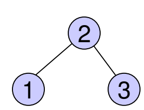
- in-order: (left, root, right)
  3, 5, 6, 7, 10, 12, 13,
  15, 16, 18, 20, 23
- pre-order: (root, left, right)
  15, 5, 3, 12, 10, 6, 7,
  13, 16, 20, 18, 23
- post-order: (left, right, root)
  3, 7, 6, 10, 13, 12, 5,
  18, 23, 20, 16, 15

In-Order Traversal

- The in-order traversal is probably the easiest to see, because it sorts the values from smallest to largest.

```
template <typename T>
void Tree<T> :: printInOrder (std::ostream& out, TreeNode<T>* rootNode)
    {
    if (rootNode != NULL) {
            printInOrder (out, rootNode->left);
            out << (rootNode->data) << "\n";
            printInOrder (out, rootNode->right);
    }
    return;
}
```

ostream is a name in std namespace.
The std:: means that we dont have to use the std namespace to use this class.



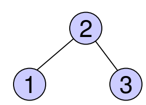
In-order: 1 2 3

- Example call in main: myTree.printInOrder (cout, myTree.getRoot());

Pre-Order Traversal

- Pre-order traversal prints in order: root, left, right.

```
template <typename T>
void Tree<T>::printPreOrder(std::ostream& out, TreeNode<T>* rootNode) {
    if (rootNode != NULL) {
          out << (rootNode->data) << "\n";
          printPreOrder (out, rootNode->left);
          printPreOrder (out, rootNode->right);
    }
    return;
}
```

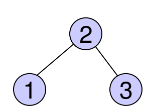


Pre-order: 2 1 3

Post-Order Traversal

- Post-order traversal prints in order: left, right, root.
- It is also called a depth-first search.

```
template <typename T>
void Tree<T>::printPostOrder(std::ostream& out, TreeNode<T>* rootNode) {
    if (rootNode != NULL) {
          printPostOrder (out, rootNode->left);
          printPostOrder (out, rootNode->right);
          out << (rootNode->data) << "\n";
    }
    return;
}
```



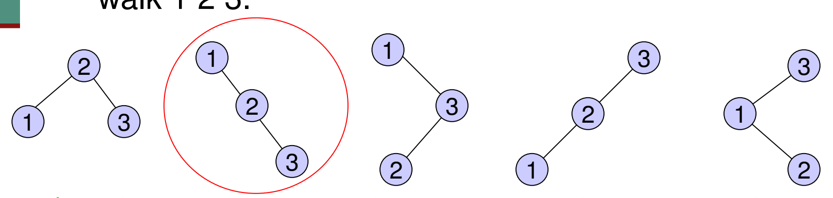
Post-order: 1 3 2

Sorting Values Using In-Order

- The in-order traversal always prints the values in sorted order from smallest to largest.
- One application of the in-order traversal is sorting a list.
- How long would it take to sort a list?
- Each insert operation takes O(h) time.
- So doing N inserts would take O(Nh) time.
- The in-order traversal is O(N), so building a tree and printing its values in sorted order takes: O(Nh) + O(N) = O(Nh) time.

Storing Trees Using Pre-Order

- Suppose we want to transmit our tree across the country to another programmer.
- Sending the in-order list would tell them the values, but would not communicate how the tree is built.
- Trees are usually stored with the pre-order traversal.
- Ex All of the tree below have the in-order walk: 1 2 3. But only one of the trees below has the pre-order walk 1 2 3.

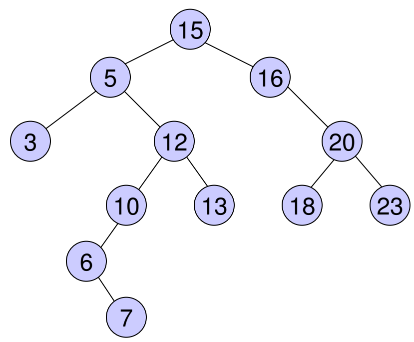


Pre-order: 2 1 3 1 2 3 1 3 2 3 2 1 3 1 2

Storing Trees Using Pre-Order

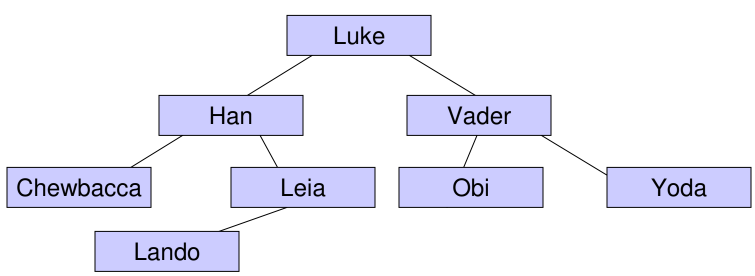
- Ex Can you recover the binary tree from its pre-order traversal?

15, 5, 3, 12, 10, 6, 7, 13, 16, 20, 18, 23



Tree Traversal Example

- Given a tree, you are expected to know how to do the in-, pre-, and post-order traversals.
- Ex Write the 3 traversals of the given tree.



In-order: Chewbacca, Han, Lando, Leia, Luke, Obi, Vader, Yoda
Pre-order: Luke, Han, Chewbacca, Leia, Lando, Vader, Obi, Yoda
Post-order: Chewbacca, Lando, Leia, Han, Obi, Yoda, Vader, Luke

Summary of Trees

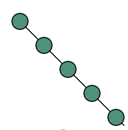
- Compared to vectors and linked lists, trees have a running time somewhere in between the best and worst.

	Vector	Linked List	Binary Tree
Insert / Erase *(at known position)*	O(N)	O(1)	O(h)
Indexing *(look up element)*	O(1)	O(N)	O(h)
Finding an Element	O(N)	O(N)	O(h)

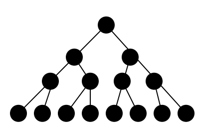
- But what is h in terms of N?

Best & Worst Height

- In the **worst case**, the tree is completely unbalanced.



- The height $h = N-1 = O(N)$.
- In the **best case**, the tree is perfectly balanced.



- Fact: A completely full tree with height h has $N = 2^{h+1}-1$ nodes.
- Solving for h gives $h = \log(N+1)-1 = O(\log N)$.
- What's the average height?

Average Height

- Let's look at a randomly built tree: a tree built from random numbers inserted in random order.
- <u>Theorem</u> The average height h of an randomly built tree with N nodes satisfies

$$h \le 2(\beta+1)\left(\sum_{i=1}^{N}\frac{1}{i}\right)\left(1-\frac{2}{N}\right)+2$$

where $\beta \approx 4.3191366$ solves the equation

$$(\ln\beta - 1)\beta = 2$$

- So on average, h = O(logN).
- So tree operations are on average O(logN).