(Stock Price Fluctuation)

(Approach 1)
(Hashmap & Heap)

HashMap
Map<Timestamp, Price> → for any update call this map is updated.

Two heaps
1) MaxHeap
2) MinHeap

① mp[timestamp] = price
latestTimestamp = max(latestTimeStamp, current).

② maxheap.offer(new int[]{p,t});

③ minheap.offer(new int[]{p,t});

getMaximum(){ → condition to check if price for that timestamp is updated one.

while(maxheap.peek().price != map.get(maxheap.peek().timestamp()){

maxheap.poll();

}

return maxheap.peek().price;

}

People might wonder that this in worst case can go up O(N-1). Yes that's correct. But atmax N{#of calls} entries goes into the heap, so its (NLogN)

Similarly for minheap

(Editorial Soln)

(Good Approach) ***

Approach 2
(HashMap & TreeMap)

this soln' is amazing soln. Really like the approach.

2 maps to be used.

1) Normal HashMap → map(timestamp, Price) → Same as approach 1.
2) TreeMap → { to fetch easily: .firstKey() → min, .lastKey() → max } easy.
}

TreeMap(Price, Frequency) ⇒ int. ⇒ int.

Main Funda Happens while handling update().

update(timestamp, price){

int oldprice = mp.get(timestamp);

So this map is just to keep frequencies and as a price is getting removed, its frequency is removed and at the end if its 0, the removed from the map.

if(oldprice is there & treeMap.contains[oldprice]){

treeMap.[oldPrice]--;

if(treemap[oldPrice] == 0){

remove oldPrice from treeMap

}

}

mp[timestamp] = price;

treeMap[price]++;

}