Documentation

The MedianFinder class is designed to efficiently manage a dynamic data stream and calculate the median of the numbers added to it. A median is defined as the middle value in an ordered list, or the average of the two middle values when the list has an even length. This class provides an optimized solution to this problem by maintaining two heaps — a max-heap and a min-heap — to store the elements in a way that allows the median to be calculated in constant time, while still being able to add new numbers efficiently.

The class uses a two-heap approach, where the max-heap is used to store the smaller half of the numbers and the min-heap stores the larger half. The key idea is that the max-heap always holds the largest value in the lower half of the numbers, and the min-heap holds the smallest value in the upper half. The median can be easily found by maintaining this balance between the two heaps. If the total number of elements is odd, the root of the max-heap gives the median, while if the number of elements is even, the median is the average of the roots of both heaps.

The method add (num) adds a number to the appropriate heap based on its value compared to the current median. After each insertion, the heaps are rebalanced to ensure that their sizes differ by no more than one element. This ensures that the max-heap can have at most one more element than the min-heap, which allows the median to be determined quickly. If the sizes of the heaps become unbalanced, elements are transferred from one heap to the other to restore the balance.

The method findMedian() calculates the median based on the current state of the two heaps. If the number of elements is odd, the median is simply the root of the max-heap. If the number of elements is even, the median is the average of the roots of the max-heap and the min-heap. Since the heaps are balanced, the median can be computed in constant time, ensuring efficiency even as more numbers are added to the stream.

Overall, the MedianFinder class provides an optimized solution for finding the median of a data stream. The use of two heaps ensures that both the insertion of numbers and the calculation of the median are performed efficiently. The time complexity for adding a number is O(logn)O(log n), and the time complexity for finding the median is O(1)O(1). This approach is highly efficient for handling large data streams and provides a practical solution for real-time median calculation.