437. Path Sum III
We will use dfs, pre-sum plus dict to record all presum and if accumulate all dict[cur_sum - sum], TC is O(n)

```
from collections import defaultdict

class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> int:
        self.result = 0
        memo = defaultdict(int)
        memo[0] = 1
        def helper(cur_sum, node):
            if not node:
                return
            cur_sum += node.val
            self.result += memo[cur_sum - sum]
            memo[cur_sum] += 1
            if node.left:
                helper(cur_sum, node.left)
            if node.right:
                helper(cur_sum, node.right)
            memo[cur_sum] -= 1
        helper(0, root)
        return self.result
```

1202. Smallest String With Swaps
We could use union find and sort letters in the same group.

```
from collections import defaultdict
class Solution:
    def smallestStringWithSwaps(self, s: str, pairs: List[List[int]]) -> str:
        arr_s = list(s)
        parents = list(range(len(s)))
        teams = defaultdict(list)
        team_members = set()
        def getParent(a):
            b = a
            while parents[a] != None and parents[a] != a:
                a = parents[a]
            parents[b] = a
            return a

        for a, b in pairs:
            p_a = getParent(a)
            p_b = getParent(b)
            if p_a != p_b:
                parents[p_b] = p_a
            team_members.add(a)
            team_members.add(b)

        for i in team_members:
            p = getParent(i)
            teams[p].append(i)



        for arr in teams.values():
            result = []
            arr.sort()
            for i in arr:
                result.append(arr_s[i])
            result.sort()
            j = 0
            for i in arr:
                arr_s[i] = result[j]
                j += 1
        return ''.join(arr_s)
```

1201. Ugly Number III
We will let the min one go to the point only less than the second largest by one more step.

```
class Solution:
    def nthUglyNumber(self, n: int, a: int, b: int, c: int) -> int:
        dp_a = 1
        dp_b = 1
        dp_c = 1
        i = 0
        a, b, c = sorted([a, b, c])
        while i < n:
            result = min(dp_a * a, dp_b * b, dp_c * c)
            if result == dp_a * a:
                d_a = (min(dp_b * b, dp_c * c) - dp_a * a) // a
                d_a = max(d_a, 1)
                if i + d_a >= n:
                    return result + (n - i - 1) * a
                else:
                    i += d_a
                    dp_a += d_a
            else:
                i += 1
            if result == dp_b * b:
                dp_b += 1
            if result == dp_c * c:
                dp_c += 1
        return result
```

1200. Minimum Absolute Difference
We will sort the array and append the pair with min difference to result, or we will empty the result. TC is O(n)

```
class Solution:
    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
        result = []
        min_dif = float('inf')
        arr.sort()
        for i in range(1, len(arr)):
            if arr[i] - arr[i - 1] < min_dif:
                result = [[arr[i - 1], arr[i]]]
                min_dif = arr[i] - arr[i - 1]
            elif arr[i] - arr[i - 1] == min_dif:
                result.append([arr[i - 1], arr[i]])
        return result
```

333. Largest BST Subtree
We will iterate from bottom to top. For every node, we will return whether is valid, nodes number, max, min to the upper layer. TC is O(n)

```
class Solution:
    def largestBSTSubtree(self, root: TreeNode) -> int:
        def helper(node):
            if not node:
                return (True, 0, float('inf'), -float('inf'))
            left = helper(node.left)
            right = helper(node.right)
            if left[0] and right[0]:
                if (not node.left or node.val > left[3]) and (not node.right or node.val < right[2]):
                    return (True, left[1] + right[1] + 1, min(left[2], right[2], node.val), max(left[3], right[3],
node.val))
                else:
                    return (False, max(left[1], right[1]), min(left[2], right[2], node.val), max(left[3],
right[3], node.val))
            else:
                return (False, max(left[1], right[1]), min(left[2], right[2], node.val), max(left[3], right[3],
node.val))
```

```
return helper(root)[1]
```