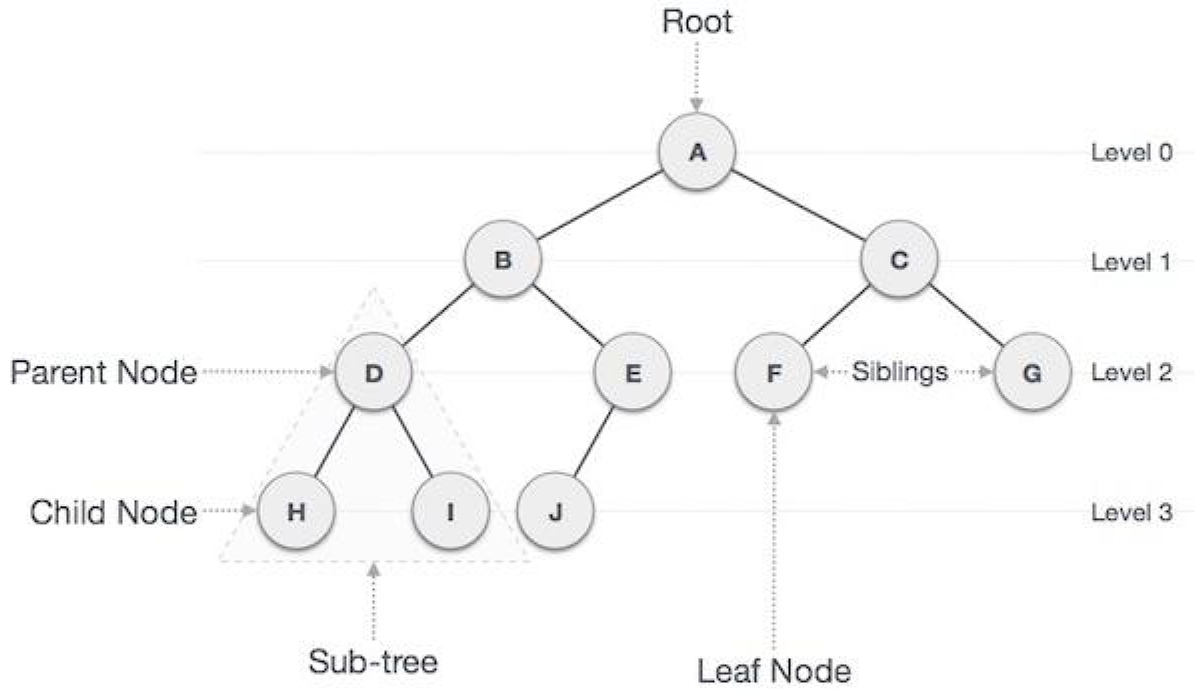
# TREE

Tree represents the nodes connected by edges. We will discuss binary trees or binary search trees specifically.

A binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operations are as fast as in the linked list.



## Important Terms

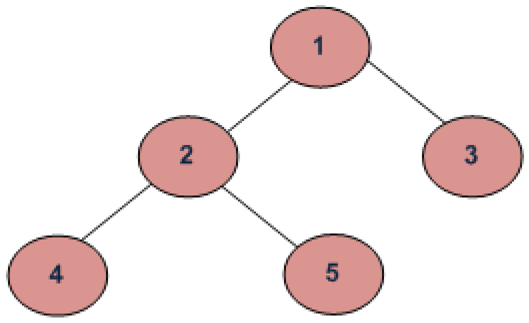
Following are the important terms with respect to trees.

- Path − Path refers to the sequence of nodes along the edges of a tree.
- Root − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- Parent − Any node except the root node has one edge upward to a node called the parent.

- Child − The node below a given node connected by its edge downward is called its child node.
- Leaf − The node which does not have any child node is called the leaf node.
- Subtree − Subtree represents the descendants of a node.
- Visiting − Visiting refers to checking the value of a node when control is on the node.
- Traversing − Traversing means passing through nodes in a specific order.
- Levels − The level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- keys − Key represents a value of a node based on which a search operation is to be carried out for a node.

# Tree Traversals (Inorder, Preorder, and Postorder)

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Depth First Traversals:

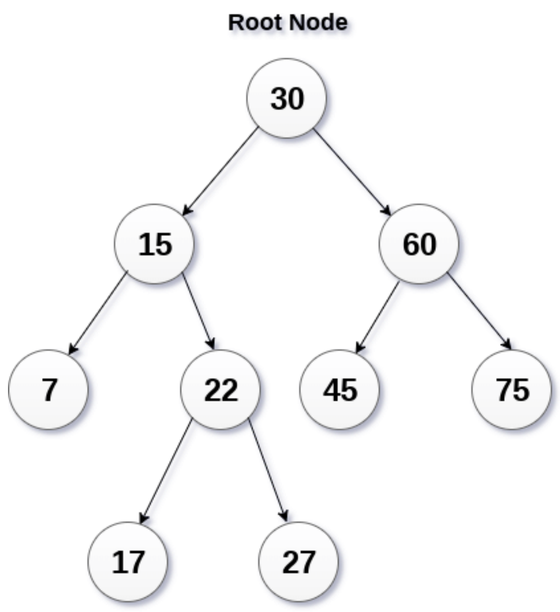
(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

# 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 an 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, the 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

A Binary search tree is shown in the above figure. As the constraint applied on the BST, we can see that the root node 30 doesn't contain any value greater than or equal to 30 in its left sub-tree and it also doesn't contain any value less than 30 in its right sub-tree.

## Advantages of using binary search tree

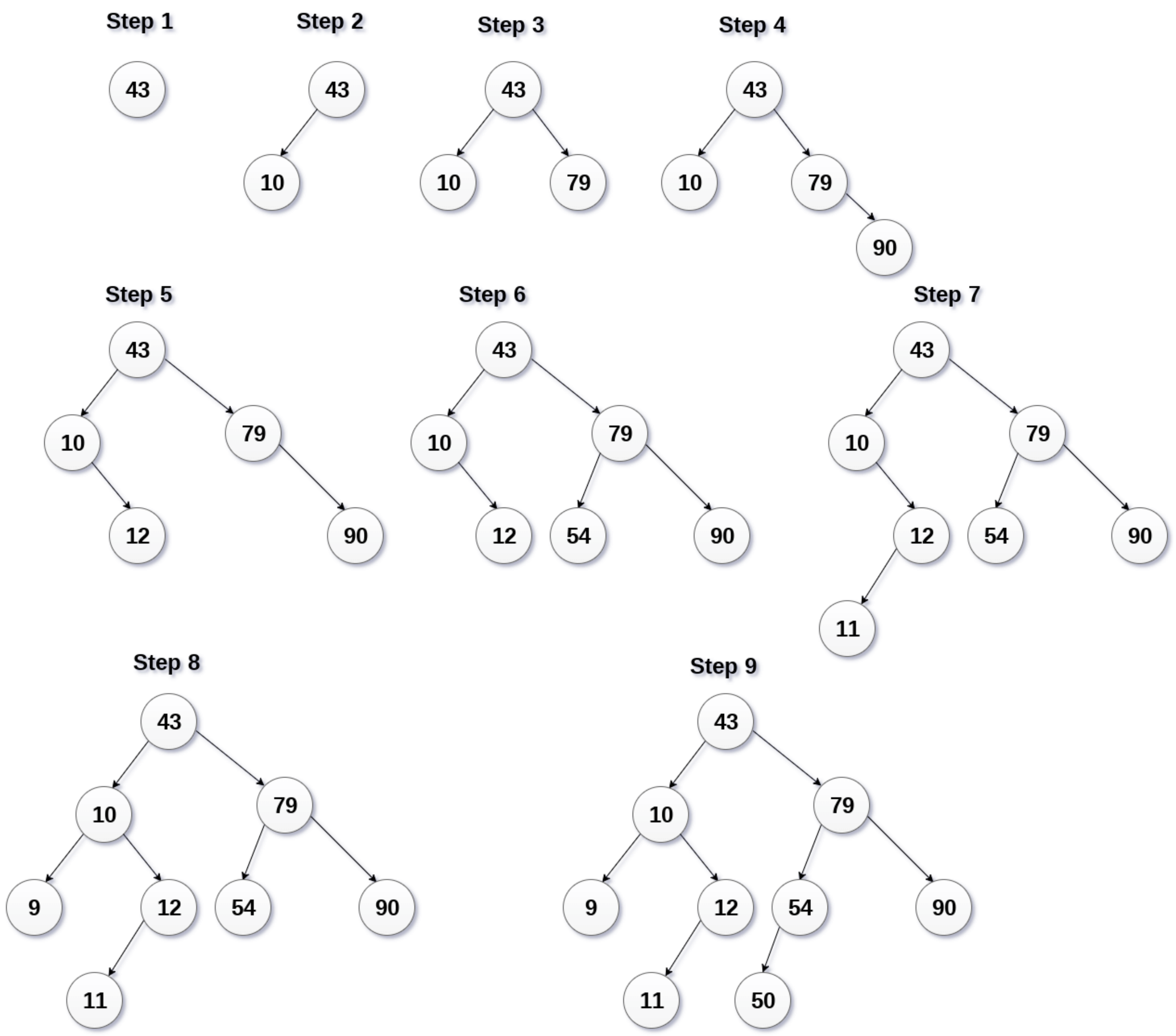
1. Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
2. The binary search tree is considered as an efficient data structure in compare to arrays and linked lists. In the searching process, it removes half the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

### Q. Create the binary search tree using the following data elements.

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

1. Insert 43 into the tree as the root of the tree.
2. Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
3. 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.



**Binary search Tree Creation**

## 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. |

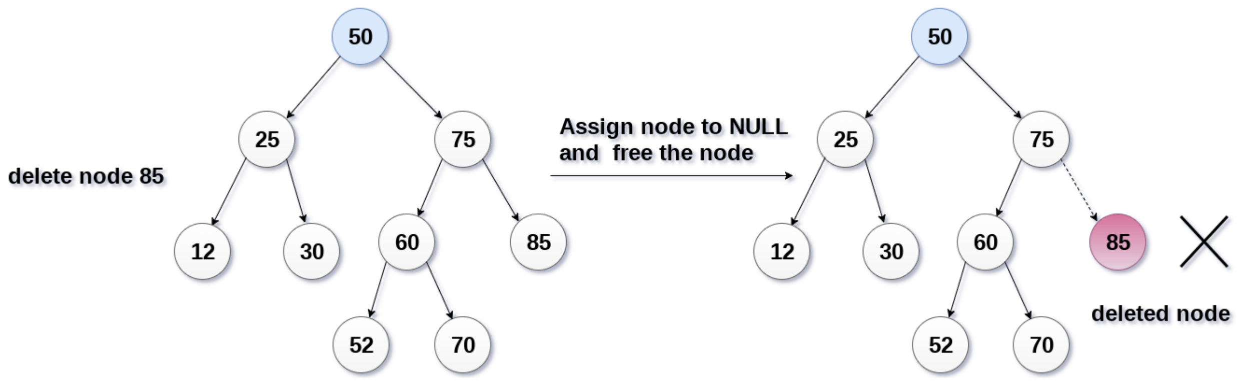
## Deletion

Delete function is used to delete the specified node from a binary search tree. However, we must delete a node from a binary search tree in such a way, that the property of a binary search tree doesn't violate. There are three situations of deleting a node from a binary search tree.

### The node to be deleted is a leaf node

It is the simplest case, in this case, replace the leaf node with the NULL and simply free the allocated space.

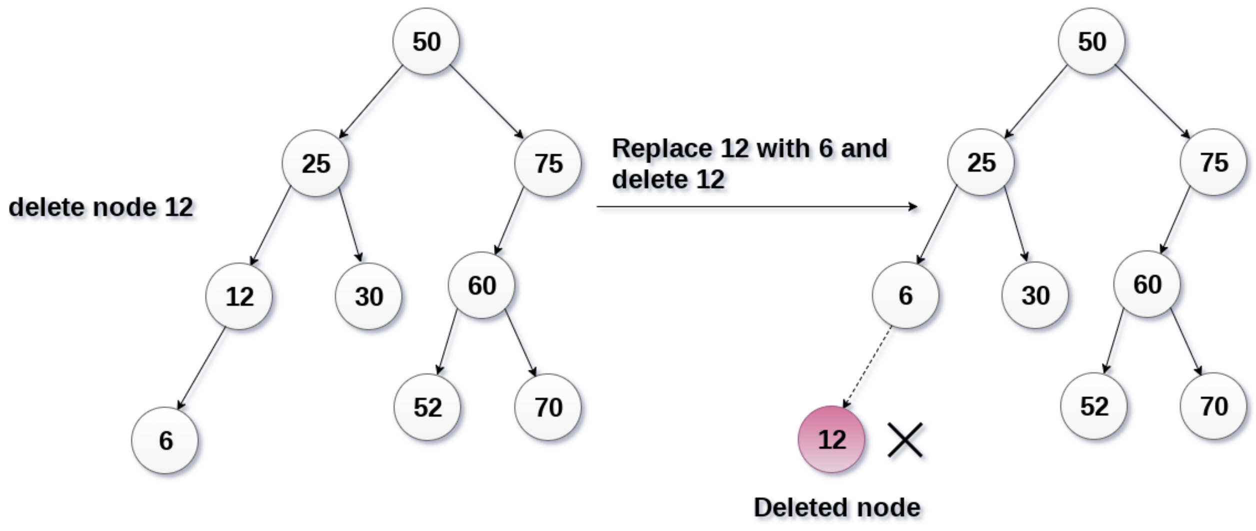
In the following image, we are deleting node 85, since the node is a leaf node, therefore the node will be replaced with NULL, and allocated space will be freed.



### The node to be deleted has only one child.

In this case, replace the node with its child and delete the child node, which now contains the value which is to be deleted. Simply replace it with the NULL and free the allocated space.

In the following image, the node 12 is to be deleted. It has only one child. The node will be replaced with its child node and the replaced node 12 (which is now leaf node) will simply be deleted.



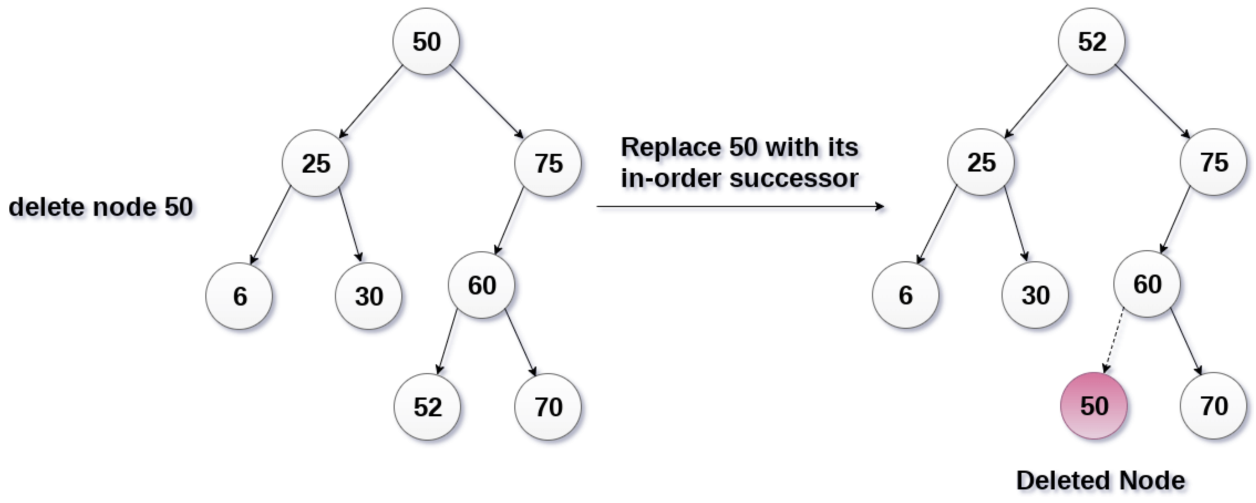
## The node to be deleted has two children.

It is a bit complexed case compare to the other two cases. However, the node which is to be deleted, is replaced with its in-order successor or predecessor recursively until the node value (to be deleted) is placed on the leaf of the tree. After the procedure, replace the node with NULL and free the allocated space.

In the following image, the node 50 is to be deleted which is the root node of the tree. The in-order traversal of the tree is given below.

6, 25, 30, 50, 52, 60, 70, 75.

replace 50 with its in-order successor 52. Now, 50 will be moved to the leaf of the tree, which will simply be deleted.


delete node 50
50
25
75
6
30
60
52
70
Replace 50 with its in-order successor
52
25
75
6
30
60
50
70
Deleted Node