```
                } else {
                    parent.right = null;
                }
                last = newLast;
            }
            size--;
            return node;
        }

    }
```

随时找到数据流的中位数

【题目】

有一个源源不断地吐出整数的数据流，假设你有足够的空间来保存吐出的数。请设计一个名叫 MedianHolder 的结构，MedianHolder 可以随时取得之前吐出所有数的中位数。

【要求】

1．如果 MedianHolder 已经保存了吐出的 N 个数，那么任意时刻将一个新数加入到 MedianHolder 的过程，其时间复杂度是 $O(\log N)$。

2．取得已经吐出的 N 个数整体的中位数的过程，时间复杂度为 $O(1)$。

【难度】

将 ★★★★

【解答】

本书设计的 MedianHolder 中有两个堆，一个是大根堆，一个是小根堆。大根堆中含有接收的所有数中较小的一半，并且按大根堆的方式组织起来，那么这个堆的堆顶就是较小一半的数中最大的那个。小根堆中含有接收的所有数中较大的一半，并且按小根堆的方式组织起来，那么这个堆的堆顶就是较大一半的数中最小的那个。

例如，如果已经吐出的数为 6，1，3，0，9，8，7，2。

较小的一半为：0，1，2，3，那么 3 就是这一半的数组成的大根堆的堆顶。

较大的一半为：6，7，8，9，那么 6 就是这一半的数组成的小根堆的堆顶。

因为此时数的总个数为偶数，所以中位数就是两个堆顶相加，再除以 2。

如果此时新加入一个数 10，那么这个数应该放进较大的一半里，所以此时较大一半的数为：6，7，8，9，10。此时 6 依然是这一半的数组成的小根堆的堆顶，因为此时数的总个数为奇数，所以中位数应该是正好处在中间位置的数，而此时大根堆有 4 个数，小根堆有 5 个数，那么小根堆的堆顶 6 就是此时的中位数。如果此时又新加入了一个数 11，那么这个数也应该放进较大的一半里，此时较大一半的数为：6，7，8，9，10，11。这时小根堆大小为 6，而大根堆的大小为 4，所以要进行如下调整：

1．如果大根堆的 size 比小根堆的 size 大 2，那么从大根堆里将堆顶弹出，并放入小根堆里。

2．如果小根堆的 size 比大根堆的 size 大 2，那么从小根堆里将堆顶弹出，并放入大根堆里。

进行这样的调整后，大根堆和小根堆的 size 相同。

总结如下：

1．大根堆每时每刻都是较小的一半的数，堆顶为这一堆数的最大值。

2．小根堆每时每刻都是较大的一半的数，堆顶为这一堆数的最小值。

3．新加入的数根据与两个堆的堆顶的大小关系，选择放进大根堆或者小根堆里。

4．当任何一个堆的 size 比另一个的 size 大 2 时，进行如上调整过程。

这样随时都可以知道已经吐出的所有数处于中间位置的两个数是什么，取得中位数的操作时间复杂度为 $O(1)$，同时根据堆的性质，向堆中加一个新的数，并且调整堆的代价为 $O(\log N)$。然而题目有一个很重要的限制“任何时刻将一个新数加入到 MedianHolder 的过程，时间复杂度是 $O(\log N)$”，为了做到“任何时刻”的要求，那么堆的设计不能采用固定数组的实现方式，因为会有扩容的代价，但是在 Java 中诸如优先级队列（PriorityQueue）等很多库提供的数据结构却都是使用固定数组的方式实现的。所以严格地说，使用这些结构的实现并不符合题目要求。本书“设计一个没有扩容负担的堆结构”问题中完成了符合要求的堆结构实现，即其中的 MyHeap 类，请读者先理解这个类的实现，然后参看本题的实现（即如下代码中的 MedianHolder 类）。

```
public class MedianHolder {
        private MyHeap<Integer> minHeap;
        private MyHeap<Integer> maxHeap;

        public MedianHolder() {
                this.minHeap = new MyHeap<Integer>(new MinHeapComparator());
                this.maxHeap = new MyHeap<Integer>(new MaxHeapComparator());
        }
```

```
        public void addNumber(Integer num) {
                if (this.maxHeap.isEmpty()) {
                        this.maxHeap.add(num);
                        return;
                }
                if (this.maxHeap.getHead() >= num) {
                        this.maxHeap.add(num);
                } else {
                        if (this.minHeap.isEmpty()) {
                                this.minHeap.add(num);
                                return;
                        }
                        if (this.minHeap.getHead() > num) {
                                this.maxHeap.add(num);
                        } else {
                                this.minHeap.add(num);
                        }
                }
                this.modifyTwoHeapsSize();
        }

        public Integer getMedian() {
                long maxHeapSize = this.maxHeap.getSize();
                long minHeapSize = this.minHeap.getSize();
                if (maxHeapSize + minHeapSize == 0) {
                        return null;
                }
                Integer maxHeapHead = this.maxHeap.getHead();
                Integer minHeapHead = this.minHeap.getHead();
                if (((maxHeapSize + minHeapSize) & 1) == 0) {
                        return (maxHeapHead + minHeapHead) / 2;
                } else if (maxHeapSize > minHeapSize) {
                        return maxHeapHead;
                } else {
                        return minHeapHead;
                }
        }

        private void modifyTwoHeapsSize() {
                if (this.maxHeap.getSize() == this.minHeap.getSize() + 2) {
                        this.minHeap.add(this.maxHeap.popHead());
                }
                if (this.minHeap.getSize() == this.maxHeap.getSize() + 2) {
                        this.maxHeap.add(this.minHeap.popHead());
                }
        }

}

//生成大根堆的比较器
public class MaxHeapComparator implements Comparator<Integer> {
        @Override
```

```
        public int compare(Integer o1, Integer o2) {
                if (o2 > o1) {
                        return 1;
                } else {
                        return -1;
                }
        }
}

//生成小根堆的比较器
public class MinHeapComparator implements Comparator<Integer> {
        @Override
        public int compare(Integer o1, Integer o2) {
                if (o2 < o1) {
                        return 1;
                } else {
                        return -1;
                }
        }
}
```

在两个长度相等的排序数组中找到上中位数

【题目】

给定两个有序数组 arr1 和 arr2，已知两个数组的长度都为 N，求两个数组中所有数的上中位数。

【举例】

arr1=[1,2,3,4]，arr2=[3,4,5,6]

总共有 8 个数，那么上中位数是第 4 小的数，所以返回 3。

arr1=[0,1,2]，arr2 =[3,4,5]

总共有 6 个数，那么上中位数是第 3 小的数，所以返回 2。

【要求】

时间复杂度为 $O(\log N)$，额外空间复杂度为 $O(1)$。

【难度】

尉 ★★☆☆