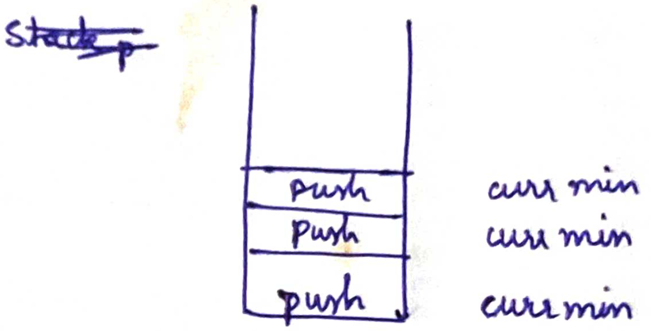
Min Stack → ~~Create~~ Design a stack that returns min element using getMin()

Brute force

1) We can see the problem as follows.

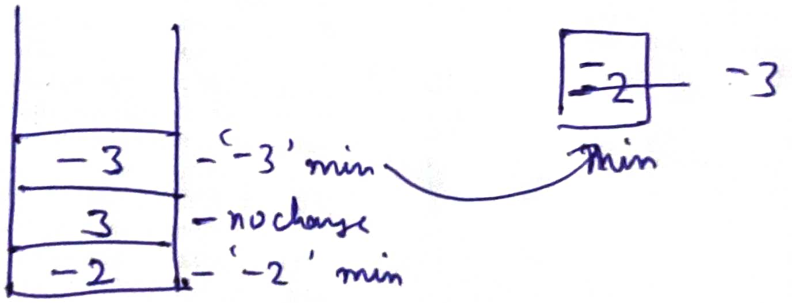


• while pushing the element itself we need to store the current minimum element somewhere so that we can return the minimum element each time.

A test case for why we have to know the current minimum as well as previous one can be as follows.

[-2, 3, -3]

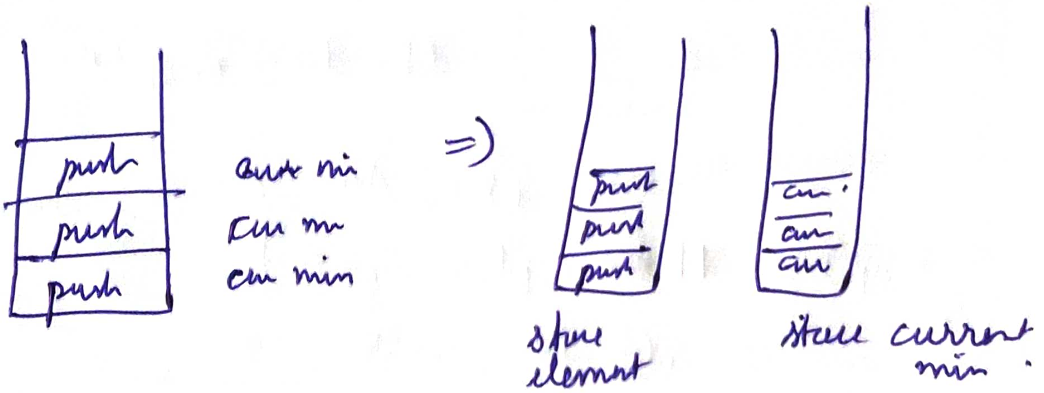
Dry Run (if we took just a variable to store the current minimum)



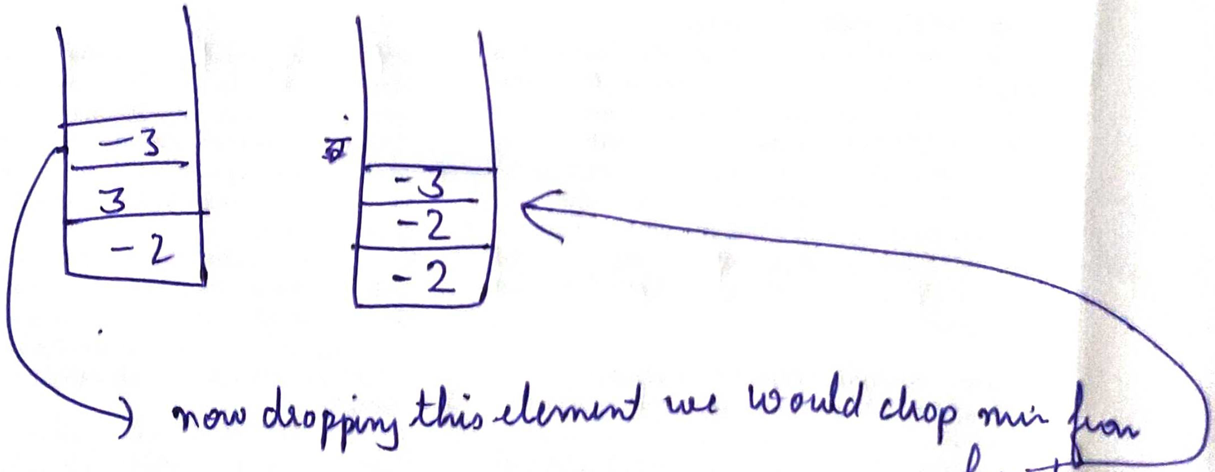
But when we pop the element '-3' then how do we know what is the current minimum know as we have not tracked it anywhere.

Thus we need a way to store the prev min somewhere at each push.

The best way to do it in brute force is to take extra space in form of vector or stack I will be using stack.



Running earlier example here [−2, 3, −3]



now dropping this element we would chop min from here too

∴ min would be easily returned.

Brute force using 2 stacks

```
class MinStack {
                                  will store minimum element at top.
    private: stack s1, s2;
    MinStack () { } ; //no initialisation.

    void push (int val)
    {
            s1. push (val);
            if (s2.empty() || s2.top() >= val)
            {     s2. push(val);
            }.
    }
    void pop ()
    {
            ~~s2.pop();~~
            if (! s2. empty () || s2. top ( ) == s1.top())
            {  s2.pop();
            }  s1. pop(),
    }
    int getMin ()
    {
            return st2.top( );
    }
    void top ()
    { }     return st1.top( );
```

T: O(1)

S: O(2N)

Optimised approach

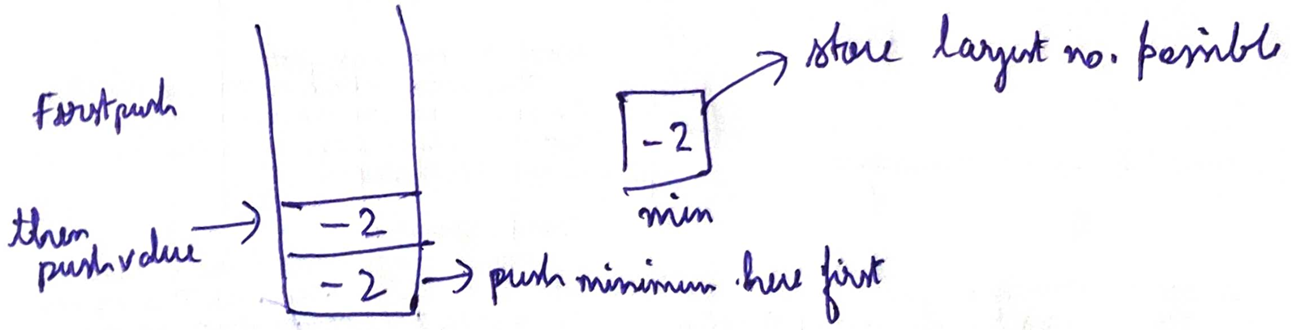
- In optimised approach we can try to remove extra space and utilising a vector for implementing the stack as well (you can use stack as well)

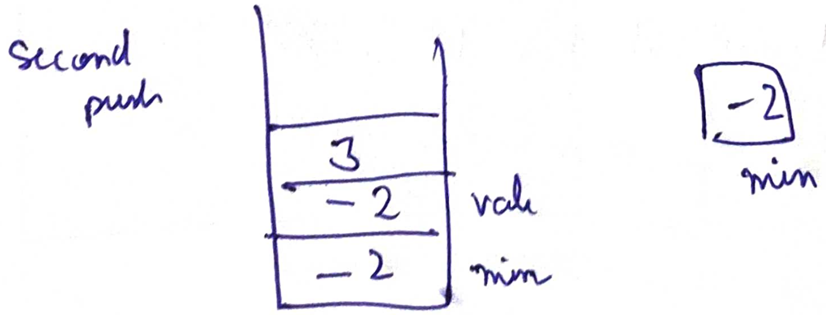
- The earlier approach relied on the principle that we need a current ~~element~~ state of min element somewhere at each push.

- Instead of doing that we can store the state change whenever it happens in stack instead. (How??)

Taking test case

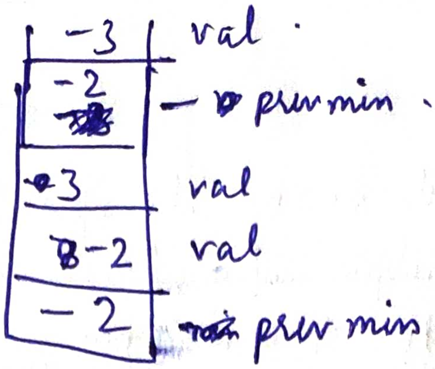
[-2, ~~2~~3, -3]





As this involves no change in state of min as min ~~8~~ < val we won't push ~~min~~ minimum on stack.

Third push



As this invalued state change as the val = '-3' was < min ⇒ min = val

and push min before pushing the element on stack.

What happens in this approach is that it takes the prev min value as we compare the min and current top of the stack, ~~and~~.

we can make amends whenever the min == st.top()
while popping the element.

Thus to summarise

two operations while pushing.

1). check the current val with min
if less replace and push

2.) push value

two operations while popping

1.) check the current min with stack top
if min == st.top()

⇒ remove stack top.

⇒ min = st.top() ← prev. minimum

⇒ remove the prev min.

Optimised Code

T: O(1)
S: O(N)

```
class MinStack {

 private :
      vector <int> stack;
      int min ;

 public :

      MinStack() { min = INT_MAX; }

      void push (int val) {
           if ( stack.empty() || min >= val)
           { // Push current min
                 stack.push_back (min);
                 min = val;
           }
            stack.push-back (val); // push val }.

 }

     void pop (){
        if (min == stack.back ()){
               stack.pop_back (); // remove original value.
               min = stack.back(); // change min to prev min
               stack.pop-back(); // remove prev min
        }
       else { stack.pop(); remove the ~~element~~ value.
        }
  }.
```

```
int top () { return stack . back () ;
           }

int getMin () { return min; }.
```