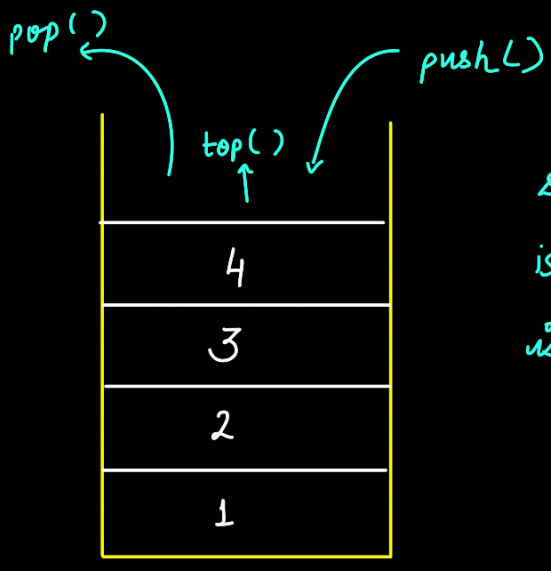
[STACKS]:

↳ Last in First Out principle followed. (LIFO)



size()
isEmpty()
isfull()

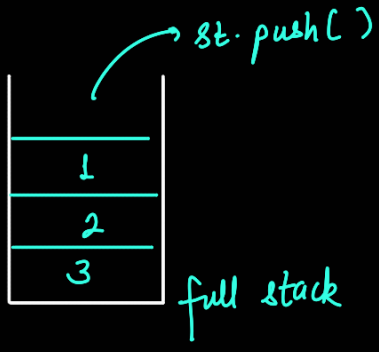
note: all stack fⁿ work under O(1) complexity (time)

Stack using arrays / linked list:

Stack size = 3

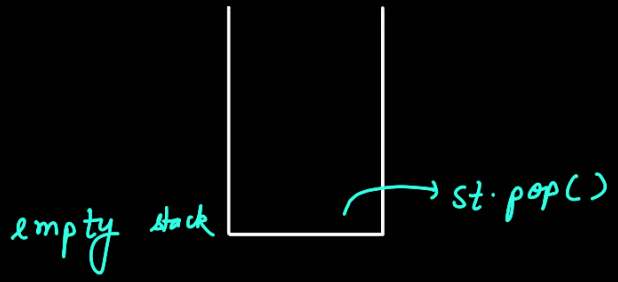
1) overflow

↳ stack ke size se jyada element add karne ke time.



11) Underflow

↳ trying to remove when the stack is currently empty chahe size kuch bhi ho.



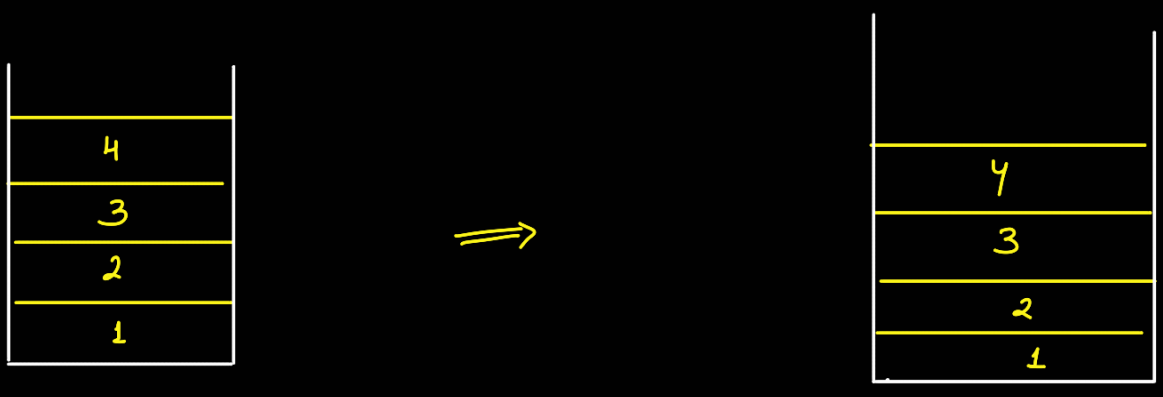
* Implementing stacks using arrays: VS code:

```
class stack {

    int capacity;  // we have hidden the implementation of array
                   // as we don't want them to be shown.
```

```
int* arr;
int top;
}
```

Copy Stack Recursive ↳



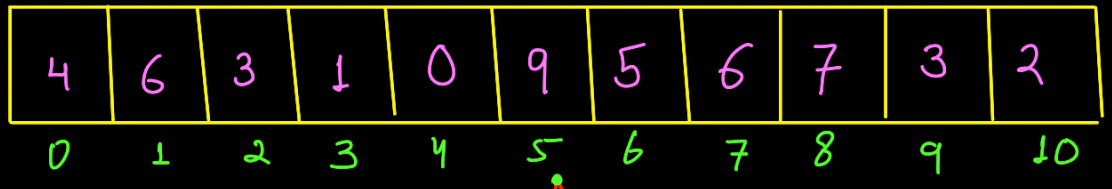
[PROBLEMS:] ① Balanced Parenthesis
② Next Greater Element / next smaller / prev smaller / prev greater.

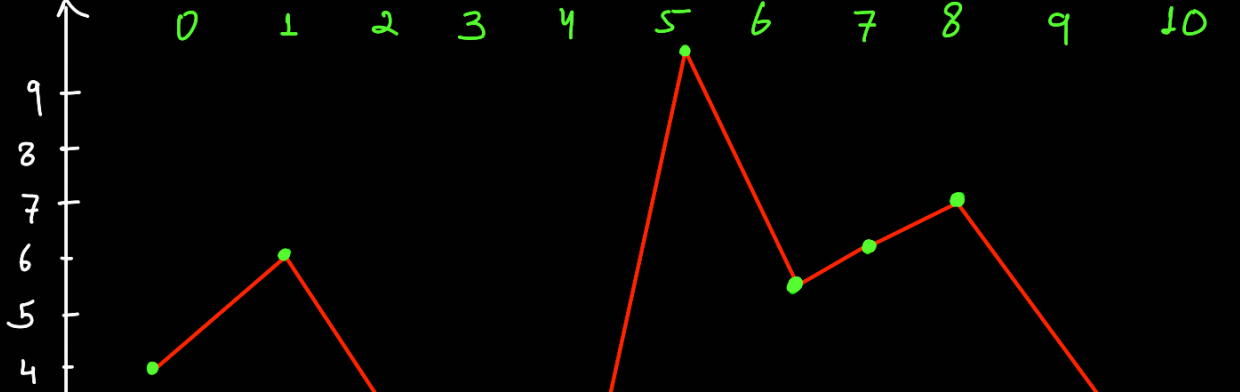
⇒ Given an array of integers return the array of next greater elements.

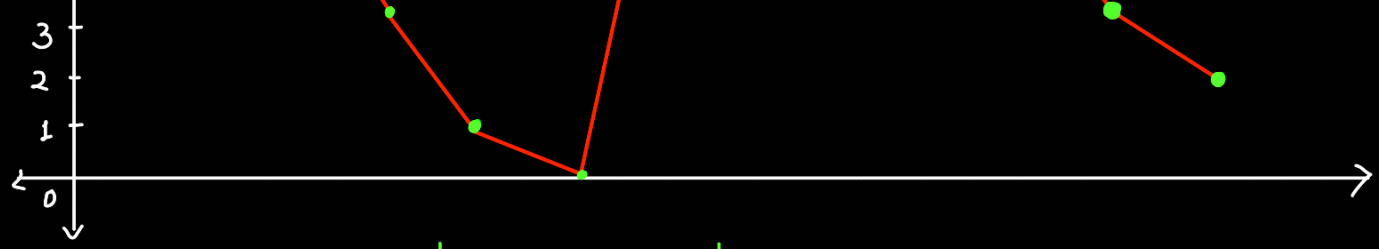
use stacks when Last element thing in question, we have to let some elements wait for their turn to get the answer.

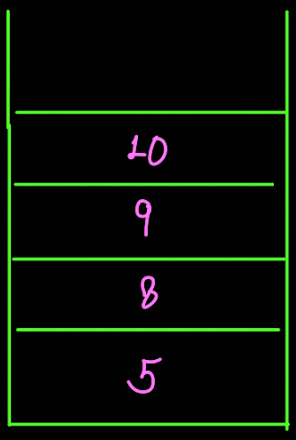
Time and Space : O(n) and O(n)

i/p →

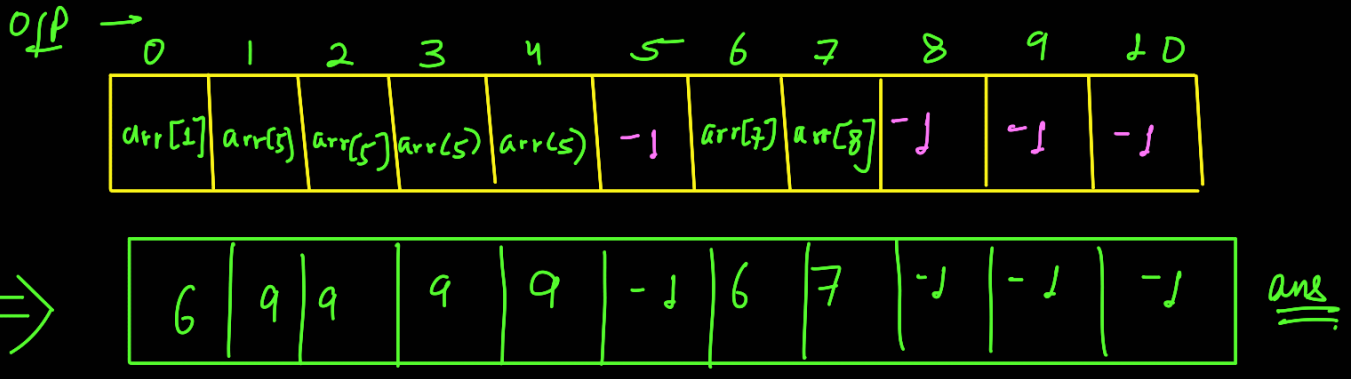






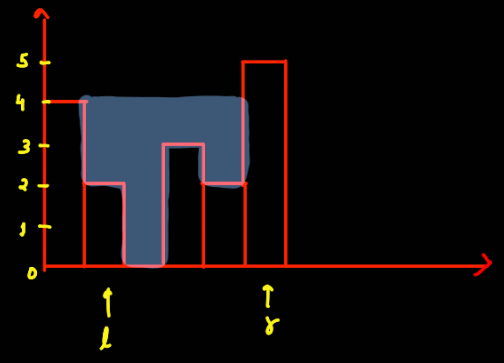


store indexes



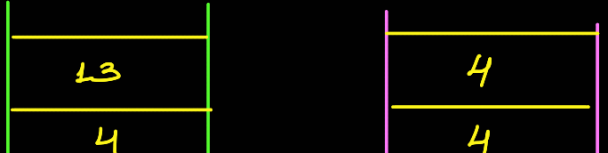
* most of the good problems of stacks are based on next greater / next smaller / prev greater / prev smaller tactics.

eg → Stock Span, Largest Area Histogram.

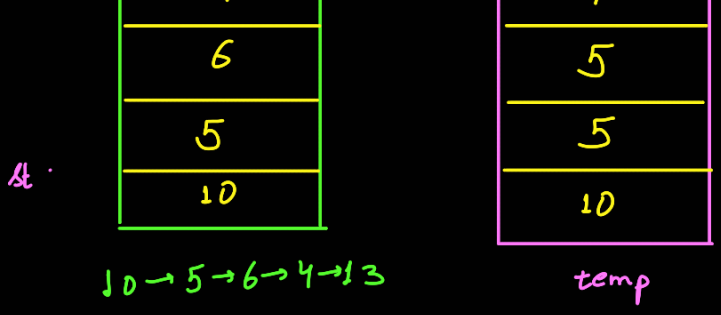


Interview Problems on Stacks: Post fix / Pre fix / In fix etc.

⇒ #1 Implement min stack:




```
min_ele = min(minele, ele);
if element < temp.top() {
    temp.push(minele);
```




```
}
```

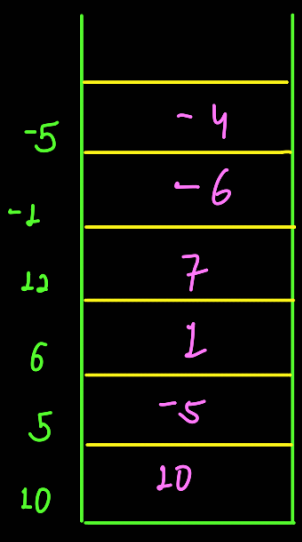
temp.top( ) = min element.

T.C. = O(1)

S.C = O(N)

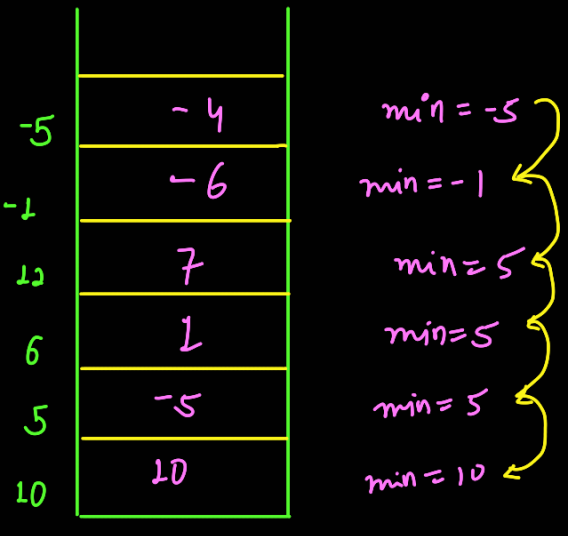
↳ maintain a temporary stack and push only if the element is less than the top element of temporary stack.

T.C. O(1) and S.C. O(1)

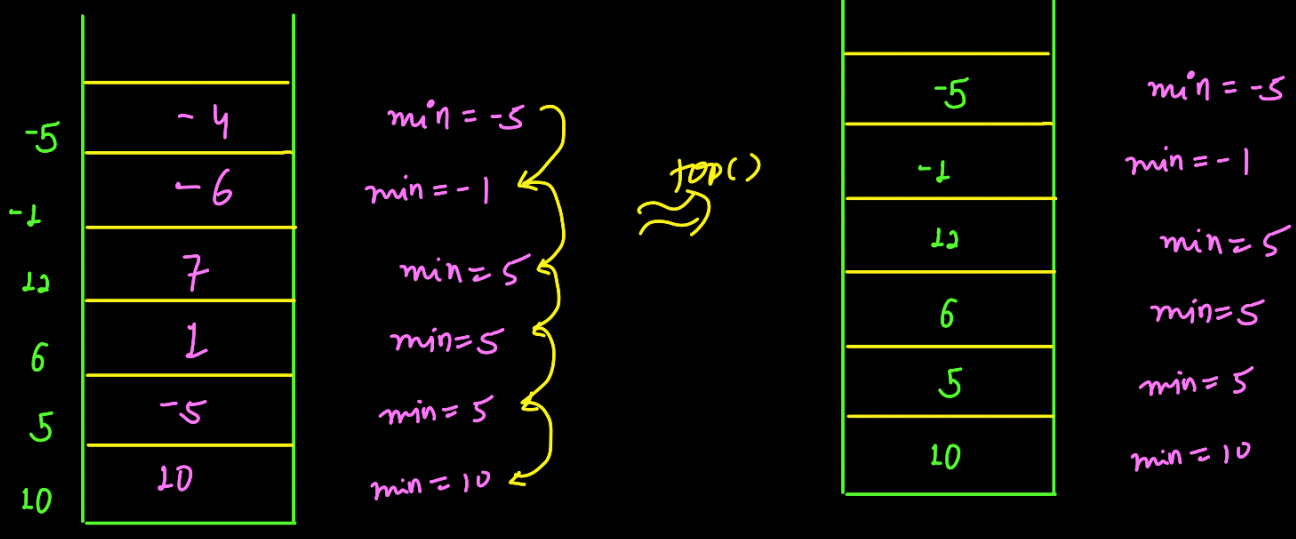



```
void push( int val ){
if ( this → st.empty ) { // for 1st element
    this → mn = val;
    this → st.push(val); }
else { if ( this → mn > val ) {
        this → st.push ( val - this → mn
    this → min = val; }
```

pop ⇒




```
void pop ( ) { // will effect minimum
    if ( not st.empty( ) ){
        if ( st.top( ) >= 0) {
            st.pop( ); }
    else { mn = mn - st.top( );
           st.pop( );
    }
```




```
int top ( ) {
  if (st.size() == 1) {
    return st.top();}
  else { if (st.top() < 0) {
           return mn;
         else return mn + st.top();
  }
```

```
int getMin() {
  return mn;
}
```

Post| Pre| Infix Implementations :

Infix expressions : a+b x (c-d+e)x r

↳ operator operand operator (BODMAS)

(x+y) - z

Postfix expressions : (Reverse Polish notation)

↳ A B C D + - +

= A B C+D - +

↓

E

= A B - E +

↓

F

= A + F = result

Prefix / Polish notation : +-ABC

+ A-B C

D

D+C = result

Evaluation of postfix expressions : → code in VS Code.

↓

also eval of prefix and infix operations.

2 3 1 * + 9-

2 3x1 + 9-

2 3 + 9-

2+3 9-

5 9-

5-9

= -4

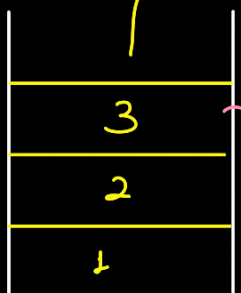
[QUEUES :]

Linear (Data Structure):

⇒ principle → FIFO (first in first Out)

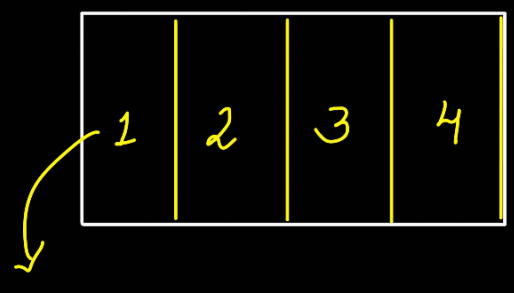
FCFS (First come first serve)

Stack



front

Queue



enqueue (1)
enqueue (2)
enqueue (3)
enqueue (4)
dequeue ( )

back/rear

* Types of Operations on Queues :

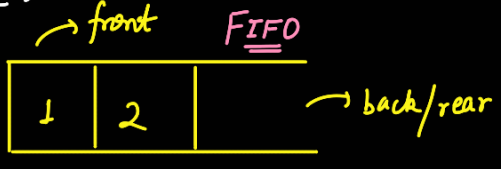
(1) Enqueue : this operation will help us to add a new element in queue.

(2) Dequeue : this operation will help us to remove a new element in the queue.

(3) isFull (4) isEmpty (5) front → gives us the element that came first.

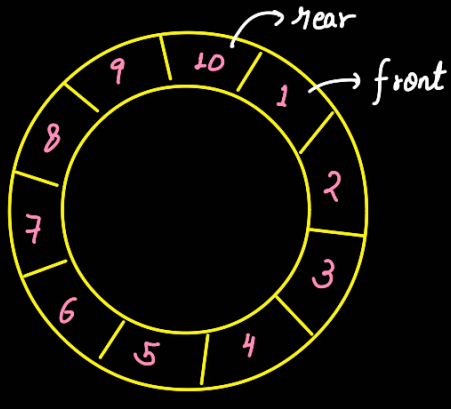
Types of Queues :

(I) Simple :

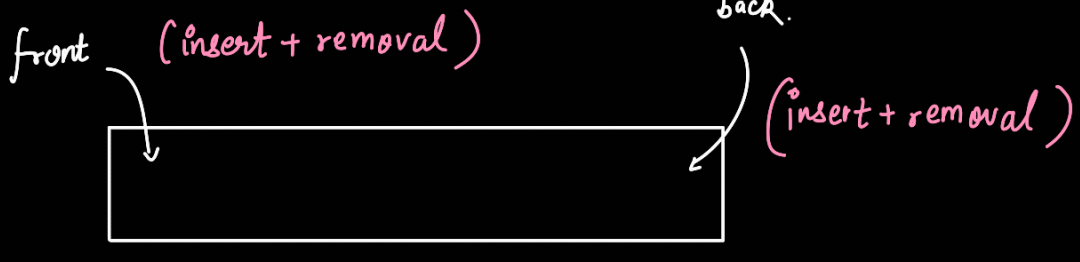


(II) Priority queue : queue based on a property not time based. (eg → max queue → max elements at front ----)

(III) Circular queue :



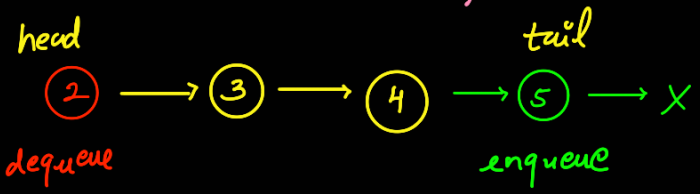
(IV) Double ended queue : (Deque)



Implementation of queues

Implementation of queues

- using arrays
  - deletion easy
- using linked-lists
  - space efficient
  - enqueue → add at tail.
  - dequeue → remove from head

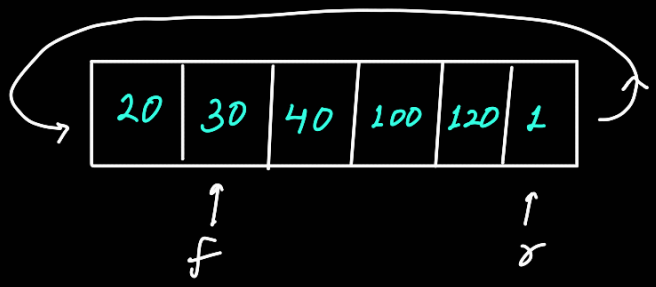


Deque : addition and removal from any end.

(read as deck)

HW ⇒ Implement deque using 2-L-L.

Circular Queues :



$\Rightarrow r = (r+1) \% n \quad \forall \quad r \in [0, n-1]$

$\Rightarrow f = (f+1) \% n$

when $f == r$ ⇒ single element present in circular queue.

when $f > r$ ⇒ Circular queue empty.