

Binary Search Tree

In a binary tree, every node can have a maximum of two children but there is no need to maintain the order of nodes basing on their values. In a binary tree, the elements are arranged in the order they arrive at the tree from top to bottom and left to right.

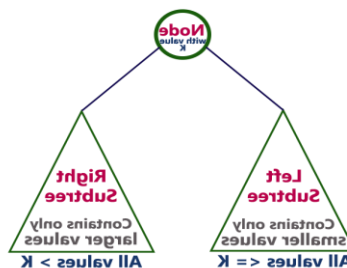
A binary tree has the following time complexities...

1. **Search Operation** - $O(n)$
2. **Insertion Operation** - $O(1)$
3. **Deletion Operation** - $O(n)$

To enhance the performance of binary tree, we use a special type of binary tree known as **Binary Search Tree**. Binary search tree mainly focuses on the search operation in a binary tree. Binary search tree can be defined as follows...

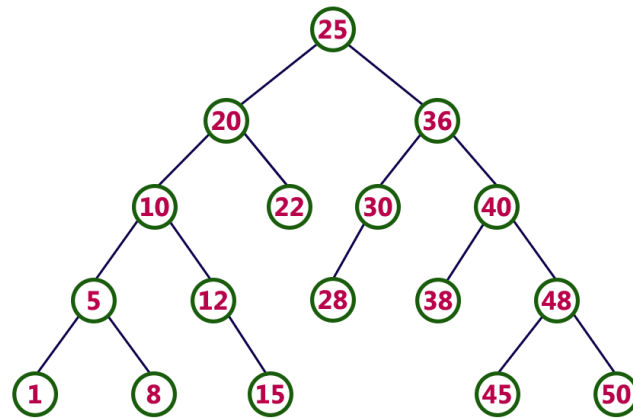
Binary Search Tree is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree.

In a binary search tree, all the nodes in the left subtree of any node contains smaller values and all the nodes in the right subtree of any node contains larger values as shown in the following figure...



Example

The following tree is a Binary Search Tree. In this tree, left subtree of every node contains nodes with smaller values and right subtree of every node contains larger values.



Every binary search tree is a binary tree but every binary tree need not to be binary search tree.

Operations on a Binary Search Tree

The following operations are performed on a binary search tree...

1. Search
2. Insertion
3. Deletion

Search Operation in BST

In a binary search tree, the search operation is performed with **$O(\log n)$** time complexity. The search operation is performed as follows...

- **Step 1** - Read the search element from the user.
- **Step 2** - Compare the search element with the value of root node in the tree.
- **Step 3** - If both are matched, then display "Given node is found!!!" and terminate the function

- **Step 4** - If both are not matched, then check whether search element is smaller or larger than that node value.
- **Step 5** - If search element is smaller, then continue the search process in left subtree.
- **Step 6** - If search element is larger, then continue the search process in right subtree.
- **Step 7** - Repeat the same until we find the exact element or until the search element is compared with the leaf node
- **Step 8** - If we reach to the node having the value equal to the search value then display "Element is found" and terminate the function.
- **Step 9** - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Insertion Operation in BST

In a binary search tree, the insertion operation is performed with **$O(\log n)$** time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

- **Step 1** - Create a newNode with given value and set its **left** and **right** to **NULL**.
- **Step 2** - Check whether tree is Empty.
- **Step 3** - If the tree is **Empty**, then set **root** to **newNode**.
- **Step 4** - If the tree is **Not Empty**, then check whether the value of newNode is **smaller** or **larger** than the node (here it is root node).
- **Step 5** - If newNode is **smaller** than **or equal** to the node then move to its **left** child. If newNode is **larger** than the node then move to its **right** child.
- **Step 6** - Repeat the above steps until we reach to the **leaf** node (i.e., reaches to **NULL**).
- **Step 7** - After reaching the leaf node, insert the newNode as **left child** if the newNode is **smaller or equal** to that leaf node or else insert it as **right child**.

Deletion Operation in BST

In a binary search tree, the deletion operation is performed with **$O(\log n)$** time complexity. Deleting a node from Binary search tree includes following three cases...

- **Case 1:** Deleting a Leaf node (A node with no children)
- **Case 2:** Deleting a node with one child
- **Case 3:** Deleting a node with two children

Case 1: Deleting a leaf node

We use the following steps to delete a leaf node from BST...

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** Delete the node using **free** function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

We use the following steps to delete a node with one child from BST...

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** If it has only one child then create a link between its parent node and child node.
- **Step 3 -** Delete the node using **free** function and terminate the function.

Case 3: Deleting a node with two children

We use the following steps to delete a node with two children from BST...

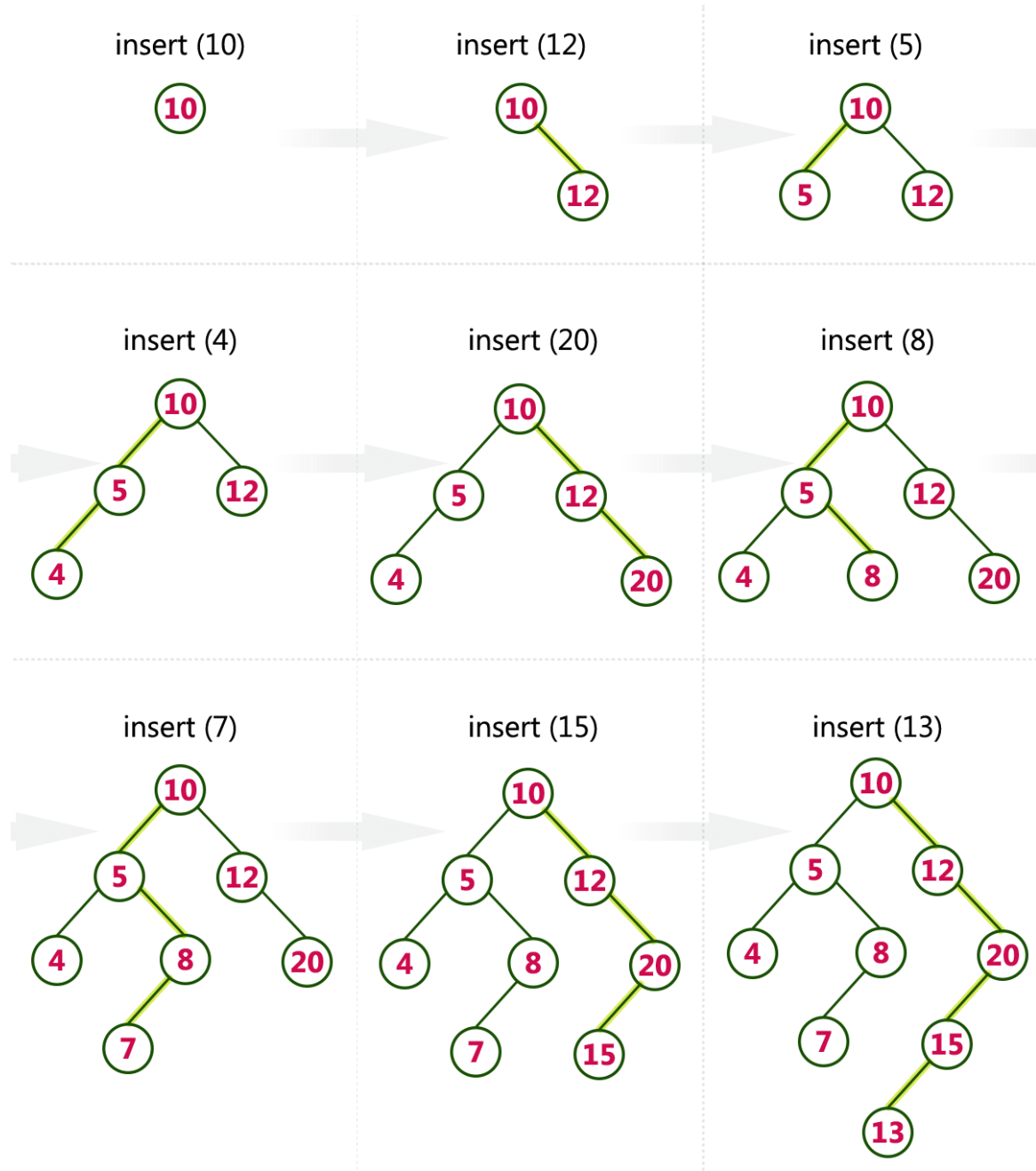
- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** If it has two children, then find the **largest** node in its **left subtree** (OR) the **smallest** node in its **right subtree**.
- **Step 3 - Swap** both **deleting node** and node which is found in the above step.
- **Step 4 -** Then check whether deleting node came to **case 1** or **case 2** or else goto step 2
- **Step 5 -** If it comes to **case 1**, then delete using case 1 logic.
- **Step 6 -** If it comes to **case 2**, then delete using case 2 logic.
- **Step 7 -** Repeat the same process until the node is deleted from the tree.

Example

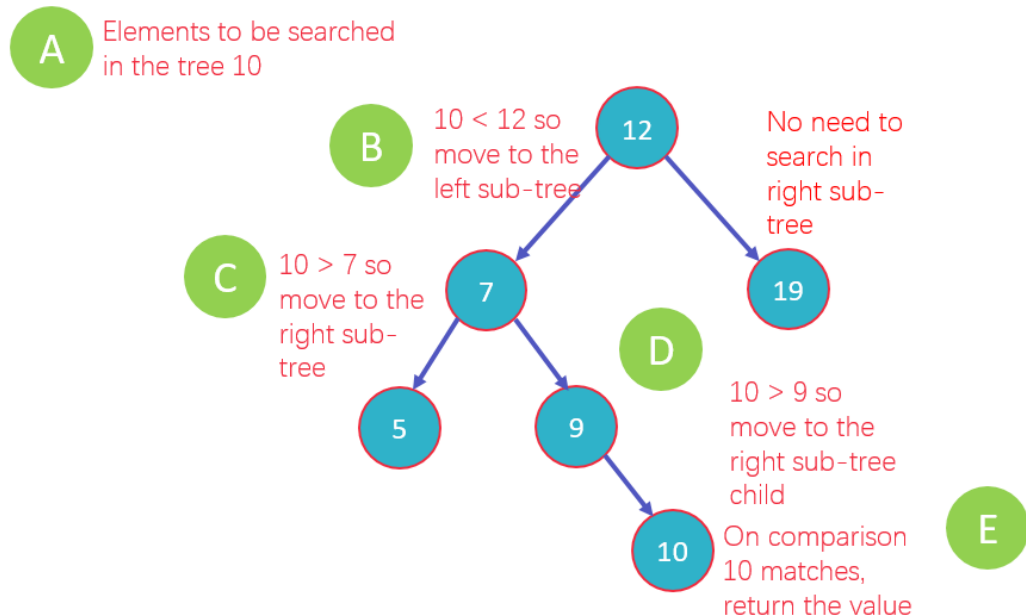
Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13

Above elements are inserted into a Binary Search Tree as follows...



Search Operation



- A. The element to be searched is 10
- B. Compare the element with the root node 12, $10 < 12$, hence you move to the left subtree. No need to analyze the right-subtree
- C. Now compare 10 with node 7, $10 > 7$, so move to the right-subtree
- D. Then compare 10 with the next node, which is 9, $10 > 9$, look in the right subtree child
- E. 10 matches with the value in the node, $10 = 10$, return the value to the user.

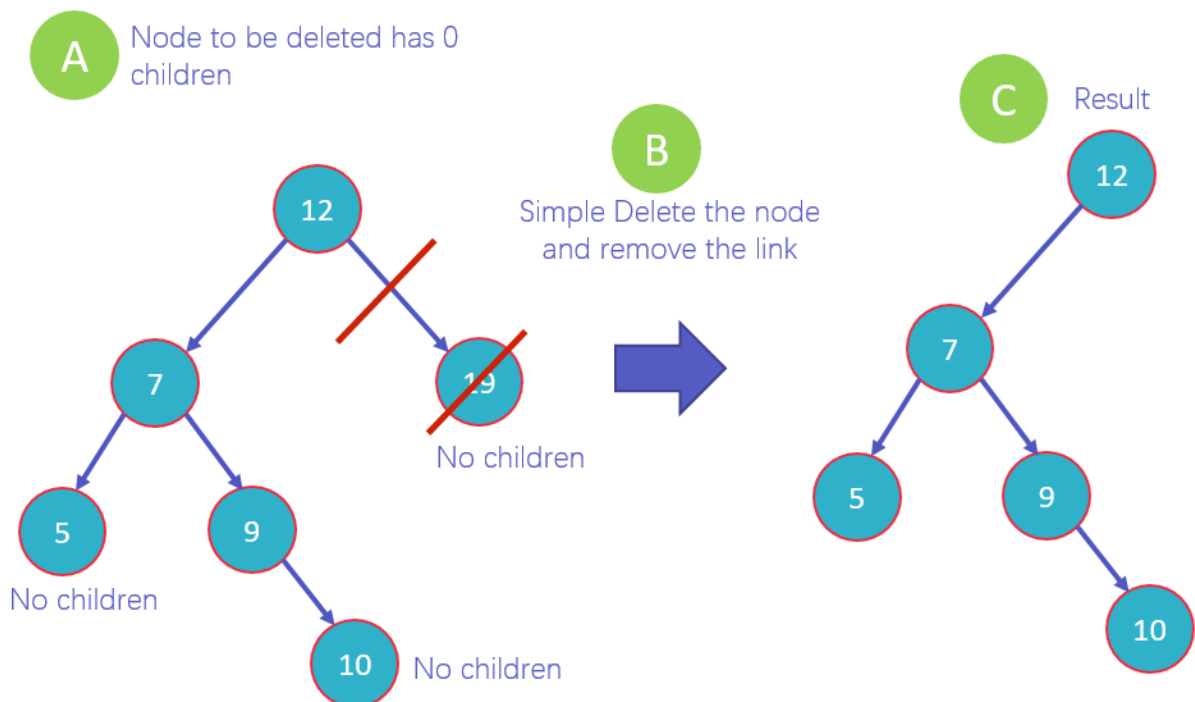
Delete Operations

Delete is the most advanced and complex among all other operations. There are multiple cases handled for deletion in the BST.

- Case 1- Node with zero children: this is the easiest situation, you just need to delete the node which has no further children on the right or left.
- Case 2 - Node with one child: once you delete the node, simply connect its child node with the parent node of the deleted value.

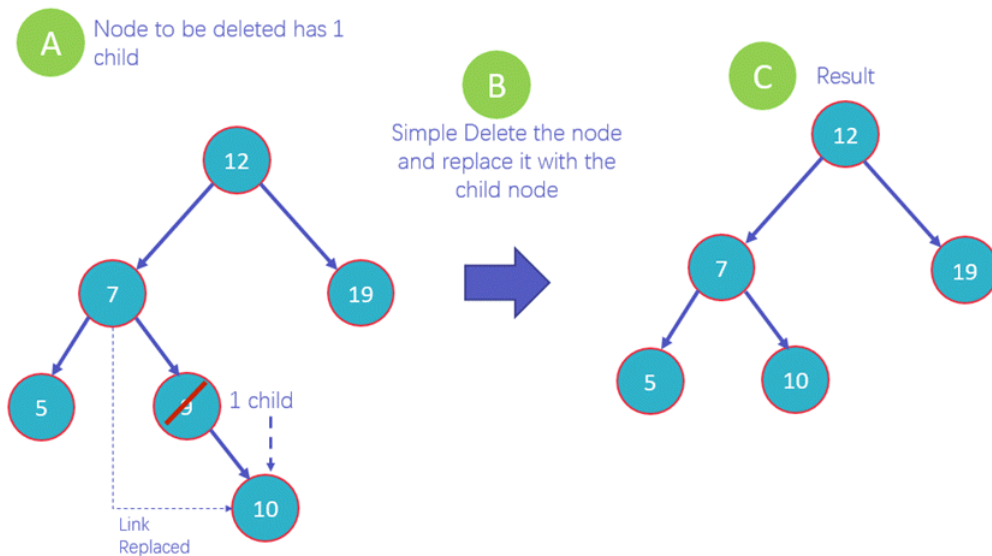
- Case 3 Node with two children: this is the most difficult situation, and it works on the following two rules
 - 3a - In Order Predecessor: you need to delete the node with two children and replace it with the largest value on the left-subtree of the deleted node
 - 3b - In Order Successor: you need to delete the node with two children and replace it with the largest value on the right-subtree of the deleted node

Delete Operation – Case 1



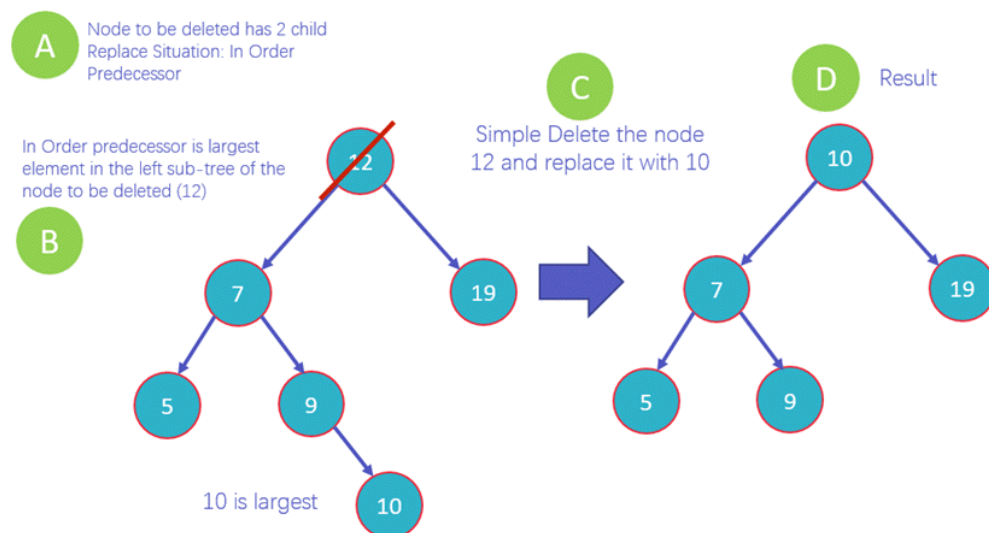
- A. This is the first case of deletion in which you delete a node that has no children. As you can see in the diagram that 19, 10 and 5 have no children. But we will delete 19.
- B. Delete the value 19 and remove the link from the node.
- C. View the new structure of the BST without 19

Delete Operation – Case 2



- This is the second case of deletion in which you delete a node that has 1 child, as you can see in the diagram that 9 has one child.
- Delete the node 9 and replace it with its child 10 and add a link from 7 to 10
- View the new structure of the BST without 9

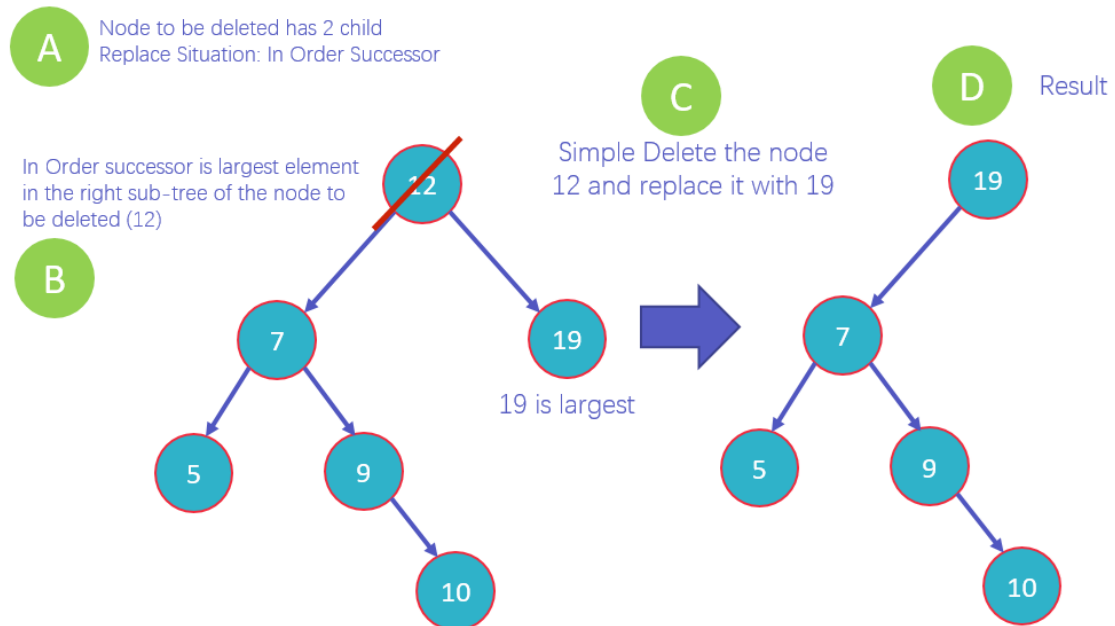
Delete Operation – Case 3 (a)



- Here you will be deleting the node 12 that has two children
- The deletion of the node will occur based upon the in order predecessor rule, which means that the largest element on the left subtree of 12 will replace it.
- Delete the node 12 and replace it with 10 as it is the largest value on the left subtree

D. View the new structure of the BST after deleting 12

Delete Operation – Case 3 (b)



- A. 1 Delete a node 12 that has two children
- B. 2 The deletion of the node will occur based upon the In Order Successor rule, which means that the largest element on the right subtree of 12 will replace it
- C. 3 Delete the node 12 and replace it with 19 as it is the largest value on the right subtree
- D. 4 View the new structure of the BST after deleting 12