**LEETCODE PROBLEMS:**

**LeetCode:1512**

class Solution:

    def numIdenticalPairs(self, nums: List[int]) -> int:

        hashMap = {}

        res = 0

        for number in nums:

            if number in hashMap:

                res += hashMap[number]

                hashMap[number] += 1

            else:

                hashMap[number] = 1

        return res

**Output:**

Input

nums =

[1,2,3,1,1,3]

Output

4

Expected

4

**LeetCode 1365:**

class Solution:

    def smallerNumbersThanCurrent(self, nums: List[int]) -> List[int]:

        ans = []

        for i in range(len(nums)):

            cnt = 0

            for j in range(len(nums)):

                if i!=j and nums[i]>nums[j]:

                    cnt+=1

            ans.append(cnt)

        return ans

**Output:**

Input

nums =

[8,1,2,2,3]

Output

[4,0,1,1,3]

Expected

[4,0,1,1,3]

**LeetCode 2574:**

class Solution:

    def leftRightDifference(self, nums: List[int]) -> List[int]:

        ans = []

        for i in range(len(nums)):

            ans.append(abs(sum(nums[:i]) - sum(nums[i+1:])))

        return ans

**Output:**

Input

nums =

[10,4,8,3]

Output

[15,1,11,22]

Expected

[15,1,11,22]

**LeetCode 1470:**

class Solution:

    def shuffle(self, nums: List[int], n: int) -> List[int]:

        result = []

        for i in range(0, n):

            result.append(nums[i])

            result.append(nums[i + n])

        return result

**Output:**

Input

nums =

[2,5,1,3,4,7]

n =

3

Output

[2,3,5,4,1,7]

Expected

[2,3,5,4,1,7]

**LeetCode 804:**

class Solution:

    def uniqueMorseRepresentations(self, words: List[str]) -> int:

        letters = "abcdefghijklmnopqrstuvwxyz"

        morse\_code = [".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.",

                      "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."]

        morse\_dict = dict(zip(letters, morse\_code))

        words2 = []

        for word in words:

            k = ""

            for i in word:

                k += morse\_dict[i]

            words2.append(k)

        return len(set(words2))

**Output:**

Input

words =

["gin","zen","gig","msg"]

Output

2

Expected

2

**LeetCode 1720:**

class Solution:

    def decode(self, A: List[int], first: int) -> List[int]:

        ans=[first]

        n=len(A)

        for i in range(n):

            a=ans[-1]^A[i]

            ans.append(a)

        return ans

**Output:**

Input

encoded =

[1,2,3]

first =

1

Output

[1,0,2,1]

Expected

[1,0,2,1]

**LeetCode 1528:**

class Solution:

    def restoreString(self, s: str, indices: List[int]) -> str:

        result=""

        for i in range(len(s)):

            b = indices.index(i)

            result += s[b]

        return result

**Output:**

Input

s =

"codeleet"

indices =

[4,5,6,7,0,2,1,3]

Output

"leetcode"

Expected

"leetcode"

**LeetCode 21:**

# 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: Optional[ListNode]) -> Optional[ListNode]:

        newHead = dummyHead = ListNode()

        while list1 and list2:

            if list1.val < list2.val:

                dummyHead.next = list1

                list1 = list1.next

            else:

                dummyHead.next = list2

                list2 = list2.next

            dummyHead = dummyHead.next

        if list1:

            dummyHead.next = list1

        if list2:

            dummyHead.next = list2

        return newHead.next

**Output:**

Input

list1 =

[1,2,4]

list2 =

[1,3,4]

Output

[1,1,2,3,4,4]

Expected

[1,1,2,3,4,4]

**Leetcode 1290:**

# Definition for singly-linked list.

# class ListNode:

#     def \_\_init\_\_(self, val=0, next=None):

#         self.val = val

#         self.next = next

class Solution:

    def getDecimalValue(self, head: ListNode) -> int:

        answer = ''

        if head is None:

            return

        current = head

        while current:

            answer = answer + str(current.val)

            current = current.next

        return int(answer,2)

o/p:

Input

head =

[1,0,1]

Output

5

Expected

5

**Leetcode 876:**

# 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):

        slow\_pointer = head

        fast\_pointer = head

        while fast\_pointer is not None and fast\_pointer.next is not None:

            slow\_pointer = slow\_pointer.next

            fast\_pointer = fast\_pointer.next.next

        return slow\_pointer

**o/p:**

Input

head =

[1,2,3,4,5]

Output

[3,4,5]

Expected

[3,4,5]

**Leetcode 1108:**

class Solution:

    def defangIPaddr(self, address: str) -> str:

        ans = address.replace(".","[.]")

        return ans

**o/p:**

Input

address =

"1.1.1.1"

Output

"1[.]1[.]1[.]1"

Expected

"1[.]1[.]1[.]1"

**Leetcode 1662:**

class Solution:

    def arrayStringsAreEqual(self, word1: List[str], word2: List[str]) -> bool:

        s1=""

        s2=""

        for i in word1:

            s1+=i

        for j in word2:

            s2+=j

        return s1==s2

**o/p:**

Input

word1 =

["ab", "c"]

word2 =

["a", "bc"]

Output

true

Expected

true

**Leetcode 771:**

class Solution:

    def numJewelsInStones(self, jewels: str, stones: str) -> int:

        jew\_list = list(jewels)

        answer = 0

        for j in jew\_list:

            answer += stones.count(j)

        return answer

**o/p:**

Input

jewels =

"aA"

stones =

"aAAbbbb"

Output

3

Expected

3

**Leetcode 2744:**

class Solution:

    def maximumNumberOfStringPairs(self, words: List[str]) -> int:

        result = 0

        for i in range(len(words)):

            if words[i][::-1] in (words[i+1:]):

                result+=1

        return result

**o/p:**

Input

words =

["cd","ac","dc","ca","zz"]

Output

2

Expected

2

**Leetcode 2418:**

class Solution:

    def sortPeople(self, names: List[str], heights: List[int]) -> List[str]:

        ans=zip(heights,names)

        l=[]

        for i,j in sorted(ans):

            l.append(j)

        return l[::-1]

**o/p:**

Input

names =

["Mary","John","Emma"]

heights =

[180,165,170]

Output

["Mary","Emma","John"]

Expected

["Mary","Emma","John"]

**Leetcode 1342:**

class Solution:

    def numberOfSteps(self, n: int) -> int:

        step=0

        while n!=0:

            if n%2==0:

                n/=2

                step+=1

            else:

                n-=1

                step+=1

        return step

**o/p:**

Input

num =

14

Output

6

Expected

6

**Leetcode 3099:**

class Solution:

    def sumOfTheDigitsOfHarshadNumber(self, x: int) -> int:

       n=x

       result=0

       while n!=0:

            result +=(n%10)

            n=n//10

       if x%result == 0:

            return result

       else:

            return -1

**o/p:**

Input

x =

18

Output

9

Expected

9

**Leetcode 20:**

class Solution(object):

def isValid(self, s):

stack = []

for i in s:

if i == '(':

stack.append(')')

elif i == '{':

stack.append('}')

elif i == '[':

stack.append(']')

elif not stack or stack.pop() != i:

return False

return not stack

"""

:type s: str

:rtype: bool

        """

**Leetcode 14:**

class Solution(object):

def longestCommonPrefix(self, strs: List[str]) -> str:

ans = ""

strs = sorted(strs)

first = strs[0]

last = strs[-1]

for i in range(min(len(first),len(last))):

if(first[i] != last[i]):

return ans

ans += first[i]

      return ans

**Leetcode 1614:**

class Solution:

    def maxDepth(self, s: str) -> int:

        max\_depth = 0

        current\_depth = 0

        for char in s:

            if char == '(':

                current\_depth += 1

                max\_depth = max(max\_depth, current\_depth)

            elif char == ')':

                current\_depth -= 1

        return max\_depth

**output:**

s =

"(1+(2\*3)+((8)/4))+1"

Output

3

Expected

3

**Leetcode 2960:**

class Solution:

    def removeStars(self, s: str) -> str:

        ans=[]

        for i in s:

            if i=='\*':

                ans.pop()

            else:

                ans+=[i]

        return "".join(ans)

**output:**

s="leet\*\*cod\*e"

**Leetcode 1700:**

class Solution:

    def countStudents(self, students: List[int], sandwiches: List[int]) -> int:

        count = Counter(students)

        res = 0

        for s in sandwiches:

            if count[s] == 0:

                break

            count[s] -= 1

            res += 1

        return len(students) - res

**ii)** class Solution:

    def countStudents(self, students: List[int], sandwiches: List[int]) -> int:

        count = len(students)

        while(sandwiches and students and sandwiches[0] in students):

            if(sandwiches[0]!=students[0]):

                students.append(students[0])

                students.pop(0)

            else:

                students.pop(0)

                sandwiches.pop(0)

                count-=1

        return count

**output:**

Input

students =

[1,1,1,0,0,1]

sandwiches =

[1,0,0,0,1,1]

Output

3

Expected

3

**Leetcode 1021:**

class Solution:

    def removeOuterParentheses(self, S: str) -> str:

        stack = []

        result = ""

        for char in S:

            if char == '(':

                if stack:

                    result += char

                stack.append(char)

            else:

                stack.pop()

                if stack :

                    result += char

        return result

**output:**

Input

s =

"(()())(())(()(()))"

Output

"()()()()(())"

Expected

"()()()()(())"

**Leetcode 1431:**

class Solution:

    def kidsWithCandies(self, candies: List[int], extraCandies: int) -> List[bool]:

       result\_lst = []

       max\_num = max(candies)

       for num in candies:

           result\_lst.append(num + extraCandies >= max\_num)

       return result\_lst

**o/p:**

candies =

[2,3,5,1,3]

extraCandies =

3

Output

[true,true,true,false,true]

Expected

[true,true,true,false,true]