LEETCODE PROBLEMS:

810. Faulty Keyboard

```
class Solution:
    def finalString(self, s: str) -> str:
        l=""
        for i in s:
            if i!="i":
                l+=i
            else:
                l=l[::-1]
        return l
```

832. Check the given string is PANGRAM or not

```
class Solution:
    def checkIfPangram(self, sentence: str) -> bool:
        a="abcdefghijklmnopqrstuvwxyz"
        for i in a:
            if i not in sentence:
              return False
        else:
          return True
```

557. Reverse words in a string III

```
class Solution:
    def reverseWords(self, s: str) -> str:
        a=""
```

```
        l=s.split(" ")
        for i in range(len(l)):
            if i==len(l)-1:
                a=a+l[i][::-1]
            else:
                a=a+l[i][::-1]+" "
        return a
```

328. Check if a String is an Acronym or not

```
class Solution:
    def isAcronym(self, words: List[str], s: str) -> bool:
       if len(words)!=len(s):
            return False
       else:
            for i in range(len(s)):
                if s[i]!=words[i][0]:
                    return False
       return True
```

804. Unique Morse Code Words

```
class Solution:
    def uniqueMorseRepresentations(self, words: List[str]) -> in
        n=[".-","-...","-.-.","-..",".","..-.","--.","....","..
        a="abcdefghijklmnopqrstuvwxyz"
        p=[]
        ans=[]
        for q in words:
            b=""
            for i in q:
                b+=n[a.index(i)]
            p.append(b)
```

```
        for q in p:
            if q not in ans:
                ans.append(q)
        return len(ans)
```

2315. Count Asterisks

```
class Solution:
    def countAsterisks(self, s: str) -> int:
        n=""
        c=0
        t=0
        for i in s:
            if i=="|":
                t=t+1
            elif t%2==0:
                c+= i=="*"
        return c
```

2744. Maximum number of String pairs

```
class Solution:
    def maximumNumberOfStringPairs(self, words: List[str]) -> in
        c=0
        for i in range(len(words)):
            for j in range(i+1,len(words)):
                if words [i][0]==words[j][1] and words[i][1]==wo
                    c+=1
        return c
```

2967. Number of strings that appear as substrings

```
class Solution:
    def maximumNumberOfStringPairs(self, words: List[str]) -> in
        c=0
        for i in range(len(words)):
            for j in range(i+1,len(words)):
                if words [i][0]==words[j][1] and words[i][1]==wo
                    c+=1
        return c
```

1919. Count the key Changes

```
class Solution:
    def countKeyChanges(self, s: str) -> int:
        n=s.lower()
        c=0
        for i in range(len(n)-1):
            if n[i]!=n[i+1]:
                c+=1
        return c
```

344. Reverse String

```
class Solution:
    def reverseString(self, s: List[str]) -> None:
        s[:]=s[::-1]
```

2710. Score of a String

```
class Solution:
    def scoreOfString(self, s: str) -> int:
        a=0
        for i in range(1,len(s)):
            a+=abs(ord(s[i])-ord(s[i-1]))
        return a
```

18. Sort the People

```
class Solution:
    def sortPeople(self, names: List[str], heights: List[int]) -
        a=[]
        for i in range(len(names)):
            a.append([heights[i],names[i]])
        a.sort(reverse=True)
        b=[]
        for i in range(len(names)):
            b.append(a[i][1])
        return b
```

1. Two Sum

```
class Solution:
    def twoSum(self, nums, target):
        a=[]
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                if nums[i]+nums[j]==target:
                    a.append(i)
                    a.append(j)
```

```
                    return a
        return(twosum(nums,target))
```

58. Length of Last Word

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        l=s.split()
        a=len(l[-1])
        return a
```

59. Sorting the sentence

```
class Solution:
    def sortSentence(self, s: str) -> str:
        l=s.split(" ")
        d={}
        for i in l:
            d[i[-1]]=i[:-1]
        s=""
        for i in range(1,len(l)+1):
            if i==len(l):
                s=s+d[str(i)]
            else:
                s=s+d[str(i)]+" "
        return s
```

20. Valid Parenthesis

```
class Solution:
    def isValid(self, s: str) -> bool:
```

```
        a=[]
        for i in s:
            if i=="(" or i=="{" or i=="[":
                a.append(i)
            elif (i==")" or i=="}" or i=="]")and len(a)==0:
                return False
            else:
                if i==")" and a[-1]=="(":
                    a.pop()
                elif i=="}"and a[-1]=="{":
                    a.pop()
                elif i=="]" and a[-1]=="[":
                    a.pop()
                else:
                    a.append(i)
        if len(a)==0:
            return True
        else:
            return False
```

29. Concatenation of Array

```
class Solution:
    def getConcatenation(self, nums: List[int]) -> List[int]:
        return 2*nums
```

70. Shuffle the Array

```
class Solution:
    def shuffle(self, nums: List[int], n: int) -> List[int]:
        n=[]
        h=len(nums)//2
```

```
        for i in range(len(nums)//2):
            n.append(nums[i])
            n.append(nums[i + h])
        return n
```

572. Richest Customer Wealth

```
class Solution:
    def maximumWealth(self, accounts: List[List[int]]) -> int:
        maxw=0
        for i in range(len(accounts)):
            t=sum(accounts[i])
            maxw=max(maxw,t)
        return maxw
```

1598. Number of employees who met the target

```
class Solution:
    def numberOfEmployeesWhoMetTarget(self, hours: List[int], ta
        c=0
        for i in hours:
            if i>=target:
                c+=1
        return c
```

1431. Kids with Greatest number of candies

```
class Solution:
    def kidsWithCandies(self, candies: List[int], extraCandies:
        maximum=max(candies)
        r=[]
```

```
        for i in range(len(candies)):
            t=candies[i]+extraCandies
            r.append(t>=maximum)
        return r
```

1724. Count pairs whose sum is less than target

```
class Solution:
    def countPairs(self, nums: List[int], target: int) -> int:
        c=0
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                if nums[i]+nums[j]<target:
                    c+=1
        return c
```

1480. Running Sum of 1D Array

```
class Solution:
    def runningSum(self, nums: List[int]) -> List[int]:
        for i in range(1,len(nums)):
            nums[i]+=nums[i-1]
        return nums
```

2974. Minimum number game

```
class Solution:
    def numberGame(self, nums: List[int]) -> List[int]:
        nums=sorted(nums)
        for i in range(0,len(nums),2):
```

```
            nums[i],nums[i+1]=nums[i+1],nums[i]
        return nums
```

88. Sum of Odd Length Subarrays

```
class Solution:
    def sumOddLengthSubarrays(self, arr: List[int]) -> int:
        r=0
        for i in range(len(arr)):
            for j in range(i,len(arr),2):
                r+=sum(arr[i:j+1])
        return r
```

232. Implement Queue using Stack

```
class MyQueue:

    def __init__(self):
        self.s1=[]
        self.s2=[]

    def push(self, x: int) -> None:
        self.s1.append(x)


    def pop(self) -> int:
        for i in range(len(self.s1)):
            self.s2.append(self.s1.pop())
        a= self.s2.pop()
        for i in range(len(self.s2)):
            self.s1.append(self.s2.pop())
        return a
```

```
    def peek(self) -> int:
        return self.s1[0]

    def empty(self) -> bool:
        if len(self.s1)==0:
            return True
        else:
            return False
```

25. Implement of Stack using Queues

```
class MyStack:

    def __init__(self):
        self.q1=[]
        self.q2=[]

    def push(self, x: int) -> None:
        self.q1.append(x)

    def pop(self) -> int:
        for i in range(len(self.q1)):
            self.q2.append(self.q1.pop())
        a=self.q2.pop(0)
        for i in range(len(self.q2)):
            self.q1.append(self.q2.pop())
        return a

    def top(self):
        return self.q1[-1]

    def empty(self):
        if len(self.q1)==0:
```

```
            return True
        else:
            return False
```

26. Remove Duplicates from the Sorted Array

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        s=set(nums)
        l=list(s)
        l.sort()
        for i in range(len(l)):
            nums[i]=l[i]
        return len(l)
```

58. Length of Last Word

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        c=0
        i=len(s)-1
        while i>=0 and s[i]==' ':
            i-=1
        while i>=0 and s[i]!=' ':
            c+=1
            i-=1
        return c
```

242. Valid Anagram

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        sorted_s=sorted(s)
        sorted_t=sorted(t)
        return sorted_s==sorted_t
```

258. Add Digits

```
class Solution:
    def addDigits(self, num: int) -> int:
        if num>=10:
            output=num//10+num%10
            if output<10:
                return output
            else:
                return self.addDigits(output)
        else:
            return num
```

876. Middle of the Linked List

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[L
        temp=head
        c=0
        while temp!=None:
            c+=1
```

```
            temp=temp.next
        temp=head
        for i in range(c//2):
            temp=temp.next
        return temp
```

206. Reverse Linked List

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional
        prev=None
        next=None
        curr=head
        while curr!=None:
            next=curr.next
            curr.next=prev
            prev=curr
            curr=next
        head=prev
        return head
```

21. Merge Two Sorted Lists

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
```

```
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Op
        curr1=list1
        curr2=list2
        curr3=None
        while curr1!=None and curr2!=None:
            if curr1.val<=curr2.val:
                newnode=ListNode(curr1.val)
                temp=curr3
                if temp==None:
                    curr3=newnode
                else:
                    while (temp.next!=None):
                        temp=temp.next
                    temp.next=newnode
                curr1=curr1.next
            else:
                newnode=ListNode(curr2.val)
                temp=curr3
                if temp==None:
                    curr3=newnode
                else:
                    while temp.next!=None:
                        temp=temp.next
                    temp.next=newnode
                curr2=curr2.next
        while curr1!=None:
            newnode=ListNode(curr1.val)
            temp=curr3
            if temp==None:
                curr3=newnode
            else:
                while temp.next!=None:
                    temp=temp.next
                temp.next=newnode
            curr1=curr1.next
```

```
        while curr2!=None:
            newnode=ListNode(curr2.val)
            temp=curr3
            if temp==None:
                curr3=newnode
            else:
                while temp.next!=None:
                    temp=temp.next
                temp.next=newnode
            curr2=curr2.next
        return curr3
```

60. Intersection Of Two Linked Lists

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListN
        a=headA
        b=headB
        c=0
        while a!=b:
            a=a.next
            b=b.next
            if a==None:
                a=headB
                c+=1
            if b==None:
                b=headA
```

```
                c+=1
            if c>=3:
                return None
        return b
```

234. Palindrome Linked List

```
# Definition for singly-linked list.
# class ListNode:
#     def _init_(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def isPalindrome(self, head: Optional[ListNode]) -> bool:
        curr=head
        newnode=ListNode(curr.val)
        a=newnode
        curr=curr.next
        while curr!=None:
            new=ListNode(curr.val)
            a.next=new
            curr=curr.next
            a=a.next
        prev=None
        next=None
        curr=head
        while curr!=None:
            next=curr.next
            curr.next=prev
            prev=curr
            curr=next
        head=prev

        curr=head
```

```
        a=newnode
        while curr!=None:
            if curr.val!=a.val:
                return False
            curr=curr.next
            a=a.next
        else:
            return True
```

83. Remove Duplicates From the List

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Opti
        curr=head
        next=None
        if head==None:
            return None
        while curr.next!=None:
            next=curr.next
            if curr.val!=next.val:
                curr=curr.next
            else:
                curr.next=next.next
        return head
```

41. Linked List Cycle

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if head==None:
            return False
        fast=head
        slow=head
        while fast.next!=None and fast.next.next!=None:
            fast=fast.next.next
            slow=slow.next
            if fast==slow:
                return True
        return False
```

203. Remove Linked List Elements

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeElements(self, head: Optional[ListNode], val: int)
        temp=ListNode
        curr=temp
        temp.next=head
        while curr.next!=None:
            if curr.next.val==val:
```

```
                curr.next=curr.next.next
            else:
                curr=curr.next
        return temp.next
```

237. Delete Node in a Linked List

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-plac
        """
        node.val=node.next.val
        node.next=node.next.next
```

2. Add Two Numbers

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional
```

```
        s=""
        s2=""
        while l1!=None:
            s=s+str(l1.val)
            l1=l1.next
        while l2!=None:
            s2=s2+str(l2.val)
            l2=l2.next
        s=s[::-1]
        s2=s2[::-1]
        a=int(s)+int(s2)
        a=str(a)
        a=a[::-1]
        q=str(a)
        z=ListNode()
        curr=z
        for i in q:
            new=ListNode(int(i))
            curr.next=new
            curr=curr.next
        return z.next
```

45. Binary Tree Postorder Traversal

```
# Definition for a binary tree node.
# class TreeNode:
#     def _init_(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def postorderTraversal(self, root: Optional[TreeNode]) -> Li
        s=[]
        def order(root,s):
```

```
            if root:
                order(root.left,s)
                order(root.right,s)
                s.append(root.val)
        order(root,s)
        return s
```

94. Binary Tree Preorder Traversal

```
# Definition for a binary tree node.
# class TreeNode:
#     def _init_(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def preorderTraversal(self, root: Optional[TreeNode]) -> Lis
        s=[]
        def order(root,s):
            if root:
                    s.append(root.val)
                order(root.left,s)
                order(root.right,s)
        order(root,s)
        return s
```

44. Binary Tree Inorder Traversal

```
# Definition for a binary tree node.
# class TreeNode:
#     def _init_(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
```

```
#         self.right = right
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List
        s=[]
        def order(root,s):
            if root:
                order(root.left,s)
                s.append(root.val)
                order(root.right,s)

        order(root,s)
        return s
```

04. Maximum Depth of Binary Tree

```
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        def height(root):
            if root:
                leftnode=height(root.left)
                rightnode=height(root.right)
                return max(leftnode,rightnode)+1
            else:
                return 0
        a=height(root)
        return a
```

11. Minimum Depth of the Binary Tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
```

```
#         self.left = left
#         self.right = right
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        def height(root):
            if root:
                if root.left is None:
                    return height(root.right)+1
                if root.right is None:
                    return height(root.left)+1
                leftnode=height(root.left)
                rightnode=height(root.right)
                return min(leftnode,rightnode)+1
            else:
                return 0
        a=height(root)
        return a
```

00. Same Tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[Tree
        def same(p,q):
            if p is None and q is None:
                return True
            if p is None or q is None:
                return False
```

```
            if p.val!=q.val:
                return False
            return same(p.left,q.left) and same(p.right,q.right
        return same(p,q)
```

101. Symmetric Tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def _init_(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSymmetric(self, root: Optional[TreeNode]) -> bool:
        def same(p,q):
                if p is None and q is None:
                    return True
                if p is None or q is None:
                    return False
                if p.val!=q.val:
                    return False
                return (p.val==q.val) and same(p.left,q.right) a
        return same(root.left,root.right)
```

108. Convert Sorted Array to Binary Search Tree

222. Count Complete Tree Nodes

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
```

```
#         self.left = left
#         self.right = right
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        s=[]
        def order(root,s):
            if root:
                order(root.left,s)
                s.append(root.val)
                order(root.right,s)
        order(root,s)
        return len(s)
```

27. Remove Elements

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        n=0
        for i in range(len(nums)):
            if nums[i]!=val:
                nums[n]=nums[i]
                n+=1
        return n
```

28. Find the Index of the First Occurrence in a String

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        for i in range(len(haystack)-len(needle)+1):
            if haystack[i:i+len(needle)]==needle:
                return i
        return -1
```

283. Move Zeroes

```
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        left=0
        for right in range(len(nums)):
            if nums[right]!=0:
                nums[right],nums[left]=nums[left],nums[right]
                left+=1
        return nums
```

268. Missing Number

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n=len(nums)
        expected_s=n*(n+1)//2
        actual_s=sum(nums)
        missing_num=expected_s-actual_s
        return missing_num
```

1748. Sum of Unique Elements

```
class Solution:
    def sumOfUnique(self, nums: List[int]) -> int:
        c=[]
        for i in nums:
            if nums.count(i)>1:
                continue
            else:
```

```
                c.append(i)
        return sum(c)
```

351. Count Negative Numbers in a Sorted Matrix

```
class Solution:
    def countNegatives(self, grid: List[List[int]]) -> int:
        c=0
        for i in grid:
            for j in i:
                if j<0:
                    c+=1
        return c
```

2716. Minimum String Length

```
class Solution:
    def minimizedStringLength(self, s: str) -> int:
        a=""
        for i in s:
            if(i not in a):
                a+=i
        return len(a)
```