# 题目

中位数是有序列表中间的数。如果列表长度是偶数，中位数则是中间两个数的平均值。

**例如：**

[2,3,4] 的中位数是 3

[2,3] 的中位数是 (2 + 3) / 2 = 2.5

设计一个支持以下两种操作的数据结构：

void addNum(int num) - 从数据流中添加一个整数到数据结构中。

double findMedian() - 返回目前所有元素的中位数。

**示例：**

addNum(1)

addNum(2)

findMedian() -> 1.5

addNum(3)

findMedian() -> 2

**进阶:**

如果数据流中所有整数都在0到100范围内，你将如何优化你的算法？

如果数据流中99%的整数都在0到100范围内，你将如何优化你的算法？

# 分析

## 方法一：优先队列

**思路：**

我们用两个优先队列queMax和queMin分别记录大于中位数的数和小于等于中位数的数。当累计添加的数的数量为奇数时，queMin中的数的数量比queMax多一个，此时中位数为queMin的队头。当累计添加的数的数量为偶数时，两个优先队列中的数的数量相同，此时中位数为它们的队头的平均值。

当我们尝试添加一个数num到数据结构中，我们需要分情况讨论：

1、num≤max{queMin}

此时num小于等于中位数，我们需要将该数添加到queMin中。新的中位数将小于等于原来的中位数，因此我们可能需要将queMin 中最大的数移动到queMax中。

2、num>max{queMin}

此时num大于中位数，我们需要将该数添加到queMin中。新的中位数将大于等于原来的中位数，因此我们可能需要将queMax中最小的数移动到queMin中。

特别地，当累计添加的数的数量为0时，我们将num添加到queMin中。

**代码：**

class MedianFinder {

public:

priority\_queue<int, vector<int>, less<int>> queMin;

priority\_queue<int, vector<int>, greater<int>> queMax;

MedianFinder() {}

void addNum(int num) {

if (queMin.empty() || num <= queMin.top()) {

queMin.push(num);

if (queMax.size() + 1 < queMin.size()) {

queMax.push(queMin.top());

queMin.pop();

}

} else {

queMax.push(num);

if (queMax.size() > queMin.size()) {

queMin.push(queMax.top());

queMax.pop();

}

}

}

double findMedian() {

if (queMin.size() > queMax.size()) {

return queMin.top();

}

return (queMin.top() + queMax.top()) / 2.0;

}

};

**复杂度分析:**

时间复杂度：

addNum: O(logn)，其中n为累计添加的数的数量。

findMedian: O(1)。

空间复杂度：O(n)，主要为优先队列的开销。