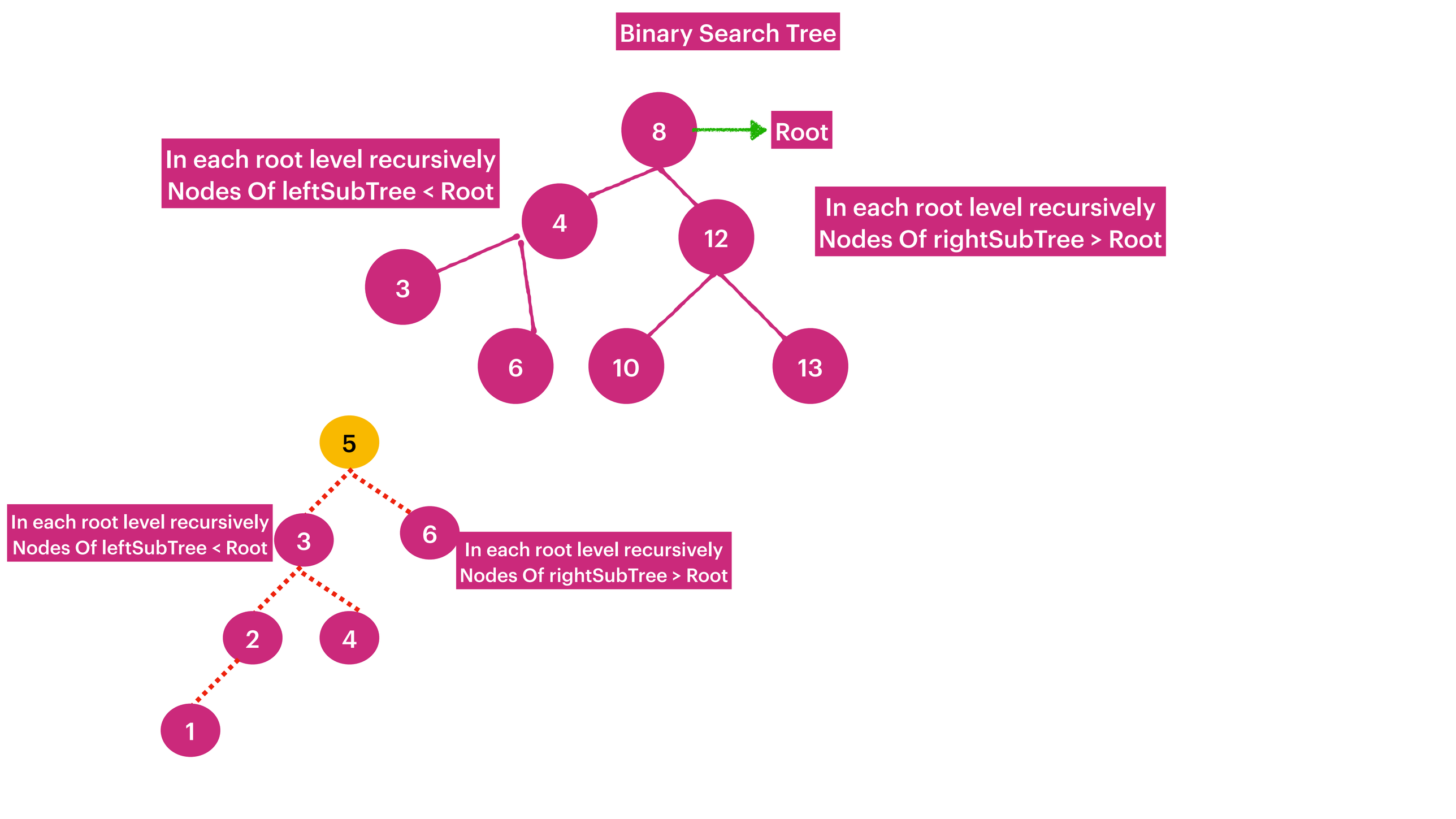
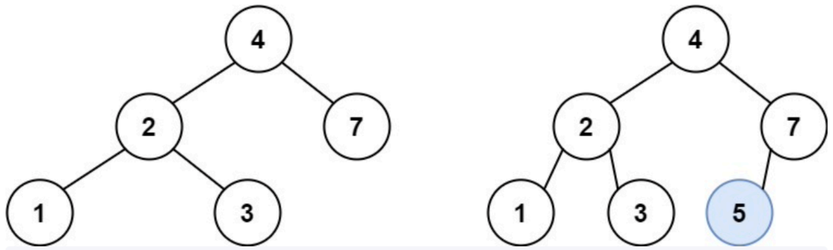

Binary Search Tree
8
Root
In each root level recursively
Nodes Of leftSubTree < Root
In each root level recursively
Nodes Of rightSubTree > Root
4
12
3
6
10
13
5
In each root level recursively
Nodes Of leftSubTree < Root
3
6
In each root level recursively
Nodes Of rightSubTree > Root
2
4
1

701. Insert into a Binary Search Tree

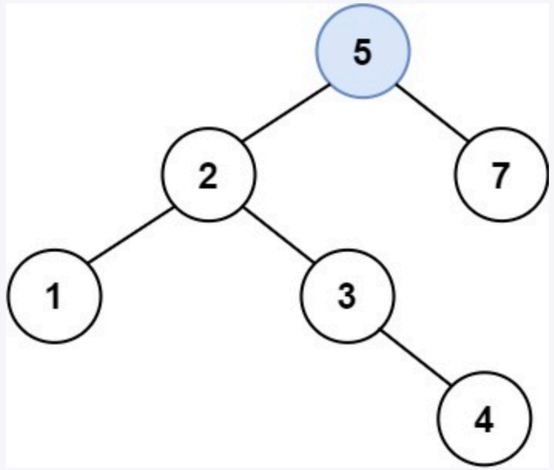
You are given the `root` node of a binary search tree (BST) and a `value` to insert into the tree. Return *the root node of the BST after the insertion*. It is **guaranteed** that the new value does not exist in the original BST.

Notice that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return **any of them**.

Example 1:




```
Input: root = [4,2,7,1,3], val = 5
Output: [4,2,7,1,3,5]
Explanation: Another accepted tree is:
```



Example 2:

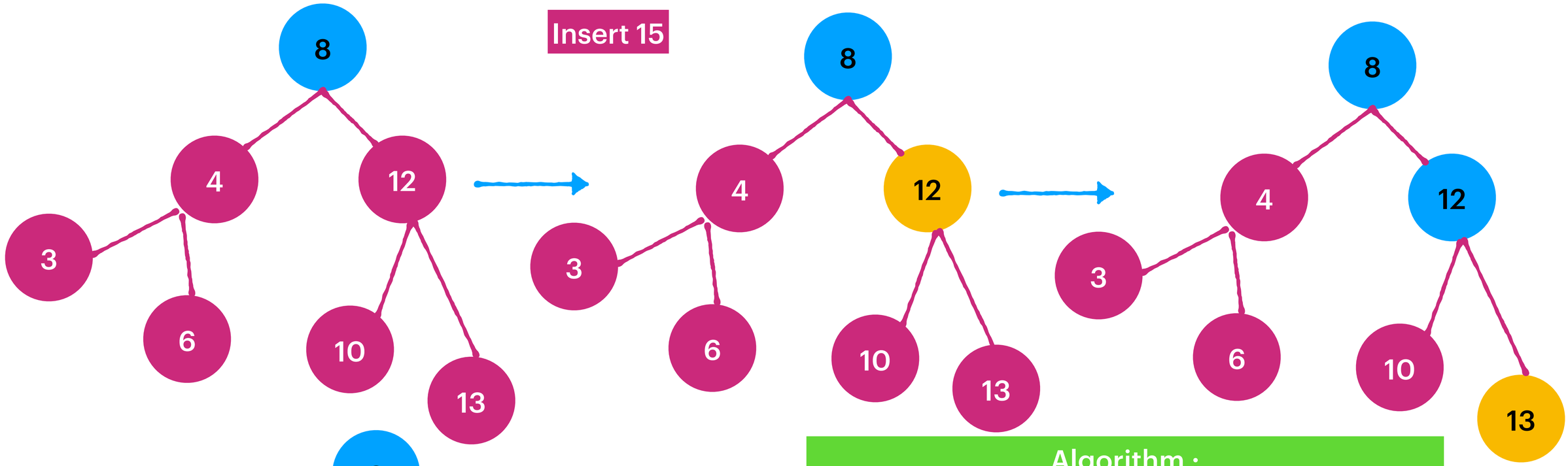
```
Input: root = [40,20,60,10,30,50,70], val = 25
Output: [40,20,60,10,30,50,70,null,null,25]
```

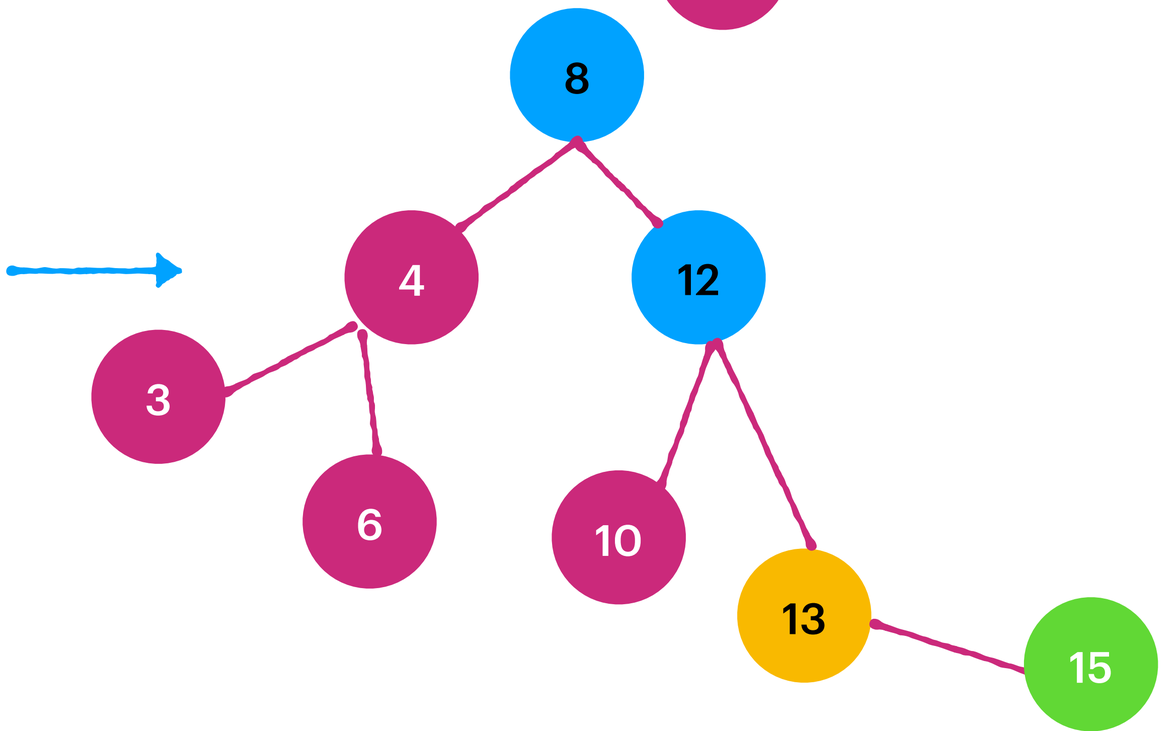
Example 3:

```
Input: root = [4,2,7,1,3,null,null,null,null,null,null], val = 5
Output: [4,2,7,1,3,5]
```

Constraints:

- The number of nodes in the tree will be in the range `[0, 10^4]`.
- `-10^8 <= Node.val <= 10^8`
- All the values `Node.val` are **unique**.
- `-10^8 <= val <= 10^8`
- It's **guaranteed** that `val` does not exist in the original BST.





Algorithm :

If the current Element > root then move right

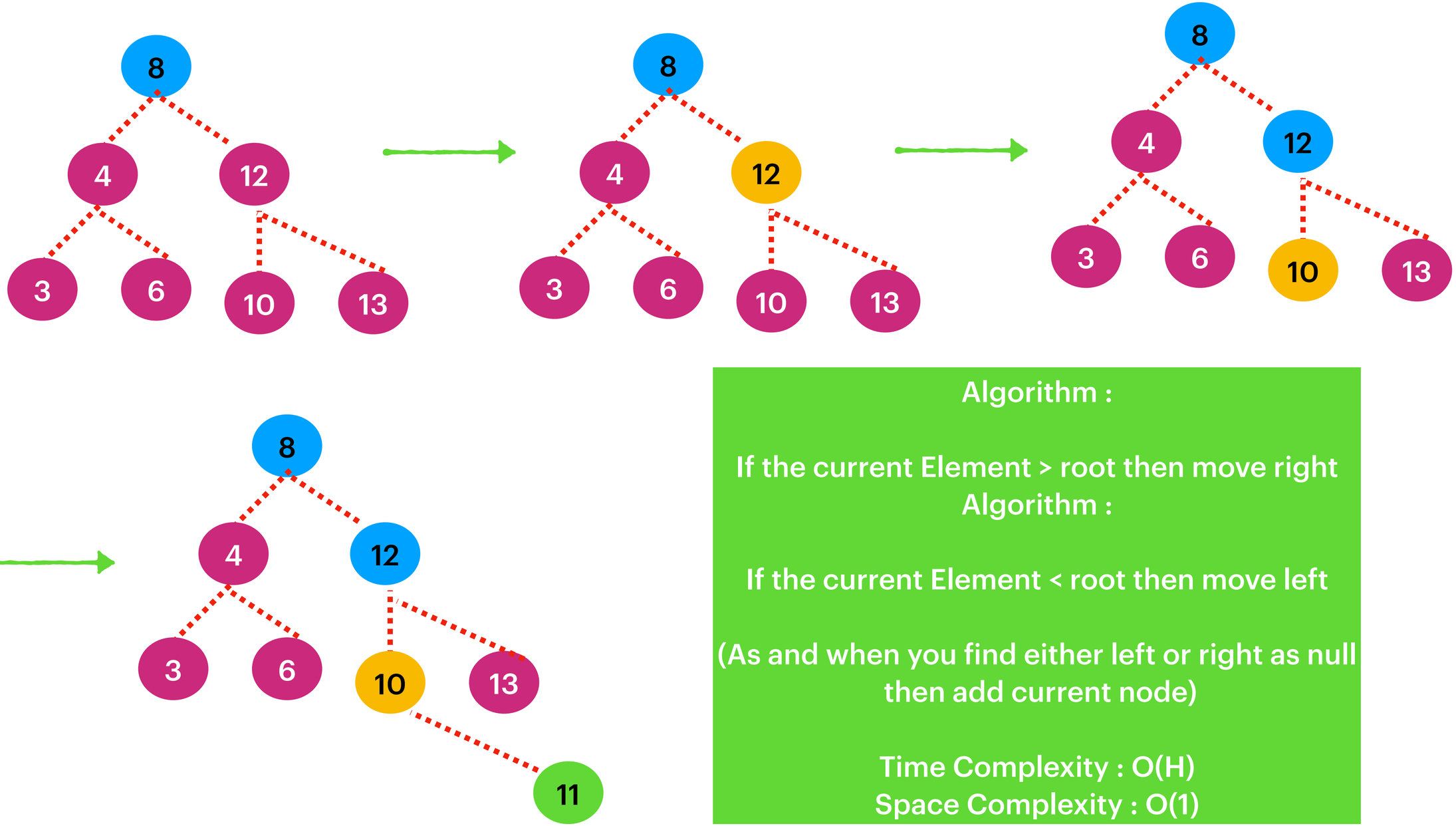
Algorithm :

If the current Element < root then move left

(As and when you find either left or right as null then add current node)

Time Complexity : O(H)
Space Complexity : O(1)

Insert 11



Algorithm :

If the current Element > root then move right

Algorithm :

If the current Element < root then move left

(As and when you find either left or right as null then add current node)

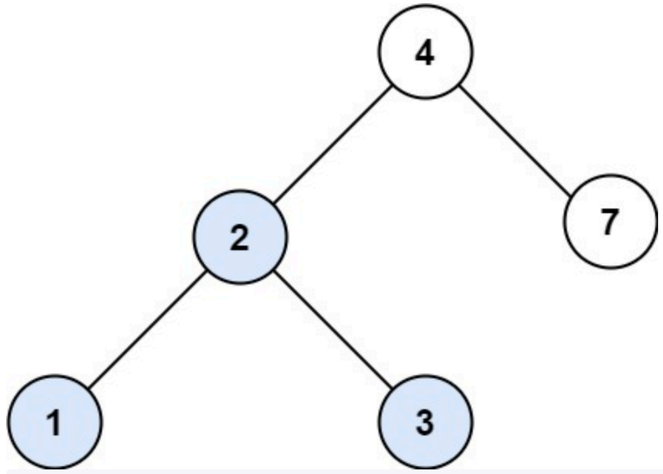
Time Complexity : O(H)
Space Complexity : O(1)

700. Search in a Binary Search Tree

You are given the `root` of a binary search tree (BST) and an integer `val`.

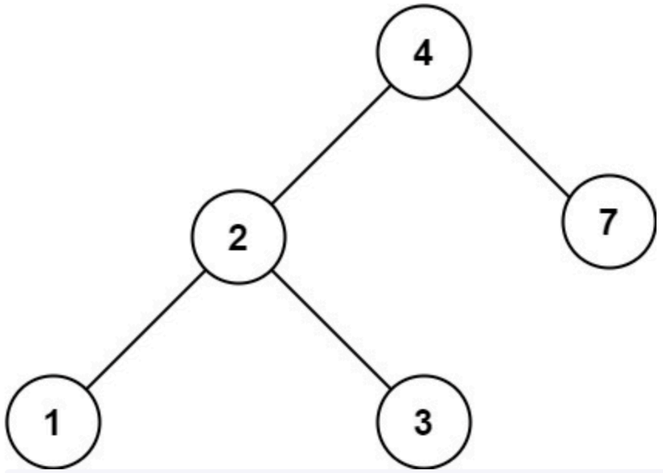
Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.

Example 1:




```
Input: root = [4,2,7,1,3], val = 2
Output: [2,1,3]
```

Example 2:

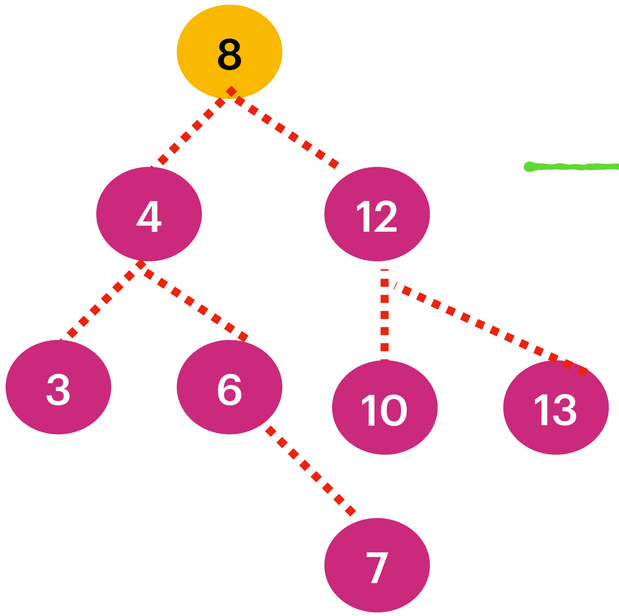


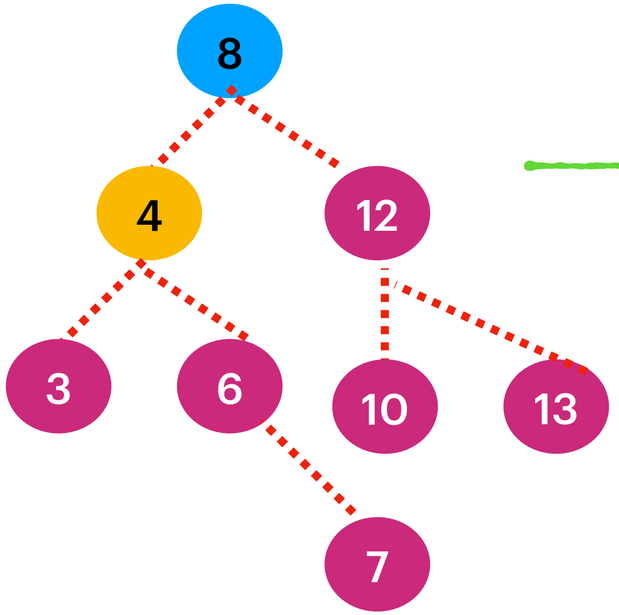

```
Input: root = [4,2,7,1,3], val = 5
Output: []
```

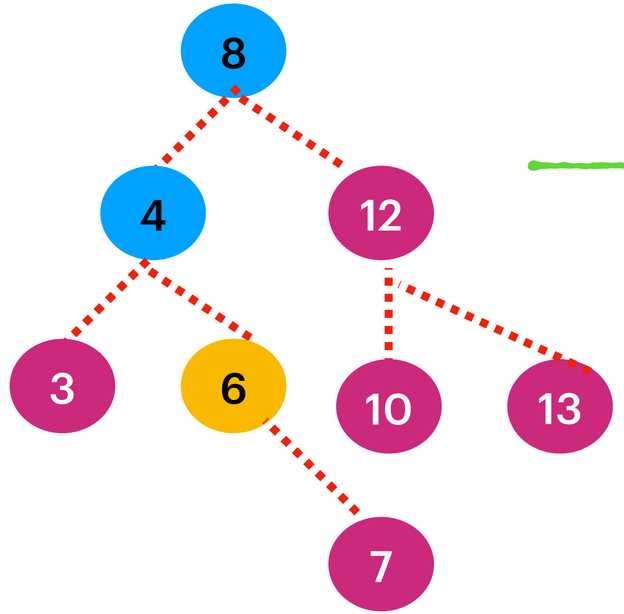
Constraints:

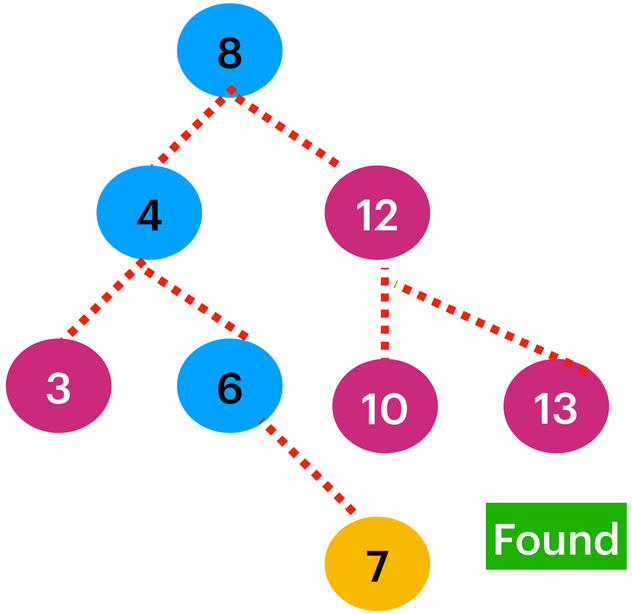
- The number of nodes in the tree is in the range `[1, 5000]`.
- `1 <= Node.val <= 10^7`
- `root` is a binary search tree.
- `1 <= val <= 10^7`

Search 7









Found

Algorithm :

If the current Element > root then move right

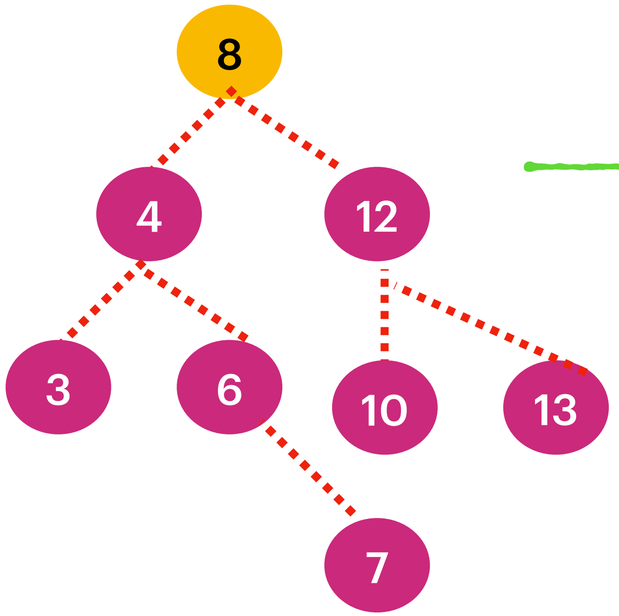
If the current Element < root then move left

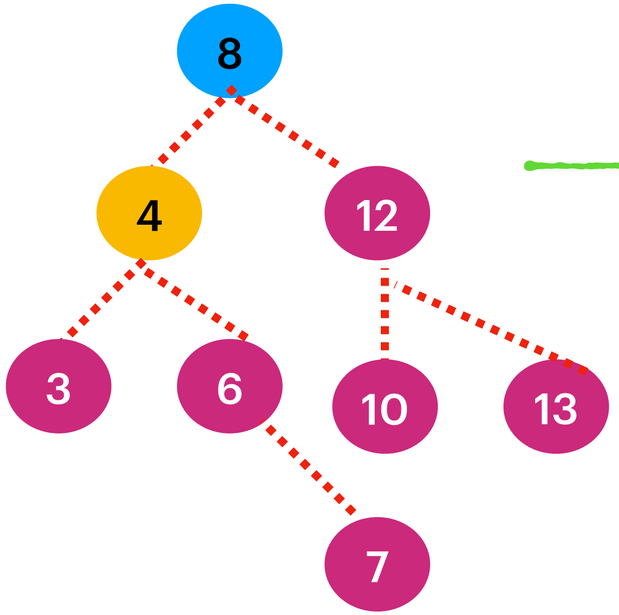
If the current root.val == searchVal return true.

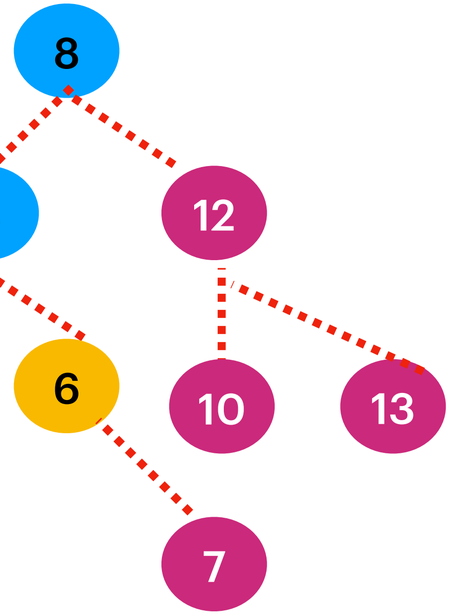
Time Complexity : O(H)

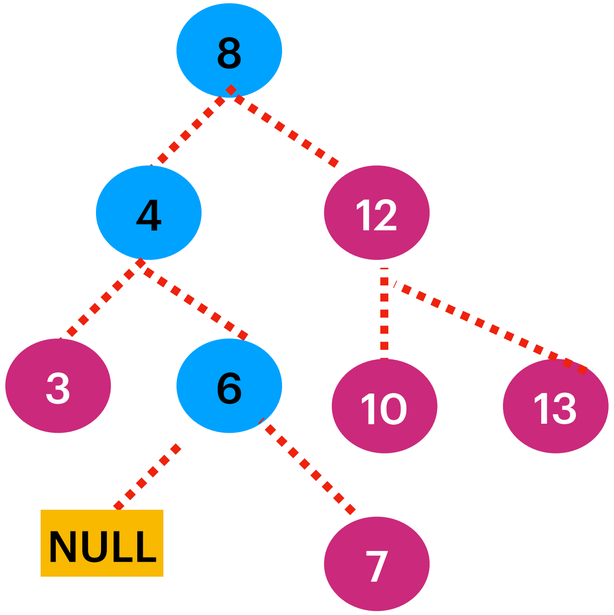
Space Complexity : O(1)

Search 5









As the 6 left is null.
Element Not Found.

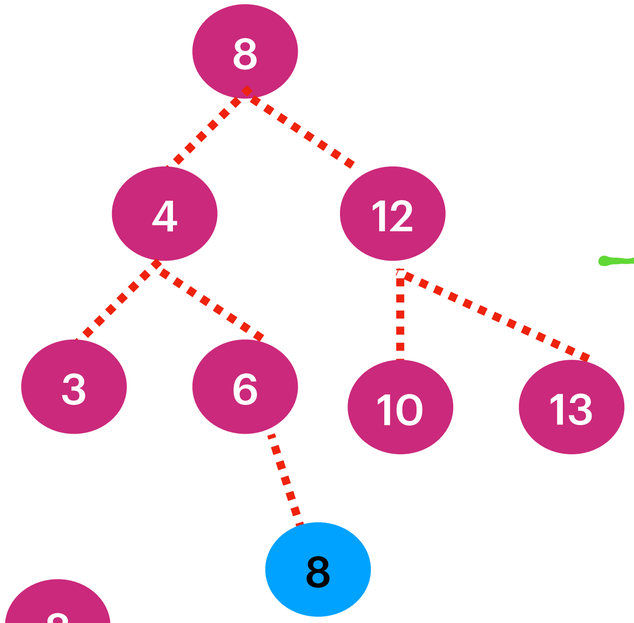
Algorithm :

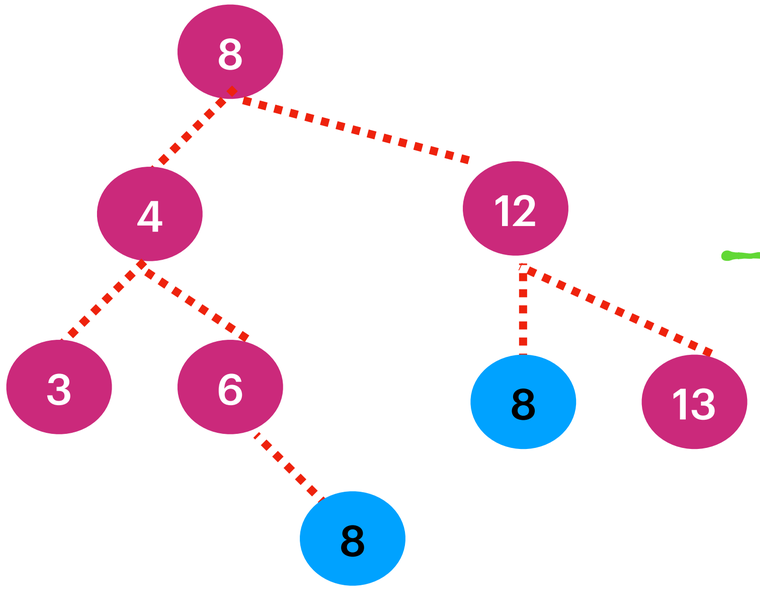
If the current Element > root then move right

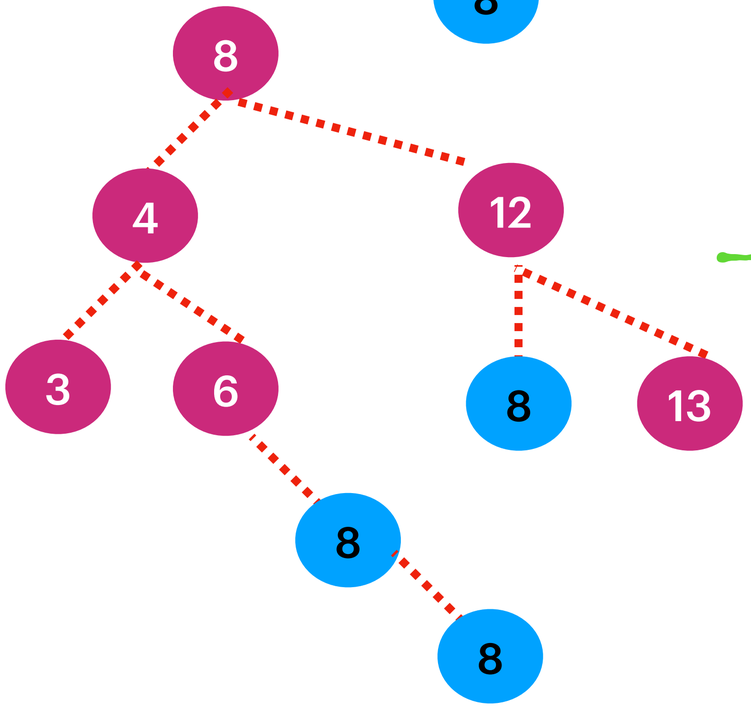
If the current Element < root then move left

If the current room.val == searchVal return true.
Time Complexity : O(H)
Space Complexity : O(1)

Why Con’t we allow duplicates in BinarySearchTree ?

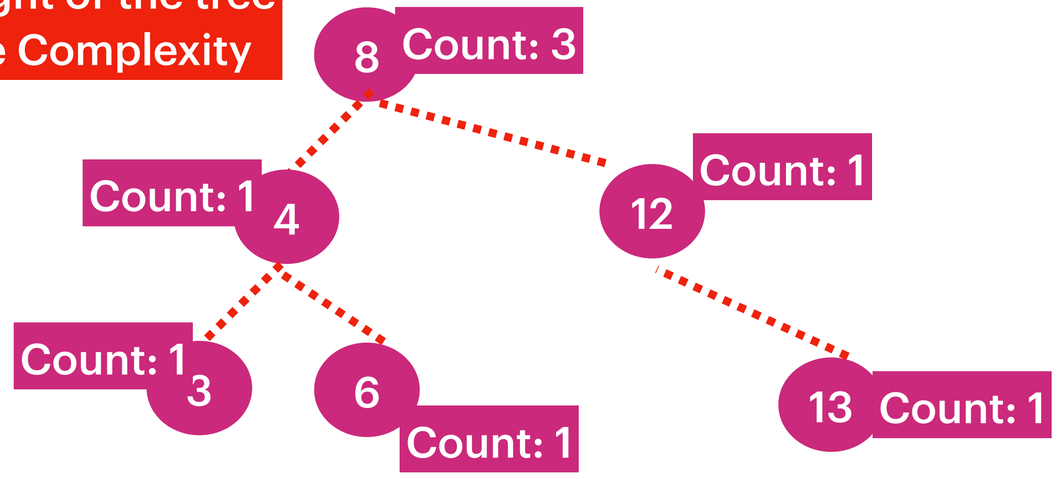






Allowing duplicates would
Increases the height of the tree
This effects Time Complexity

To Allow duplicate values have Counter:

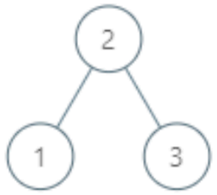


285. Inorder Successor in BST

Given the `root` of a binary search tree and a node `p` in it, return *the in-order successor of that node in the BST*. If the given node has no in-order successor in the tree, return `null`.

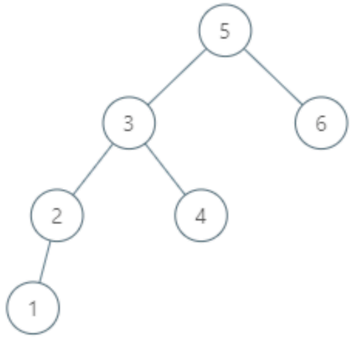
The successor of a node `p` is the node with the smallest key greater than `p.val`.

Example 1:




```
Input: root = [2,1,3], p = 1
Output: 2
Explanation: 1's in-order successor node is 2. Note that both p
and the return value is of TreeNode type.
```

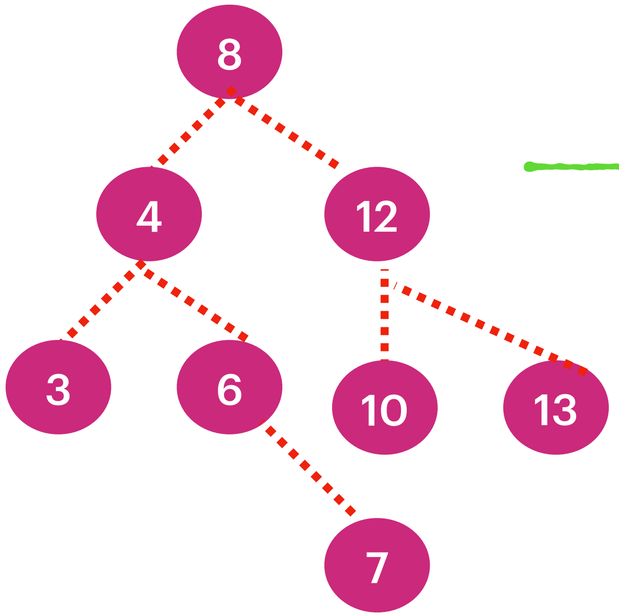
Example 2:




```
Input: root = [5,3,6,2,4,null,null,1], p = 6
Output: null
Explanation: There is no in-order successor of the current
node, so the answer is null.
```

Constraints:

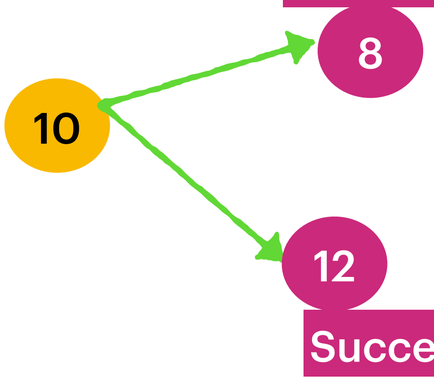
- The number of nodes in the tree is in the range `[1, 10^4]`.
- `-10^5 <= Node.val <= 10^5`
- All Nodes will have unique values.



InOrder Traversal [3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 12 -> 13]

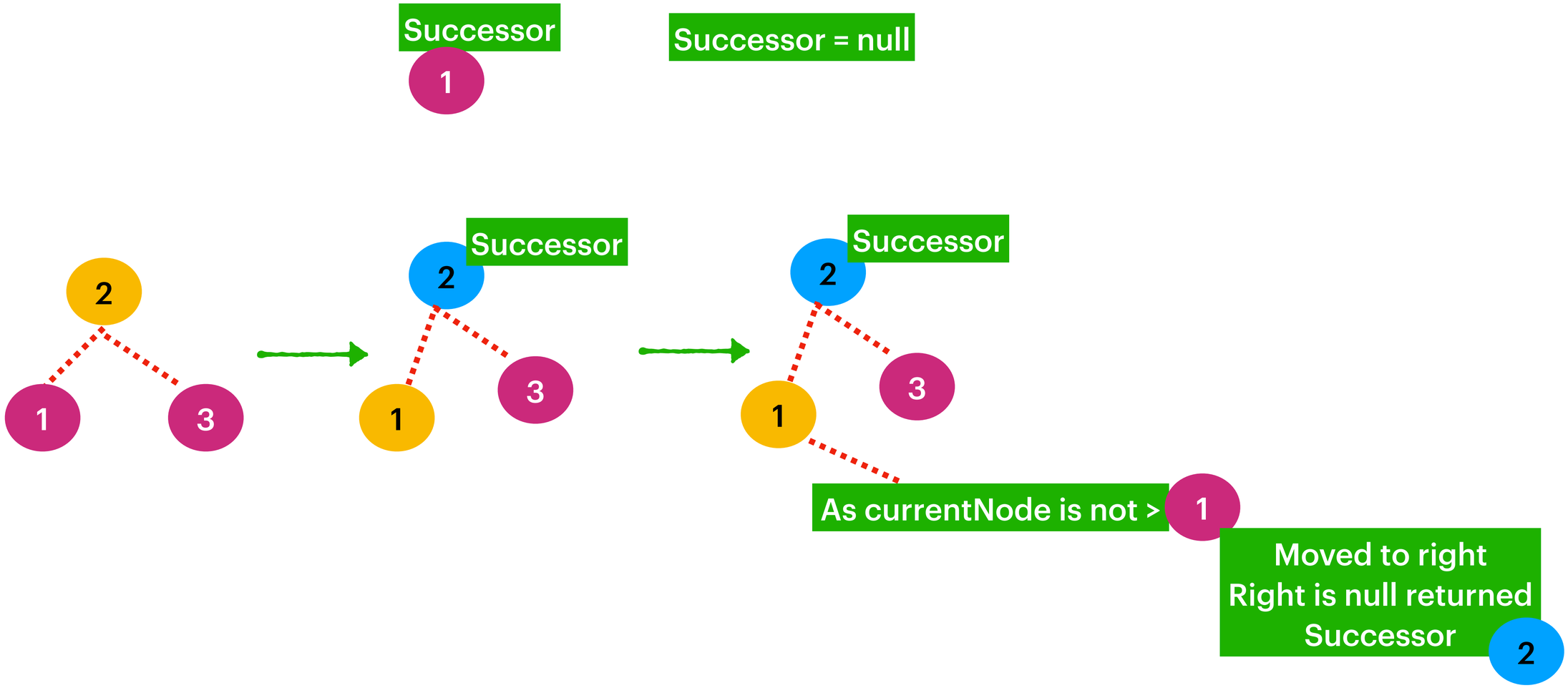
Returns elements in
Ascending order in a Binary Search Tree

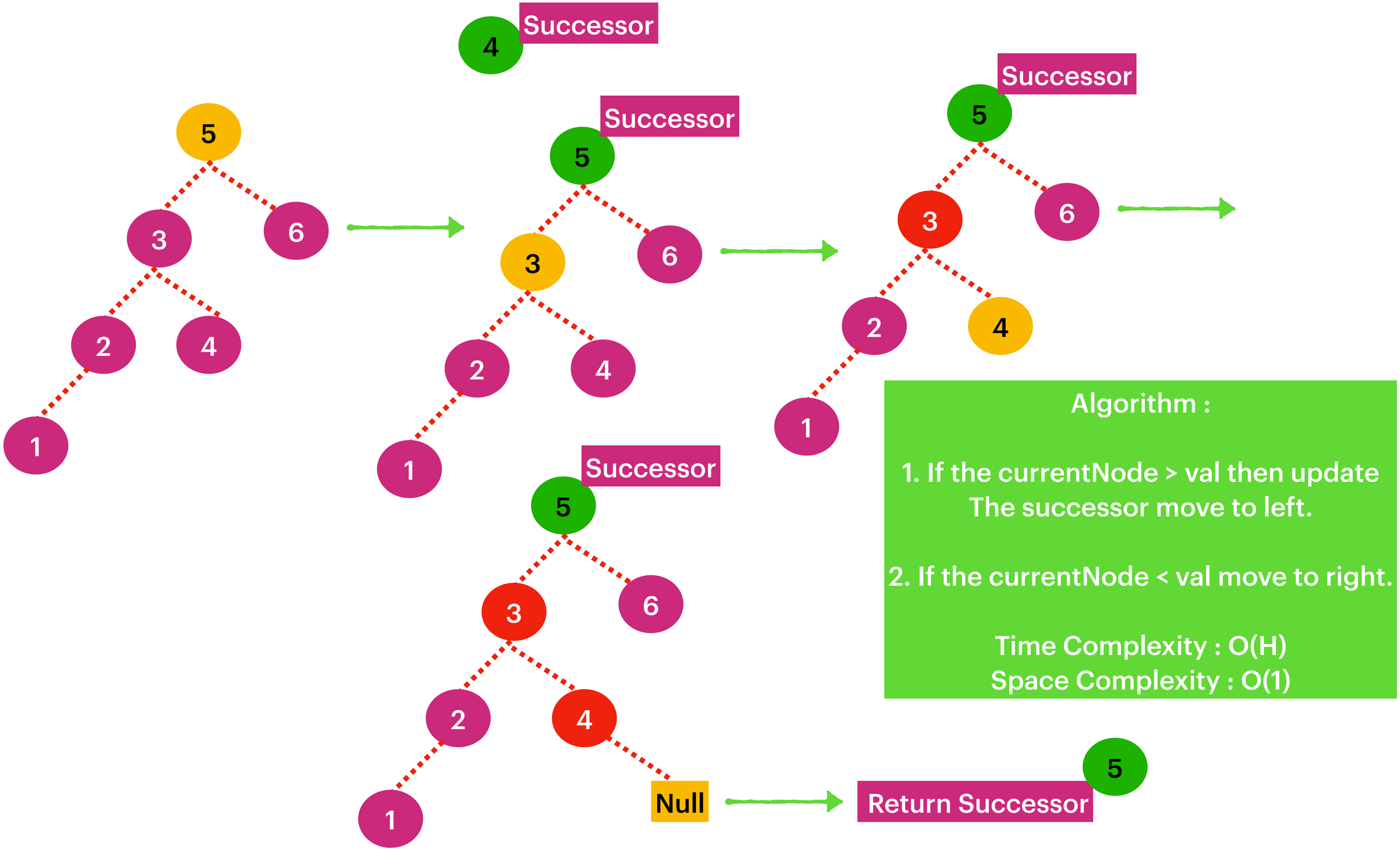
Predecessor : Immediate Smaller value.

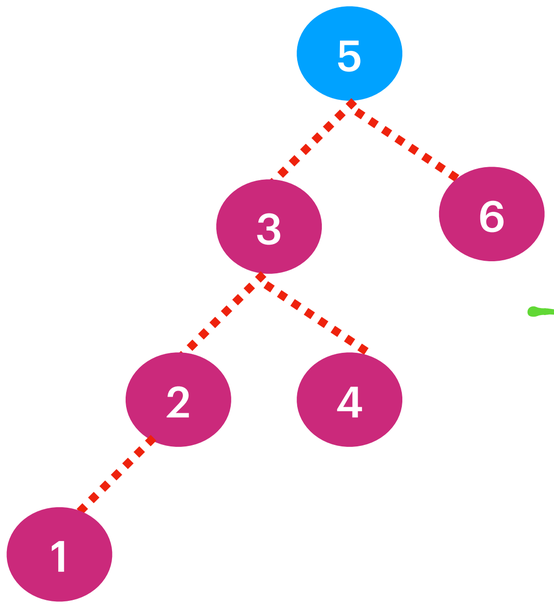


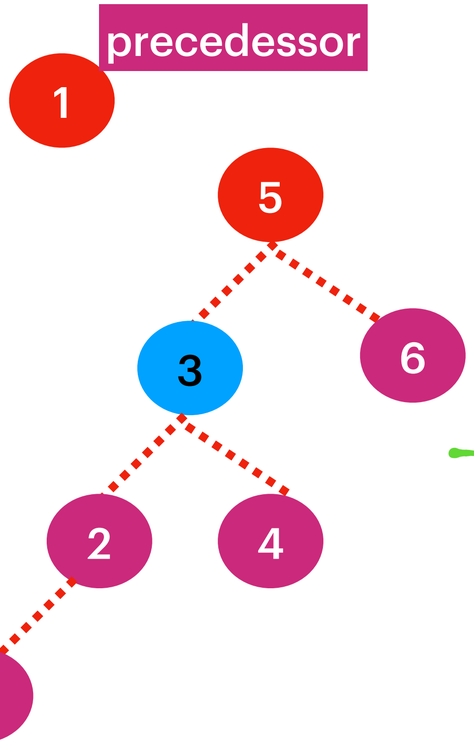
Successor : Immediate Greater value.

Time Complexity : O(N)
Space Complexity : O(N)

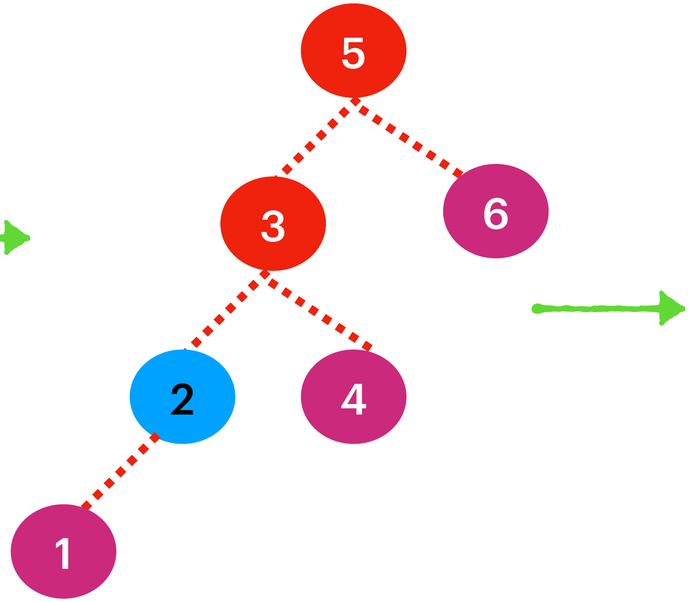

Successor
1
Successor = null
2
1
3
Successor
2
1
3
Successor
2
1
3
As currentNode is not > 1
Moved to right
Right is null returned
Successor
2


4
Successor
5
3
6
2
4
1
Successor
5
3
6
2
4
1
Successor
5
3
6
2
4
1
Successor
5
3
6
2
4
Null
Return Successor
5
Algorithm :
1. If the currentNode > val then update
The successor move to left.
2. If the currentNode < val move to right.
Time Complexity : O(H)
Space Complexity : O(1)





P = null



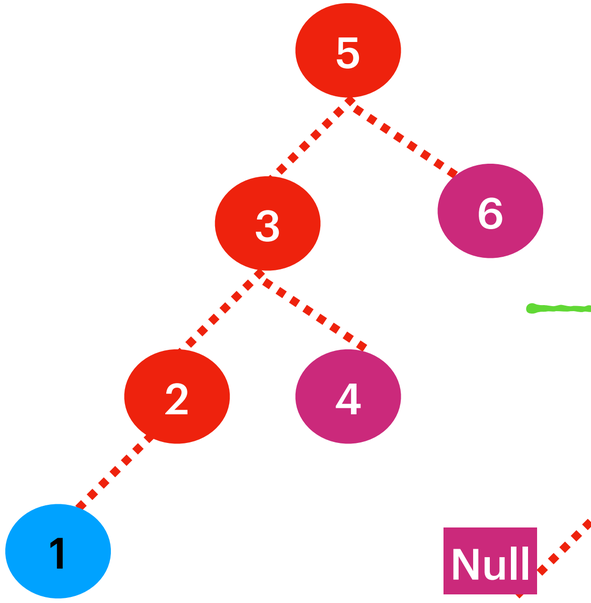

```
TreeNode p = null;
while(root != null){
if(root.val < val)
{
P = root;
root = root.right
}else
{
root = root.left
}
}
return p;
```

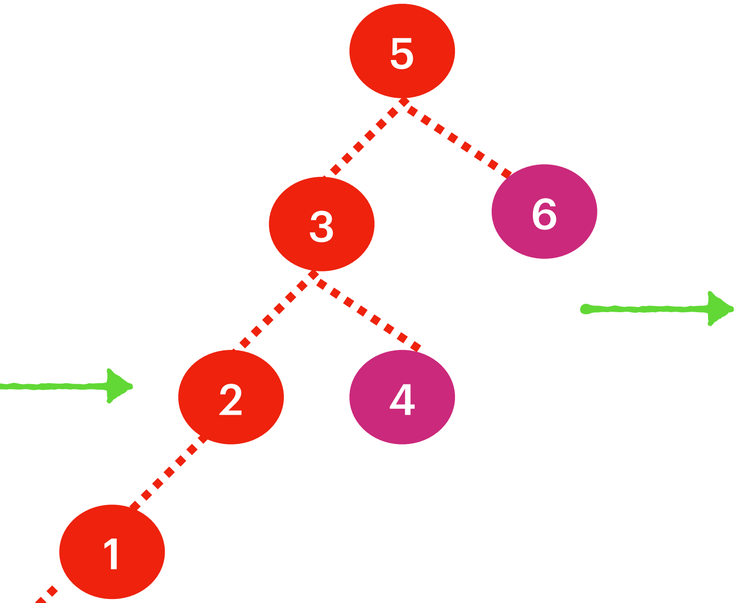
Algorithm :

1. If the currentNode < val then update The precedessor move to right.

2. If the currentNode > val move to left.

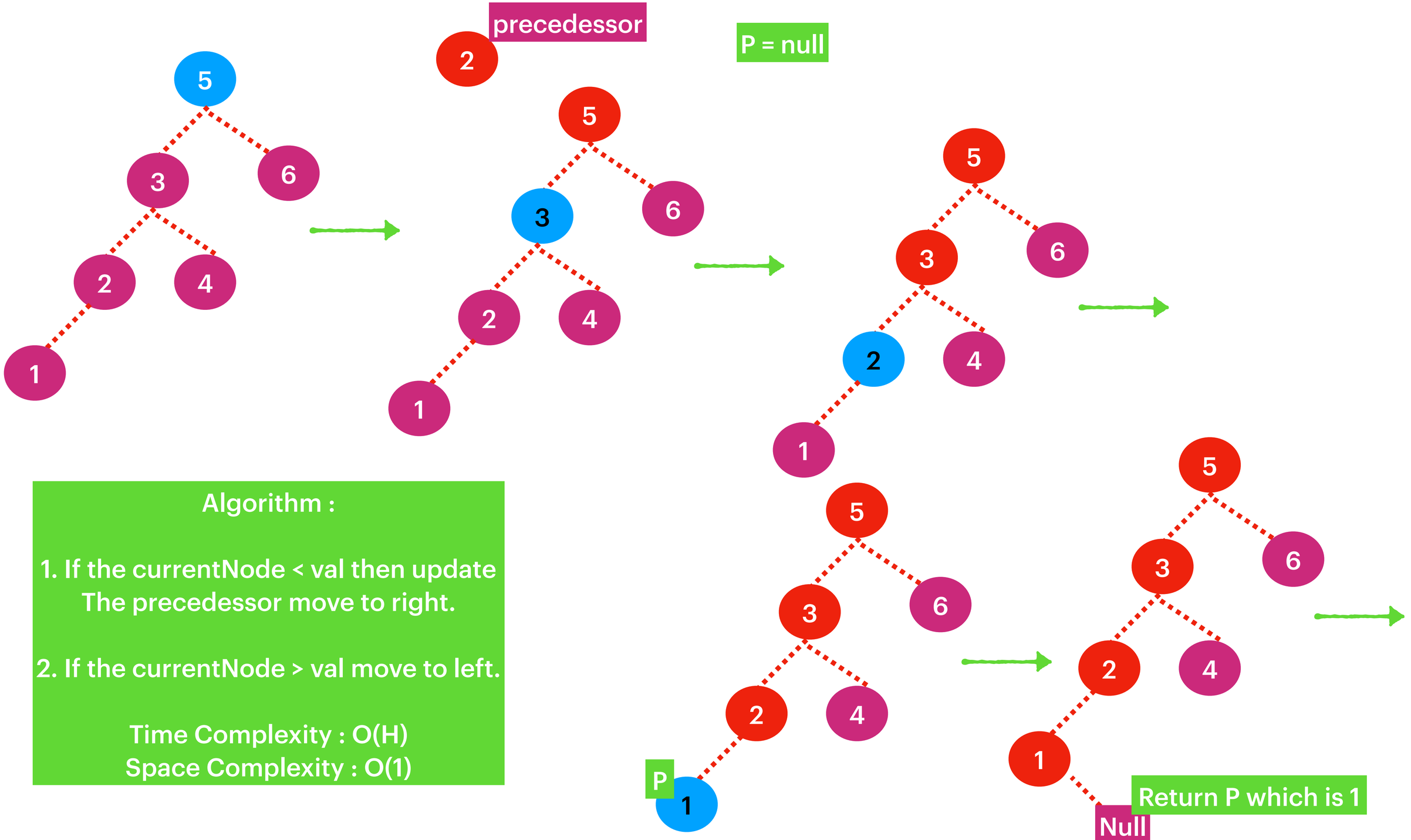
Time Complexity : O(H)
Space Complexity : O(1)





Null

Return P which is NULL


precedessor
P = null
Algorithm :
1. If the currentNode < val then update
The precedessor move to right.
2. If the currentNode > val move to left.
Time Complexity : O(H)
Space Complexity : O(1)
P
Return P which is 1
Null