Find Median from Data Stream

Core idea: two heaps:

Maintain two priority queues;

- max-heap L (lower half): contains the smaller half of numbers; top is the largest of the lower half.
- min-heap R (upper half): contains the larger half; top is the smallest of the upper half.

Invariants:

1. All elements in $L \le$ all elements in R.
2. The sizes are balanced:
   - either $|L| == |R|$, or
   - $|L| == |R| + 1$ (i.e., allow L to hold one extra when odd count).

With these, the median is:

- if total count odd: top(L)
- if even: (top(L) + top(R)) / 2.0.

addNum(x): how to place and rebalance:

1. Place:
   - If L is empty or $x \le$ top(L), push x into L; else push into R.
2. Replace (fix sizes):
   - If $|L| > |R| + 1$, move top(L) $\to$ R.
   - If $|R| > |L|$, move top(R) $\to$ L.

This guarantees invariants after every insertion.

Complexity:

- addNum: $O(\log n)$ (heap push / top).
- findMedian: $O(1)$.

Follow-ups:

1) All numbers within [0, 100]
   Use counting instead of heaps:
   - Keep freq[101] and track total cnt.
   - find Median:
     - If cnt is odd, scan cumulative frequency until you reach (cnt+1)/2 → that index is the median.
     - If cnt is even, find the two middle positions cnt/2 and cnt/2+1, scan cumulatively to both, average their indices.
   - Time:
     - addNum: $O(1)$
     - findMedian: $O(R)$ with $R = 101$ → effectively $O(1)$

2) 99% within [0, 100], some outliers:
   Two practical options:
   - Hybrid counting + balanced BST / Heap for outliers:
     - Keep freq[0...100] as above.
     - For numbers <0 or >100, store separately (e.g two balanced multisets, one for low outliers, one for high). Track their sizes.
     - While finding median, decide whether it lies among outliers or in the bucketed range by comparing cumulative sizes. If it lies in the bucketed range, scan freq; otherwise walk the appropriate multiset to the needed rank (or maintain order-statistics tree to get $O(\log n)$ select).
   - Buckets with prefix sums:
     - Keep freq and also maintain an incremental prefix count if you do any queries (or recompute on demand since 101 is tiny). Outliers managed by min/max-heaps or balanced trees as above.