```python
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    #二叉树的前序遍历：递归（DFS：根-左-右）
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def helper(root):
            if not root: return #（叶子）节点为空即可返回
            res.append(root.val)
            helper(root.left)
            helper(root.right)
        helper(root)
        return res

    #二叉树的前序遍历：迭代
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        if not root: return []
        deque = collections.deque()
        deque.append(root)
        res = []
        while deque:
            for _ in range(len(deque)):
                node = deque.pop()
                res.append(node.val)
                if node.right: deque.append(node.right)
                if node.left: deque.append(node.left)
        return res
```

```python
    #二叉树的中序遍历：递归（DFS：左-根-右）
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def helper(root):
            if not root: return
            helper(root.left)
            res.append(root.val)
            helper(root.right)
        helper(root)
        return res

    #来一个超级吊的写法：
    def helper(root):
            return helper(root.left) + [root.val] + helper(root.right) if root else []

    #二叉树的中序遍历：迭代
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        stack = []
        while root or stack:
            while root:
                stack.append(root)
                root = root.left
            root = stack.pop()
            res.append(root.val)
            root = root.right
        return res
```

```python
    #二叉树的后序遍历（DFS：左-右-根）
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def helper(root):
            if not root: return
            helper(root.left)
            helper(root.right)
            res.append(root.val)
        helper(root)
        return res

    #二叉树的后序遍历：迭代
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if not root: return []
        deque = collections.deque()
        deque.append(root)
        res = []
        while deque:
            for _ in range(len(deque)):
                node = deque.pop()
                res.append(node.val)
                if node.left: deque.append(node.left)
                if node.right: deque.append(node.right)
        return res[::-1]
```

```python
    #二叉树的层序遍历（DFS）
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        res = []
        def helper(cur_layer, root):
            if not root: return []
            if len(res) < cur_layer: res.append([])
            res[cur_layer-1].append(root.val)
            if root.left: helper(cur_layer+1, root.left)
            if root.right: helper(cur_layer+1, root.right)
        helper(1, root)   #一般来说，树的顶层从第1层开始，不是第0层
        return res

    #二叉树的层序遍历（BFS）
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root: return []
        deque = collections.deque()
        deque.append(root)
        res = []
        while deque:
            cur_layer = []
            for _ in range(len(deque)):
                node = deque.popleft()
                cur_layer.append(node.val)
                if node.left: deque.append(node.left)
                if node.right: deque.append(node.right)
            res.append(cur_layer)
        return res
```