

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  $O(n\log n)$  空间复杂度  $O(1)$ 

        ##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]

```

[回 题目描述](#)
[回 评论 \(1.3k\)](#)
[回 题解 \(1.8k\)](#)
[回 提交记录](#)

Python3

智能模式

模拟面试

1

class Solution:

2

def findKthLargest(self, nums: List[int], k: int) -> int:

3

##方法一

4

##排序后求第k个元素

5

nums.sort(reverse=True)

6

return nums[k-1]

7

##时间复杂度 O(nlogn) 空间复杂度 O(1)

8

##方法二 快速选择，进行划分

9

手动构造长度为K的小顶堆

10

堆顶元素最小

11

如果发现比它大的，则入堆，调整后堆顶仍是当前第K大的数

12

hpe=[None]*nums[k]

13

14

15

def hpsort(hp, i):

16

堆排序

17

lc=2*i

18

rc=lc+1

19

newi=i

20

if lc<len(hp) and hp[lc]<hp[newi]:

21

newi=lc

22

if rc<len(hp) and hp[rc]<hp[newi]:

23

newi=rc

24

if newi!=i:

25

hpsort(hp, newi)

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

通过

显示详情

添加备注

执行用时: 52 ms

在所有 Python3 提交中击败了 59.93% 的用户

内存消耗: 15.4 MB

在所有 Python3 提交中击败了 51.17% 的用户

炫耀一下:

👍

👎

👏

👍

👎

👏

与题解, 分享我的解题思路

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	52 ms	15.4 MB	Python3	2021/05/21 15:58	添加备注
解答错误	N/A	N/A	Python3	2021/05/21 15:49	添加备注
执行出错	N/A	N/A	Python3	2021/05/21 15:42	添加备注

[测试用例](#)
[代码执行结果](#)
[调试器](#)
[new](#)

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

```

提交结果

SPRING (1.1k)

剑指 Offer (16.4k)

SQL (1.1k)

剑指 Offer (16.4k)

快慢指针，确立边界条件

执行结果：通过

显示详情 >

快慢指针，确立边界条件

执行用时：84 ms

在所有 Python3 提交中击败了 10.52% 的用户

内存消耗：18.2 MB

在所有 Python3 提交中击败了 42.07% 的用户

炫耀一下：

👍

👎

👤

📄

🔗

与题解，分享你的解题思路

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	84 ms	18.2 MB	Python3	2021/05/21 16:31	快慢指针，确立边界条件
执行出错	N/A	N/A	Python3	2021/05/21 16:30	添加备注
通过	96 ms	18.2 MB	Python3	2021/05/21 16:16	添加备注

python3

剑指 Offer (16.4k)

DEFAUL

```

10 # 快慢指针，确立边界条件
11 # 虚拟头节点
12 # 空间复杂度O(N) 空间复杂度O(N)
13 # dic = dict()
14 # while head is not None:
15 #     if head in dic:
16 #         return True
17 #     else:
18 #         dic[head] = 1
19 #     head = head.next
20 # else:
21 #     return False
22
23 ##空间复杂度为O(1)的解法
24 ##快慢指针
25 # 确定两个指针初始值
26 # 假设存在虚拟头节点，使得，慢指针在head 快指针在head->next
27 if head is None:
28     return False
29 slow = head
30 quick = head.next
31 while slow is not quick :
32     if quick is None or quick.next is None:
33         return False
34     slow = slow.next
35     quick = quick.next.next
36 return True

```

测试用例

代码执行结果

调试器

🔍