

# Min Stack

🕒 Created	@September 23, 2022 8:03 PM
▼ Class	
▼ Type	
🔗 Materials	
☑ Reviewed	<input type="checkbox"/>

## Problem:

We need to implement a Stack, which can support the following functions:

- `push(x)` : to insert an element at the top of the stack
- `pop()` : to remove an element from the top of the stack
- `getMin()` : to return the minimum most element present in the stack
- `top()` : to return the element present at the top

Note: Stack as a data structure supports LIFO (last in first out).

## Solutions:

We need a mechanism to support `getMin` because this is not a traditional stack function. We will also try to make sure that all the above 4 operations are working in `O(1)` complexity.

### Approach 1:

Let's say that to implement push, pop and top, we can use a simple array, we can insert the element at the end of the array to implement `push`, remove the element from the end of the array to support `pop` as LIFO and return the last index element for `top`.

Now how can we implement `getMin` with the above approach.

To implement `getMin` we can try to read all the elements of the stack by one by one pushing them in a new temporary stack, and then get the minimum. (As we cannot access anything apart from the top of the stack, we need to put all the elements in a new temp stack, and while putting them we will remove them from original stack and then only we can read them).

The time complexity of this approach for `getMin` will be  $O(n)$  and space will be  $O(n)$

This is kind of like the brute force approach.

Can we improve ???

## Approach 2:

We already have push, pop and top in  $O(1)$  time so let's try to optimise `getMin`.

Now if we don't want to read the whole stack again and again whenever `getMin` is called, then we have to somehow maintain the current `min` element while inserting it in the stack. How can we do it ? May be we can maintain a variable `currentMinimum` that will be compared with every new inserted element, and if the new element is less than previous value of `currentMinimum` then it will get updated.

But, what if we start removing elements from the stack ? When we remove some elements how do we know after the removals, did we already removed the previous minimum or not ? And what should be the new minimum.

operation	Stack	currentMinimum
push(6)	[6]	6
push(7)	[6, 7]	6
push(4)	[6,7,4]	4
push(8)	[6,7,4,8]	4
push(-1)	[6,7,4, 8, -1]	-1
pop()	[6,7,4,8]	Now we cant say for sure that 8 will be the new minimum after removal of -1

So after doing removals it becomes difficult to get back the previous value of `currentMinimum`.

So how can rectify this issue ? Now we need to somehow keep the track of previous minimum values, and whenever we have to keep track previous set of values, Stacks

themselves are one of the best choices.

So, we can maintain a parallel stack (say temp) which can store the `currentMinimum` value at the top of it. Whenever we insert a new element in the original stack, we will compare this new element with the top of the temp stack. If the new element is less than the top of the temp stack, then we insert this new element in the temp stack, otherwise we push the previous top value of temp again.

When we pop an element from the original stack, we pop an element from temp stack also, so that we can get access to the previous minimum. Every time top of temp will give minimum till now.

operation	original stack	temp	currentMinimum (top of temp)
push(5)	[5]	[5]	5
push(3)	[5,3]	[5,3]	3
push(8)	[5,3,8]	[5,3,3]	3
push(1)	[5,3,8,1]	[5,3,3,1]	1
pop()	[5,3,8]	[5,3,3]	3
pop()	[5,3]	[5,3]	3
getMin	[5,3]	[5,3]	3

So by this approach we are technically maintaining the current minimum upto now, inside another stack, this helps us to get the minimum directly from the top of stack so `getMin` works on  $O(1)$  time complexity, but because we are taking extra space the space complexity will be  $O(n)$ .

But here also we are maintain  $O(n)$  extra space. Can we optimise ?

one small optimisation we can do is, instead of inserting previous top again when the new element is not less than previous top, we can just avoid to insert anything to the temp stack. But still in worst case when every element is unique and a new element is less than previous element, we will get space complexity as  $O(n)$

Can we optimise even further ?

## Approach 3

Now we don't want to maintain another extra data structure for maintaining minimum till now. So how can do it ? When we were maintaining a single variable, the only problem

was that we were not able to detect when to change the minimum and with what value to change the minimum when we pop the elements.

So somehow, we need to keep a tracker or a flag kind of a thing, that can tell us during pop when to update this `currentMinimum` variable and with what value we should update it.

So let's say, `x` is the new element to be inserted and `currentMinimum` is the minimum element of the whole stack before we add `x`

So, after insertion of `x` we are going to update `currentMinimum` to `x` if and only if

```
x < currentMinimum
```

```
→ x < currentMinimum
```

```
→ x - currentMinimum < 0
```

So, how about when `x < currentMinimum`

this condition holds true, then instead of inserting `x` we insert `x - currentMinimum` (technically we are inserting `x - previousMinimum`) and after it update `currentMinimum` to be `x`.

In this approach we are trying that instead of inserting `x`, we insert something even less than `x`, which is `x - currentMin`.

operation	Stack	<code>x - currentMinimum</code>	<code>currentMinimum</code>
push(7)	[7]	No need for this because it's the first element	7
push(8)	[7,8]	As 8 is not less than 7, do nothing, and insert 8	7
push(5)	[7,8,-2]	$5 < 7 ? \rightarrow \text{true}$ , so instead of inserting 5, we insert $5 - 7 = -2$ , and <code>currentMinimum</code> becomes 5	5
push(3)	[7,8,-2,-2]	$3 < 5 \rightarrow \text{true}$ , so instead of inserting 3, we insert $3 - 5 = -2$ , and current Minimum becomes 3	3
push(10)	[7,8,-2,-2, 10]	As 10 is not less than 3, we just push 10	3
pop()	[7,8,-2,-2]	As top of the stack was greater than <code>currentMinimum</code> we don't need to	3

		update it	
pop()	[7,8,-2]	As top of the stack (viz -2) is even less than. the currentMinimum, this is an indicator that when we remove now, we need to update currentMinimum, and $\text{currentMinimum} = \text{currentMinimum} - \text{st.top}()$	5
pop()	[7,8]	As top of the stack (viz -2) is even less than the currentMinimum, this is an indicator that when we remove now, we need to update currentMinimum , and $\text{currentMinimum} = \text{currentMinimum} - \text{st.top}()$	7
push(3)	[7,8,-4]	$3 < 7 \rightarrow \text{true}$ , so instead of inserting 3 we insert $3 - 7 = -4$ and currentMinimum becomes 3	3
push(-2)	[7,8,-4,-5]	$-2 < 3 \rightarrow \text{true}$ , so instead of inserting -2, we insert $(-2 - 3) = -5$ , and currentMinimum becomes -2	-2
push(11)	[7,8,-4,-5,11]	As 11 is not less than -2, we insert 11, and currentMinimum stays as it is	-2
push(-5)	[7,8,-4,-5,11,-3]	$-5 < -2 \rightarrow \text{true}$ , so instead of inserting -5, we insert $(-5 - (-2)) = -3$	-5

Now this is a problem, for negative numbers, the above statement might not work as expected !

Is there any solution ? We want a method that can handle both negatives and positives.

Now let's say when  $x > \text{currentMinimum}$  , the value of  $x - \text{currentMinimum}$  will be always positive.

And when  $x < \text{currentMinimum}$ , the value of  $x - \text{currentMinimum}$  will be always negative.

So now, instead of inserting  $x - \text{currentMinimum}$  only when  $x < \text{currentMinimum}$  how about whenever we get a new element  $x$  doesn't matter if  $x < \text{currentMinimum}$  or  $x > \text{currentMinimum}$  for both cases we only insert  $x - \text{currentMinimum}$  . Can we do something ?

This means if I am having a positive value at top of stack, then it would not have affected our currentMinimum so it's removal also doesn't bother us. but if the value is

negative then it's removal will bother the currentMinimum.

operation	x-currentMinimum	Stack	currentMinimum
push(7)	- for first element do nothing	[7]	7
push(8)	$8 > 7 \rightarrow 8 - 7 = 1$ which is positive	[7,1]	7
push(5)	$5 < 7 \rightarrow 5 - 7 = -2$ which is negative	[7,1,-2]	5
push(3)	$3 < 5 \rightarrow 3 - 5 = -2$ , which is negative	[7,1,-2,-2]	3
push(10)	$10 > 3 \rightarrow 10 - 3 = 7$ , which is positive	[7,1,-2,-2,7]	3
pop()	Now here stack top is positive i.e. 7, that means whatever value 7 is representing, it must not have updated currentMinimum, so we can just remove it	[7,1,-2,-2]	3
pop()	now here stack top is negative, i.e. -2, that means whatever value -2 was representing, it must have updated currentMinimum also, so we will update $\text{currentMinimum} = \text{currentMinimum} - \text{st.top}()$ and then pop the element	[7,1,-2]	5
pop()	now here stack top is negative, i.e. -2, that means whatever value -2 was representing, it must have updated currentMinimum also, so we will update $\text{currentMinimum} = \text{currentMinimum} - \text{st.top}()$ and then pop the element	[7,1]	7
push(3)	$3 < 7 \rightarrow 3 - 7 = -4$ which is negative	[7,1,-4]	3
push(-2)	$-2 < 3 \rightarrow -2 - 3 = -5$ which is negative	[7,1,-4,-5]	-2
push(11)	$11 > -2 \rightarrow 11 - (-2) = 13$ , which is positive	[7,1,-4,-5,13]	-2
push(-5)	$-5 < -2 \rightarrow -5 - (-2) = -3$ , which is negative	[7,1,-4,-5,13,-3]	-5
push(10)	$10 > -5, 10 - (-5) = 15$ , which is positive	[7,1,-4,-5,13,-3,15]	-5
pop()	top element is positive so just remove it	[7,1,-4,-5,13,-3]	-5

pop()	top element is negative, so update currentMinimum and then remove	[7,1,-4,-5,13]	-2
top()	return currentMinimum + <u>st.top</u> → -2 + 13 → 11	[7,1,-4,-5,13]	-2
pop()	top element is positive so just remove it	[7,1,-4,-5]	-2
top()	top is negative, that means this must have affected currentminimum, that means top of the stack is representing minimum element, i.e. currentMinimum, so we return currentMinimum i.e. -2	[7,1,-4,-5]	-2
pop()	top is negative , that means update currentMinimum and then pop it	[7,1,-4]	3
top()	as top is negative, return currentMinimum = 3	[7,1,-4]	3
pop()	top is negative so update currentMinimum and then pop	[7,1]	3 - (-4) = 7
top()	stack top is positive so we return currentminimum + st.top() = 8	[7,1]	7
pop()	top is positive so just remove it	[7]	7
pop()	we only have single element so it is the same element, just remove it	[]	-

Why this worked because, `x - currentMinimum` is this value is positive we don't update minimum other wise we do, so it works like a marker for us.

when we have to get the top and `x - currentMinimum` is negative, then the currentMinimum is the element which is represented by `x - currentMinimum` so we just return currentMinimum otherwise is it is positive then we return `x + currentMinimum` as it must not have affected after minimum value and just got inserted as `x - currentMinimum`

```
var MinStack = function() {
    this.stack = [];
    this.currentMin = Infinity;
};

/**
 * @param {number} val
```

```

    * @return {void}
    */
    MinStack.prototype.push = function(val) {
        if(this.stack.length == 0) {
            this.stack.push(val);
            this.currentMin = val;
        } else {
            if(val < this.currentMin) {
                this.stack.push(val - this.currentMin);
                this.currentMin = val;
            } else {
                this.stack.push(val - this.currentMin);
            }
        }
    };

    /**
     * @return {void}
     */
    MinStack.prototype.pop = function() {
        if(this.stack.length > 0) {
            if(this.stack[this.stack.length - 1] >= 0) this.stack.pop();
            else {
                this.currentMin = this.currentMin - this.stack[this.stack.length - 1];
                this.stack.pop();
            }
        }
    };

    /**
     * @return {number}
     */
    MinStack.prototype.top = function() {
        if(this.stack.length == 1) {
            return this.stack[0];
        }
        if(this.stack[this.stack.length - 1] >= 0) {
            return this.currentMin + this.stack[this.stack.length - 1];
        } else {
            return this.currentMin;
        }
    };

    /**
     * @return {number}
     */
    MinStack.prototype.getMin = function() {
        if(this.stack.length == 1) return this.stack[0];
        return this.currentMin;
    };

    /**
     * Your MinStack object will be instantiated and called as such:
     * var obj = new MinStack()


```



```
* obj.push(val)
* obj.pop()
* var param_3 = obj.top()
* var param_4 = obj.getMin()
*/
```

### Min Stack - LeetCode

Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

 <https://leetcode.com/problems/min-stack/>

