Listing : **2.py**

```
# 设计一个找到数据流中第 k 大元素的类 ()。注意是排序后的第class k 大元素，不是第 k 个不同的元素。
#
# 请实现    类： KthLargest
#
# KthLargest(int k, int[] nums) 使用整数 k 和整数流 nums 初始化对象。
# int add(int val) 将 val 插入数据流 nums 后，返回当前数据流中第 k 大的元素。
#
# 示例:
#
# 输入:
# ["KthLargest", "add", "add", "add", "add", "add"]
# [[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
# 输出:
# [null, 4, 5, 5, 8, 8]
#
# 解释:
# KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);
# kthLargest.add(3);   // return 4
# kthLargest.add(5);   // return 5
# kthLargest.add(10);  // return 5
# kthLargest.add(9);   // return 8
# kthLargest.add(4);   // return 8
#
# 提示:
# 1 <= k <= 104
# 0 <= nums.length <= 104
# -104 <= nums[i] <= 104
# -104 <= val <= 104
# 最多调用 add 方法 104 次
# 题目数据保证，在查找第 k 大元素时，数组中至少有 k 个元素

# 方法一：优先队列
# 我们可以使用一个大小为 kk 的优先队列来存储前 kk 大的元素，其中优先队列的队头为队列中最小的元素，也就是第 kk 大的元素。
#
# 在单次插入的操作中，我们首先将元素 \textit{val}val 加入到优先队列中。如果此时优先队列的大小大于，我们需要将优先队列的队头元素弹出，以保证优先队列的大小为 kk 。kk
#
# 复杂度分析
#
# 时间复杂度:
#
# 初始化时间复杂度为: O(n \log k)O(nlogk) ，其中 nn 为初始化时 \textit{nums}nums 的长度;
#
# 单次插入时间复杂度为: O(\log k)O(logk)。
#
# 空间复杂度: O(k)O(k)。需要使用优先队列存储前 kk 大的元素。

class KthLargest( object):
```

```
68              self.myhq.pop()
69          return self.myhq.heap[0]
70
71  class hq( object):
72      def __init__(self):
73          self.heap = []
74
75      def _shift_up(self,index):
76          while index > 0:
77              parent = (index-1)//2
78              if self.heap[parent] < self.heap[index]:
79                  break
80
81              self.heap[index],self.heap[parent] = self.heap[parent],self.heap
                    [index]
82              index = parent
83
84      def _shift_down(self,index):
85          while index*2 + 1 <  len(self.heap):
86              left = index*2 + 1
87              right = index*2 + 2
88              parent = index
89              smallest = parent
90              if self.heap[left] < self.heap[parent]:
91                  smallest = left
92              if right <
                    len(self.heap) and self.heap[right] < self.heap[smallest]:
93                  smallest = right
94              if smallest == parent:
95                  break
96              self.heap[parent],self.heap[smallest] = self.heap[smallest],self
                    .heap[parent]
97              index = smallest
98
99      def pop(self):
100         last =  len(self.heap) - 1
101         self.heap[0],self.heap[last] = self.heap[last],self.heap[0]
102         peek = self.heap.pop()
103         self._shift_down(0)
```

```
104         return peek
105
106     def push(self,data):
107         self.heap.append(data)
108         self._shift_up( len(self.heap)-1)
109 # Your KthLargest object will be instantiated and called as such:
110 # obj = KthLargest(k, nums)
111 # param_1 = obj.add(val)
```

# Code Insert

Discount

2021 年 4 月 2 日