Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

For example,

```
MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3
```

Solution 1

```
class MovingAverage {
private:
    queue<int> qu;
    int avergeSize;
    double sum;
public:
    MovingAverage(int size):avergeSize(size),sum(0) {}

    double next(int val) {
        sum += val;
        qu.push(val);
        int queueSize = (int)qu.size();
        if(queueSize <= avergeSize){
            return sum / queueSize;
        }
        else{
            sum -= qu.front();
            qu.pop();
            return sum / avergeSize;
        }
    }
};
```

written by sxycwzwzq original link here

Solution 2

```java
public class MovingAverage {

    private int size;
    private List<Integer> queue;
    private int sum;
    public MovingAverage(int size) {
        this.size = size;
        this.queue = new LinkedList<>();
        this.sum = 0;
    }

    public double next(int val) {
        if (queue.size() < size) {
            sum = sum + val;
        } else {
            sum = sum - queue.remove(0) + val;
        }
        queue.add(val);
        return sum / queue.size();
    }
}
```

I used linked list which in interchangeable with queue. We just have to keep tracking of running sum and remove the element when the list is full.

However, if for some reason you don't want to use queue as data structure, we can use circular buffer too.

```java
public class MovingAverage {

    /** Initialize your data structure here. */
    private int sum;
    private int[] circularBuffer;
    private int pointer;
    public MovingAverage(int size) {
        this.pointer = 0;
        this.circularBuffer = new int[size];
        this.sum = 0;
    }

    public double next(int val) {
        int i = pointer++ % circularBuffer.length;
        sum = sum - circularBuffer[i] + val;
        circularBuffer[i] = val;
        return sum / pointer > circularBuffer.length ? circularBuffer.length : po
inter;

    }
}
```

So the pointer kept getting rewrap to starting position the delete it and add the new value. The size is determined by taking the minimum between the pointer and the size.

written by ratchapong.t original link here