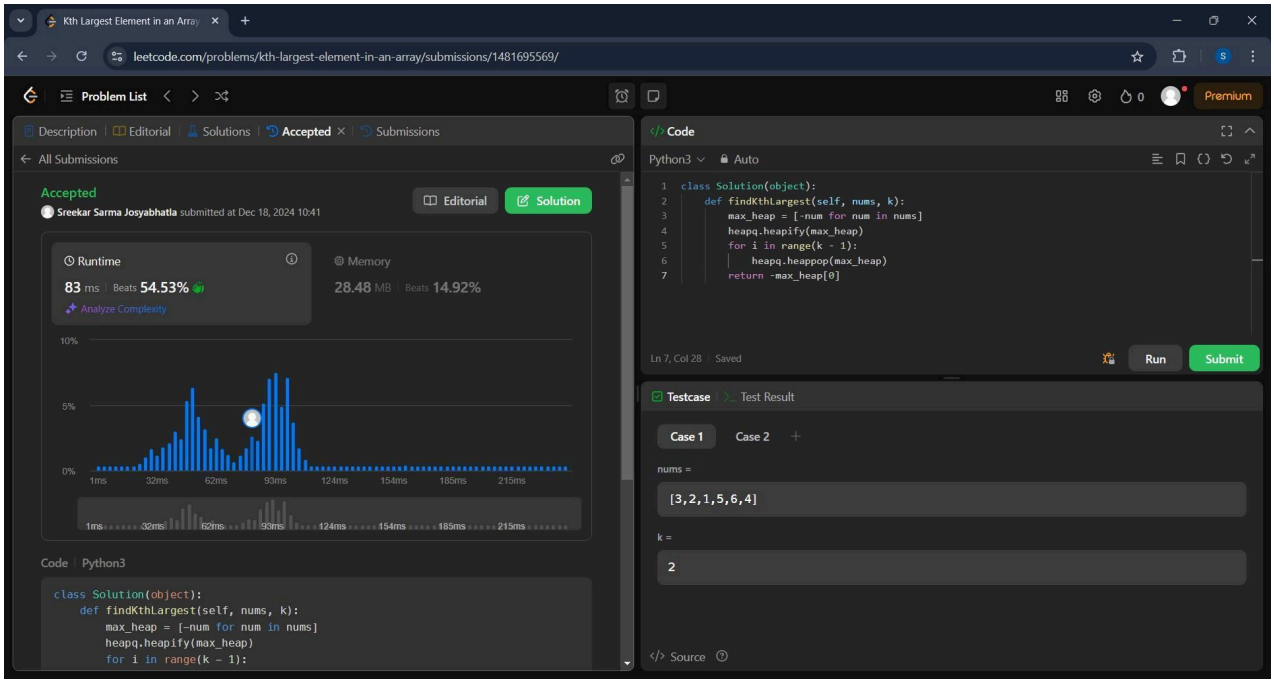
Task-3 **M.Vanshika**

VU21CSEN0300016

1. **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

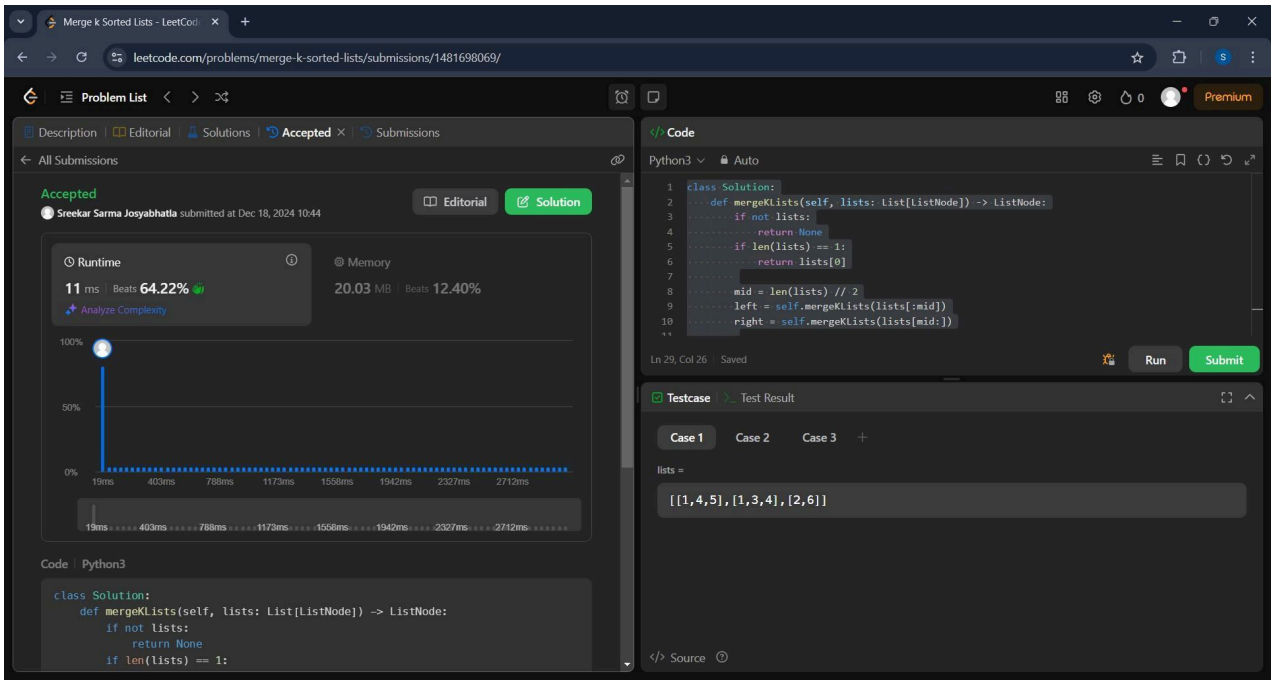


2. **Merge k Sorted Lists**

```
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



3. **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
d
    e
    f
    i
    n
    s
    e
    r
    t
    F
    r
    o
    n
    t
    (
    s
    e
    l
    f
    ,
    v
    :
    i
    n
    t
    )
    -
    >
    b
    o
    o
    l
    :
    i
    f
    s
    e
    l
    f
    .
    i
    s
    F
    u
    l
    l
    (
```

OUTPUT

```
):
return False
self.f=(self.f-1+self.cap)%self.cap
self.d[self.f]=v
self.sz+=1
return True
def insertLas
```

```
t(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:
    i
    f
    s
    e
    l
    f
    .
    i
    s
    E
    m
    p
    t
    y
    (
    )
    :
    r
    e
    t
    u
    r
    n
    F
    a
    l
    s
    e
    s
    e
    l
    f
    .
    f
    =
    (
    s
    e
    l
    f
    .
    f
    +
    1
    )
    %
    s
    e
    l
    f
    .
    c
    a
    p
    s
    e
    l
    f
    .
    s
    z
    -
    =
    1
    return True
def deleteLast(self) -> bool:
    if self.isEmpty(): return False
    s
    e
    l
    f
    .
    r
    =
    (
    s
    e
    l
    f
    .
    r
    -
    1
    +
    s
    e
    l
    f
    .
    c
    a
```

```
    p
    )
    %
    s
    e
    l
    f
    .
    c
    a
    p
    s
    e
    l
    f
    .
    s
    z
    -
    =
    1
    return True
def getFront(self) -> int:
    r
et
ur
n
-1
if
se
lf.i
s
E
m
pt
y(
)
el
se
se
lf.
d[
se
lf.f
]
d
ef
g
et
R
e
ar
(s
elf
)
->
int
:
    return -1 if self.isEmpty()
else self.d[(self.r - 1 +
self.cap) % self.cap] def
isEmpty(self) -> bool:
    r
e
t
u
r
n
s
e
l
f
.
s
z
=
=
0
d
e
f
i
s
F
u
l
l
(
s
e
l
f
)
-
>
b
```

o l

o :

return self.sz == self.cap