4 - Binary Trees + BST

Joseph Afework
CS 241

Dept. of Computer Science
California Polytechnic State University, Pomona, CA

Agenda

- Binary Tree
- Binary Tree Properties
- Terms
- Tree Representations
- Tree Traversals
- Binary Search Trees (BST)
- BST Operations

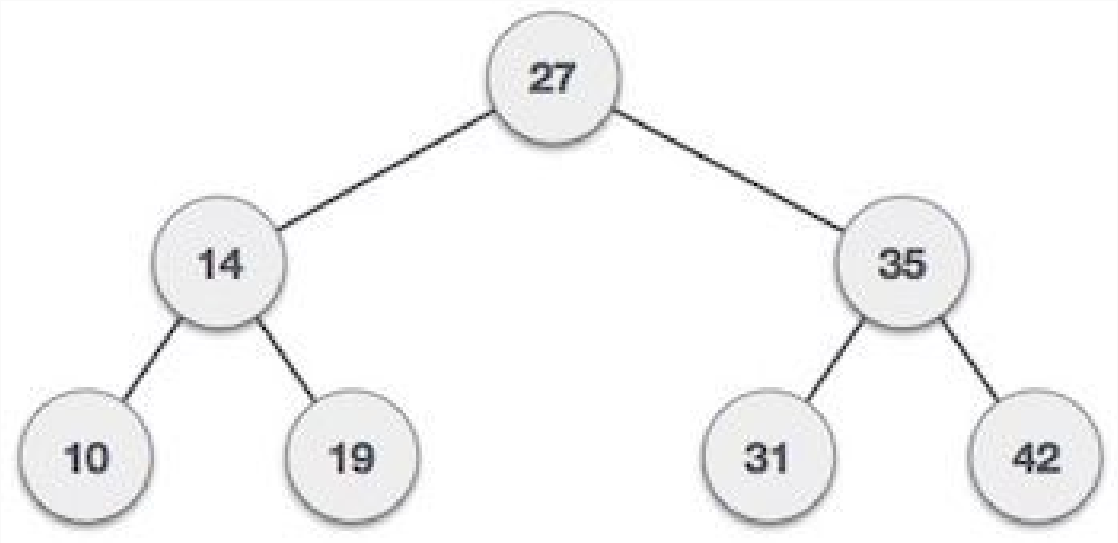
Reading Assignment

- Read Chapter 23 - Trees Contd.
  - Chapter 23 All Sections
  - Read Chapter Summary Section (end of chapter)

- Read Chapter 24 - Binary Trees
    - Chapter 24 All Sections
    - Read Chapter Summary Section (end of chapter)

Binary Trees

Binary Tree



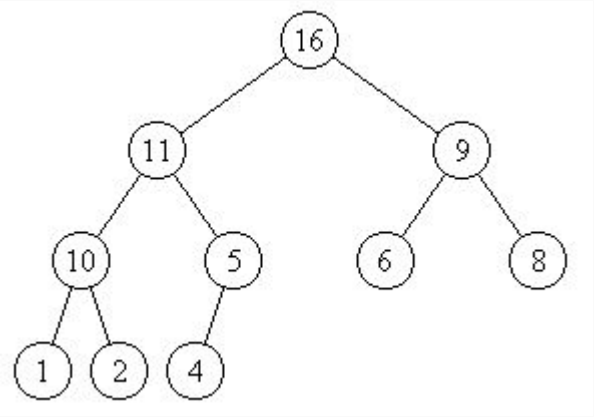
Properties

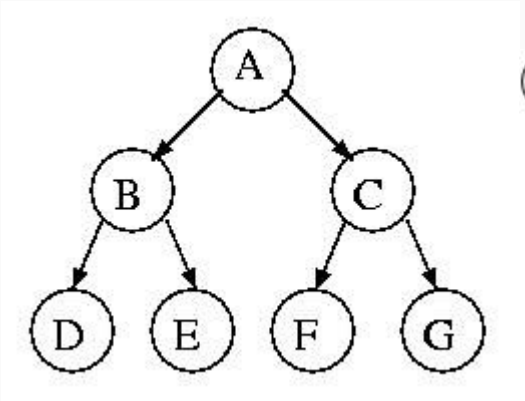
- **Binary Tree** nodes can have no more than two children nodes.
  - Left Node
  - Right Node
  - Maximum number of 2 branches

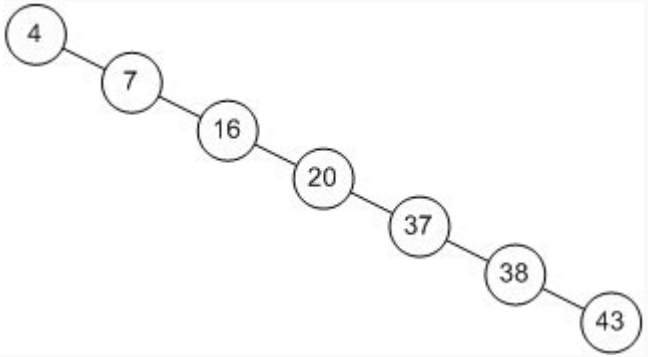
- Each node has exactly one parent node (excluding the root node)

ICE 4.1 Binary Trees

Which of the following are valid binary trees?

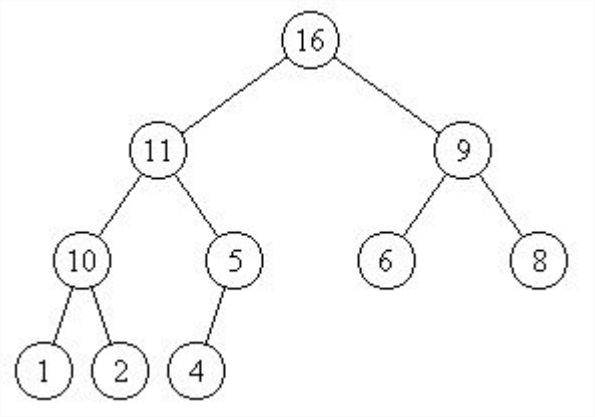






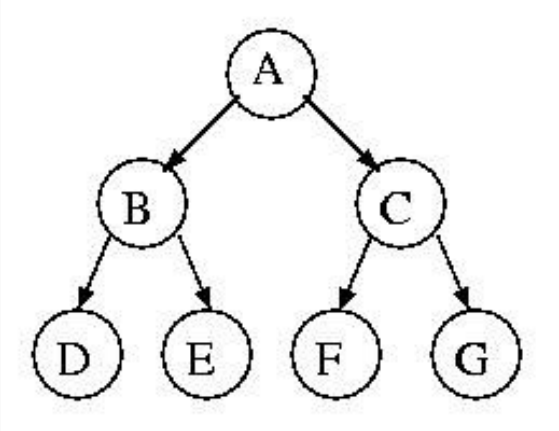
Terms

- **Complete binary tree**: every level except for the deepest level must contain as many nodes as possible; and at the deepest level, all the nodes are as far left as possible.



Terms Contd.

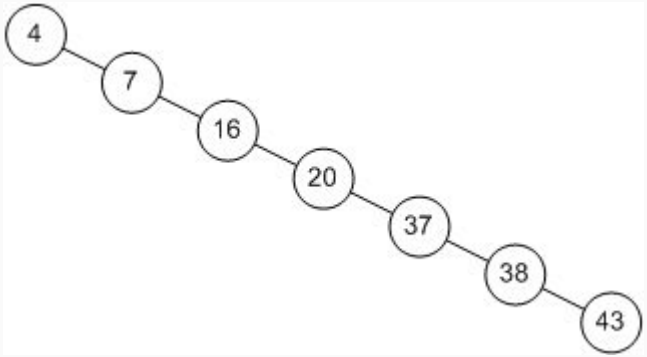
- **Full binary tree**: every leaf has the same depth and every non leaf has two children.
- Maximum number of nodes in a binary tree = $\mathbf{2^{k+1}-1}$, where k = height of the tree



Terms Contd

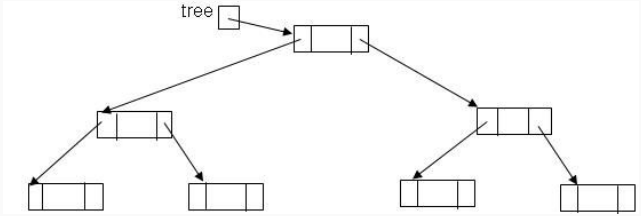
Worst Case for a Tree:

- resembles a linked lists…. But still a valid binary tree

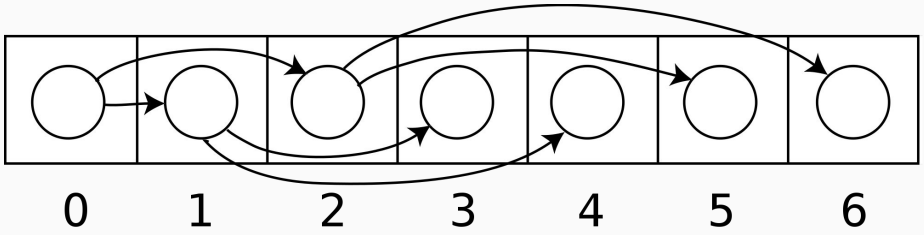


Tree Representations

1. **Node Based Representation**



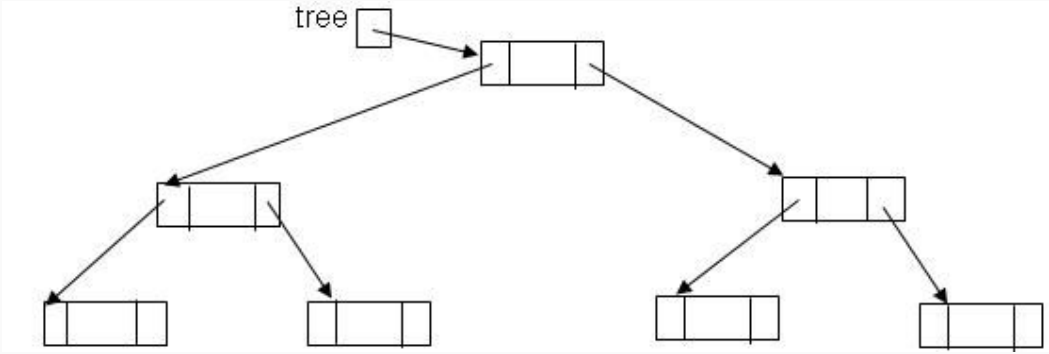
2. **Array Based Representation**



Node Representation

1. Each node of a binary tree can be stored as an object of a binary tree node class.

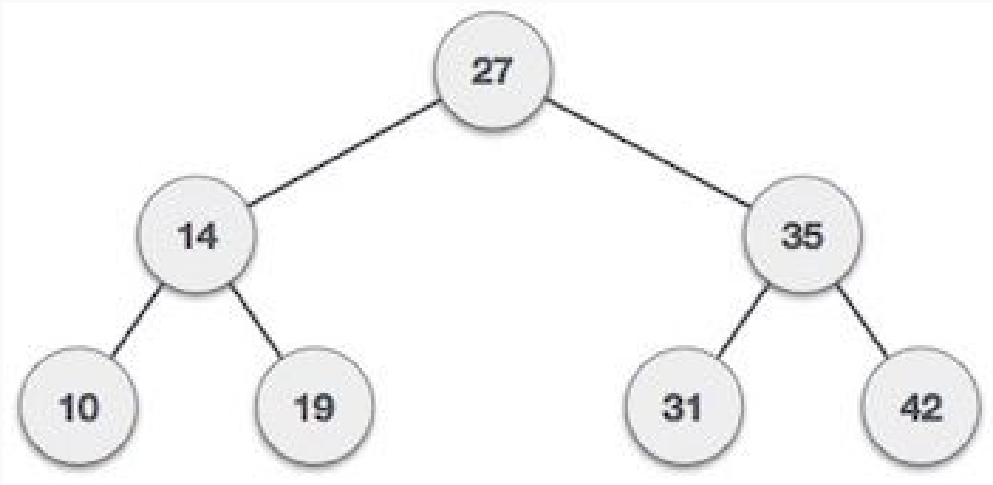
2. An entire tree is represented as a reference to the root node.



Node Representation Contd.

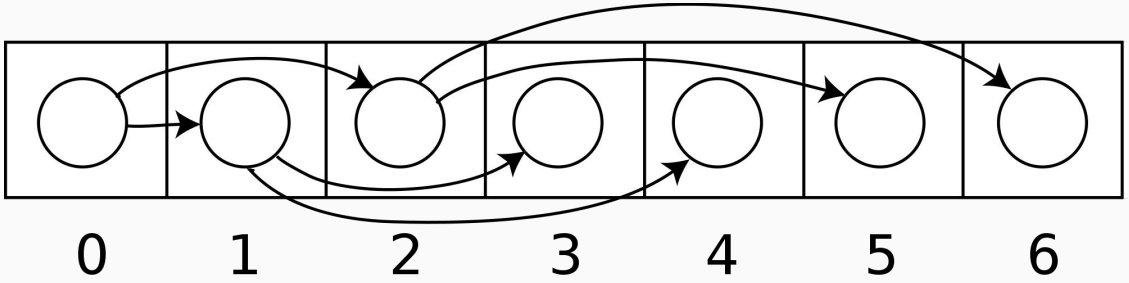
```
public class Node
{
      Node left;
      Node right;
      String data;
}
      …

      Node root = new Node();
      root.data = 27;
      root.left = new Node();
      root.left.data = 14;
      root.right = new Node();
      root.right.data = 35;
```



Array Representation

1. Root always appears in the **[0]** index of the array.
2. For a node located at **[i]**, the parent can be found at location **[(*i* - 1)/2]**.
3. For a node located at **[i]**, the children (if they exist) will always be found at:
   - **Left child [2*i* + 1]**
   - **Right child [2*i* + 2]**



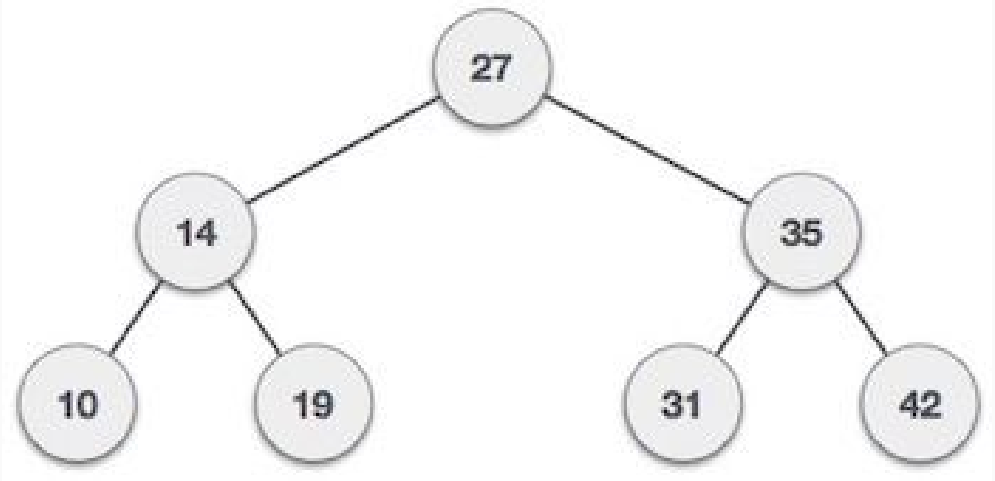
Array Representation Contd.

Note:

- You need to keep track of the size of the array, and resize the array as needed (too many insertions).
- The actual links between the nodes are not stored, but are determined by the formula.
- You need to be sure not to run into array out of bounds issues when traversing the tree

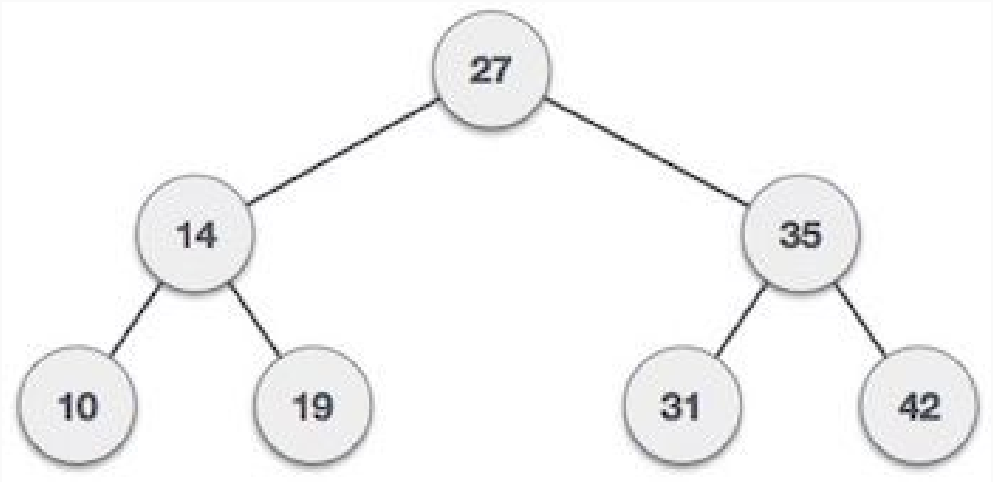
ICE 4.2 Binary Trees w/ Arrays

What is the array representation of the following binary tree?

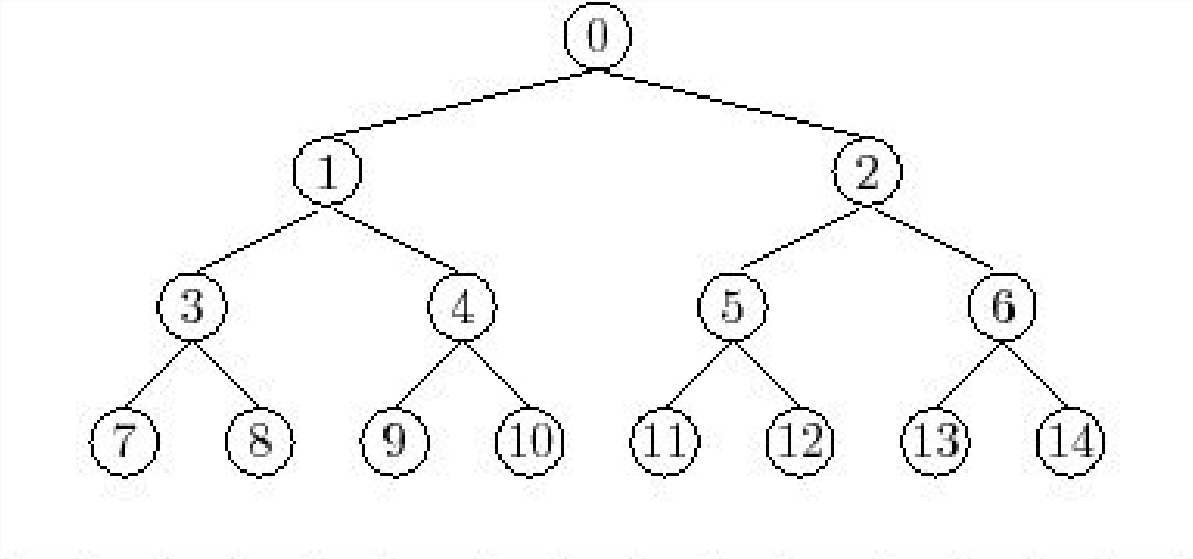


ICE 4.2 Solution

[27, 14, 35, 10 , 19, 31, 42]



Pattern



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Tree Traversals

- **Traversal**: the process of visiting all the nodes in a tree in a certain order.

- All nodes in a tree are connected… The root is always the starting point for any common traversal.
- Tree Traversal algorithms generally utilize lots of recursion

Types of Traversals

- **Three well known types of traversals:**
  1. Pre-order traversal
  2. In-order traversal
  3. Post-order traversal

- **Bonus**
  4. Level-order traversal

Pre-order Traversal

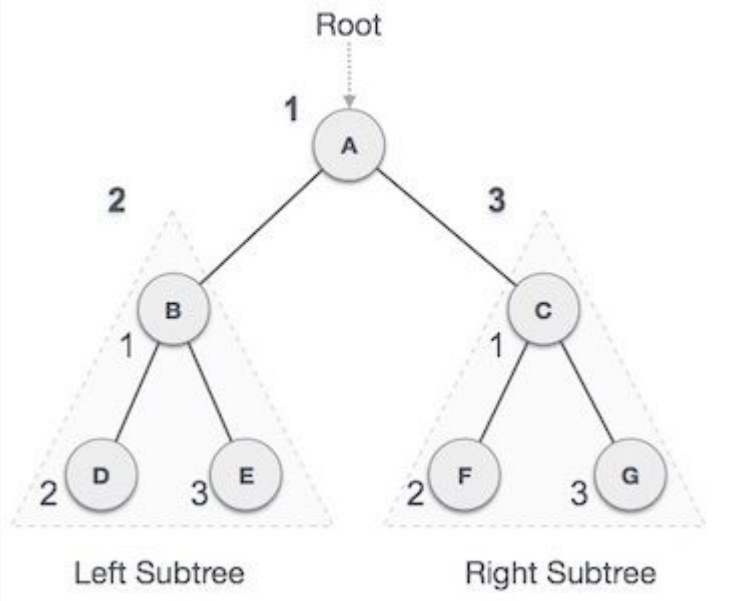
Pseudocode

1. Process the node.
2. Process the nodes in the left subtree with a recursive call.
3. Process the nodes in the right subtree with a recursive call.

Pre-order Example

A → B → D → E → C → F → G

1. Print
2. Left
3. Right



In-order Traversal

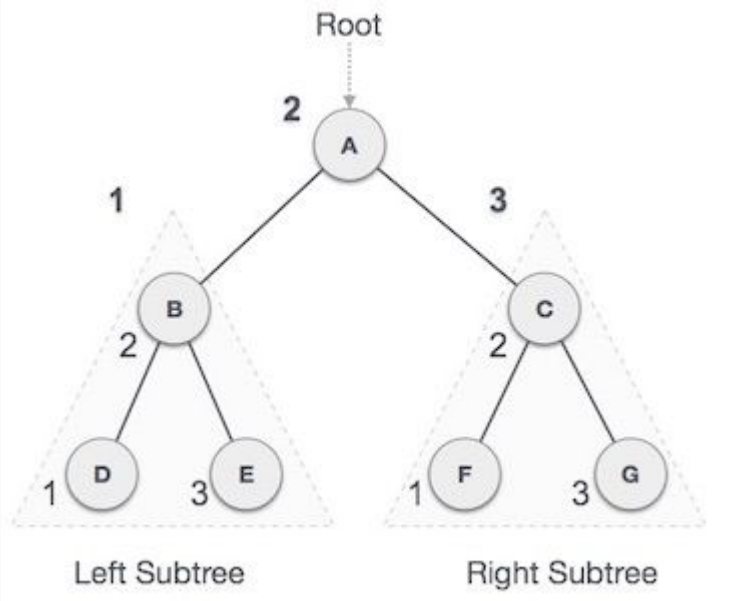
Pseudocode

1. Process the nodes in the left subtree with a recursive call.
2. Process the node.
3. Process the nodes in the right subtree with a recursive call.

In-order Example

D → B → E → A → F → C → G

1. Left
2. Print
3. Right



Post-order Traversal

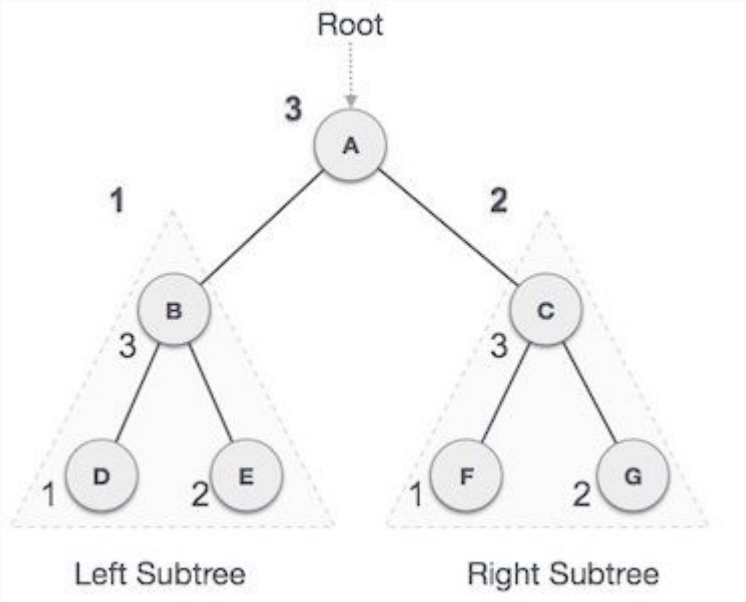
Pseudocode

1. Process the nodes in the left subtree with a recursive call.
2. Process the nodes in the right subtree with a recursive call.
3. Process the node.

Post-order Example

D → E → B → F → G → C → A

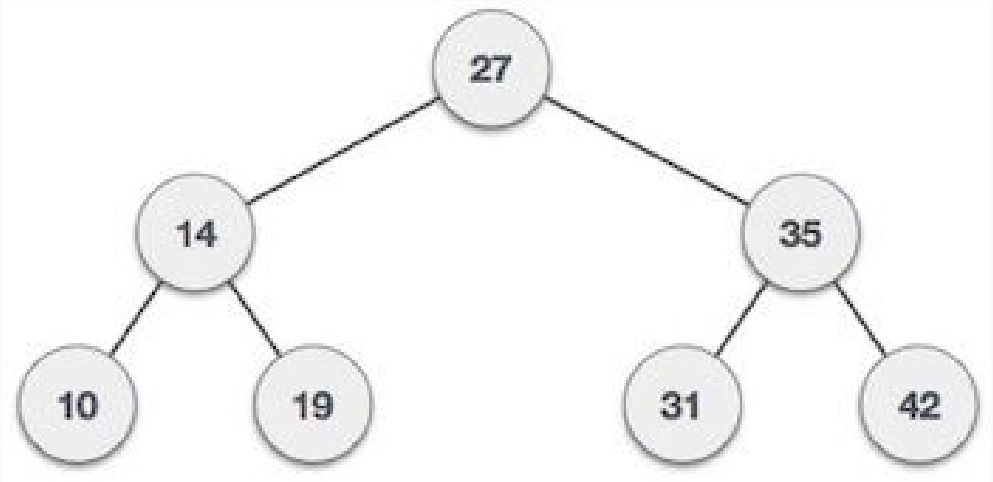
1. Left
2. Right
3. Print



Level-order Traversal

- Print all the nodes level by level

- **Ex.**
  - **27, 14, 35, 10 , 19, 31, 42**



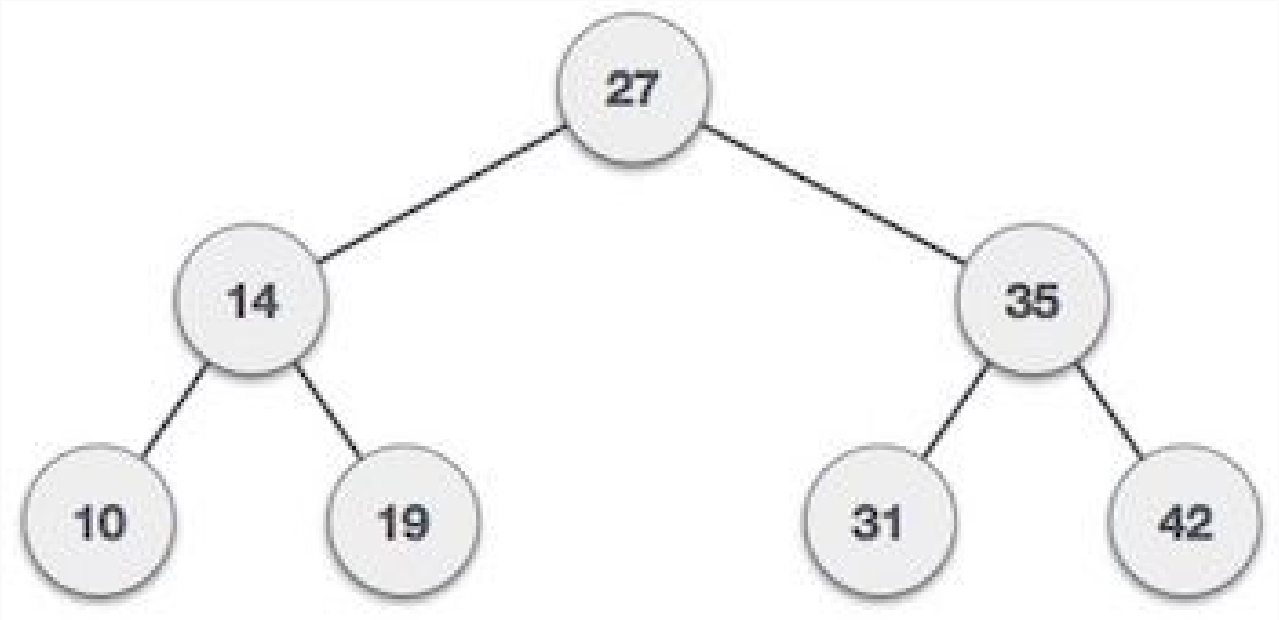
Binary Search Trees (BST)

Binary Search Tree

A Binary Search Tree (BST) is a type of binary tree that follows the following rules:

1. Every node of the binary tree holds a key value.
2. All the key values in the left subtree (left children) of a node are less than or equal to the key value of that node.
3. All the key values in the right subtree (left children) of a node are greater than the key value of that node.

BST Contd.



BST Contd.

Note:

- The rules for the BST must be preserved. If the rules are violated, nodes must be rearranged in the tree to restore the properties.

Operations

- **Insert** - Insert a value into the BST
- **Search** - Search for a value in the BST
- **Delete** - Remove a value from the BST

- **Traversal** - traverse a BST (in-order, post-order, pre-order etc..)

Insert

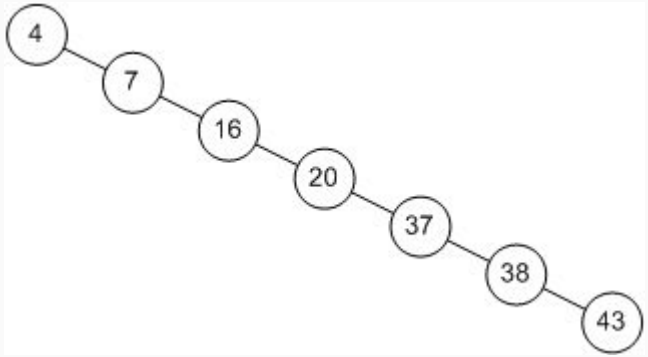
Steps

1. Find the location for insertion (Begin Search at the root)
2. Compare the key at each node:
    a. If the insertion key is less than the key at the current node, move left.
    b. If the insertion key is greater than the key at the current node move right.
3. Stop when an empty node is found (empty leaf slot). Insert the node here.

Insert Contd.

Worst Case: O(n)

Remember this structure….



Search

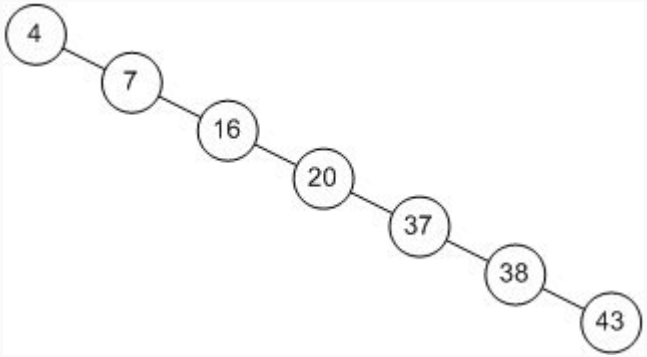
Steps

1. Begin Search at the root
2. Compare the key at each node:
    a. If the insertion key is less than the key at the current node, move left.
    b. If the insertion key is greater than the key at the current node move right.
3. Stop when an empty node is found (empty leaf slot), or when the target key is found.

Search Contd.

Worst Case: O(n)

Remember this structure again….



Delete

A little more complicated:

1. First Locate the node to delete in the tree
    a. Follow steps for search to locate the node
2. Work through the rules for deleting the node

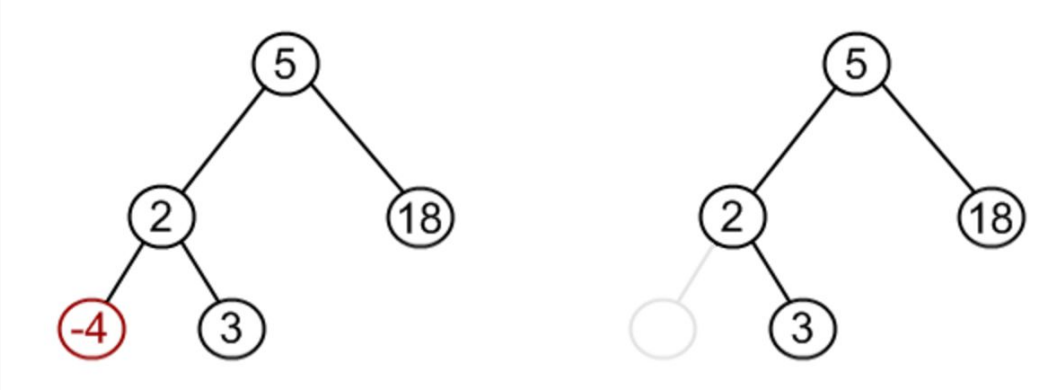
Delete Contd.

Once the node has been found for removal, there are **three possible situations:**

1. Node to be removed has no children.
2. Node to be removed has one child.
3. Node to be removed has two children.

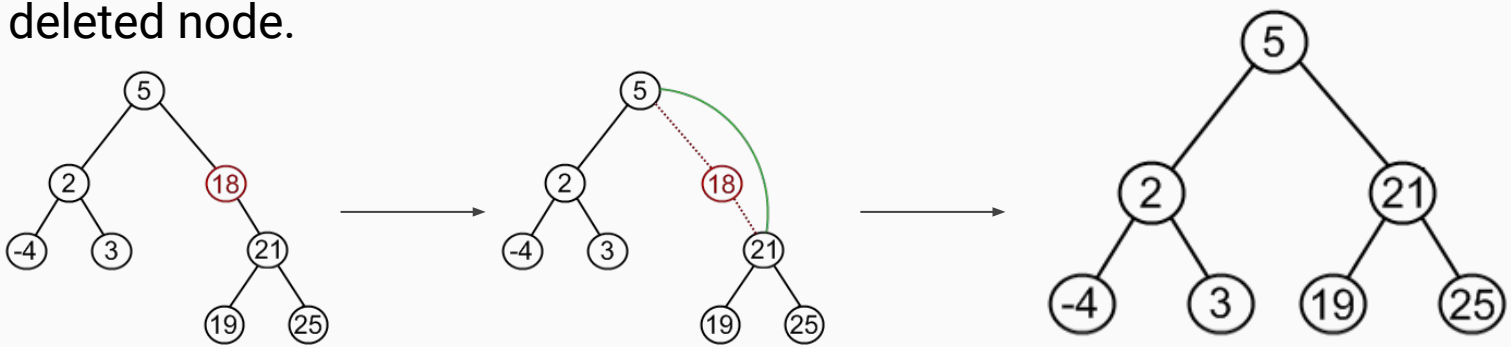
Delete Contd.

1. **Node to be removed has no children.**
   - Easiest situation
   - Simply remove the node, no further actions are required



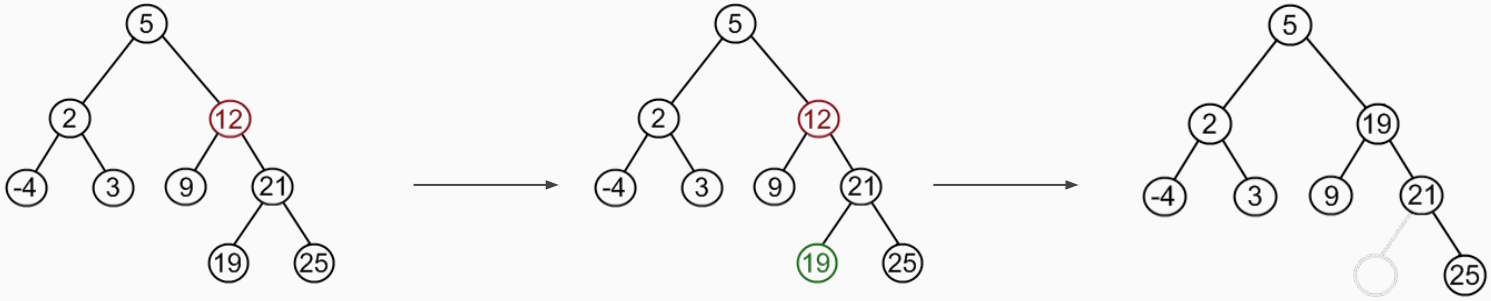
Delete Contd.

2. **Node to be removed has one child.**
   - Medium complexity
   - Removing the node will break the tree. The tree must always be connected, so link the parent of the deleted node, to the child of the deleted node.



Delete Contd.

3. **Node to be removed has two children.**
   - High complexity
   - Find a minimum value in the right subtree of the node to be removed. Replace the node to be deleted with the minimum node.

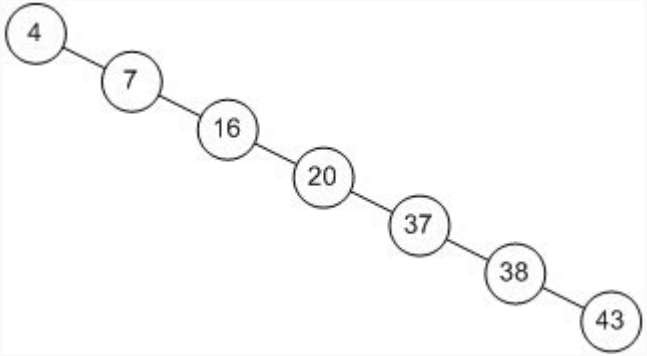


Delete Contd.

Worst Case: O(n)

Remember this structure again….

Similar to Search….



ICE 4.3 Binary Search Trees

Construct a binary search tree with the following values:

51, 29, 68, 90, 36, 40, 22, 59, 44, 99, 77, 60, 29, 83, 15, 75, 3.

Then **insert** 33, 88, 1, 36 to the BST.

Now, **delete** 44, 90, 68, 77 from the BST.

Learning Outcomes

- Understand the tree data structure and its related terminologies.

References

https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm

http://www.algolist.net/Data_structures/Binary_search_tree/Removal