递归：

二叉树：

1 前序遍历 --根-左-右

```
void preorder(struct TreeNode *root, int* returnSize, int *res){
    if(!root)
        return;
    //根
    res[(*returnSize)++]= root->val;
    // 左
    preorder(root->left, returnSize, res);
    // 右
    preorder(root->right, returnSize, res);
}
```

N叉树的前序遍历，左右都是子节点

```
void preorder(struct TreeNode *root, int* returnSize, int *res){
    if(!root)
        return;

    //根
    res[(*returnSize)++]= root->val;
    // 子
    for (int i = 0; i < root->numChildren; i++)
      visit(root->children[i], result, returnSize);
}
```

2 中序遍历：左-根-右

```
void inorder(struct TreeNode *root, int* returnSize, int *res){
    if(!root)
        return;
    // 左
    inorder(root->left, returnSize, res);
    //根
    res[(*returnSize)++]= root->val;
    // 右
    inorder(root->right, returnSize, res);
}
```

3 后序遍历：左右根

```
void postorder(struct TreeNode *root, int* returnSize, int *res){
    if(!root)
        return;
    // 左
    postorder(root->left, returnSize, res);
    // 右
    postorder(root->right, returnSize, res);
    //根
    res[(*returnSize)++]= root->val;
}
```

N叉树的后序遍历，左右都是子节点

```
void* postvisit(struct Node* root, int* returnSize, int* res)
{
    if(!root)
        return;
    //子
    for(int i =0; i< root->numChildren; i++)
        postvisit(root->children[i], returnSize, res);
    res[(*returnSize)++] = root->val;
}
```

调用时：

```
#define MAX_SIZE 10240
int* postorderTraversal(struct TreeNode* root, int* returnSize){
    int* res = (int*)malloc(MAX_SIZE * sizeof(int));
    *returnSize =0;
    if (root == NULL)
        return;
    postvisit(root, returnSize, res);
    return res;
}
```

迭代：（栈操作，难点在于压站的顺序）

二叉树的前序遍历 根-左-右

递归时根，左，右，那么栈就是右，左，根， 所以，如果是root：res第一个值时root，右子树入栈，左子树成为的当前的根，如果没有根了，也是没有左子树了，那就出栈，开始处理右子树

```
def preorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        res = []
        stack = []
        stack.append(root)
        while stack:
             if root:
                 res.append(root.val)
                 stack.append(root.right)
                 root = root.left
             else:
                 root = stack.pop()
         return res
```

N叉树的前序遍历：

```
def preorder(self, root):
        """
        :type root: Node
        :rtype: List[int]
        """
        if not root:
            return []
        res=[root.val]
        for node in root.children:
             res.extend(self.preorder(node))
         return res

         res = []
         def pre(root):
             if not root:
                 return
             res.append(root.val)
             for child in root.children:
                 pre(child)
         pre(root)

```

```
        # 迭代:
        if not root:
            return []
        s = [root]
        res = []
        while s:
            node = s.pop()
            res.append(node.val)
            s.extend(node.children[::-1])
        return res
```

二叉树的中序遍历--左，根，右。递归的顺序是左，根，右，那么入栈的顺是右，根，左，但是，由于第一个是根且根据根能找到右，那么只需要把根，左成为新的根。为空时，说明左子树处理完了，输出根，加到res里，然后时根是右子树。

```
def inorderTravode
        :rtype: List[int]
        """
        res =[]
        stack = []
        while stack or root:
            #入栈: 根左
            if root:
                stack.append(root)
                 root = root.left
             else:
             #入栈结束，输出
                 # 左
                 tmp = stack.pop()
                 #根
                 res.append(tmp.val)
                 # 右
                 root = tmp.right
         return res
```

二叉树的后序遍历是左，右，根，前序遍历是： 根，左，右，
前序遍历是：根左右根左右根左右
后序遍历是：左右根左右根左右根
所以

1 前序的左右互换：根右左根右左根右左
2 在逆过来：左右根左右根左右根
所以只需要把前序遍历的方法压栈，前序的左--> 后序的右，前序的右-->后序的左
在将结果逆回来

```
def postorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        res =[]
        stack = []
        while stack or root:
            if root :
                res.append(root.val)
                stack.append(root.left)
                root = root.right
             else:
                root = stack.pop()
         return res[::-1]

```

N叉树的后续遍历：

```
def postorder(self, root):
        """
        :type root: Node
        :rtype: List[int]
        """
        if not root:
            return []
        s = [root]
        res = []
         while s:
             node = s.pop()
             res.append(node.val)
             s.extend(node.children) #这里和前序遍历逆一次
         return res[::-1]   # 这里再和前序遍历逆一次

```

leetcode：

144:二叉树的前序遍历
94: 二叉树的中序遍历
145:二叉树的后序遍历
589: N叉树的前序遍历
590: N叉树的后序遍历