

■ Introduction

■ Approach 1 — HashSet + DFS

■ Approach 2 — Inorder + Two Pointers

■ Java Implementation

```

while (left < right) {
    int sum = sorted.get(left) + sorted.get(right);
    if (sum == k) return true;
    else if (sum < k) left++;
    else right--;
}
return false;
}

private void inorder(TreeNode node, List<Integer> list) {
    if (node == null) return;
    inorder(node.left, list);
    list.add(node.val);
    inorder(node.right, list);
}
}

```

■ Time & Space Complexity

Case	Time	Space	Why
HashSet DFS	$O(n)$	$O(n)$	Visit all nodes, HashSet stores all values
Inorder + 2ptr	$O(n)$	$O(n)$	Inorder $O(n)$, two pointer $O(n)$

■ Common Mistakes

Avoid These

- Using the same node twice — "two distinct nodes" means different nodes (even if same value).
- Forgetting $k - \text{root.val} == \text{root.val}$ case — node cannot pair with itself.
- Treating it like a sorted array Two Sum — BST is not a flat array, traverse fully.

■ Practice Problems

Case	Time	Space	Why
Easy	LC 653	Two Sum IV Input is BST	Direct application
Medium	LC 1214	Two Sum BSTs	Two separate BSTs
Hard	LC 15	3Sum	Extend two pointer idea

■ Mental Model

Remember Forever

- "BST gives you sorted order for free via inorder. Any sorted-array algorithm (binary search, two pointers) can be applied after inorder traversal."
- HashSet approach is more general — works even on non-BST binary trees.

■ BST — Overall Quick Revision

- BST Property: $\text{left} < \text{root} < \text{right}$ at EVERY subtree level
- Search/Insert/Delete: all $O(\log n)$ average, $O(n)$ worst (skewed)
- Validate: pass min/max bounds down; or check inorder is strictly increasing
- Inorder BST = sorted array — use for Kth smallest, Two Sum, range queries
- LCA in BST: walk down, divergence point is the answer — $O(h)$ no extra space
- Delete 3 cases: leaf, one child, two children (inorder successor trick)
- Always use Long bounds in validation to handle Integer edge values