Pre-order: 33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
Trees: Post-order

Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33
A **binary tree** is an ordered tree in which each node has at most two children. It is called **proper** if each internal node has two children (**left** and **right child**).
A **binary tree** is an ordered tree in which each node has at most two children. It is called **proper** if each internal node has two children (**left** and **right child**).

**Methods**

- `leftChild(v)`: return the left child of `v` if `v` is internal.
- `rightChild(v)`: return the right child of `v` if `v` is internal.
A **binary tree** is an ordered tree in which each node has at most two children. It is called **proper** if each internal node has two children (**left** and **right child**).

**Methods**

- `leftChild(v)`: return the left child of $v$ if $v$ is internal.
- `rightChild(v)`: return the right child of $v$ if $v$ is internal.

A third traversal order: **inorder**.
Traversals of Binary Trees

**Algorithm** \( \text{bPreorder}(T, v) \)

"visit" the node \( v \)

\[\text{if } v \text{ is internal then} \]

\[\text{bPreorder}(T, T.\text{leftChild}(v)) \]

\[\text{bPreorder}(T, T.\text{rightChild}(v)) \]

\[\text{end if} \]

**Algorithm** \( \text{bPostorder}(T, v) \)

\[\text{if } v \text{ is internal then} \]

\[\text{bPostorder}(T, T.\text{leftChild}(v)) \]

\[\text{bPostorder}(T, T.\text{rightChild}(v)) \]

\[\text{end if} \]

"visit" the node \( v \)
Binary Tree: Inorder

Algorithm blnorder( T, v )
if v is internal then
    bPostorder( T, T.leftChild(v) )
end if
"visit" the node v
if v is internal then
    bPostorder( T, T.rightChild(v) )
end if
Trees: In-order

In-order: 5, 10, 12, 15, 18, 20, 33, 36, 38, 39, 47, 49, 51
Identify Trees from preorder, inorder, postorder visits

- Pre-order: 33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
- Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33
- In-order: 5, 10, 12, 15, 18, 20, 33, 36, 38, 39, 47, 49, 51

How?

- Use Pre-order or Post-order to identify the root.
- Use In-order to identify both sub-trees.
- Apply the above procedure recursively.
Identify Trees from preorder, inorder, postorder visits

- Pre-order: 33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
- Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33
- In-order: 5, 10, 12, 15, 18, 20, 33, 36, 38, 39, 47, 49, 51

How?
- Use Pre-order or Post-order to identify the root.
Identify Trees from preorder, inorder, postorder visits

- Pre-order: 33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
- Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33
- In-order: 5, 10, 12, 15, 18, 20 , 33, 36, 38, 39, 47, 49, 51

How?

- Use Pre-order or Post-order to identify the root.
- Use In-order to identify both sub-trees.
Identify Trees from preorder, inorder, postorder visits

- Pre-order: 33, 15, 10, 5, 12, 20, 18, 47, 38, 36, 39, 51, 49
- Post-order: 5, 12, 10, 18, 20, 15, 36, 39, 38, 49, 51, 47, 33
- In-order: 5, 10, 12, 15, 18, 20, 33, 36, 38, 39, 47, 49, 51

How?
- Use Pre-order or Post-order to identify the root.
- Use In-order to identify both sub-trees.
- Apply the above procedure recursively.
Arithmetic Expression

(((3 + 1) \times 3)/((9 - 5) + 2)) - ((3 \times (7 - 4)) + 6)
Arithmetic Expression: Preorder

\[- \frac{3}{1} \times \frac{2}{3} - 9 + 5 \times \left( \frac{2}{3} - 7 \times 4 \right) + 6 \]
Arithmetic Expression: Preorder

\[ - \left( \frac{\times (+(3)(1))(3)}{+(-9)(5)(2))} \right) (\times (3)(-7)(4))(6) \]
Arithmetic Expression: Postorder

\[
\begin{align*}
\text{Postorder:} & \quad 3 - 7 \\
\text{Expression:} & \quad \frac{((3)(1)+)(3) \times (((9)(5)-2)+)}{(((3)((7)(4)-5) \times)(6)+)-6}\end{align*}
\]
Arithmetic Expression: Postorder

\[ (((3)(1)+)(3\times)((9)(5)-(2)+)/)((3)((7)(4)-)\times)(6)+)- \]
Arithmetic Expression: Inorder

\[- \frac{((3 + 1) \times 3)}{((9 - 5) + 2)} - ((3 \times (7 - 4)) + 6)\]
Arithmetic Expression: Inorder

\[ (((3 + 1) \times 3)/((9 - 5) + 2)) - ((3 \times (7 - 4)) + 6) \]
Theorem

Let $T$ be a (proper) binary tree with $n$ nodes, $h$ the height of $T$. We have

- # external nodes of $T$ is between $h + 1$ and $2^h$.
- # internal nodes of $T$ is between $h$ and $2^h - 1$.
- The height of $T$ is between $\log(n + 1) - 1$ and $(n - 1)/2$. 
Properties about Binary Trees

Theorem (Theorem 2.9, on page 85)

In a (proper) binary tree $T$, the number of external nodes is 1 more than the number of internal nodes.

Proof. By induction,

- If $T$ only has one node, it must be external. Thus, no internal node. The statement holds.
- Otherwise, $T$ has at least one external node with its parent. Remove any external node $w$ and its parent $v$, then connect $w$'s sibling to $v$'s parent. The tree remains proper and binary, but smaller.
Properties about Binary Trees

Theorem (Theorem 2.9, on page 85)

*In a (proper) binary tree* \( T \), *the number of external nodes is 1 more than the number of internal nodes.*

Proof.

By induction,

- If \( T \) only has one node, it must be external. Thus, no internal node. The statement holds.

- Otherwise, \( T \) has at least one external node with its parent. Remove any external node \( w \) and its parent \( v \), then connect \( w \)'s sibling to \( v \)'s parent. The tree remains proper and binary, but smaller.
Properties about Binary Trees

Theorem (Theorem 2.9, on page 85)

In a (proper) binary tree $T$, the number of external nodes is 1 more than the number of internal nodes.

Proof.
By induction,

- If $T$ only has one node, it must be external. Thus, no internal node. The statement holds.
- Otherwise, $T$ has at least one external node with its parent. Remove any external node $w$ and its parent $v$, then connect $w$’s sibling to $v$’s parent. The tree remains proper and binary, but smaller.
Properties about Binary Trees

Let $\#e$, $\#i$ be the external/internal nodes of a (proper) binary tree.

- $\#e = \#i + 1$ and $\#e + \#i = n$. 

- $n \geq 2^h + 1$. What is this case?

- $n \leq 2^h + 1 - 1$. What is this case?

- $\frac{n - 1}{2} \leq h \leq \log(n + 1) - 1$. 

- $h + 1 \leq \#e \leq 2^h$. 

- $h \leq \#i \leq 2^h - 1$. 

Properties about Binary Trees

Let \( e \), \( i \) be external/internal nodes of a (proper) binary tree.

- \( e = i + 1 \) and \( e + i = n \).
- \( n \geq 2h + 1 \). What is this case?
Properties about Binary Trees

Let \( \#e, \#i \) be \( \# \) external/internal nodes of a (proper) binary tree.

- \( \#e = \#i + 1 \) and \( \#e + \#i = n \).
- \( n \geq 2h + 1 \). What is this case?
- \( n \leq 2^{h+1} - 1 \). What is this case?
Properties about Binary Trees

Let \( \#e, \#i \) be \# external/internal nodes of a (proper) binary tree.

- \( \#e = \#i + 1 \) and \( \#e + \#i = n \).
- \( n \geq 2h + 1 \). What is this case?
- \( n \leq 2^{h+1} - 1 \). What is this case?
- \( \frac{n - 1}{2} \leq h \leq \log(n + 1) - 1 \).
- \( h + 1 \leq \#e \leq 2^h \).
- \( h \leq \#i \leq 2^h - 1 \).
Implementation of Binary Trees

Vector-based Structure

- \( p(v) \): the rank of \( v \) stored in array \( A \) of size \( N \).
- If \( v \) is the root, then \( p(v) = 1 \).
- If \( v \) is the left child of \( u \), then \( p(v) = 2p(u) \).
- If \( v \) is the right child of \( u \), then \( p(v) = 2p(u) + 1 \).
Implementation of Binary Trees

Vector-based Structure

- $p(v)$: the rank of $v$ stored in array $A$ of size $N$.
- If $v$ is the root, then $p(v) = 1$.
- If $v$ is the left child of $u$, then $p(v) = 2p(u)$.
- If $v$ is the right child of $u$, then $p(v) = 2p(u) + 1$.

- methods: leftChild(), rightChild(), root(), parent(), children(), $O(1)$ time.
- Space could be as large as $O(2^{(n+1)/2})$. 
Implementation of Binary Trees

Linked Structure: similar to doubly linked list

- Each node: pointers to parent, leftChild, rightChild, and the element stored.
- methods: leftChild(), rightChild(), root(), parent(), children(), $O(1)$ time.
- Space usage $O(n)$. 

Lab logistics

- Lab 1 due mid-night this Sunday (1/24).
- Lab 2: 3 weeks, on implementing Trees.
Implementation of Binary Trees

Linked Structure: similar to doubly linked list

- Each node: pointers to parent, leftChild, rightChild, and the element stored.
- methods: leftChild(), rightChild(), root(), parent(), children(), $O(1)$ time.
- Space usage $O(n)$.

Lab logistics

- Lab 1 due mid-night this Sunday (1/24).
- Lab 2: 3 weeks, on implementing Trees.