814. Binary Tree Pruning

We will traverse our binary tree recursively in post order. TC is O(n), SC is O(1)

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {TreeNode}
 */
var pruneTree = function(root) {
    function postTraverse(node) {
        let left, right;
        if (!node) {
            return null;
        }
        if (node.left) {
            left = postTraverse(node.left)
        }

        if (node.right) {
            right = postTraverse(node.right)
        }

        if (!left) {
            node.left = null;
        }
        if (!right) {
            node.right = null;
        }

        return node.val === 1 || left || right;
    }
    const dummy = new TreeNode(1);
    dummy.left = root;
    postTraverse(dummy);
    return dummy.left;
};
```

112. Path Sum

We will traverse our tree by pre-order and check whether there is a node meeting our requirement. TC is O(n), SC is O(n)

```
var hasPathSum = function(root, sum) {
  function traverse(node, cur_sum) {
    if (!node) {
      return false;
    }
    if (cur_sum + node.val === sum && !node.left && !node.right) {
      return true;
    }
    if (traverse(node.left, cur_sum + node.val) || traverse(node.right, cur_sum + node.val)) {
      return true;
    } else {
      return false;
    }
  }
  return traverse(root, 0)
};
```

124. Binary Tree Maximum Path Sum

We will add max values from left and right to current node value, then compare the current sum with our max_val. Then return max(left, right) + current.val. TC is O(n), SC is O(n)

```
var maxPathSum = function(root) {
  let max_val = -Number.MAX_VALUE;
  function traverse(node) {
    if (!node) {
      return 0;
    }
    let left = Math.max(traverse(node.left), 0);
    let right = Math.max(traverse(node.right), 0);
    max_val = Math.max(left + node.val + right, max_val);
    return Math.max(left, right) + node.val;
  }
  traverse(root);
  return max_val;
};
```

129. Sum Root to Leaf Numbers

We will traverse from root to leaf. Add it to our resut. TC is O(n), SC is O(n)

```
var sumNumbers = function(root) {
  let result = 0
  function postTraverse(node, cur) {
    if (!node) {
```

```
      return;
    }
    if (!node.left && !node.right) {
      result += cur * 10 + node.val;
    } else {
      if (node.left) {
        postTraverse(node.left, cur * 10 + node.val);
      }
      if (node.right) {
        postTraverse(node.right, cur * 10 + node.val);
      }
    }
  }
  postTraverse(root, 0);
  return result;
};
```

236. Lowest Common Ancestor of a Binary Tree

We will check left branch and right branch whether there is a target node. If there is, we will return the current node, or will return left or right node from branches. TC is O(n), SC is O(n)

```
var lowestCommonAncestor = function(root, p, q) {
    function traverse(node) {
      if (!node) {
        return null;
      }
      let left = traverse(node.left);
      let right = traverse(node.right);
      if (left && right) {
        return node;
      } else if (node === p || node === q) {
        return node;
      } else if (left || right) {
        return left || right;
      }
      return null;
    }
    return traverse(root);
};
```