Binary Tree

Instructor: Ashok Singh Sairam

Lecture Plan

- Tree
  - Definitions
  - Subtree, height, depth, etc
- Binary Tree
  - Types
  - Properties
  - Traversal

Quicksort - clarification

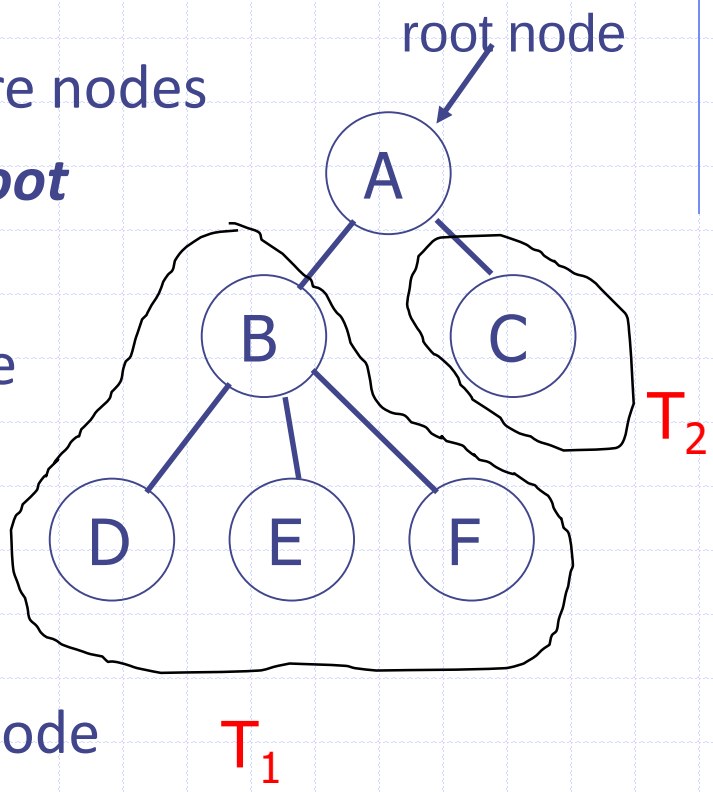
```
Algorithm PARTITION(A, p, r)
    x ← A[p]      // pivot is now the first element
    i ← r + 1

    for j ← r down to p+1 do
        if A[j] ≥ x then
            i ← i - 1
            exchange A[i] ↔ A[j]

    exchange A[i-1] ↔ A[p]   // Place pivot in final position
    return i-1               // Index of pivot
```

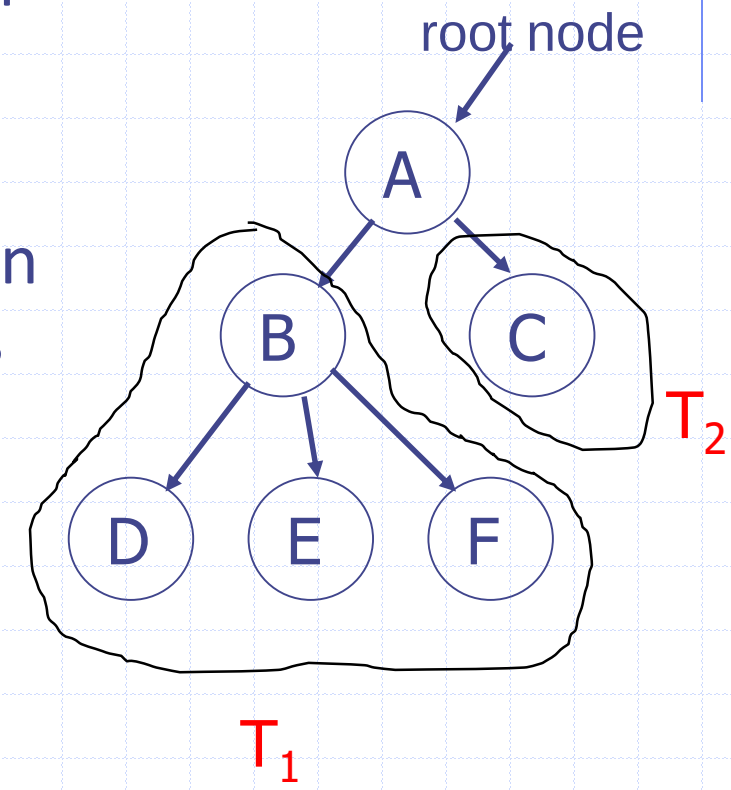
Definition: Tree

- A *tree (T)* is a finite set of one or more nodes
  - specially designated node called ***root***
  - remaining nodes partitioned into
    $n \geq 0$ disjoint sets $T_1, T_2, .., T_n$, where each is a tree (recursive defn)
  - $T_1, T_2, .., T_n$ are called subtrees of the root
- Degree of a node: #subtrees of the node
- Degree of a tree: Maximum of the degree of the nodes in the tree



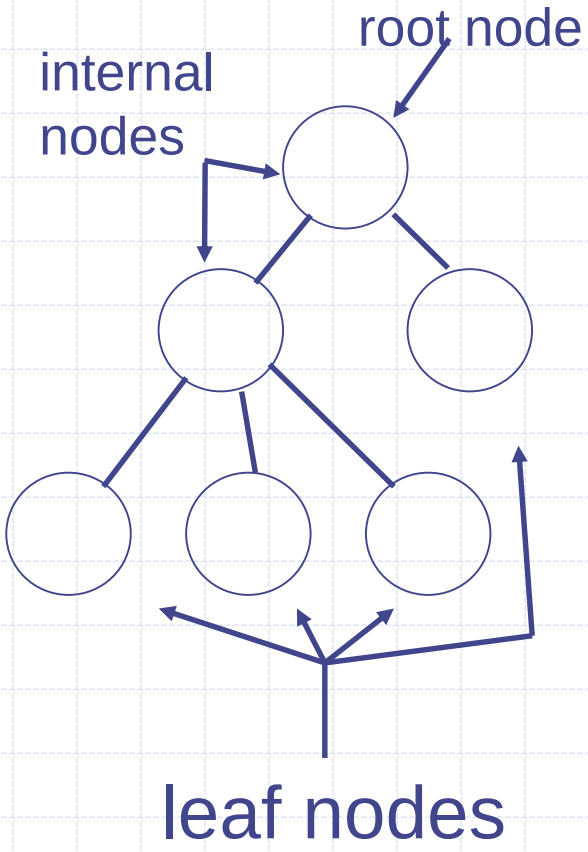
Subtree

- Subtree(x) : Consists of x and all its descendants, where x is the root of the tree
- *descendants*: any nodes that can be reached via 1 or more edges from this node
- *ancestors:* any nodes for which this node is a descendant
-



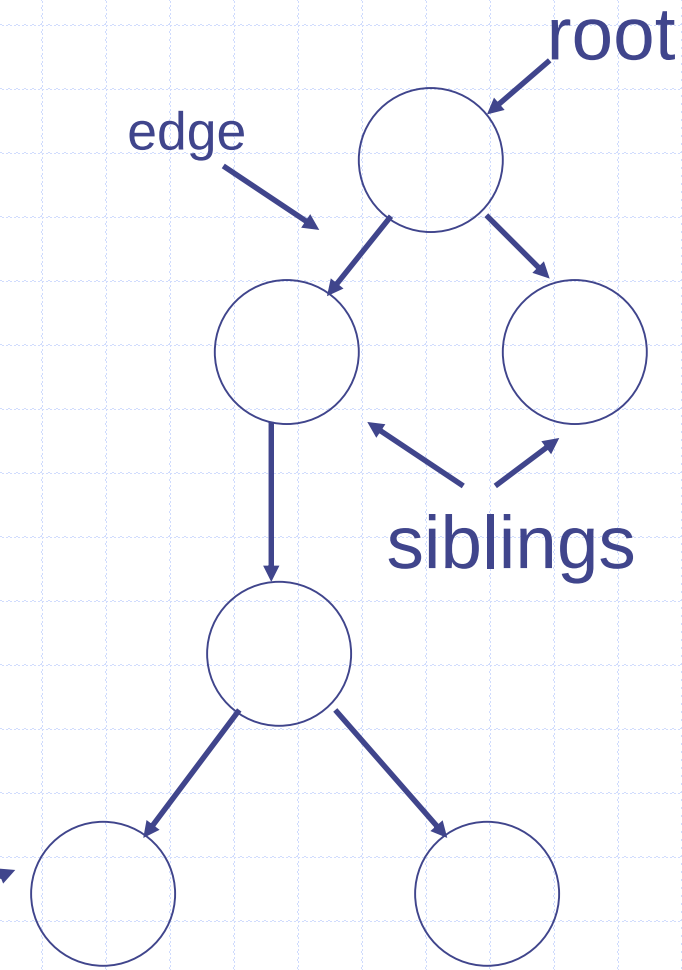
Internal and Leaf Node

- A *tree (T)* is an abstract data type
  - one entry point called ***root***
  - Each node is either a ***leaf*** or an *internal node*
  - An internal node has 1 or more ***children***, nodes that can be reached directly from that internal node.
  - The internal node is said to be the ***parent*** of its child nodes



Definition: Siblings, path length

- *siblings:* two nodes that have the same parent
- *edge:* the link from one node to another
- *path length:* the number of edges that must be traversed to get from one node to another

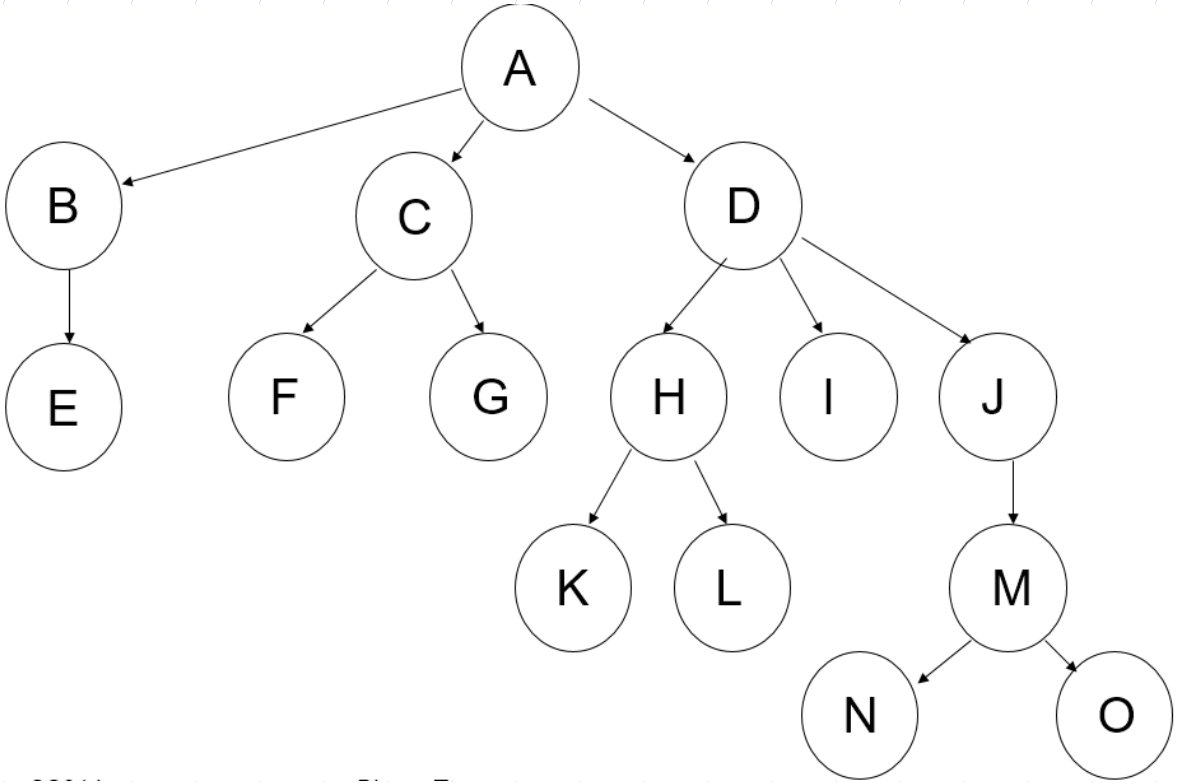


Definition: Depth, height

- Depth*:* path length from the root of the tree to this node
- Height of a node*:* The maximum distance (path length) of any leaf from this node
  - a leaf has a height of 0
  - the height of a tree is the height of the root of that tree
- Height of tree (h): height of root

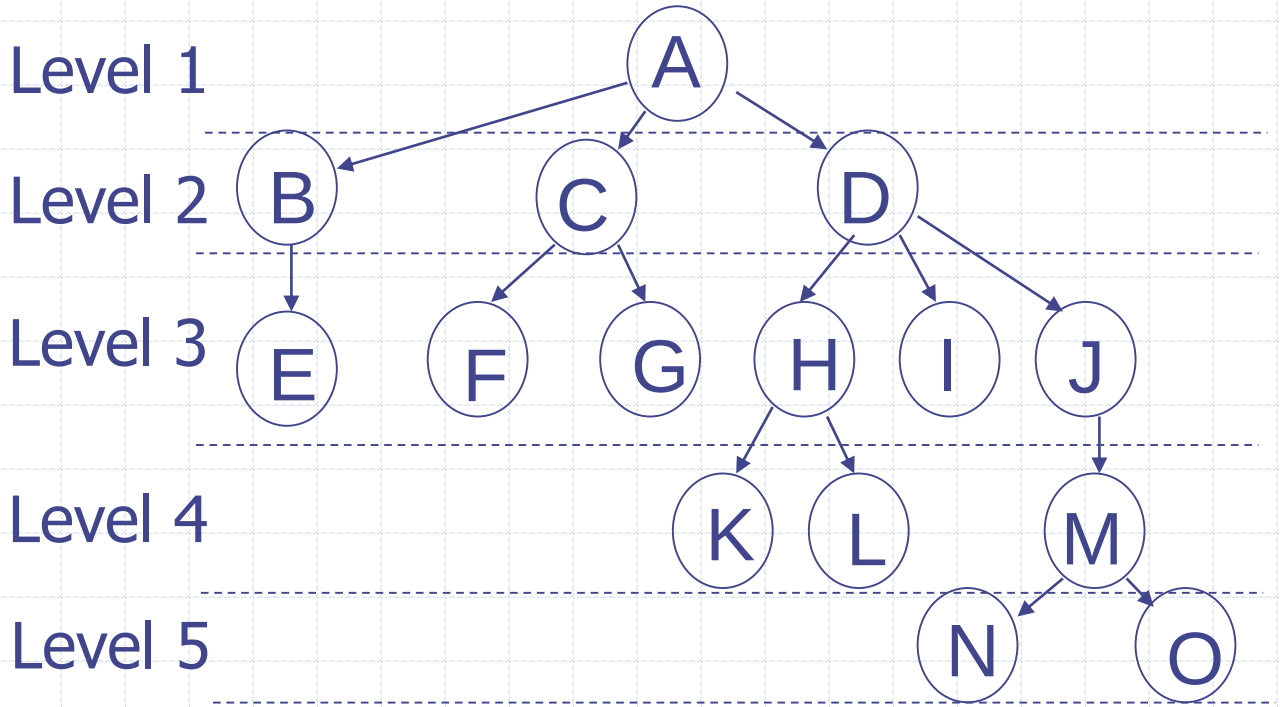
Example: Depth and Height

- What is the depth and height of node M, J and D?



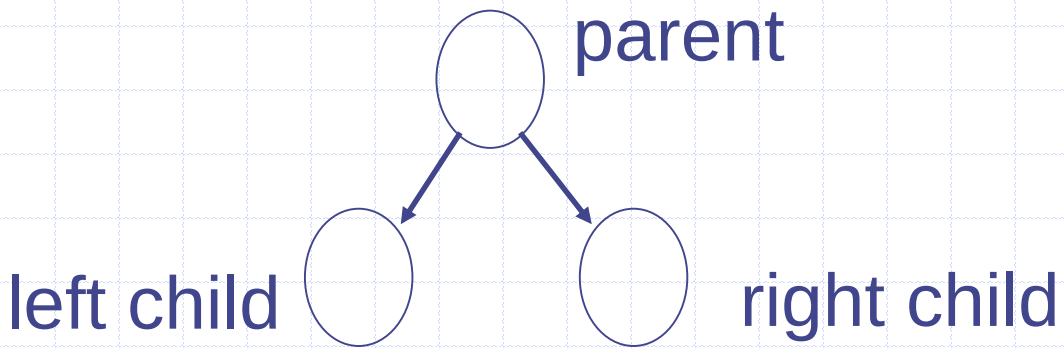
Level of a node

- Level(root) = 1
- Level(node)=l+1, l is the level of the node's parent



Binary Tree

- A finite set of nodes that either is empty or consists of a root and two disjoint binary trees – left and right subtree (aka as rooted BT)
  - each node has at most two children
- In general a node has a link to parent, a pointer to left child and a pointer to right child



Binary Tree vs. Regular Tree

Characteristic	Binary Tree	Regular Tree
Has Ø nodes	Yes	No
Order of children	Important	Doesn't matter

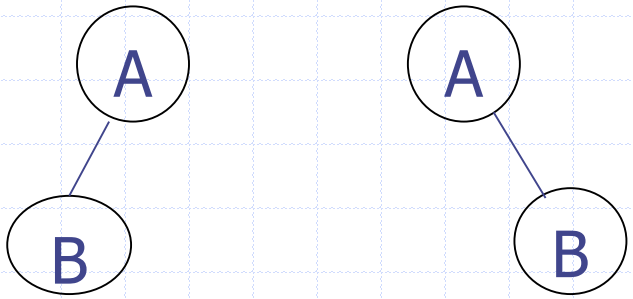


Fig.: Different binary trees

Special Binary Trees

- Skewed binary tree
- Full Binary Tree
- Complete Binary Tree

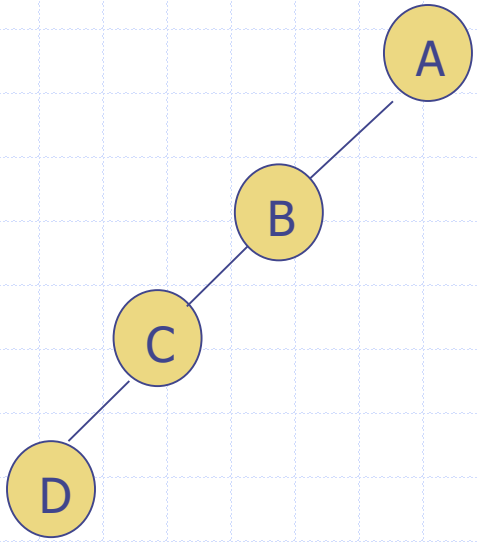


Fig 1: Skewed to the left

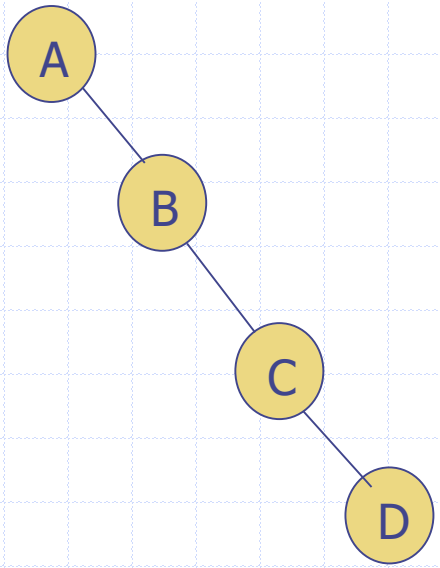
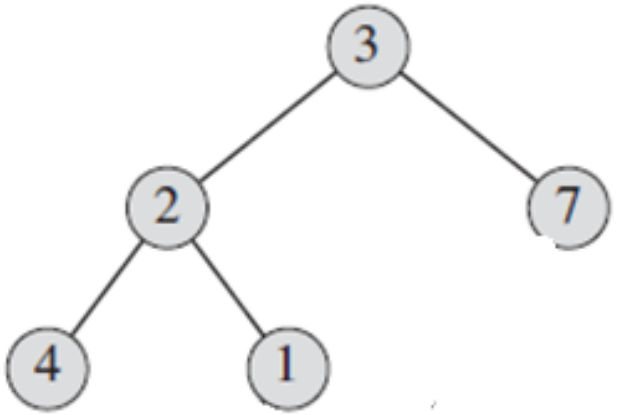


Fig 2: Skewed to the right

Full Binary Tree

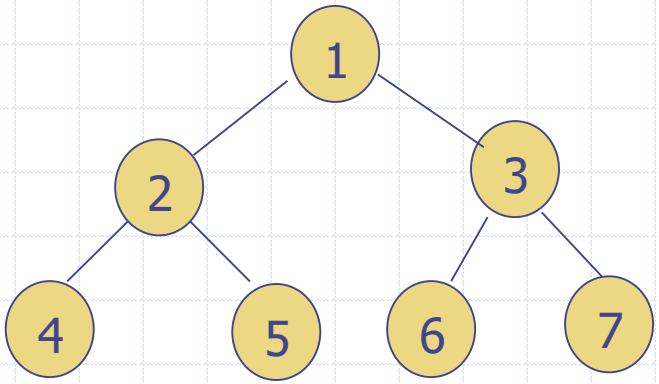
- **Full binary tree***:* Each node is either a leaf or has degree exactly two



Complete Binary Tree

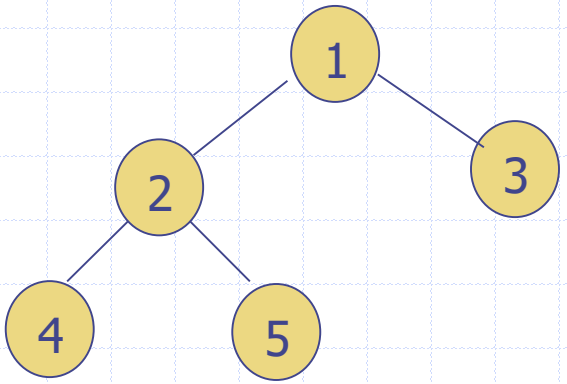
- **Complete binary tree***:* A binary tree of depth k with 2^{k+1}-1 nodes
  - That is a binary tree which has maximum nodes
- Also know as Full Tree

Height of a complete BT with n nodes?



Nearly Complete Binary Tree

- **Nearly Complete binary tree***:* a binary tree of depth k with n nodes is called complete *iff* its nodes correspond to the nodes numbered from 1 to n in the complete binary tree
  - That is every level, except possibly the deepest is completely filled.
  - At depth n, the height of the tree, all nodes are as far left as possible



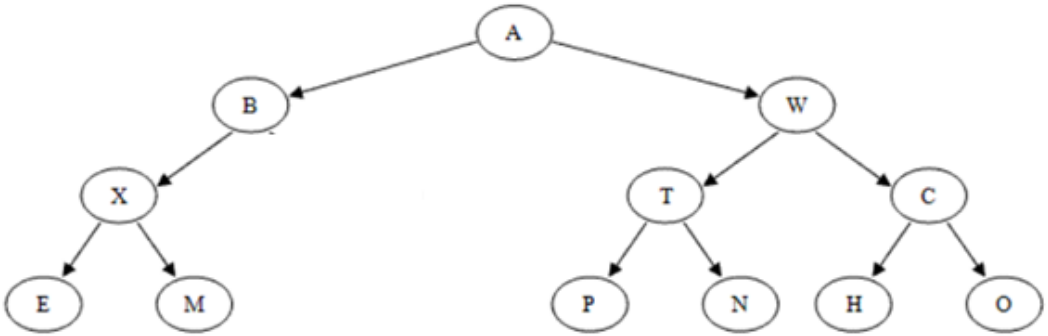
Property 1: Internal nodes of complete binary tree

- A complete binary tree has 2^h-1 internal nodes

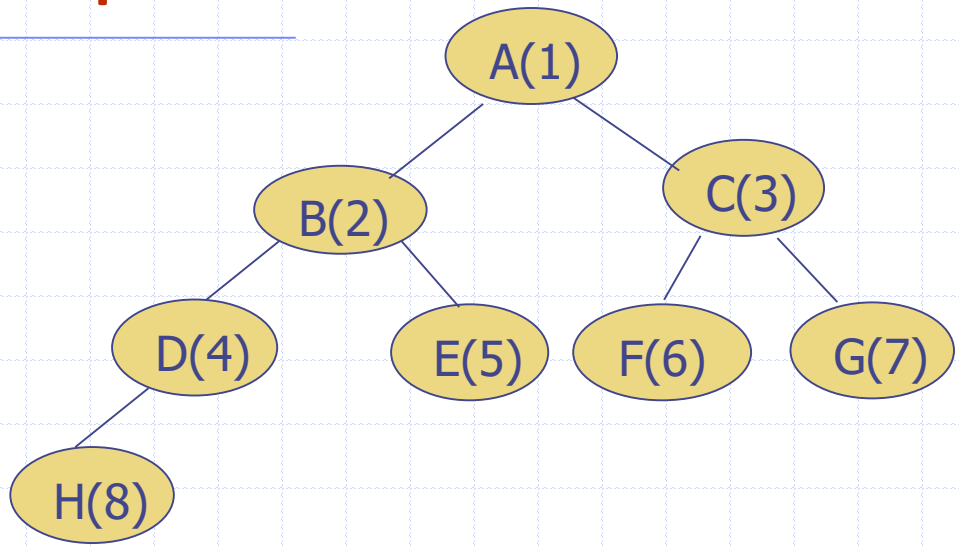
Property 2: Relation between #leaf node and degree 2 nodes

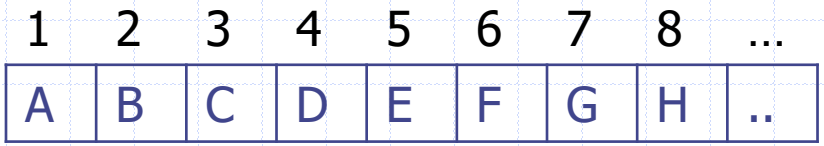
- n_0: #leaf nodes
- $n_0=n_2+1$

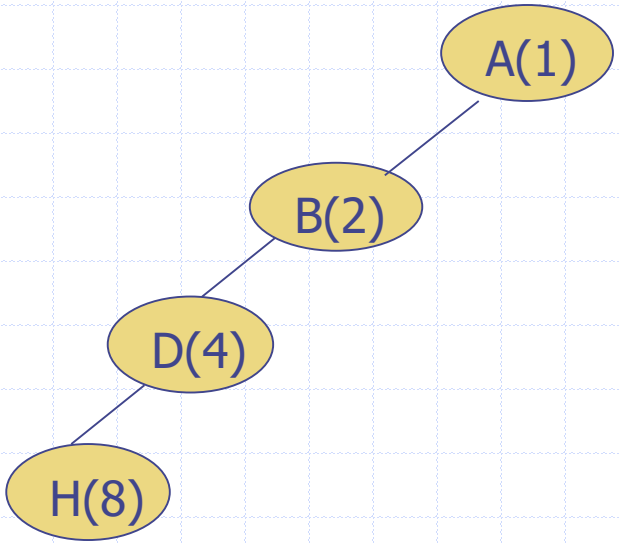
n_2: #degree 2 node

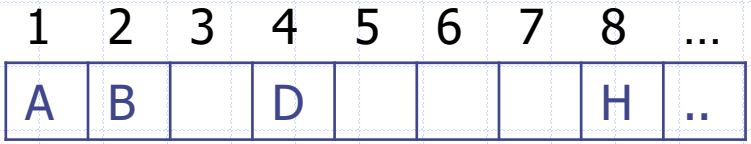


Representation of Trees









- The numbering scheme suggest that we can use a 1-D array to store the nodes

Representation of Trees: Arrays

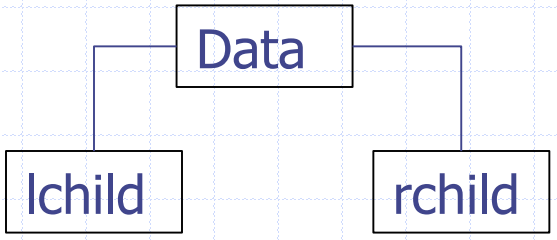
- Adv: Easy to determine the locations of the parent, left and right child of any node
  - $\text{parent(i)} = \left\lfloor \frac{i}{2} \right\rfloor$ if i $\neq 1$; if i=1, i is root has no parent
  - $\text{leftchild}(i) = 2\text{i}, \text{if } 2\text{i} \leq \text{n}$, if 2i > n, i has no left child
  - $\text{rightchild}(i) = 2\text{i} + 1, \text{if } 2\text{i} + 1 \leq \text{n}$, if 2i+1 > n, i has no right child
- Dis: (i) Wastage of storage for skewed tree

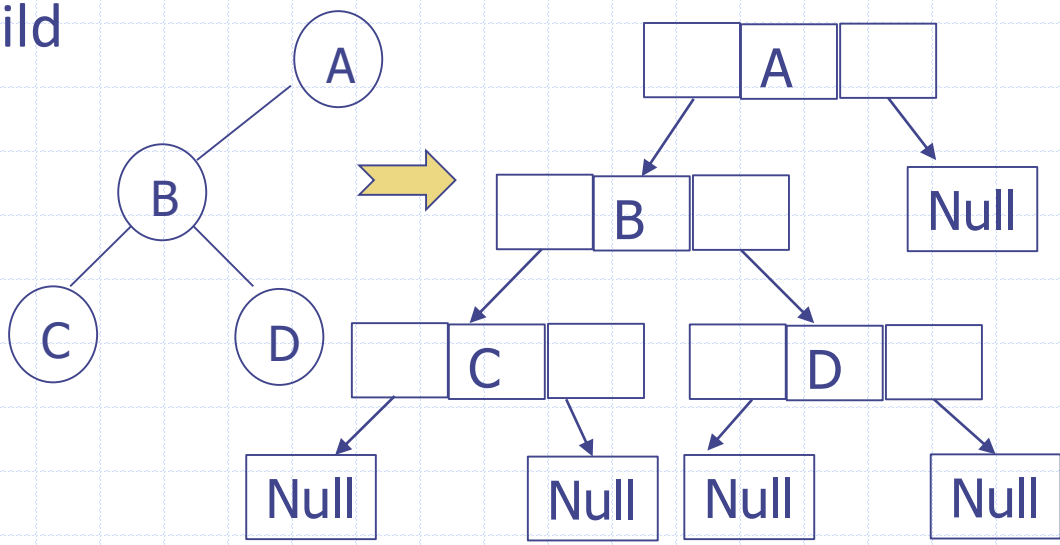
  (ii) Insertion and deletion of nodes require movement of a large part of nodes

Representation of Trees: Linked

- Linked Representation
  - Similar to linked list
  - Each tree consists of three fields – lchild, data, rchild

```
struct node{
        struct node *lchild;
                int data;
        struct node *rchild;
};
```

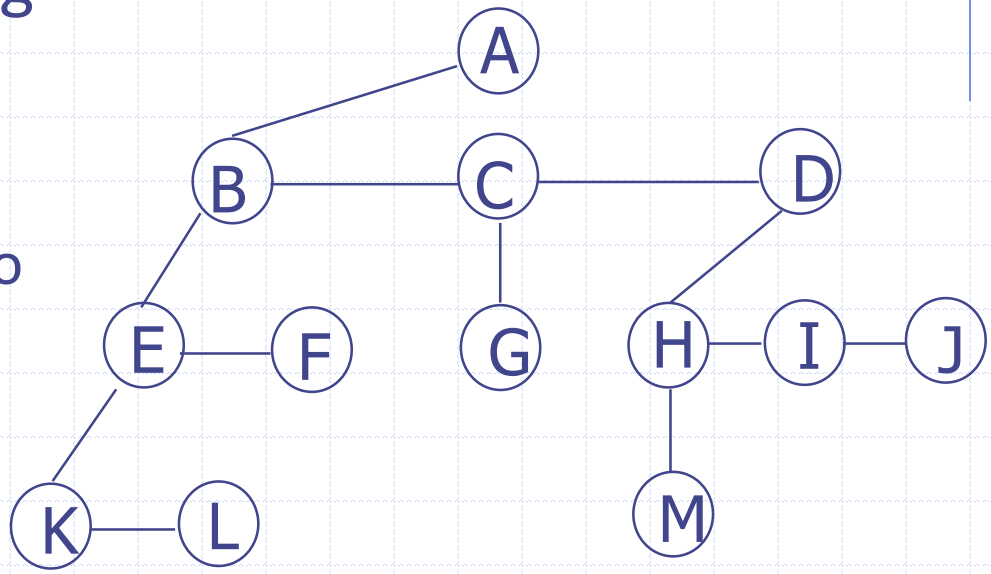




- Some links will be unused

Representation of Trees: Left Child – Right Sibling

- Left Child - Right Sibling
  - Each node has exactly two link fields
  - Left link (child): points to leftmost child node
  - Right link (sibling): points to closest sibling node



Binary Tree Traversals

- Traversing a tree means visiting each node exactly once
  - Treat each node and its subtrees in the same fashion
  - Let L, V, and R stand for moving left, visiting the node, and moving right
  - There are six possible combinations of traversal

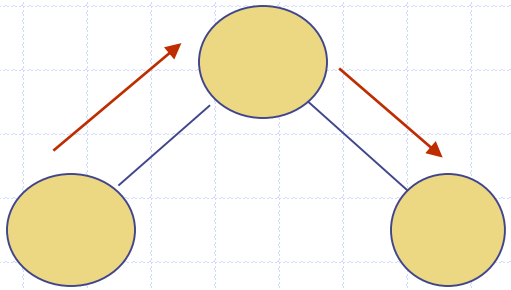
    - LVR, LRV, VLR, ~~VRL~~, ~~RVL~~, ~~RLV~~
  - Adopt convention that we traverse left before right, only 3 traversals remain
  - LVR (inorder), LRV (postorder), VLR (preorder)

Inorder Traversal

Recursive version

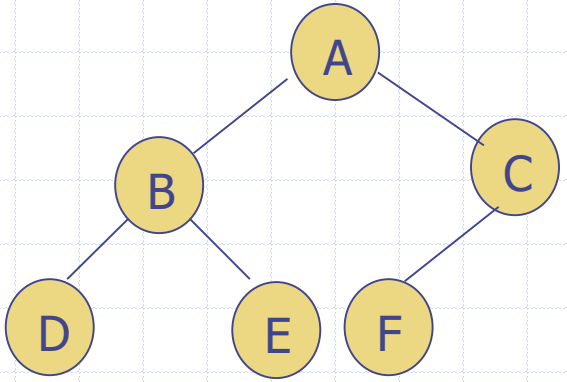
1. Move down the tree towards the left until you can go down no further
2. Visit the node
3. Move one node to the right and continue with step 1

```
void inorder(tree *ptr)
/* inorder tree traversal */
{
    if (ptr) {
      inorder(ptr->lchild);
      printf("%d", ptr->data);
      inorder(ptr->rchild);
     }
}
```



Successor in traversal order

- Successor(X) : Node which comes next after X in the inorder traversal



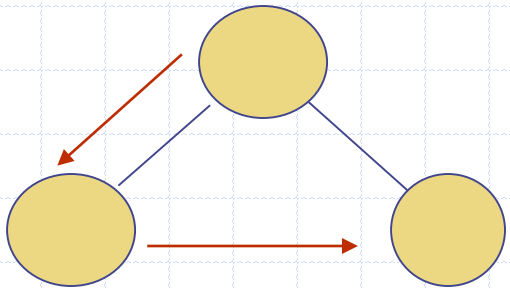
- Predecessor(X): Node which comes before X in the inorder traversal

Preorder Traversal

Recursive version

1. Visit the node
2. Move one node to the left and go step 1, continue until you can go down no further
3. Move one node to the right and continue with step 1

```
void preorder(tree *ptr)
/* preorder tree traversal */
{
    if (ptr) {
      printf("%d", ptr->data);
      preorder(ptr->lchild);
      preorder(ptr->rchild);
     }
}
```



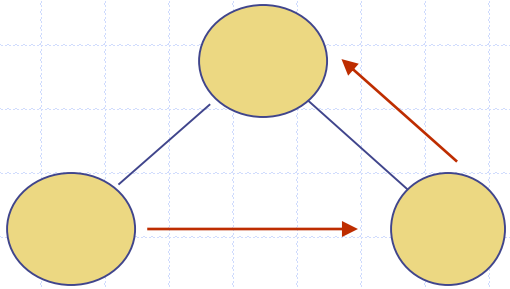
Postorder Traversal

Recursive version

1. Move down the tree towards the left until you can go down no further
2. Move one node to the right
3. Visit the node and continue with step 1

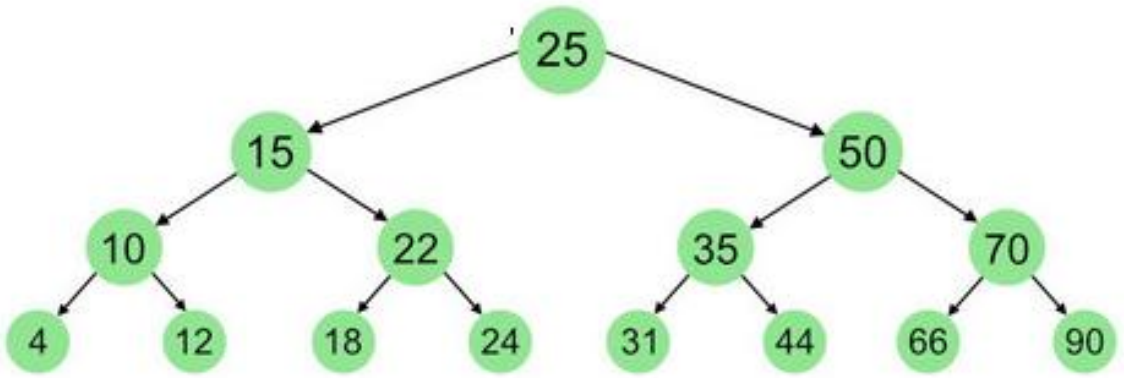
```
void postorder(tree *ptr)
/* postorder tree traversal */
{
    if (ptr) {
      postorder(ptr->lchild);
      postorder(ptr->rchild);
      printf("%d", ptr->data);

    }
}
```



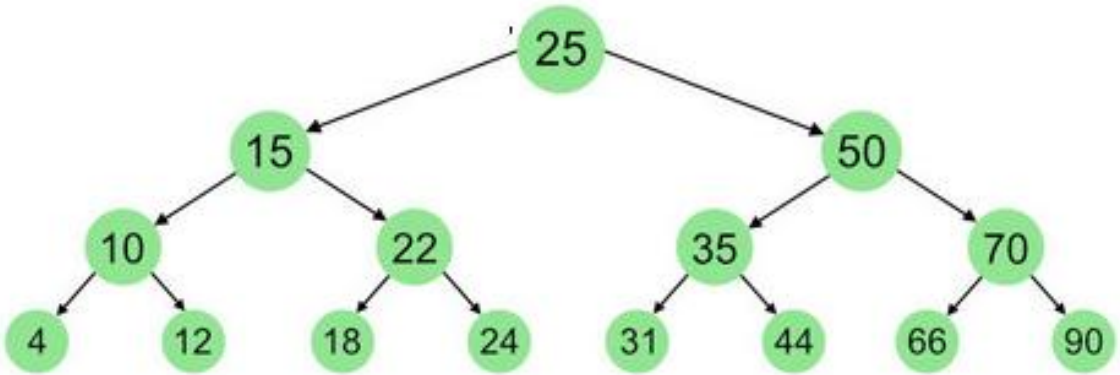
Example: Tree Traversal

- Inorder -
- Preorder -
- Postorder -



Example: Tree Traversal

- Inorder – 4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90
- Preorder – 25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90
- Postorder – 4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Exercise

- Show by induction that any binary tree of height h has at <u>most</u> 2^h leaves

B.5-3

Show by induction that the number of degree-2 nodes in any nonempty binary tree is 1 fewer than the number of leaves. Conclude that the number of internal nodes in a full binary tree is 1 fewer than the number of leaves.

B.5-4

Use induction to show that a nonempty binary tree with n nodes has height at least $\lfloor \lg n \rfloor$.