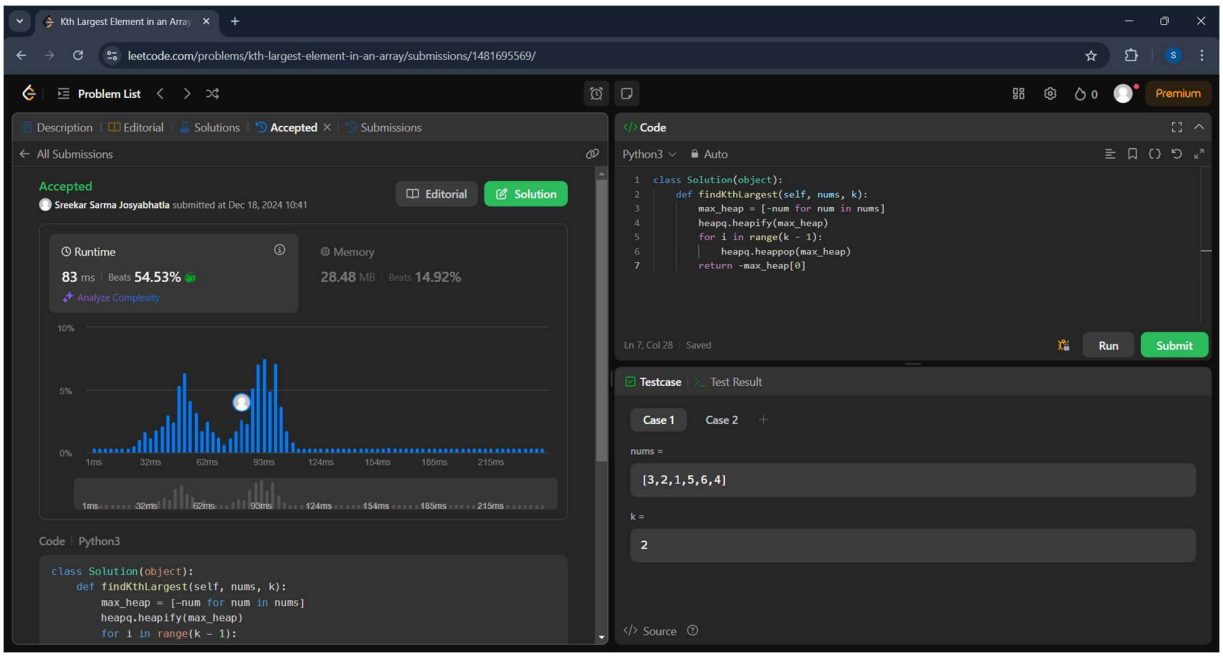
# Advanced Coding -3

VU21CSEN0300056
KIRANMAI TIRUPATI

Kth Largest Element in an Array

```
class Solution(object):
    def findKthLargest(self, nums, k):
        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

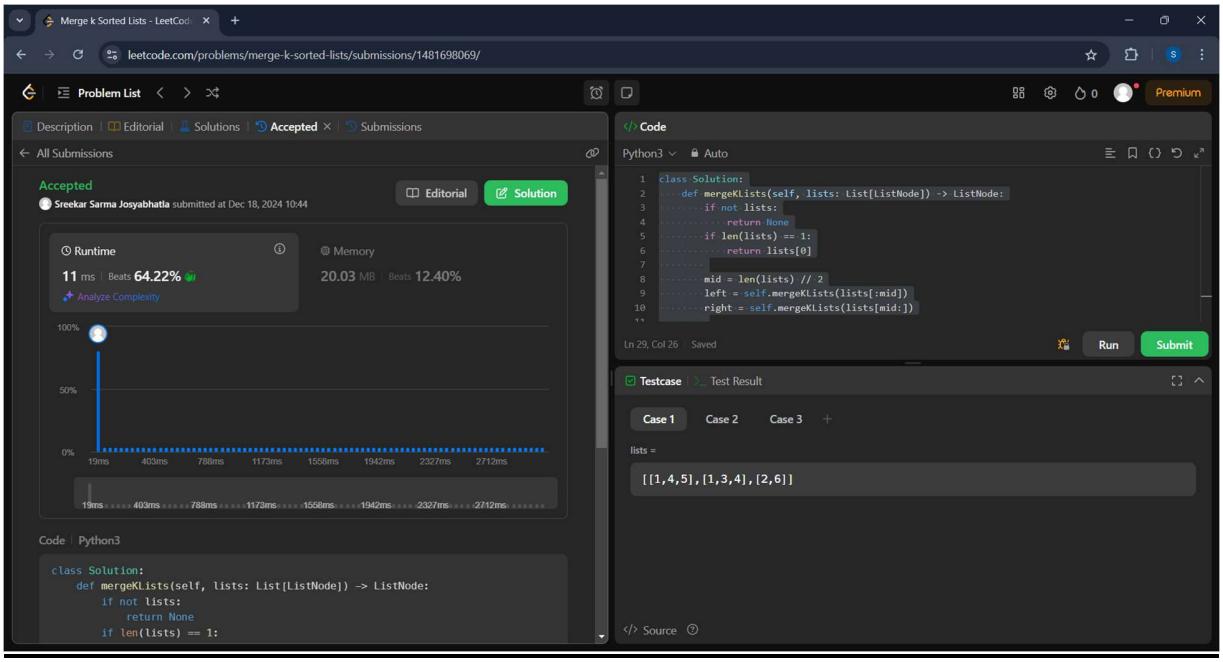



1. **Merge k Sorted Lists**

```
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
```

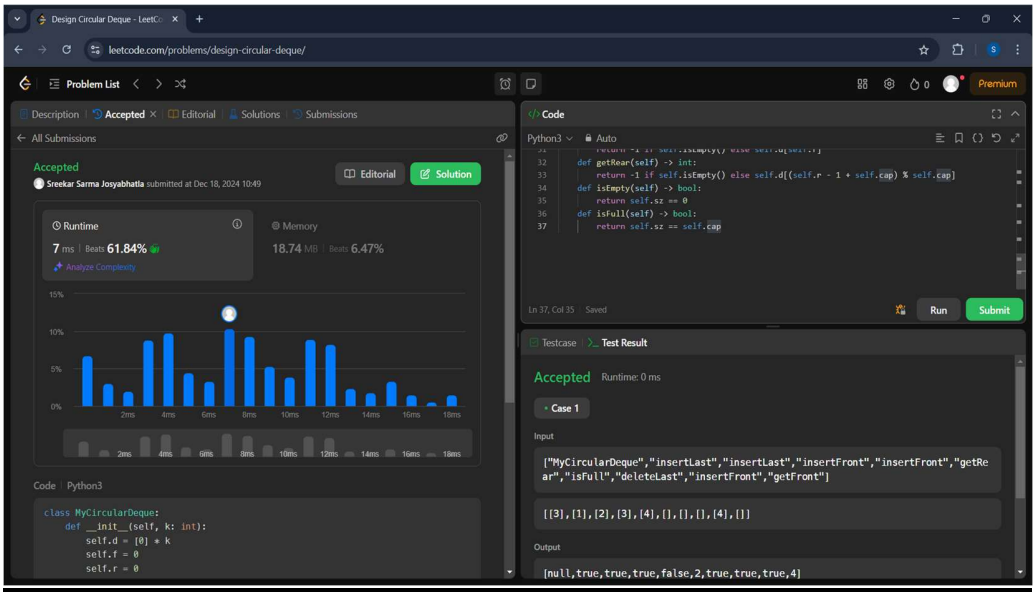
OUTPUT




## 2. Design Circular Deque

```
class MyCircularDeque:
    def __init__(self, k: int):
        self.d = [0] * k
        self.f = 0
        self.r = 0
        self.sz = 0
        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
```

## OUTPUT:


Design Circular Deque - LeetCo
leetcode.com/problems/design-circular-deque/
Problem List
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted
Sreekar Sarma Josyabhatla submitted at Dec 18, 2024 10:49
Editorial
Solution
Runtime
7 ms | Beats 61.84%
Analyze Complexity
Memory
18.74 MB | Beats 6.47%
15%
10%
5%
0%
2ms
4ms
6ms
8ms
10ms
12ms
14ms
16ms
18ms
Code | Python3
class MyCircularDeque:
def __init__(self, k: int):
self.d = [0] * k
self.f = 0
self.r = 0
Code
Python3
Auto
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
Ln 37, Col 35
Saved
Run
Submit
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Input
["MyCircularDeque","insertLast","insertLast","insertFront","insertFront","getRear","isFull","deleteLast","insertFront","getFront"]
[[3],[1],[2],[3],[4],[],[],[],[4],[]]
Output
[null,true,true,true,false,2,true,true,true,4]