📓 Find Mode in Binary Search Tree – Complete Documentation

1. Problem Statement

Given the root of a Binary Search Tree (BST) where duplicates are allowed, return all mode(s)—the element(s) that appear most frequently in the tree.

If multiple values have the same highest frequency, return them in any order.

🔒 Constraints:

- Number of nodes: $1 \leq n \leq 10^4$
- Node values: $-10^5 \leq val \leq 10^5$

2. Intuition

In a BST, an in-order traversal gives values in sorted (non-decreasing) order. This property can be used to detect and count consecutive duplicates efficiently—without storing all node values.

3. Key Observations

- In-order traversal ensures values are visited in sorted order.
- Duplicate values appear consecutively during traversal.
- You can track the frequency of current values on-the-fly by comparing them with the previous value.
- Modes are values with maximum frequency, so track the max frequency and update the result list accordingly.

4. Approach

- Initialize:
    - current_val: current node value during traversal.

    - current_count: count of the current value.
    - max_count: highest frequency seen so far.
    - modes: list of mode values.
- Use a recursive in-order traversal:
    - Visit left subtree.
    - Process current node:
        - Compare it to current_val.
        - Update current_count accordingly.
        - If current_count > max_count, reset modes list.
        - If current_count == max_count, append to modes.
    - Visit right subtree.
- Return the modes list.

5. **Edge Cases**

- Tree with only one node → Return that node's value.
- All nodes have the same value → Return that value.
- Every node has a unique value → Return all values.
- Tree has multiple modes → Return all modes.

6. **Complexity Analysis**

✅ Time Complexity

- O(n), where n is the number of nodes. Each node is visited exactly once during in-order traversal.

✅ Space Complexity

- O(h), where h is the height of the tree (recursion stack).
    - For balanced BST: O(log n)
    - For skewed BST: O(n)
- No extra space used apart from recursion stack (meets follow-up requirement).

7. Alternative Approaches

🔹 A. Using Hash Map (Extra Space)

- Use DFS to count the frequency of each value using a dictionary.
- Identify the maximum frequency and return all values that match it.

Pros: Simple to implement.
Cons: Uses O(n) extra space.

🔹 B. Morris In-order Traversal (O(1) Space)

- Modify tree structure temporarily to perform in-order traversal without recursion or stack.
- Track values similar to current approach.

Pros: Uses constant space.
Cons: More complex and error-prone.

8. Test Cases

Input BST (Level Order)	**Expected Output**	**Explanation**
[1,null,2,2]	[2]	2 appears twice, most frequent.
[0]	[0]	Only one value.
[2,1,3]	[1,2,3]	All values appear once.
[1,1,2,null,null,2,2]	[2]	2 appears 3 times, others twice.
[1,null,2,null,2,null,2]	[2]	Skewed right, 2 appears 3 times.

9. Final Thoughts

- This solution utilizes in-order traversal to exploit BST properties and detect modes efficiently without using extra space.
- By maintaining a simple counter and comparing consecutive values, we can easily identify the most frequent elements in one traversal.
- This is an elegant and optimal approach for solving the problem within the given constraints.