Solution 1

```python
        self.updateMedian()

    def rebalanceHeaps(self):
        if self.lowers.length - self.greaters.length == 2:
            self.greaters.insert(self.lowers.remove())
        elif self.greaters.length - self.lowers.length == 2:
            self.lowers.insert(self.greaters.remove())

    def updateMedian(self):
        if self.lowers.length == self.greaters.length:
            self.median = (self.lowers.peek() + self.greaters.peek()) / 2
        elif self.lowers.length > self.greaters.length:
            self.median = self.lowers.peek()
        else:
            self.median = self.greaters.peek()

    def getMedian(self):
        return self.median


class Heap:
    def __init__(self, comparisonFunc, array):
        self.heap = self.buildHeap(array)
        self.comparisonFunc = comparisonFunc
        self.length = len(self.heap)

    def buildHeap(self, array):
        firstParentIdx = (len(array) - 2) // 2
        for currentIdx in reversed(range(firstParentIdx + 1)):
            self.siftDown(currentIdx, len(array) - 1, array)
        return array

    def siftDown(self, currentIdx, endIdx, heap):
        childOneIdx = currentIdx * 2 + 1
        while childOneIdx <= endIdx:
            childTwoIdx = currentIdx * 2 + 2 if currentIdx * 2 + 2 <= endIdx else -1
            if childTwoIdx != -1:
                if self.comparisonFunc(heap[childTwoIdx], heap[childOneIdx]):
                    idxToSwap = childTwoIdx
                else:
                    idxToSwap = childOneIdx
            else:
                idxToSwap = childOneIdx
            if self.comparisonFunc(heap[idxToSwap], heap[currentIdx]):
                self.swap(currentIdx, idxToSwap, heap)
                currentIdx = idxToSwap
                childOneIdx = currentIdx * 2 + 1
            else:
                return

    def siftUp(self, currentIdx, heap):
        parentIdx = (currentIdx - 1) // 2
        while currentIdx > 0:
            if self.comparisonFunc(heap[currentIdx], heap[parentIdx]):
                self.swap(currentIdx, parentIdx, heap)
                currentIdx = parentIdx
                parentIdx = (currentIdx - 1) // 2
            else:
                return

    def peek(self):
        return self.heap[0]

    def remove(self):
        self.swap(0, self.length - 1, self.heap)
        valueToRemove = self.heap.pop()
        self.length -= 1
        self.siftDown(0, self.length - 1, self.heap)
        return valueToRemove

    def insert(self, value):
        self.heap.append(value)
        self.length += 1
        self.siftUp(self.length - 1, self.heap)

    def swap(self, i, j, array):
        array[i], array[j] = array[j], array[i]


def MAX_HEAP_FUNC(a, b):
    return a > b


def MIN_HEAP_FUNC(a, b):
    return a < b
```