{Assuming you have read the question properly} (Max Stack)

[push, pop, top] → these 3 are standard operations of a stack.

(peekMax & popMax) → these are what we need to support

the thought process which my brain had.

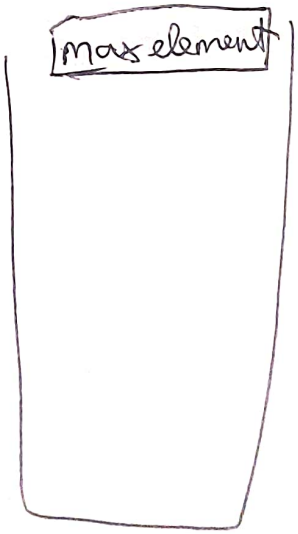
*) we need to implement our own stack which will have doubly linked list, so that we can pinpoint to the node we want to remove and just do doubly Linked List removal of that node

*). peekMax → for this operation my brain thought we need to maintain a heap of nodes, which are being stored sorted in descending order by their values.

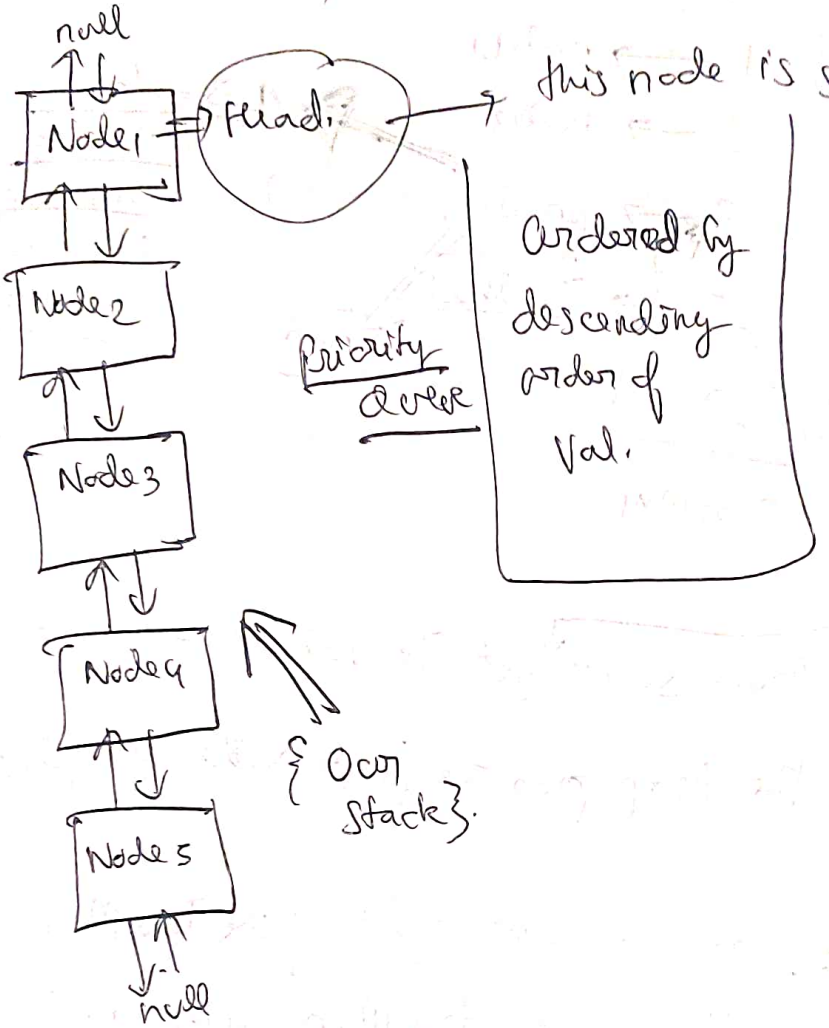
(priorityqueue)

node

whenever peekMax would be called we would simply pop it from the priority queue, and since we are getting the node itself. which is a part of the doubly Linked List Stack. Normal deletion.



Problem arises, what happens when pop() is called.



Now what happens is that to find this node1 in the heap you have to scan the full heap. which makes it an O(N) operation. which is not feasible

So, this approach won't work

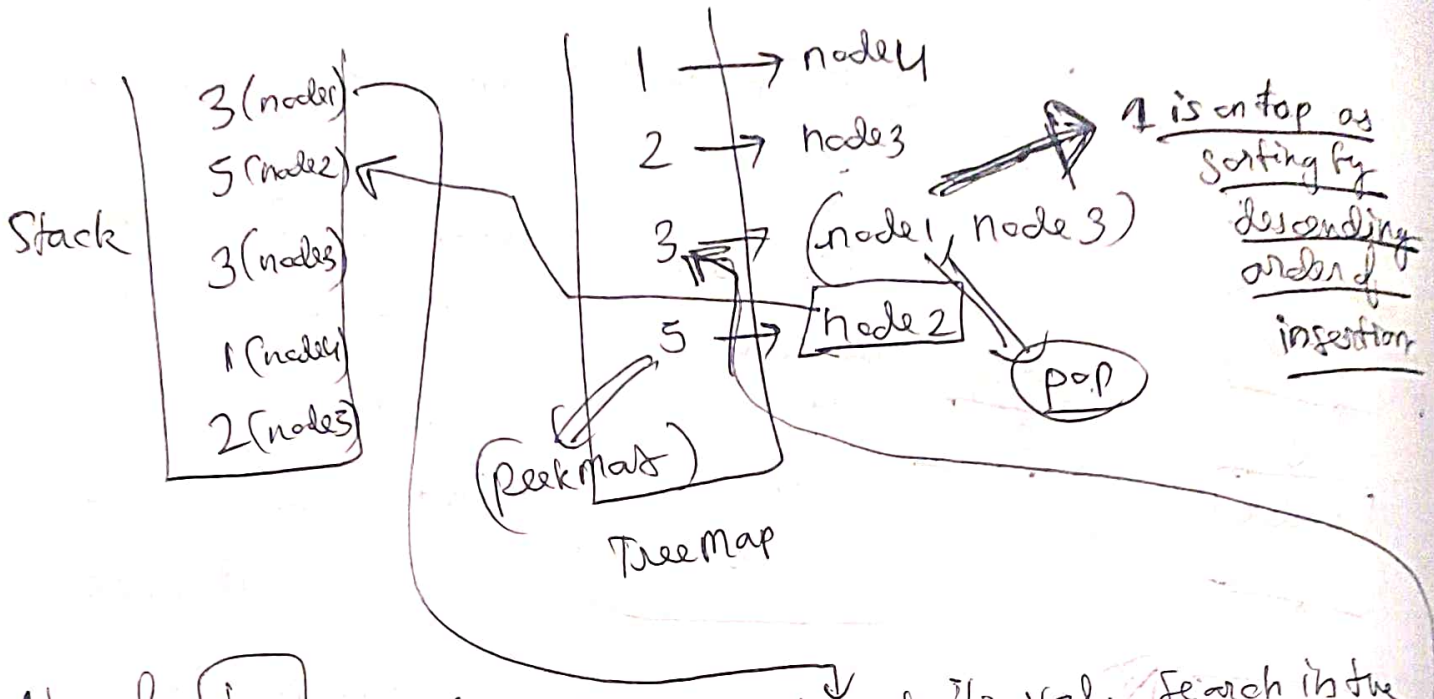
Now my main thought about using a map.

(Value → PriorityQueue Map).

value → Heap → this contains nodes where node.val = key of the treeMap.

And in the heap, we will do sorting based on descending insertion time.

(P.T.O)



Now for pop → just pop from stack get its val, search in the treemap → and from the heap pop the first node, simple.

Similarly for peekMax, get the lastKey() of the TreeMap, get its topnode from the Heap of the treeMap, and in the doubly linked list pop it from the stack.

(Optimisation)

Upon seeing other solns I realised, we don't need to maintain a ~~val~~ (key → PriorityQueue) in the TreeMap, a simple Stack would also be fine. Concern is maintain the latest insertion on top right? So a Stack.top() or an ArrayList's last element would suffice.