Assignment - 3

SOORYA.S-192311060

21. Merge Two Sorted List

**Code:**

class Solution:

def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:

cur = dummy = ListNode()

while list1 and list2:

if list1.val < list2.val:

cur.next = list1

list1, cur = list1.next, list1

else:

cur.next = list2

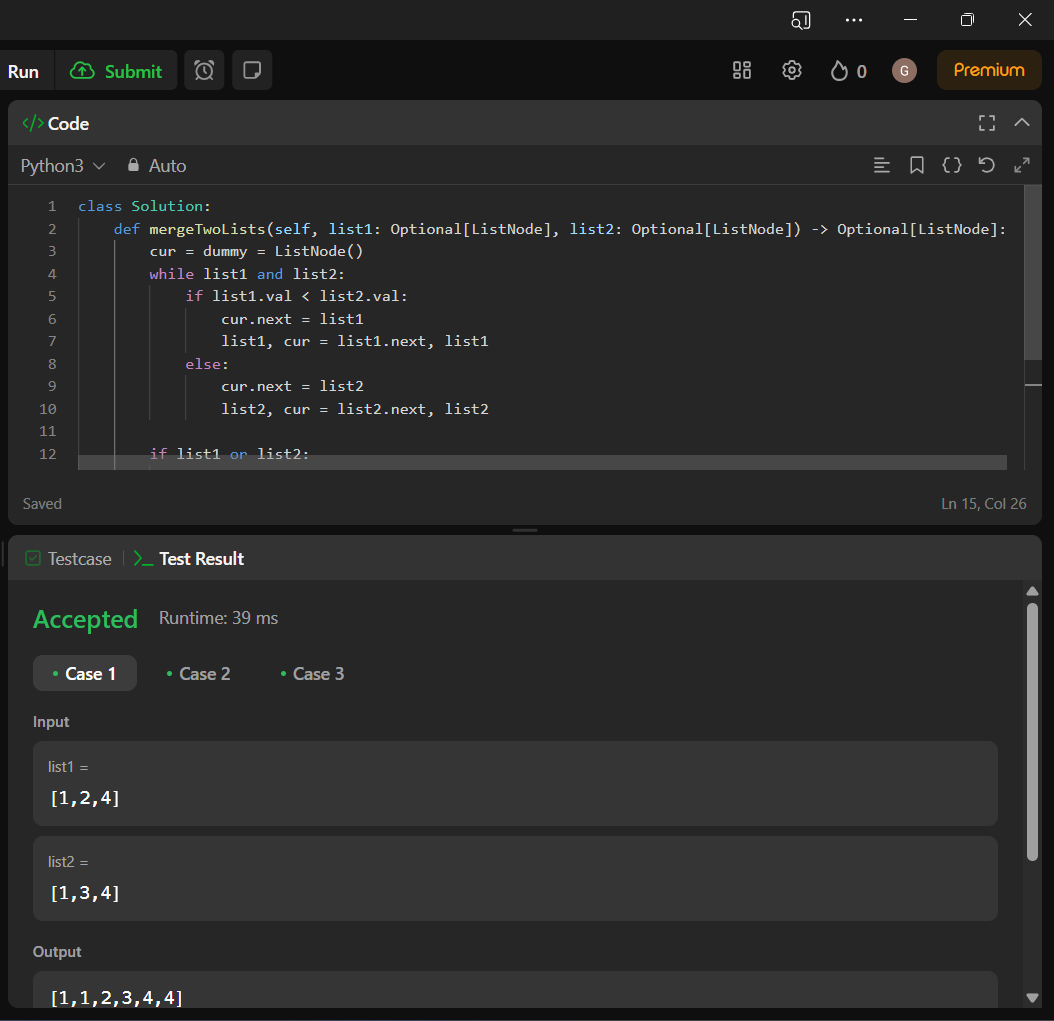
list2, cur = list2.next, list2

if list1 or list2:

cur.next = list1 if list1 else list2

return dummy.next

**Screenshot:**



**Time Complexity:** O(n)

22. Generate Parenthesis

**Code:**

class Solution(object):

def generateParenthesis(self, n):

def backtrack(S='', left=0, right=0):

if len(S) == 2 \* n:

result.append(S)

return

if left < n:

backtrack(S + '(', left + 1, right)

if right < left:

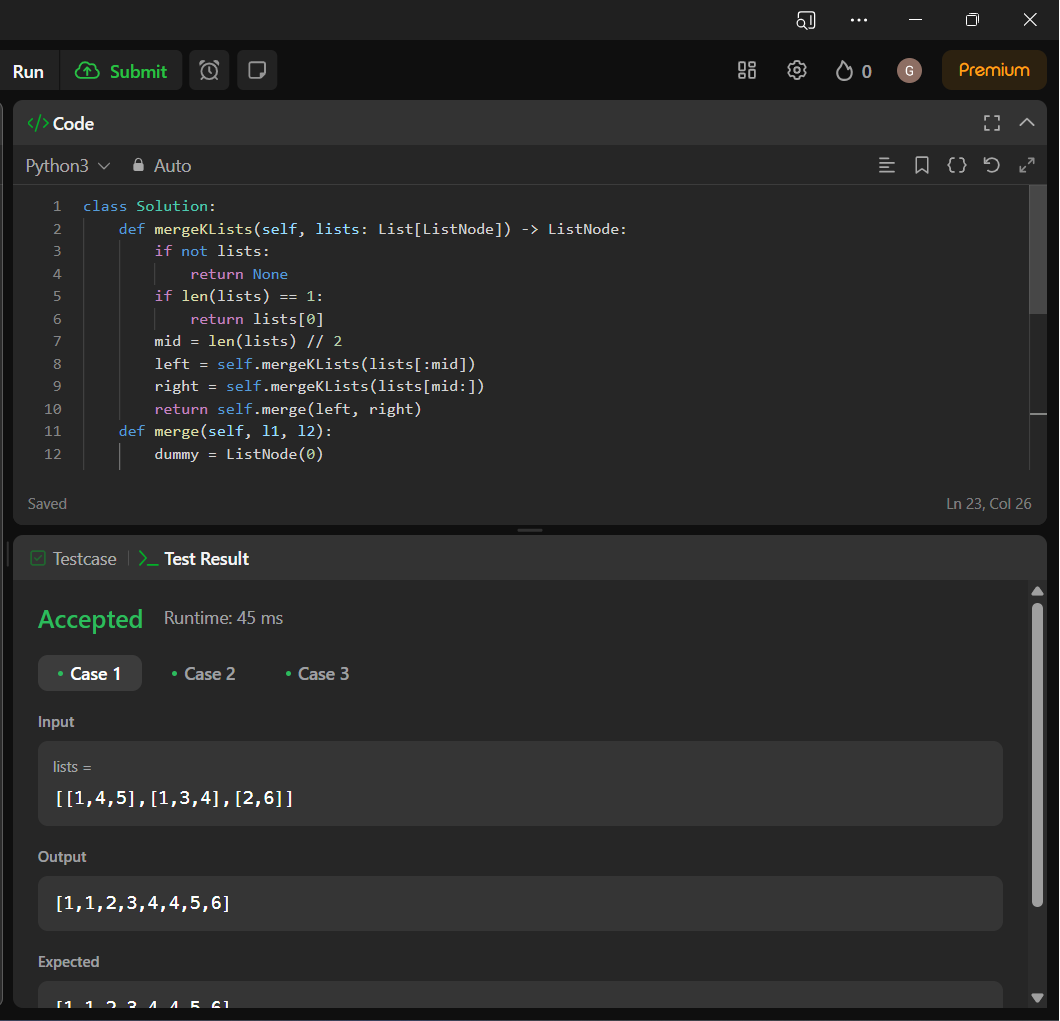
backtrack(S + ')', left, right + 1)

result = []

backtrack()

return result

**Screenshot for I/O:**



**Time Complexity:** O(n)

23. Merge K Sorted Lists

**Code:**

class Solution:

def mergeKLists(self, lists: List[ListNode]) -> ListNode:

if not lists:

return None

if len(lists) == 1:

return lists[0]

mid = len(lists) // 2

left = self.mergeKLists(lists[:mid])

right = self.mergeKLists(lists[mid:])

return self.merge(left, right)

def merge(self, l1, l2):

dummy = ListNode(0)

curr = dummy

while l1 and l2:

if l1.val < l2.val:

curr.next = l1

l1 = l1.next

else:

curr.next = l2

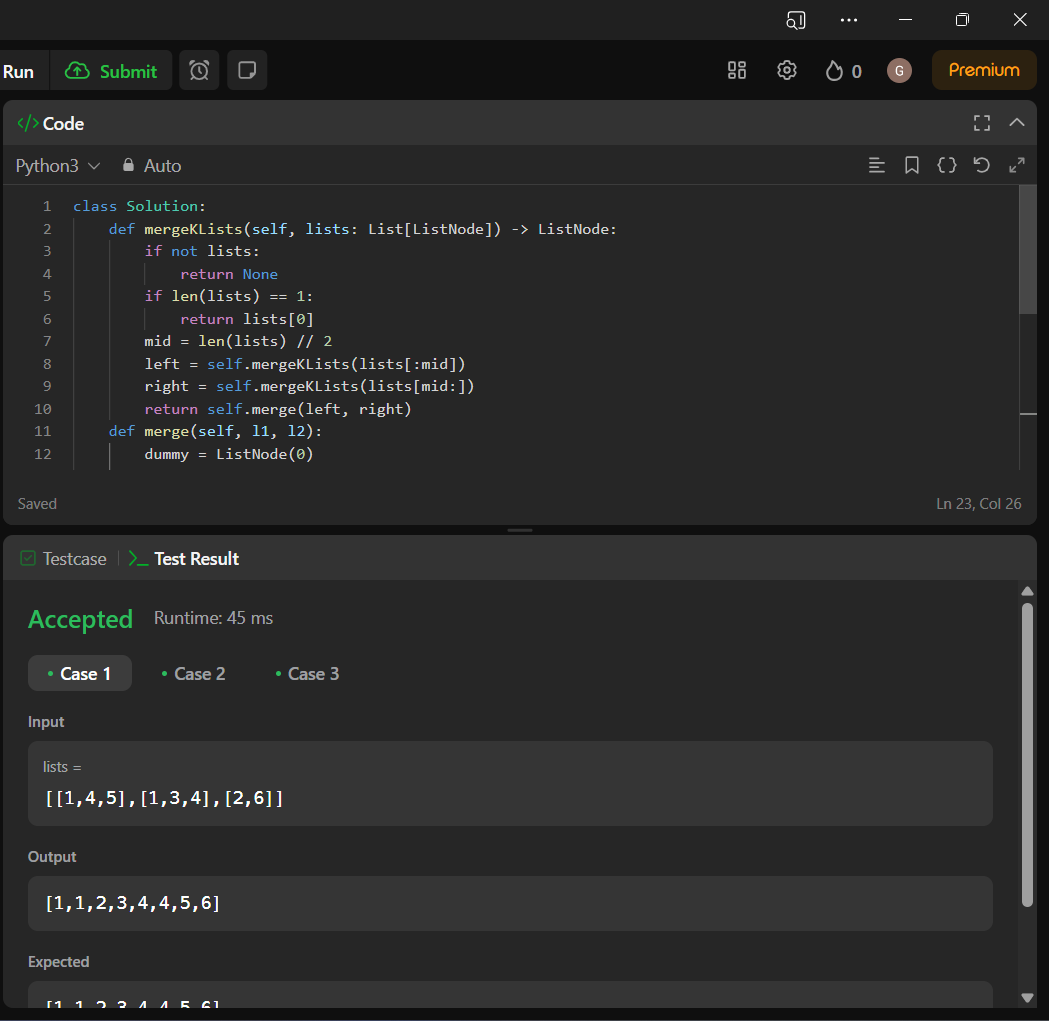
l2 = l2.next

curr = curr.next

curr.next = l1 or l2

return dummy.next

**Screenshot:**

****

**Time Complexity: O(n)**

24. Swap Nodes in Pairs

**Code:**

class Solution:

def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:

lst=[]

while head:

lst.append(head.val)

head=head.next

ans=[]

for i in range(0,len(lst),2):

val=lst[i:i+2]

ans+=val[::-1]

final=ListNode(0)

tmp=final

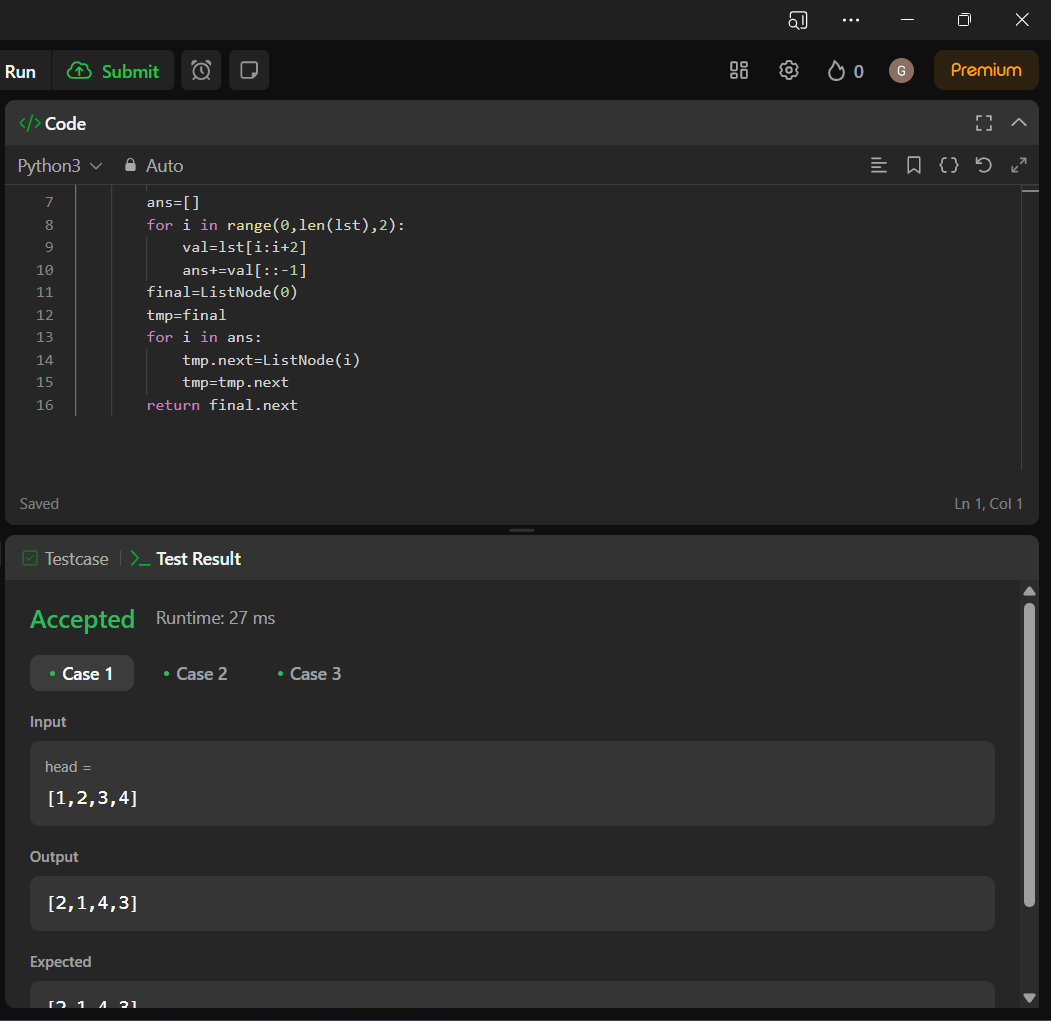
for i in ans:

tmp.next=ListNode(i)

tmp=tmp.next

return final.next

**Screenshot:**

****

**Time Complexity: O(n)**

25. Reverse Nodes in k-Group

**Code:**

class Solution:

def getLength(self,head):

length = 0

while head:

length+=1

head = head.next

return length

def reverseGroup(self,head,k,length):

if length<k:

return head

curr = head

prev = None

next = None

count = 0

while curr and count<k:

next = curr.next

curr.next = prev

prev = curr

curr = next

count+=1

if next:

head.next = self.reverseGroup(next,k,length-k)

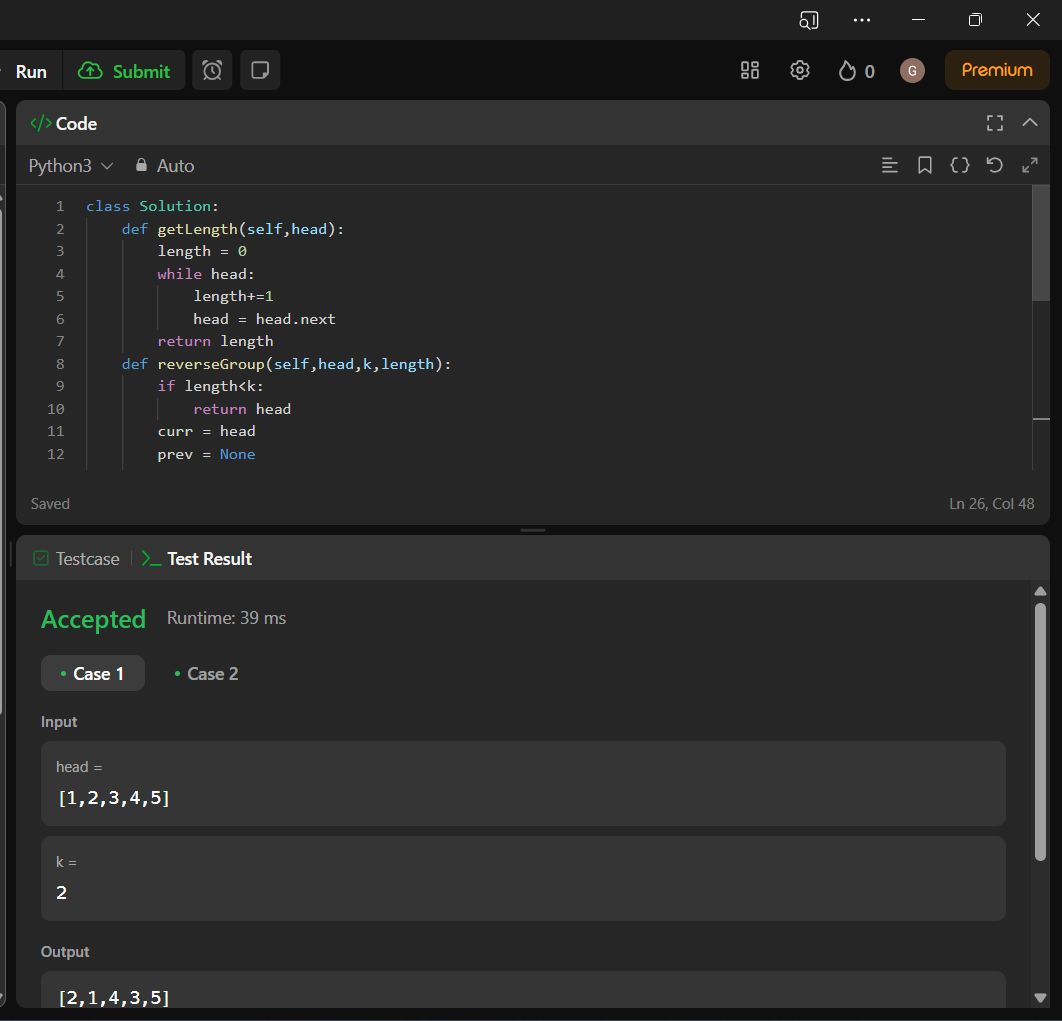
return prev

def reverseKGroup(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:

length = self.getLength(head)

return self.reverseGroup(head,k,length)

**Screenshot:**

****

**Time Complexity: O(n)**

26. Remove Duplicate from Sorted Array

**Code:**

class Solution:

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

i,j=0,1

while i<=j and j<len(nums):

if nums[i]==nums[j]:

j+=1

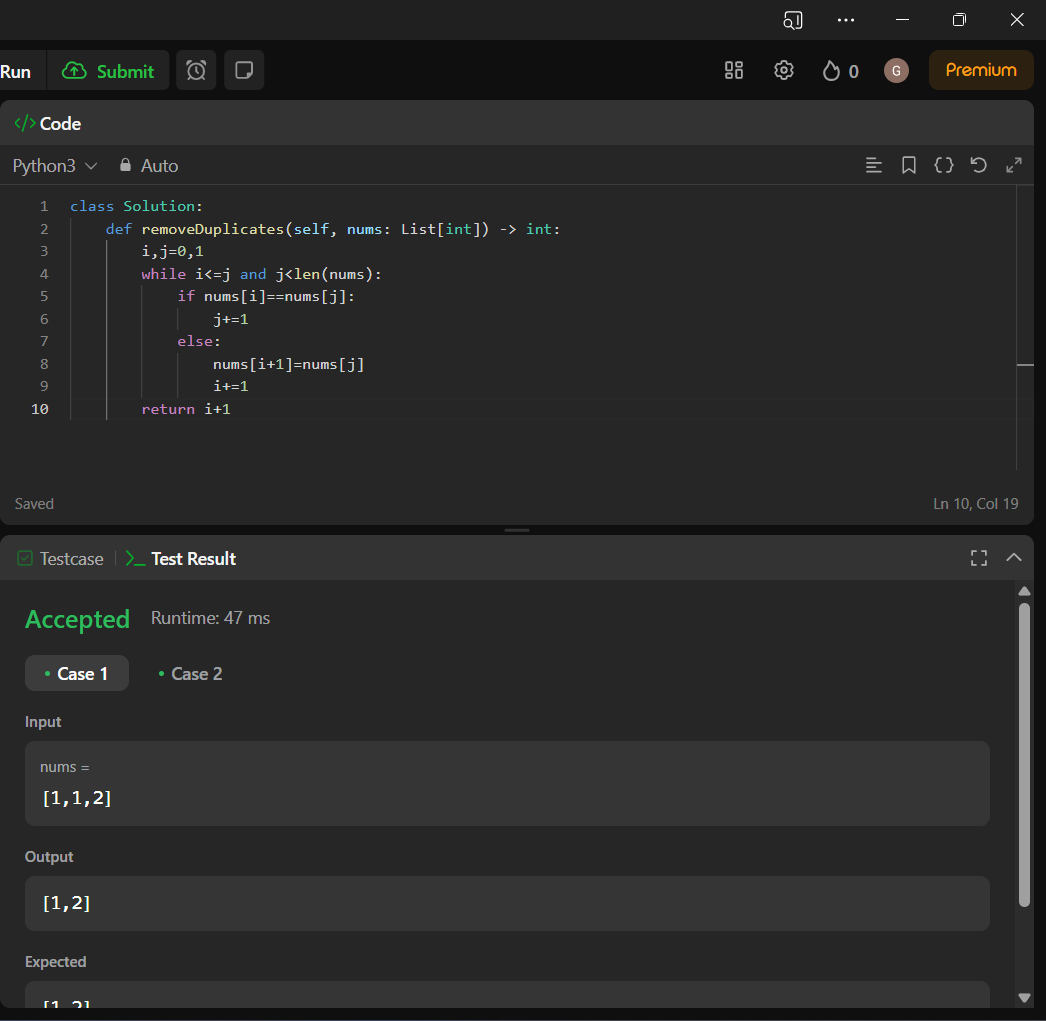
else:

nums[i+1]=nums[j]

i+=1

return i+1

**Screenshot:**

****

**Time Complexity: O(n)**

27. Remove Element

**Code:**

class Solution:

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

c=nums.count(val)

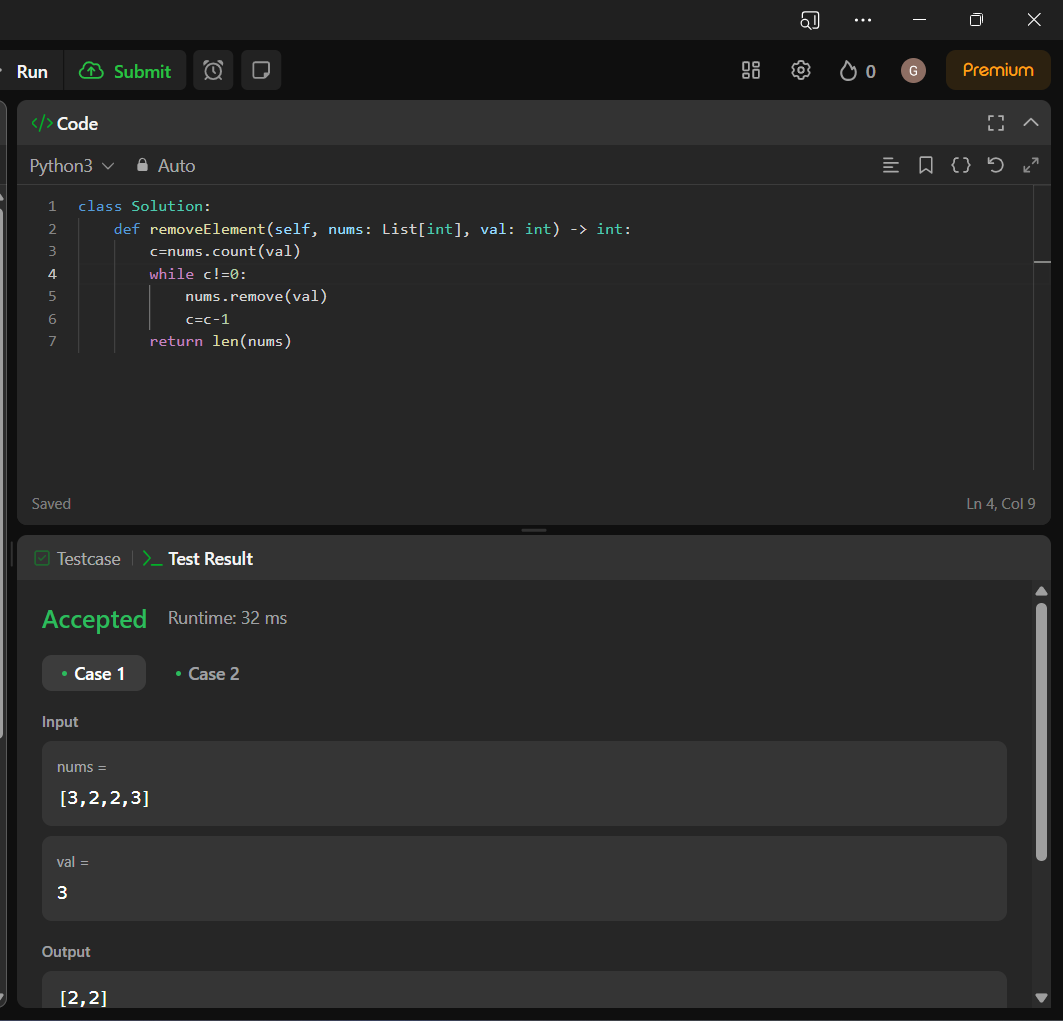
while c!=0:

nums.remove(val)

c=c-1

return len(nums)

**Screenshot:**

****

**Time Complexity: O(n)**

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

**Code:**

class Solution(object):

def strStr(self, haystack, needle):

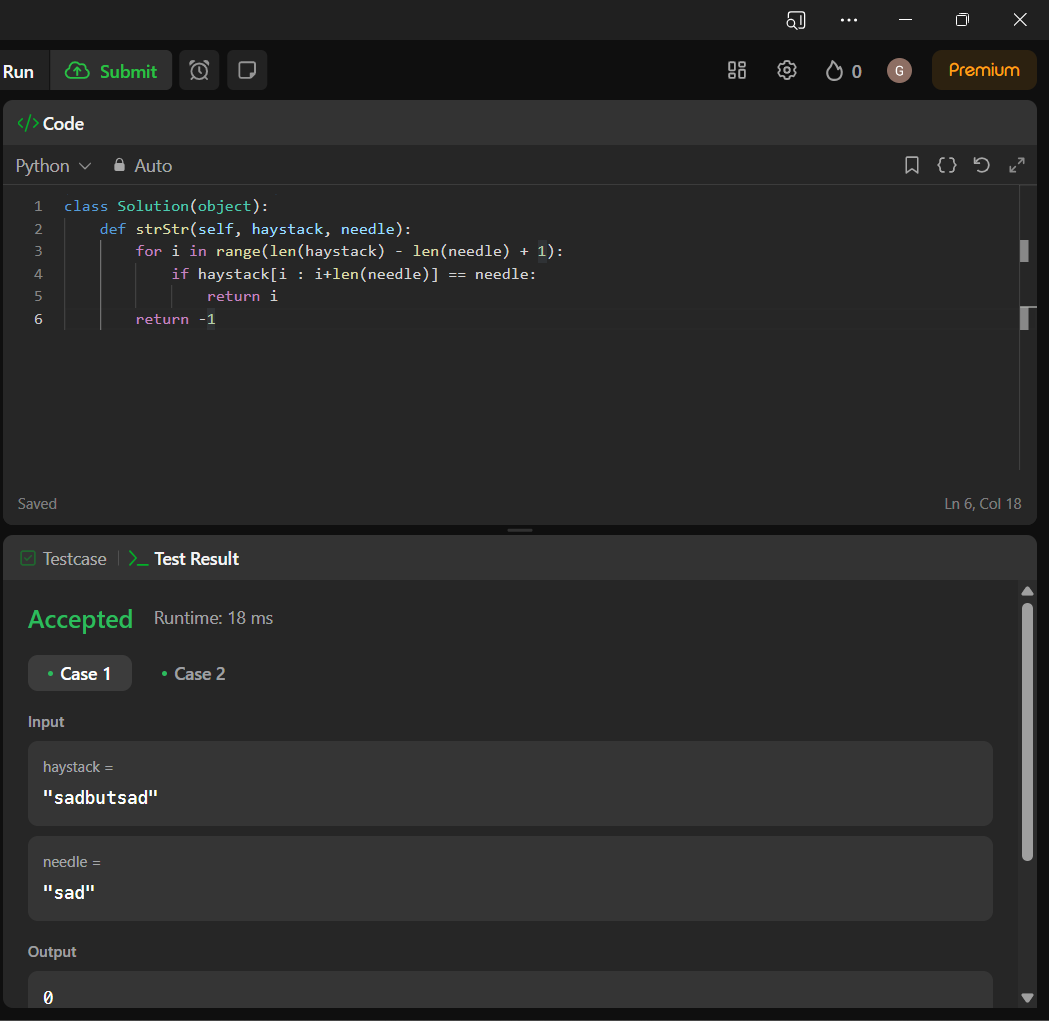
for i in range(len(haystack) - len(needle) + 1):

if haystack[i : i+len(needle)] == needle:

return i

return -1

**Screenshot:**

****

**Time Complexity: O(n)**

29. Divide Two Integers

**Code:**

class Solution:

def divide(self, dividend: int, divisor: int) -> int:

sign = -1 if (dividend >= 0 and divisor < 0) or (dividend < 0 and divisor >= 0) else 1

dividend = abs(dividend)

divisor = abs(divisor)

result = len(range(0, dividend-divisor+1, divisor))

if sign == -1:

result = -result

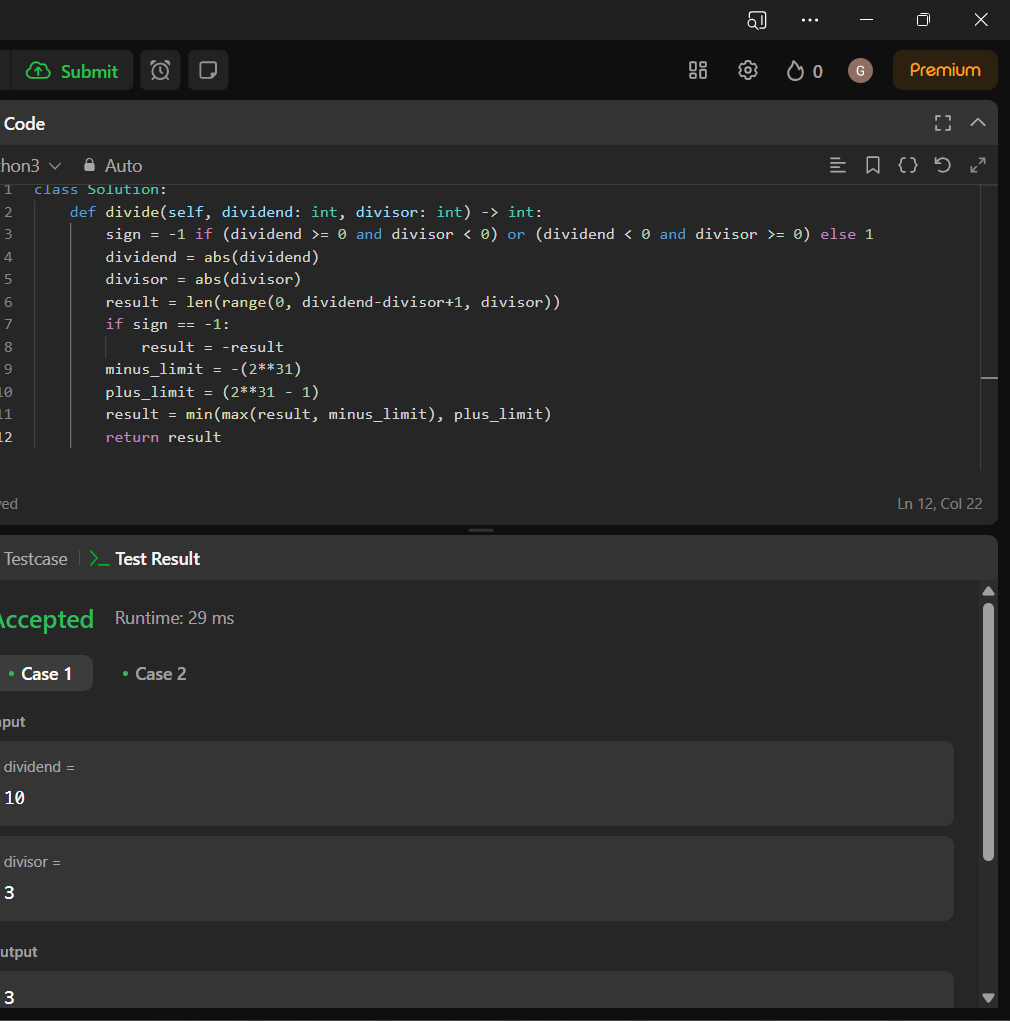
minus\_limit = -(2\*\*31)

plus\_limit = (2\*\*31 - 1)

result = min(max(result, minus\_limit), plus\_limit)

return result

**Screenshot:**

****

**Time Complexity: O(n)**

30. Substring with Concatenation of All Words

**Code:**

class Solution:

def calc(self ,i):

cnt = 0

ind = i

while (ind < i+self.pl):

news = self.s[ind : ind + self.n]

if news in self.dic :

self.dic[news] -= 1

if self.dic[news] == 0 :

cnt += 1

ind += self.n

else :

return False

if cnt == len(self.dic) :

return True

else :

return False

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

self.s = s

self.n = len(words[0])

d = {}

for x in words :

if x in d :

d[x] += 1

else :

d[x] = 1

self.pl = len(words)\*len(words[0])

ans = []

i = 0

while(i< len(s) - self.pl + 1):

self.dic = {x : d[x] for x in d}

if self.calc(i) :

ans += [i]

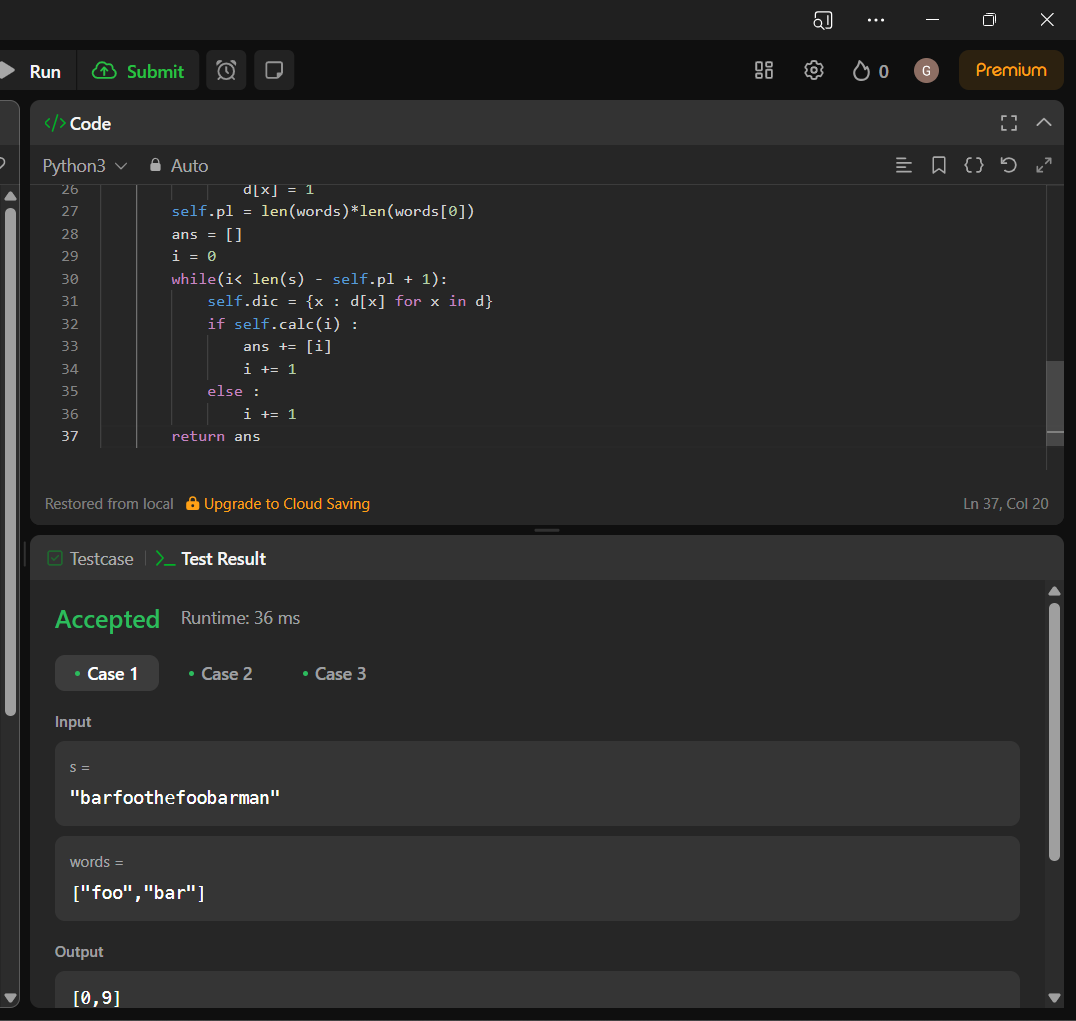
i += 1

else :

i += 1

return ans

**Screenshot:**

****