Solution 1

```swift
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class Program {
    class ContinuousMedianHandler {
        var median: Double
        var lowers: Heap
        var greaters: Heap

        init() {
            median = 0.0

            lowers = Heap(array: [], comparisonFunction: Program.maxHeapFunc(_:_:))
            greaters = Heap(array: [], comparisonFunction: Program.minHeapFunc(_:_:))
        }

        func getMedian() -> Double {
            return median
        }

        // O(log(n)) time | O(n) space
        func insert(number: Int) {
            if lowers.length == 0 || number < Int(lowers.peek()) {
                lowers.insert(value: number)
            } else {
                greaters.insert(value: number)
            }

            rebalanceHeaps()
            updateMedian()
        }

        func rebalanceHeaps() {
            if lowers.length - greaters.length == 2 {
                greaters.insert(value: lowers.remove())
            } else if greaters.length - lowers.length == 2 {
                lowers.insert(value: greaters.remove())
            }
        }

        func updateMedian() {
            if lowers.length == greaters.length {
                median = Double((lowers.peek() + greaters.peek()) / 2)
            } else if lowers.length > greaters.length {
                median = Double(lowers.peek())
            } else {
                median = Double(greaters.peek())
            }
        }
    }

    class Heap {
        var length = 0
        var heap = [Int]()
        var comparisonFunction: (Int, Int) -> Bool
        typealias comparisonFuncTypeAlias = (Int, Int) -> Bool

        init(array: [Int], comparisonFunction: @escaping comparisonFuncTypeAlias) {
            self.comparisonFunction = comparisonFunction
            heap = buildHeap(array: array)
            length = heap.count
        }

        func buildHeap(array: [Int]) -> [Int] {
            var heapToReturn = array

            var firstParentIndex = Double((array.count - 2) / 2)
            firstParentIndex = firstParentIndex.rounded(.down)

            if array.count > 0 {
                for var currentIndex in (0 ... Int(firstParentIndex)).reversed() {
                    var endIndex = array.count - 1

                    siftDown(currentIndex: &currentIndex, endIndex: &endIndex, heap: &heapToReturn)
                }
            }

            return heapToReturn
        }

        func siftDown(currentIndex: inout Int, endIndex: inout Int, heap: inout [Int]) {
            var firstChildIndex = (2 * currentIndex) + 1

            while firstChildIndex <= endIndex {
                var secondChildIndex = -1

                let potentialSecondChild = (2 * currentIndex) + 2

                if potentialSecondChild <= endIndex {
                    secondChildIndex = potentialSecondChild
                }

                var indexToSwap = -1

                if secondChildIndex != -1, comparisonFunction(heap[secondChildIndex], heap[firstChildIndex]) {
                    indexToSwap = secondChildIndex
                } else {
                    indexToSwap = firstChildIndex
                }

                if comparisonFunction(heap[indexToSwap], heap[currentIndex]) {
                    swap(firstIndex: currentIndex, secondIndex: indexToSwap, heap: &heap)

                    currentIndex = indexToSwap

                    firstChildIndex = (2 * currentIndex) + 1
                } else {
                    return
                }
            }
        }

        func siftUp(currentIndex: inout Int, heap: inout [Int]) {
            var parentIndex = Double((currentIndex - 1) / 2)
            parentIndex = parentIndex.rounded(.down)

```

```
116             while currentIndex > 0 {
117                 if comparisonFunction(heap[currentIndex], heap[Int(parentIndex)]) {
118                     swap(firstIndex: currentIndex, secondIndex: Int(parentIndex), heap: &heap)
119 
120                     currentIndex = Int(parentIndex)
121 
122                     parentIndex = Double((currentIndex - 1) / 2)
```