CS2x1:Data Structures and Algorithms

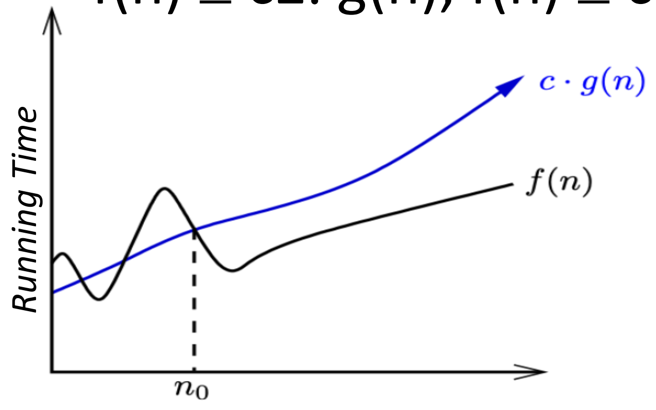
Koteswararao Kondepu

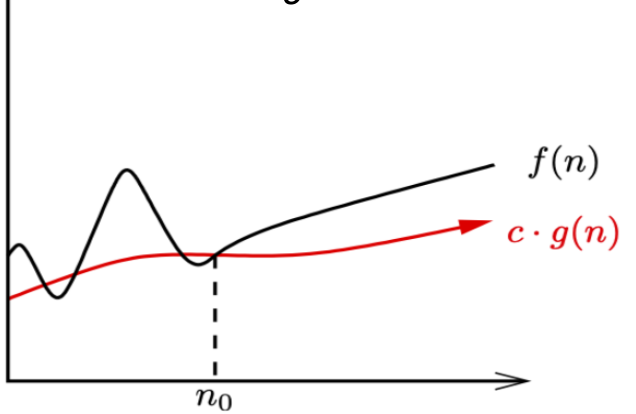
k.kondepu@iitdh.ac.in

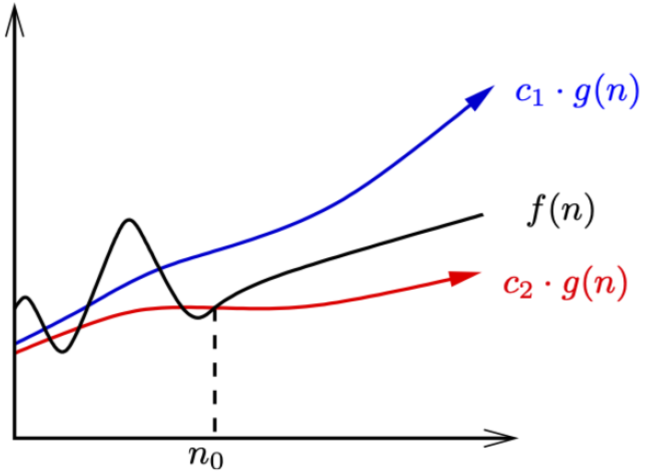
Recap

Asymptotic Notations	Symbol
Worst-case analysis	big-Oh → O−Notation
Average-case analysis	big-Theta → Θ−Notation
Best-case analysis	big-Omega → Ω−Notation

- big-Oh → O
  - **Definition**: $f(n) = O(g(n))$, if there are positive constants c and n_0 such that $0 \leq f(n) \leq c.\ g(n)$ for all $n \geq n_0$; $c > 0$
- big-Omega → Ω
  - **Definition**: f(n) = Ω (g(n)), if there are positive constants c and n_0 such that $f(n) \geq c.\ g(n)$ for all $n \geq n_0$; $c > 0$; $n_0 \geq 1$
- big-Theta → Θ
  - **Definition**: f(n) = Θ (g(n)), if there are positive constants c and n_0 such that $0 \leq c1.\ g(n) \leq f(n) \leq c2.\ g(n)$, $f(n) \geq c.\ g(n)$ for all $n \geq n_0$; c1, c2> 0; $n_0 \geq 1$







Exercise: Asymptotic Analysis

- ```
  for(i=1;i<=n;)
        i = i * 2; //statement
  ```

- ```
  k=0;
  for (i=1; i<n; i=i*2)
        k++;
  for(j=1; j<k; j=j*2)
        s= s+1; //statement
  ```

Exercise: Asymptotic Analysis (2)

- ```
  for (i=1; i<n; i++)
   for(j=1; j<n; j=j*2)
      s= s+1; //statement
  ```

- ```
  void fun (int n) {
    int i=0, s=0;
    while (s<=n) {
        i++;
        s=s+i;
     }
  }
  ```

Asymptotic Notation: little-Oh → o

- **Definition**: $f(n) = o\ (g(n))$, if there are positive constants c and n_0 such that $0 \leq f(n) < c.\ g(n)$ for all $n \geq n_0$; $c > 0$; $n_0 > 0$

- *g(n) is an upper bound for f(n) that is not asymptotically tight.*

  - $f(n) = o\ (g(n))$
    - ✓ $2n = o\ (n^2)$
    - ✓ $2n^2 \neq o\ (n^2)$

- *f(n) becomes arbitrarily large relative to g(n) as n approaches infinity:*

$$\lim_{n \to \infty} [f(n) / g(n)] = \infty$$

Asymptotic Notation: little-Omega → ω

- **Definition**: *f(n) = w (g(n))*, if there are positive constants *c* and n_0 such that $0 \leq c.\ g(n) < f(n)$ for all $n \geq n_0$; $c > 0$; $n_0 > 0$

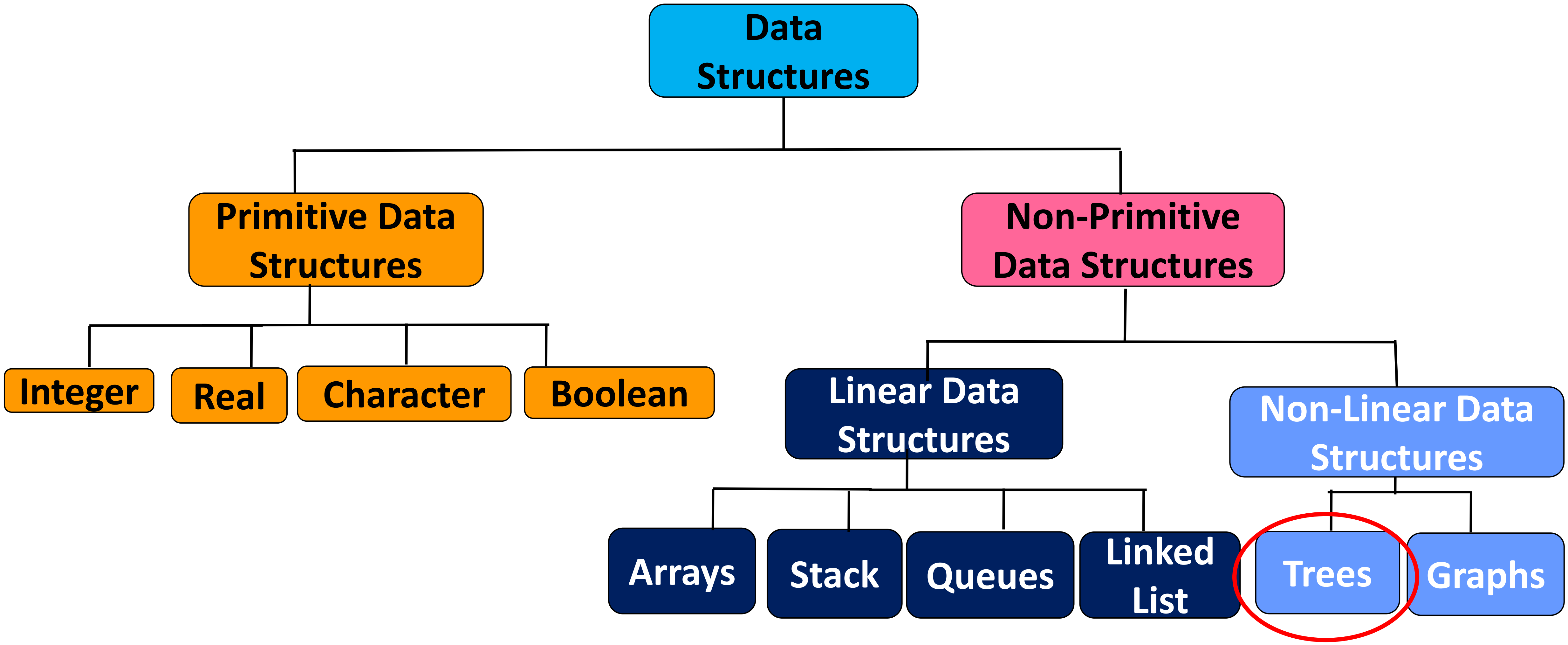
- *g(n) is a <u>lower bound for</u> f(n) that is <u>not asymptotically tight</u>.*

  - *f(n) = w (g(n))*
    - ✓ $n^2/2 = w(n)$
    - ✓ $n^2/2 \neq w(n^2)$

- *f(n) becomes arbitrarily large relative to g(n) as n approaches infinity:*

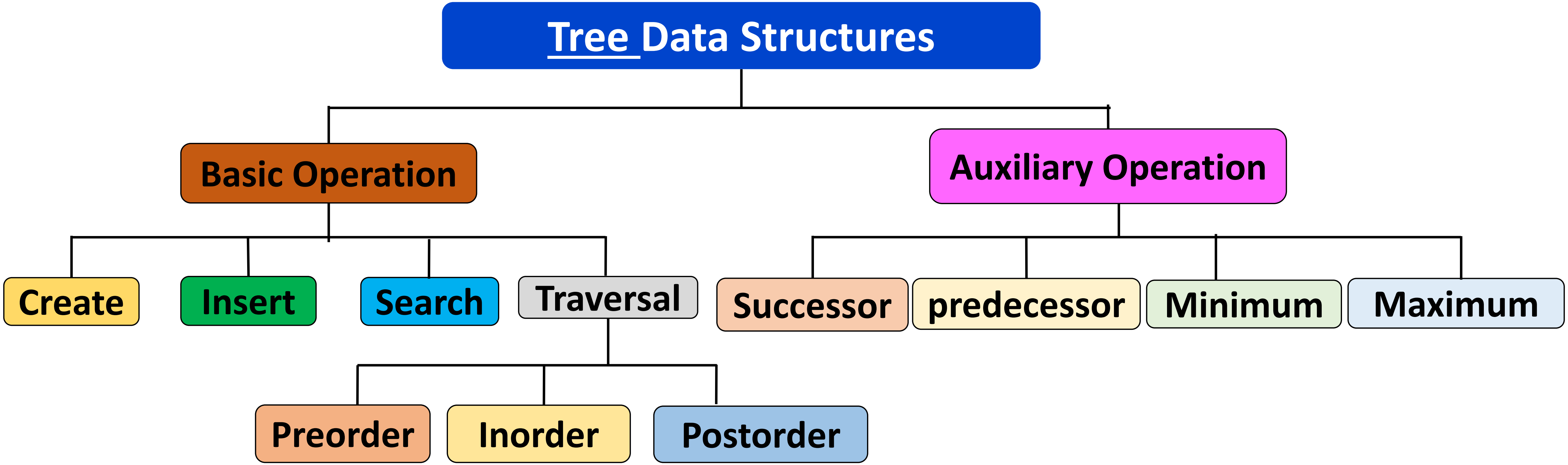
$$\lim_{n \to \infty} [f(n) / g(n)] = \infty$$

Classification of Data Structures



Tree Data Structures

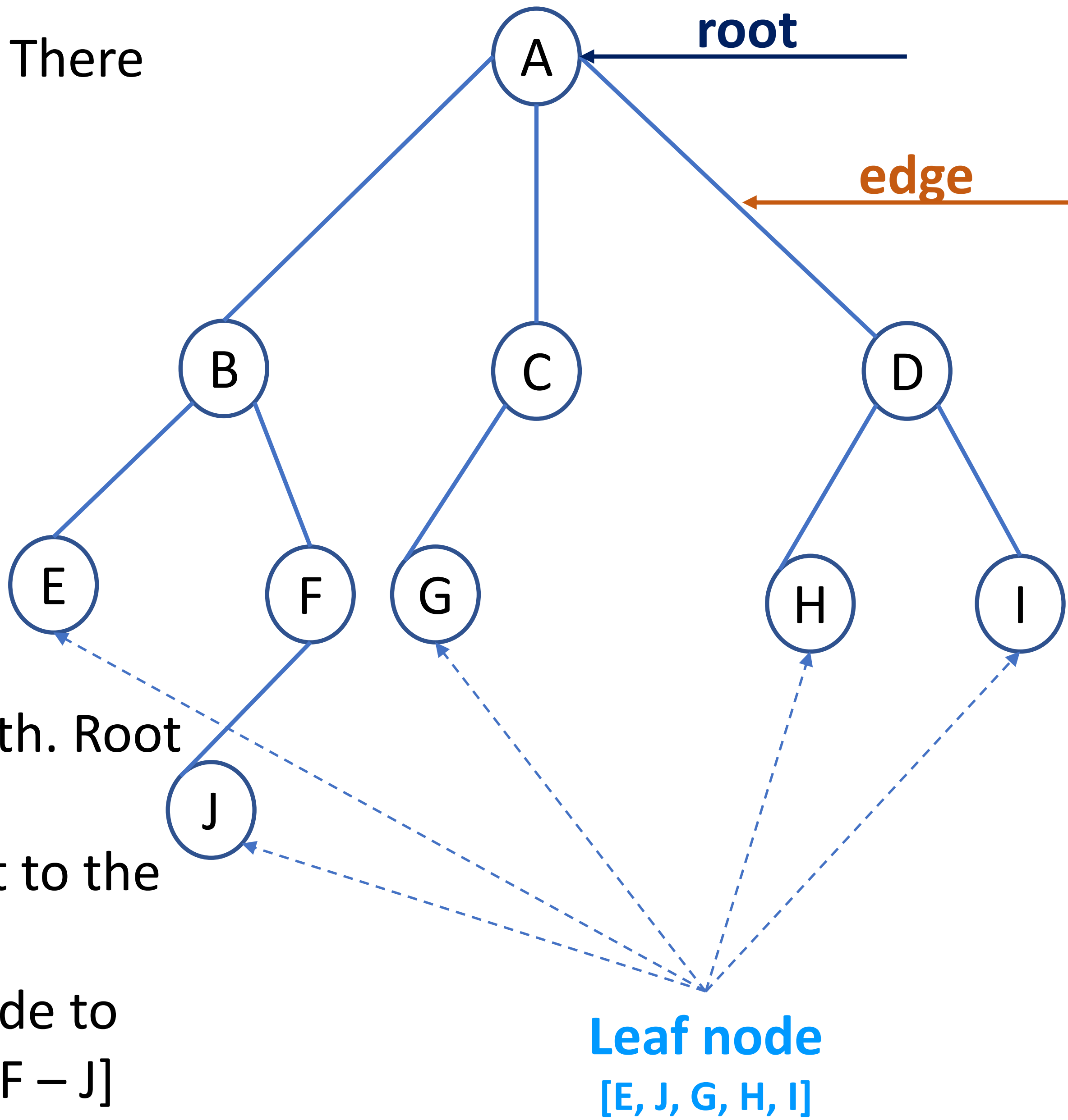
- Non-linear data structure
- A *data structure* in which each element is dynamically allocated and has more than one potential *successor*



Tree: Terminology (1)

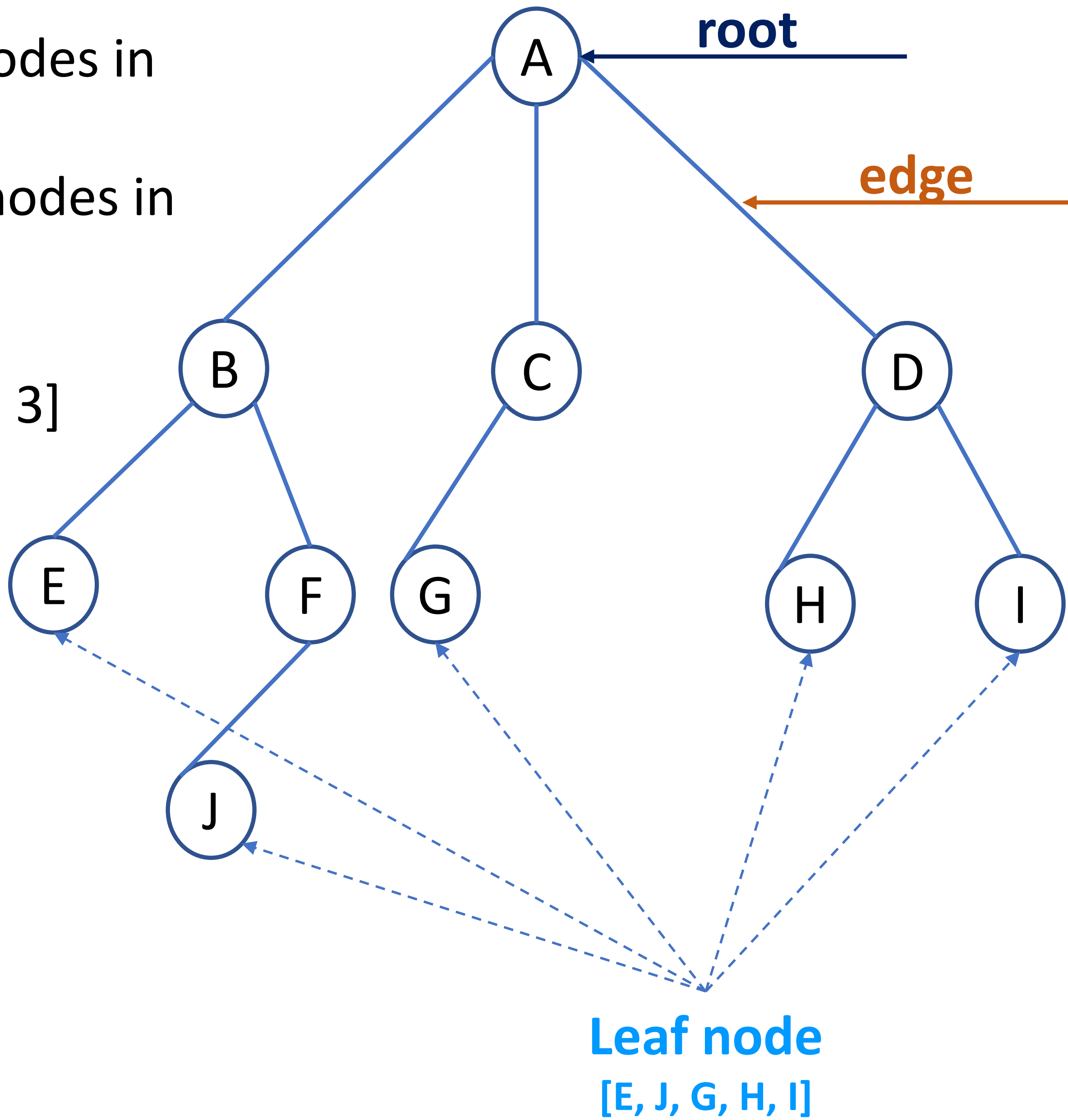
- Root *refers to a n*ode with no parents in the tree. There can be at most one root node in a tree
- Edge refers to the link from the parent → child
- Leaf node: a node with no children
- Siblings: children of the same parent

  [*B, C, and D are siblings of A*]
- Level of the tree: set of all nodes at the given depth. Root node is at level zero.
- Depth of a node: Length of the path from the root to the node. [depth of E is 2, A – B – E]
- Height of a node: Length of the path from that node to the deepest node in the tree [height of B is 2, B – F – J]



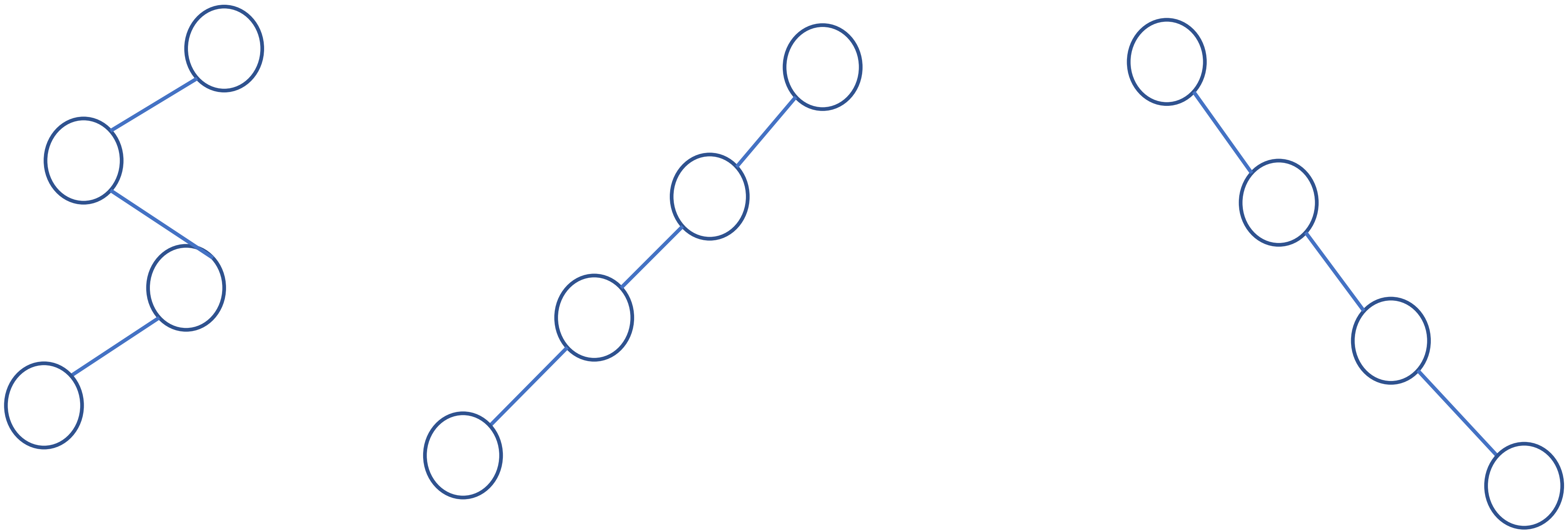
Tree: Terminology (2)

- Depth of a tree: Maximum depth among all the nodes in the given tree
- Height of a tree: Maximum height among all the nodes in tree
- Size of a node: the number of descendants it has including itself [size of subtree B is 3]



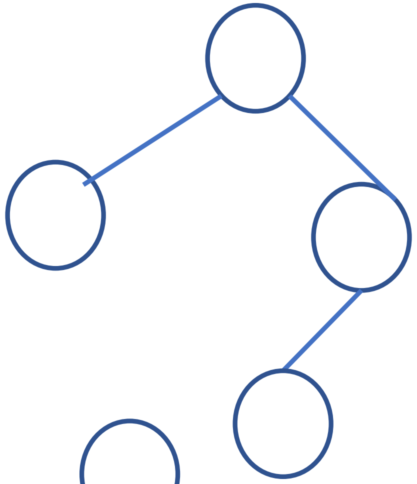
Tree: Skew

- Skew Tree: If every node in a tree has only one child excluding leaf node
- Left Skew Tree: If every node in a tree has only left child
- Right Skew Tree: If every node in a tree has only right child

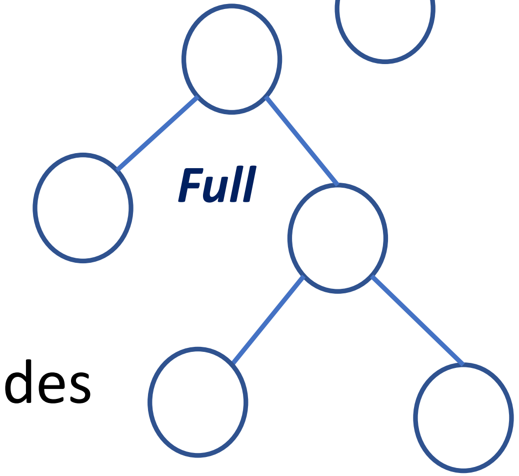


Tree: Binary

- <u>Binary Tree</u>: If every node in a tree has zero, one or two children
  - ✓ <u>Note: </u>An empty tree is also a valid binary tree

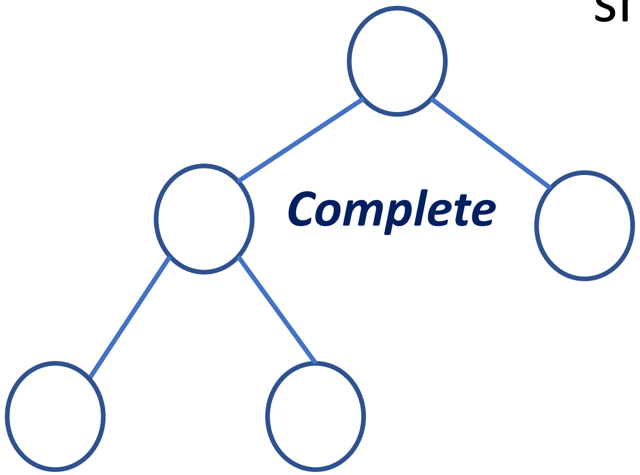


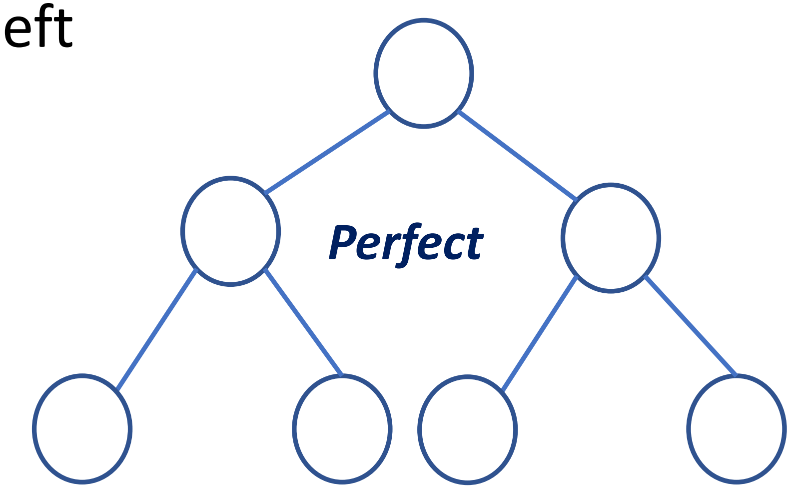
- <u>Full Binary Tree</u>: If each node has exactly two children or no children



- <u>Perfect Binary Tree</u>: If each node has exactly two children and all leaf nodes are at the same level

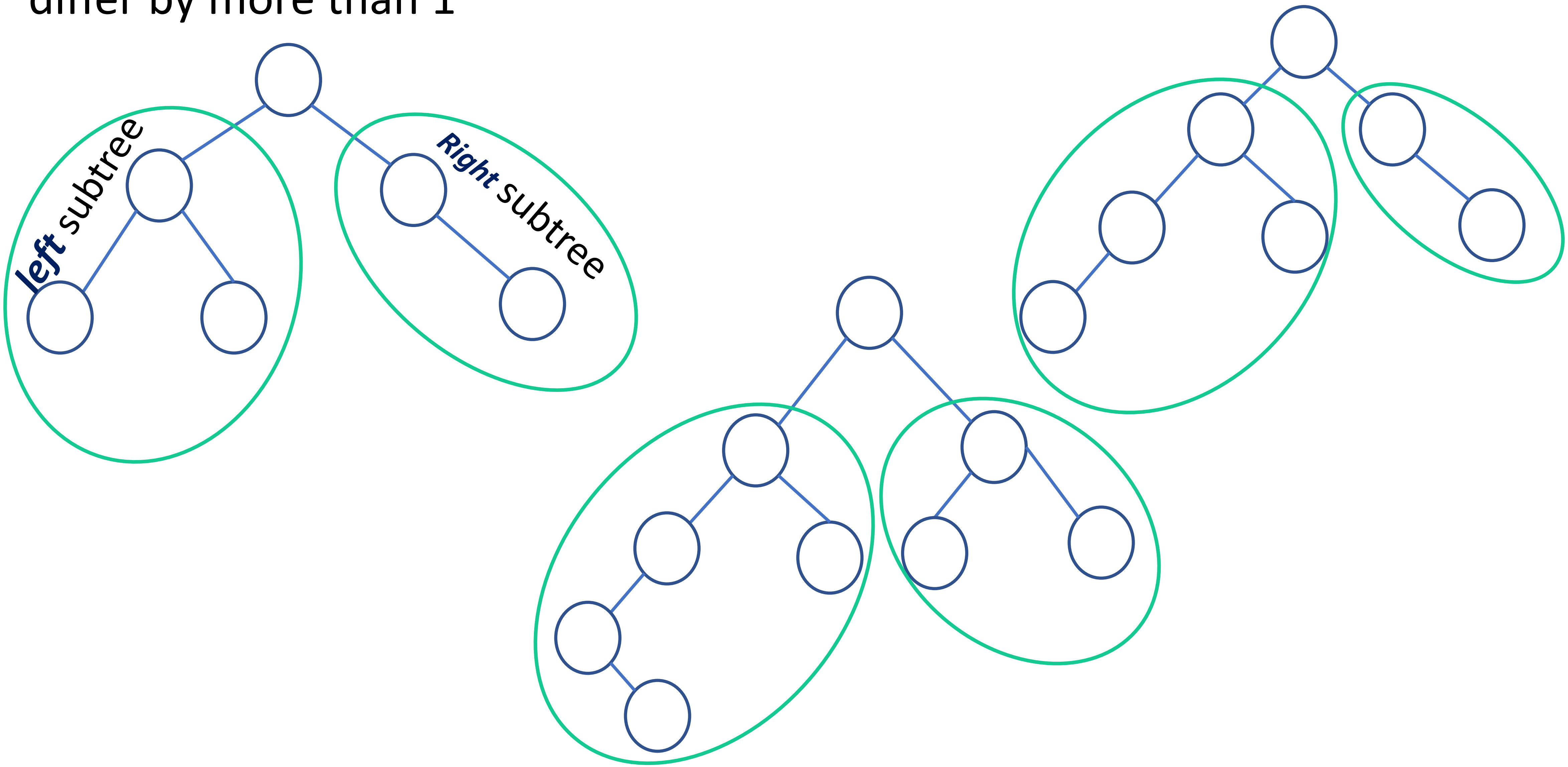
- <u>Complete Binary Tree</u>: If all the leaf nodes are at height *h or h-1* and nodes shall be filled from the left



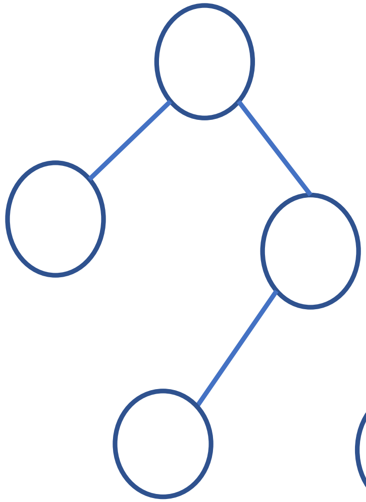


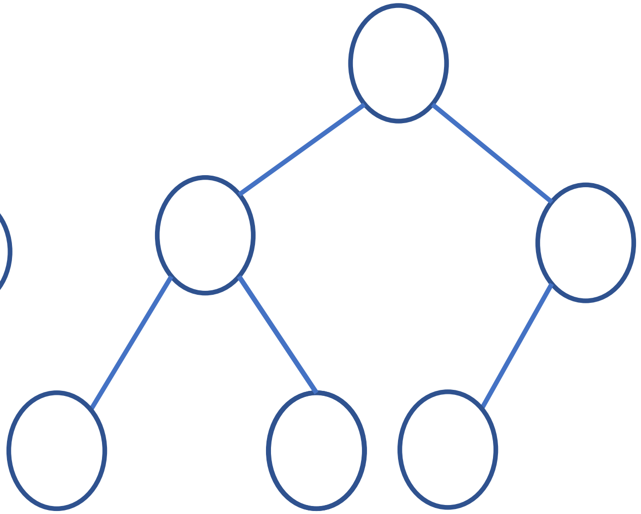
Tree: Binary

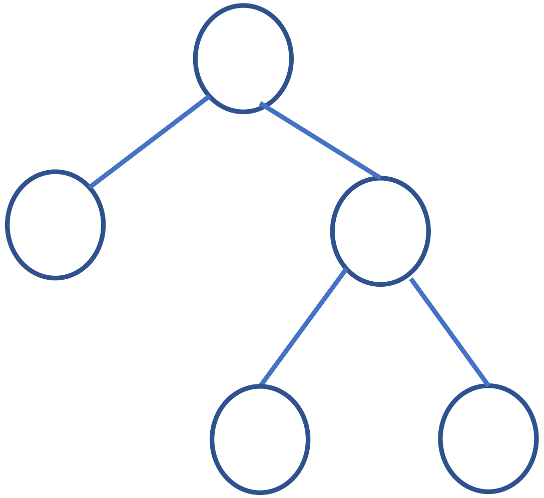
- <u>Balanced Binary Tree</u>: At any node the height of left and right sub tree do not differ by more than 1

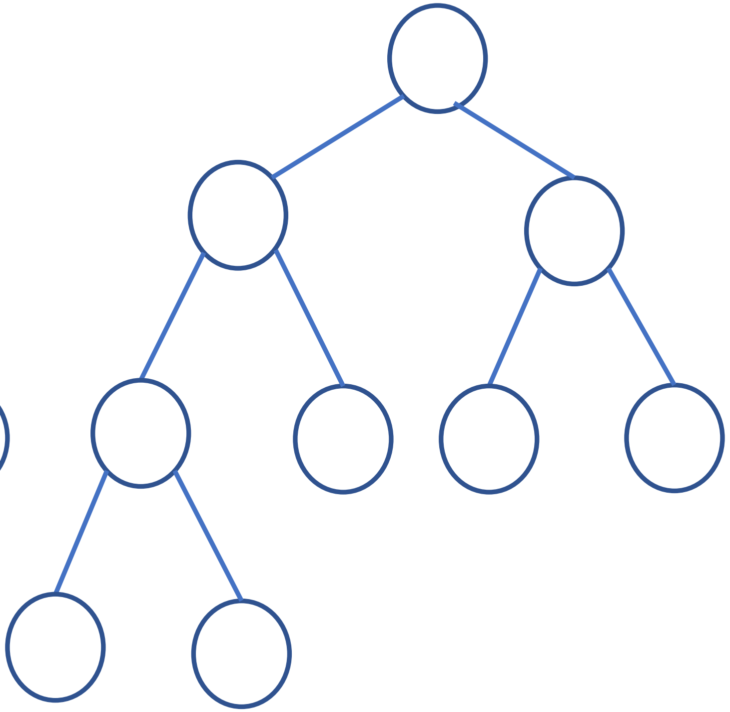


Exercise: Binary Tree





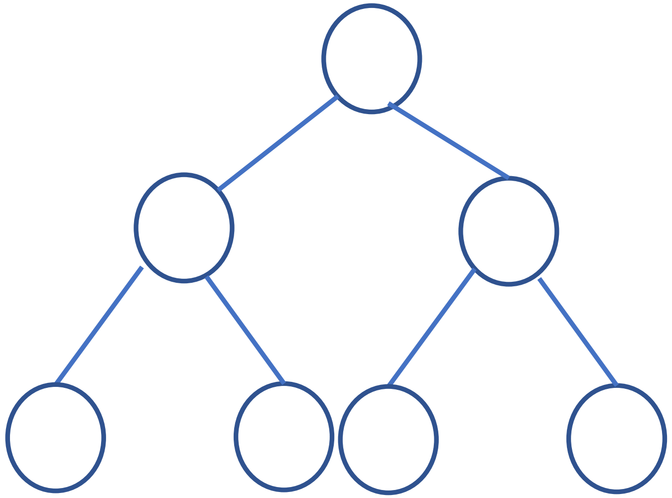




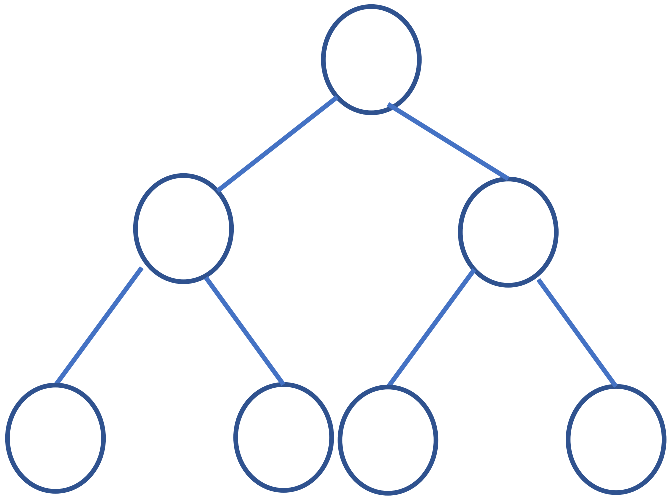
Tree
Complete Binary Tree

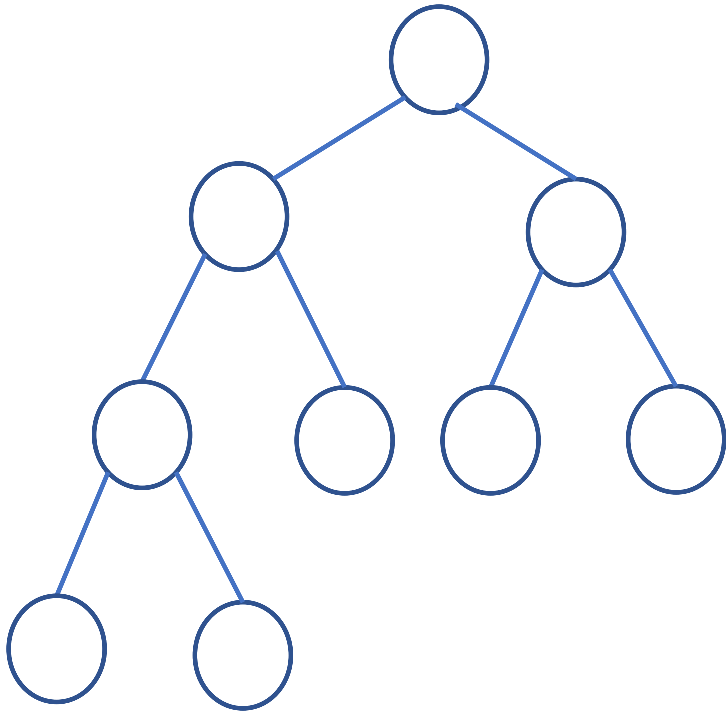
Full Binary Tree

Balanced Binary Tree



Exercise: Binary Tree





Tree

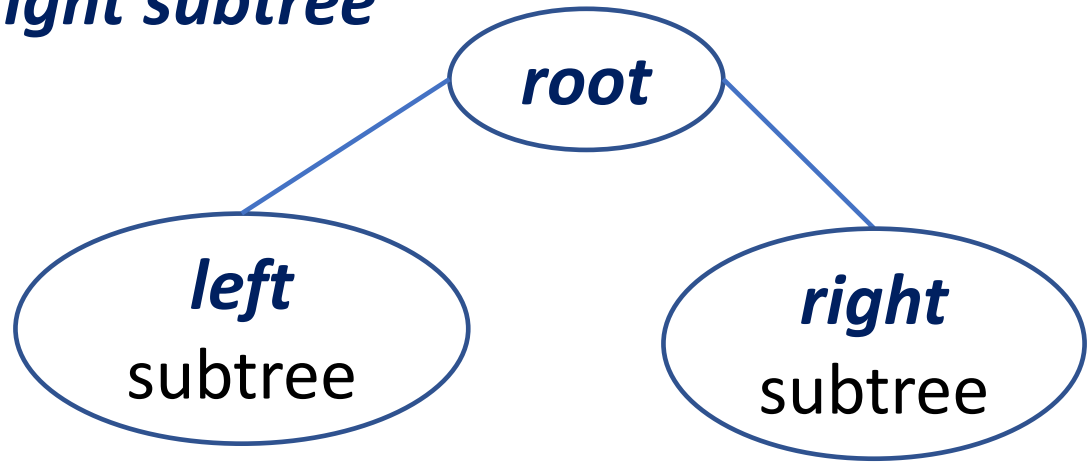
Complete Binary Tree

Full Binary Tree

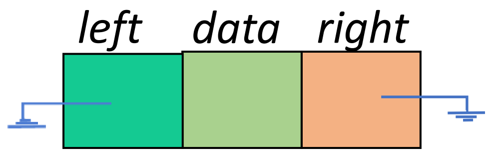
Balanced Binary Tree

Binary Tree: Representation

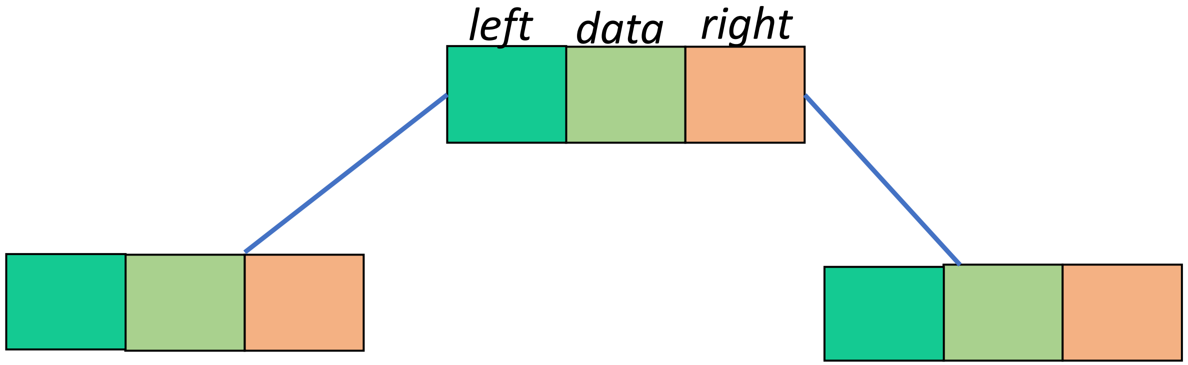
- <u>Binary Tree</u>: A binary tree is defined recursively, it consists of a ***root.*** a ***left sub tree,*** and a ***right subtree***




```
struct btree {
        struct btree *left;
        int data;
        struct btree *right;
}
```

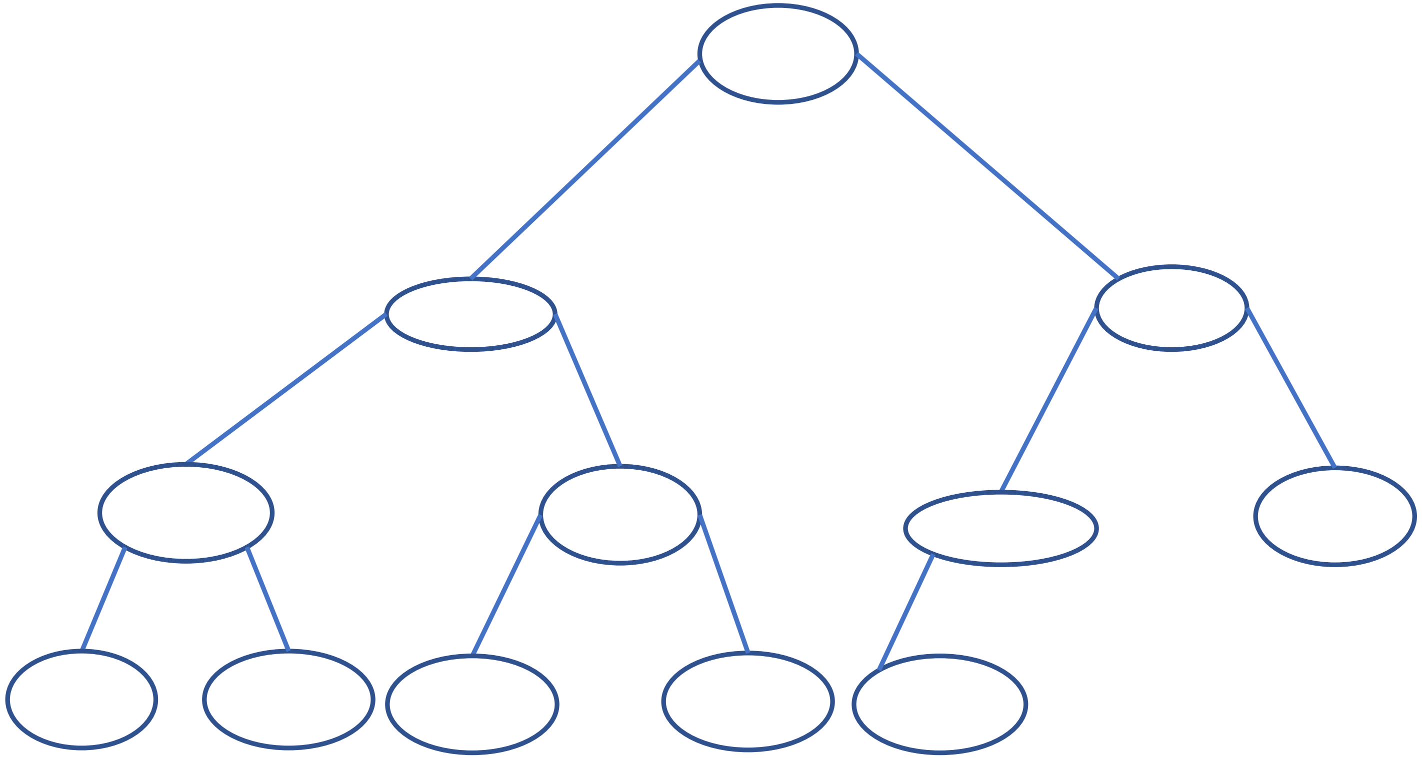


- <u>Traverse</u>: Visit each node in the binary tree exactly only once. Tree traversals are naturally recursive

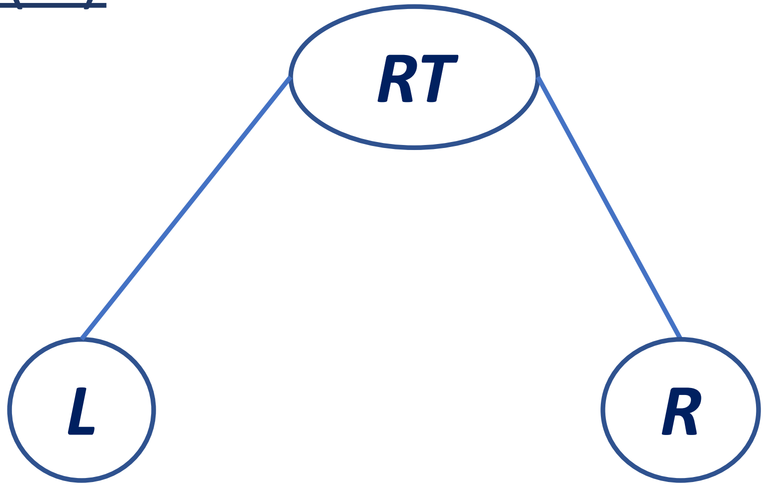


Binary Tree: Construct complete binary tree by using the following input

Input: {2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12}



Binary Tree: Traversal (1)



RT= Root;
L = Left;
R = Right;

Binary tree has three "parts" → Six possible ways to traverse the binary tree

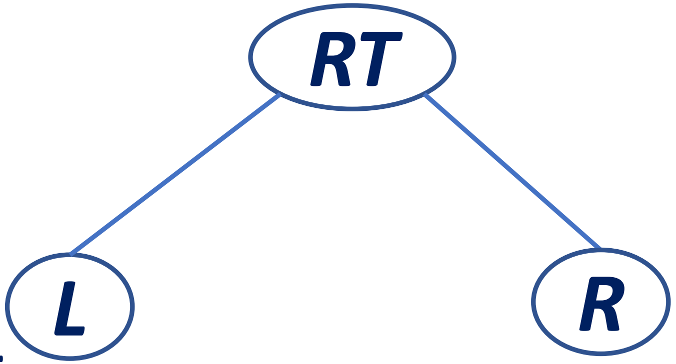
- **Root, Left, Right [RT, L, R]**
- **Left, Right, Root [L, R, RT]**
- **Left, Root, Right [L, RT, R]**
- **Root, Right, Left [RT, R, L]**
- **Right, Root, Left [R, RT, L]**
- **Right, Left, Root [R, L, RT]**

Binary Tree: Traversal (2)

RT= Root;

L = Left;

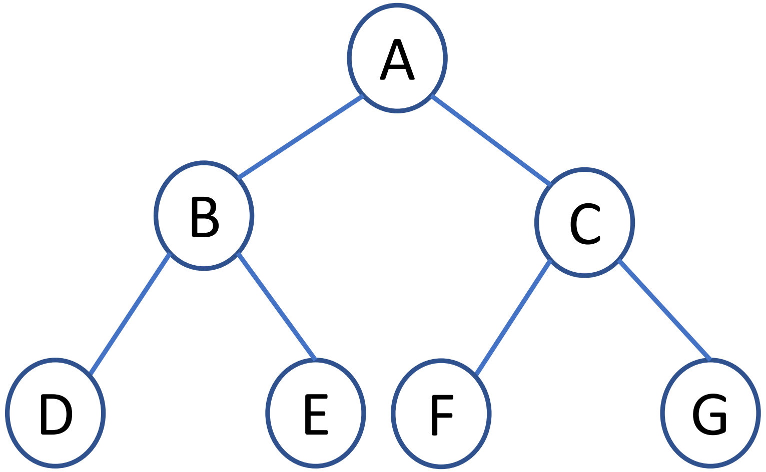
R = Right;



Tree Traversal

Preorder	**Inorder**	**Postorder**
Root, Left, Right	**Left, Root, Right**	**Left, Right, Root**
[RT, L, R]	**[L, RT, R]**	**[L, R, RT]**

Travel Policies →



Preorder → ABDECFG

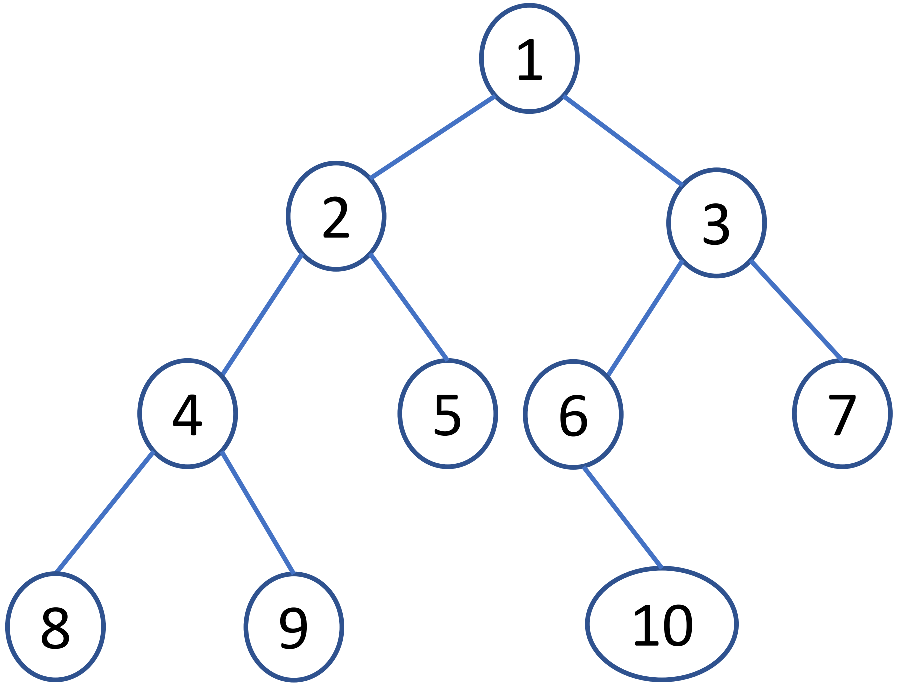
Inorder → DBEAFCG

Postorder →

Binary Tree: Traversal (3)

Travel Policies →

Preorder	**Inorder**	**Postorder**
Root, Left, Right	**Left, Root, Right**	**Left, Right, Root**
[RT, L, R]	**[L, RT, R]**	**[L, R, RT]**



Preorder →

Inorder →

Postorder →

Binary Tree: Traversal (4)

Preorder

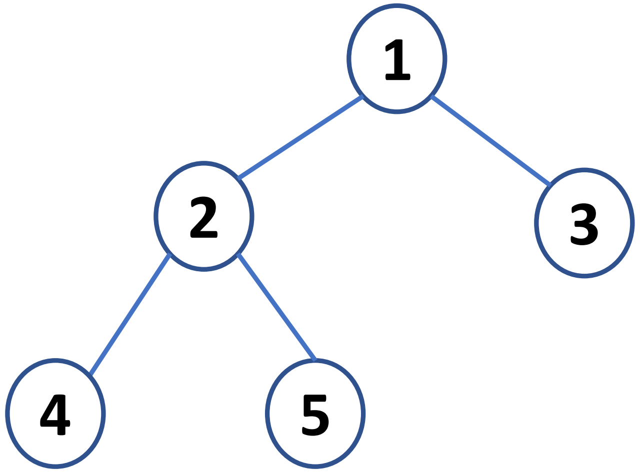
Root, Left, Right
[RT, L, R]

Inorder

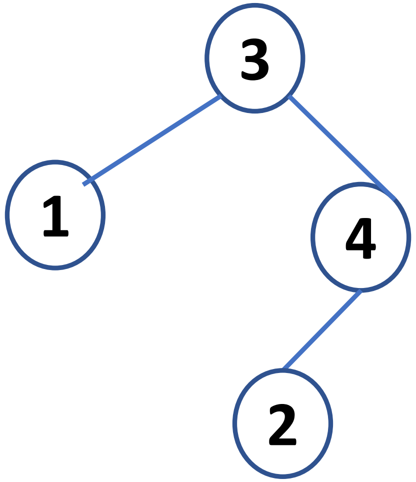
Left, Root, Right
[L, RT, R]

Postorder

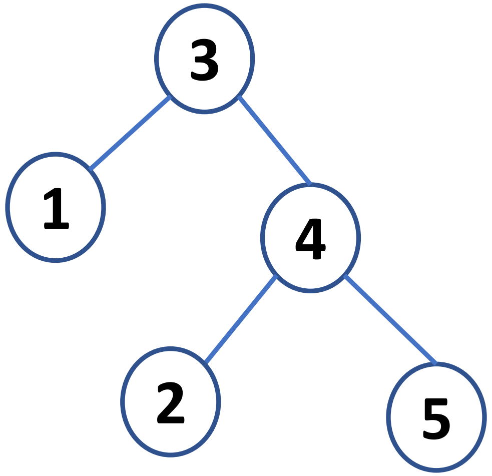
Left, Right, Root
[L, R, RT]



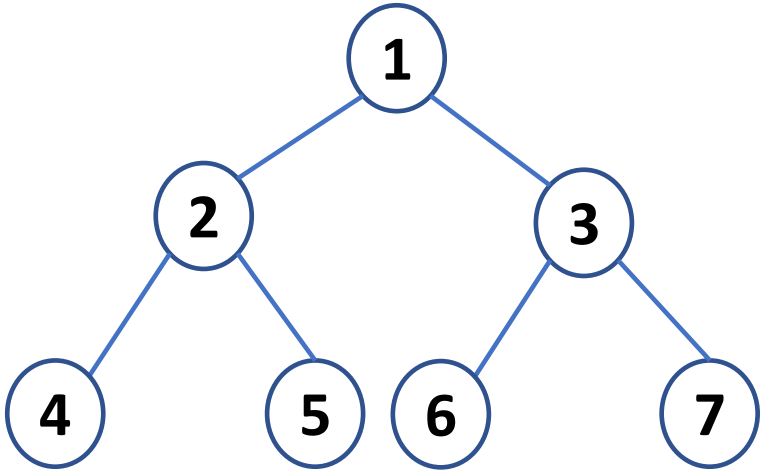
Pre-order :
In-order :
Post-order :



Pre-order:
In-order :
Post-order:



Pre-order :
In-order :
Post-order:



Pre-order :
In-order :
Post-order:

Exercise: Binary Tree Traversal (1)

Travel Policies →

Preorder	Inorder	Postorder
Root, Left, Right	Left, Root, Right	Left, Right, Root
[RT, L, R]	[L, RT, R]	[L, R, RT]

Inorder → DBEAFC

Preorder → ABDECF

What is the post-order traversal sequence of resultant tree?

Postorder →

Exercise: Binary Tree Traversal (2)

Travel Policies →

Preorder	Inorder	Postorder
Root, Left, Right	Left, Root, Right	Left, Right, Root
[RT, L, R]	[L, RT, R]	[L, R, RT]

Inorder → 8,6,9,4,7,2,5,1,3
Postorder → 8,9,6,7,4,5,2,3,1

What is the pre-order traversal sequence of resultant tree?

Preorder →

thank you!

email:
k.kondepu@iitdh.ac.in