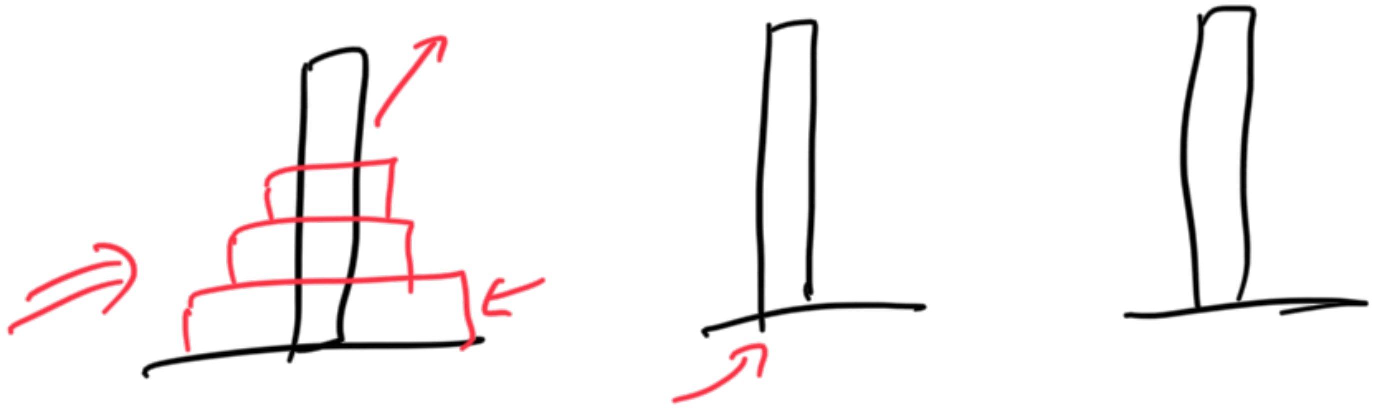
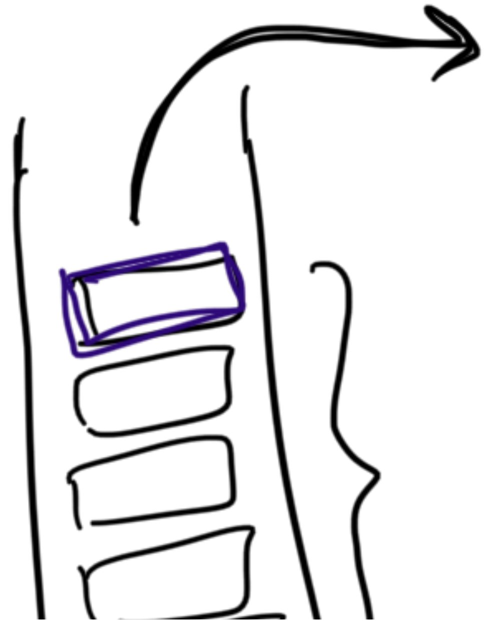
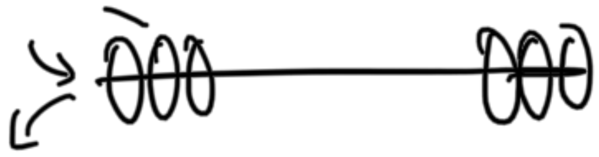
Stacks and Queues

Stack

Last In
First Out (LIFO)

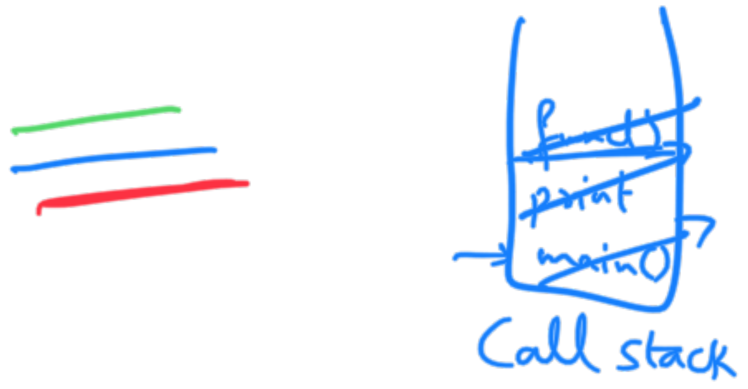


stack.

Abstract Data type

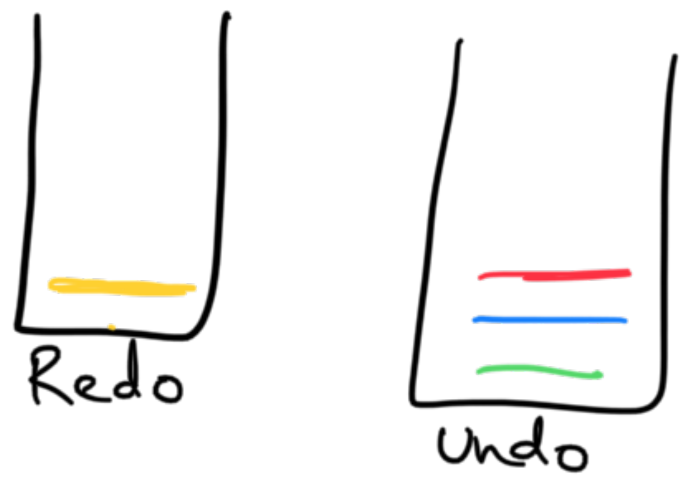
Stack
- push (insert)
- pop (pull/remove)
- top (peek)
- isEmpty()
- size

Applications
- Undo/Redo
- Recursion/Call stack
- Browser back button



main()
print(func())

fun(a)
fun(a-1)

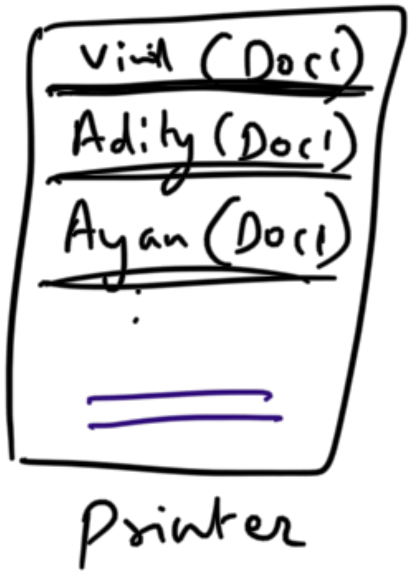


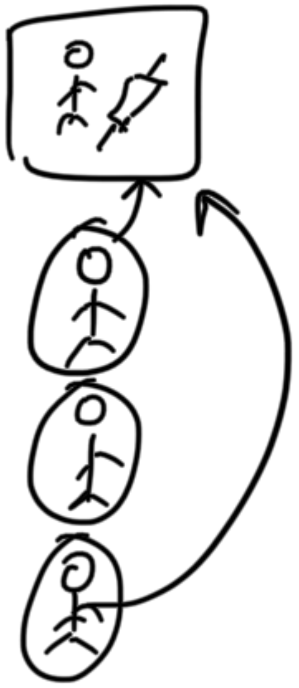
first out

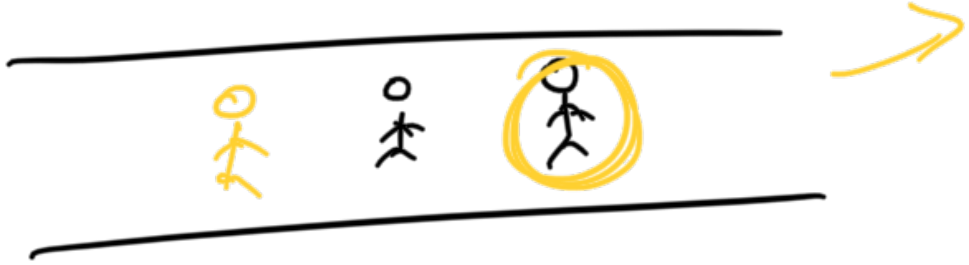
First [illegible] (FIFO)

Queue

- enqueue (push / add)
- dequeue (remove/pull)
- isEmpty()
- size()
- front()







Applications!

- Kafka/RabbitMQ/ActiveMQ (HLD)
- Scheduling
  - processor
  - [illegible]

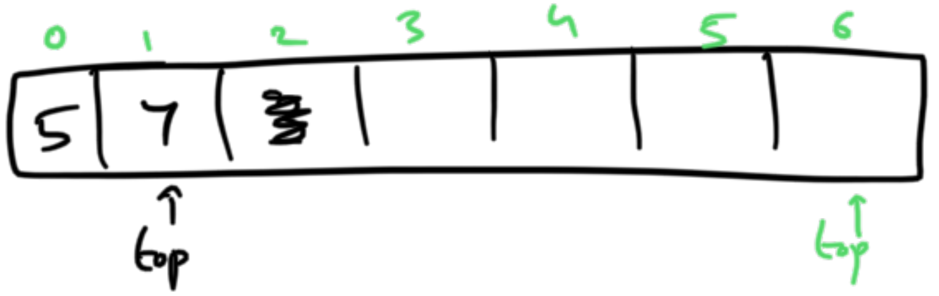
→ printer
→ download Queue.

Implementation.

① Stack → Arrays / dynamic arrays. (Fixed size) (H.W.)
→ Linked List

- push()
- pop()
- size()
- top()
- isEmpty()

```
int size;
int arr[size];
int top = -1; //Always points to top most element
```



```
void push(int data){
 if(top != size-1)
 { top++;
   arr[top]= data
```

T.C = O(1)

push(5)
push(7)
push(3)
pop()
get top()

```
} else error!
```

push(1)
size() -

```
void pop(){
  if (top != -1)
  {
     top--;
  }
}
```

} T.C. = O(1)

```
bool isEmpty()
{
   return (top == -1);

}
```

} T.C = O(1)

S.C => O(N)

```
int getsize()
{
  return top+1;
}
```

} T.C = O(1)

C++
⇓
STL

Java

Stack<Int> st
= new stack<int>

python/JS/Ruby

A = [ ]

A.append( )

A.pop( )

Stack<int> st;

✓ st.push(5)

✓ st.pop()

st.push(10)

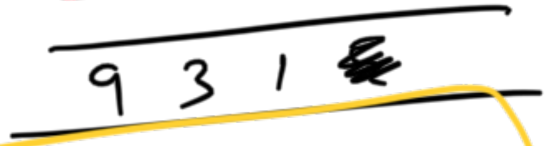
st.pop();

A.pop()

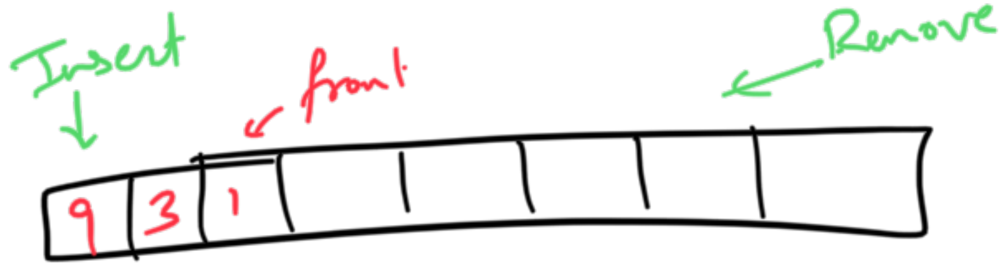
Queue using Array

Insert → O(N) → O(1)

Removal → O(1) → O(N)



H.W.: Implement Queue using arrays



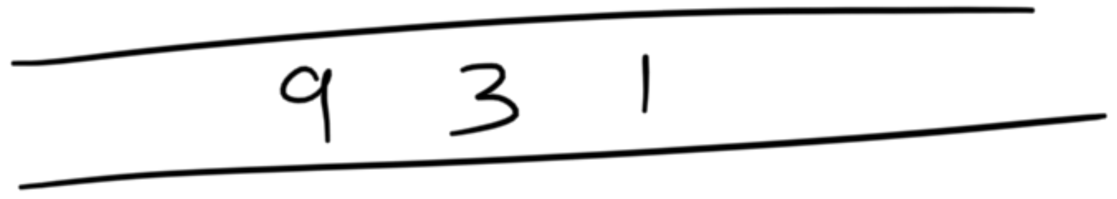
enq(5)

enq(1)

en(3)

deq()

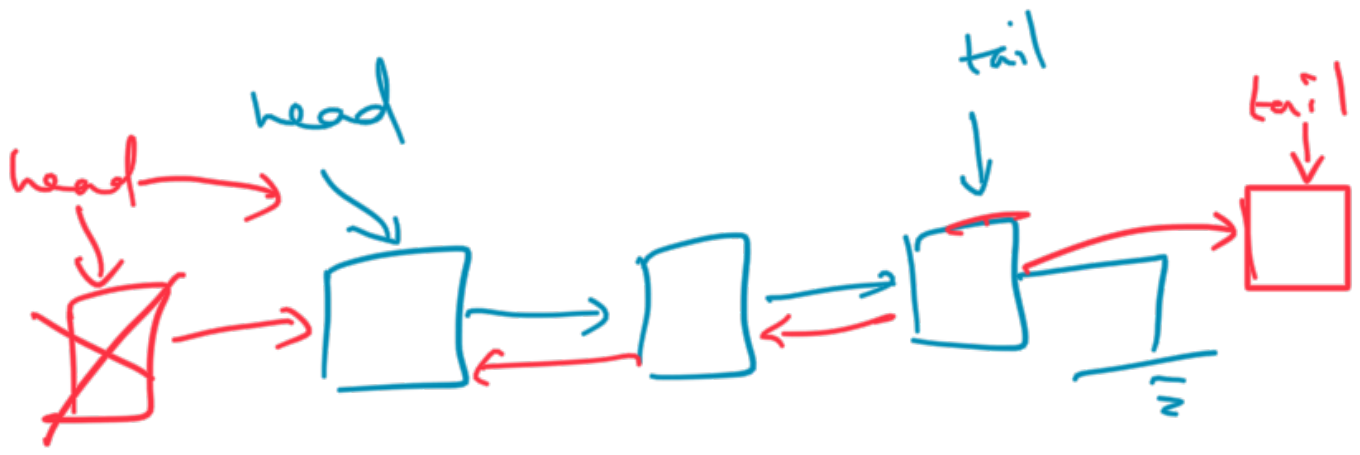
en(9)



using L.L.

~~Stack using~~

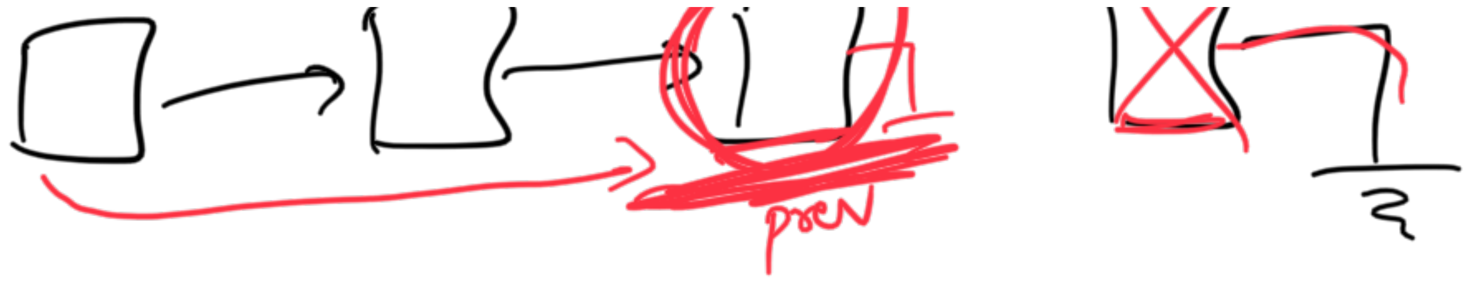
```
Class Node{
    int data;
    Node next;
    public Node(int a){
        this.data = a;
        this.next = null;
    }
}
```

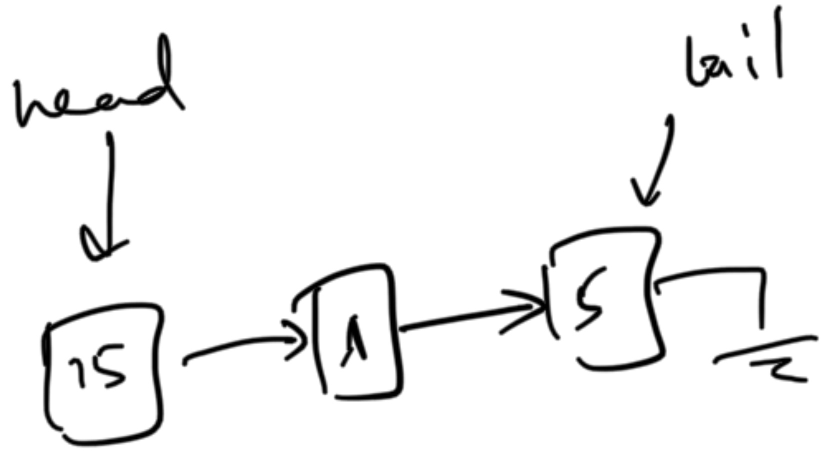


push → insert
pop → remove
} $\underline{O(1)}$.

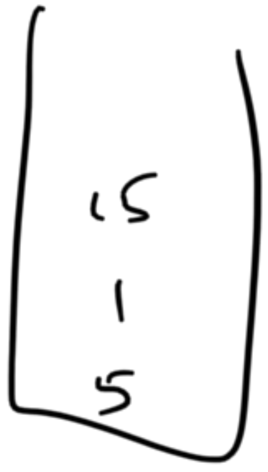
Linked List

- Add At Front (a) ⇒ O(1)
- Add At Tail (a) ⇒ O(1)
- Delete At Front ( ) ⇒ O(1)
- Delete At Tail ( ) ⇒ O(N)





push(5)
push(1)
push(10)
~~pop()~~
push(15)



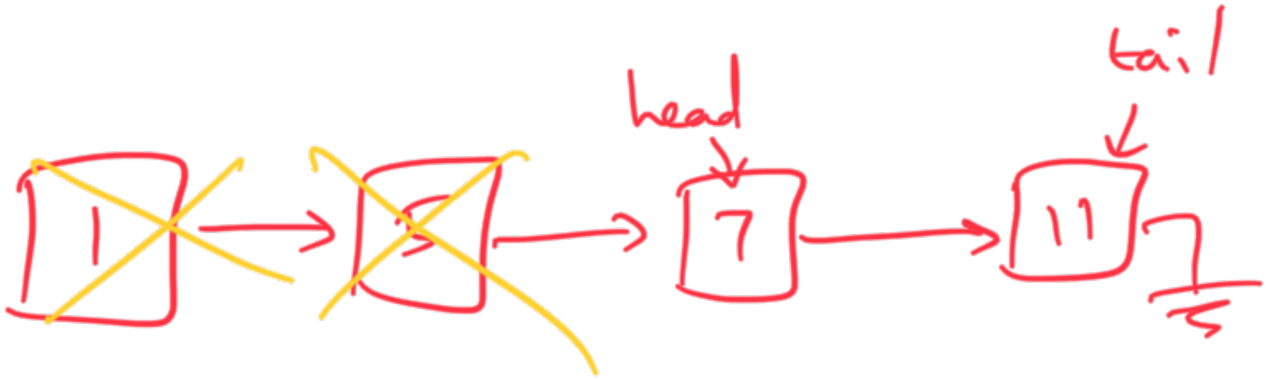
stack → push ⇒ Insert At Front
pop ⇒ Delete from Front } T.C ⇒ O(1)

Queue using L.L.

enq(1)
enq(5)
enq(7)

Enqueue ⇒ Add At Tail (a)
Dequeue ⇒ Remove At Front() } O(1)

deq()
deq()
enq(11)

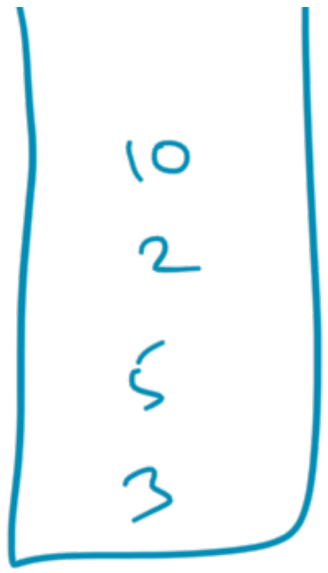


Break ⇒ 10:29

Q. Given a stack (Library/DS) (List stack). Need to implement a stack which allows you to perform getMin().

New Stack → push(a)
→ pop()
→ getMin() ⇒ Returns min of the stack

push(3)



push(5)
push(8)
getMin() $\Rightarrow$ 3
pop()
push(2)
getMin() $\Rightarrow$ 2
push(10)
push(1)
pop()
getMin() $\Rightarrow$ 2

var 'min' will not work!

stack <int>

Class Stack
{
stack<int>

push() $\rightarrow$ O(1)

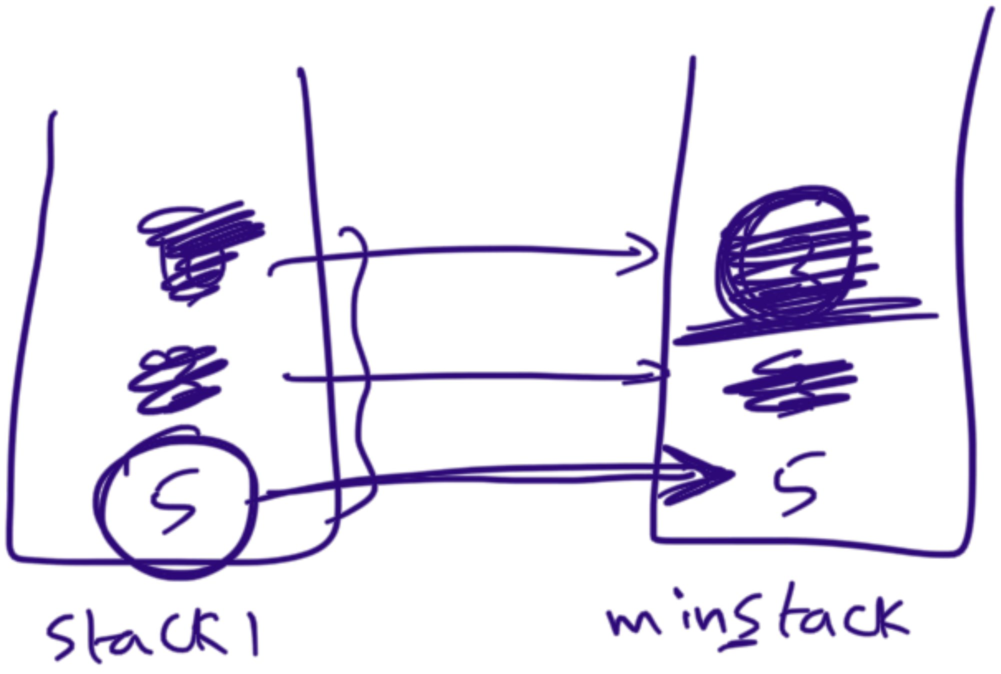
pop() $\rightarrow$ O(1)

getMin() $\rightarrow$ O(1)

( )

Approach (1)

push(5)
pus(3)
push(10)
getMin()
pop()
pop()
getMin()



Create 2 stacks

```
stack<int> st1, min_stack;

void push (int a)
{
      st1.push(a);
 curr_min = min(a, min_stack.top())
 min_stack.push(curr_min);

}
```

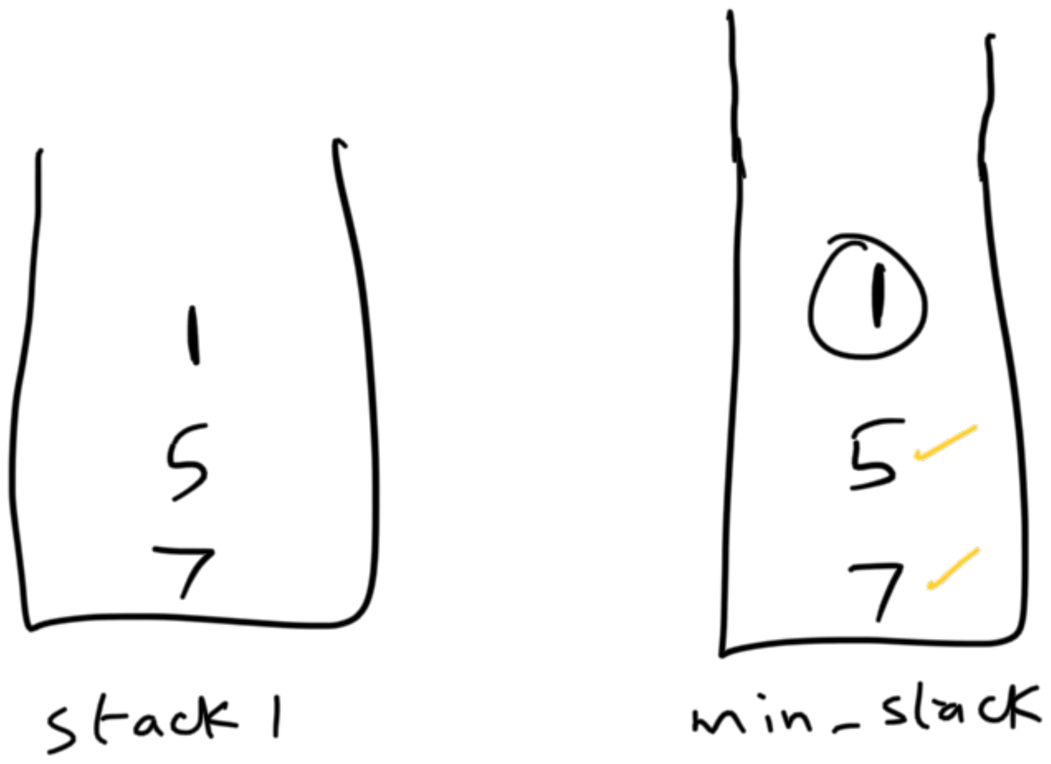
what if min_stack is empty?

```
void pop()
{
        st1.pop()
        min_stack.pop()

}
```

```
int getmin()
{
    min_stack.top();
}
```

```
push (7)
push (5)
push (1)
push (9)
getMin() => 1
pop()
pop()
getMin() => 5
push(1)
getMin() => 1
```



T.C ⇒ O(1) per operation.

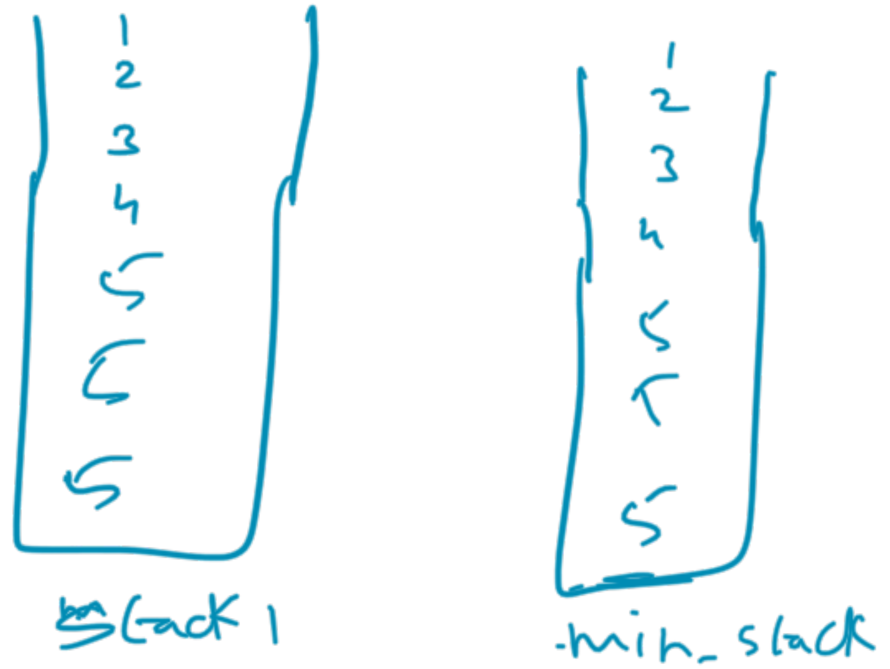
Approach 2

Only insert in min_stack when min is update

$\downarrow$

Fails for duplicates

→ insert when
element ≤ min_element.



```
void push(int a)
{
        stack1.push(a)
        if( min_stack.isEmpty()
               || a ≤ min_stack.top())
               minstack.push(a)
}
```

edge case → empty stack

```
void pop()
{
   [illegible] != stack1.top())
```

```
    if (minstack.top
            minstack.pop()

    stack1.pop()
}
```

$$\boxed{\begin{array}{l} T.C. \Rightarrow O(1) \\ S.C \Rightarrow O(N) \end{array}}$$ → per operatⁿ

Also a way to not use extra stack

Doubts

support@scaler.com

T.A.