## Kth Largest Element in an Array

class Solution(object):

def findKthLargest(self, nums, k):

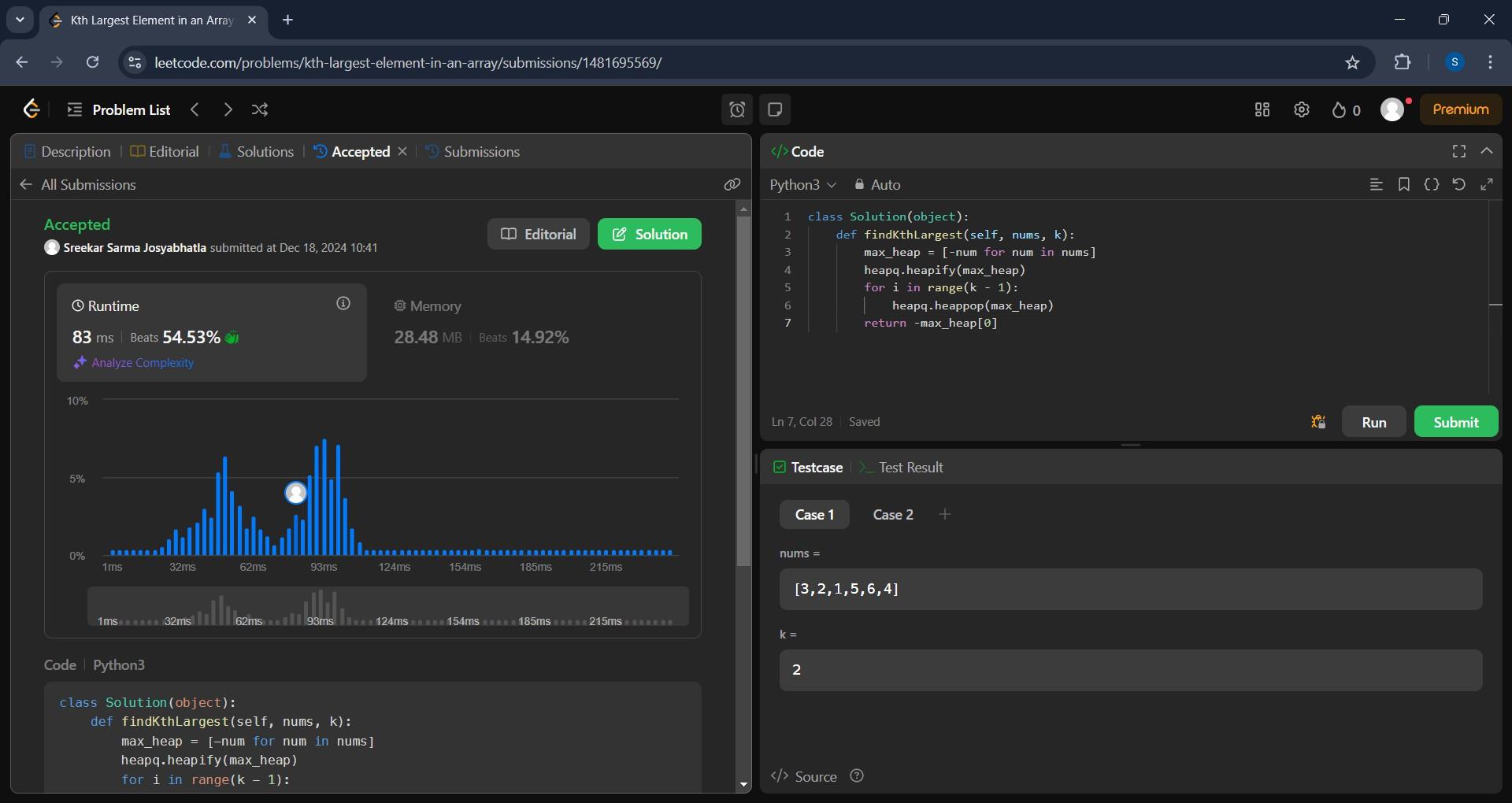
# Task-3

max\_heap = [-num for num in nums] heapq.heapify(max\_heap)

for i in range(k - 1): heapq.heappop(max\_heap)

return -max\_heap[0]

OUTPUT



## Merge k Sorted Lists

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:

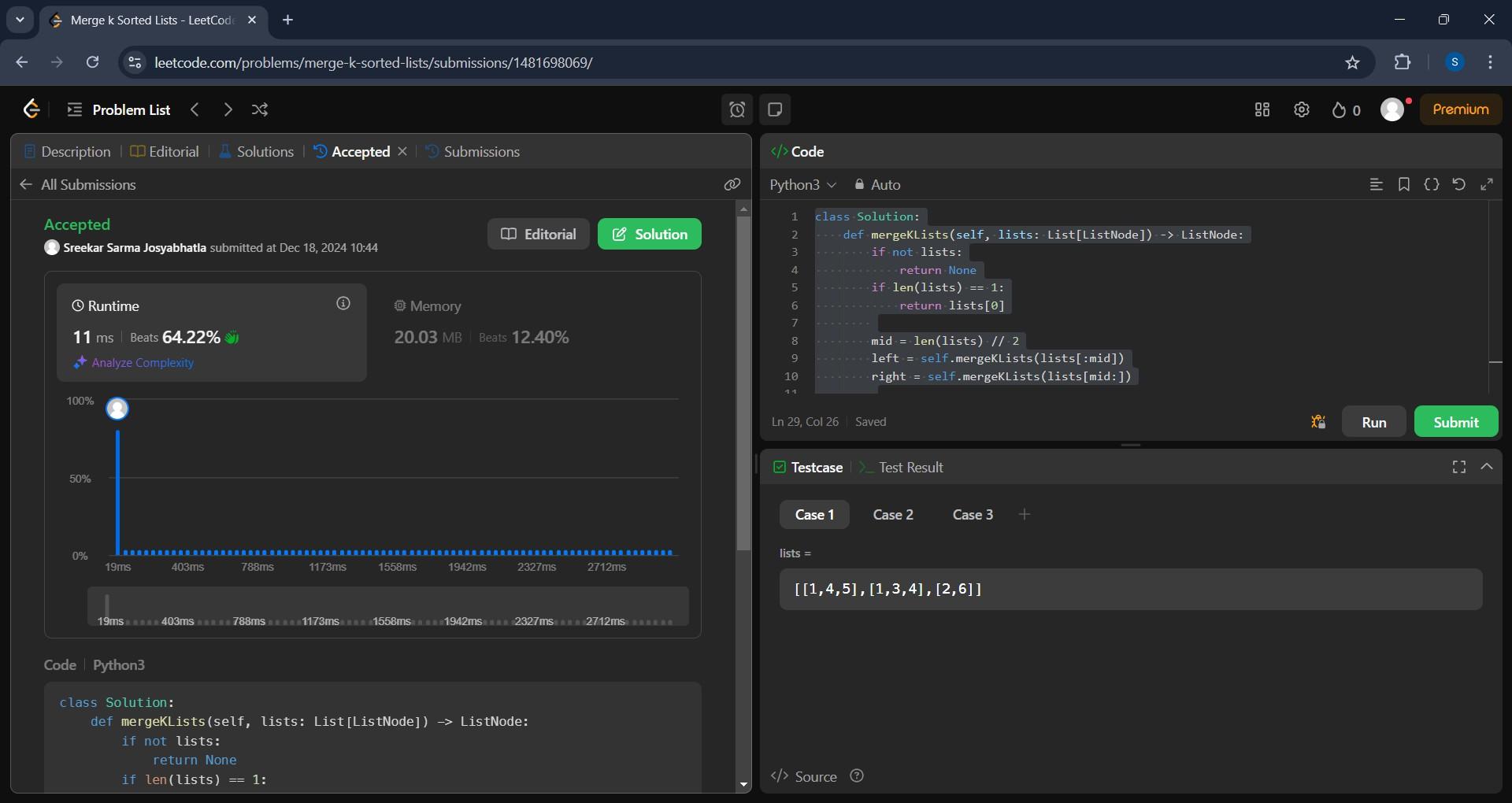
if l1.val < l2.val: curr.next = l1 l1 = l1.next

else:

curr.next = l2 l2 = l2.next

curr = curr.next curr.next = l1 or l2 return dummy.next

OUTPUT



## Design Circular Deque

class MyCircularDeque: def init (self, k: int):

self.d = [0] \* k self.f = 0

self.r = 0

self.sz = 0

**OUTPUT**

self.cap = k

def insertFront(self, v: int) -> bool: if self.isFull(): return False

self.f = (self.f - 1 + self.cap) % self.cap self.d[self.f] = v

self.sz += 1 return True

def insertLast(self, v: int) -> bool: if self.isFull(): return False self.d[self.r] = v

self.r = (self.r + 1) % self.cap self.sz += 1

return True

def deleteFront(self) -> bool:

if self.isEmpty(): return False self.f = (self.f + 1) % self.cap self.sz -= 1

return True

def deleteLast(self) -> bool:

if self.isEmpty(): return False

self.r = (self.r - 1 + self.cap) % self.cap self.sz -= 1

return True

def getFront(self) -> int:

return -1 if self.isEmpty() else self.d[self.f] def getRear(self) -> int:

return -1 if self.isEmpty() else self.d[(self.r - 1 + self.cap) % self.cap] def isEmpty(self) -> bool:

return self.sz == 0 def isFull(self) -> bool:

return self.sz == self.cap

