Assignment - 2

K VISAGHAN – 192321013

# Container With Most Water

**Code:**

class Solution:

def maxArea(self, height: List[int]) -> int: left = 0

right = len(height) - 1 maxArea = 0

while left < right:

currentArea = min(height[left], height[right]) \* (right

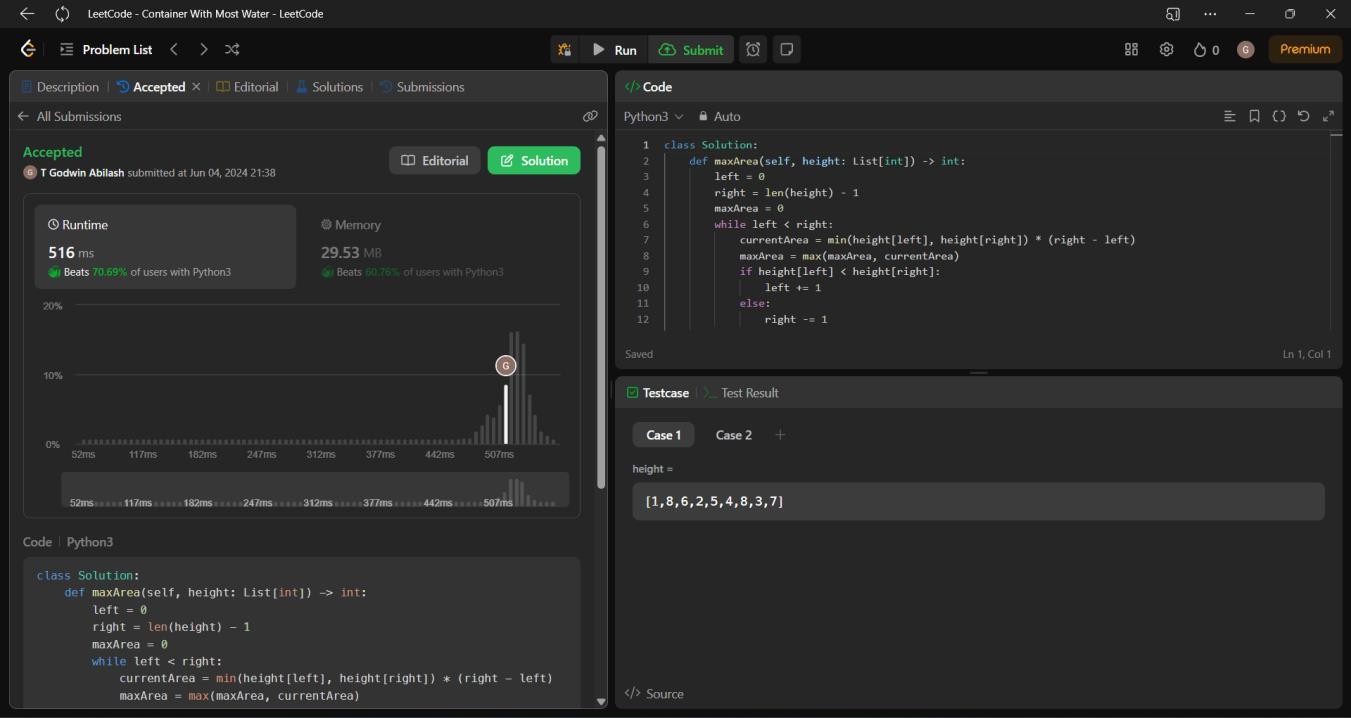
- left)

maxArea = max(maxArea, currentArea) if height[left] < height[right]:

left += 1 else:

right -= 1 return maxArea

**Screenshot:**



**Time Complexity:** O(n)

# Integer to Roman

**Code:**

class Solution:

def intToRoman(self, num: int) -> str: Roman = ""

storeIntRoman = [[1000, "M"], [900, "CM"], [500, "D"], [400,

"CD"], [100, "C"], [90, "XC"], [50, "L"], [40, "XL"], [10, "X"], [9,

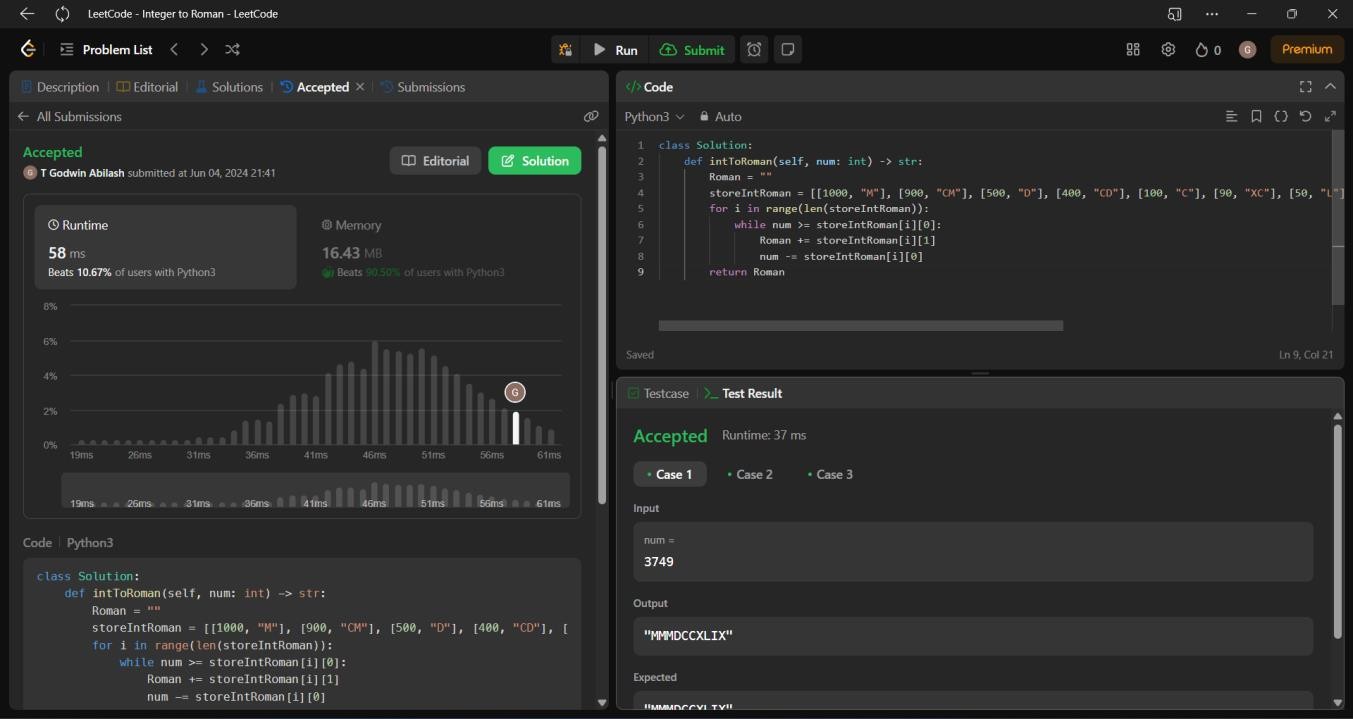
"IX"], [5, "V"], [4, "IV"], [1, "I"]]

for i in range(len(storeIntRoman)): while num >= storeIntRoman[i][0]:

Roman += storeIntRoman[i][1] num -= storeIntRoman[i][0]

return Roman

**Screenshot for I/O:**



**Time Complexity:** O(max(m,n))

# Roman to Integer

**Code:**

class Solution:

def romanToInt(self, s: str) -> int: roman={"I":1,"V":5,"X":10,"L":50,"C":100,"D":500,"M":1000}

number=0

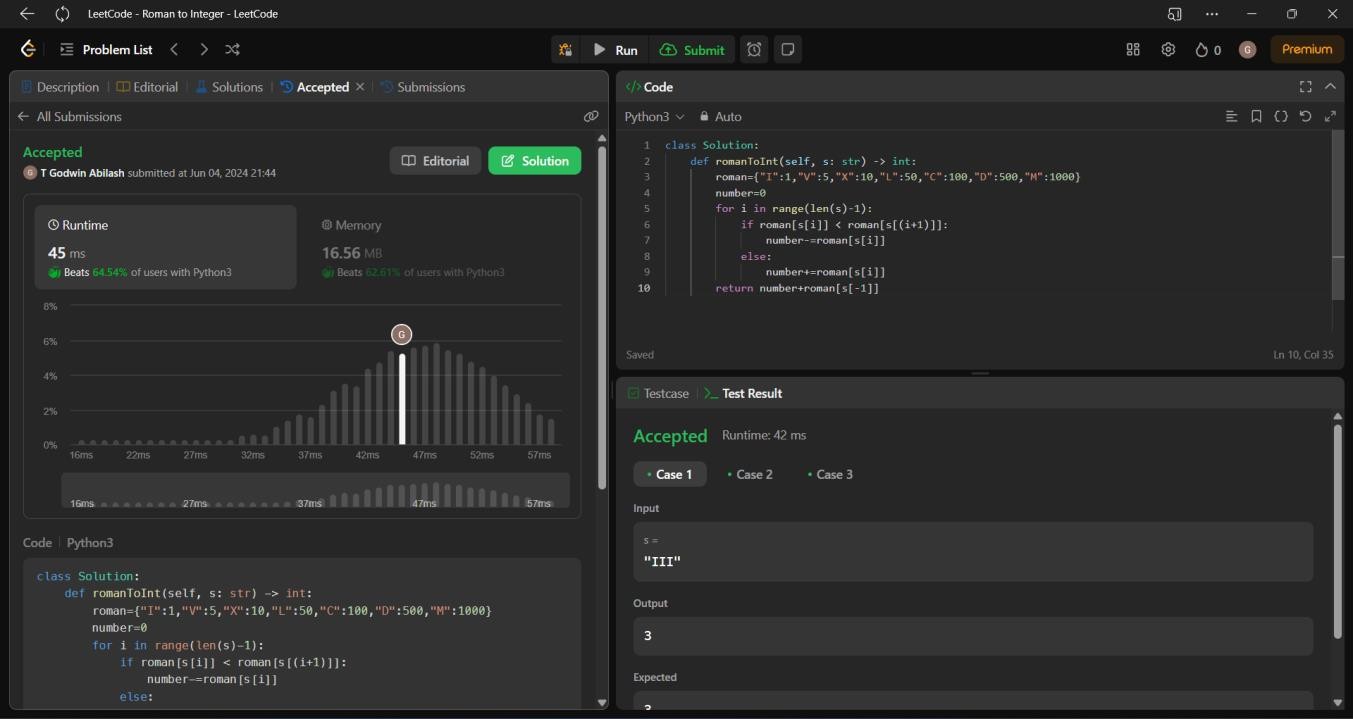
for i in range(len(s)-1):

if roman[s[i]] < roman[s[(i+1)]]: number-=roman[s[i]]

else:

number+=roman[s[i]] return number+roman[s[-1]]

**Screenshot:**



**Time Complexity: O(n)**

# Longest Common Prefix

**Code:**

class Solution:

def longestCommonPrefix(self, strs): if not strs:

return "" prefix = strs[0]

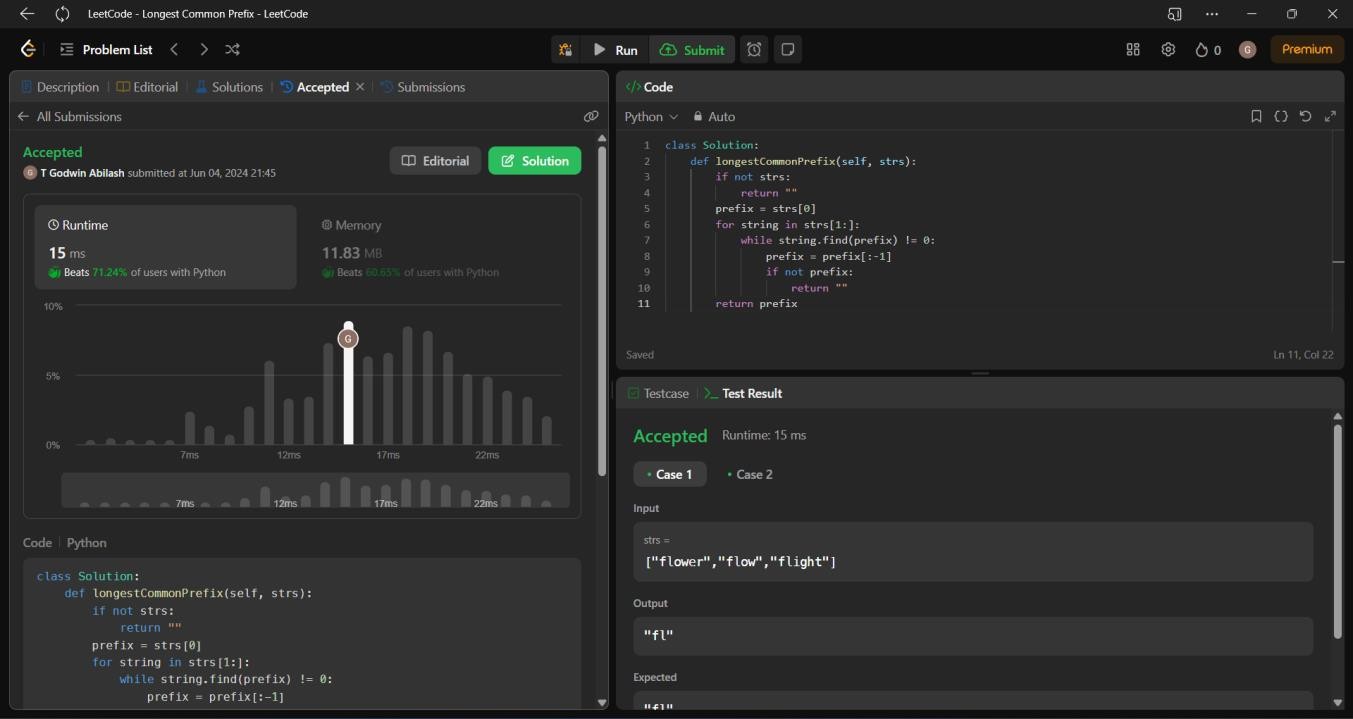
for string in strs[1:]:

while string.find(prefix) != 0: prefix = prefix[:-1]

if not prefix: return ""

return prefix

**Screenshot:**



**Time Complexity: O(n)**

# 3Sum

**Code:**

class Solution:

def threeSum(self, nums: List[int]) -> List[List[int]]: res = []

nums.sort()

for i in range(len(nums)):

if i > 0 and nums[i] == nums[i-1]: continue

j = i + 1

k = len(nums) - 1 while j < k:

total = nums[i] + nums[j] + nums[k] if total > 0:

k -= 1

elif total < 0: j += 1

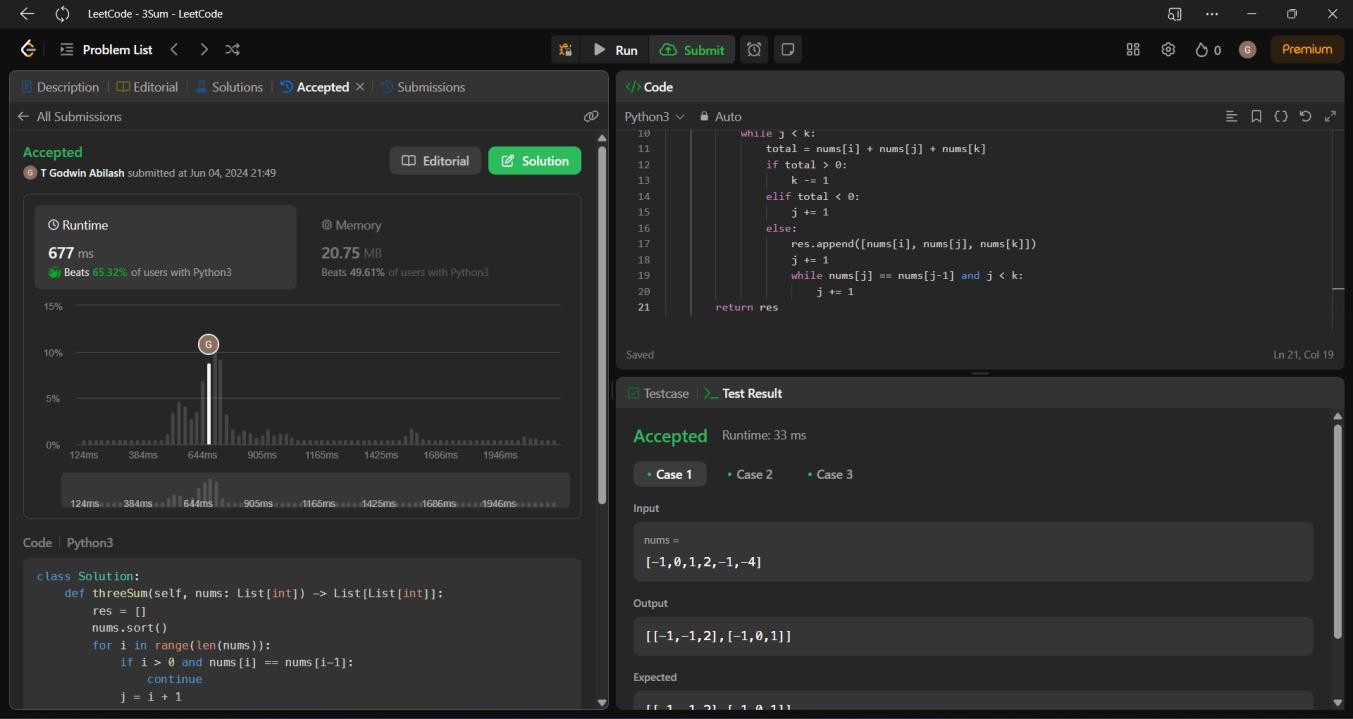
else:

res.append([nums[i], nums[j], nums[k]]) j += 1

while nums[j] == nums[j-1] and j < k: j += 1

return res

**Screenshot:**



**Time Complexity: O(n2)**

# 3Sum Closet

**Code:**

class Solution:

def threeSumClosest(self, nums: List[int], target: int) -> int: nums.sort()

answer = nums[0] + nums[1] + nums[2] for left in range(len(nums) - 2):

middle = left + 1 right = len(nums) - 1 while middle < right:

guess = nums[left] + nums[middle] + nums[right] if abs(guess - target) < abs(answer - target):

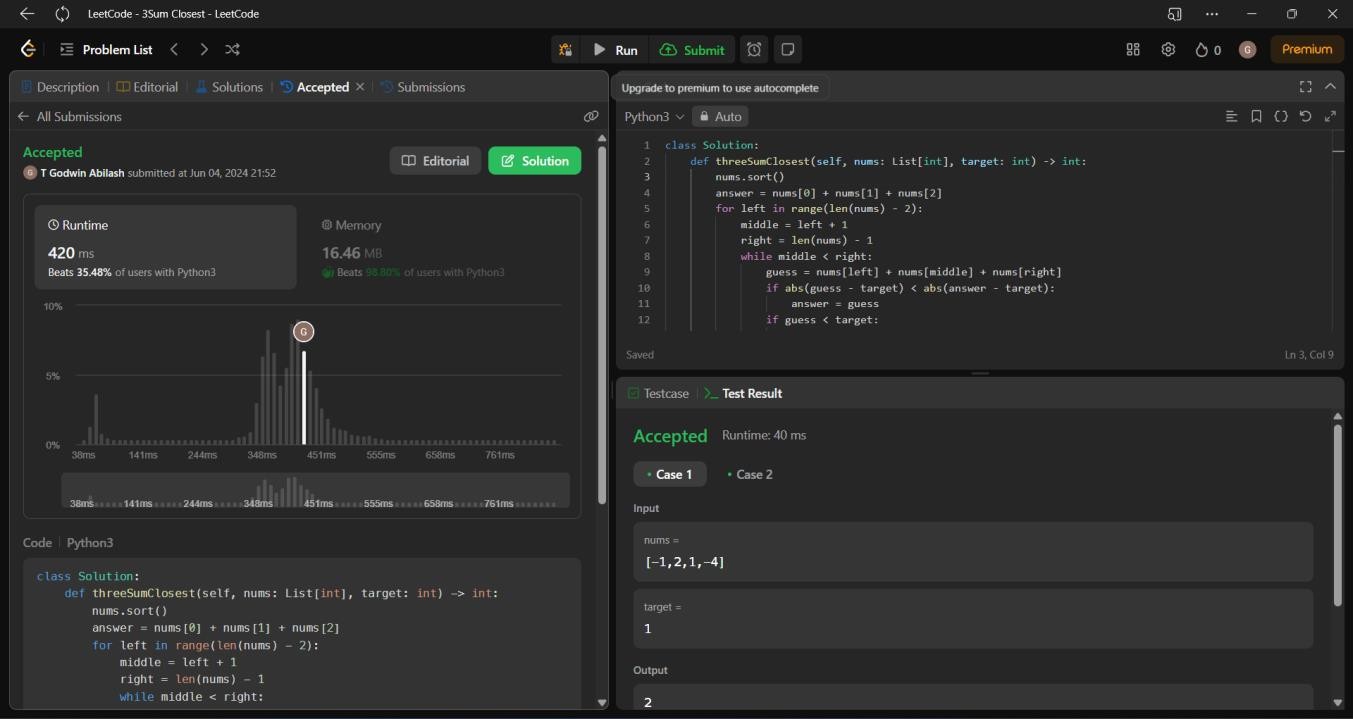
answer = guess if guess < target:

middle += 1 elif guess > target:

right -= 1 else:

return target return answer

**Screenshot:**



**Time Complexity: O(n)**

# Letter Combinations of the phone number

**Code:**

class Solution(object):

def letterCombinations(self, digits):

dic = { "2": "abc", "3": "def", "4":"ghi", "5":"jkl", "6":"mno", "7":"pqrs", "8":"tuv", "9":"wxyz"}

res=[]

if len(digits) ==0: return res

self.dfs(digits, 0, dic, '', res) return res

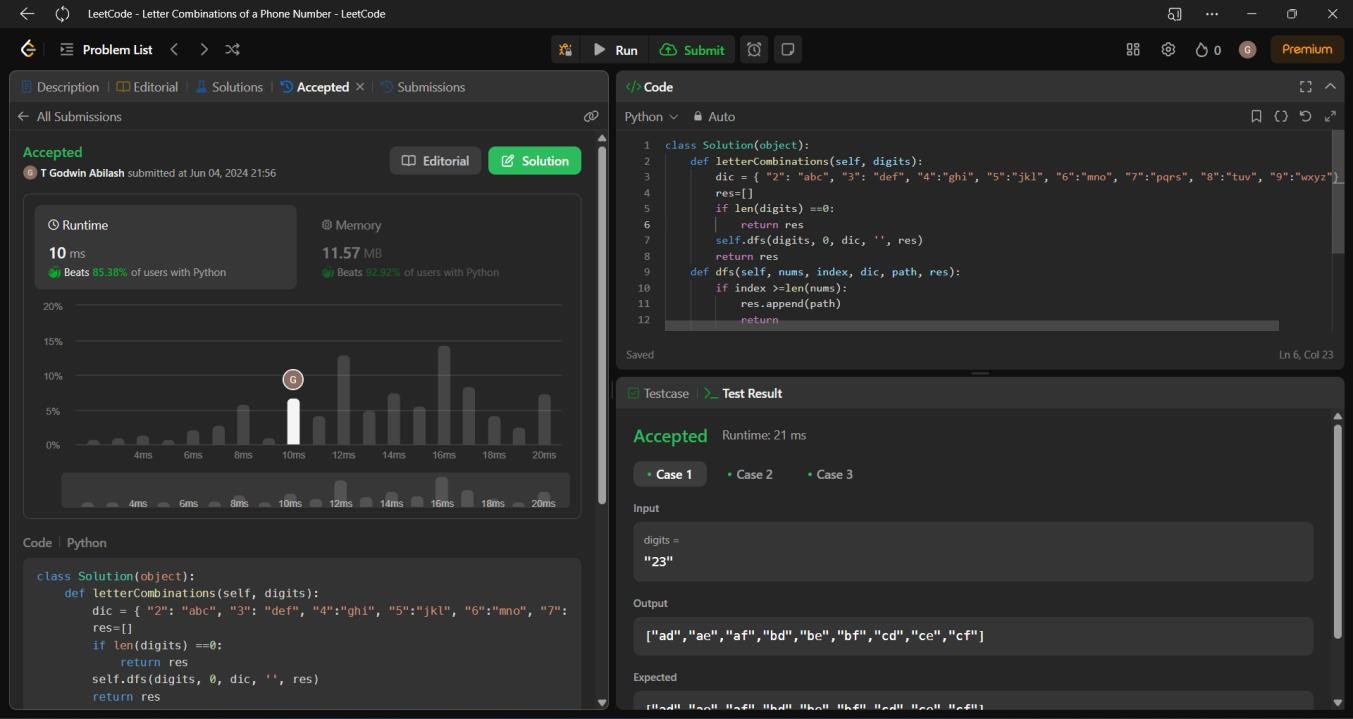
def dfs(self, nums, index, dic, path, res): if index >=len(nums):

res.append(path) return

string1 =dic[nums[index]] for i in string1:

self.dfs(nums, index+1, dic, path + i, res)

**Screenshot:**



**Time Complexity: O(n)**

# 4Sum

**Code:**

class Solution:

def fourSum(self, nums: List[int], target: int) -> List[List[int]]: nums.sort()

s = set() output = []

for i in range(len(nums)):

for j in range(i+1, len(nums)):

k = j + 1

l = len(nums) - 1 while k < l:

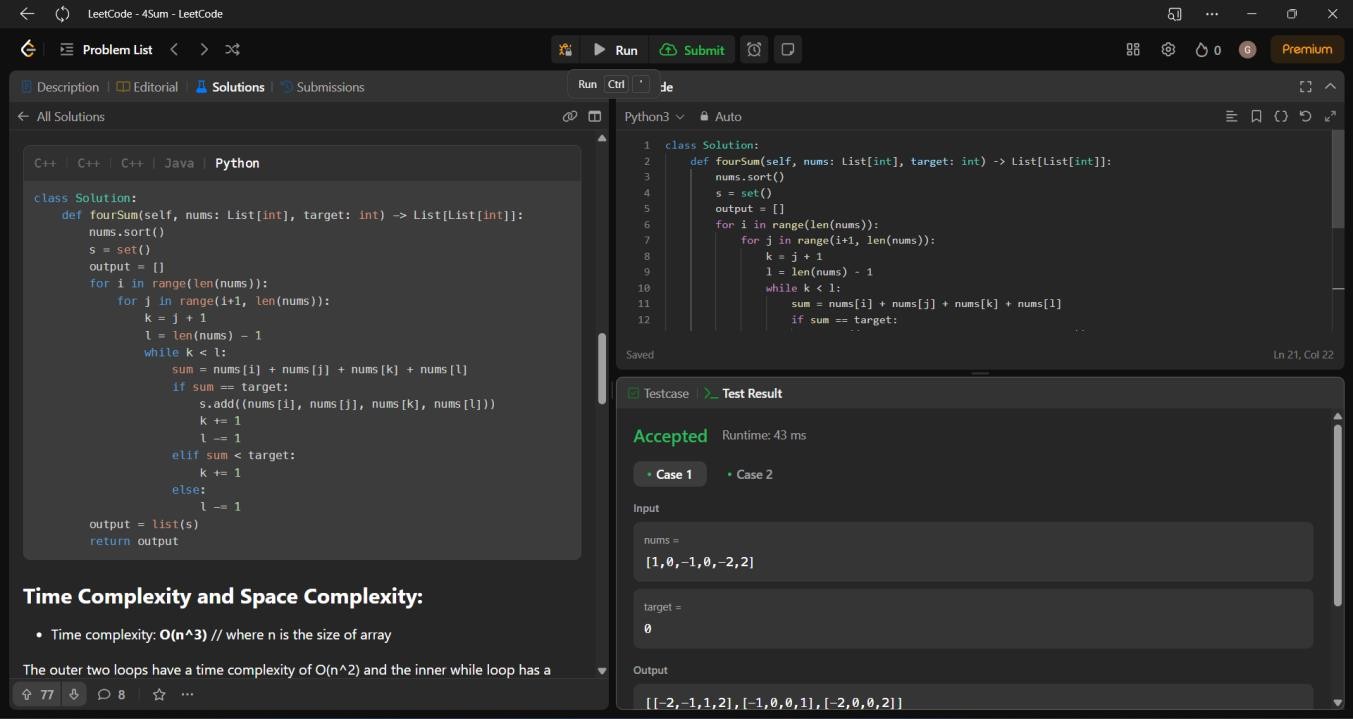
sum = nums[i] + nums[j] + nums[k] + nums[l] if sum == target:

s.add((nums[i], nums[j], nums[k], nums[l]))

|  |  |  |
| --- | --- | --- |
| k += | 1 |  |
| l -= | 1 |
| elif sum | < | target: |
| k += | 1 |  |
| else: |  |  |
| l -= | 1 |  |

output = list(s) return output

**Screenshot:**



**Time Complexity: O(n2)**

# Remove Nth Node from end of the list

**Code:**

class Solution:

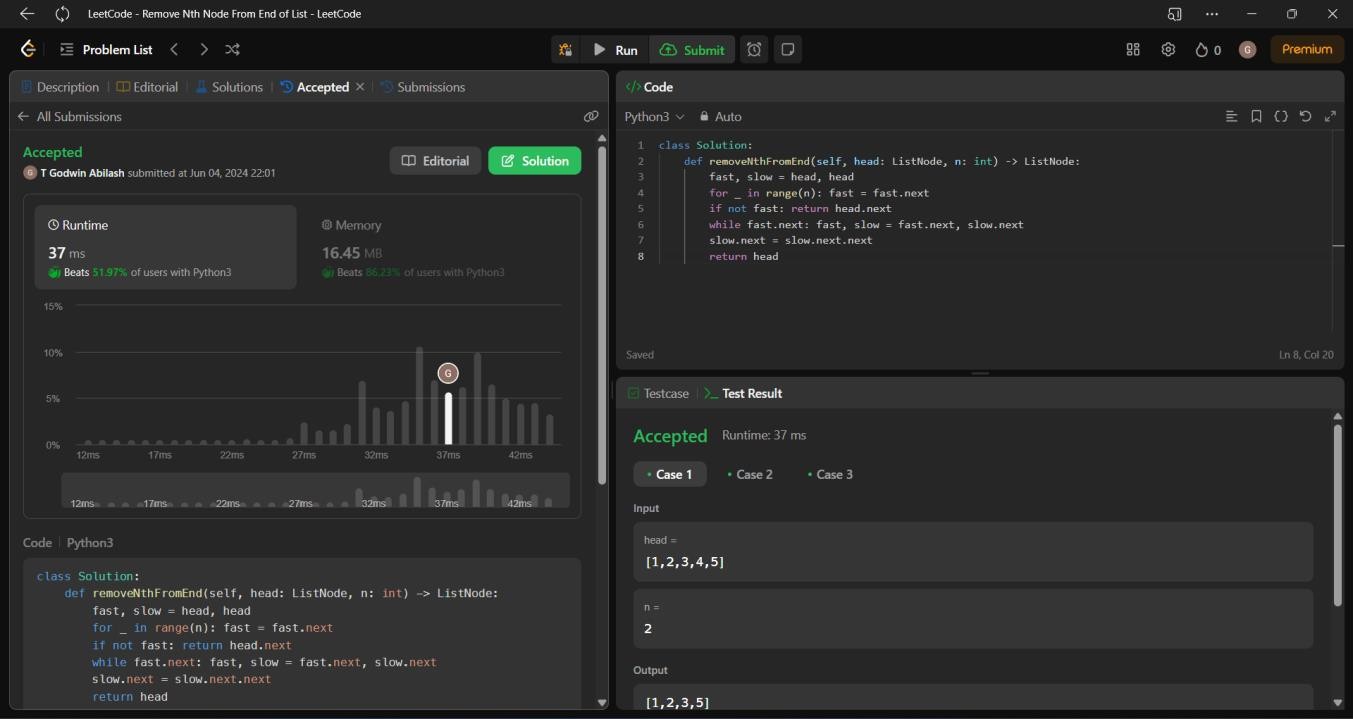
def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode: fast, slow = head, head

for \_ in range(n): fast = fast.next if not fast: return head.next

while fast.next: fast, slow = fast.next, slow.next slow.next = slow.next.next

return head

**Screenshot:**



**Time Complexity: O(n)**

# 10. Regular Expression Matching

**Code:**

class Solution(object): def isValid(self, s):

stack = [] pairs = {

'(': ')',

'{': '}',

'[': ']'

}

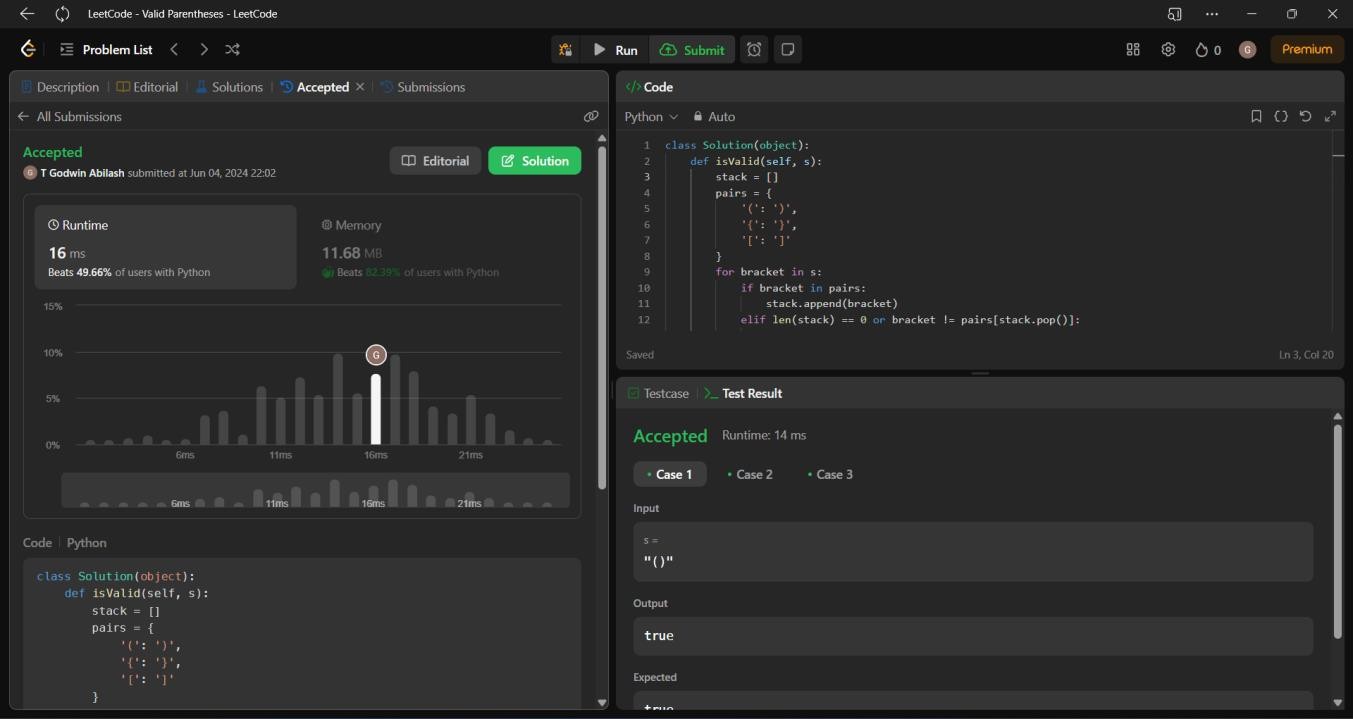
for bracket in s:

if bracket in pairs: stack.append(bracket)

elif len(stack) == 0 or bracket != pairs[stack.pop()]: return False

return len(stack) == 0

**Screenshot:**



**Time Complexity: O(n)**