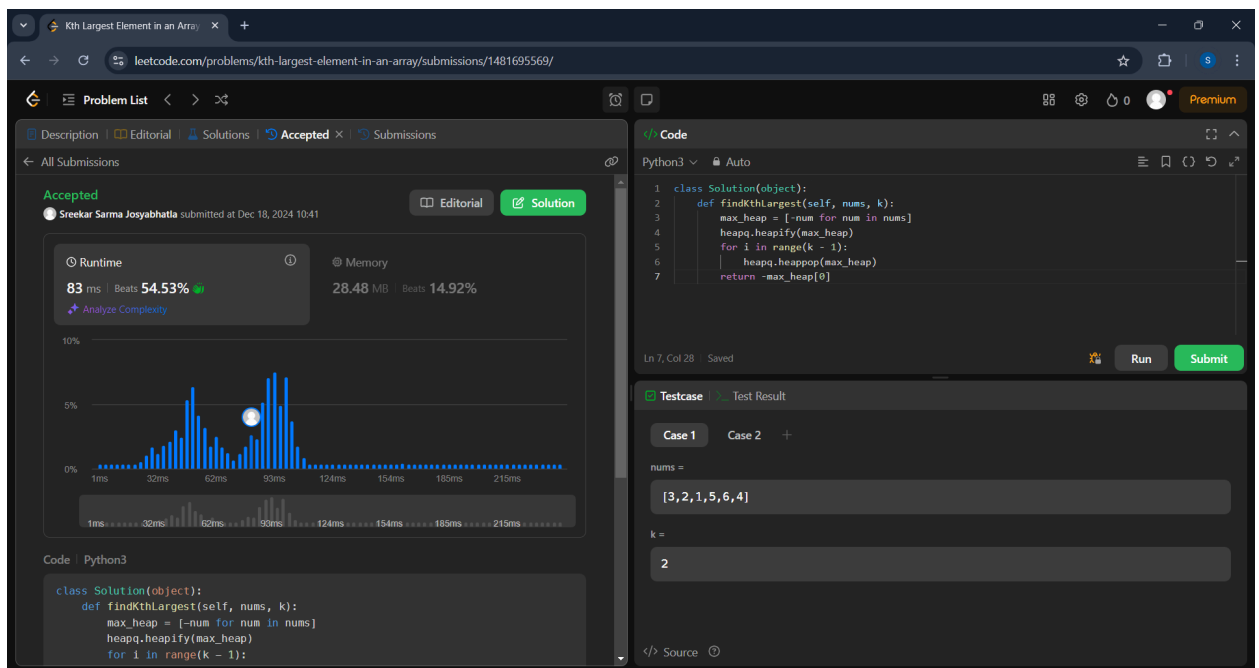


# Advanced Coding -3

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

The screenshot displays the LeetCode interface for the "Merge k Sorted Lists" problem. The submission is by Sreekar Sarma Josyabhatia, submitted on Dec 18, 2024, at 10:44. The submission is marked as "Accepted".

**Runtime:** 11 ms | Beat: 64.22%  
**Memory:** 20.03 MB | Beat: 12.40%

The code is written in Python3 and uses a recursive approach to merge k sorted lists. The test case shows the input lists as `[[1,4,5], [1,3,4], [2,6]]` and the output as a single sorted list.

```
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)
```

## 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:

The screenshot displays the LeetCode interface for the "Design Circular Deque" problem. The submission status is "Accepted", with a runtime of 7 ms (beating 61.84% of solutions) and a memory usage of 18.74 MB (beating 6.47%). The code is written in Python3 and implements a circular deque using a list and pointers. The test result shows a single case passing with the expected output.

**Code:**

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

    def insertFront(self, value):
        if self.isFull():
            return False
        if self.isFrontEmpty():
            self.f -= 1
            if self.f == -1:
                self.f = self.d[-1]
        self.d[self.f] = value
        self.f -= 1
        return True

    def insertLast(self, value):
        if self.isFull():
            return False
        if self.isRearEmpty():
            self.r += 1
            if self.r == self.d[0]:
                self.r = 0
        self.d[self.r] = value
        self.r += 1
        return True

    def deleteFront(self):
        if self.isEmpty():
            return False
        self.f += 1
        if self.f == self.d[0]:
            self.f = 0
        return True

    def deleteLast(self):
        if self.isEmpty():
            return False
        self.r -= 1
        if self.r == -1:
            self.r = self.d[-1]
        return True

    def getFront(self):
        if self.isEmpty():
            return -1
        return self.d[self.f]

    def getRear(self):
        if self.isEmpty():
            return -1
        return self.d[self.r - 1]

    def isEmpty(self) -> bool:
        return self.f == self.r

    def isFull(self) -> bool:
        return (self.f + 1) % len(self.d) == self.r
```

**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]