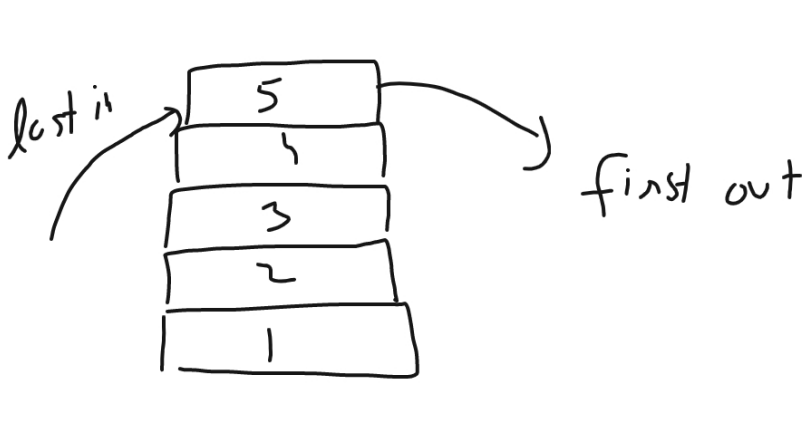
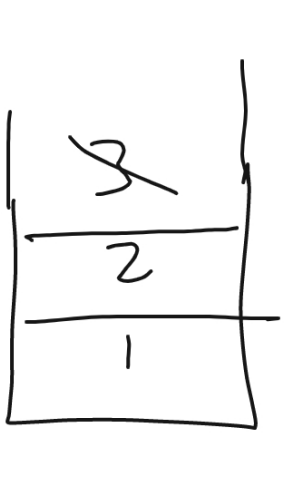
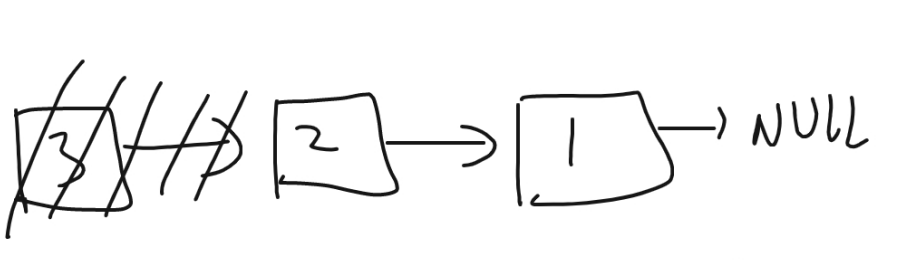
STACK

=> LIFO : Last In First Out

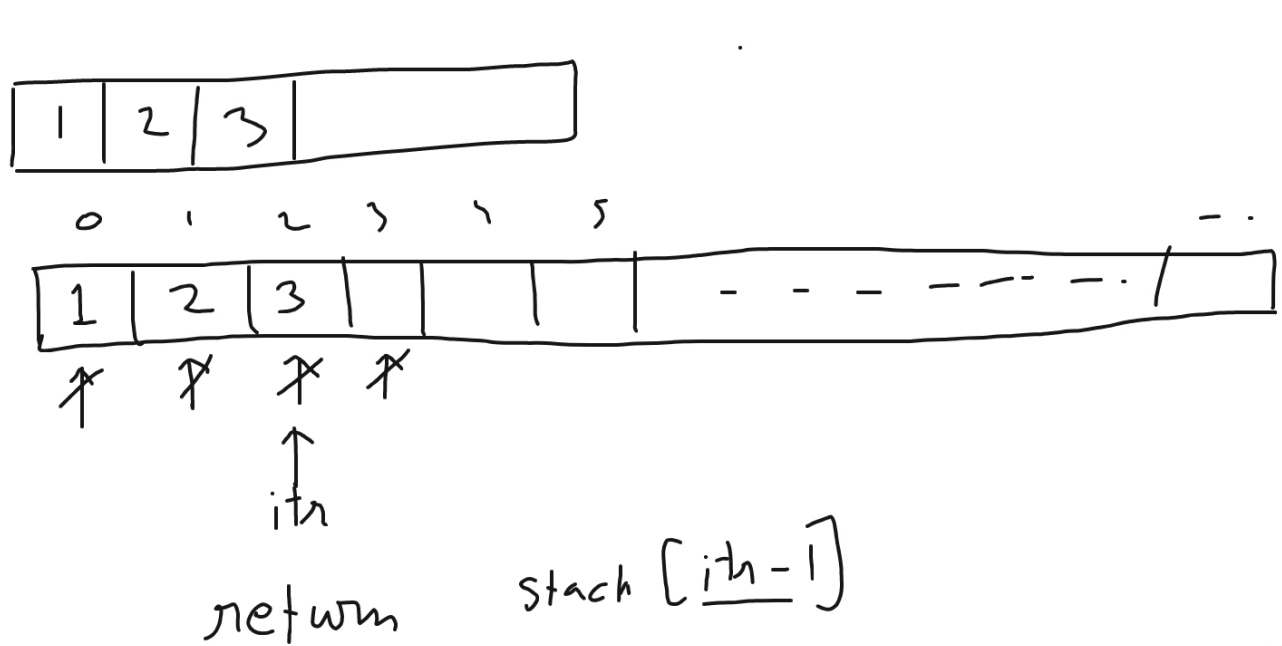


1) Push (1)
2) Push (2)
3) Push (3)
4) Pop
5) print (stack. top ()) => 2





If implemented via LL: Push => Pushing at beginning
Pop => Popping from the beginning



{ ([ ]) }
Balanced

{ [( ]) }
Not-Balanced

) ( ) (
Not-Balanced

[([ ])]

{ { } }

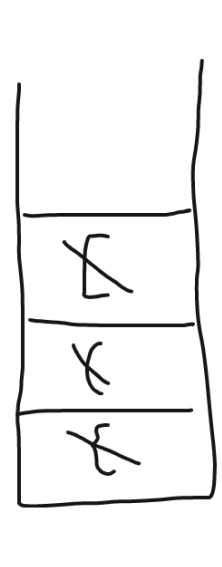
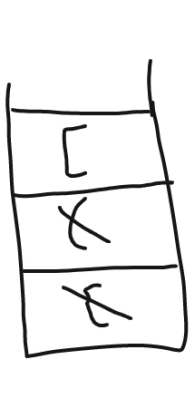
" { ( ) } [ ] "

=> Order of closing brackets should be the reverse of the order of opening brackets

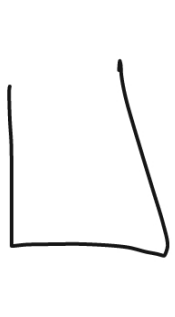
=> If a kind of a bracket opened at the end, then that kind of a bracket should be closed first.

" { ( ) } [ ] "

" { ( ) } ["

" } "



" ( )] "

" (((((((("

" (( )) "
Balanced

" ( )) "
Not-Balanced - - -

") ("

" (( )) "

[c1 : 0
c2 : 0]

" (( )) ( ) ( ) "

[op = ~~0~~ 1 2
cl = ~~0~~ 1 2]

=> [At any moment, the no. of opening bracket should be greater than or equal to the no. of closing brackets]

") ("

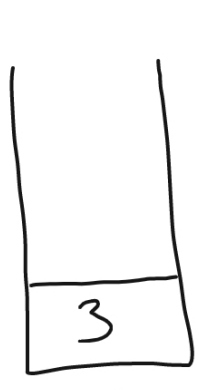
op = 0
clo = ~~0~~ 1

I/P = [6, 8, 0, 1, 3]
O/P => [8, -1, 1, 3, -1]

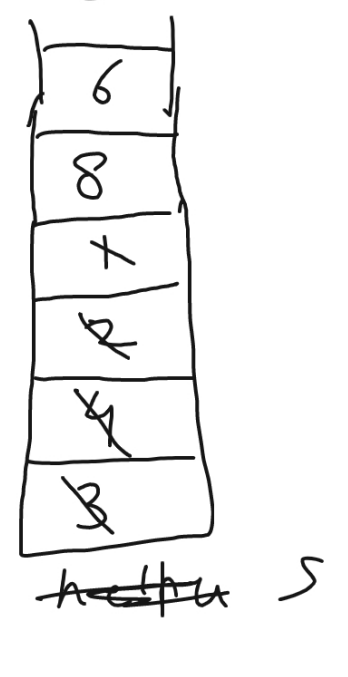
I/P = [6, 8, 0, 3, 1]
O/P = [8, -1, 3, -1, -1]

Brute-force
```
for i : 0 -> n-1
    for j : i+1 -> n-1
        if arr[j] > arr[i]:
            => out[i] = arr[j]
               break;
```
[TC: $O(n^2)$
Aux. space: $O(1)$]

[6, 8, 0, 1, 3]
-1

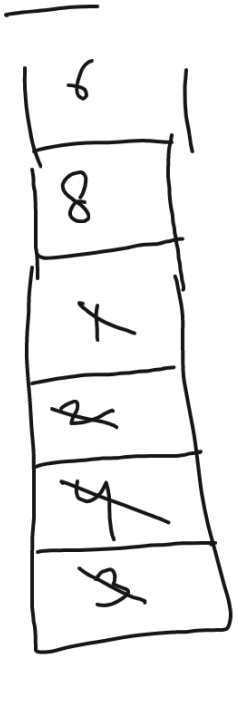


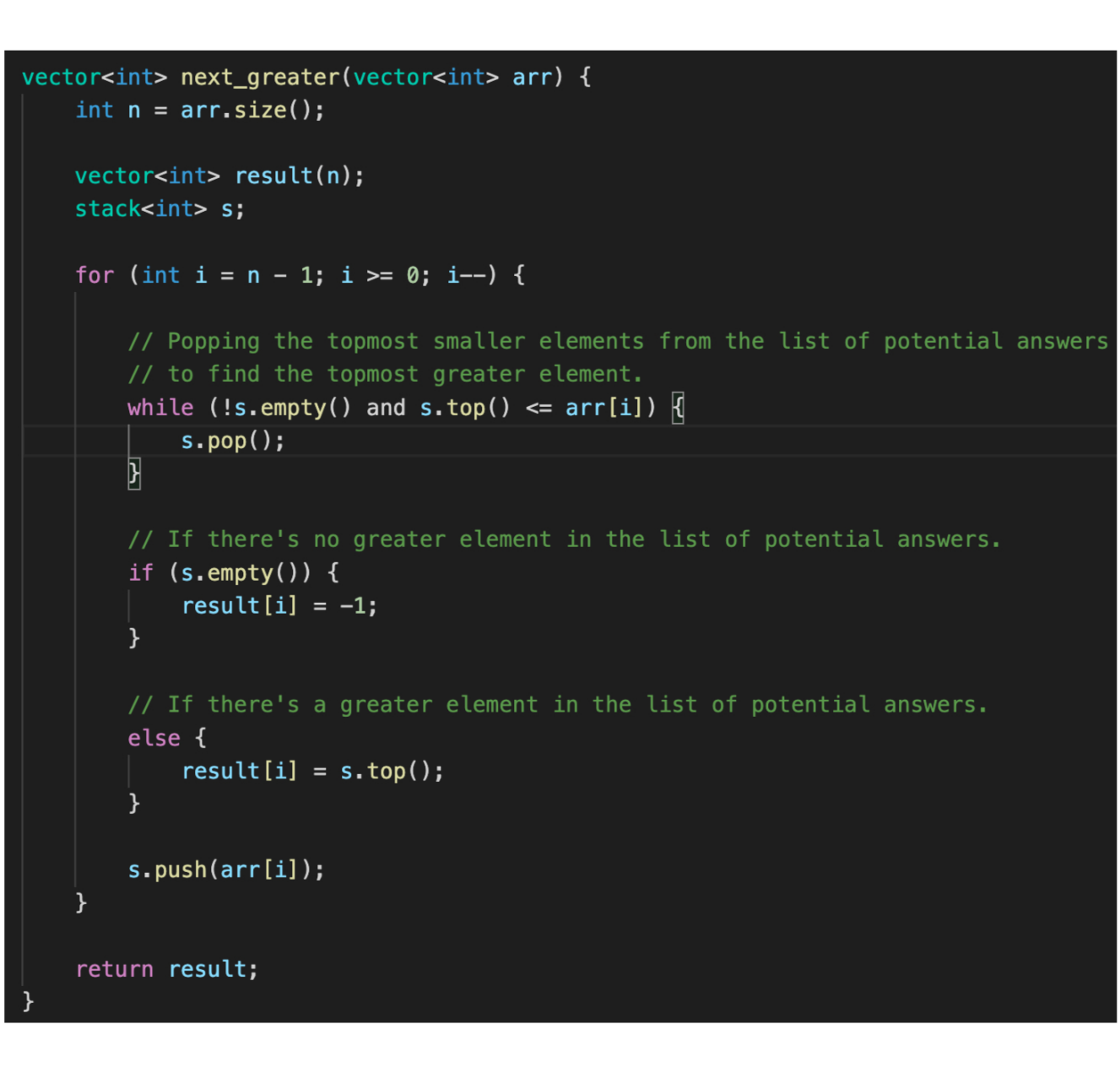
I/P [6, 8, 1, 2, 4, 3]
O/P [8, -1, 2, 4, -1, -1]



Iterate the array linearly from the end: (i):
```
-> while (! s. empty() && s.top() < arr[i]) {
        s. pop();
   }
   if (s. empty()) {
        out [i] = -1 ;
   }
   else {
        out [i] = s. top();
   }
   s. push (arr [i])
```

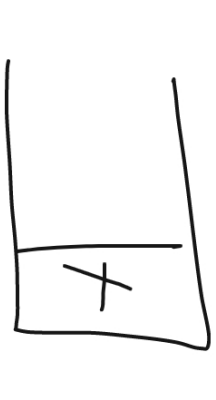
[6, 8, 1, 2, 4, 3]
[8, -1, 2, 4, -1, -1]





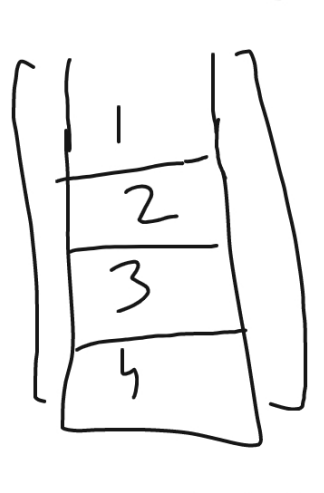
```
vector<int> next_greater(vector<int> arr) {
    int n = arr.size();

    vector<int> result(n);
    stack<int> s;

    for (int i = n - 1; i >= 0; i--) {

        // Popping the topmost smaller elements from the list of potential answers
        // to find the topmost greater element.
        while (!s.empty() and s.top() <= arr[i]) {
            s.pop();
        }

        // If there's no greater element in the list of potential answers.
        if (s.empty()) {
            result[i] = -1;
        }

        // If there's a greater element in the list of potential answers.
        else {
            result[i] = s.top();
        }

        s.push(arr[i]);
    }

    return result;
}
```

[4, 3, 2, 1]



[10, 5 3, 2, 4]
4

[1, 2, 3, 4]
2 3 4 -1



---

0 1 2 3 4 5 6
[6, 2, 5, 4, 5, 1, 6]
prev_smaller : [-1, -1, 1, 1, 3, -1, 5]
next_smaller : [1, 5, 3, 5, 5, -1, -1]

area = arr [i] × (next_smaller [i] − prev_smaller [i] − 1)
= 4 × (5 − 1 − 1)
= 4 × 3 = 12

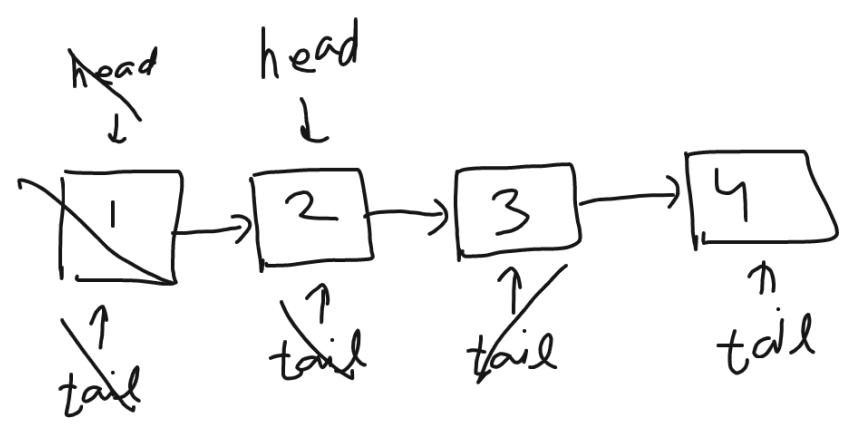
QUEUE

[FIFO : First In First Out]

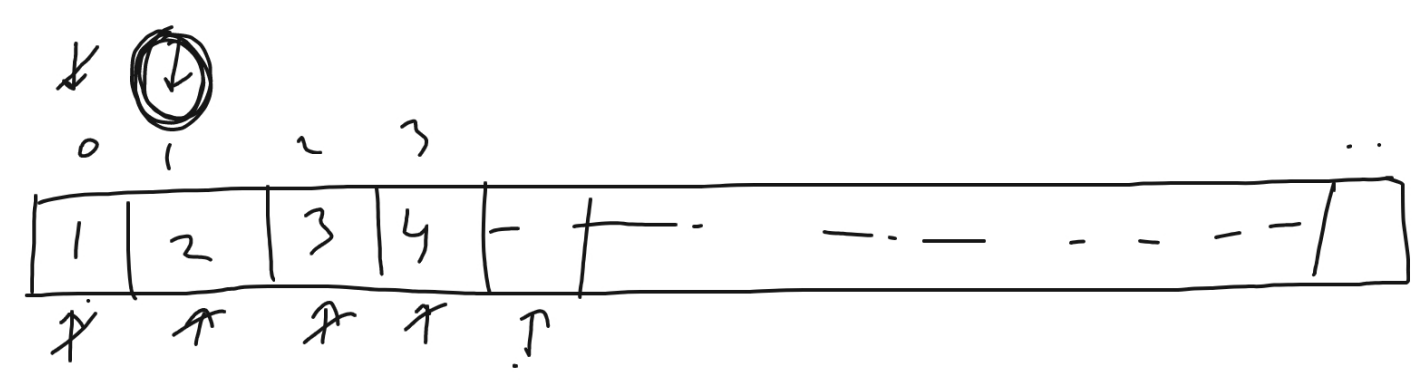
1) Push (1)
2) Push (2)
3) Push (3)
=> 4) Pop ()
5) Push (4)
6) Print (queue. front ()) => 2

~~1~~, 2, 3, 4

LL : Insert at the end
Delete from the beginning

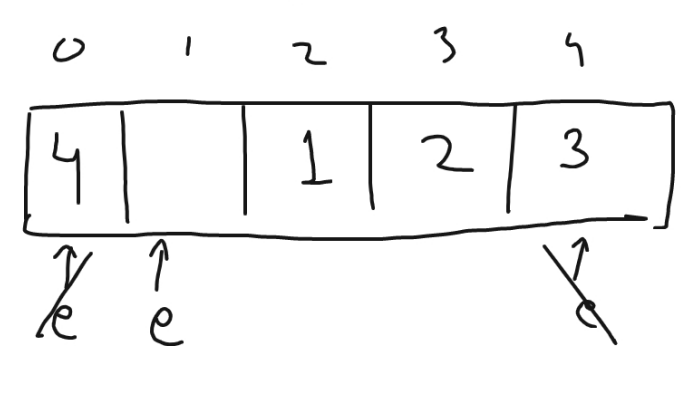


Array:



([(i+1) % n])

n = 5



Push (3)
Push (4)

s = 2
e = ~~4~~ 0

(4+1) % 5 = 0
(0+1) % 5 = 1

(n-1 +1) % n = 0

n (0 -> n-1)

[rand() % n]

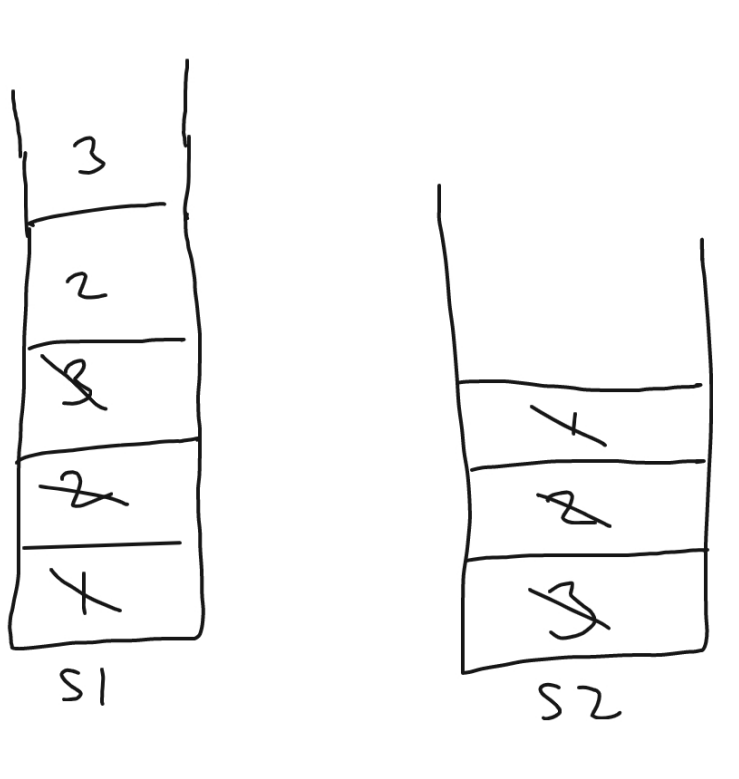
---

class Queue {
data-members: Two stacks
push()
pop()
~~front~~()
is_empty ()
size()
}

1) Push(1)
~~2) Push(2)~~
~~3) Push(3)~~
-> 4) Pop()
5) Push(4)
6) Push(5)
7) Pop()

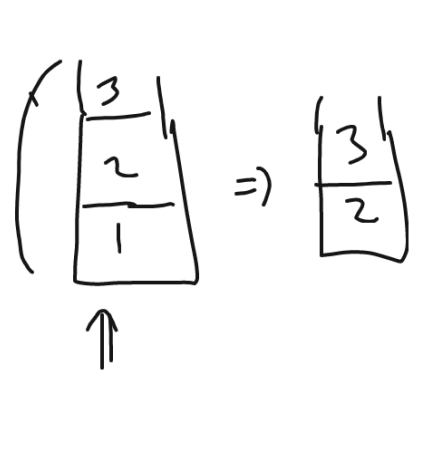
~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~

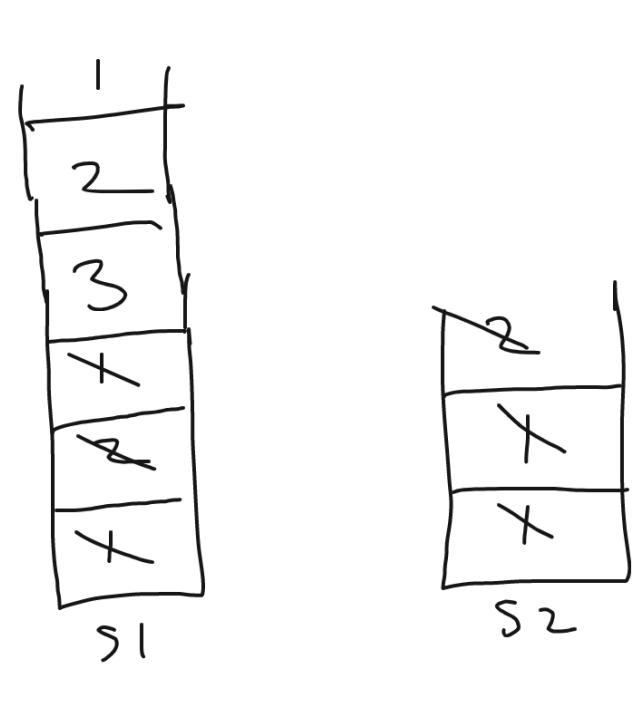
O/P
1
2
3
4
5
error



(answer = 1)

[Queue-push : O(1)
Queue-pop : O(n)]





=> [Queue_push : O(n)
Que-pop : O(1)]

N = 10
1 -> 10

-> 1
-> 1 0 (1)
-> 1 1
-> 1 0 0 (10)
-> 1 0 1
-> 1 1 0 (11)
1 1 1
1 0 0 0 (100)
1 0 0 1
1 0 1 0 (101)
1 0 1 1
1 1 0 0 (110)
1 1 0 1

O/P : 1, 10, 11, 100

~~1~~, ~~10~~, ~~11~~, ~~100~~, 101, 110, 111, 1000, 1001

current + "0"
current + "1"