Binary Search Trees

Data Structures & Algorithms

Yahnit Sirineni

Binary Search Trees!

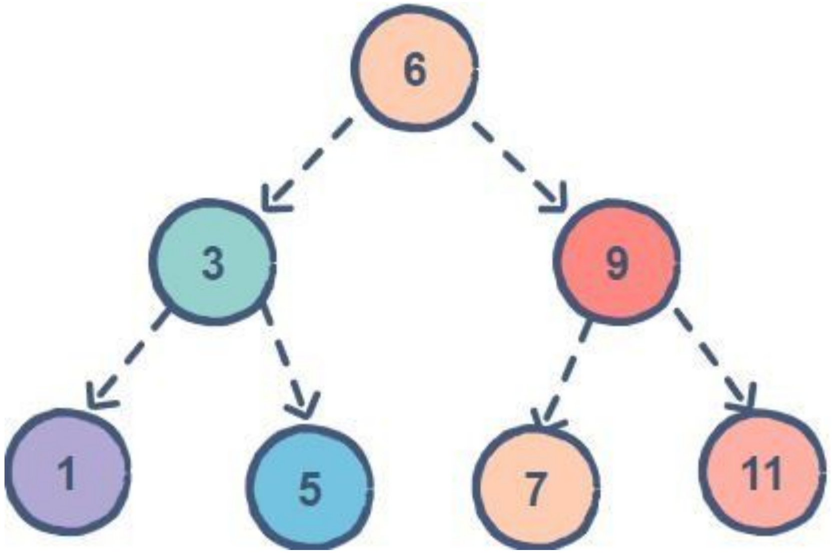
	Array	Linked list	Sorted Array	Hash table	BST
Search					
Insertion					
Deletion					

Binary Search Trees!

	Array	Linked list	Sorted Array	Hash table	BST
Search	O(n)	O(n)	O(logn)	O(1)	O(logn)
Insertion	O(1)	O(1)	O(n)	O(1)	O(logn)
Deletion	O(n)	O(n)	O(n)	O(1)	O(logn)

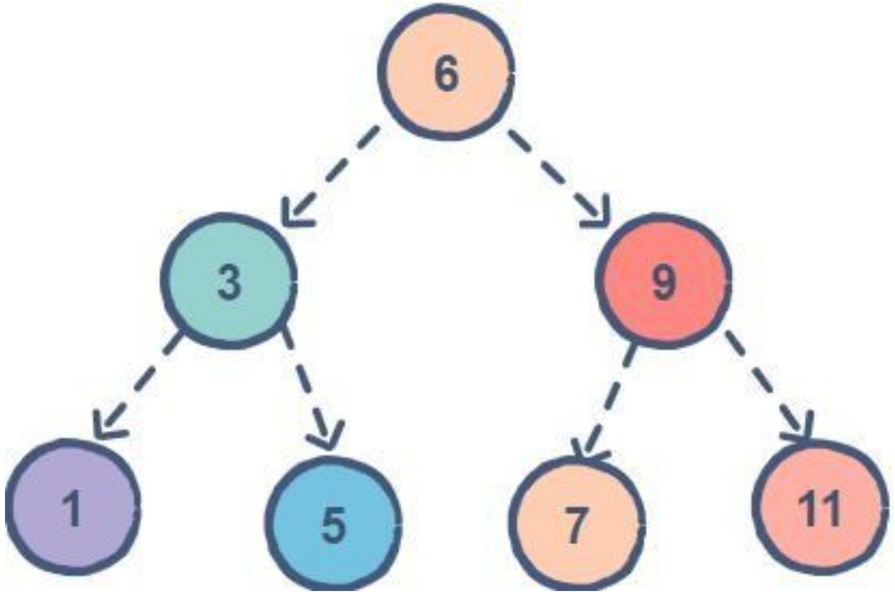
Why BST over hash table?

- Nodes are ordered!
- Range queries, Successors, kth smallest ele
- Better memory utilization
- Upper bound is O(logn) and not amortized.
- Easy to implement :P



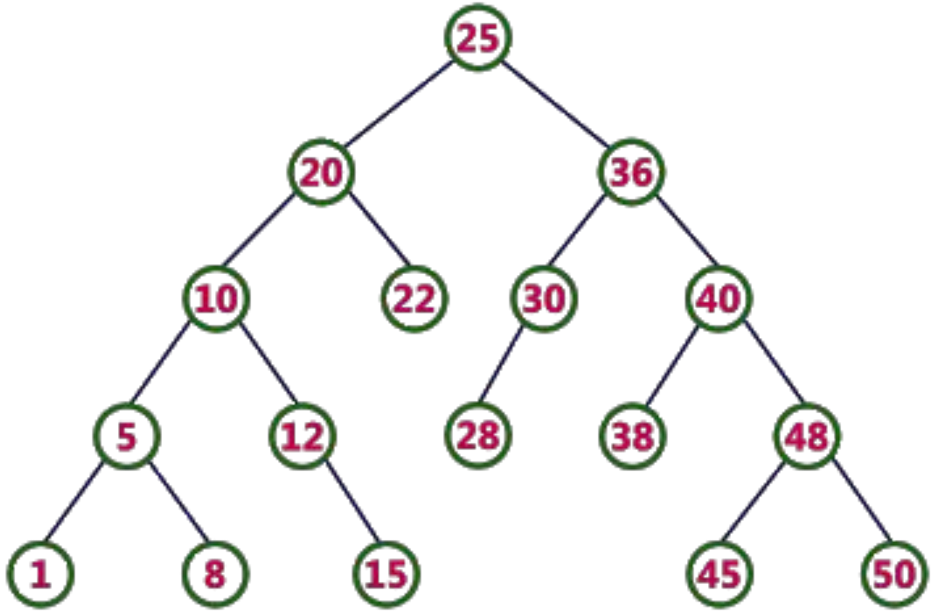
Outline

- What is a BST?
- Searching in a BST
- Traversals
- Insertion
- Deletion
- Problems



Binary Search Tree!

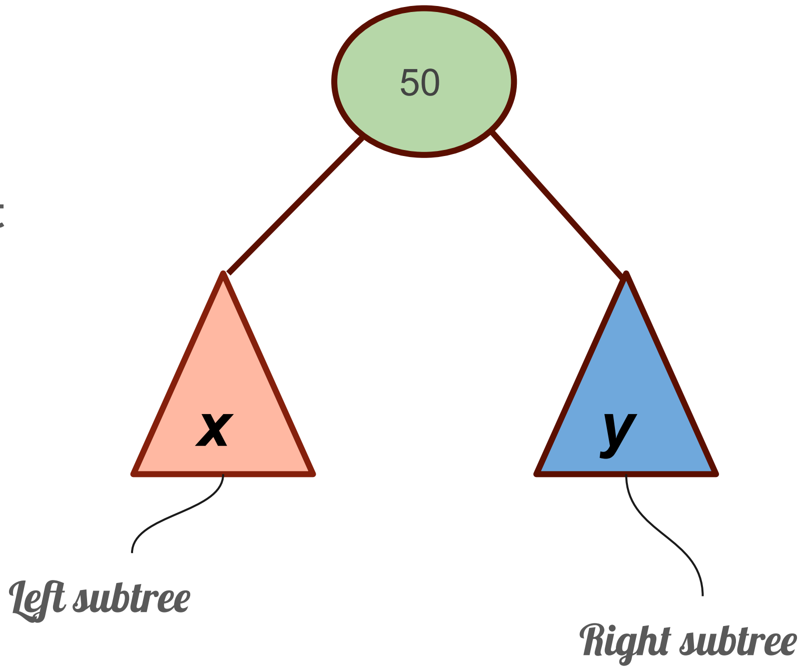
- Tree
- Binary Tree
- Search Tree



Binary Search Tree!

- All nodes of Left Subtree are less than root

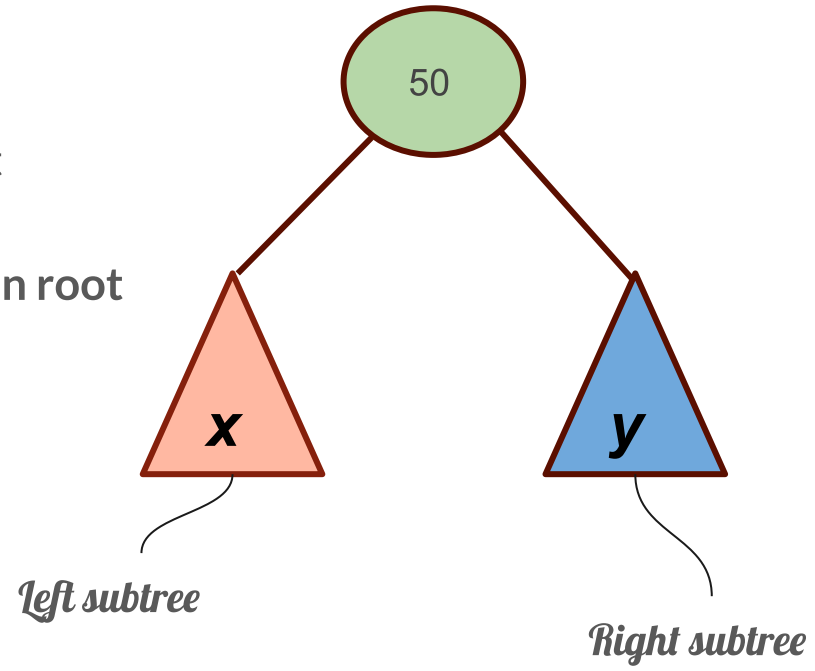
$x < 50$



Binary Search Tree!

- All nodes of Left Subtree are less than root
- **All nodes of Right Subtree are greater than root**

$y > 50$



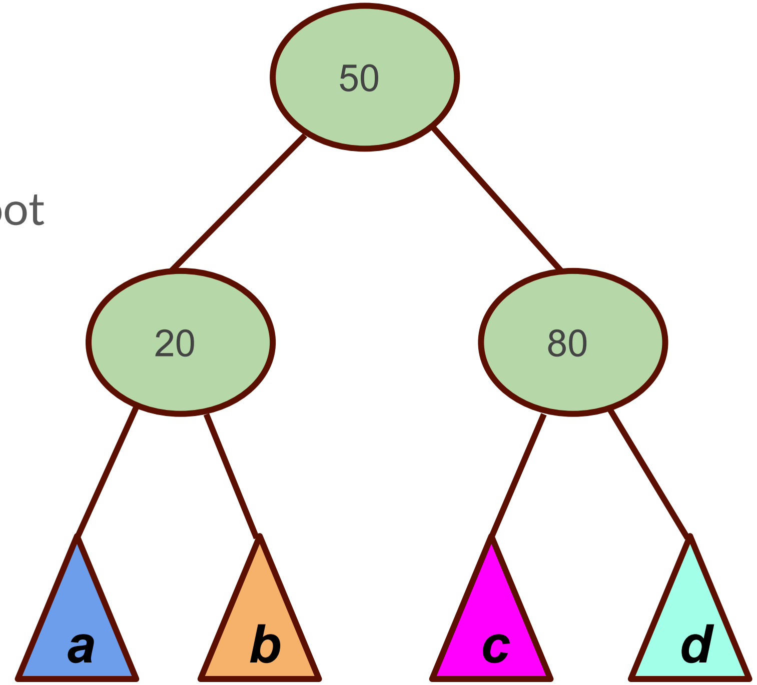
Binary Search Tree!

- All nodes of Left Subtree are less than root
- All nodes of Right Subtree are greater than root
- **Left subtree and Right subtrees are BSTs**

$a < 20$ $c < 80$

$b > 20$ $d > 80$

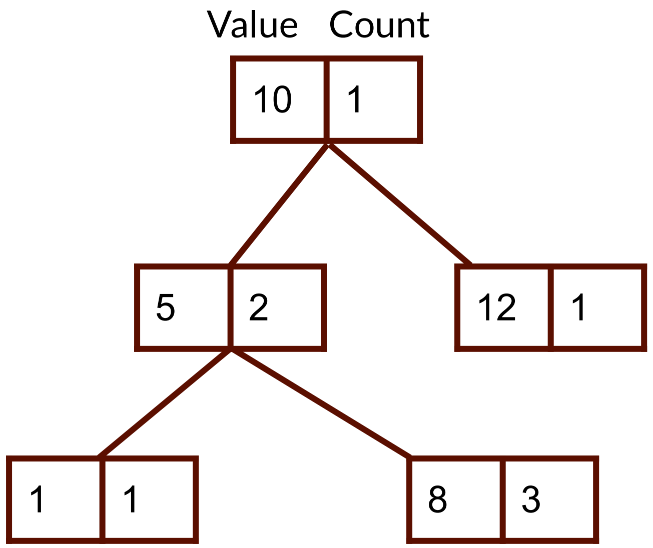
$a, b < 50$ $c, d > 50$



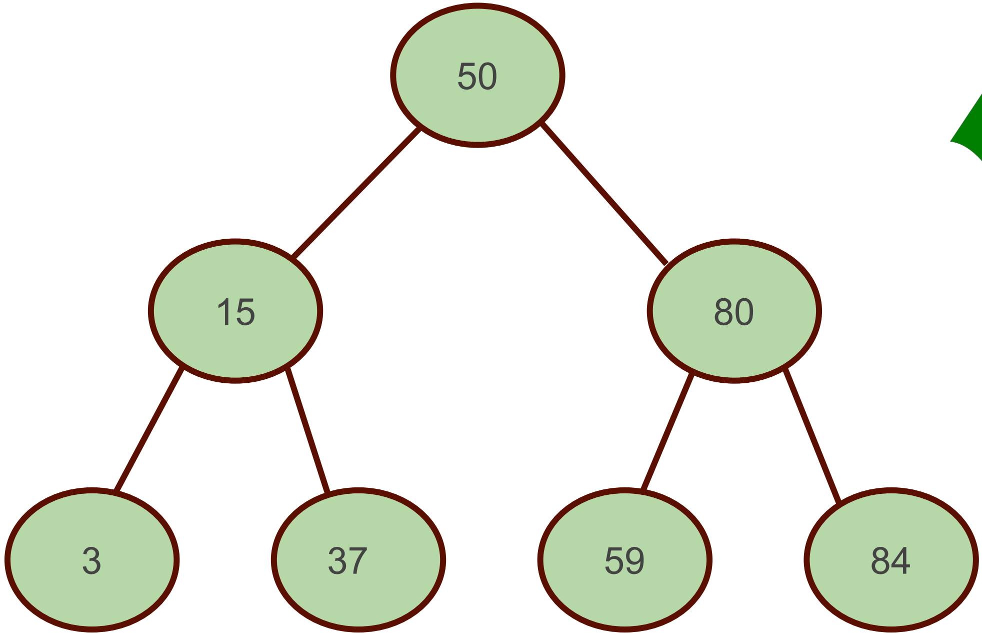
Binary Search Tree!

How to handle duplicates?

- Have a count variable for each node
- Insert it in the Left subtree
- Insert it in the Right subtree

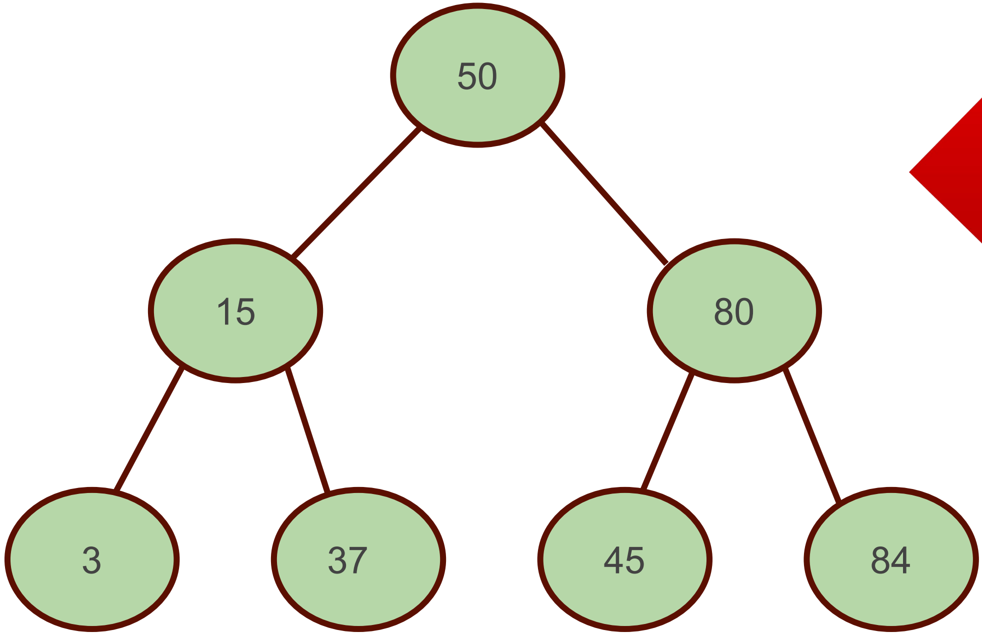


Is this a BST?

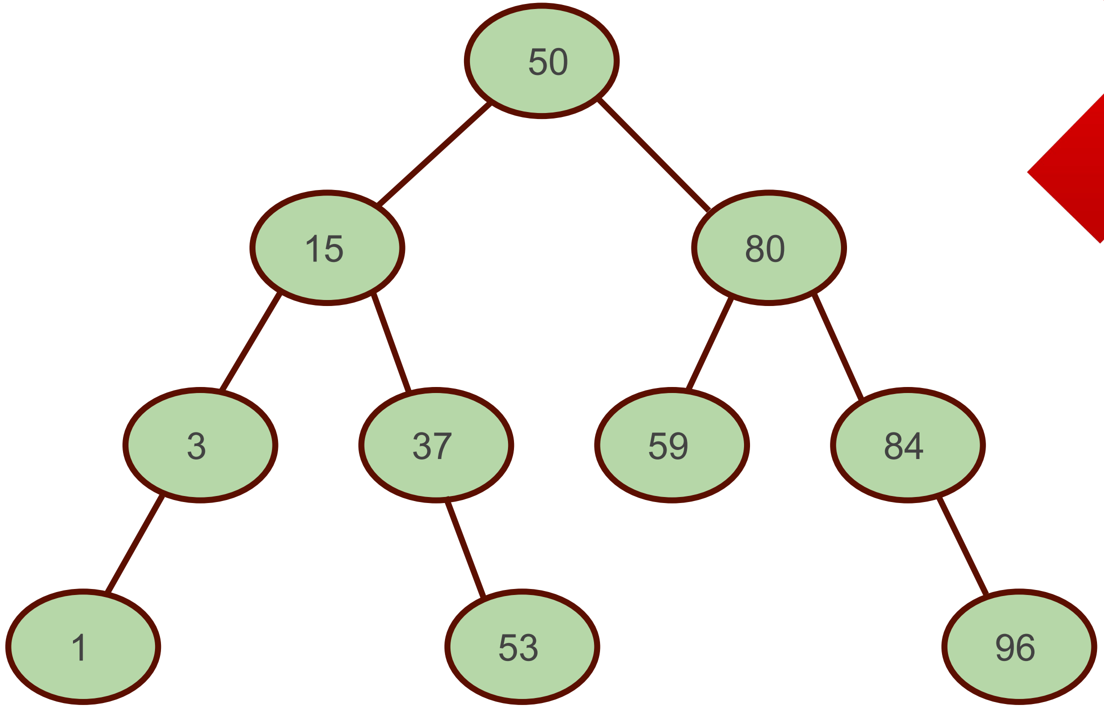


✓

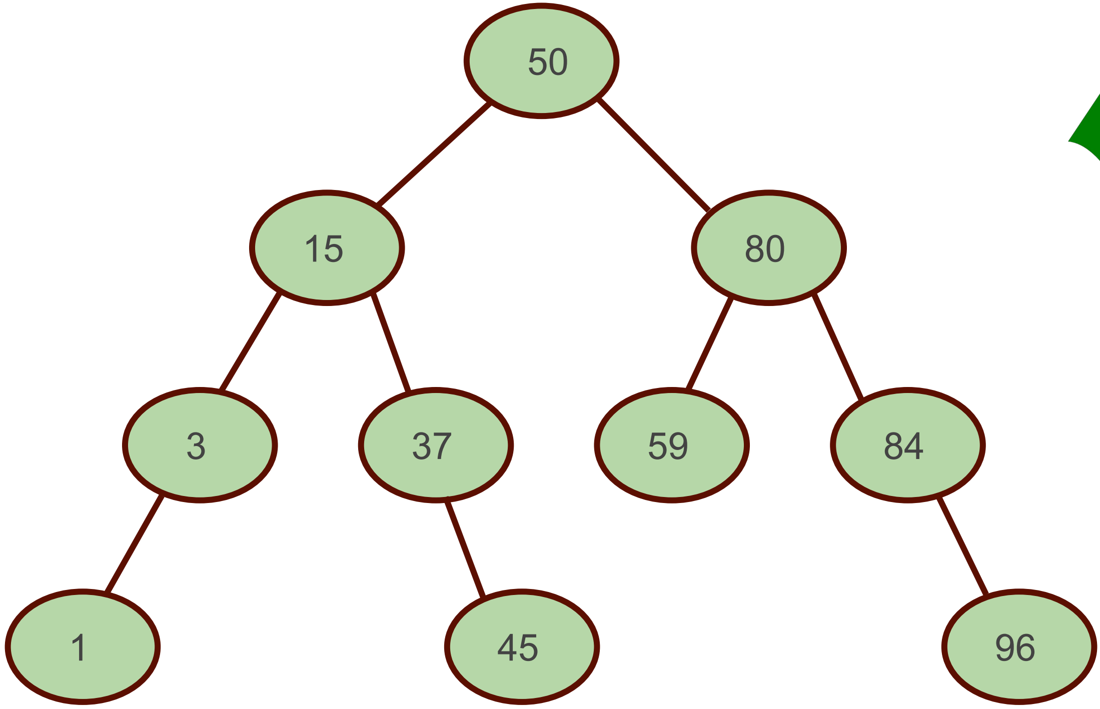
Is this a BST?



Is this a BST?

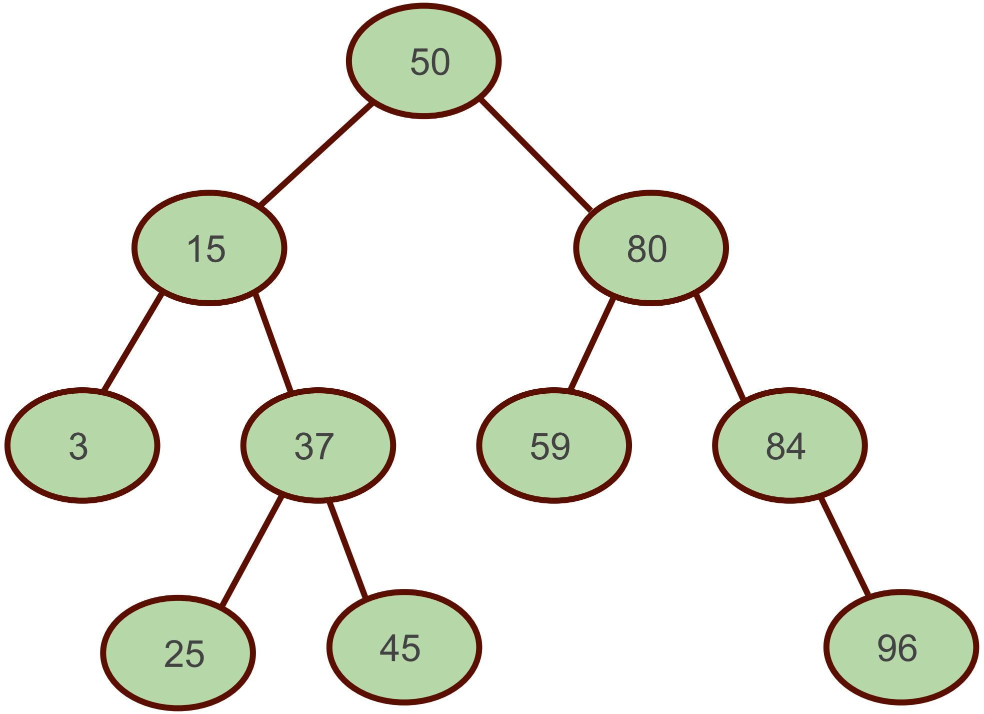


Is this a BST?



✓

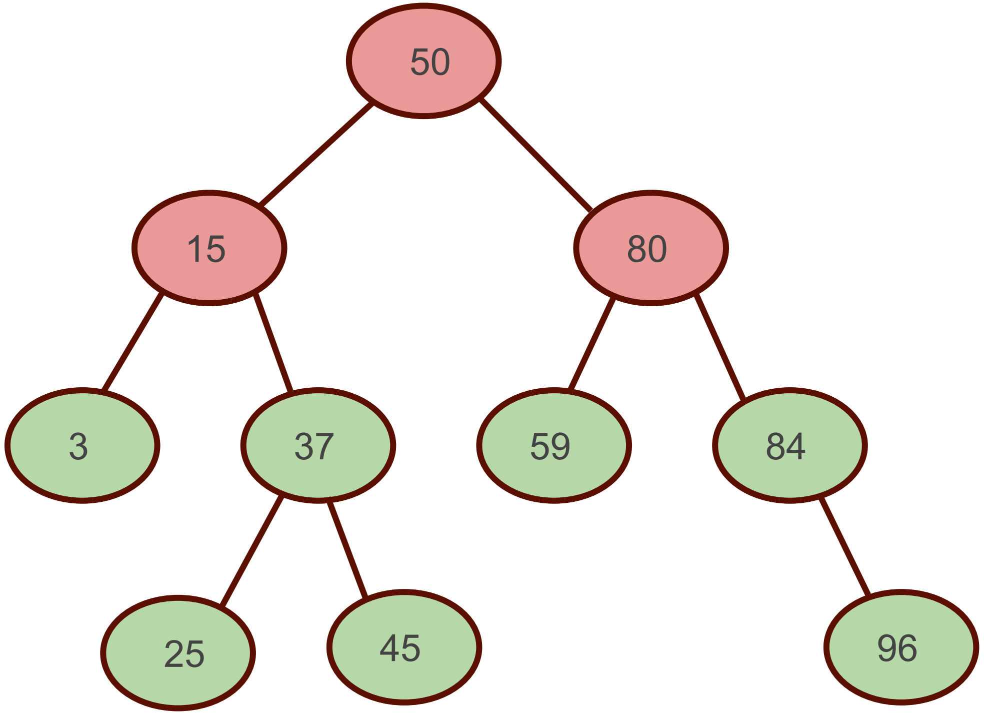
Is this a BST?



Example 1

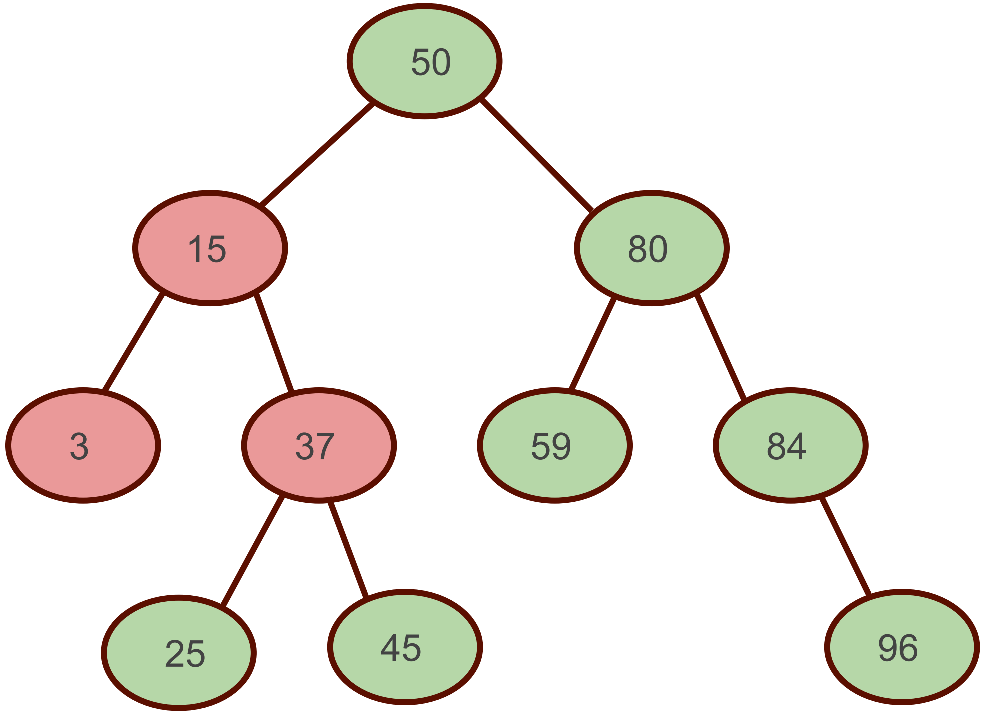
Is this a BST?

✓



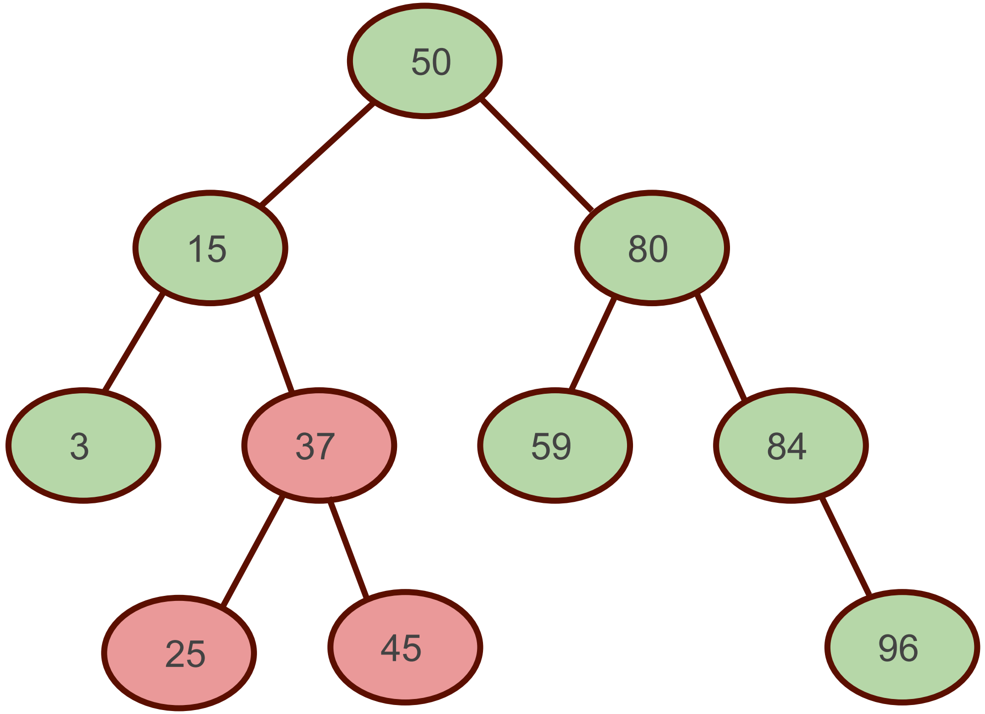
Example 1

Is this a BST?



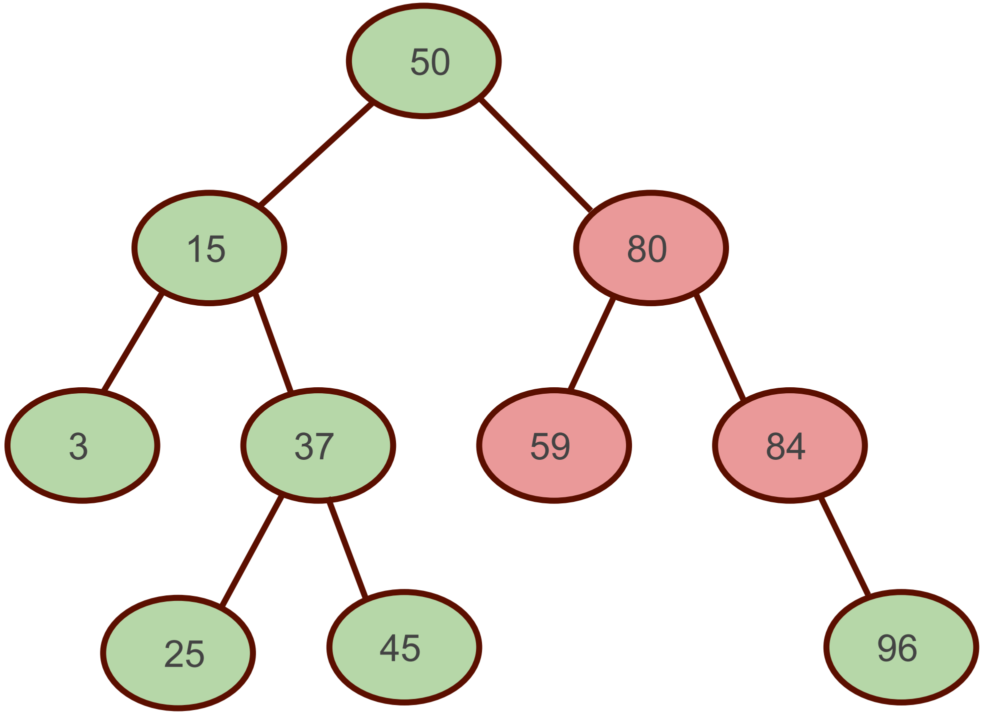
Example 1

Is this a BST?



Example 1

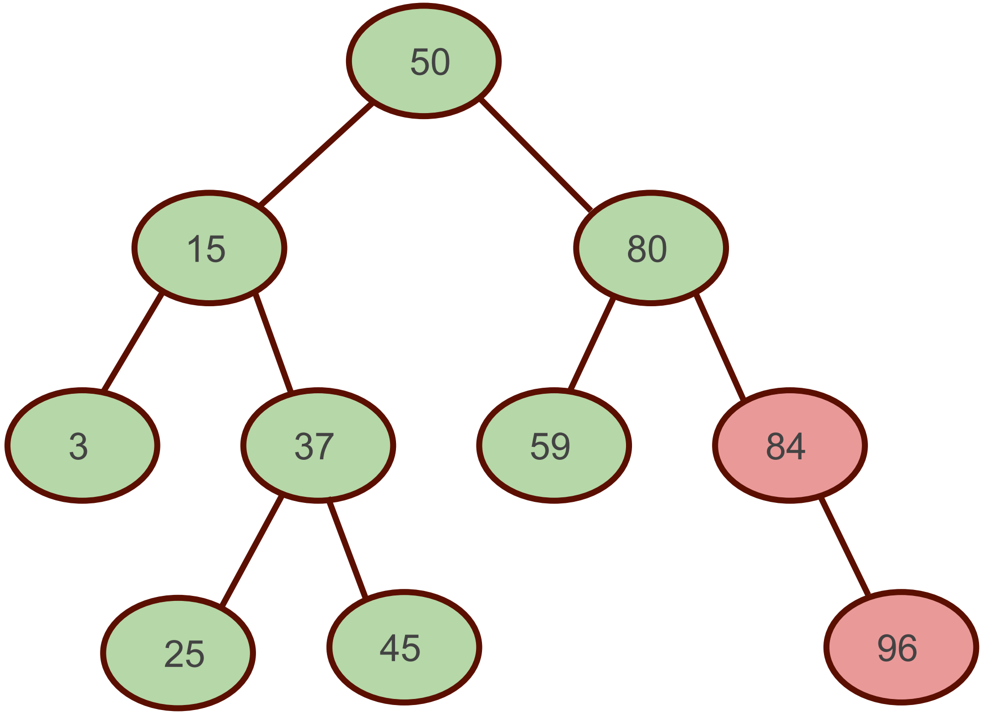
Is this a BST?



Example 1

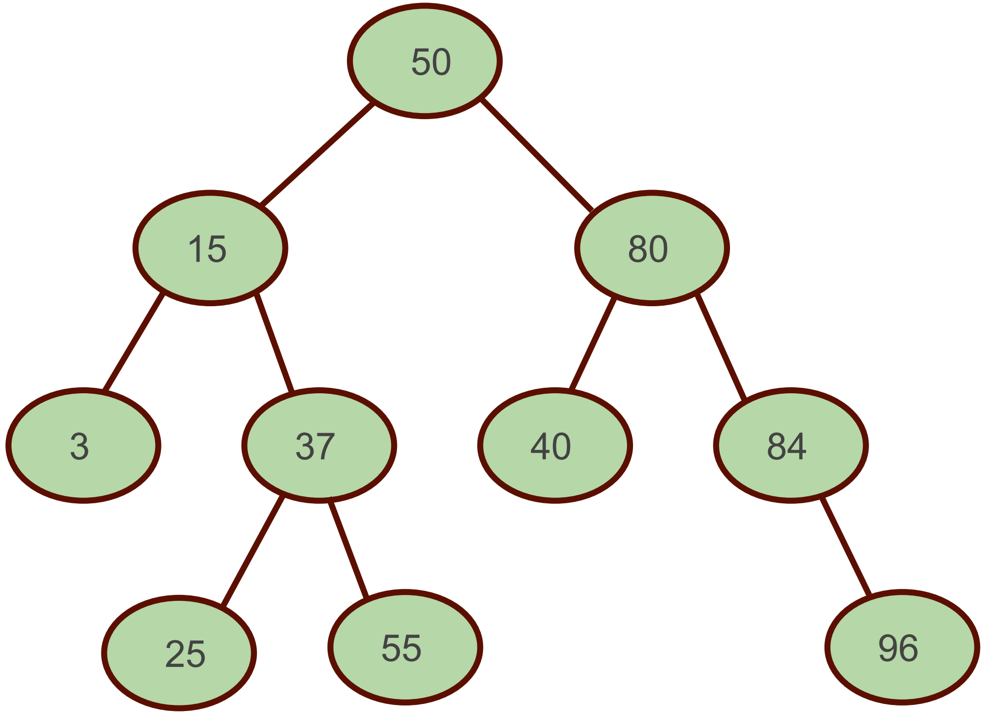
Is this a BST?

This is a BST!



Example 1

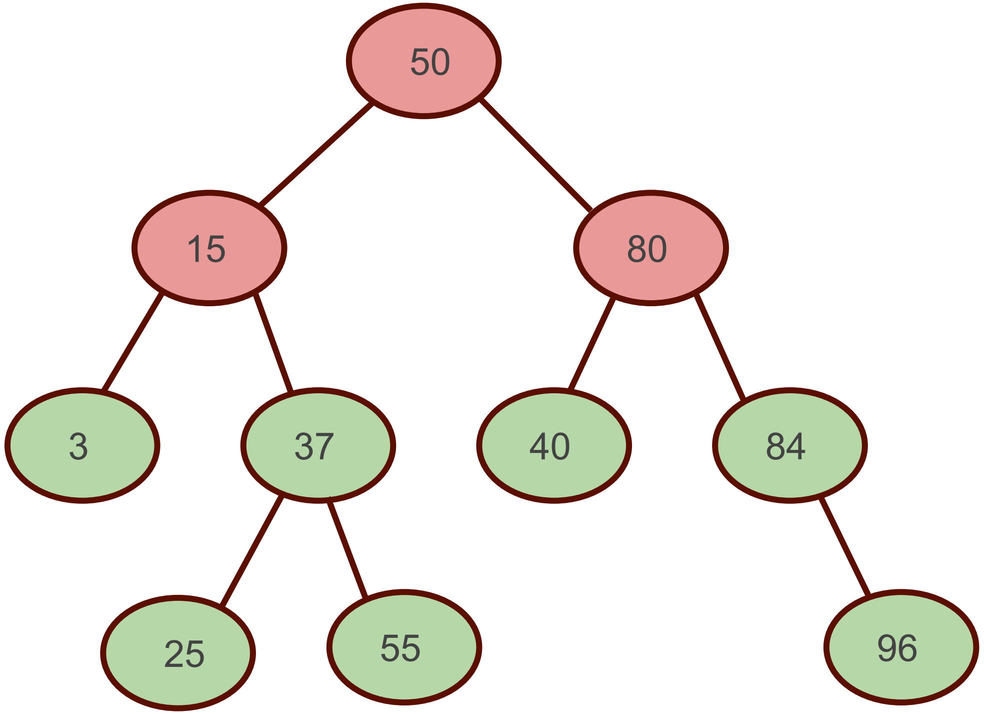
Is this a BST?



Example 2

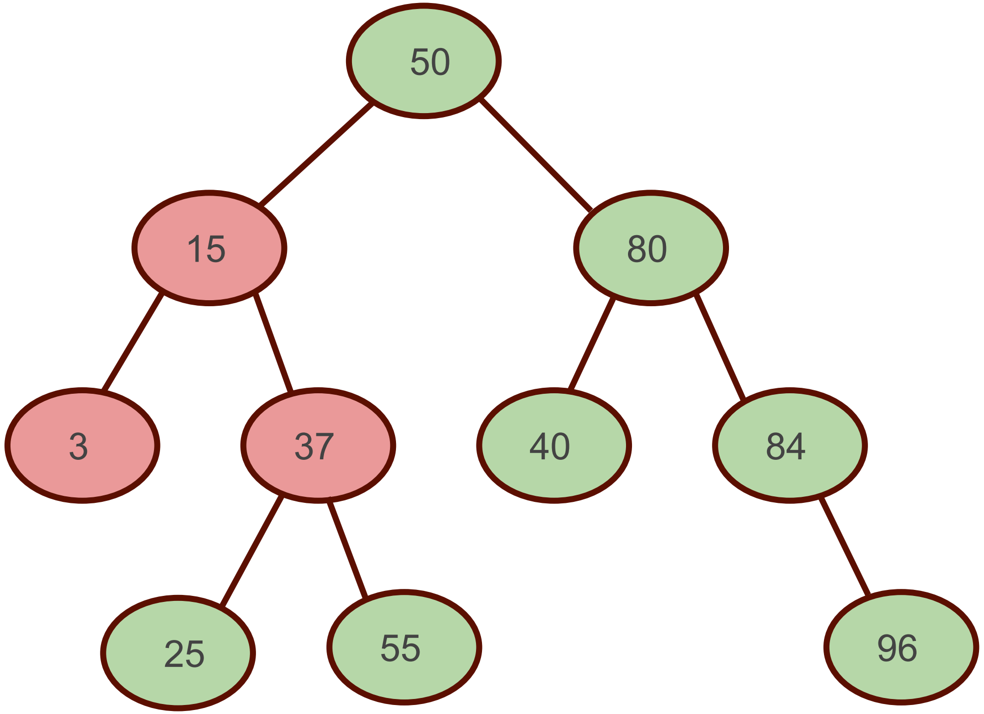
Is this a BST?

✓



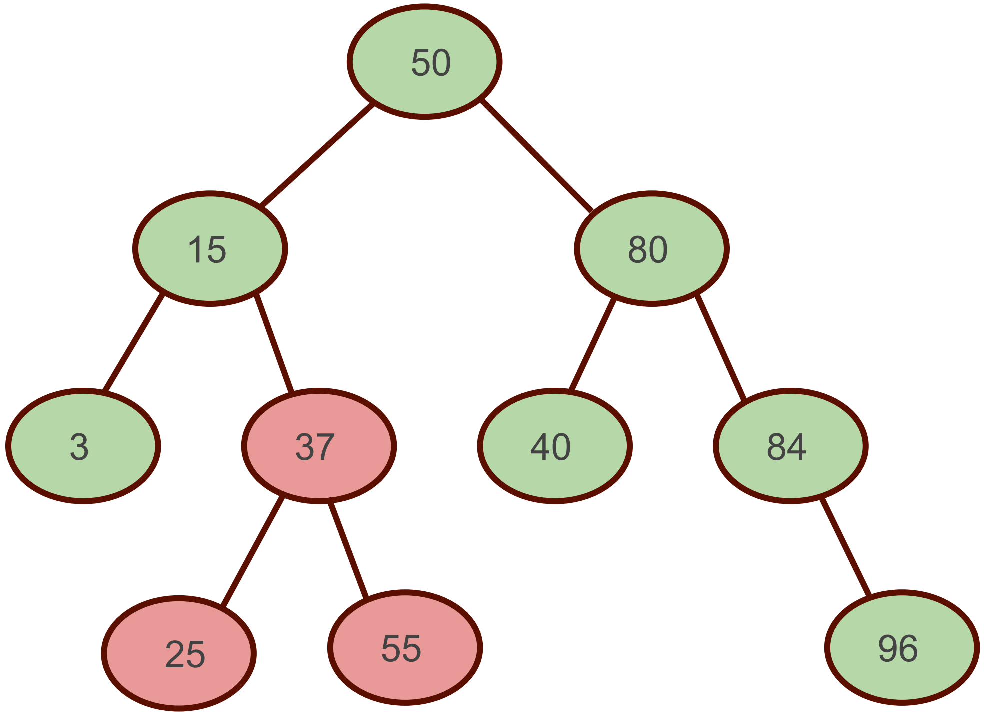
Example 2

Is this a BST?



Example 2

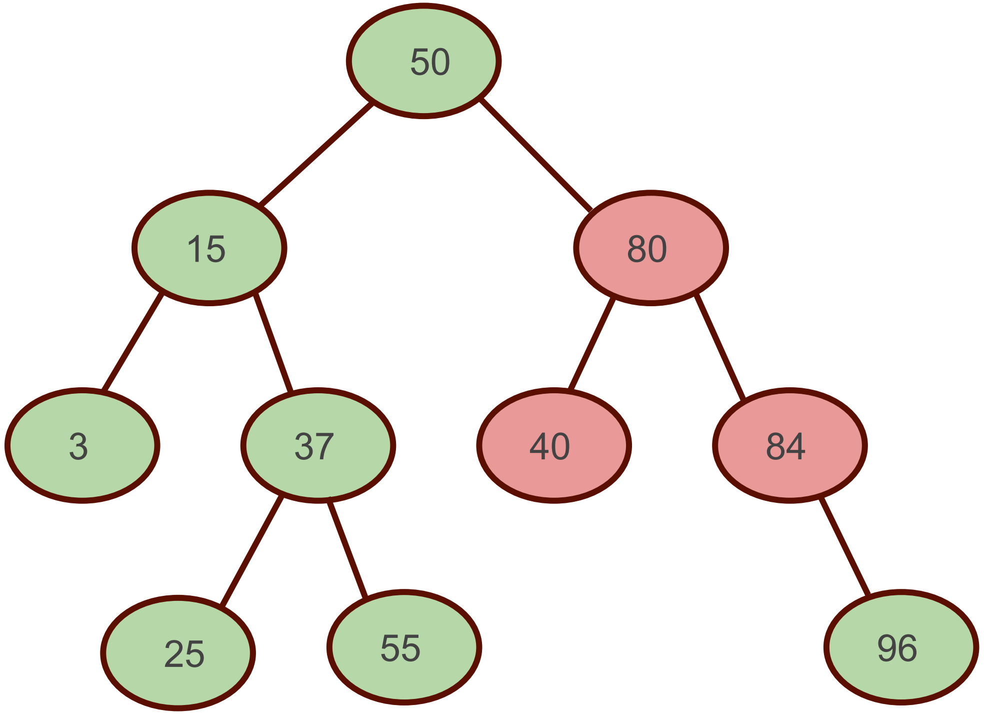
Is this a BST?



✓

Example 2

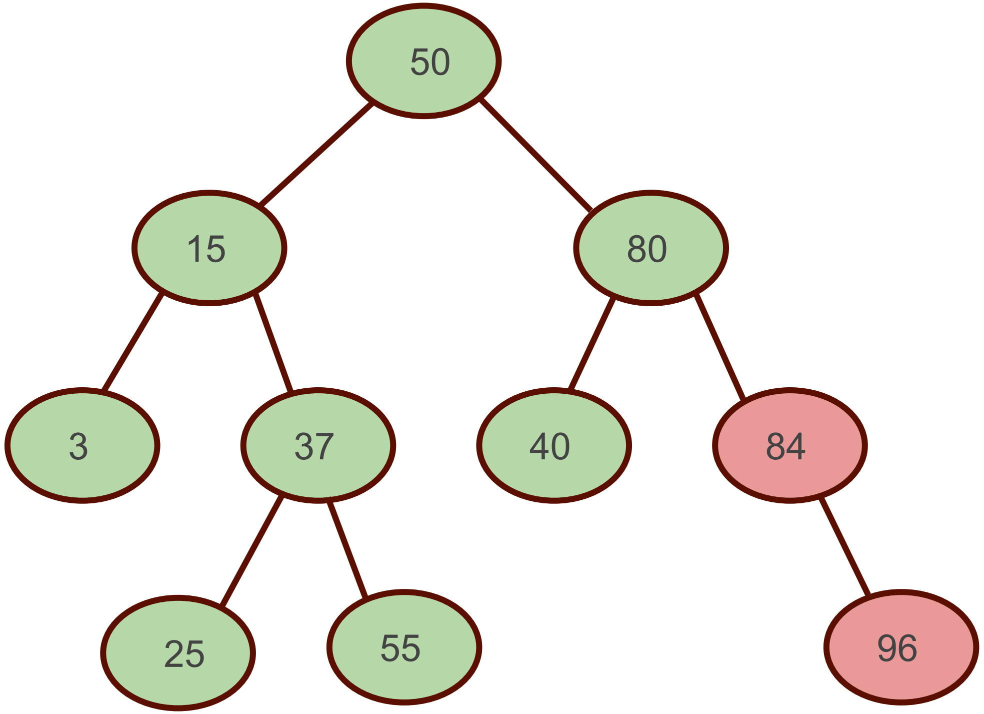
Is this a BST?



Example 2

Is this a BST?

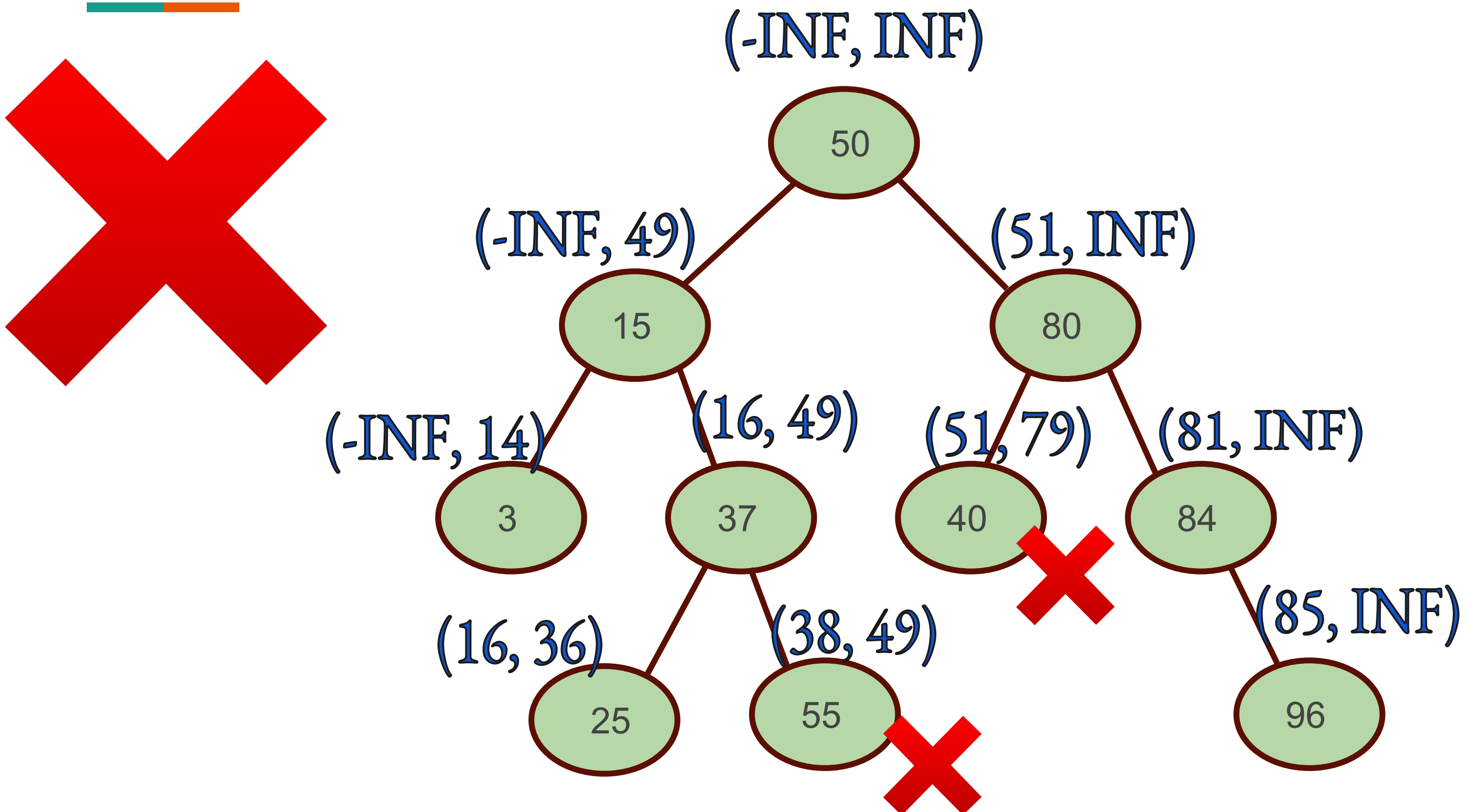
This is not a BST!



Example 2

Is this a BST?

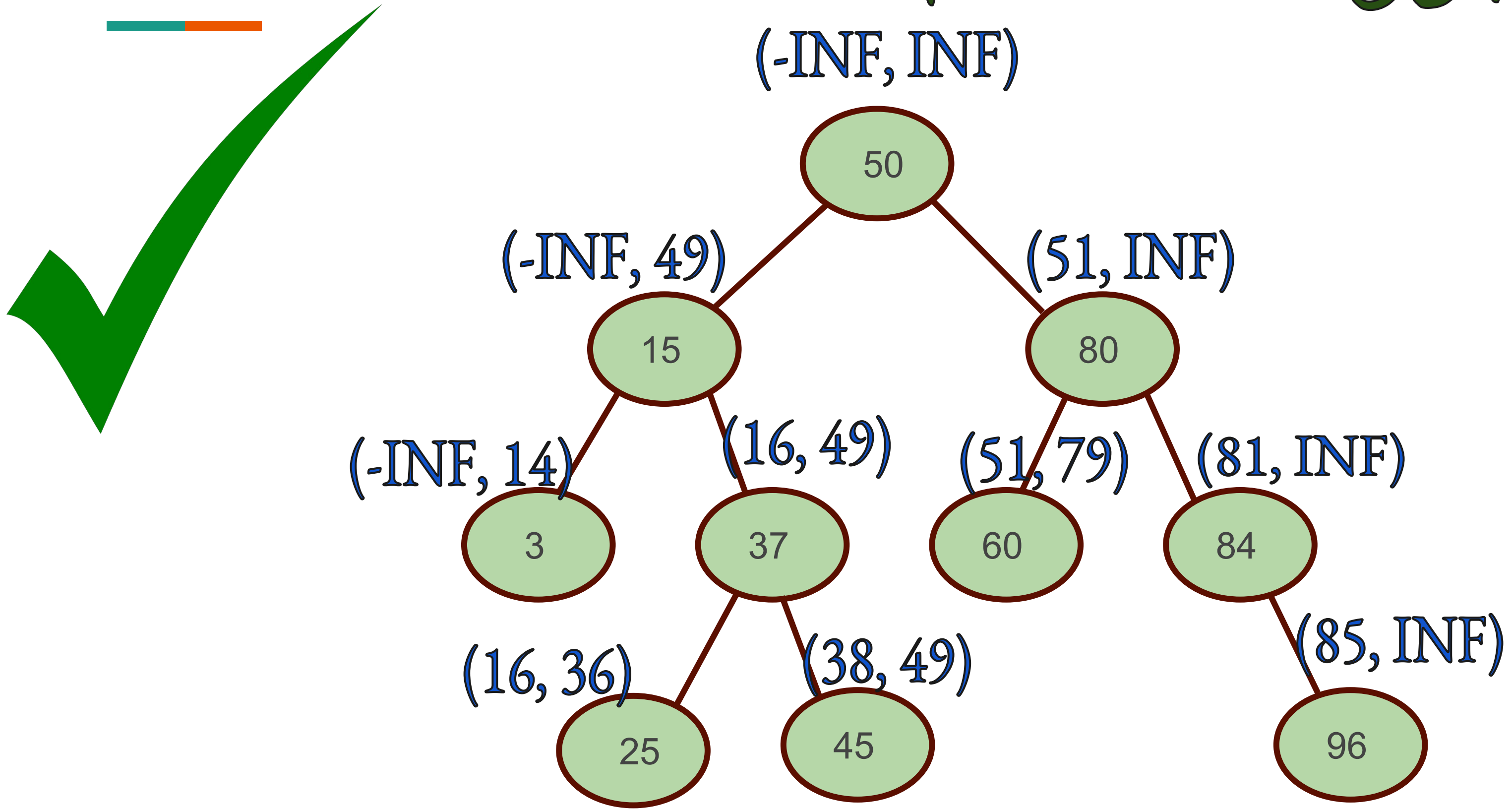
This is not a BST!



Example 1

Is this a BST?

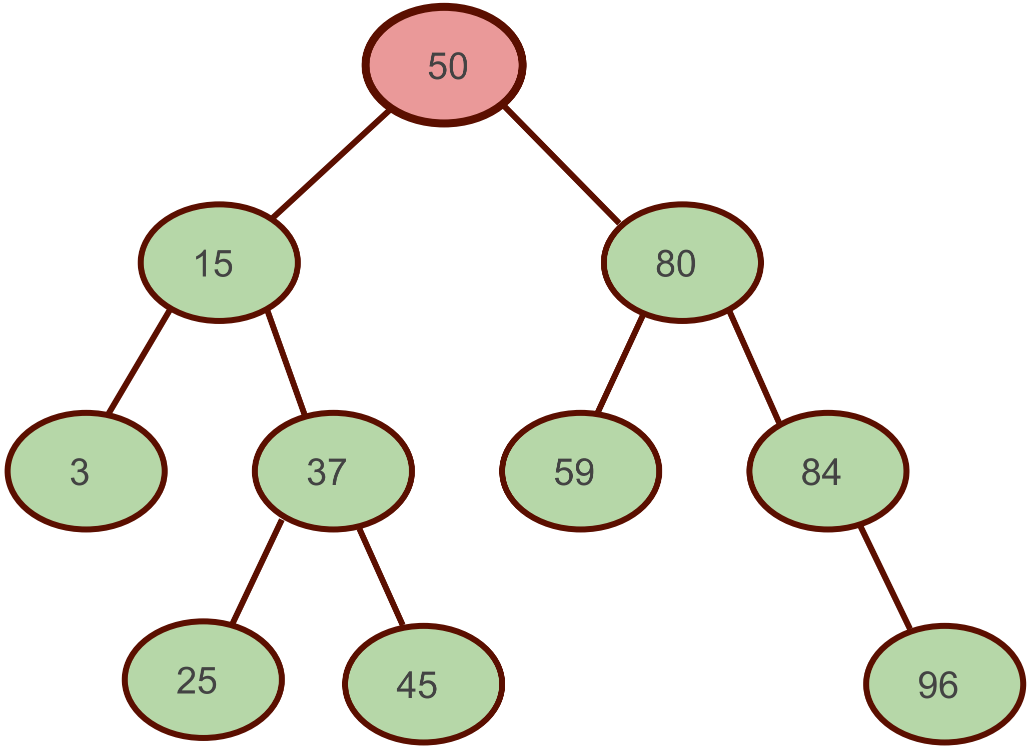
This is a BST!



Example 1

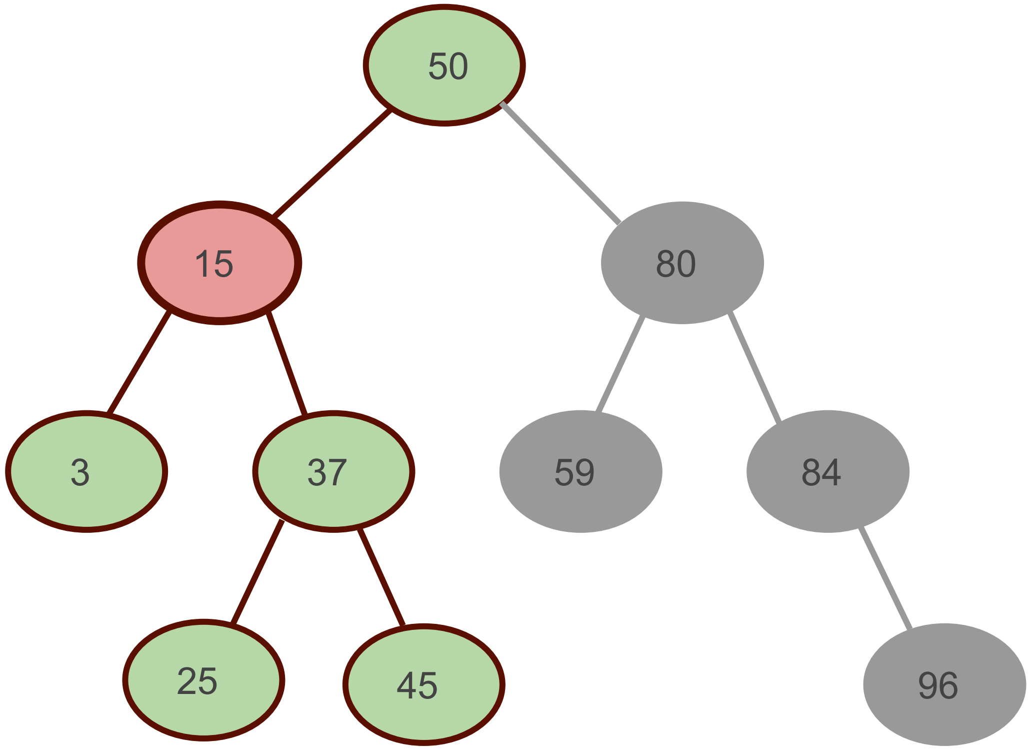
Searching in a BST!

Search for 45!



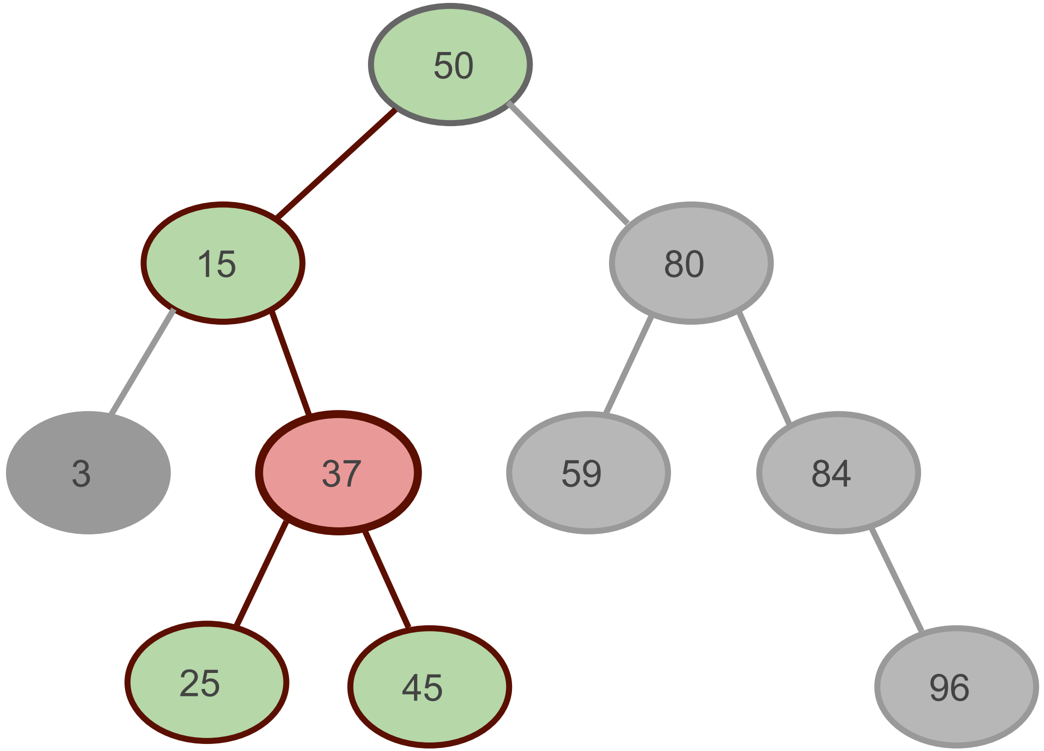
Searching in a BST!

Search for 45!



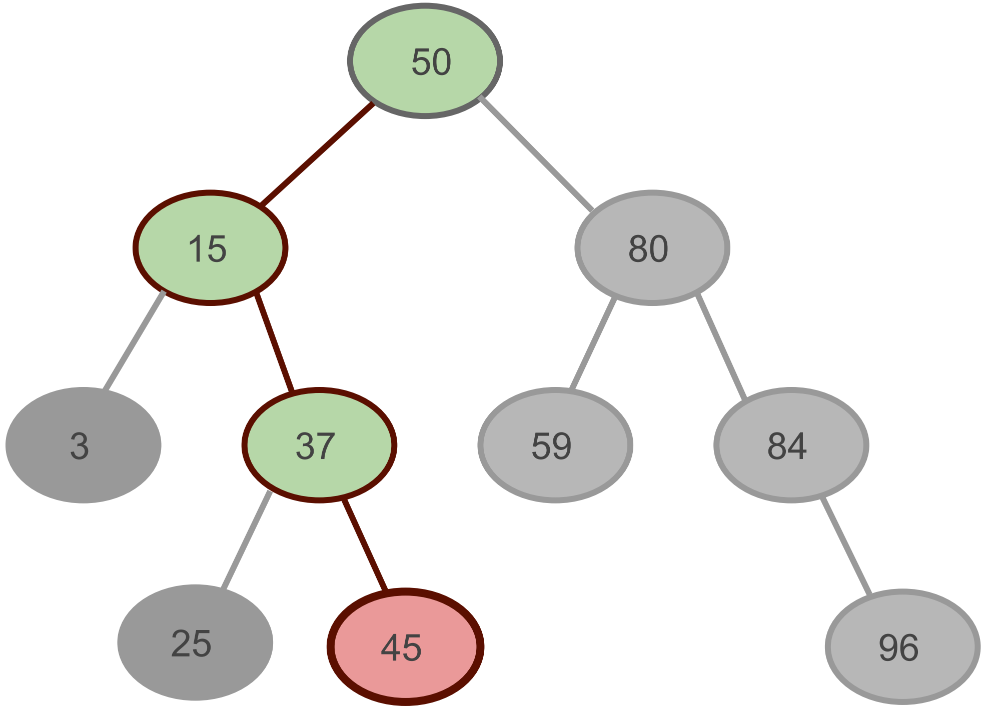
Searching in a BST!

Search for 45!



Searching in a BST!

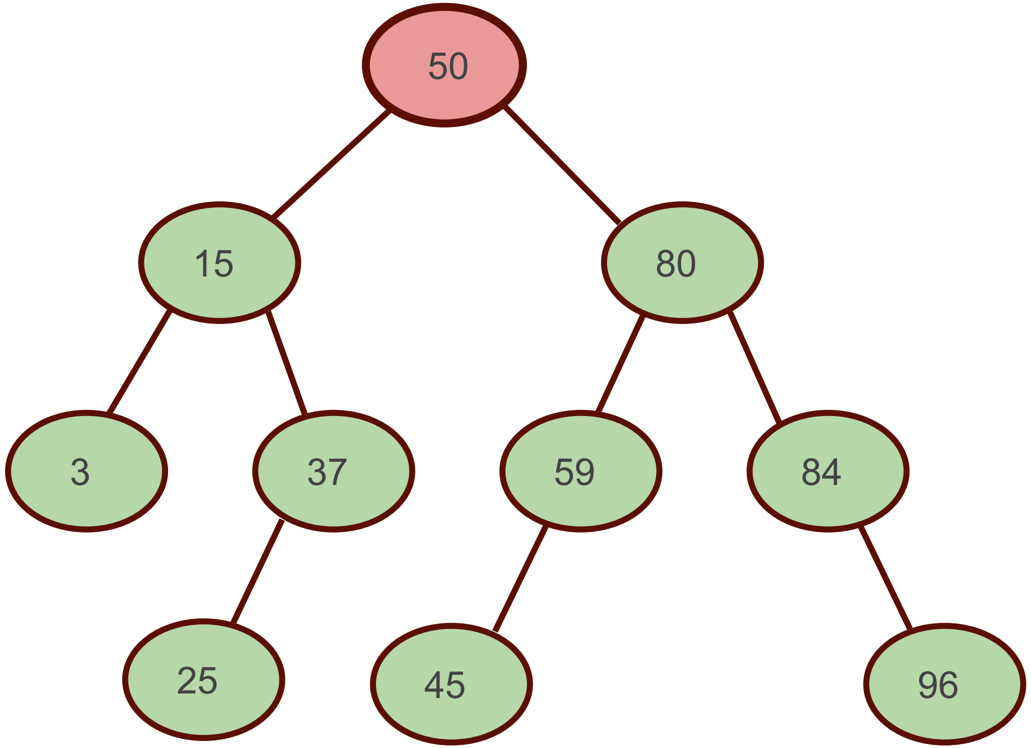
Search for 45!



Found :)

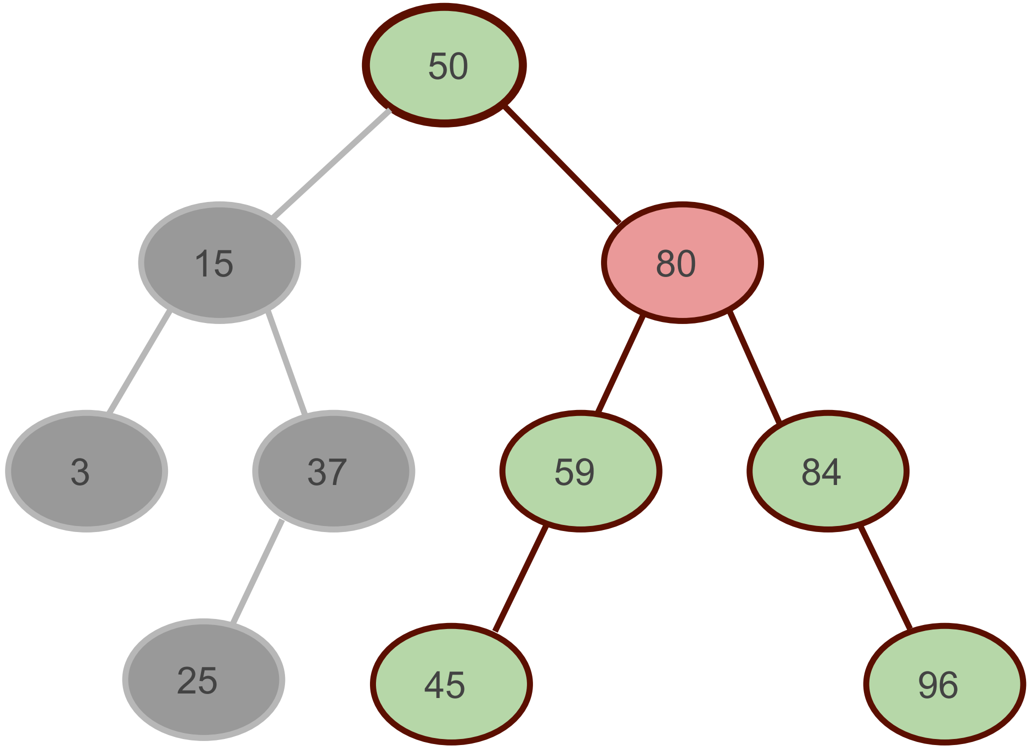
Searching in a BST!

Search for 82!



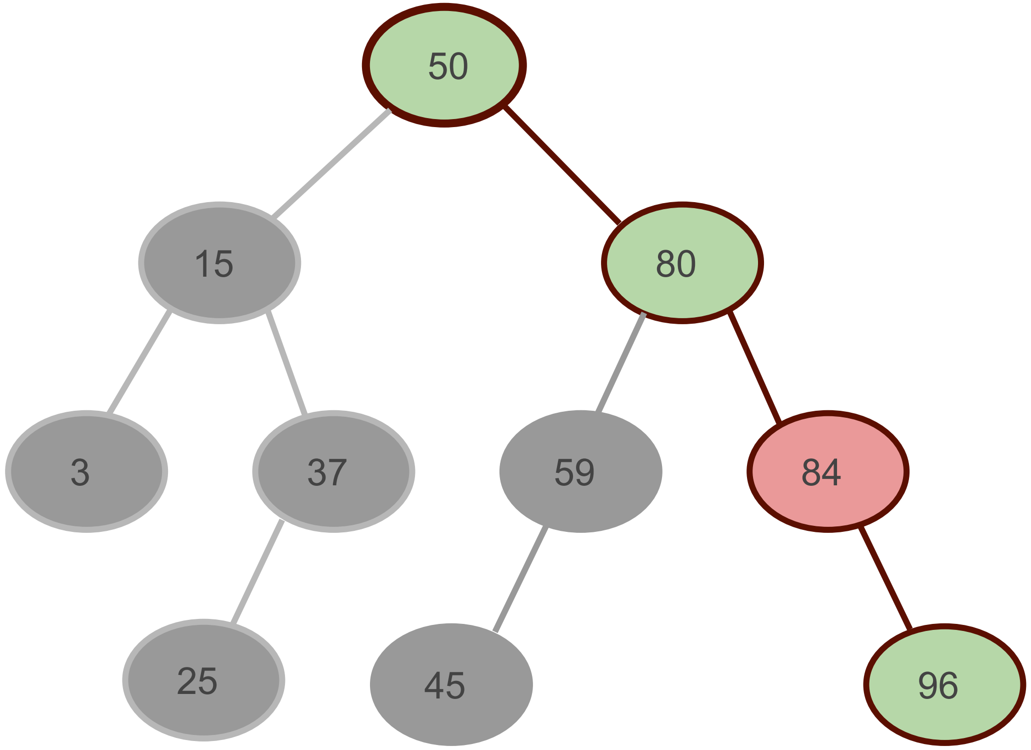
Searching in a BST!

Search for 82!



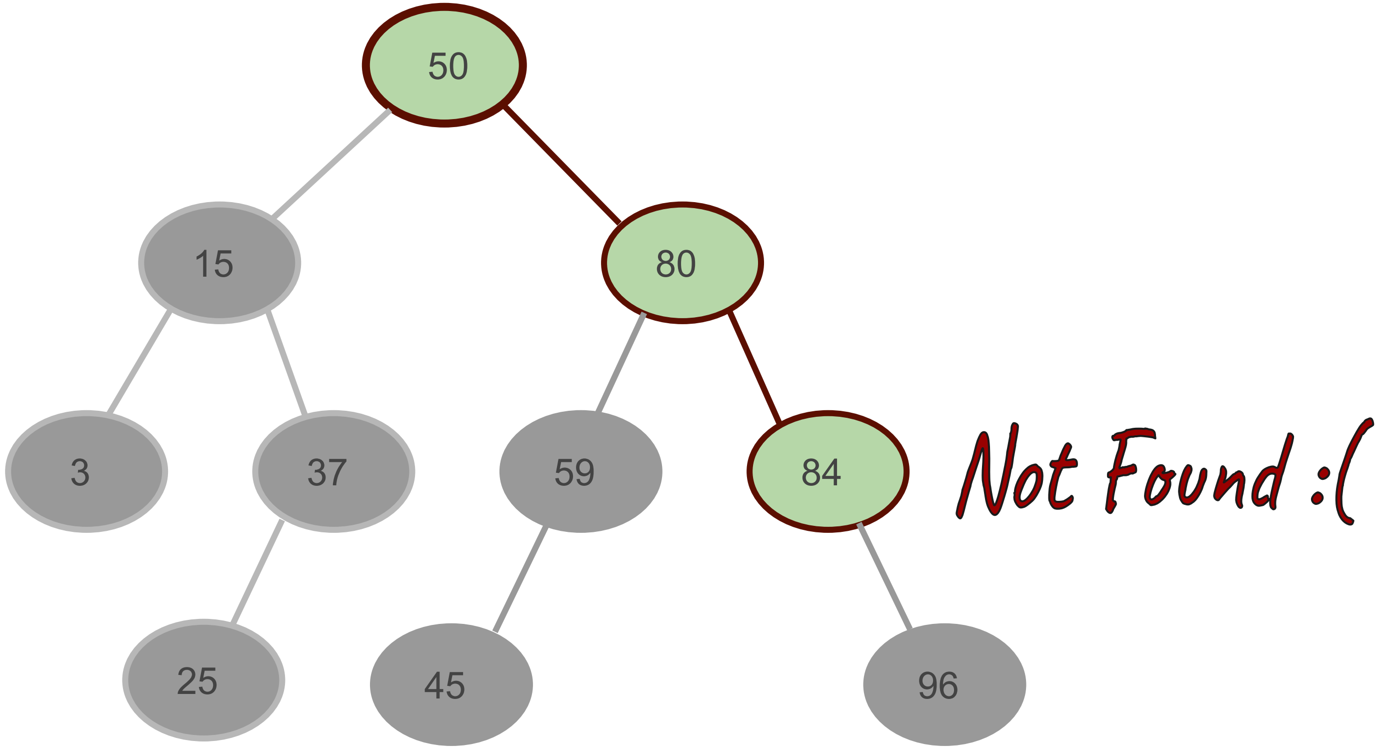
Searching in a BST!

Search for 82!

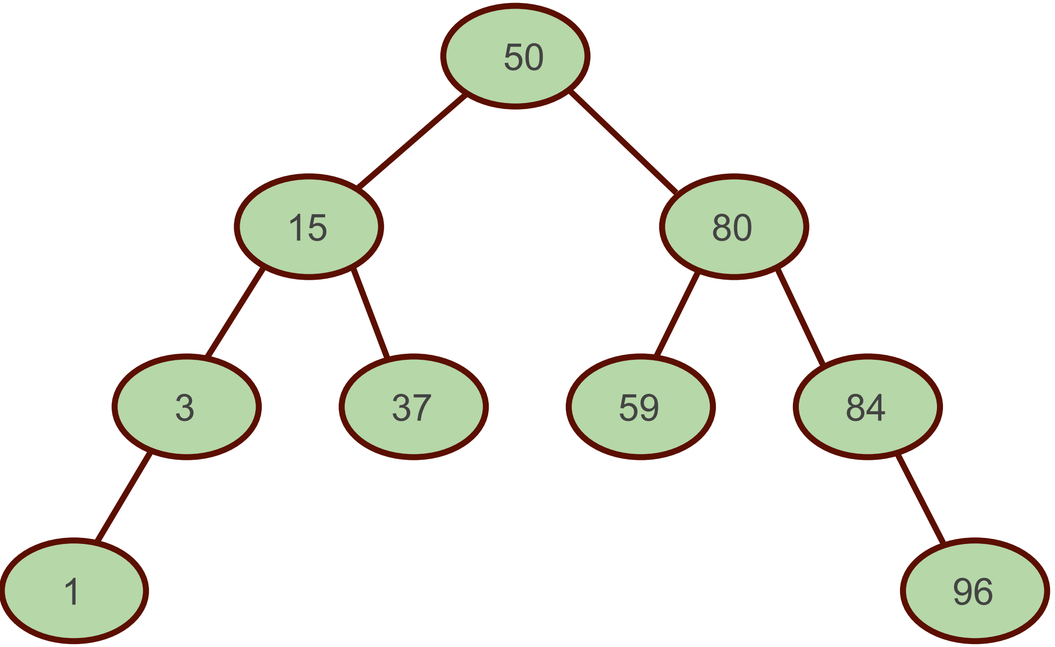


Searching in a BST!

Search for 82!

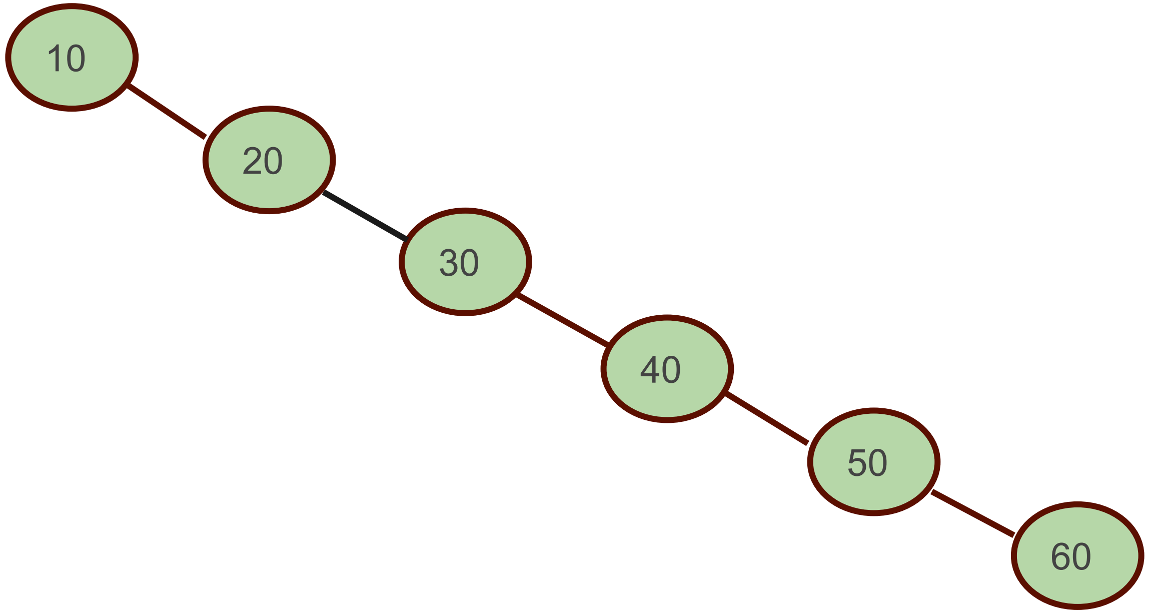


Complexity of Searching in a BST!



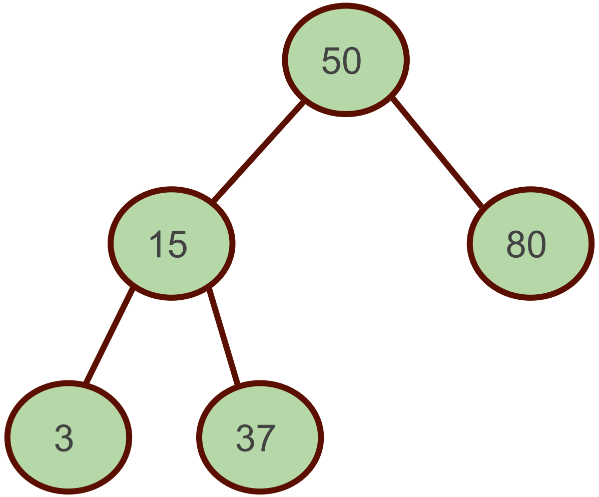
- O(logn)?
- O(h) where h is the height of tree
- h can be n in the worst case
- Worst case complexity : O(n)
- Average case : O(logn)

Complexity of Searching in a BST!



$O(n)$

Find all the 3 traversals

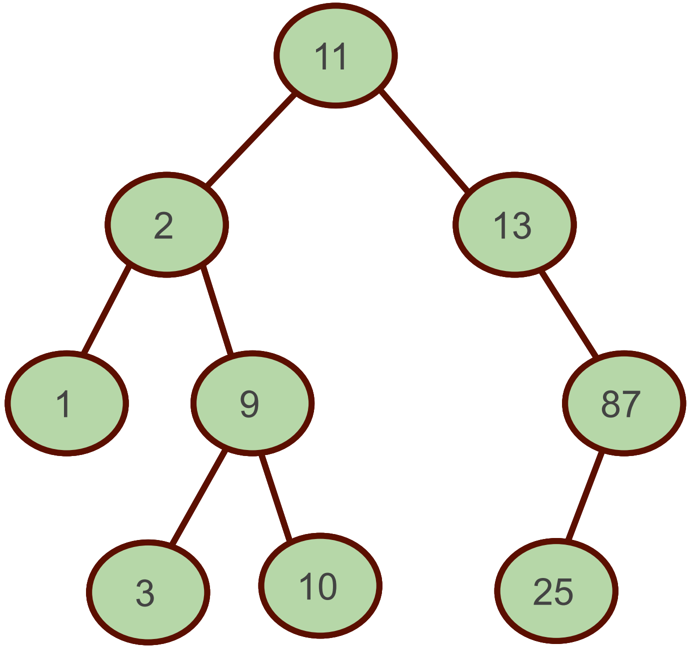


__Inorder__ : 3 15 37 50 80

__Preorder__ : 50 15 3 37 80

__Postorder__ : 3 37 15 80 50

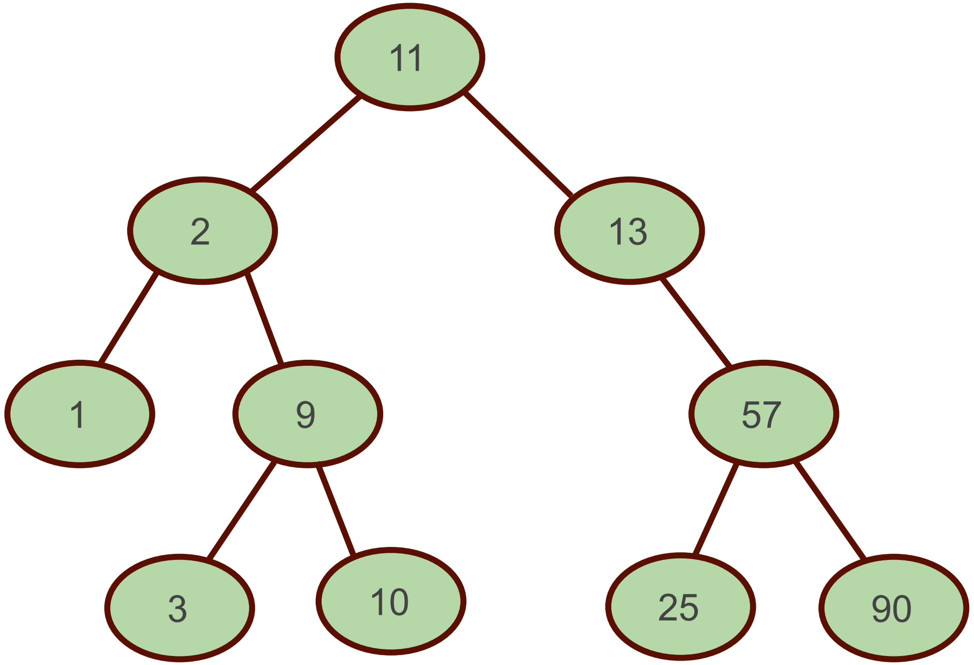
Inorder Traversal



1 2 3 9 10 11 13 25 87

Insertion in a BST!

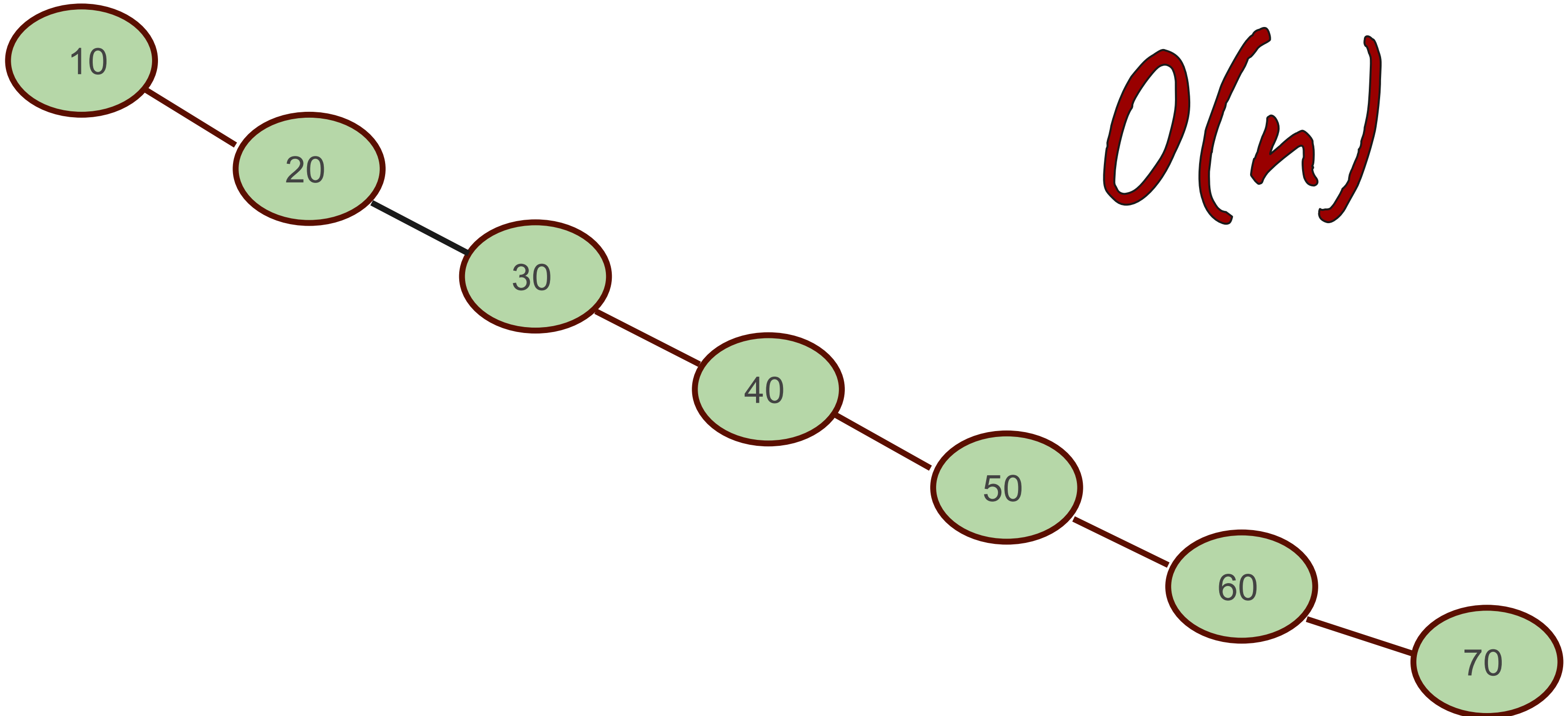
```
Insert 11, 2, 9, 13, 57, 25, 1, 90, 3, 10
```



- O(h) where h is the height of tree
- h can be n in the worst case
- Worst case complexity : O(n)
- Average case : O(logn)

Insertion in a BST!

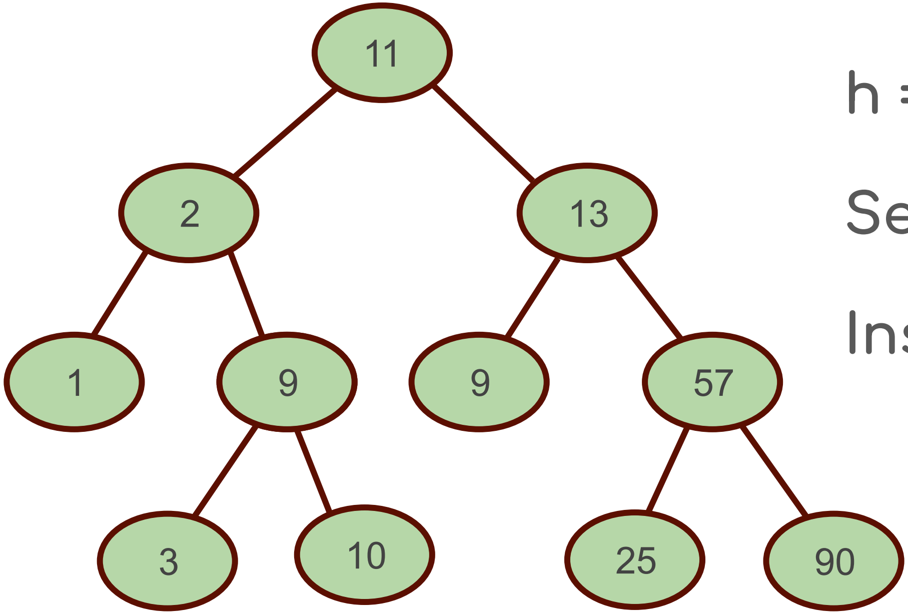
`Insert 10, 20, 30, 40, 50, 60, 70`



$O(n)$

Balanced BST!

- A binary search tree is *balanced* if and only if the height of the two subtrees of every node never differ by more than 1.
- Height of a balanced tree would be O(logn)



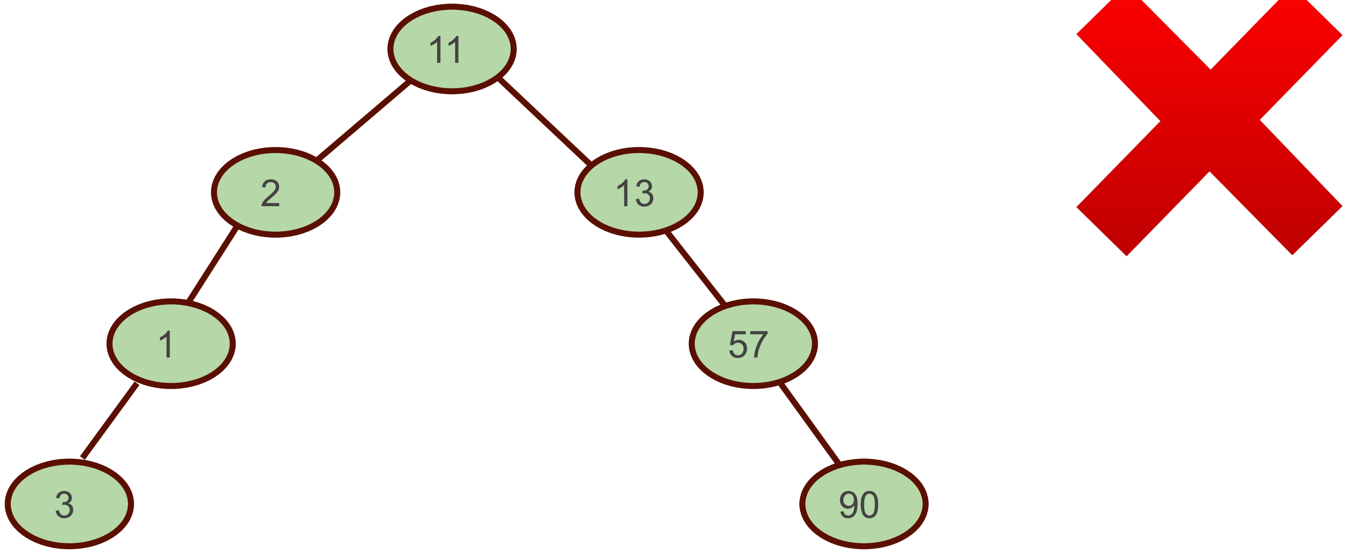
h = Log(n)

Search : O(Logn)

Insertion : O(Logn)

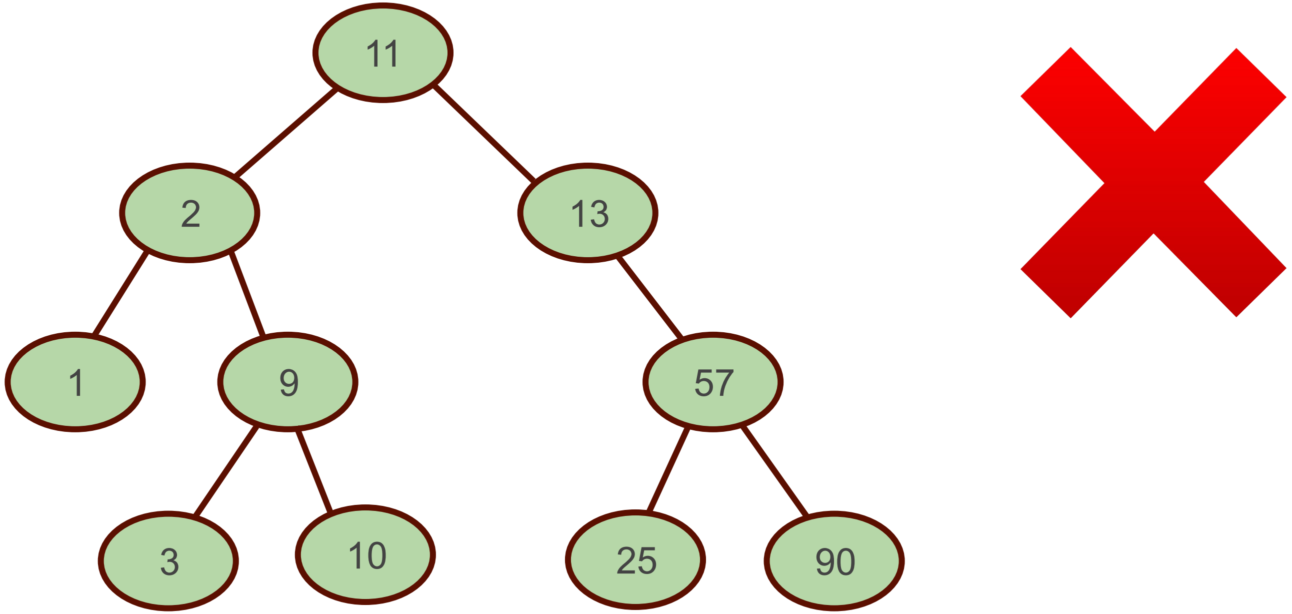
Balanced BST!

- A binary search tree is *balanced* if and only if the depth of the two subtrees of every node never differ by more than 1.
- Height of a Balanced tree would be O(logn)

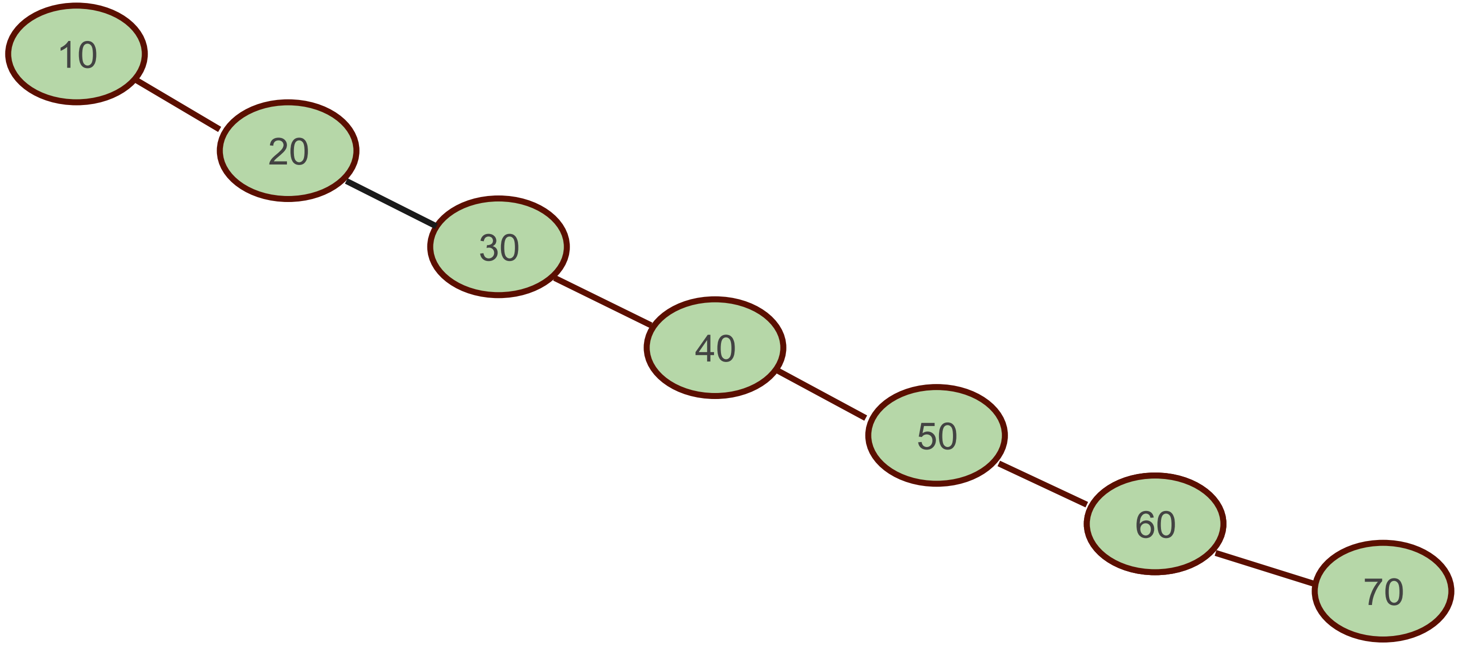


Balanced BST!

- A binary search tree is *balanced* if and only if the depth of the two subtrees of every node never differ by more than 1.
- Height of a Balanced tree would be O(logn)



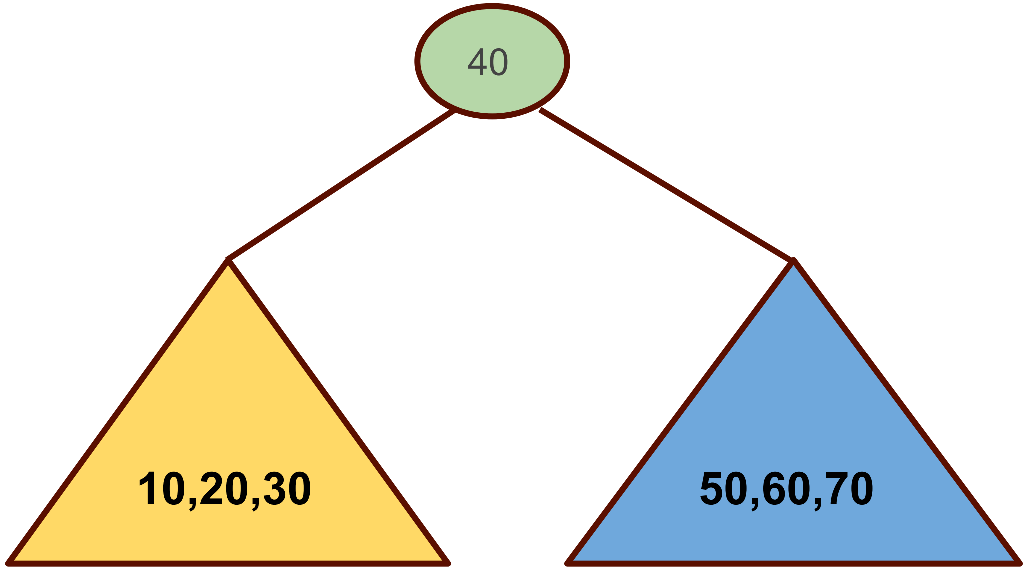
Balancing a BST!



```
Inorder Traversal : 10, 20, 30, 40, 50, 60, 70
```

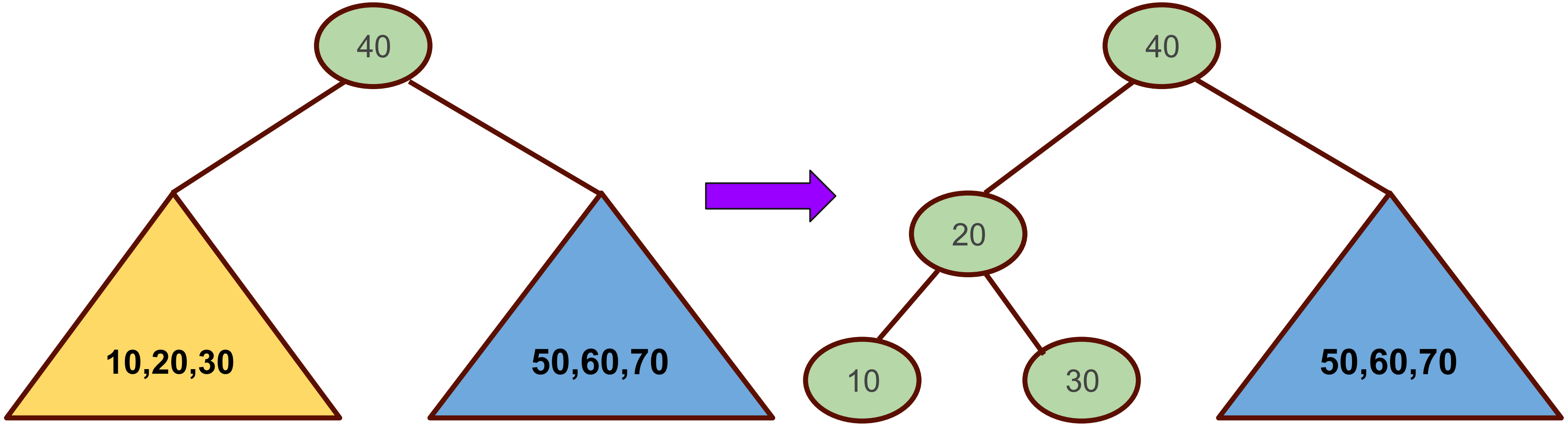
Balancing a BST!

```
Inorder Traversal : 10, 20, 30, 40, 50, 60, 70
```



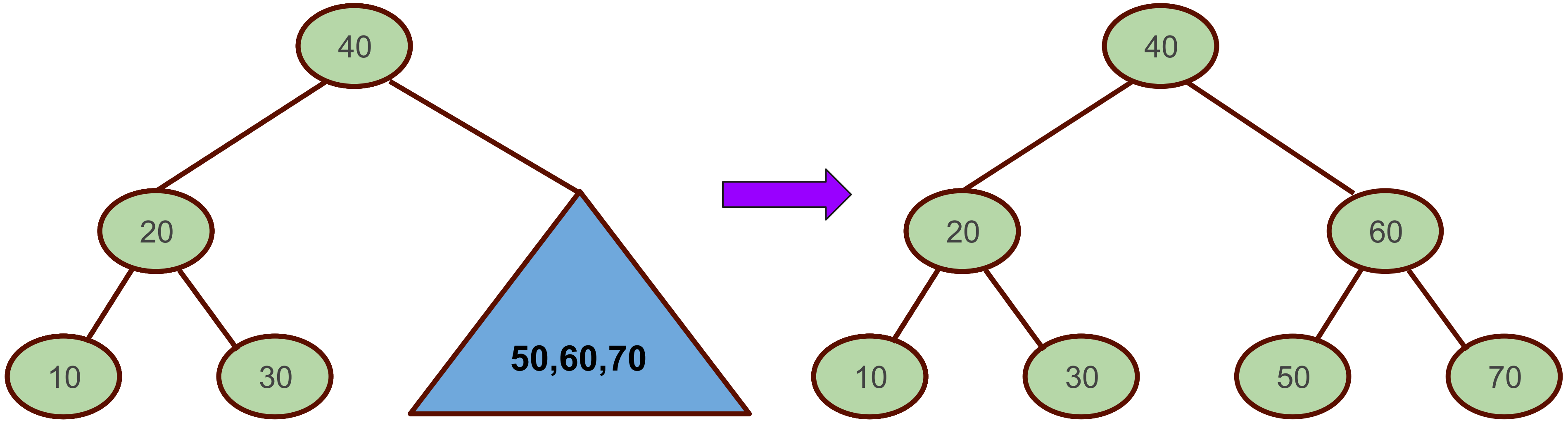
Balancing a BST!

`Inorder Traversal : 10, 20, 30, 40, 50, 60, 70`



Balancing a BST!

```
Inorder Traversal : 10, 20, 30, 40, 50, 60, 70
```

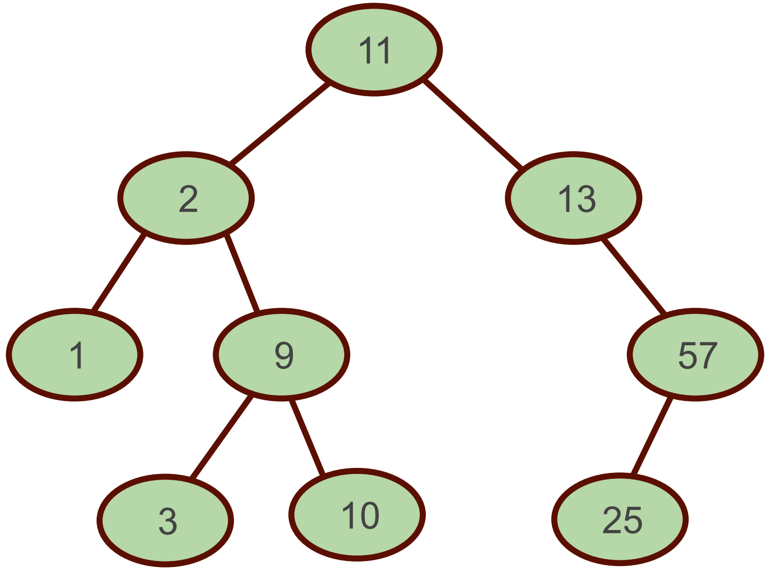


Deletion in a BST!

Case 1 : Node to be deleted is a leaf

Case 2 : Node to be deleted has 1 child

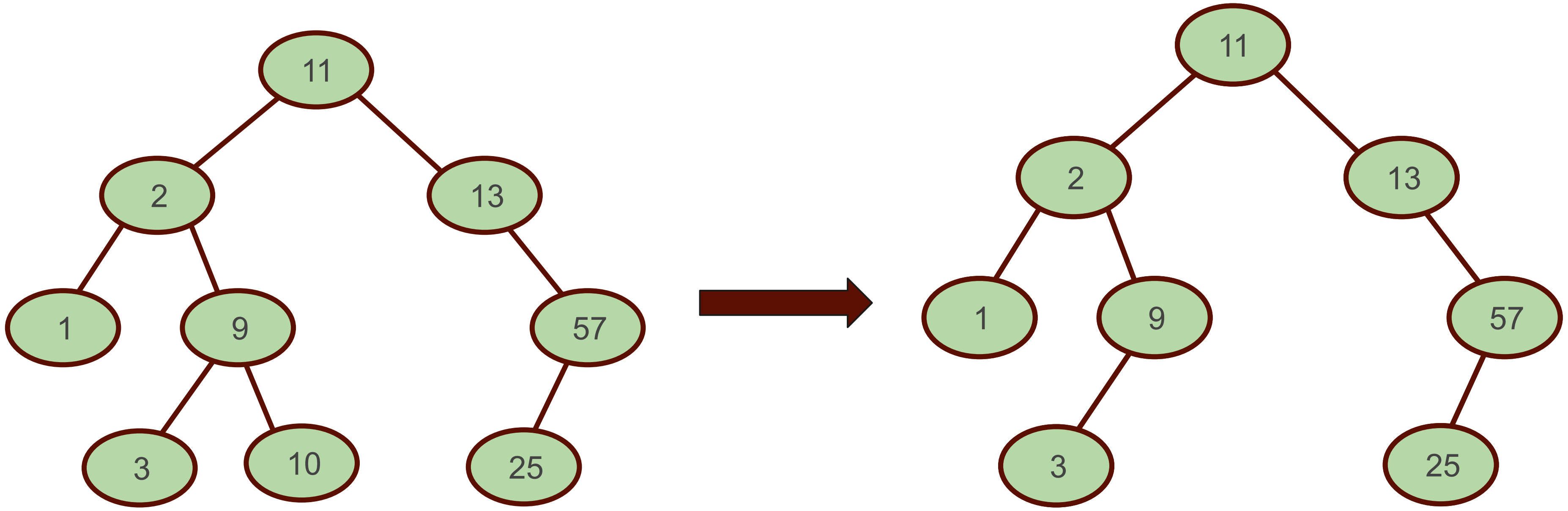
Case 3 : Node to be deleted has 2 children



Deletion in a BST!

$O(h)$

Case 1 : Node to be deleted is a leaf

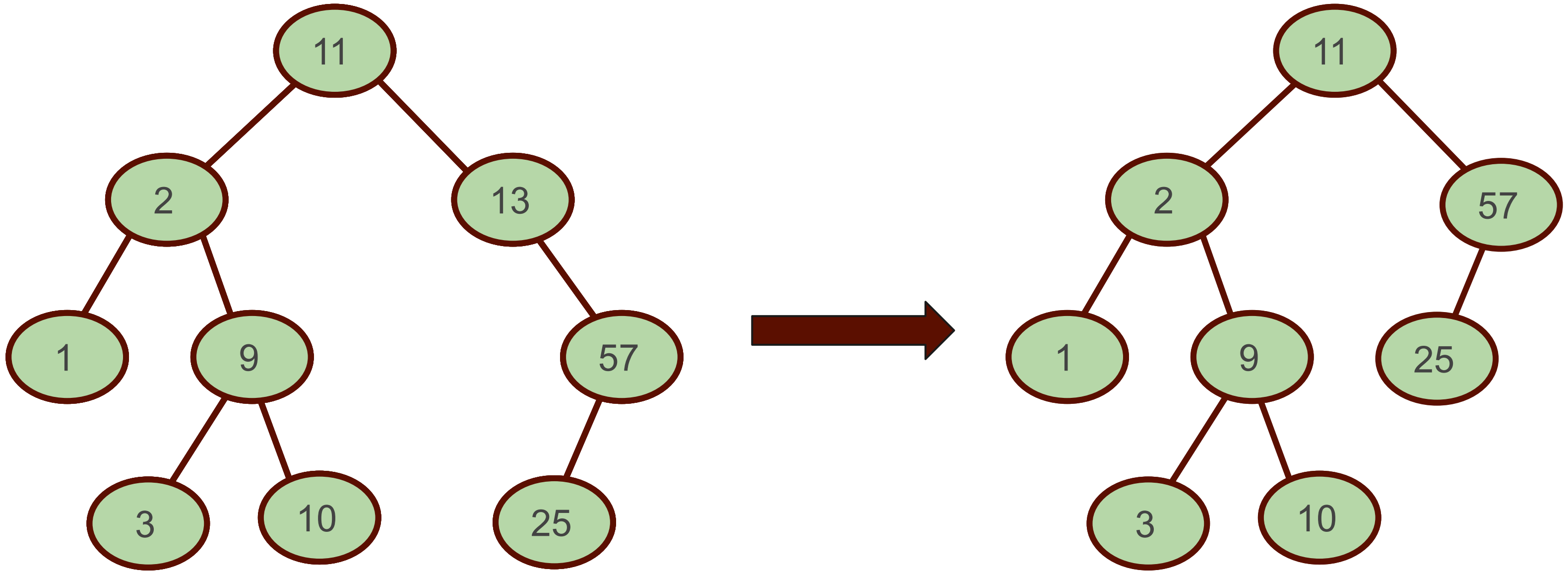


Simply delete the node from the tree!

Deletion in a BST!

$O(h)$

Case 2 : Node to be deleted has 1 child



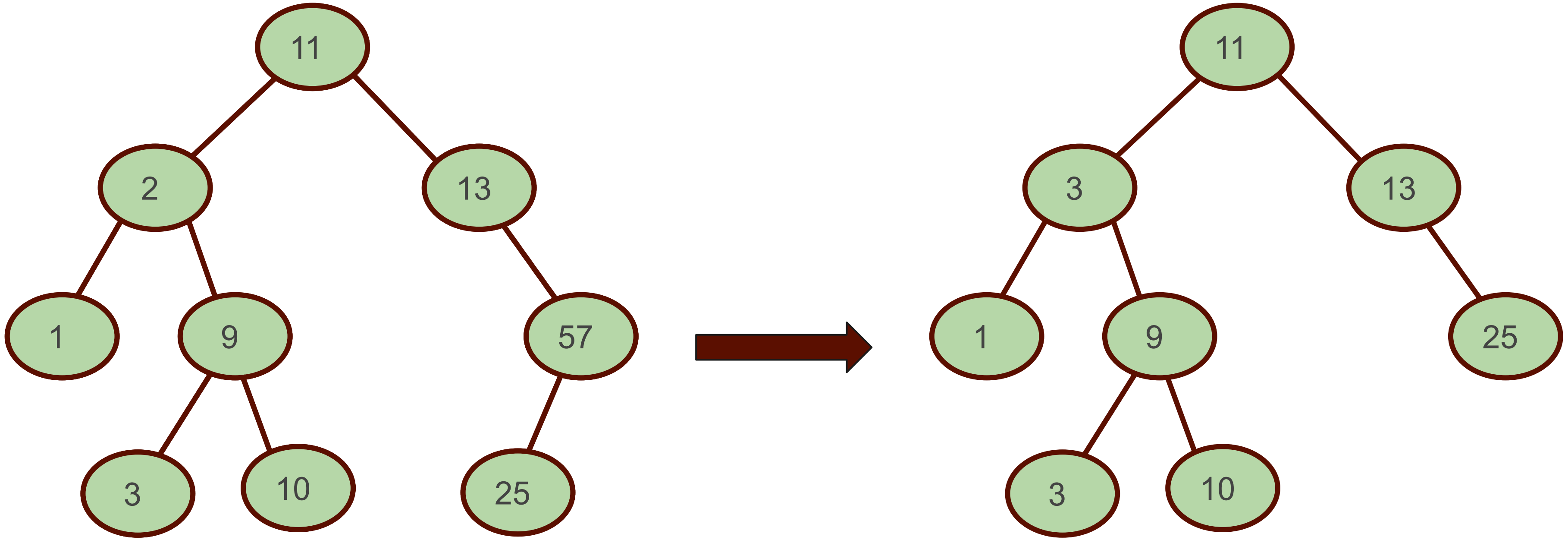
Attach the child of node to its parent!

Delete 2

Deletion in a BST!

$O(h)$

Case 3 : Node to be deleted has 2 children



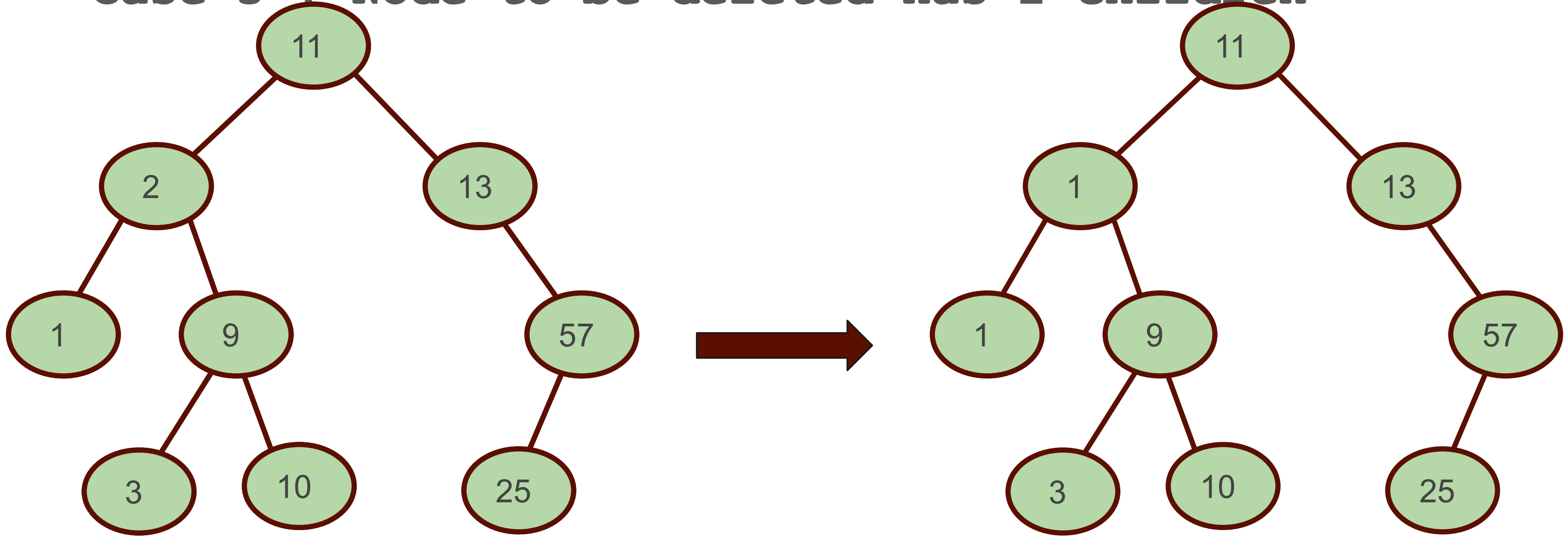
The node is replaced with the inorder successor (smallest element in the right subtree) recursively until the node to be deleted is placed on the leaf, finally delete the leaf!

Delete 2

Deletion in a BST!

O(h)

Case 3 : Node to be deleted has 2 children



The node is replaced with the inorder predecessor(largest element in the left subtree) recursively until the node to be deleted is placed on the leaf, finally delete the leaf!

Thank You!