Binary Tree: DFS (Recursive) — Coding Interview Notes (Light Theme)

General Pattern Template

```
def dfs(root):
    if not root:
        return

    ans = 0

    # do logic
    dfs(root.left)
    dfs(root.right)
    return ans
```

Concept:
The **Depth-First Search (DFS)** recursive pattern is the most common way to traverse or process nodes in a binary tree. It explores as far as possible along each branch before backtracking. DFS can be used for *preorder, inorder,* and *postorder* traversals, and for solving problems involving recursion on subtrees.

Time Complexity: O(n) **Space Complexity:** O(h), where h is tree height (stack depth).

Key Ideas

1. Recursive DFS processes each node once (O(n)).
2. The order of recursive calls determines traversal type (pre/in/post).
3. State can be accumulated (return value) or modified via outer scope (global/closure).
4. Use recursion for clarity and when the tree depth is not excessive.

Example 1: Preorder Traversal

Goal: Visit nodes in the order: Root → Left → Right.
Approach: Process the current node before visiting its children.

```
def preorder(root):
    if not root:
        return []
    return [root.val] + preorder(root.left) + preorder(root.right)
```

Example 2: Inorder Traversal

Goal: Visit nodes in the order: Left → Root → Right.
Approach: Useful for retrieving sorted values from a Binary Search Tree.

```
def inorder(root):
    if not root:
        return []
    return inorder(root.left) + [root.val] + inorder(root.right)
```

Example 3: Postorder Traversal

Goal: Visit nodes in the order: Left → Right → Root.
Approach: Often used for deletion, height computation, or evaluating expressions in trees.

```
def postorder(root):
    if not root:
        return []
    return postorder(root.left) + postorder(root.right) + [root.val]
```

Example 4: Compute Sum of All Node Values

Goal: Return the sum of all values in a binary tree.
Approach: Use DFS recursion to accumulate sums from left and right subtrees.

```
def tree_sum(root):
    if not root:
        return 0
    return root.val + tree_sum(root.left) + tree_sum(root.right)
```

Summary Table

Traversal TypeOrderUse Case PreorderRoot → Left → RightClone tree, expression prefix InorderLeft → Root → RightRetrieve sorted order (BST) PostorderLeft → Right → RootDelete tree, compute height Custom DFSFlexible logic placementSum, path, diameter, etc.