12. Find median in a stream

Hard Accuracy: 51.32% Submissions: 6248 Points: 8

Given an input stream of **N** integers. The task is to insert these numbers into a new stream and find the median of the stream formed by each insertion of **X** to the new stream.

Example 1:

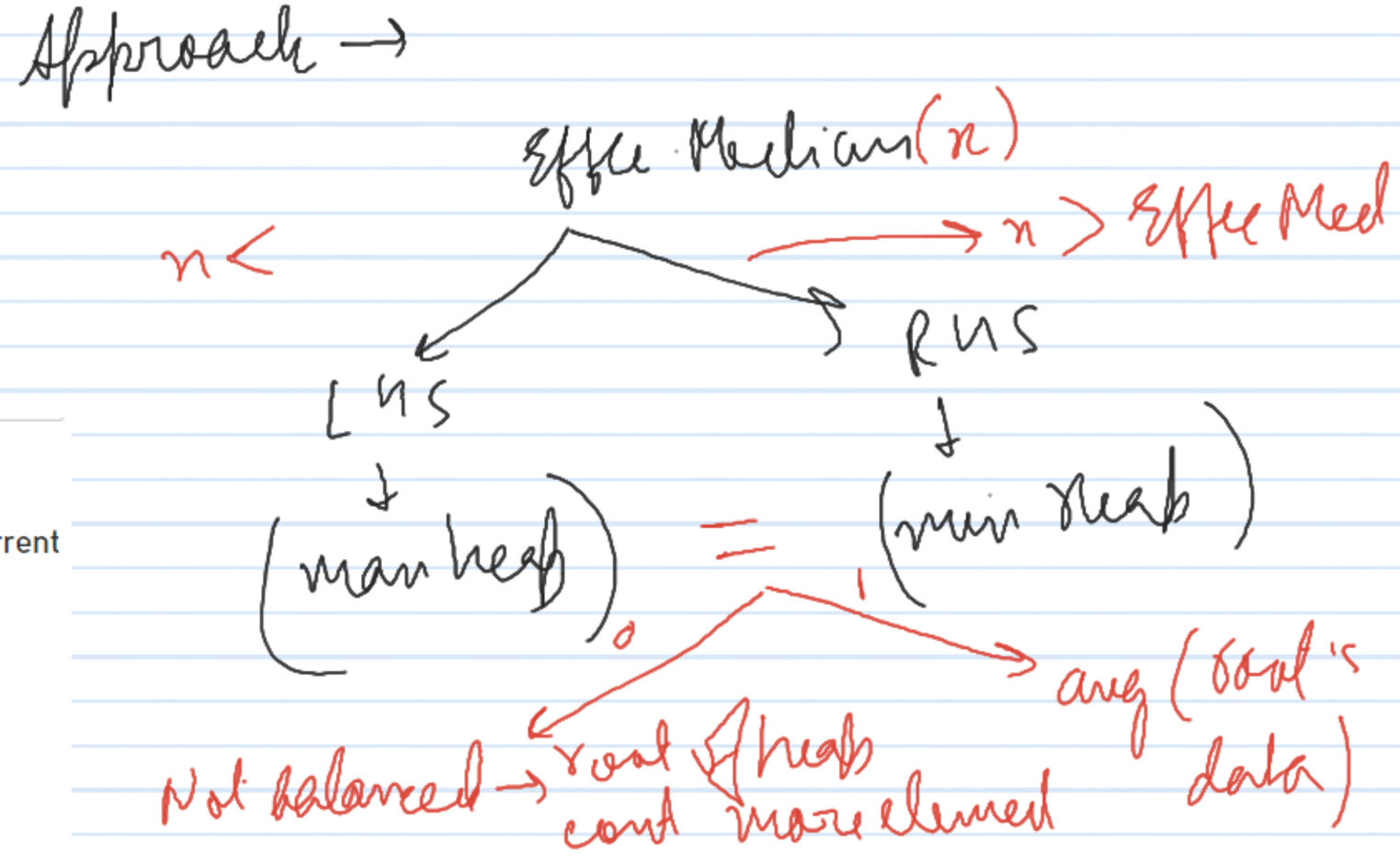
```
Input:
N = 4
X[] = 5,15,1,3
Output:
5
10
5
4
Explanation:Flow in stream : 5, 15, 1, 3
5 goes to stream --> median 5 (5)
15 goes to stream --> median 10 (5,15)
1 goes to stream --> median 5 (5,15,1)
3 goes to stream --> median 4 (5,15,1 3)
```

Your Task:
You are required to complete 3 methods **InsertHeap()** which takes 1 argument, **balanceHeaps()** and **getMedian()** and returns the current median.
Expected Time Complexity : O(nlogn)
Expected Auxilliary Space : O(n)

Constraints:
$1 <= N <= 10^6$
$1 <= x <= 10^6$

We can use a max heap on left side to represent elements that are less than *effective median*, and a min heap on right side to represent elements that are greater than *effective median*.

After processing an incoming element, the number of elements in heaps differ utmost by 1 element. When both heaps contain same number of elements, we pick average of heaps root data as *effective median*. When the heaps are not balanced, we select *effective median* from the root of heap containing more elements.



① Initialization → median, minHeap, maxHeap

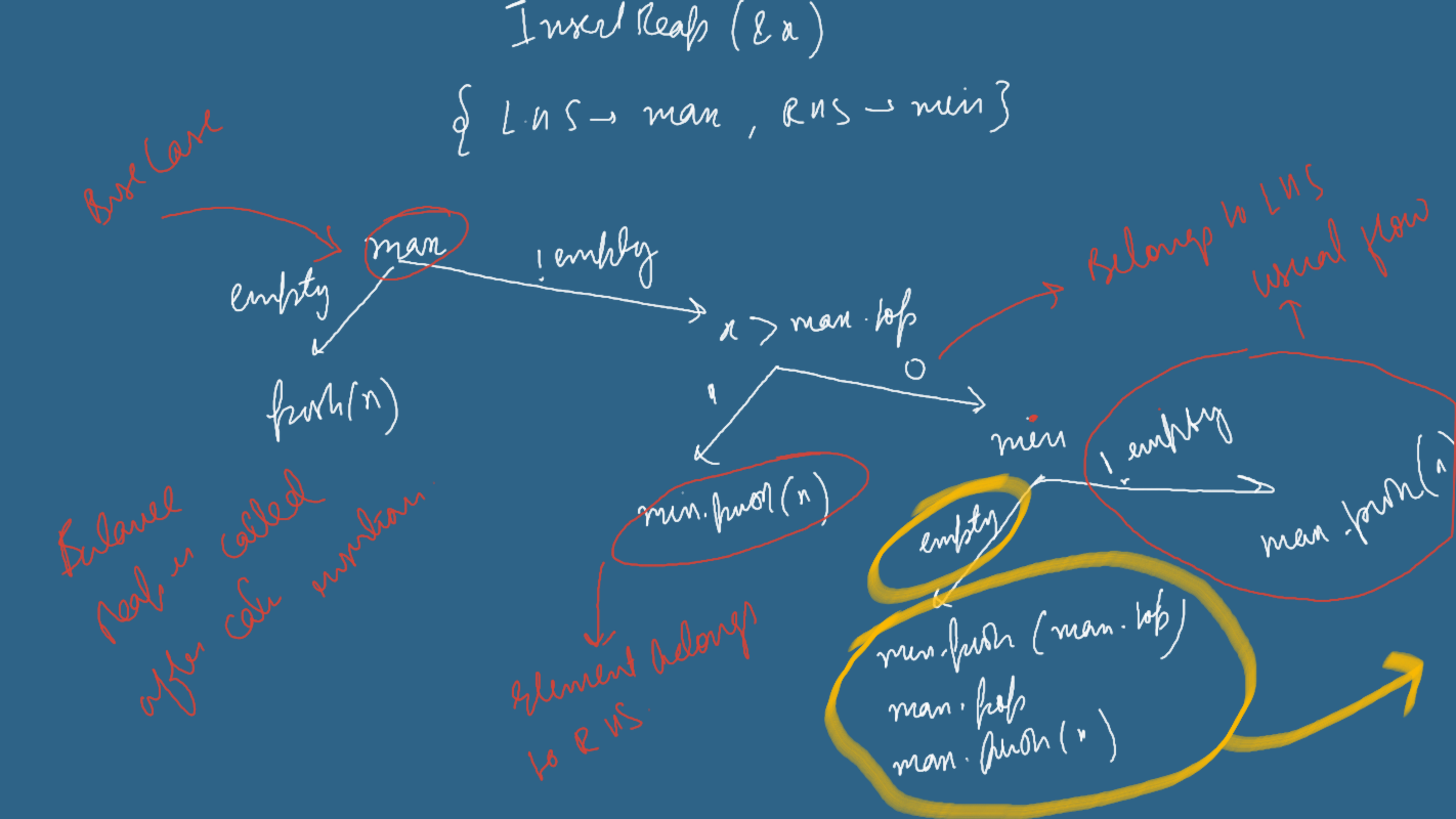
② Get Median

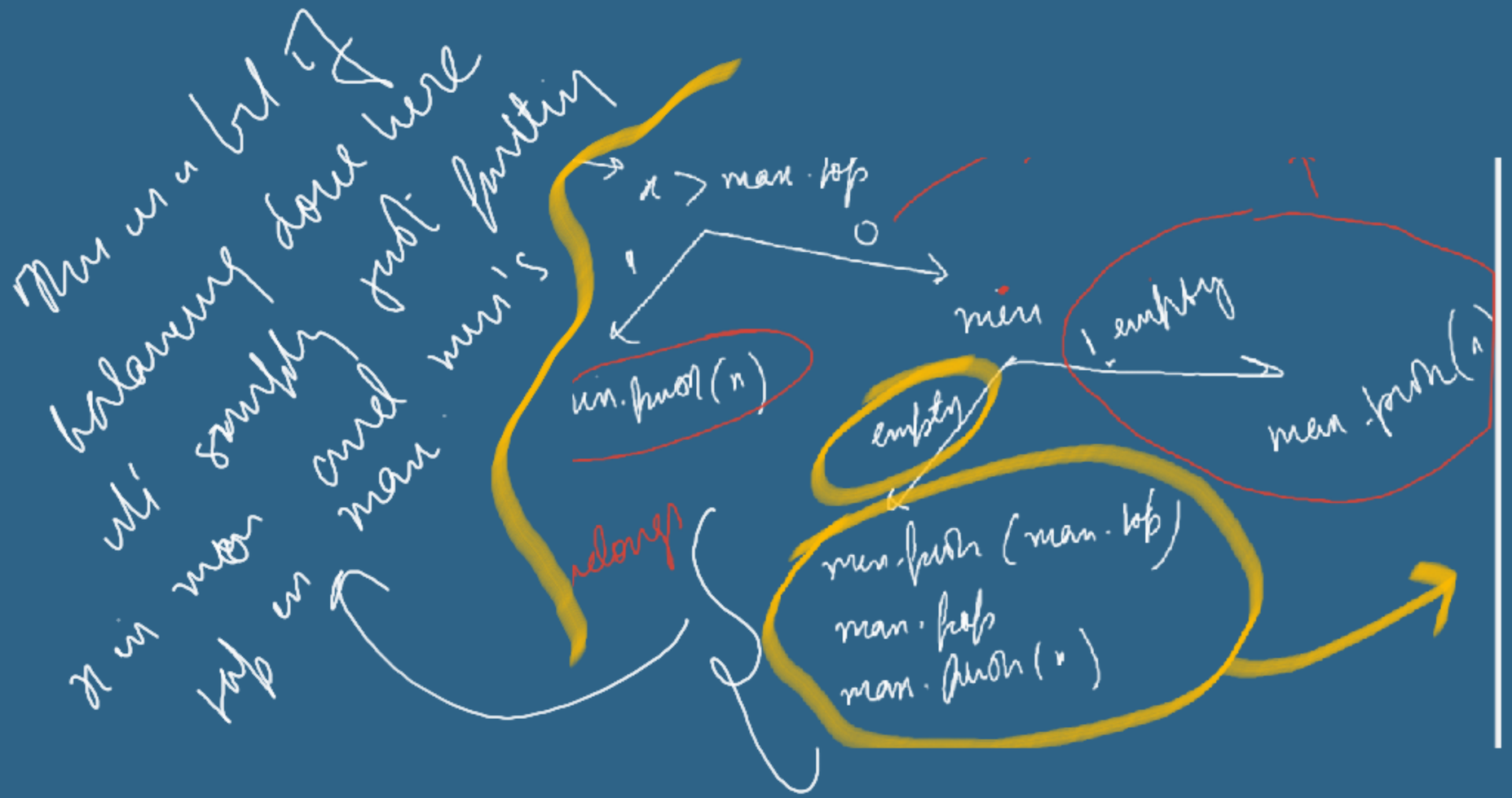
- size(min) = (max)
  - ↓
  - Return (min.top + max.top)/2

Here the size disparity at max could be 1, so (max + min).size % 2

Return '0' or 1

- Both not equal
  - if(min == empty) → only 1 element case
    - ↓
    - Return max.top
  - size(min < max)
    - 1 → return max.top
    - 0 → return min.top


Insert Heap (& a)
{ LHS → max , RHS → min }
Base Case
empty
max
! empty
x > max . top
push(x)
Balance Heap is called after each insertion
x
min . push(x)
Element belongs to RHS
Belongs to LHS
usual flow
min
! empty
max . push(x)
empty
min . push (max . top)
max . pop
max . push(x)


x > max · top
0
1
min
1 empty
min · push(x)
min · pop(x)
empty
redoing
min · push (max · top)
max · pop
max · push(x)

Function

Balance Heap

① Balance check → $\text{size}(\text{max} - \text{min}) > 1$

↓ yes

$\text{max.size}() > \text{min.size}$

yes → ① Put max top in min
② {remove} {max top} {in}

No → min is >

↓

① Put min top in max
② pop from min