0521

215. 数组中的第K个最大元素

力扣 在未排序的数组中找到第 k 个最大的元素。请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

示例 1:

输入: [3,2,1,5,6,4] 和 k = 2 输出: 5 示例 2:

输入: [3,2,3,1,2,4,5,5,6] 和 k = 4 输出: 4

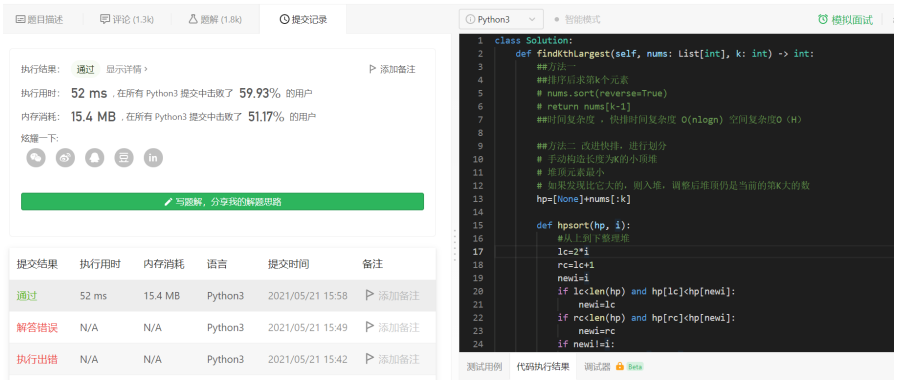
```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        ##方法一
        ##排序后求第k个元素
        # nums.sort(reverse=True)
        # return nums[k-1]
        ##时间复杂度，快排时间复杂度 O(nlogn) 空间复杂度O（H）

        ##方法二 改进快排，进行划分
        # 手动构造长度为K的小顶堆
        # 堆顶元素最小
        # 如果发现比它大的，则入堆，调整后堆顶仍是当前的第K大的数
        hp=[None]+nums[:k]

        def hpsort(hp, i):
            #从上到下整理堆
            lc=2*i
            rc=lc+1
            newi=i
            if lc<len(hp) and hp[lc]<hp[newi]:
                newi=lc
            if rc<len(hp) and hp[rc]<hp[newi]:
                newi=rc
            if newi!=i:
                hp[newi], hp[i]=hp[i], hp[newi]
                hpsort(hp, newi)
        def hpbuild(hp):
            import math
            for i in range(math.floor((len(hp)/2)), 0, -1):
                hpsort(hp, i)

        hpbuild(hp)
        for i in range(k, len(nums)):
```

```
    for i in range(k, len(nums)):
        if nums[i]>hp[1]:
            hp[1]=nums[i]
            hpsort(hp,1)
    return hp[1]
```



141.环形链表

leetcode 给定一个链表，判断链表中是否有环。

如果链表中有某个节点，可以通过连续跟踪 next 指针再次到达，则链表中存在环。 为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 pos 是 -1，则在该链表中没有环。注意：pos 不作为参数进行传递，仅仅是为了标识链表的实际情况。

如果链表中存在环，则返回 true 。 否则，返回 false

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        #链表中某个位置不能两次到达
        #尝试用字典
        #时间复杂度O(N) 空间复杂度O(N)
        # dic = dict()
        # while head is not None:
        #     if head in dic:
        #         return True
        #     else:
        #         dic[head] = 1
        #     head = head.next
        # else:

        #     return False
```

```
#     return False

##空间复杂度为O(1)的解法
##快慢指针
# 确定两个指针初始值
# 假设存在虚拟头节点，使得，慢指针在head 快指针在head->next
if head is None:
    return False
slow = head
quick = head.next
while slow is not quick :
    if quick is None or quick.next is None:
        return False
    slow = slow.next
    quick = quick.next.next
return True
```