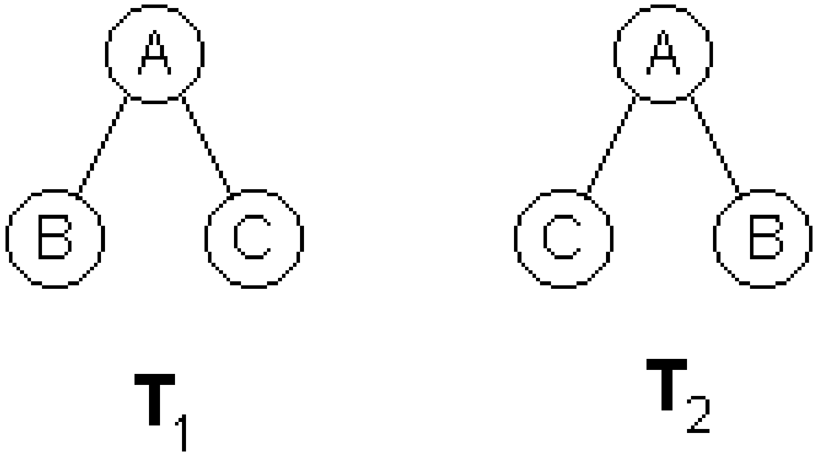
Ordered Trees

Definition

An ordered tree is an oriented tree in which the children of a node

are somehow "ordered."

If T1 and T2 are ordered trees then T1 ≠ T2 else T1 = T2.



Types of Ordered Trees

There are several types of ordered trees:

- k-ary tree
- Binomial tree
- Fibonacci tree

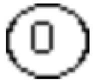
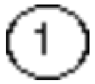
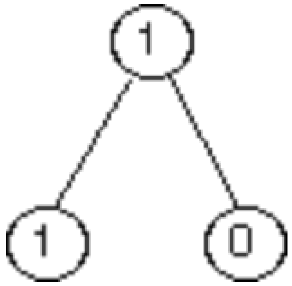
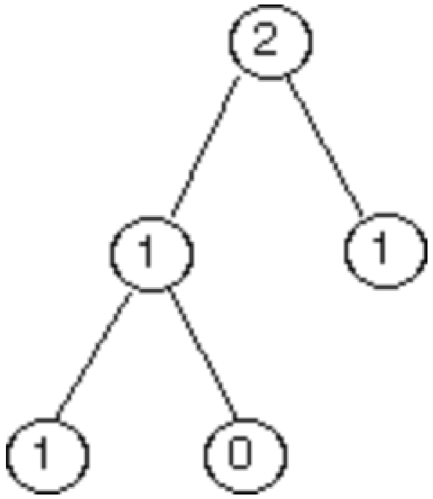
Fibonoacci Trees

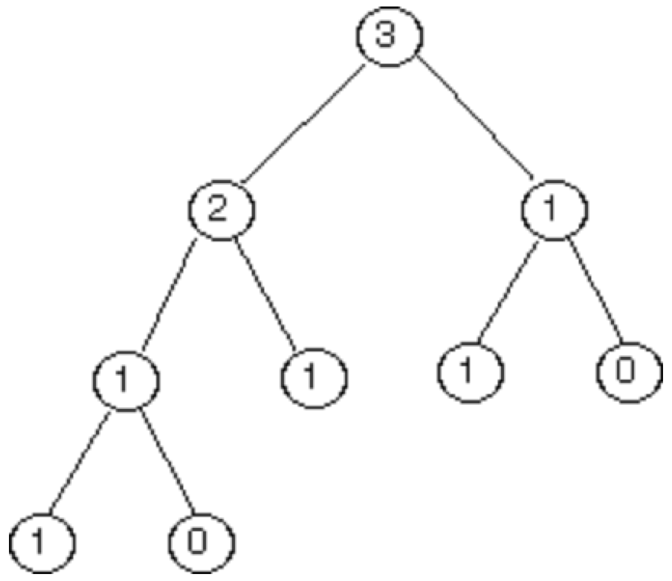
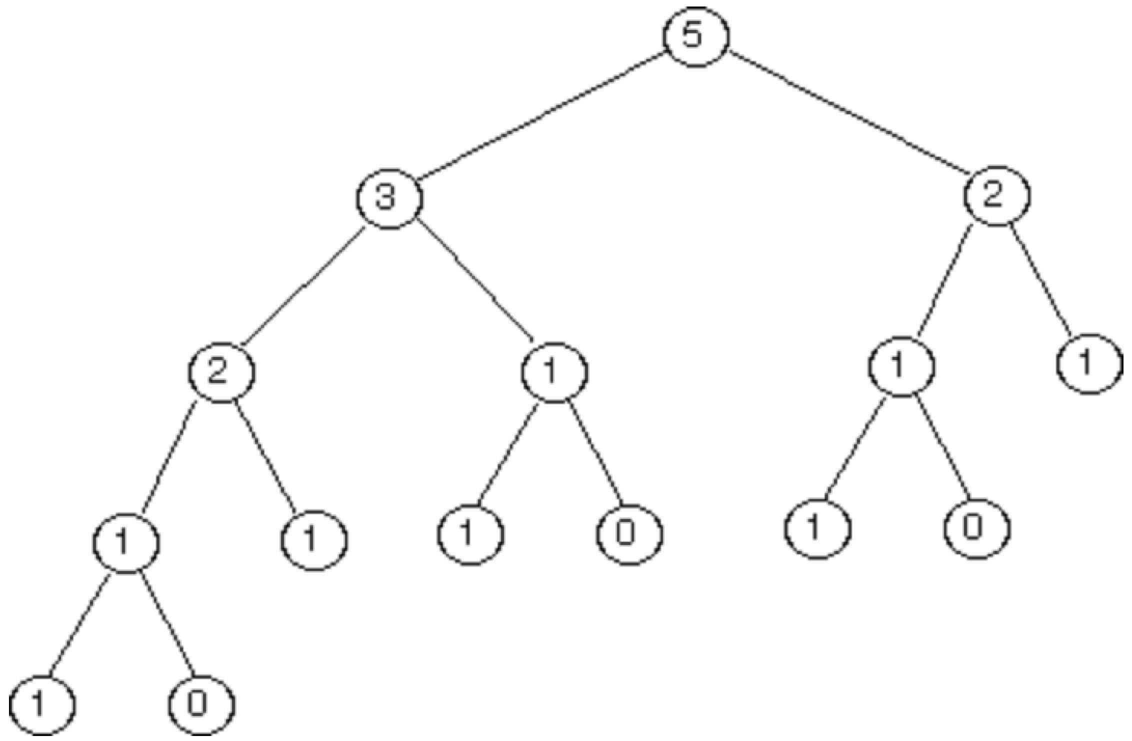
A **Fibonacci Tree** (F_k) is defined by

- F_0 is the empty tree
- F_1 is a tree with only one node
- F_{k+2} is a node whose left subtree is a F_{k+1} tree and whose right subtree is a F_k tree.

The Fibonacci sequence is defined as follows: F0 = 0, F1 = 1, and **each subsequent number in the sequence is the sum of the previous two**.

The root of a Fibonacci tree should contain the value of the nth Fibonacci number the left subtree should be the tree representing the computation of the n-1st Fibonacci number, and the right subtree should be the tree representing the computation of the n-2nd Fibonacci number. The first few Fibonacci trees appear below.

Function	Graph
`fibtree(0)`	
`fibtree(1)`	
`fibtree(2)`	
`fibtree(3)`	

`fibtree(4)`	
`fibtree(5)`	

Binomial Trees

The **Binomial Tree** (B_k) consists of a node with k children. The first child is the root of a B_{k-1} tree, the second is the root of a B_{k-2} tree, etc.

A binomial tree is a general tree with a very special shape:

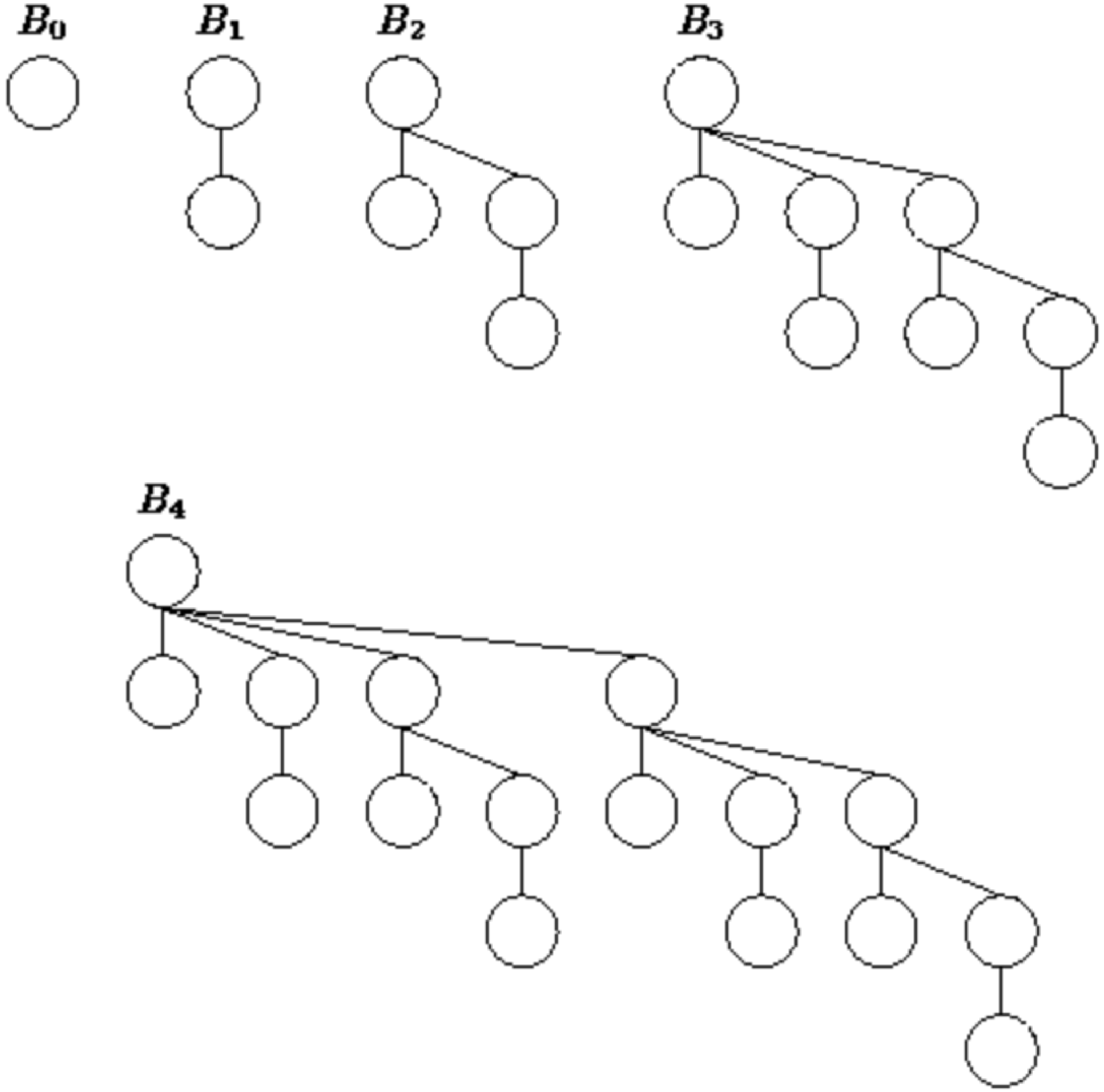
Definition (Binomial Tree) The *binomial tree of order* with root *R* is the tree defined as follows

If *k*=0, . $B_k = B_0 = \{R\}$. i.e., the binomial tree of order zero consists of a single node, *R*.

If *k*>0 $B_k = \{R, B_0, B_1, \ldots, B_{k-1}\}$. i.e., the binomial tree of order *k*>0 comprises the root *R*, and *k* binomial subtrees,.

$B_0, B_1, \ldots, B_{k-1}$.

Figure shows the first five binomial trees, $\boldsymbol{B_0} \dots \boldsymbol{B_4}$. It follows directly from Definition that the root of $\boldsymbol{B_k}$, the binomial tree of order ***k***, has degree ***k***. Since *k* may arbitrarily large, so too can the degree of the root. Furthermore, the root of a binomial tree has the largest fanout of any of the nodes in that tree.

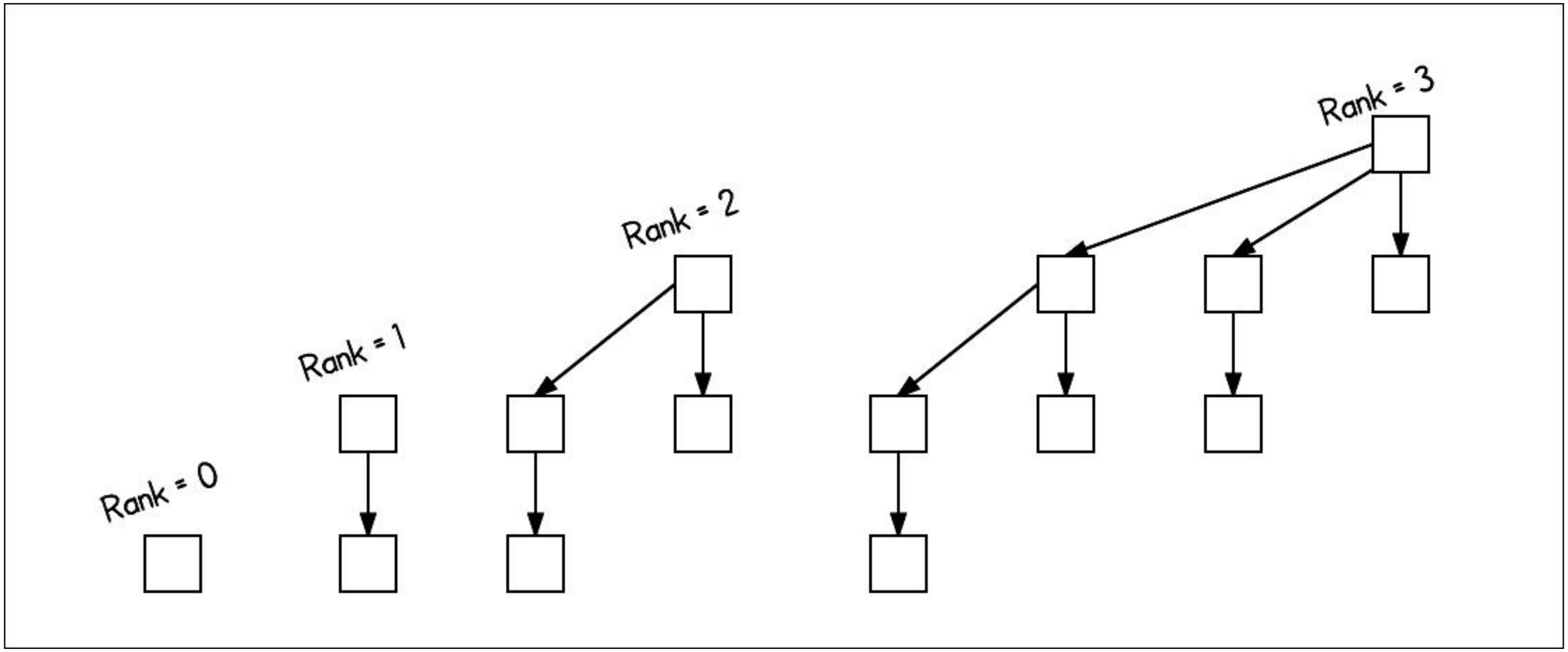

B_0
B_1
B_2
B_3
B_4

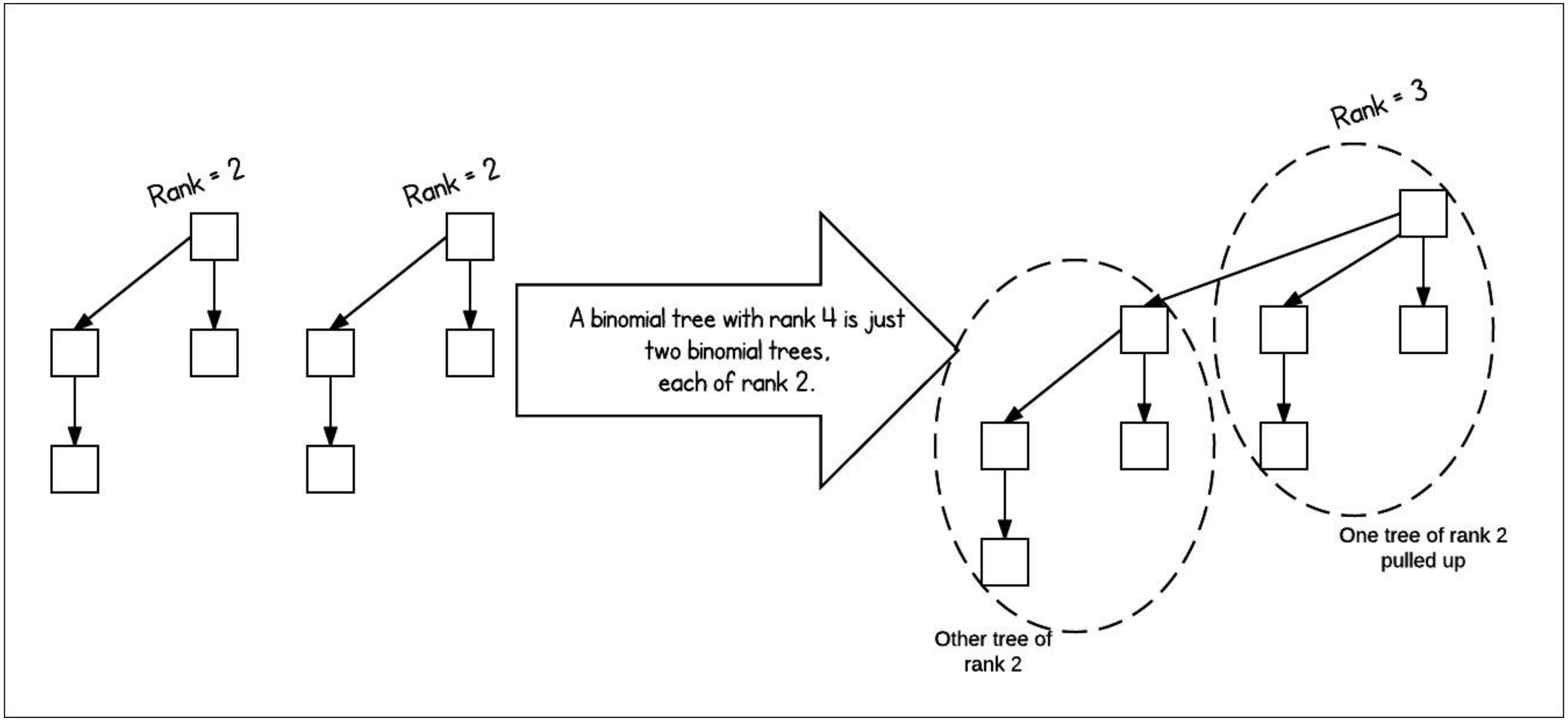
The number of nodes in a binomial tree of order *k* is a function of *k*:

Theorem The binomial tree of order *k*, $\boldsymbol{B_K}$ contains $\boldsymbol{2^K}$ nodes.

The first binomial tree has just one node. Its height is equal to **0**. A binomial tree of height **1** is formed from two binomial trees, each of height **0**. A binomial tree of height **2** is formed from two binomial trees, each of height **1**.

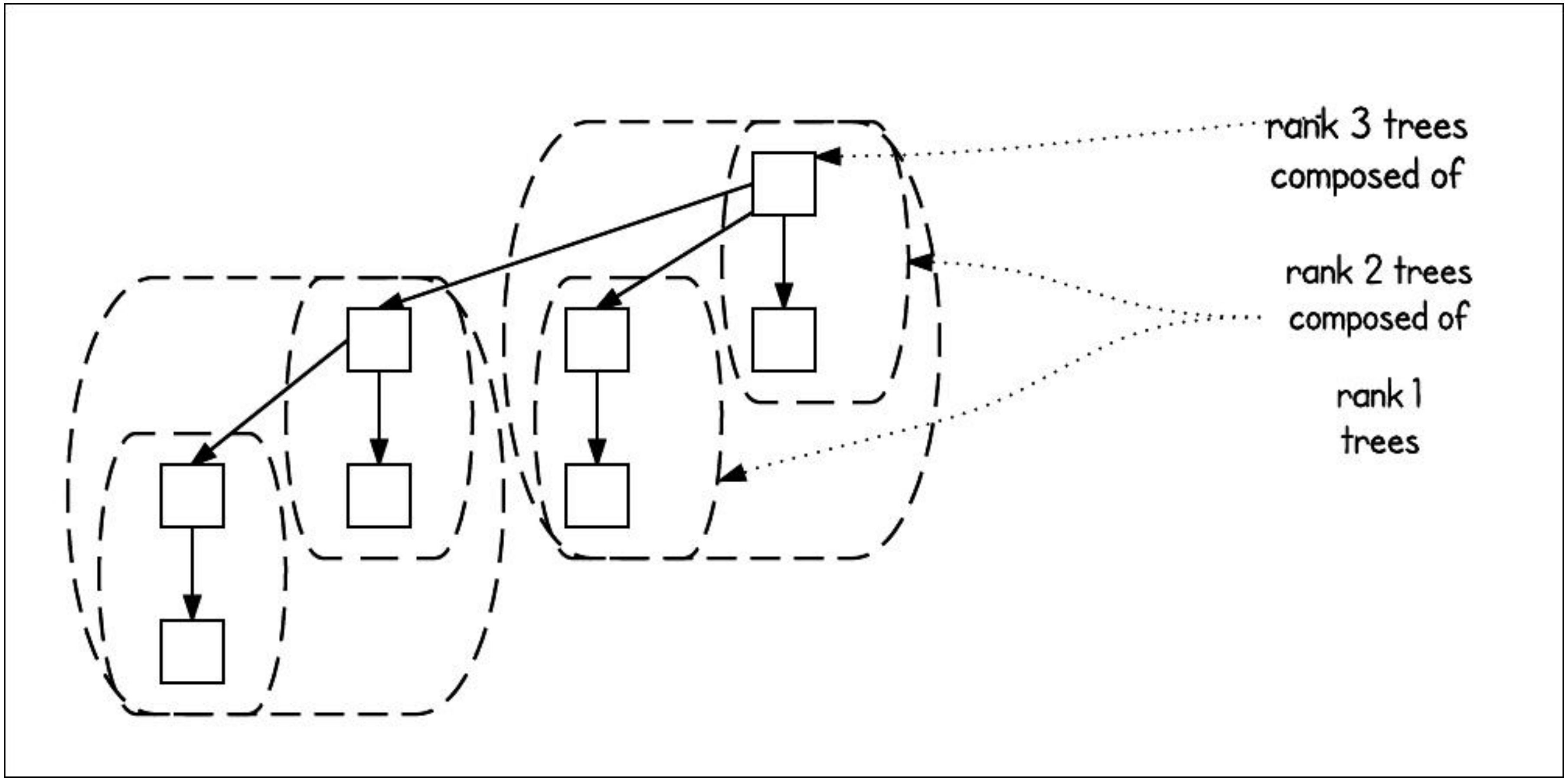
The tree is defined in terms of itself, recursively. For example, the following figure shows two binomial trees of rank **2**. When the right tree is pulled up, the left tree becomes its left child. The resulting tree is of rank 3 with $\mathbf{2^3 = 8}$ nodes.


Rank = 0
Rank = 1
Rank = 2
Rank = 3

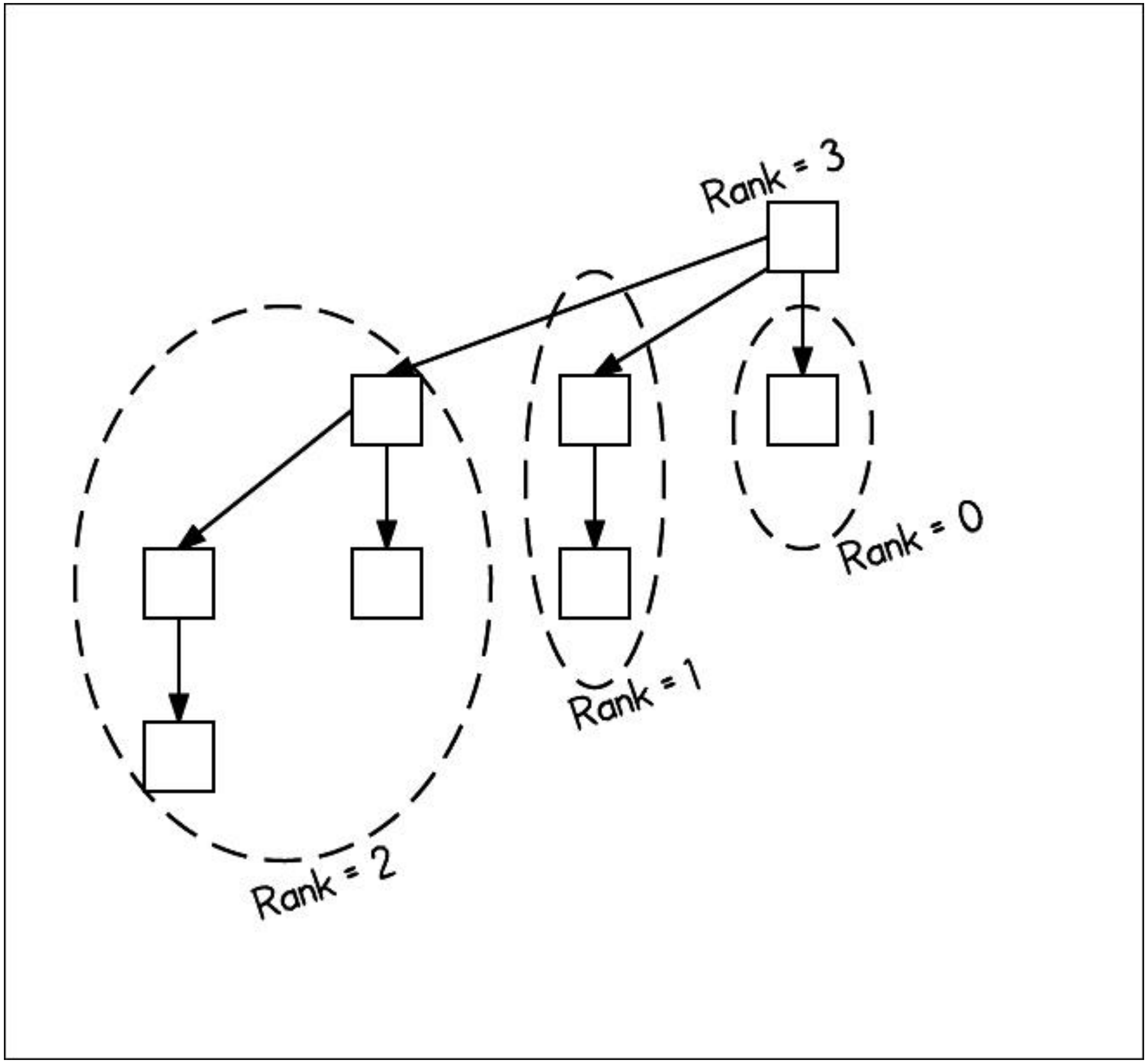

Rank = 2
Rank = 2
A binomial tree with rank 4 is just two binomial trees, each of rank 2.
Rank = 3
One tree of rank 2 pulled up
Other tree of rank 2

The definition of binomial trees could also be stated as follows:

a **binomial tree with rank k is composed of subtrees with rank (k-1)**. Here is a pictorial way of stating this:



You can also look at a binomial tree as a list of binary trees.



k-ary Trees

A k-ary tree is a tree in which the chilren of a node appear at distinct index positions in 0..k-1. *This means the maximum number of children for a node is k*.

Some k-ary trees have special names

2-ary trees are called **binary trees**.

3-ary trees are called **trinary trees** or **ternary trees**.

1-ary trees are called **lists**.

You have to draw k-ary trees carefully. In a binary tree, if a node has one child, you can't draw a vertical line from the parent to the child. Every child in a binary tree is either a left or a right child;

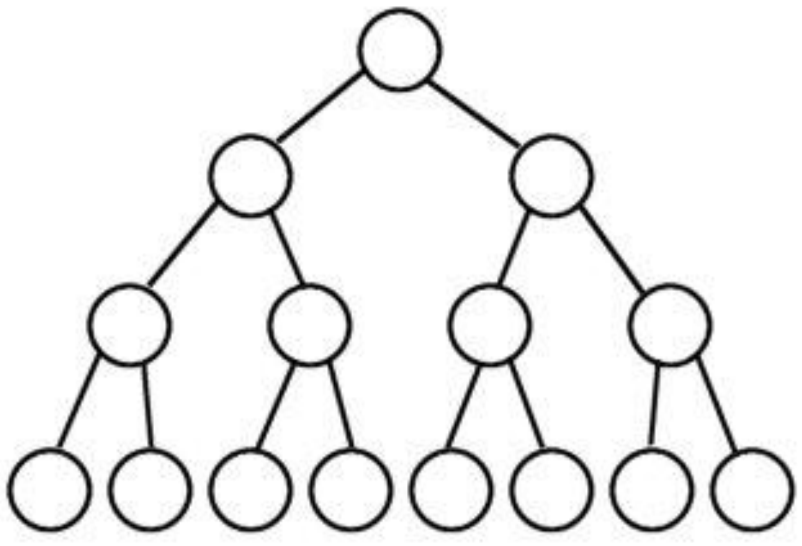
Complete Binary Tree

A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.

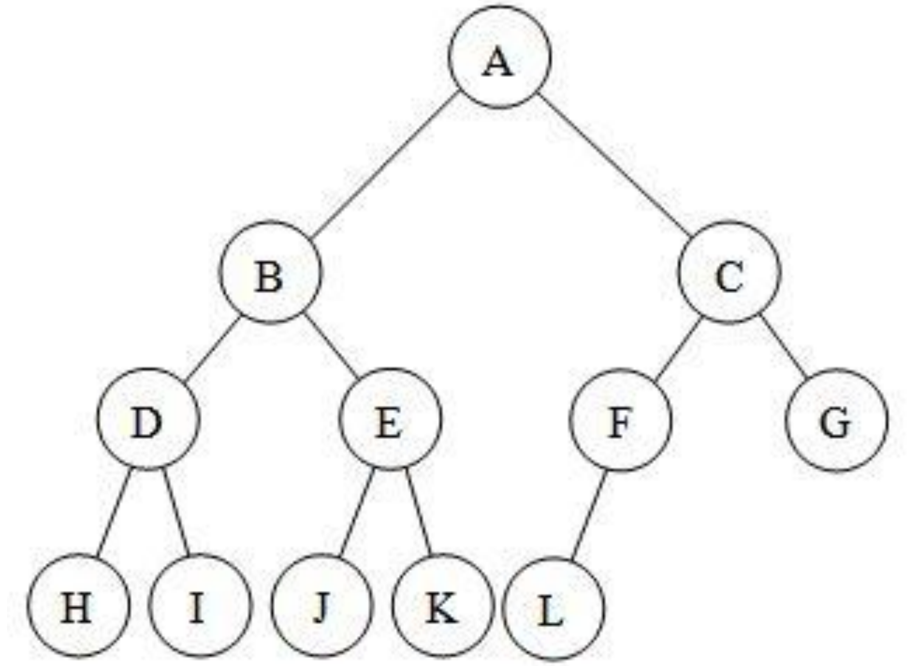
A complete binary tree is just like a full binary tree, but with two major differences

- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.

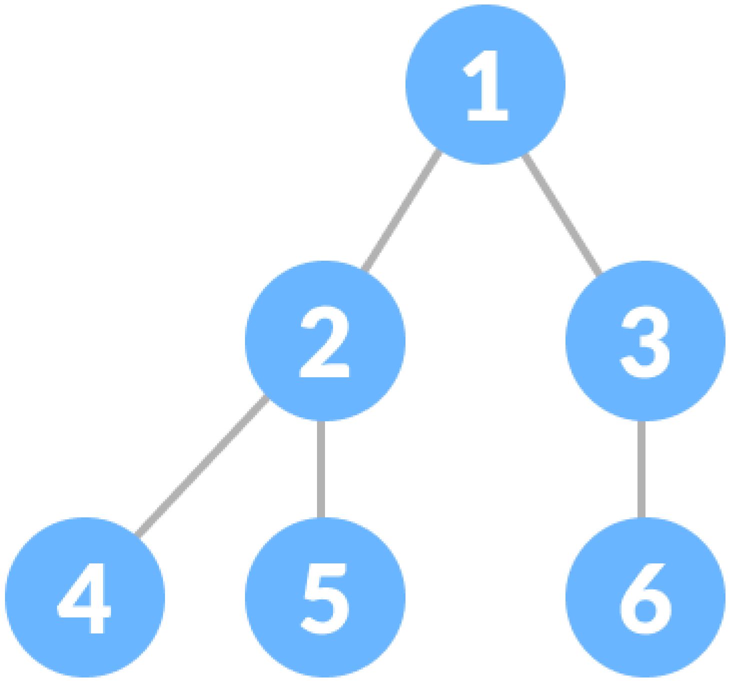
Full Binary Tree



A **full binary tree** (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children.

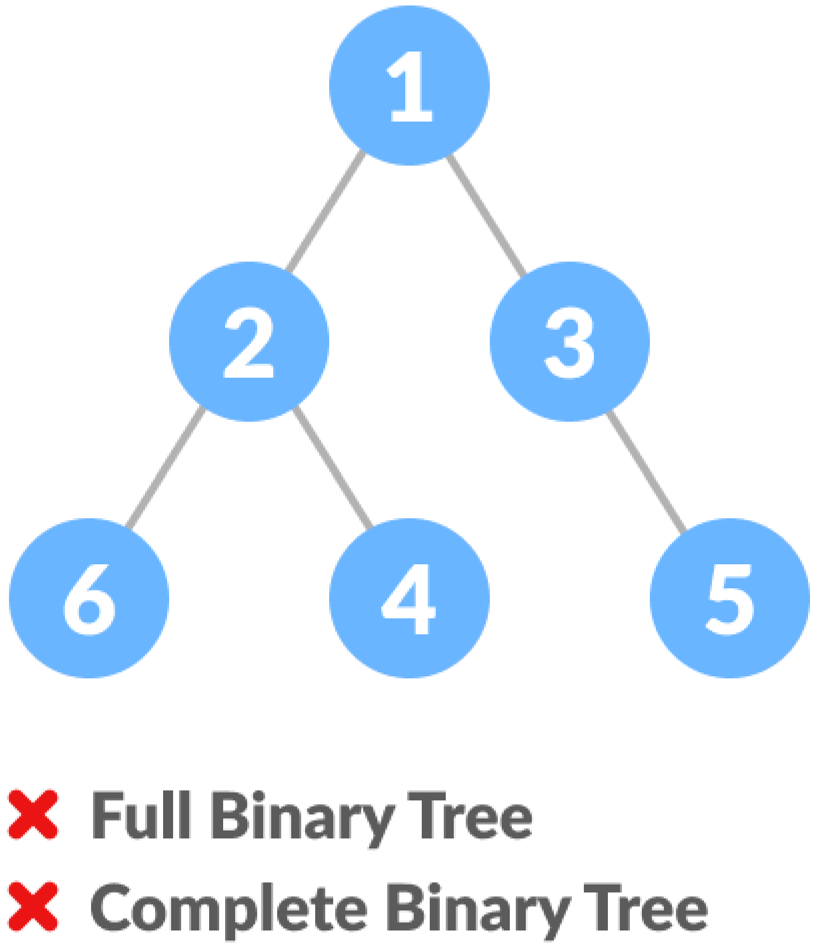


A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

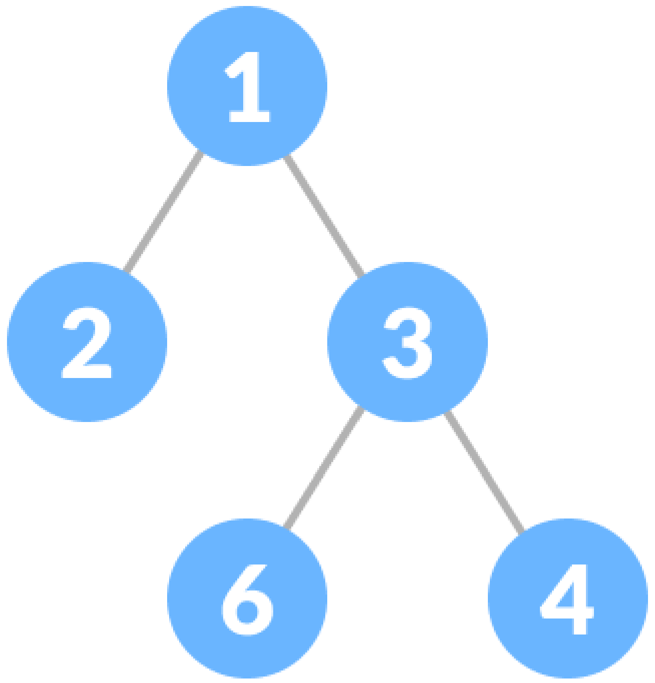


Complete Binary Tree

Full Binary Tree vs Complete Binary Tree



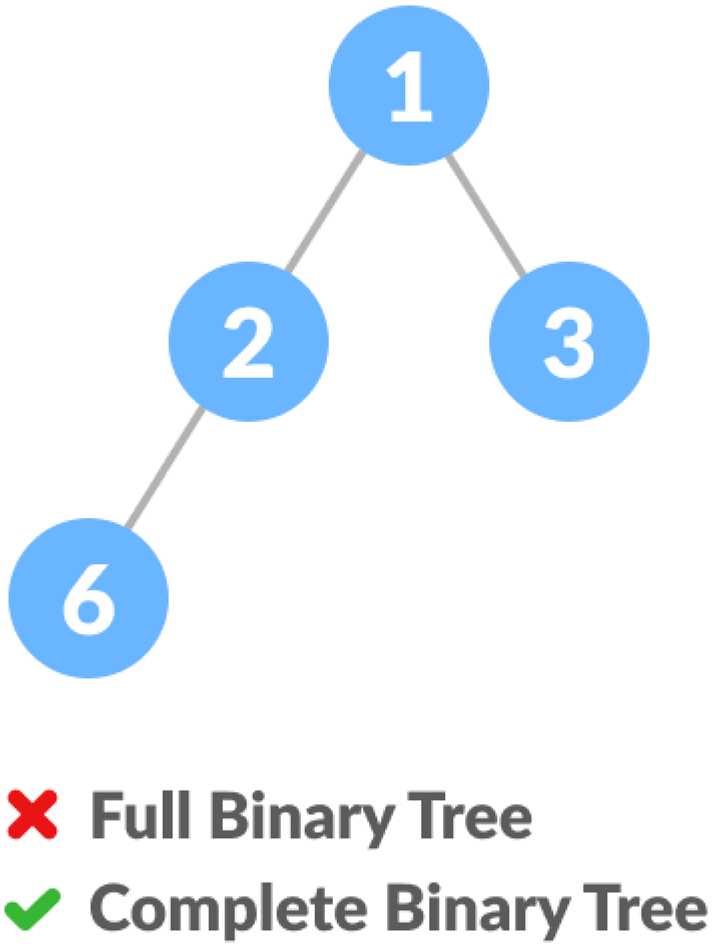
Comparison between full binary tree and complete binary tree



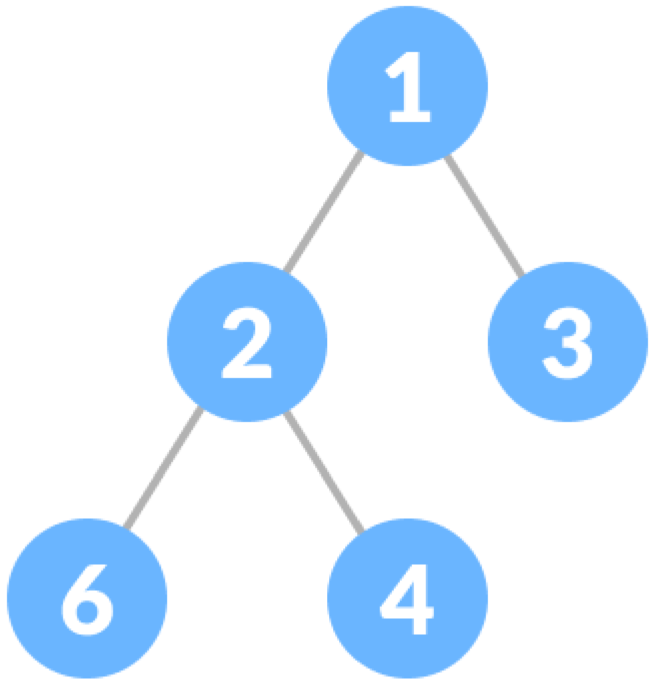
✓ **Full Binary Tree**

✗ **Complete Binary Tree**

Comparison between full binary tree and complete binary tree



Comparison between full binary tree and complete binary tree



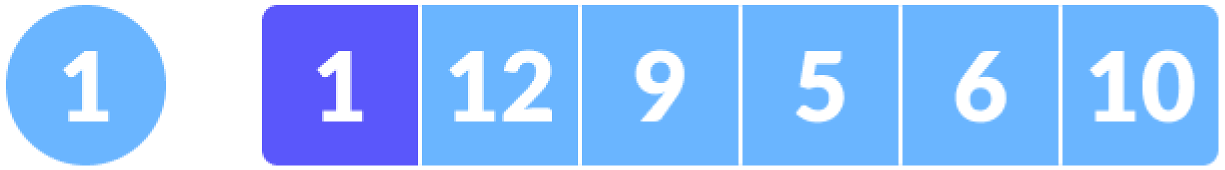
✓ Full Binary Tree

✓ Complete Binary Tree

Comparison between full binary tree and complete binary tree

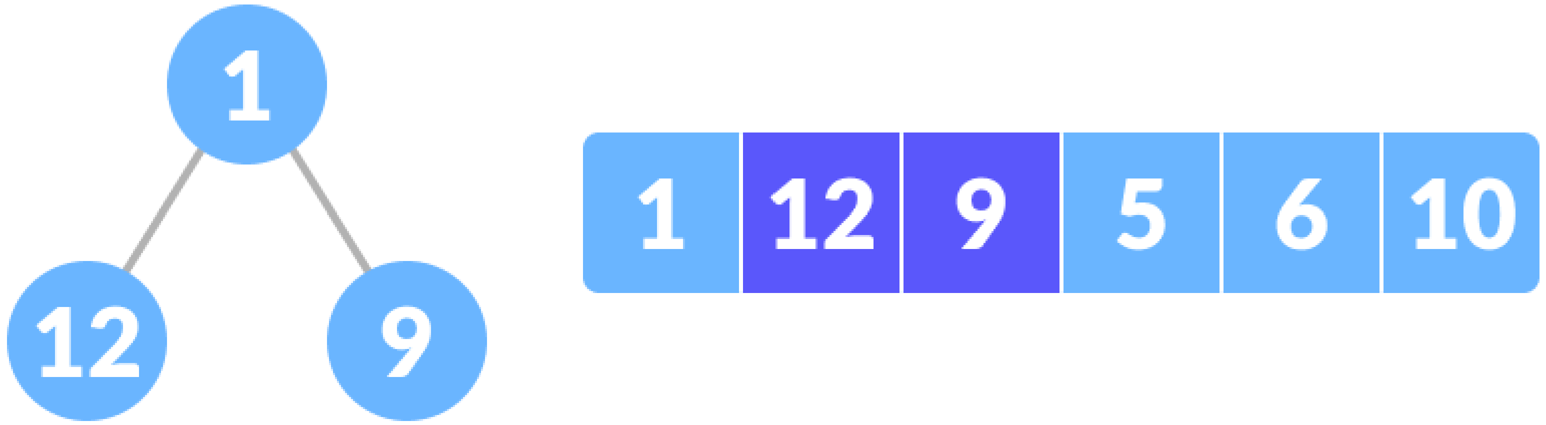
How a Complete Binary Tree is Created?

Select the first element of the list to be the root node. (no. of elements on level-I: 1)



Select the first element as root

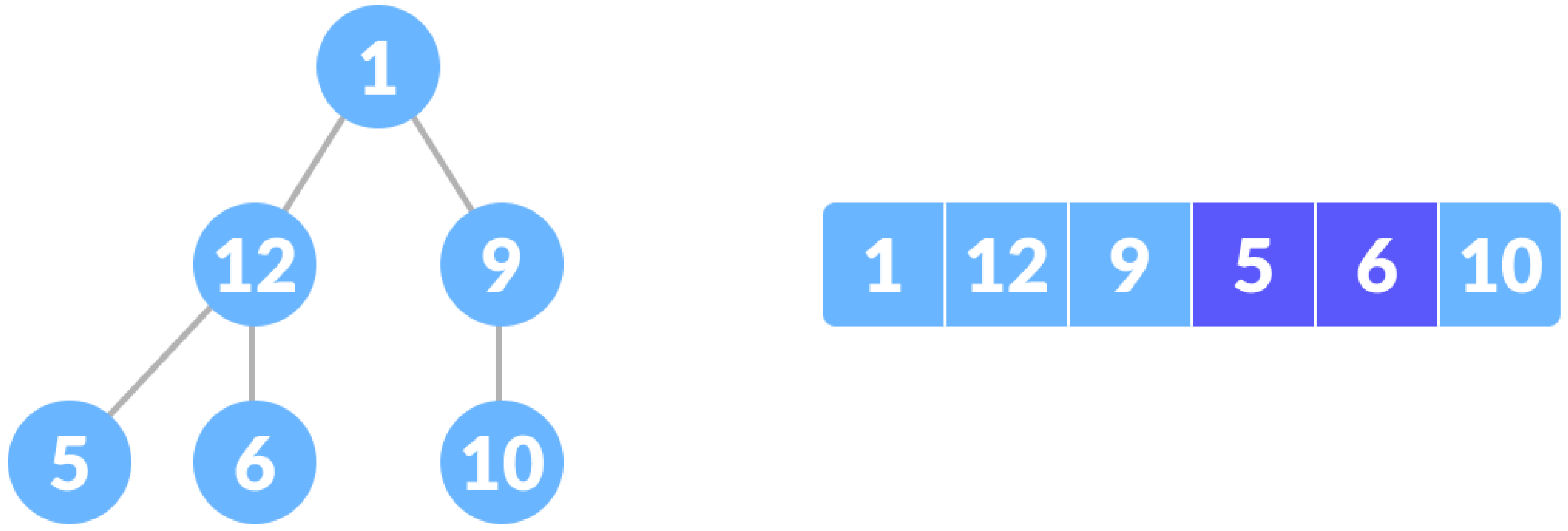
Put the second element as a left child of the root node and the third element as the right child. (no. of elements on level-II: 2)



12 as a left child and 9 as a right child

Put the next two elements as children of the left node of the second level. Again, put the next two elements as children of the right node of the second level (no. of elements on level-III: 4) elements).

Keep repeating until you reach the last element.



5 as a left child and 6 as a right child

Binary Search tree

A binary search tree is a binary tree with a special property called the BST-property, which is given as follows:

For all nodes x and y, if y belongs to the left subtree of x, then the key at y is less than the key at x, and if y belongs to the right subtree of x, then the key at y is greater than the key at x.

We will assume that the keys of a BST are pairwise distinct.

Each node has the following attributes:

• p, left, and right, which are pointers to the parent, the left child, and the right child, respectively, and

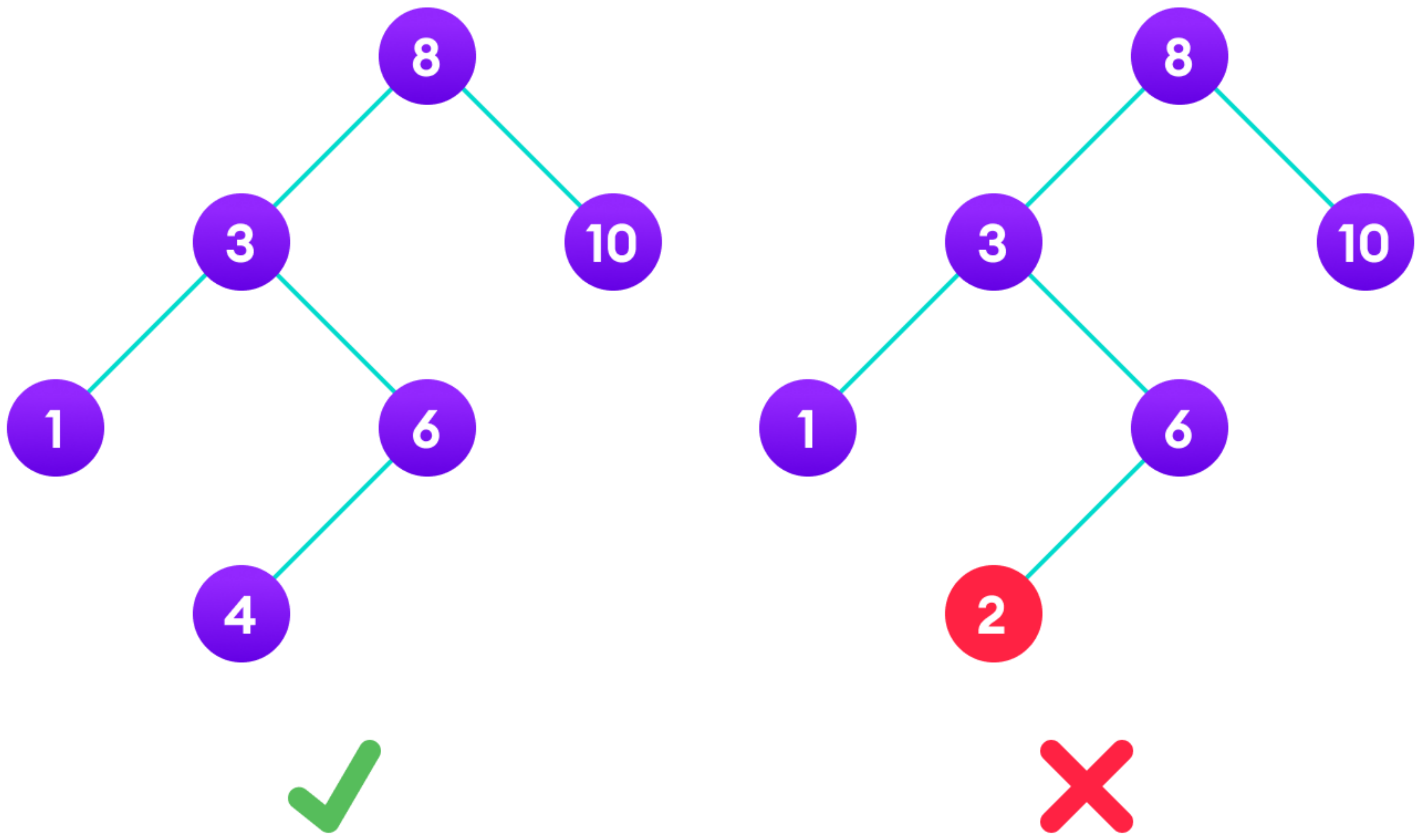
• key, which is key stored at the node.

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

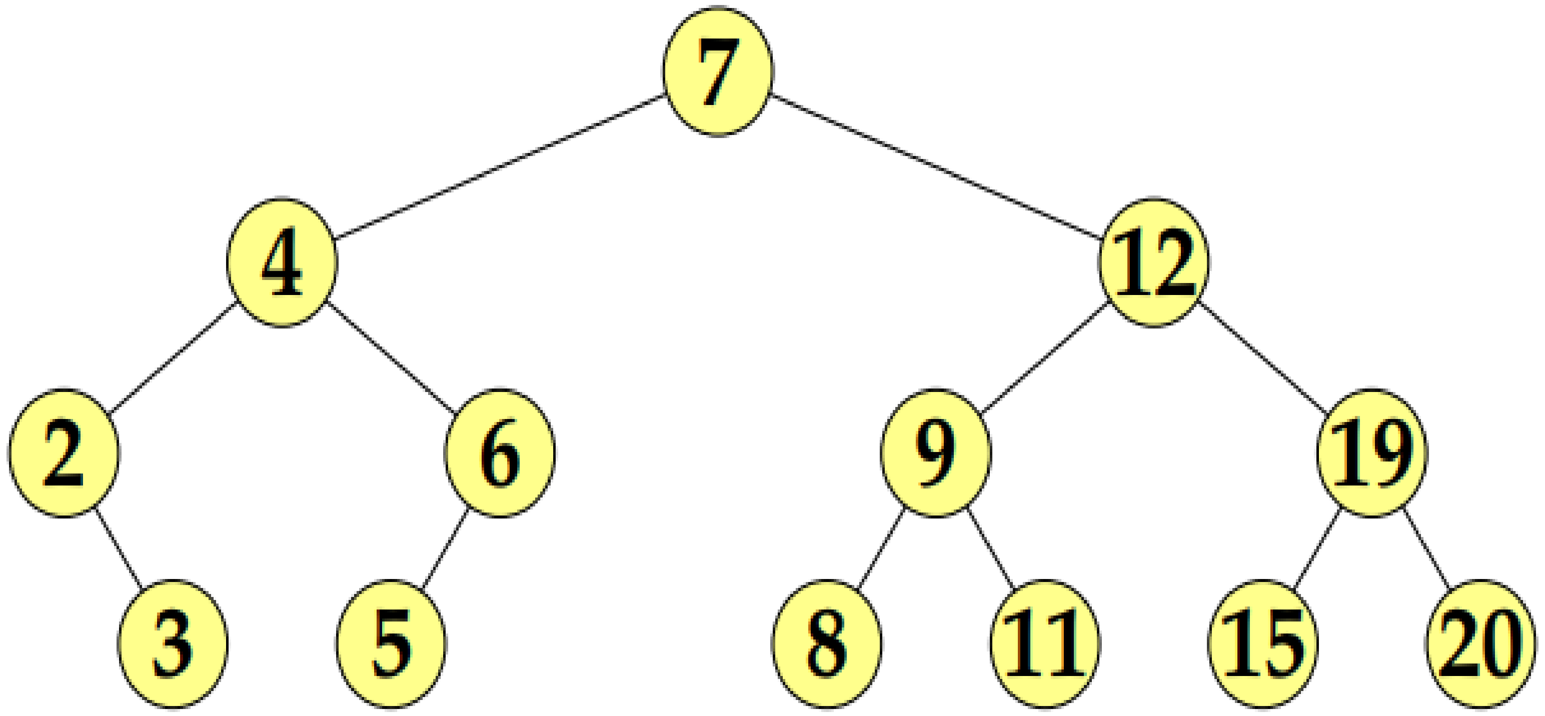
- It is called a binary tree because each tree node has a maximum of two children.
- It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.

The properties that separate a binary search tree from a regular binary tree is

1. All nodes of left subtree are less than the root node
2. All nodes of right subtree are more than the root node
3. Both subtrees of each node are also BSTs i.e. they have the above two properties

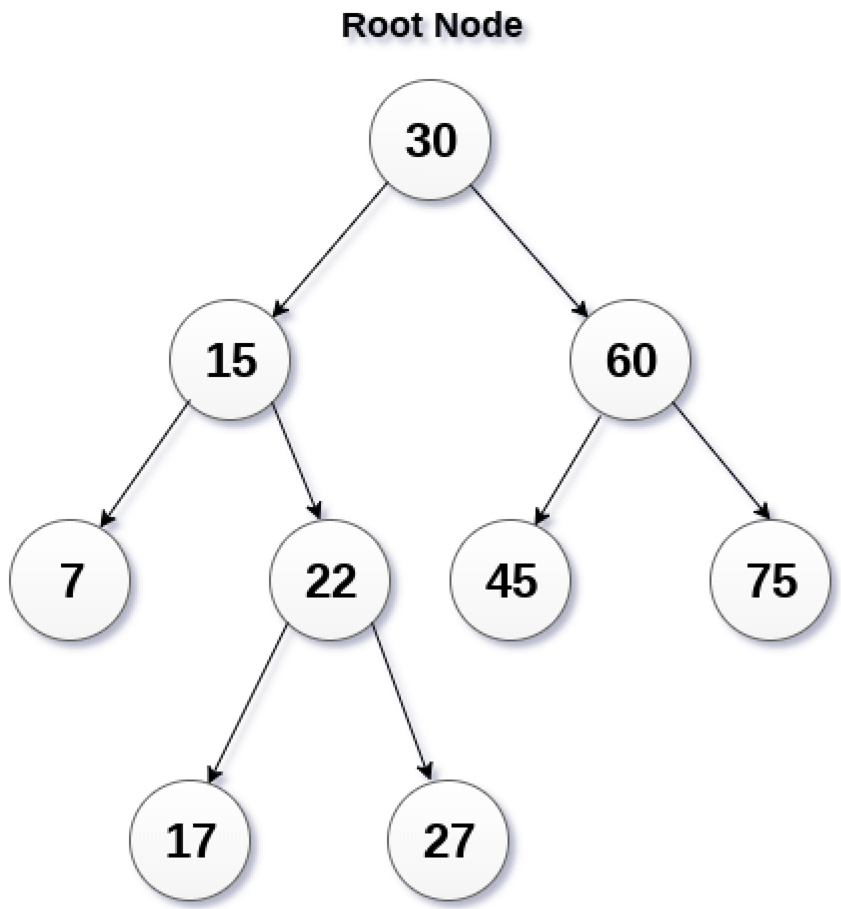


A tree having a right subtree with one value smaller than the root is shown to demonstrate that it is not a valid binary search tree


7
4
12
2
6
9
19
3
5
8
11
15
20

Binary Search Tree

1. Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.
2. In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
3. Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.
4. This rule will be recursively applied to all the left and right sub-trees of the root.



Binary Search Tree

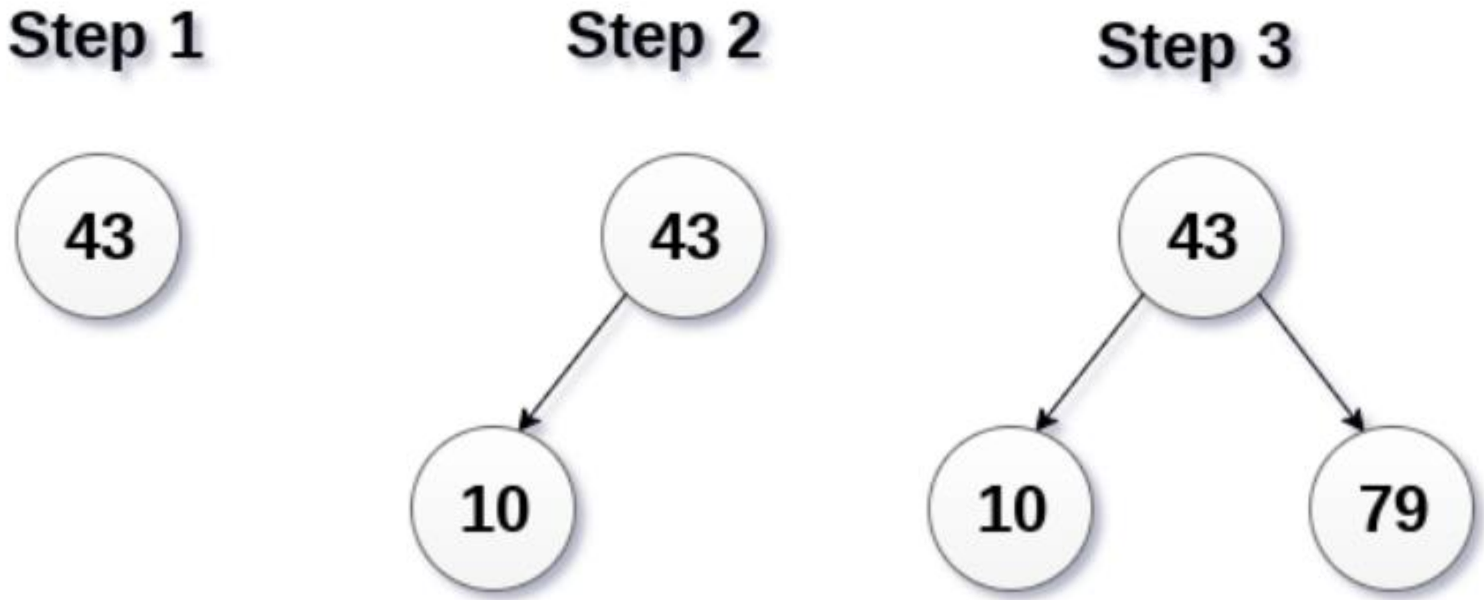
Advantages of using binary search tree

1. Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
2. The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes $o(\log_2 n)$ time. In worst case, the time it takes to search an element is $0(n)$.
3. It also speed up the insertion and deletion operations as compare to that in array and linked list.

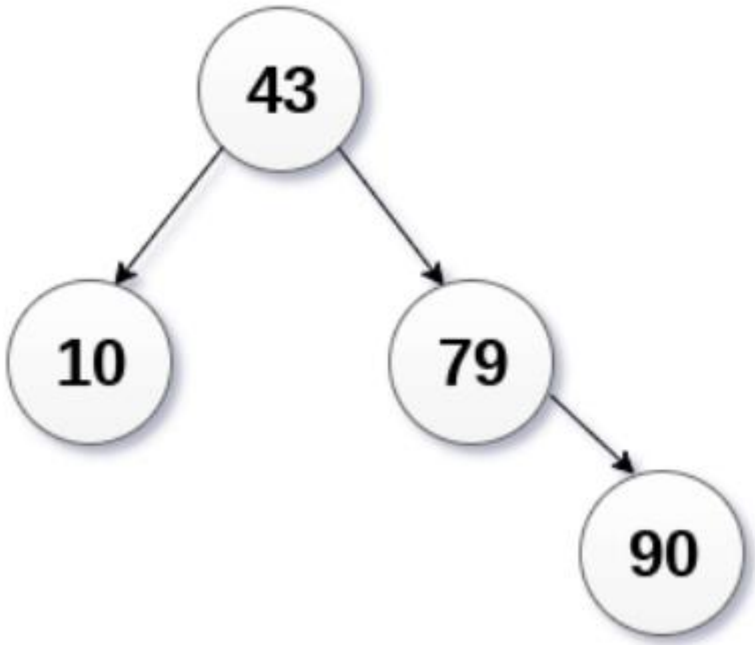
Create the binary search tree using the following data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50

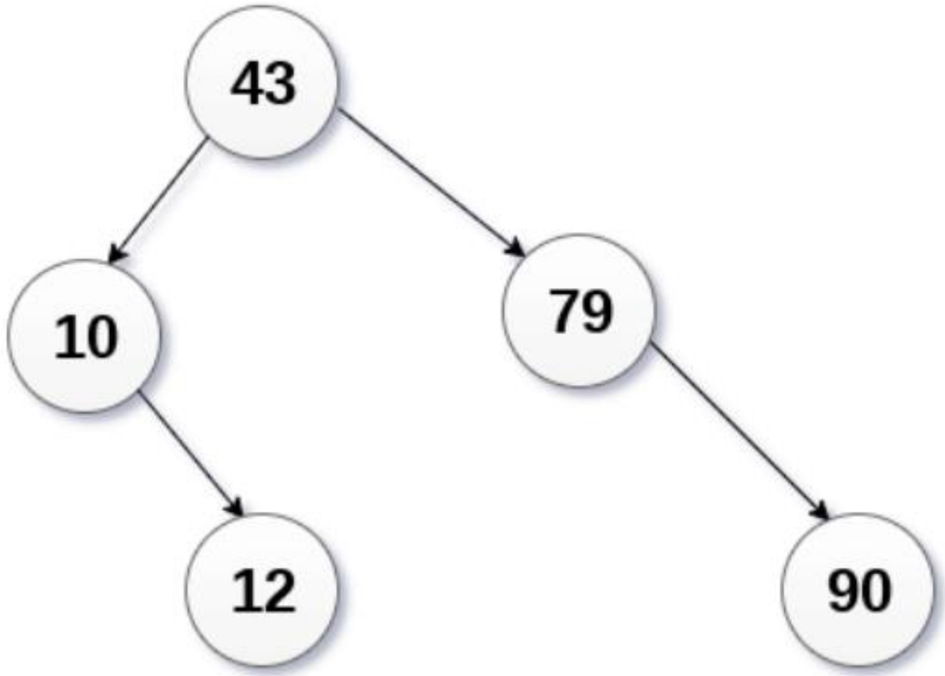
- Insert 43 into the tree as the root of the tree.
- Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
- Otherwise, insert it as the root of the right of the right sub-tree.
- The process of creating BST by using the given elements, is shown in the image below.


Step 1
Step 2
Step 3
43
43
10
43
10
79

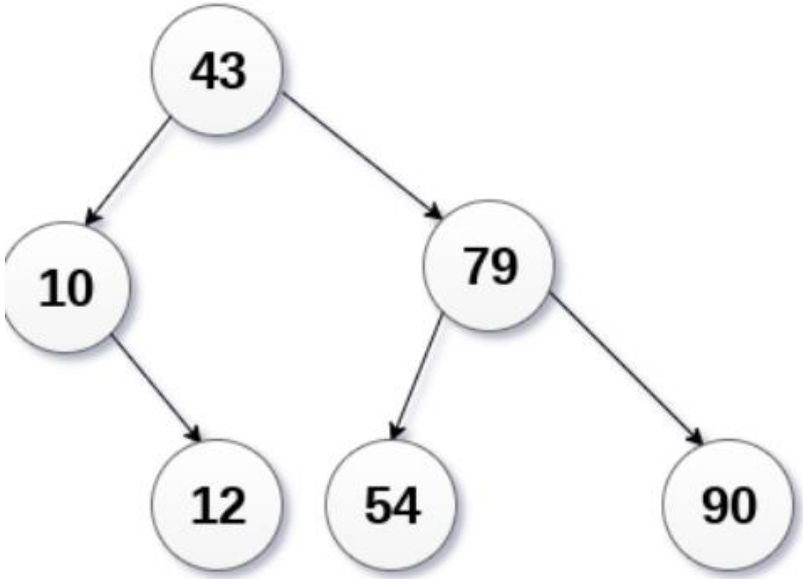
Step 4



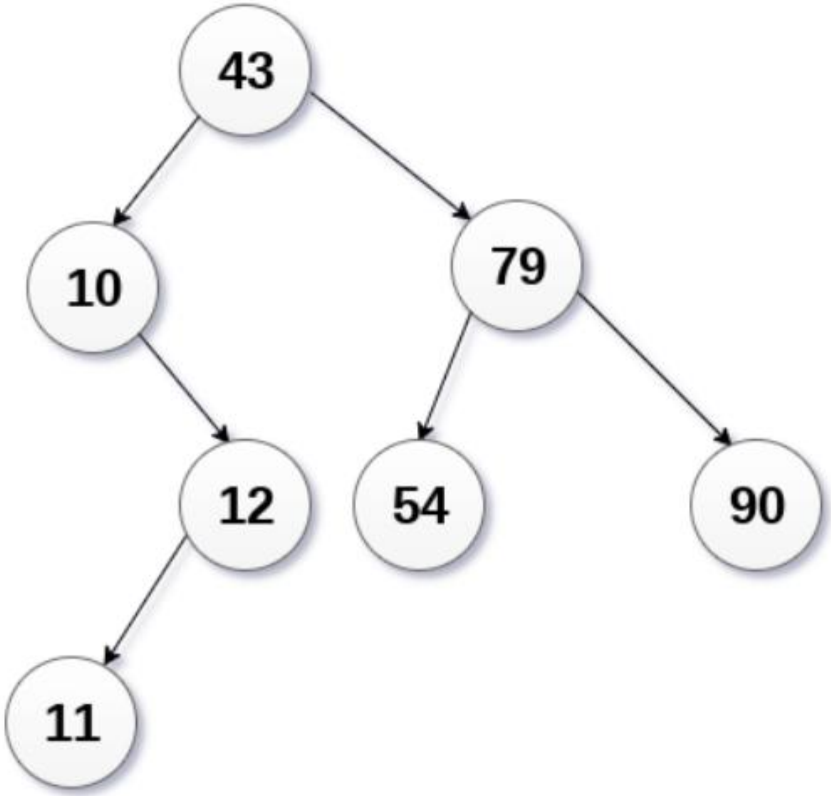
Step 5



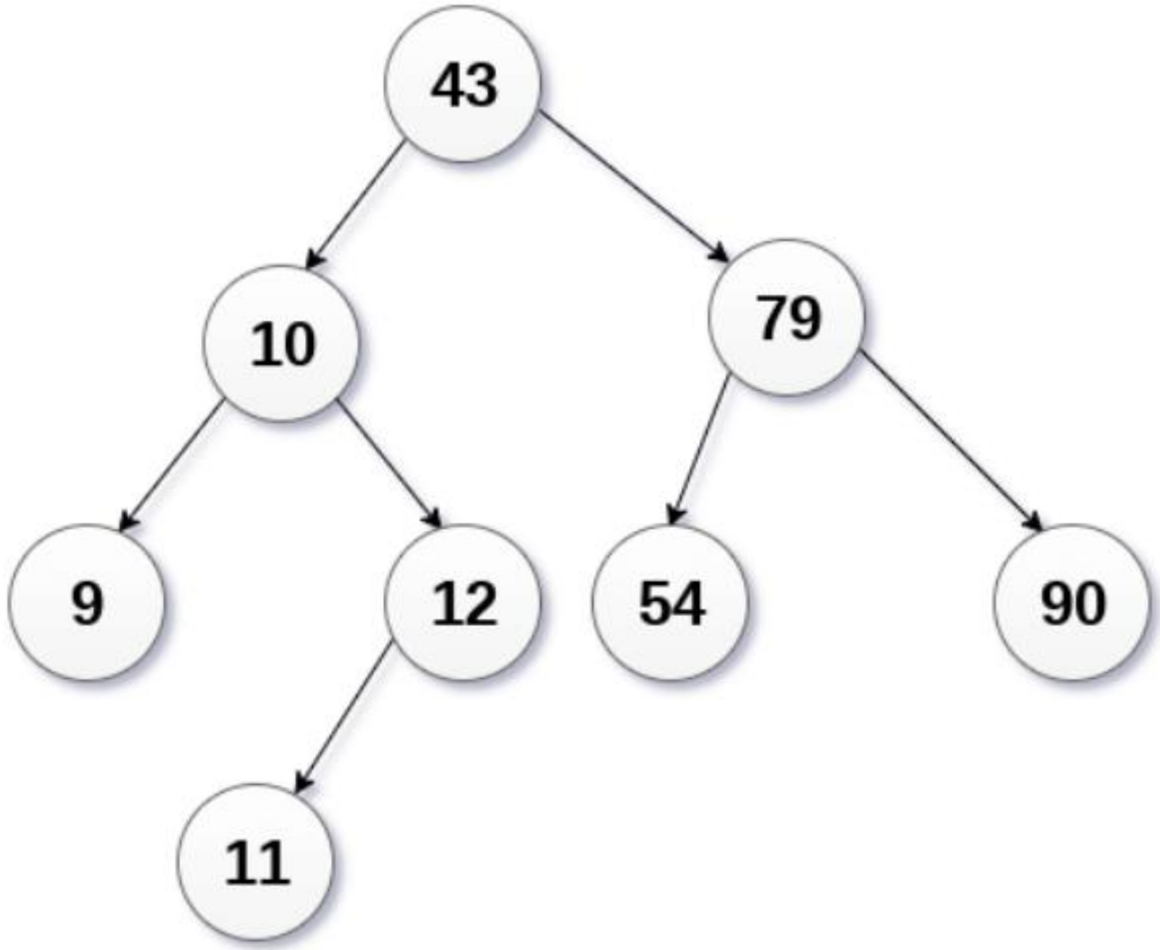
Step 6



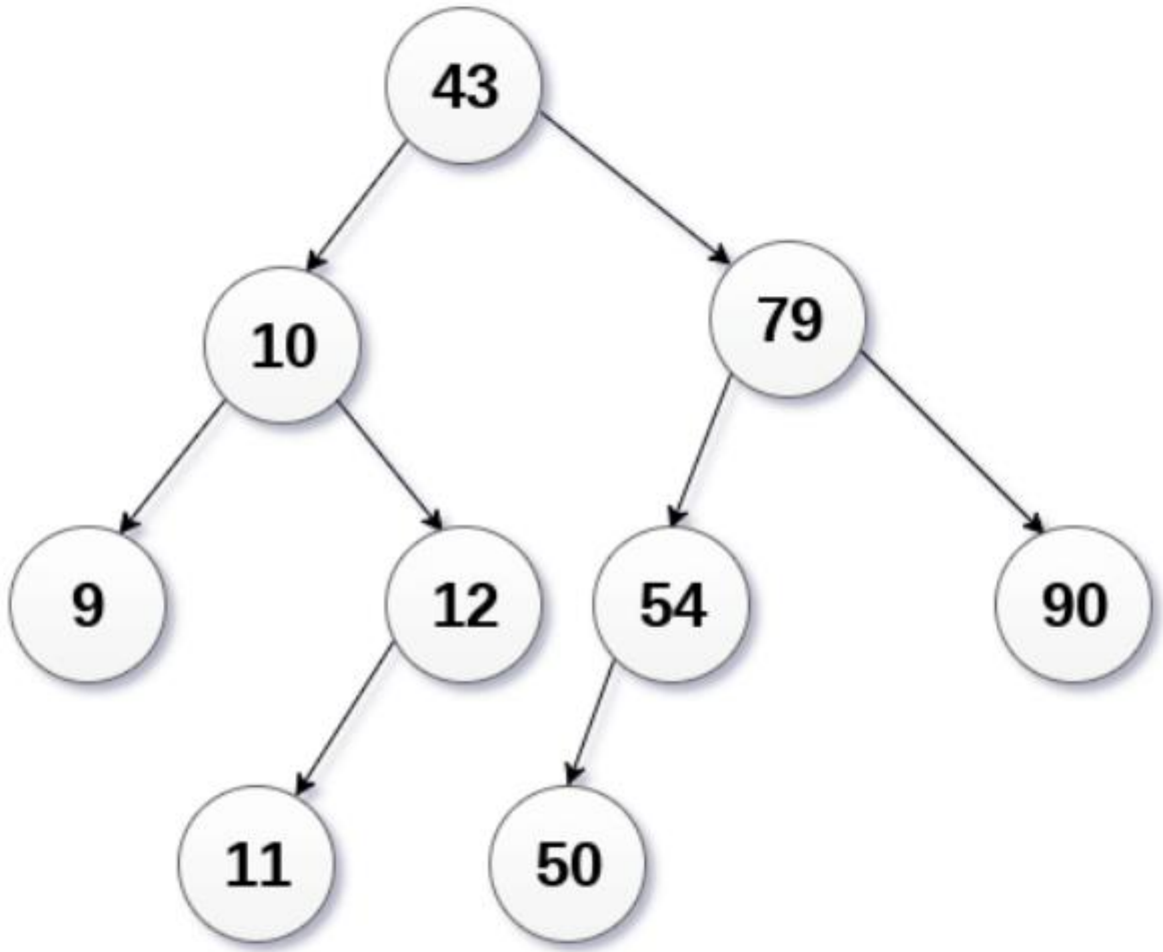
Step 7



Step 8



Step 9



Operations on Binary Search Tree

There are many operations which can be performed on a binary search tree.

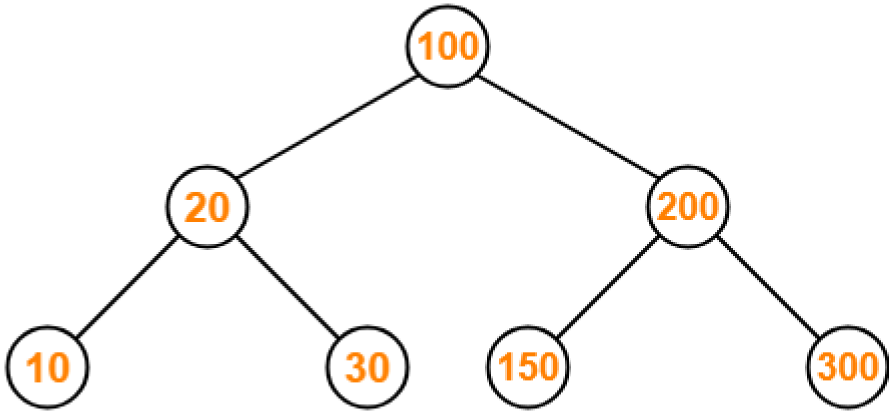
SN	Operation	Description
1	Searching in BST	Finding the location of some specific element in a binary search tree.
2	Insertion in BST	Adding a new element to the binary search tree at the appropriate location so that the property of BST do not violate.
3	Deletion in BST	Deleting some specific node from a binary search tree. However, there can be various cases in deletion depending upon the number of children, the node have.

Types of Traversals

The Binary Search Tree can be traversed in the following ways:

- Pre-order Traversal
- In-order Traversal
- Post-order Traversal

1. The *pre-order* traversal will visit nodes in **Parent-LeftChild-RightChild** order.
2. The *in-order* traversal will visit nodes in **LeftChild-Parent-RightChild** order. In this way, the tree is traversed in an ascending order of keys.
3. The *post-order* traversal will visit nodes in **LeftChild-RightChild-Parent** order.



Binary Search Tree

Preorder Traversal-

100 , 20 , 10 , 30 , 200 , 150 , 300

Inorder Traversal-

10 , 20 , 30 , 100 , 150 , 200 , 300

Postorder Traversal-

10 , 30 , 20 , 150 , 300 , 200 , 100

Notes :

- Inorder traversal of a binary search tree always yields all the nodes in increasing order.

Unlike Binary Trees,

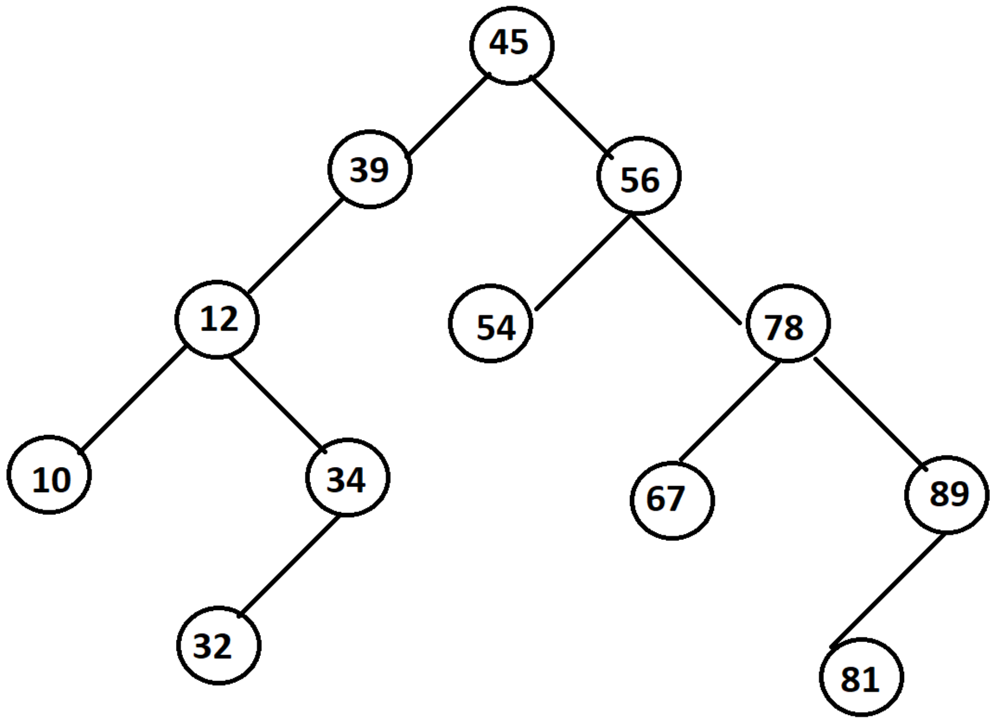
- A binary search tree can be constructed using only preorder or only postorder traversal result.
- This is because inorder traversal can be obtained by sorting the given result in increasing order.

Q. Create a Binary Search Tree using following sequence of numbers 7 , 5 , 1 , 8 , 3 , 6 , 0 , 9 , 4 , 2.

Q. Create a Binary Search Tree using following sequence of numbers 30 , 20 , 10 , 15 , 25 , 23 , 39 , 35 , 42

Q. Create a Binary Search Tree using following sequence of numbers 98, 2, 48, 12, 56, 32, 4, 67, 23, 55, 46.

(a) Insert 21, 39, 45, 54, and 63 into the tree.
(b) Delete values 23, 56, 2, and 45 from the tree.



Q. (i) Find the result of in-order, preorder, and post order traversals.

(ii) Insert 11, 22, 33, 44, 55, 66, and 77 in the tree.

Rebuild a binary tree from Inorder and Preorder traversals

The following procedure demonstrates on how to rebuild tree from given inorder and preorder traversals of a binary tree:

- Preorder traversal visits Node, left subtree, right subtree recursively
- Inorder traversal visits left subtree, node, right subtree recursively
- Since we know that the first node in Preorder is its root, we can easily locate the root node in the inorder traversal and hence we can obtain left subtree and right subtree from the inorder traversal recursively

Preorder Traversal: 1 2 4 8 9 10 11 5 3 6 7

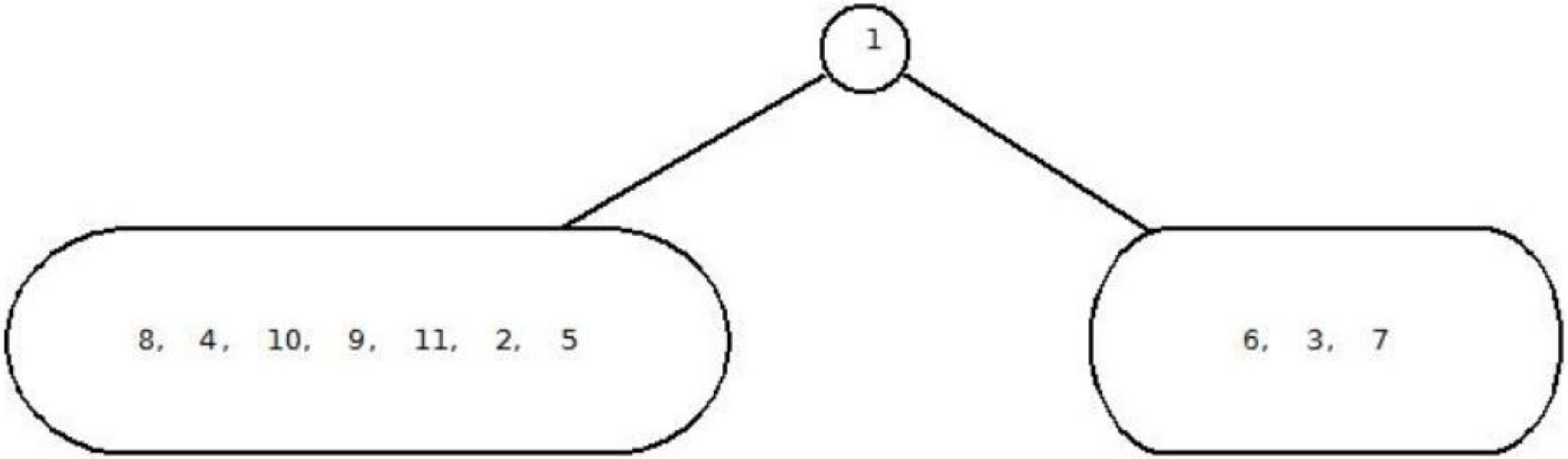
Inorder Traversal: 8 4 10 9 11 2 5 1 6 3 7

Iteration 1:

Root – {1}

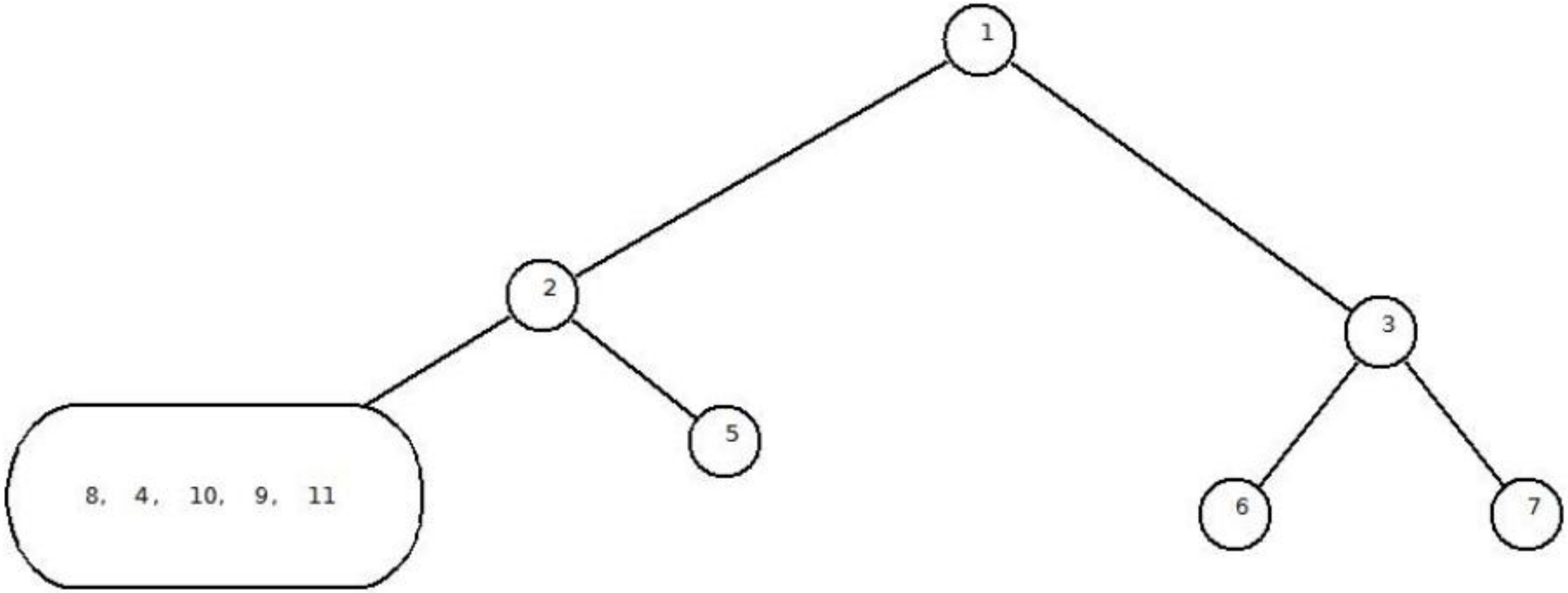
Left Subtree – {8,4,10,9,11,2,5}

Right Subtree – {6,3,7}



Iteration 2:

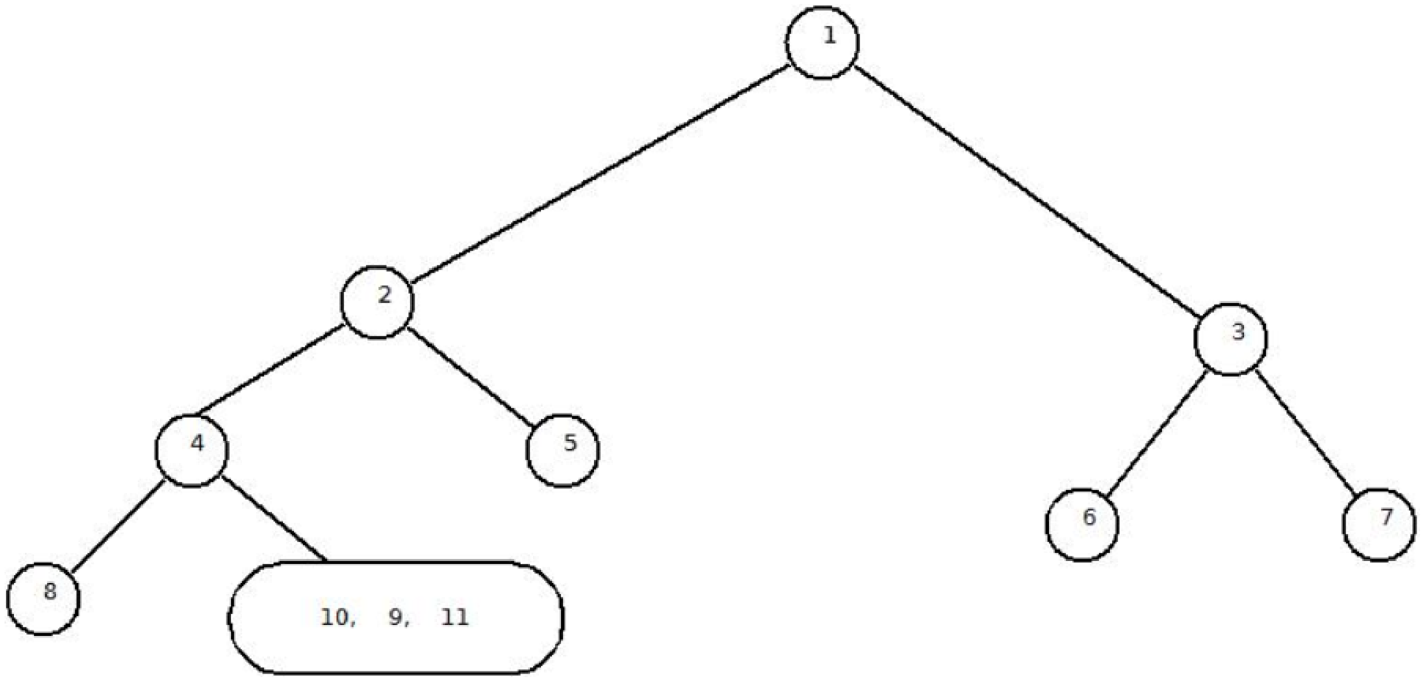
Root – {2}
 	
Left Subtree – {8,4,10,9,11}
 	
Right Subtree – {5}
	Root – {3}
	
	Left Subtree – {6}
	
	Right Subtree – {7}



Iteration 3:

Root – {2}
Left Subtree – {8,4,10,9,11}
Right Subtree – {5}
		Root – {3}
		Left Subtree – {6}
		Right Subtree – {7}

Root – {4}
Left Subtree – {8}
Right Subtree – {10,9,11}
	Done	Done

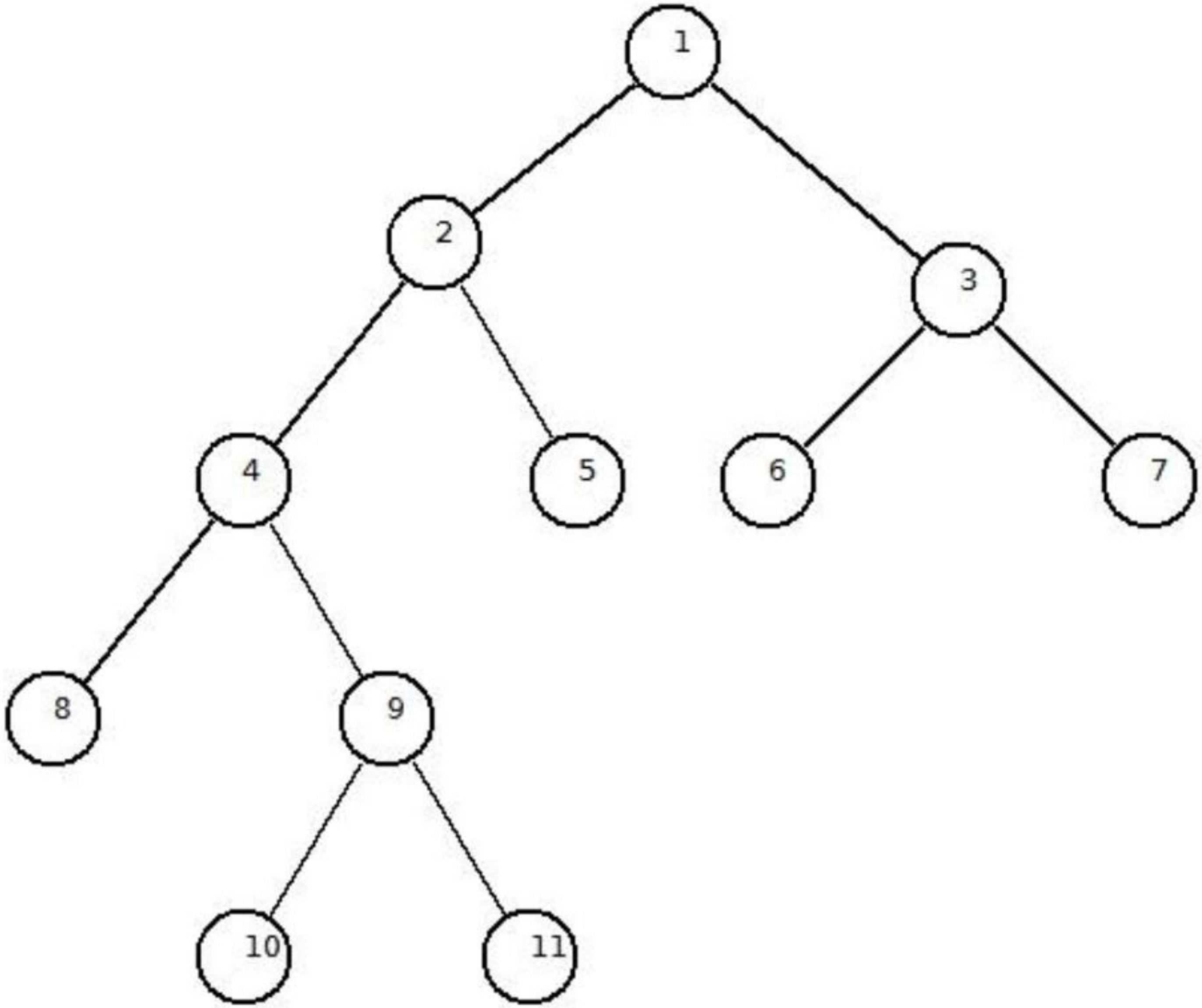


Iteration 4:

Root – {2}
Left Subtree – {8,4,10,9,11}
Right Subtree – {5}
			Root – {3}
			Left Subtree – {6}
			Right Subtree – {7}

Root – {4}
Left Subtree – {8}
Right Subtree – {10,9,11}
		Done	Done

Done	R – {9}
	Left ST – {10}
	Right ST- {11}
		Done	Done


1
2
3
4
5
6
7
8
9
10
11

Construct a binary tree from inorder and preorder traversal

```
Input:


Inorder Traversal:  { 4, 2, 1, 7, 5, 8, 3, 6 }
Preorder Traversal: { 1, 2, 4, 3, 5, 7, 8, 6 }
```

Inorder sequence: D B E A F C

Preorder sequence: A B D E C F

inorder = {2,5,6,10,12,14,15};

preorder = {10,5,2,6,14,12,15};

Construct Binary Tree from Inorder and Postorder Traversal

in-order: 4 2 5 (1) 6 7 3 8

post-order: 4 5 2 6 7 8 3 (1)