Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Two sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

$T(n) = O(n^2)$

$S(n) = O(1)$

```
int[] twoSum(int[] nums, int target) {
    for(int i=0; i<nums.length; i++) {
        for(int j=i+1; j<nums.length; j++) {
            if(nums[i]+nums[j]==target) {
                return new int[]{i,j};
            }
        }
    }
    return new int[]{};
}
```

Two sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

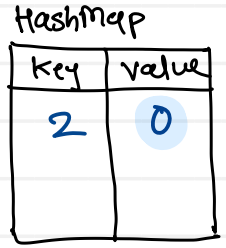
$T(n) = O(n)$

$S(n) = O(n)$

Example 1:

Input: nums = [2,7,11,15], target = 9

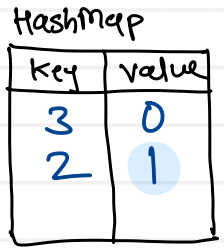
Output: [0,1]



Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]



Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

```
int[] twoSum(int[] nums, int target){
    Map<Integer, Integer> tbl = new HashMap<>();
    for(int i = 0;  i < nums.length; i++){
        if(tbl.containsKey(target - nums[i]))
            return new int[]{tbl.get(target-nums[i]), i};
        tbl.put(nums[i], i);
    }
    return new int[]{};
}
```

Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:
Input: nums = [1,2,3,1]
Output: true

i

Hashset
1, 2, 3

Example 2:
Input: nums = [1,2,3,4]
Output: false

i

Hashset
1, 2, 3, 4

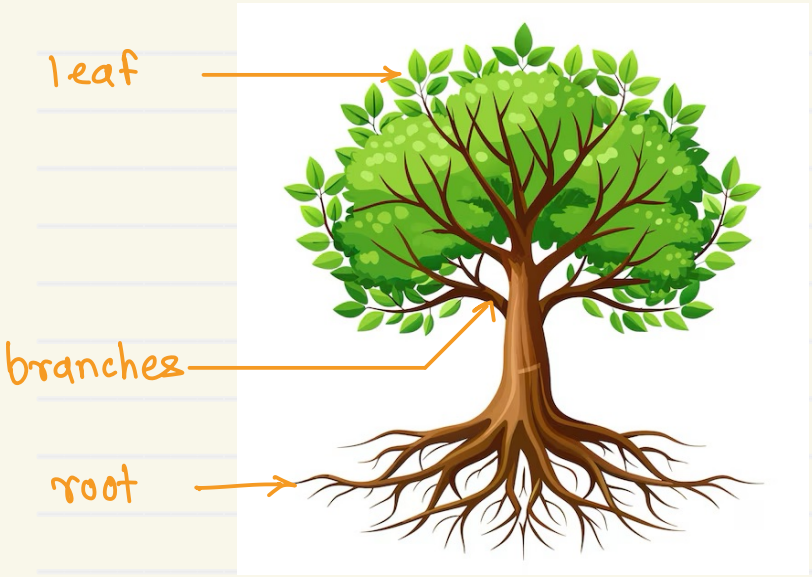
Example 3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

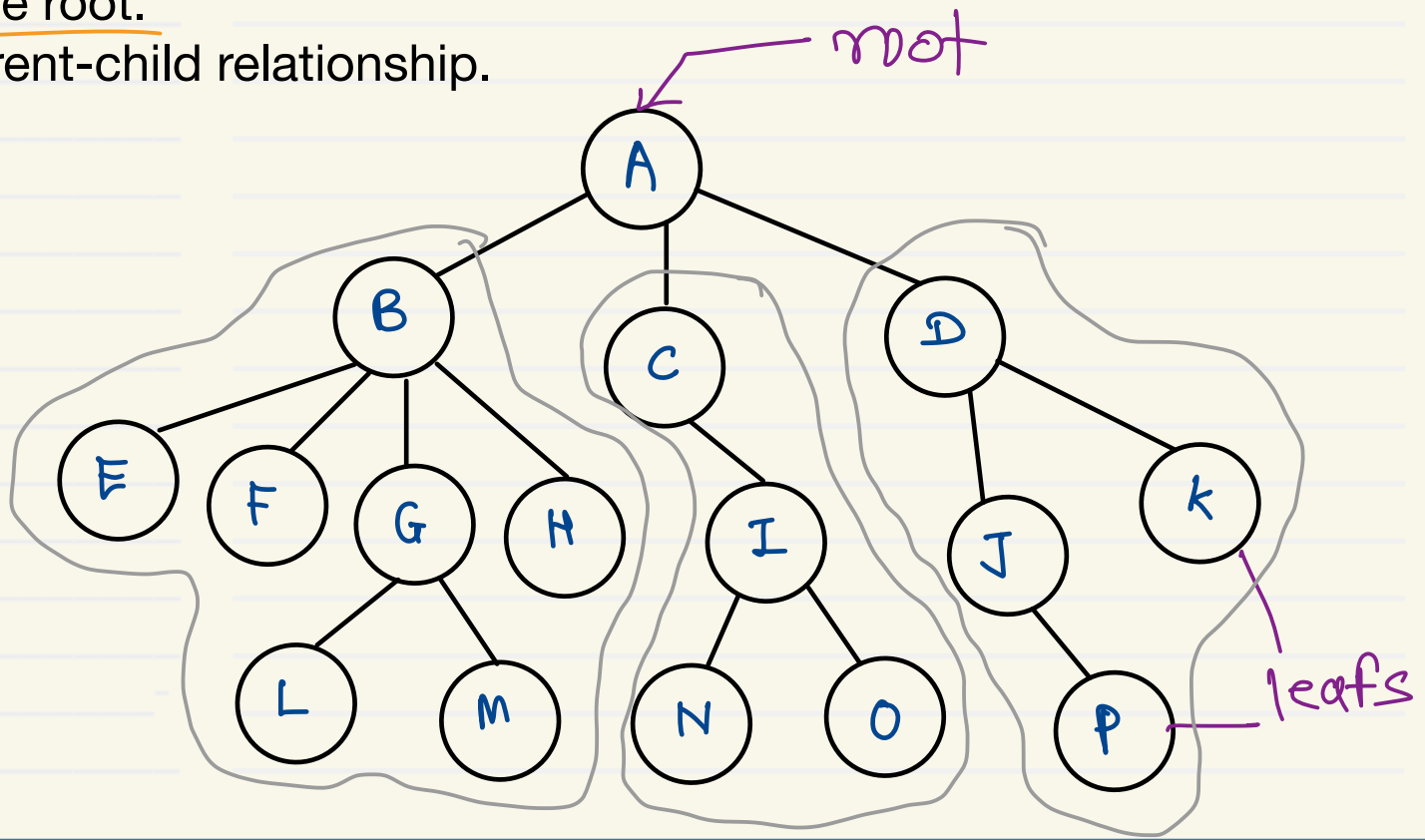
T(n) = O(n)

```
boolean containsDuplicate(int[] nums) {
    Set<Integer> s = new HashSet<>();
    for(int n : nums) {
        if(s.add(n) == false)
            return true;
    }
    return false;
}
```

Tree - Definition

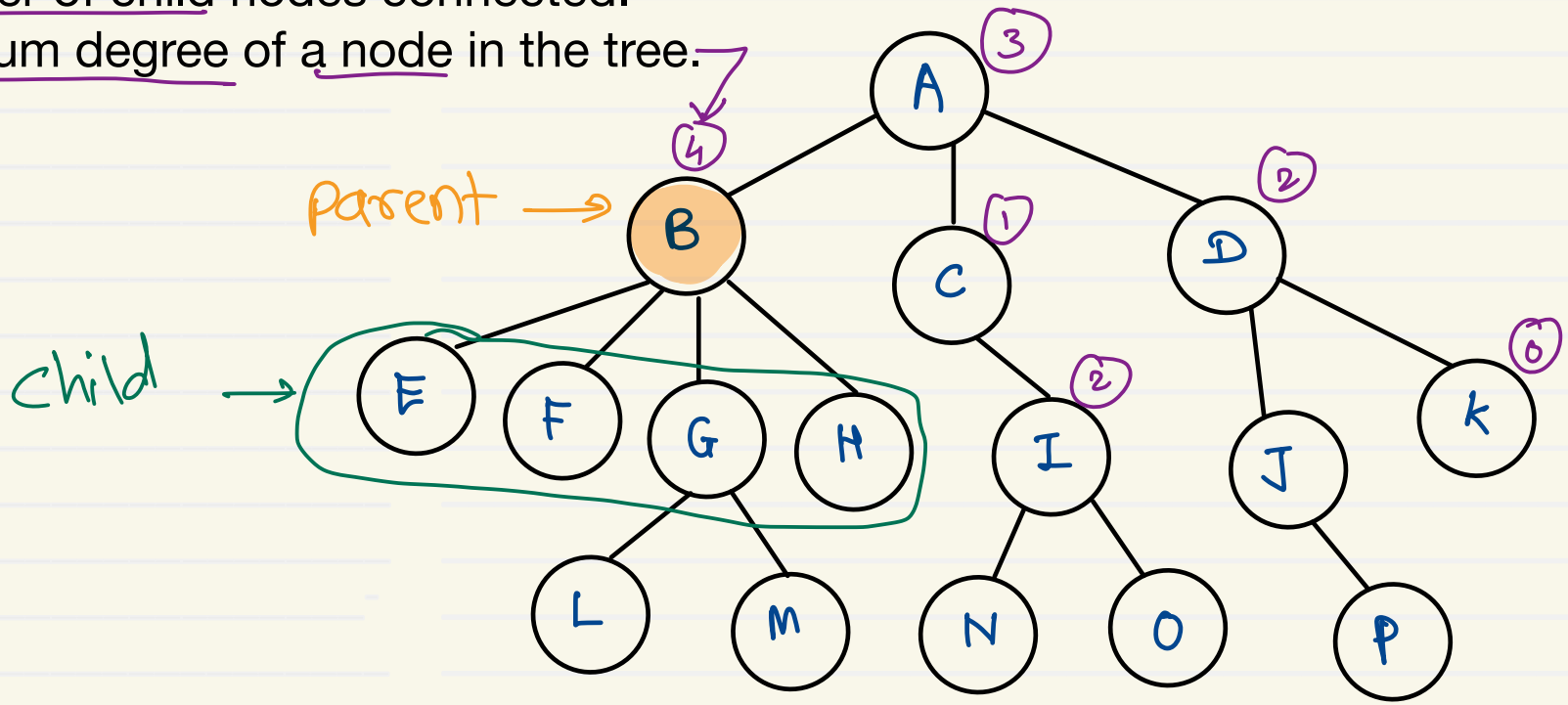
- **Tree** is a non linear data structure.
- **Tree** is a finite set of nodes with one specially designated called as "**root**" and the remaining nodes are partitioned into m disjoint subsets where each subset is a tree.
- Each subset is called as **sub tree** of the root.
- Tree represents hierarchical data ie parent-child relationship.





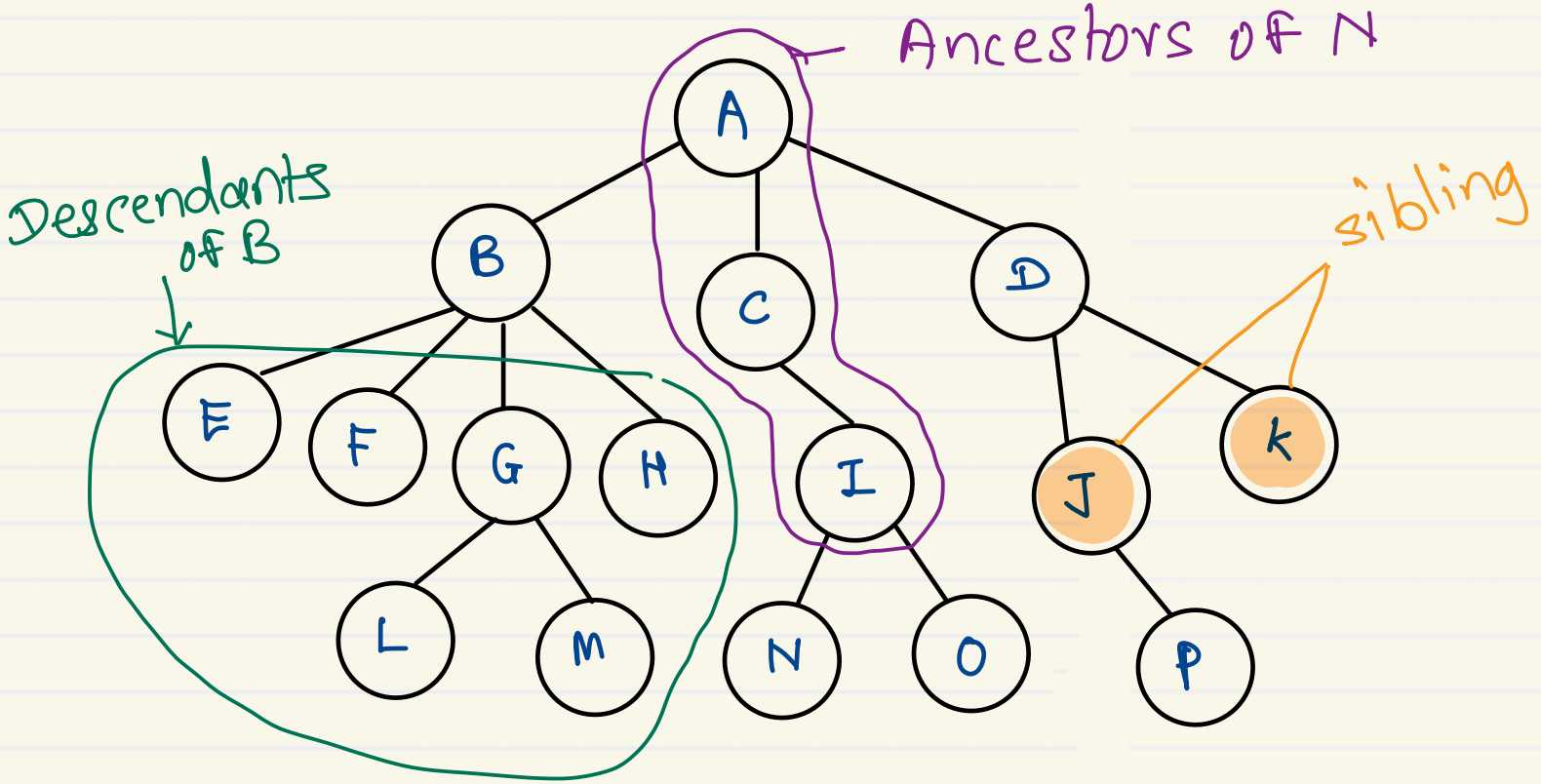
Tree - Terminologies

- **Node -** an item in tree that holds data and pointer to other (child) nodes.
- **Null tree -** empty tree ie tree with no nodes.
- **Parent node -** having other child nodes connected.
- **Child node -** immediate descendant of a node.
- **Leaf node -** terminal node of a tree. Does not have any child node.
- **Degree of a node -** number of child nodes connected.
- **Degree of a tree -** maximum degree of a node in the tree.



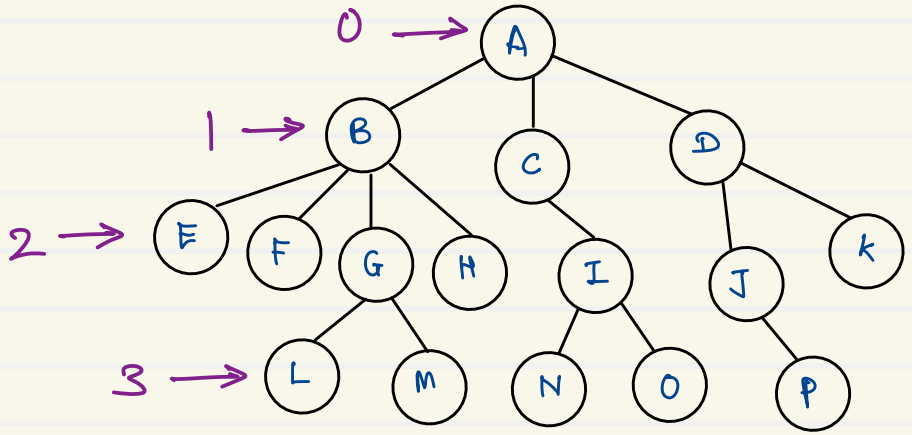
Tree - Terminologies

- **Sibling -** child nodes of the same parents.
- **Descendants -** all child nodes reachable from the node.
- **Ancestors -** all the nodes in the path from the node to the root.



Tree - Terminologies

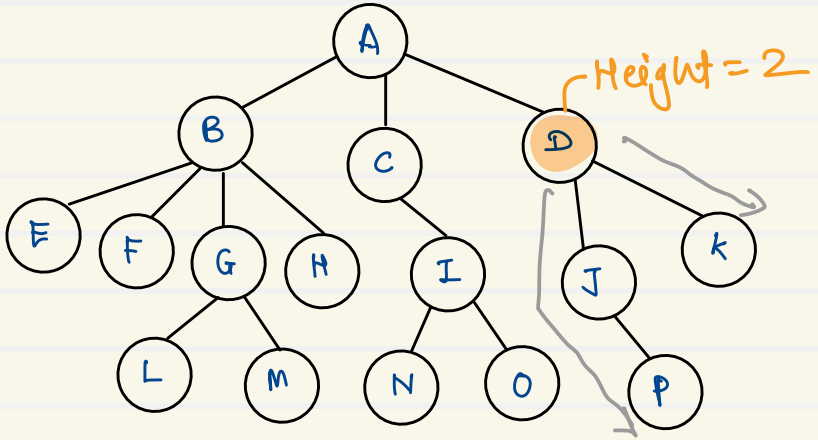
- **Level of a node**
  - Indicates the position of the node in the hierarchy
  - Level of any node is level of it's parent + 1
  - Level of root is 0



- **Depth of a tree** =3
  - Max level of any leaf in the tree

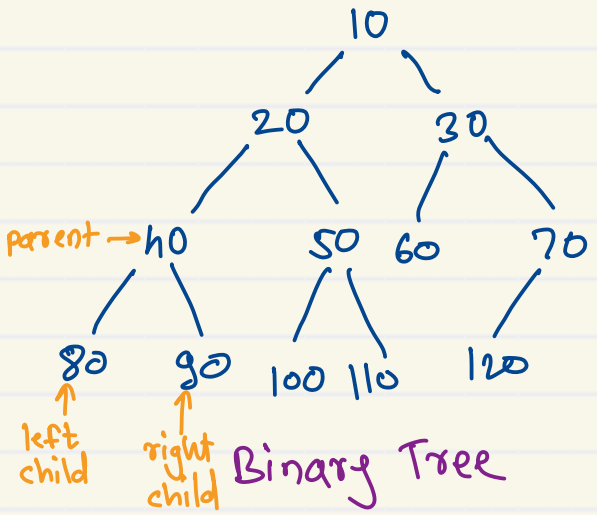
- **Height of a node**
  - Number of nodes from the node to it's deepest leaf.
  - Height = 1 + MAX(left sub tree height, right sub tree height)

- **Height of a tree**
  - Height of root of the tree
  - Height of null/empty tree is -1

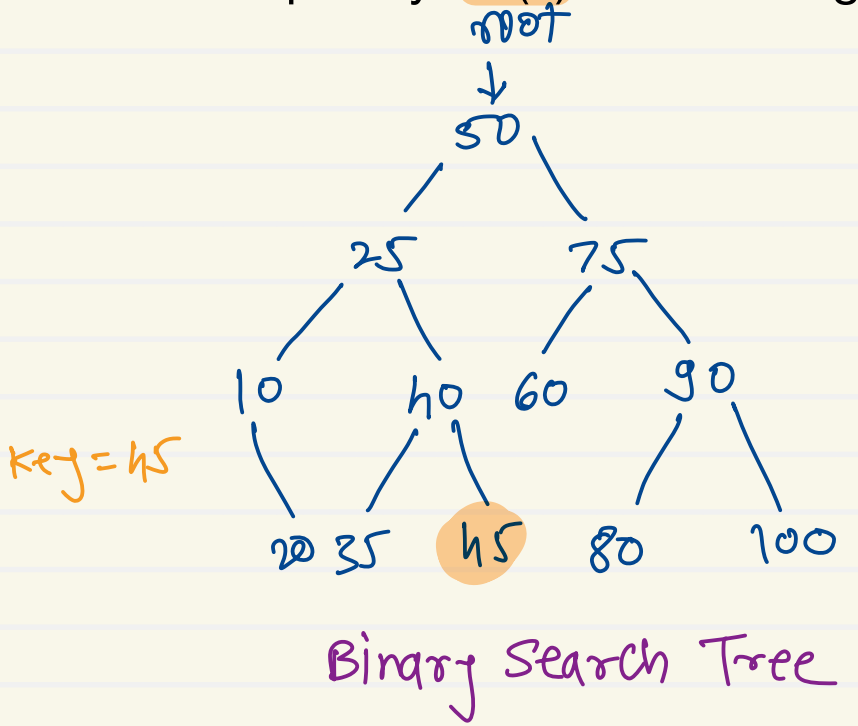


Tree - Types

- **Binary Tree**
  - Tree in which each node has maximum two child nodes.
  - Binary tree has degree 2. Hence it is also called as 2-tree.



- **Binary Search Tree**
  - A binary tree in which left child node is always smaller and right child node is always greater or equal to the parent.
  - Searching is faster.
  - Time complexity : O(h) h - height of tree



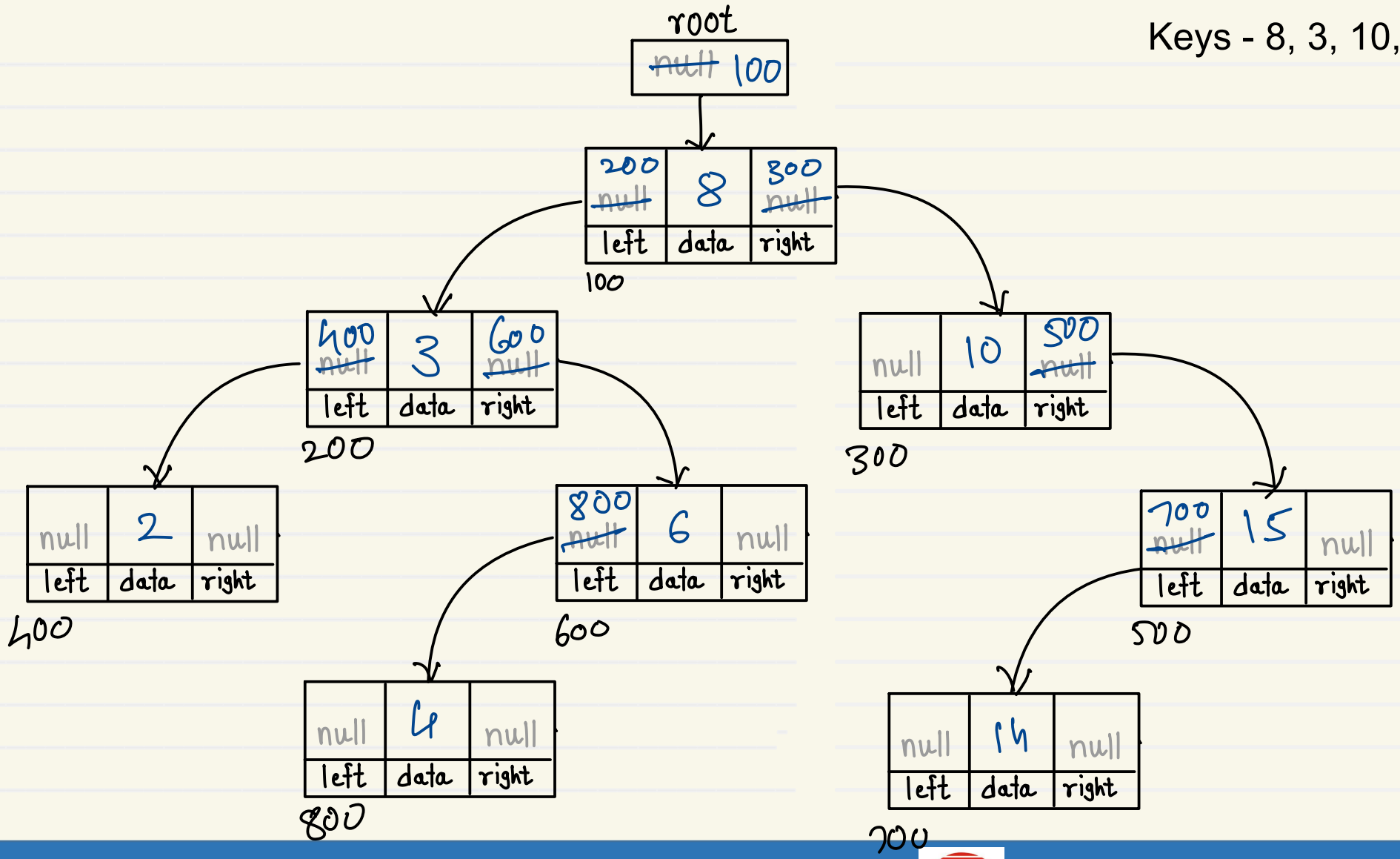
Binary Search Tree - Implementation

Node :
- data – anything
- left – reference /addr of left child
- right – reference /addr of right child

```
class BST {
    static class Node {
        int data;
        Node left;
        Node right;
        public Node() {  }
    }
    Node root;
    public BST() {...}
    public isEmpty() {...}
    public addNode() {...}
    public deleteNode() {...}
    public searchNode() {...}
    public traverse() {...}
}
```

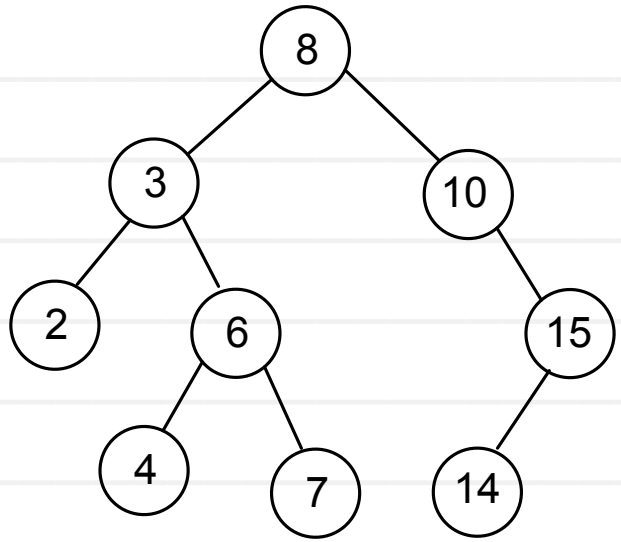
Binary Search Tree - Add node

Keys - 8, 3, 10, 2, 15, 6, 14, 4, 7, 9



Binary Search Tree – Add Node

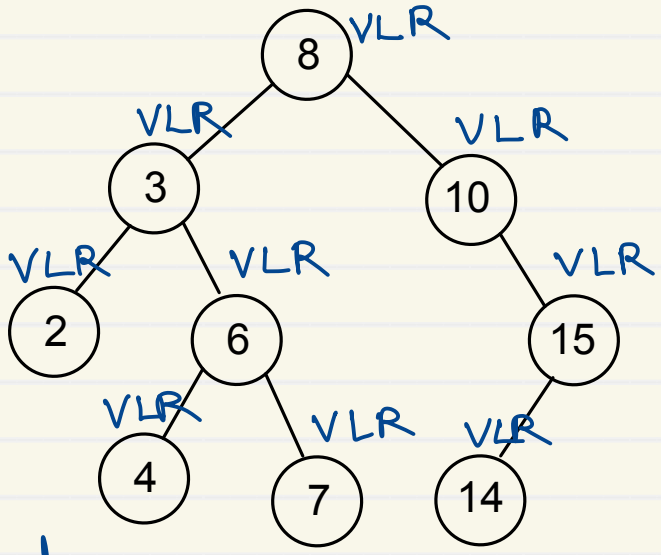
```
//1. create node for given value
//2. if BSTree is empty
      // add newnode into root itself
//3. if BSTree is not empty
      //3.1 create trav reference and start at root node
      //3.2 if value is less than current node data (trav.data)
            //3.2.1 if left of current node is empty
                  // add newnode into left of current node
            //3.2.2 if left of current node is not empty
                  // go into left of current node
      //3.3 if value is greater or equal than current node data (trav.data)
            //3.3.1 if right of current node is empty
                  // add newnode into right of current node
            //3.3.2 if right of current node is not empty
                  // go into right of current node
      //3.4 repeat step 3.2 and 3.3 till node is not getting added into BSTree
```



Tree Traversal Techniques

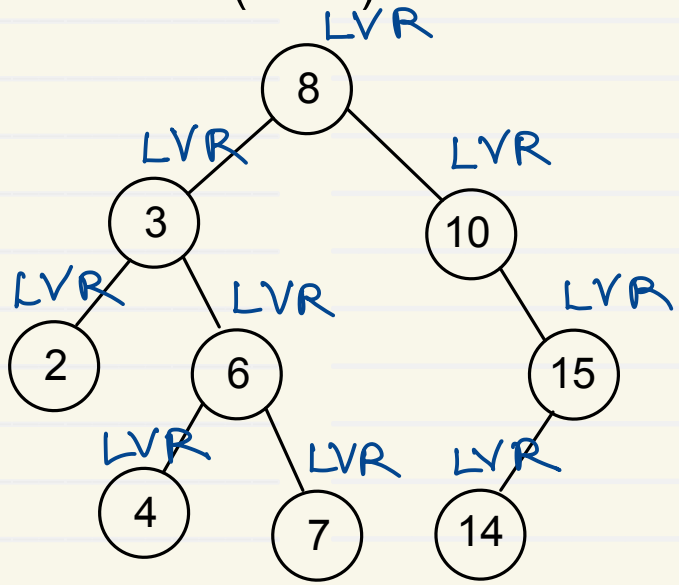
- The traversal algorithms can be implemented easily using recursion.
- Non recursive algorithms for implementing traversal needs stack to store node pointers.

Preorder
(VLR)



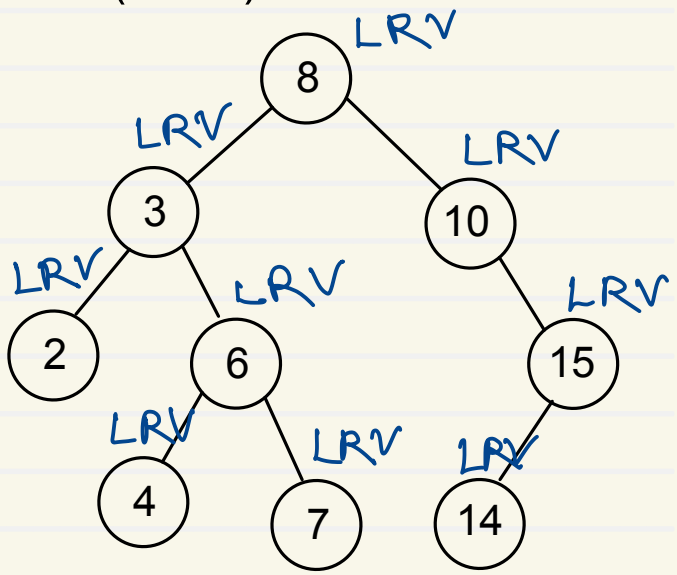
Preorder:
8, 3, 2, 6, 4, 7, 10, 15, 14

Inorder
(LVR)



Inorder:
2, 3, 4, 6, 7, 8, 10, 14, 15
↳ sorted in case of BST

Postorder
(LRV)



Post Order:
2, 4, 7, 6, 3, 14, 15, 10, 8