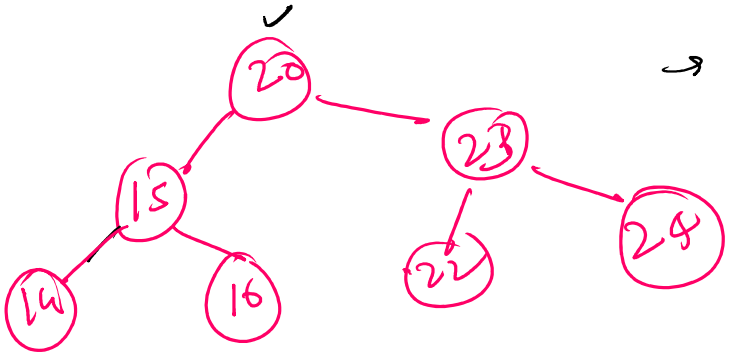
# 4.1 Binary Search Trees

Sunday, August 24, 2025 10:43 AM → Module 4

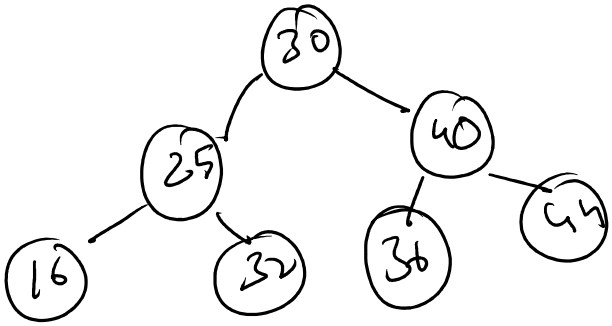
Special case of a Binary Tree

⇒ A BST is a binary tree where every node follow this property

- The left subtree will have smaller values than the node
- The right subtree has greater values than the node.

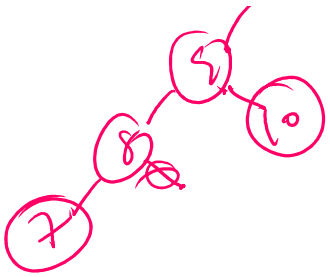



→ BST



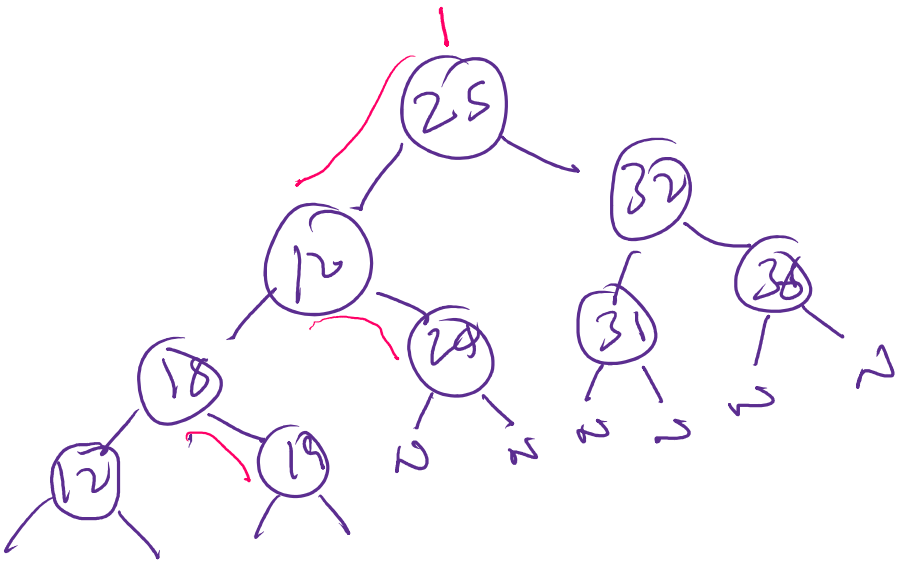

→ Not a BST


→ skewed BST




⇒ Some properties of BST

- The smallest node in a BST is the left most deepest node.
- The largest node in a BST is the right most deepest node.




Pre: 25 12 18 12 19 24 32 31 36

In: 12 18 19 25 24 25 31 32 36

- Inorder of a BST is always sorted.

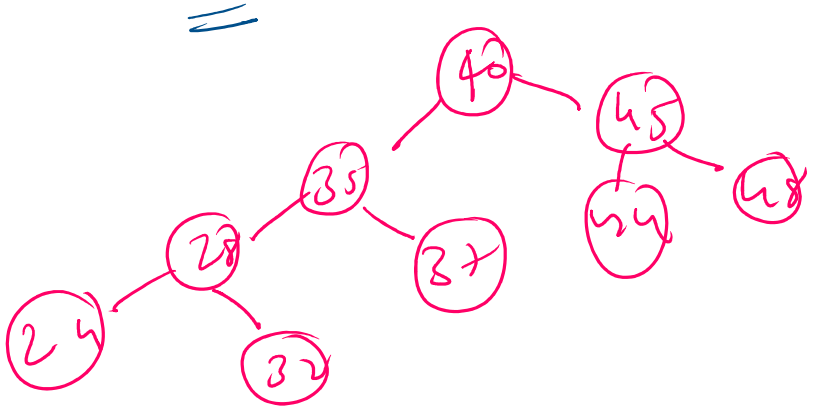
```
void inorder ( Node root ) {
    if (root == null) return;
    inorder ( root.left);
    cout (root.data);
    inorder (root.right);
}
```

⇒ BSTs are used to make search efficient

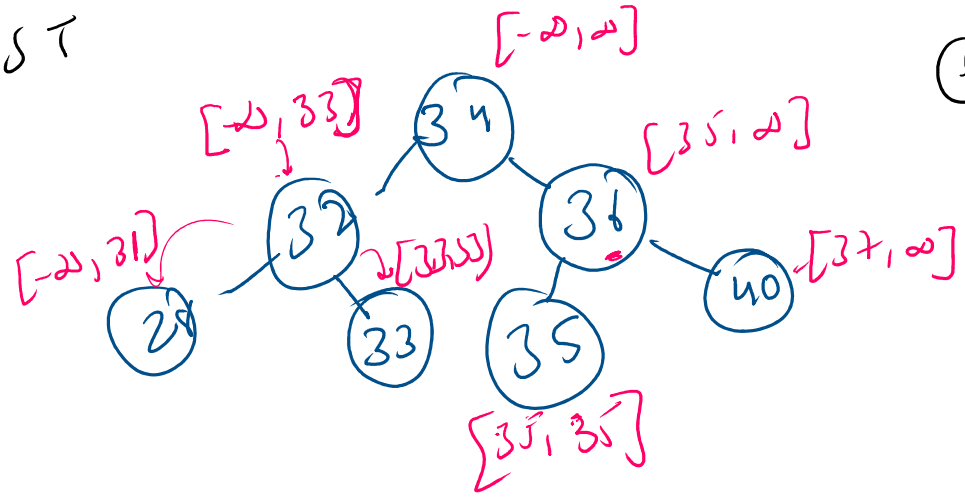
→ complexity of searching in a BST is $O(\log_2 N)$.

→ keys in a BST are unique unless specified.

# Insert a Node in BST



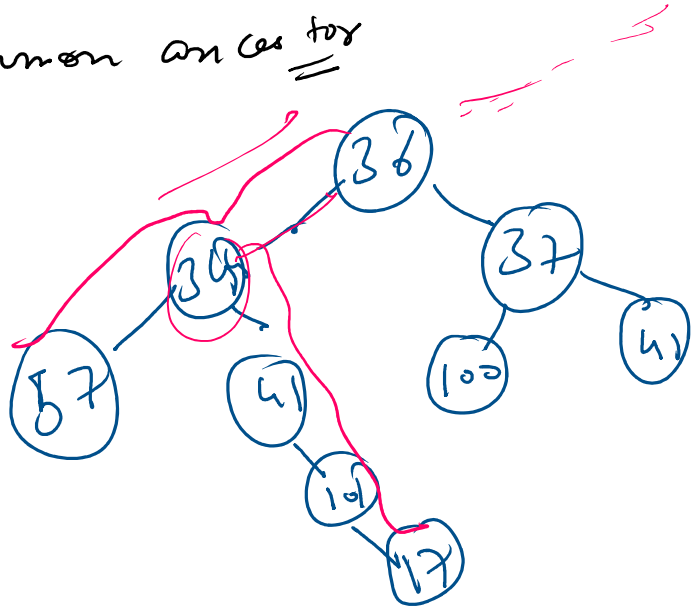
# Validate BST




# LCA
↳ lowest common ancestor

→ [36, 34, 57]

→ [36, 34, 41, 101, 17]




```
Node lca(Node root, int n1, int n2) {
    if(root == null) ret root;
    if(root.data == n1 || root.data == n2) ret root;
    Node left = lca(root.l, n1, n2)
    Node right = lca(root.r, n1, n2);
    if(left != null) ret root;
```

Node right = leaf node;

if(left != null && right) = null) return root;

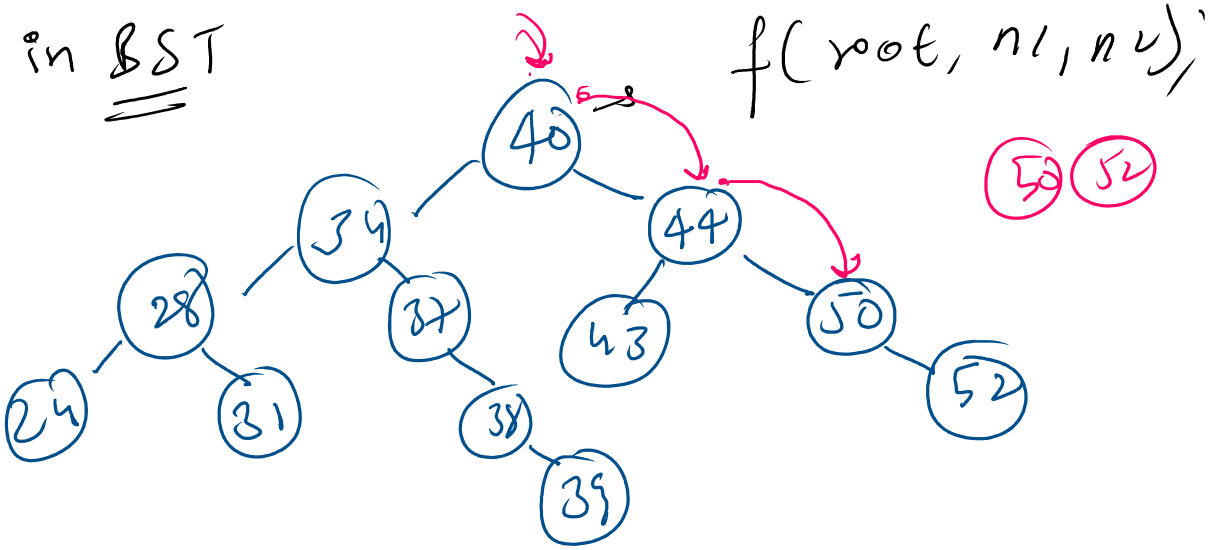
if(right == null) return left;

return right;

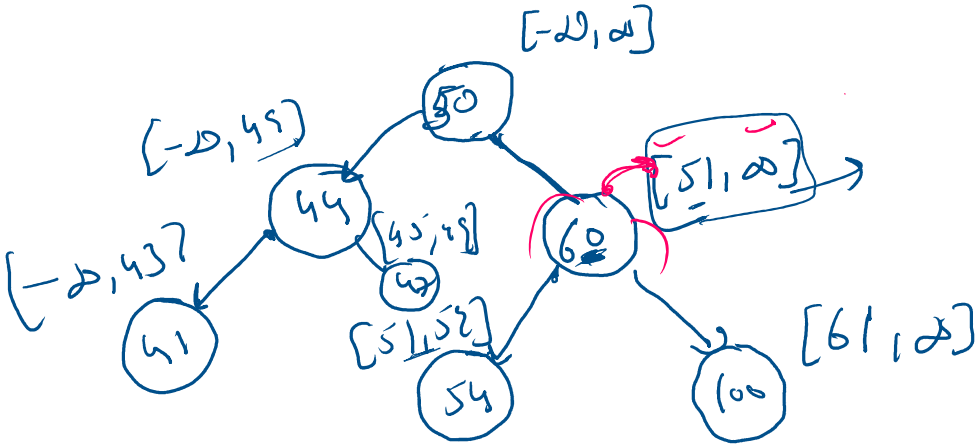
↓

## # LCA in BST

f(root, n1, n2);




50 52

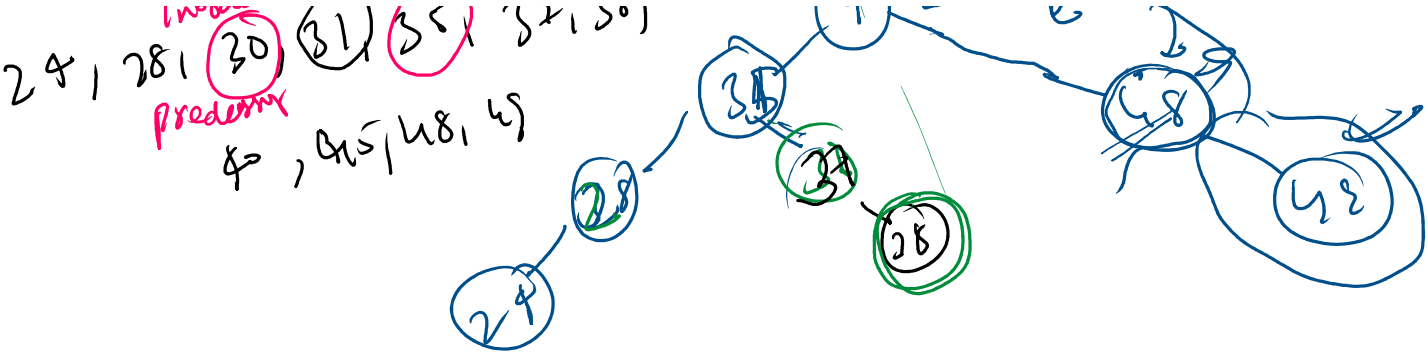



## # Delete a Node in BST

inorder successor

inorder

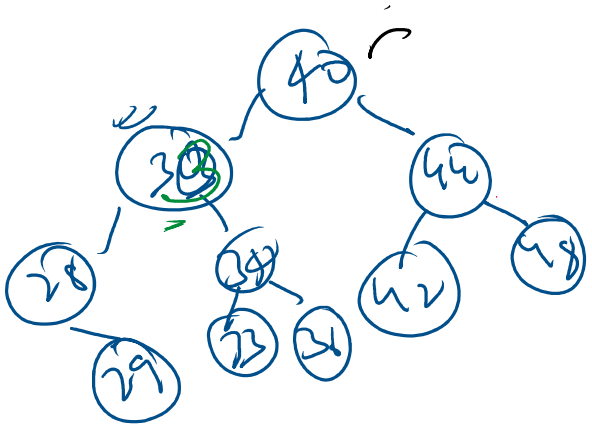
24, 28, 30, 31, 35, 37, 38



Case I Node is leaf → remove

Case II left == null → node.right

Case III right == null → node.left

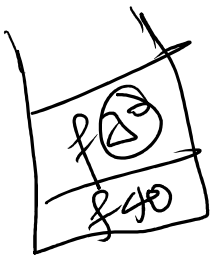


```
if(root==null) return null;

if(val < root.val) {
    root.left = deleteNode(root.left,val);
}
else if(val > root.val) {
    root.right = deleteNode(root.right,val);
}
else {
    if(root.left==null) return root.right;
    if(root.right==null) return root.left;
    Node inorderSucc =
findMin(root.right);
    root.val = inorderSucc.val;

deleteNode(root.right,inorderSucc.val);
}

return root;
```



# Target Sum Pairs

# Target Sum Pairs

Pre → Let-rec