The main problem is handling the increment operation.

increment(index, value) → { we know how to work around the operations. We can do Lazy propagation on this array because of one condition.

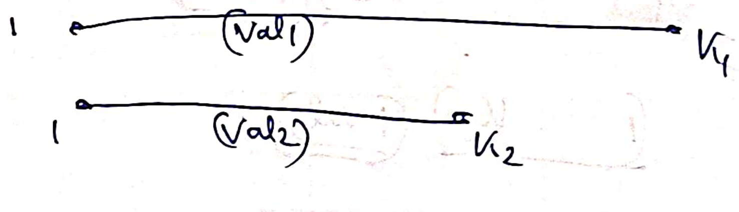
If inc (index, value) if no of elements in the stack is less than index, then only increment the values which are left in the stack.

<u>(My thought process)</u>

Well I was thinking $[1 \longrightarrow k_1] = val_1$

that means indexes from $[1, k_1]$ to be increased by val_1.

again increment $[1, k_2]$ to be increased by val_2



I was thinking abt using segment tree with Range Updates Using lazy propagation.

update (1, K_1 , val_1)

update (1, K_2, val_2)

⇒ then You pop, a value from the stack.

Check which element is getting popped like which number or index. then just get value

sum (index, index) ⇒ get Range sum.

again sum update happens

update (1, K_3, val_3) ⇒ { Segment tree Lazy propagation update. }

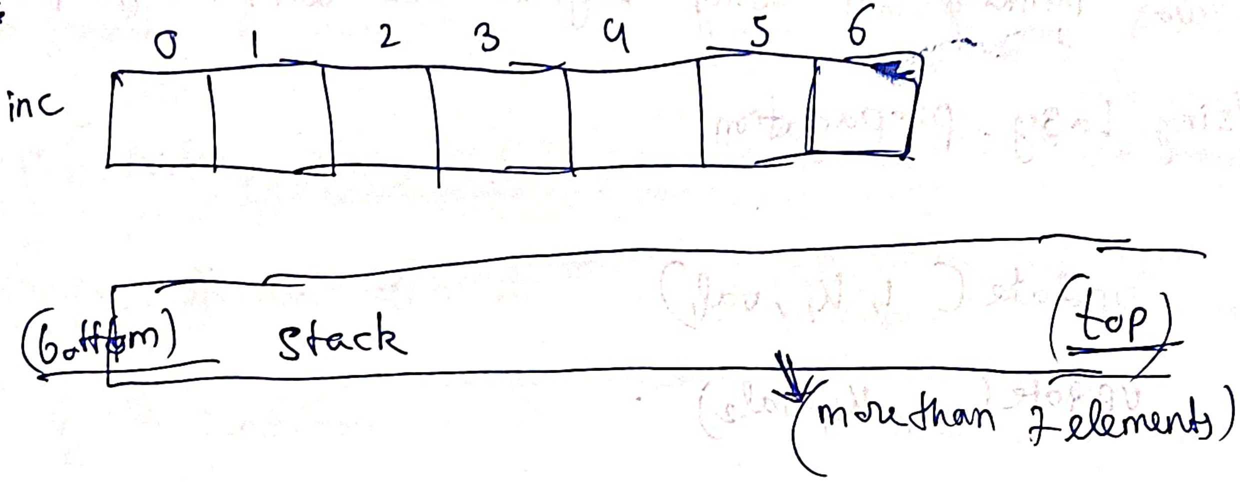
Again some index gets fetched, then value is generated.

Well, this approach will definitely work, but can you think of something simpler?

We don't need ~~very~~ such complex data structure to solve it.

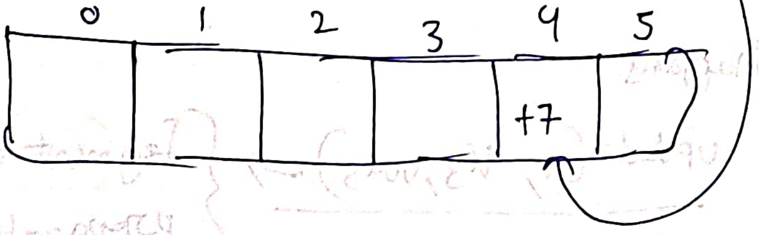
Why don't we have an array which has these index wise updates.

Soln.

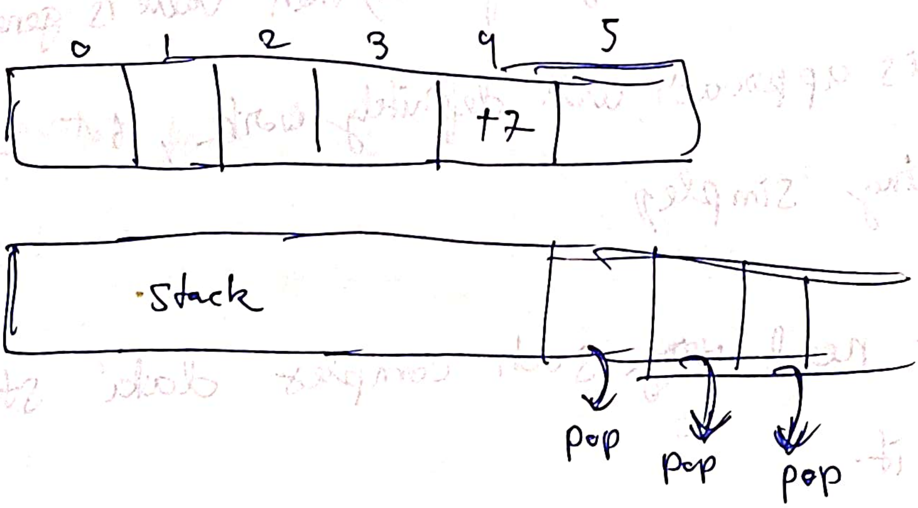


lets say update (5, 7) comes, For bottom 5 increase +7 value. ~~at index 5~~

You could just do (inc [5-1] = 7)

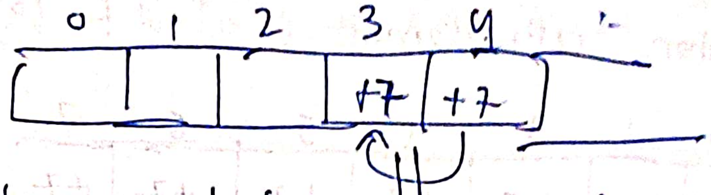


Now lets say popping happens

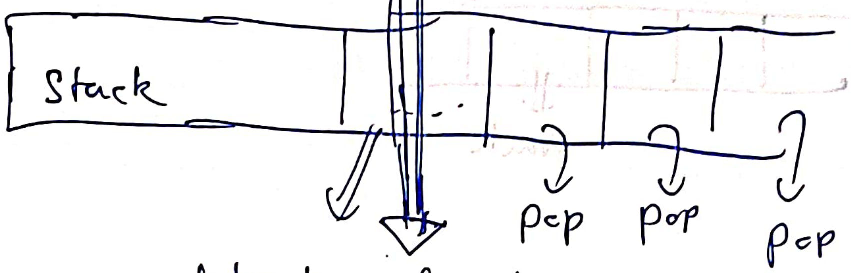


while popping if we see

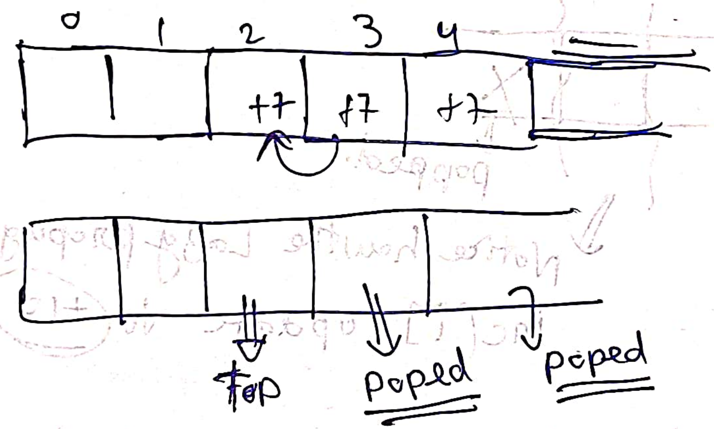
```
if(stack.size() ==5) → 5th element is getting popped
    stack.poll();
    inc[i-1] += inc[i]
```



Propagate +7 to the previous index.

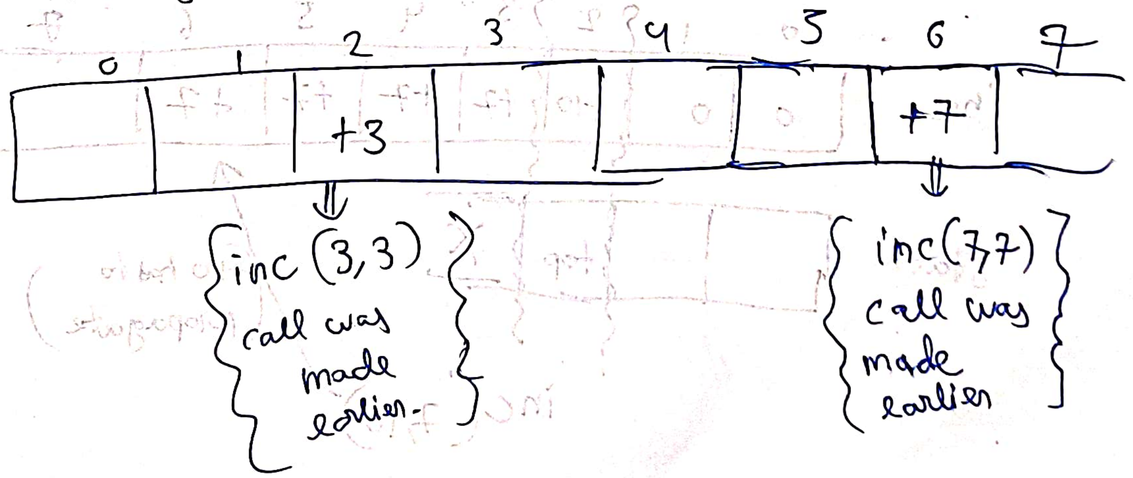


When this element will get Popped, propagate the val (+7) to the previous index.

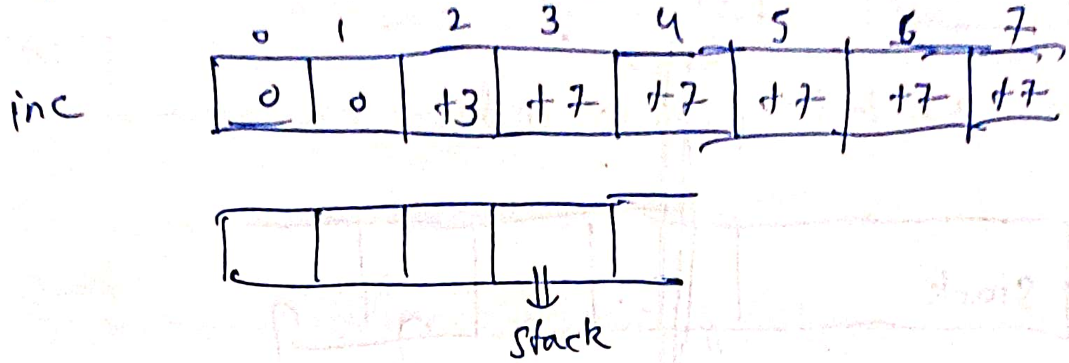


Why is this approach so sexy,

lets say you had somewhere.


0
1
2
3
4
5
6
7
+3
+7
inc (3,3) call was made earlier.
inc(7,7) call was made earlier

Picture of inc array when 4th element is about to be popped.



	0	1	2	3	4	5	6	7
inc	0	0	+3	+7	+7	+7	+7	+7

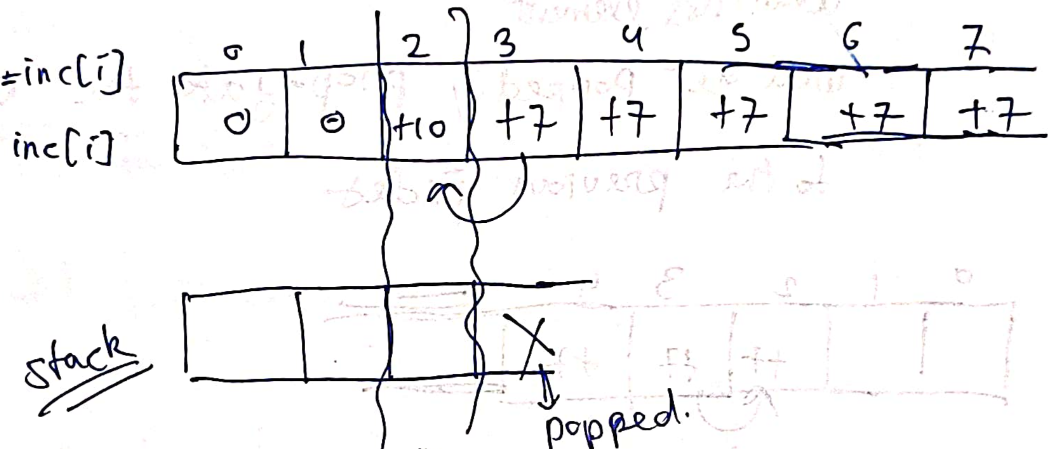
Now pop. st.pop.

↓

inc[i-1] += inc[i]

st.top() + inc[i]

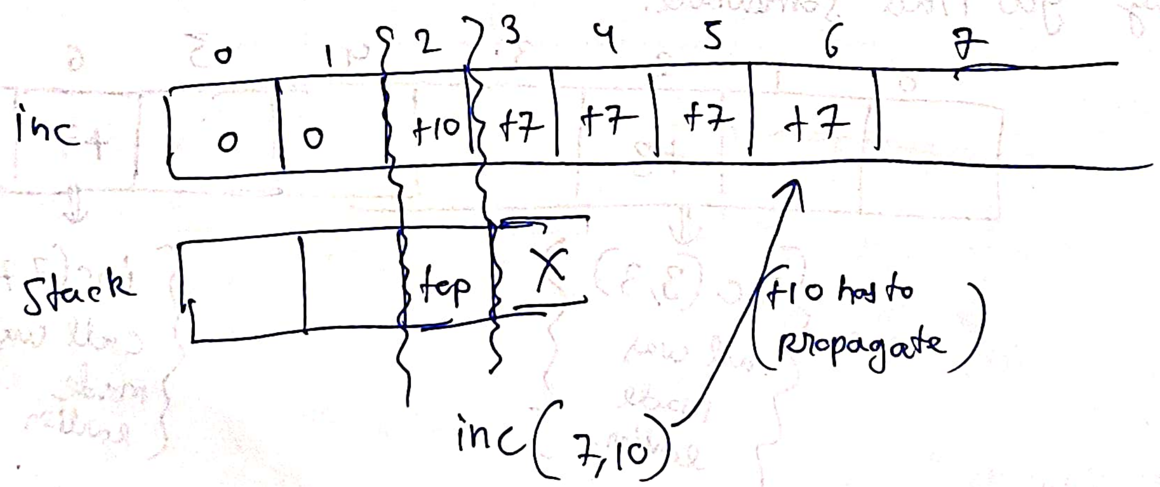
i--;



0	1	2	3	4	5	6	7
0	0	+10	+7	+7	+7	+7	+7

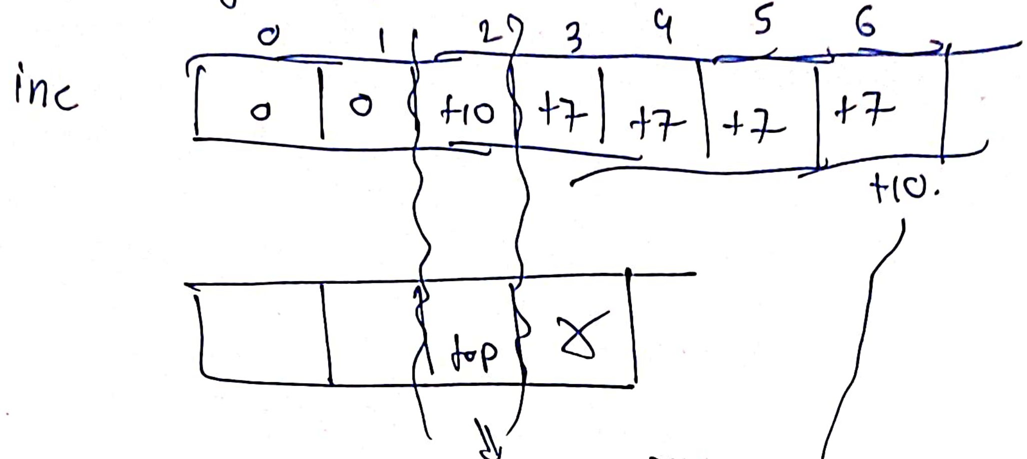
⇓ Notice how the Lazy propagation made inc[i] update to +10.

Well, Well I had doubt in another senario, what if at this current scenario, what if inc(7, 10) call comes.



	0	1	2	3	4	5	6	7
inc	0	0	+10	+7	+7	+7	+7	

My worry was.

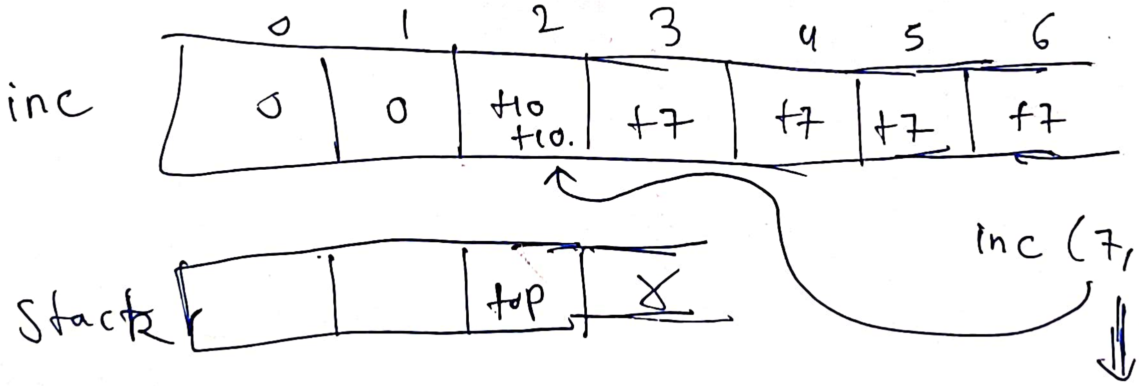


my worry was when I pop this top, how is this +10 going to propopage to this index.

Now I was thinking segment tree will be reqired for sure

* In the question, there was a condition for inc (index, val).
If no of elements in the stack is less than index, then add val to those elements only.

So index = min (~~st.top()~~ st.size(), index) -1;



inc (7,10)
⇓ (becomes)
inc (3,10).

Now when you pop the top, you get top+20 and do inc[i-1] += inc[i]