215.数组中的第k个最大元素

> 题目：Kth Largest Element in an Array
> 语言: python3
> 英文版链接：https://leetcode.com/problems/kth-largest-element-in-an-array/description/
> 中文版链接：https://leetcode-cn.com/problems/kth-largest-element-in-an-array/

题目分析

本题是经典的**Kth Element**问题，一般常见的做法是先进行排序，然后再进行选择。所以实际本题考察的还是排序问题。当然对于排序，各个语言都已经有底层封装好的函数，**我们要求在本题中不能使用系统库函数自带的排序函数。并且使用快速排序和堆排序两种方法进行解决。**

答案

快速排序

时间复杂度 O(N)，**空间复杂度** O(1)

现在基本的语言中都会自带一个排序函数，而最常用的库排序函数就是快速排序。快速排序也是面试中被问的最多的题目。而针对本题来说，如果使用库函数，那么只能等整个数组排序完成才能取到想要的元素，但实际并不需要整体排序完成，我们说快速排序每一轮确定一个位置，如果我们想要位置的元素被确定了，我们就可以提前结束排序。

代码请参考：

```python
class Solution:

    def findKthLargest(self, nums: 'List[int]', k: 'int') -> 'int':
        return self.quick_sort(nums, k)

    def quick_sort(self, nums, k):
        # 这个位置的元素是我们想要的结果元素
        k = len(nums) - k
        left = 0
        right = len(nums) - 1
        while left < right:
            j = self.partition(nums, left, right)
            if j == k:
                break
            elif j < k:
                left = j + 1
            else:
                right = j - 1
        return nums[k]

    def partition(self, nums, left, right):
        while True:
            while nums[left] < nums[right]:
                right -= 1
            else:
                nums[left], nums[right] = nums[right], nums[left]
                if left >= right:
                    break
                left += 1

            while nums[left] < nums[right]:
                left += 1
            else:
                nums[left], nums[right] = nums[right], nums[left]
                if left >= right:
                    break
                right -= 1

        return left
```

堆排序

时间复杂度： O(NlogK)，**空间复杂度**： O(K)。

对于堆排序一般升序采用大顶堆，降序采用小顶堆。本题中想要寻找的是第K个最大的元素，自然需要的是升序排列，因而采用**大顶堆**。当然本题并不是一个纯粹的排序题，因而并不需要全部排序完成，只需进行k次交换，即可找到第k个最大的元素。

代码请参考：

```python
class Solution:

    def findKthLargest(self, nums: 'List[int]', k: 'int') -> 'int':
        return self.heap_sort(nums, k)

    def heap_sort(self, nums, k):
        # 构建大顶堆
        for i in range(len(nums) // 2 - 1, -1, -1):
            self.heap_adjust(nums, i, len(nums))

        cnt = 0
        # 交换堆顶元素，然后重新调整堆结构
        for i in range(len(nums) - 1, 0, -1):
            self.heap_swap(nums, 0, i)
            cnt += 1
            if cnt == k:
                return nums[i]
            self.heap_adjust(nums, 0, i)

        # 说明k==len(nums)
        return nums[0]

    def heap_adjust(self, nums, start, length):
        tmp = nums[start]
        k = start * 2 + 1
        while k < length:  # 对于完全二叉树来讲，没有左节点一点没有右节点
            # 当前节点左节点序号
            left = start * 2 + 1
            # 当前节点右节点序号
            right = left + 1

            if right < length and nums[right] > nums[left]:
                # 如果右节点比较大
                k = right

            if nums[k] > tmp:
                # 如果子节点比父节点大，则将子节点赋值给父节点
                nums[start] = nums[k]
                start = k
            else:
                break

            k = k * 2 + 1

        nums[start] = tmp

    def heap_swap(self, nums, i, j):
        nums[i], nums[j] = nums[j], nums[i]
        return nums
```