

## 155. Min Stack

### Problem statement

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with  $O(1)$  time complexity for each function.

#### Example 1:

##### Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]  
[[], [-2], [0], [-3], [], [], [], []]
```

##### Output

```
[null,null,null,null,-3,null,0,-2]
```

##### Explanation

```
MinStack minStack = new MinStack();  
minStack.push(-2);  
minStack.push(0);  
minStack.push(-3);  
minStack.getMin(); // return -3  
minStack.pop();  
minStack.top();    // return 0  
minStack.getMin(); // return -2
```

1.

## Problem Intuition

1. Lets consider a stack and a single variable minEle...which stores the min\_ele

Toy:

Op	Stack	minEle
		2
push(2)	2	2
push(1)	2, 1	1
push(4)	2, 1, 4	1
push(6)	2, 1, 4, 6	1
push(-1)	2, 1, 4, 6, -1	-1
-1 ← pop()	2, 1, 4, 6	?

X

here

first we have pushed 2 and min ele is 2....push(1) then min\_ele is 1..

2. In the end we have push(-1) as it is the min element..we keep min\_ele = 1...and later we have pop() it from the stack..now how can we obtain the min\_ele? So using single variable min\_ele..we cannot retrieve min\_ele
3. So what we have learned from this is we need to preserve the previous min\_ele

Op	Stack	minEle
		2
push(2)	2	2
push(1)	2, 1	1
push(4)	2, 1, 4	1
push(6)	2, 1, 4, 6	1
push(-1)	2, 1, 4, 6, -1	-1
-1 ← pop()	2, 1, 4, 6	?

X

Lesson: preserve

4. So here...we'll change the way we push and pop elements

5. Here let's consider...the element which we are inserting is  $n$ ...so when we push(1)...it is less than curr min\_ele

op	stack	minEle
push(2)	2	2
push(1) ↑ $n$	2, 1	1 = $n$

$1 < 2$       $n < \text{minEle}$

6. Now as the  $n < \text{minEle}$ ...we can write it as  $n - \text{minEle} < 0$  and we add  $n$  on both sides ..then

$$\begin{aligned}
 & n < \text{minEle} \\
 & \rightarrow n - \text{minEle} < 0 \\
 & 2n - \text{minEle} < n
 \end{aligned}$$

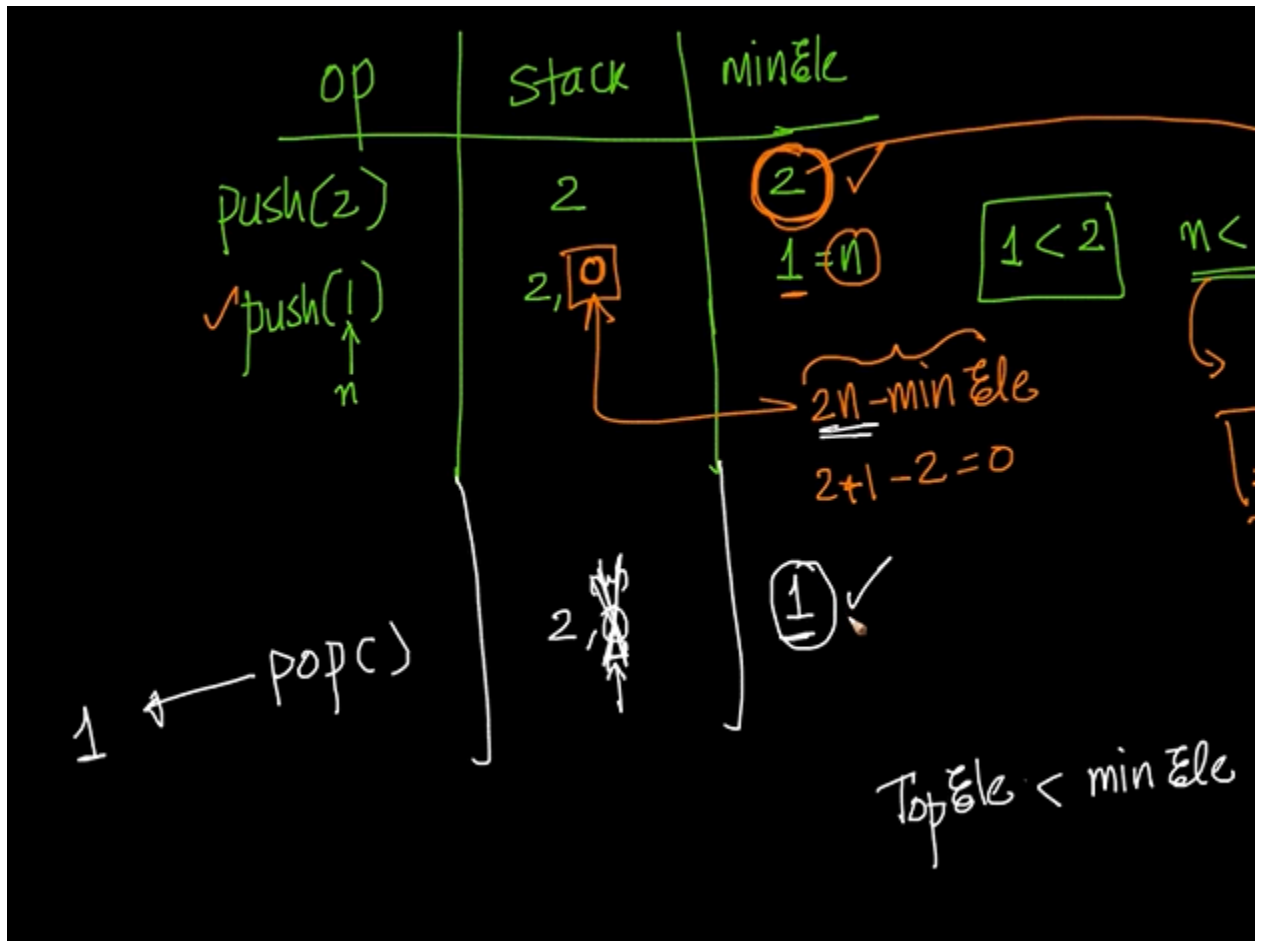
we get

7. So now ..in the place push(1)..we'll push ( $2n - \text{min\_ele}$ )

op	stack	minEle
push(2)	2	2
✓ push(1) ↑ $n$	2, 0	1 = $n$

$2n - \text{minEle}$   
 $2 + 1 - 2 = 0$

8. Now let's do pop operation...so here first we'll return the min\_element and if top element  $\leq$  min\_ele ...then we'll update the min\_ele

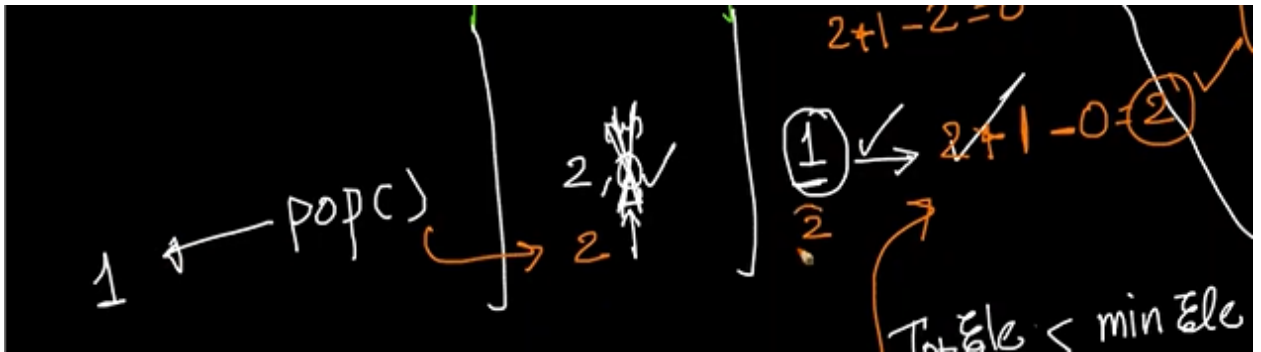


9. Now how our top\_ele came into existence...it because of the function

$$\text{TopEle} < \text{minEle}$$

$$\text{TopEle} = 2 * \text{CurrMinEle} - \text{PrevMinEle}$$

10. Now from here..we can get the Prev\_min\_ele



11. Refer this example...of how we used Push and pop operations

Op	stack	minEle
push(2)	2	2
push(1)	2, 0	1
push(4)	2, 0, 4	1
push(6)	2, 0, 4, 6	1
push(-1)	2, 0, 4, 6, -3	-1
push(7)	2, 0, 4, 6, -3, 7	-1
pop()	2, 0, 4, 6, -3	-1
pop()	2, 0, 4, 6	1

Handwritten notes and calculations:

- $2+1-2=0$  (2+n-minEle)
- $(2+1)-1=-3$
- $2+\text{minEle}-n$
- $(2+1)-(-3)=1$