数据流中的中位数

> 计算数据流中的中位数

分析： 假设将数据倒入一个三角形漏斗中，在这个漏斗中数据自动排序，那么中位数一定在三角形中间部分。 为方便获取三角形的中间部分值，我们将这个三角形切开，构成两个堆。 **对三角形的部分，可以看做一个大根堆，存放数值较小的部分；梯形部分看做一个小根堆，存放数值较大的部分。** 这样：

- 对大根堆，因为存放的数值较小，所以堆顶元素相较于其他部分值较大；
- 对小根堆，存放数值较大，所以堆顶元素相较于其他部分较小。
- 这样就保证了中间部分的数据，都在堆顶部分。

对中位数的获取：

1. 如果两个堆大小相同，中位数为两个堆的堆顶之和的一半；
2. 否则取两个堆中数据较多的堆顶即为所求中位数。

如何向堆中增加数据：

1. 不能直接将数据加入到大根堆或小根堆中；
2. 而是将数据加入到两者之间元素较多的一个堆中，同时将该堆的堆顶元素压入到另外一个堆中； 如：如果加入小根堆，那么无论加入的元素是否大于堆顶，都会被压入大根堆中，完成两个堆中数据的排序维护。

```
class MedianFinder {
public:
  MedianFinder() {
  }

  void addNum(int num) {
    // 添加元素
    if (small.size() >= big.size()) {
      small.push(num);
      int top = small.top();
      small.pop();
      big.push(top);
    } else {
      big.push(num);
      int top = big.top();
      big.pop();
      small.push(top);
    }
  }

  double findMedian() {
    // 如果大根堆元素多，返回大根堆的栈顶
    if (big.size() > small.size()) {
      return big.top() * 1.0;
    } else if (big.size() < small.size()) {
```

```
        // 如果小根堆元素多，返回小根堆的栈顶
        return small.top() * 1.0;
      } else {
        // 如果元素一样多，返回两者栈顶之和的一半
        return (small.top() + big.top()) / 2.0;
      }
    }

  private:
    struct greator {
      bool operator()(const int &a, const int &b) const {
        return a > b;
      }
    };

    std::priority_queue<int, std::vector<int>>          big;    // 默认大根堆
    std::priority_queue<int, std::vector<int>, greator> small;  // 小根堆
  };
```