LeetCode #703 数据流中的第 K 大元素

作者：matthewhan
原文链接：https://ld246.com/article/1591592165371
来源网站：链滴

Problem Description

设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的素。

你的 KthLargest 类需要一个同时接收整数 k 和整数数组nums 的构造器，它包含数据流中的初始元。每次调用 KthLargest.add，返回当前数据流中第 k 大的元素。

note

你可以假设 nums 的长度 ≥ k - 1 且 k ≥ 1。

e.g.

```
int k = 3;
int[] arr = [4,5,8,2];
KthLargest kthLargest = new KthLargest(3, arr);
kthLargest.add(3);   // returns 4
kthLargest.add(5);   // returns 5
kthLargest.add(10);  // returns 5
kthLargest.add(9);   // returns 8
kthLargest.add(4);   // returns 8
```

Solution

1. 额外数组空间

```
class KthLargest {
```

```
    private int len;
    private final int index;
    private final List<Integer> newData;

    public KthLargest(int k, int[] nums) {
        index = k;
        len = nums.length;
        newData = new ArrayList<>(nums.length);
        for (int num : nums) {
            newData.add(num);
        }
    }

    public int add(int val) {
        len++;
        newData.add(val);
        Collections.sort(newData);
        return newData.get(len - index);
    }
}
```

不过还是用原始数组，更底层更快一点：

```
class KthLargest {

    private int len;
    private int[] data;
    private final int index;

    public KthLargest(int k, int[] nums) {
        index = k;
        len = nums.length;
        data = nums;
    }

    public int add(int val) {
        len++;
        int[] temp = new int[len];
        System.arraycopy(data, 0, temp, 0, data.length);
        temp[len - 1] = val;
        Arrays.sort(temp);
        data = new int[len];
        System.arraycopy(temp, 0, data, 0, temp.length);
        return data[len - index];
    }
}
```

2. 小顶堆

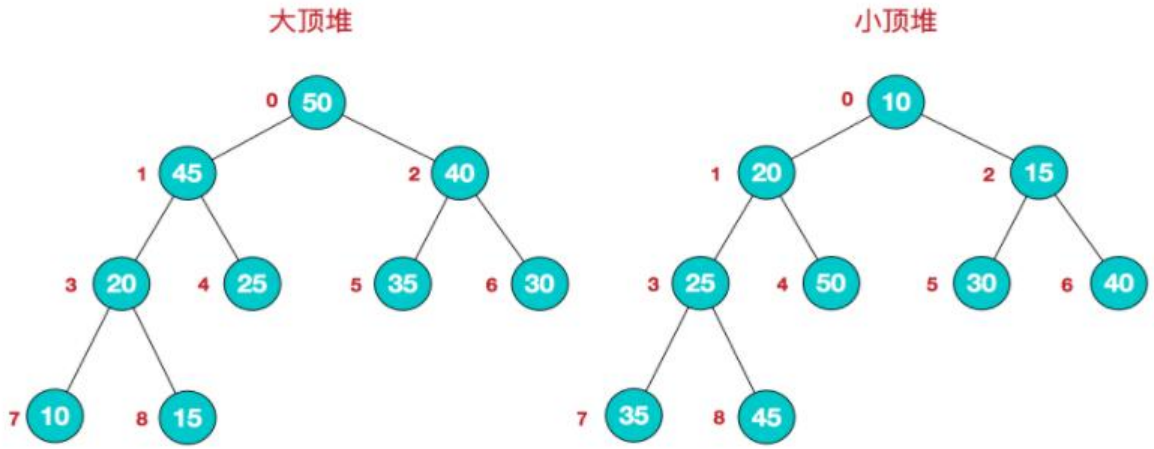
题目中表述了 add方法返回第 k 大元素即可，所以其实我们不必维护整个数组，只需要维护 k 个从到小的元素即可。表面 add方法而已，不是真的新增。

什么是堆？堆是一种**非线性结构**，可以把堆看作一个数组，也可以被看作一个完全二叉树，通俗来讲

其实就是利用完全二叉树的结构来维护的一维数组。

按照堆的特点可以把堆分为**大顶堆**和**小顶堆**：

1. 大顶堆：每个结点的值都 **大于**或**等于**其左右孩子结点的值
2. 小顶堆：每个结点的值都 **小于**或**等于**其左右孩子结点的值



直接说用法：

- **升序：使用大顶堆**
- **降序：使用小顶堆**

我们这边每次 add方法需要返回第 k 个从大到小元素，所以每次执行 add方法需要保留 k 个降序的素，这里就是采用了小顶堆。我们可以用优先队列 PriorityQueue<Integer>来做（大顶堆的话需要写排序方法，默认是升序）。**优先队列的作用是保证每次取出的元素都是队列中权值最小的。**

当堆内元素的个数小于等于 k 时，调用 add方法入队；当第 k + 1 个元素想入队时，和队首比较大小队首是最小的）。比队首大，队首元素出队，新元素入队；比队首小，不作任何操作。这样就能保证列的元素个数是 k 个，并且是队首是最小的（不一定是从小到大排序，**但是队首一定是整个队列里最的**）。

所以就是两步：

1. 控制队列大小为 k
2. 保证队首的元素比新加入的元素大

所以整个数组的第 k 大元素就是队列的队首 peek()方法。

```
class KthLargest {

    // 小顶堆
    private PriorityQueue<Integer> q;
    private int k;

    public KthLargestElementInaStream(int k, int[] nums) {
        this.k = k;
        q = new PriorityQueue<Integer>(k);
        for (int i : nums) {
```

```
            add(i);
        }
    }
    public int add(int val) {
        if (q.size() < k) {
            q.offer(val);

        } else if (q.peek() < val) {
            q.poll();
            q.offer(val);
        }
        return q.peek();
    }
}
```

补充

PriorityQueue<Integer> queue = new PriorityQueue<>(1);这样一个优先队列，将数组 [8, 5, 4, 1, 3] 按顺序执行 offer(element)方法，打印的结果是 [3, 4, 5, 12, 8]，在执行 offer(1)，打印的结果是 1, 4, 3, 12, 8, 5]。排序就是在二叉树的基础上，叶子节点和父节点比大小，比父节点大，交换位置，到比父节点小或到根节点。