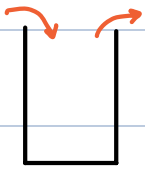
SCALER

Stacks 1

TABLE OF CONTENTS

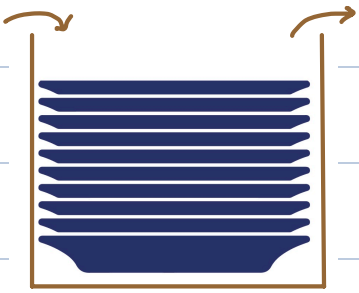
SCALER

Stacks

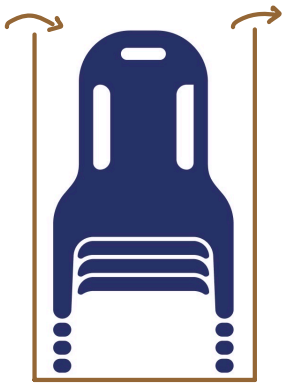


LIFO Last In First Out

1. Stack of Plates

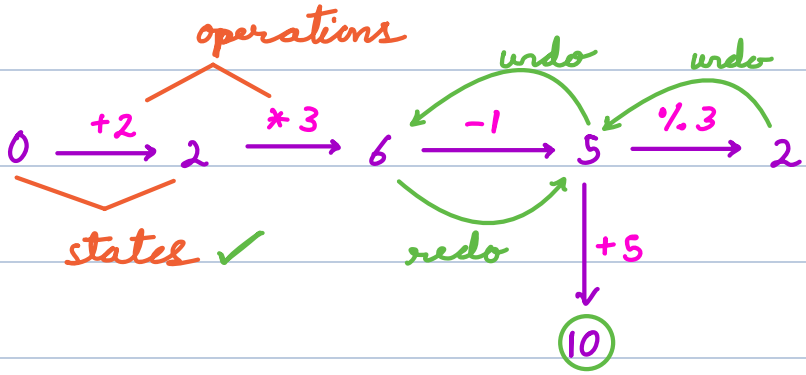


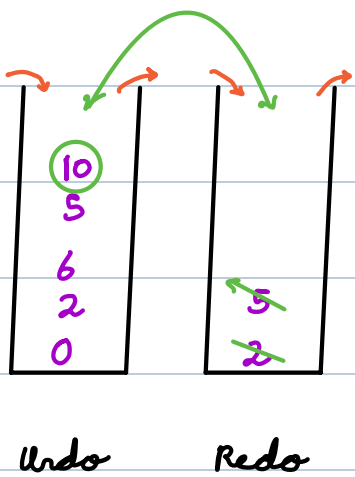
2. Stack of Chairs



3. Stack of Functions (Recursion)

4. Undo / Redo



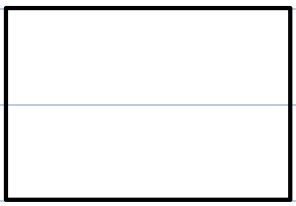


If we perform a new operation → the redo stack becomes empty.

Scenario

In a Flipkart warehouse, boxes are stacked one over another. Each box is tagged with a weight, and for safety, heavier boxes must be placed below lighter ones. Workers need a system to check if adding a new box on top is safe.

Workers want to perform two kinds of operations:-

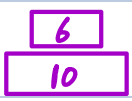


Type ADD : a new box of some weight on the top

Type REMOVE : the topmost box

Example :

Query type	Value	Answer (True/False)
ADD	10Kg	true ✓
ADD	5Kg	true ✓
ADD	12Kg	false (not allowed) ✗
REMOVE	-	true ✓
ADD	6Kg	true



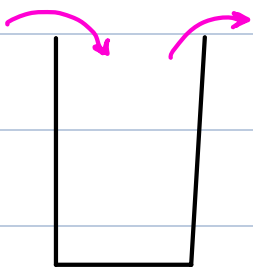
insert → if (wt of new box > wt of top box)
return false
else return true

SCALER

Operations in Stack

1. **Push (x)** - Insert X on the top of stack.
2. **Pop ( )** - Remove the top element.
3. **Top ( ) / Peek ( )** - check / Get the top element.
4. **isEmpty ( )** - checks if stack is empty.

TC = O(1)

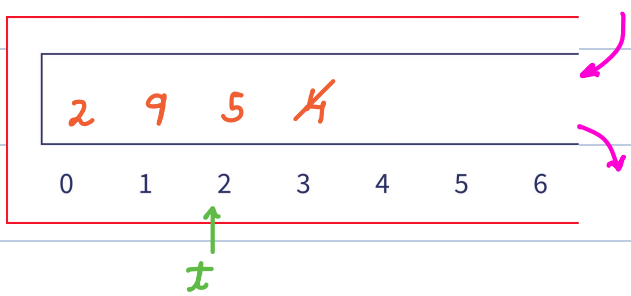


Implementing Stacks using Arrays

push (2)
push (9)
push (3)
peek () → 3
push (4)
pop () → 4
pop () → 3
push (5)

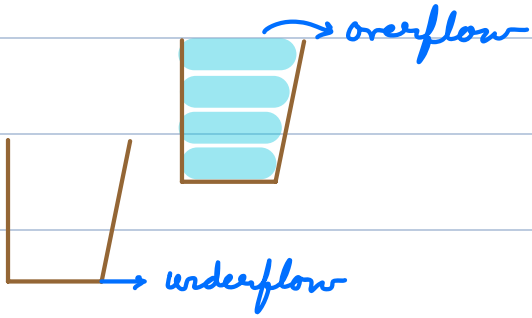
Stack

0 —— t



t = -1

```
boolean isEmpty () {
    return (t == -1)
}
```



SCALER

```
void push (x) {                    // overflow
    t++                            1) Use dynamic array ✓✓
    A[t] = x                       2) Do not insert if array
}                                     is full. (t == N-1)
```

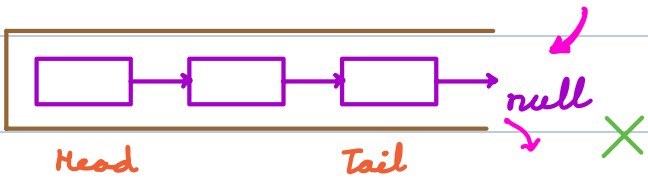
```
int peek () {
    if (isEmpty ()) return -1      // underflow
    return A[t]
}
```

```
int pop () {
    if (isEmpty ()) return -1
    top = A[t]
    t--
    return top
}
```

TC ∀ operations → O(1)

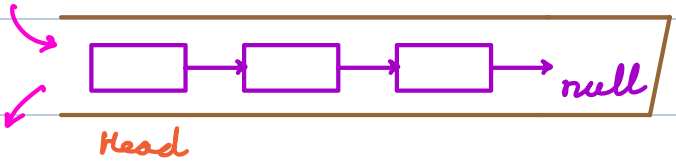
SCALER

Implementing Stacks using Linked List



insert as last node → TC = O(N) → O(1)

removing last node → TC = O(N)



insert head → TC = O(1)

remove head → TC = O(1)

push (x) → insert x as head

peek () → return head data

pop () → remove head node

} check underflow

isEmpty () → check if head is null

TC = O (1)

< *Question* > : Check whether the given sequence of parenthesis [ ], { }, ( ) is valid or not?

Example : ()[{}()] ✓

()[] ✓

({}) ✓

({)} x

([{]) x

([{}[]()]) ✓

([{}[]()])

([{}

store data but use latest data first ⇒ LIFO ⇒ stack

open bracket → store

close bracket → check & remove pair.

([{}[]()])

~~(~~	{} ✓
~~[~~	[] ✓
~~{~~	[] ✓
~~[~~	() ✓
~~(~~	() ✓

SCALER

```
// stack → st
for i → 0 to (N-1) {
    ch = s[i]
    if (ch == '{' || ch == '(' || ch == '[') {
        st.push(ch)
    } else if (ch == '}') {
        if (st.peek() != '{') return false
        st.pop()
    } else if (ch == ']') {
        if (st.peek() != '[') return false
        st.pop()
    } else {
        if (st.peek() != '(') return false
        st.pop()
    }
}
return st.isEmpty()
```

TC = O(N) SC = O(N)

SCALER

Double Character Trouble

Given a string str. Remove equal pair of consecutive elements till possible.

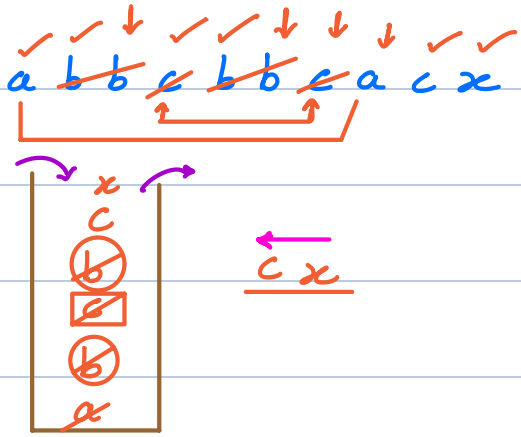
Return the strings without adjacent duplicates.

1. a ~~b b~~ d → ad
2. a b ~~c c~~ b d e → a ~~b b~~ d e → a d e
3. a ~~b b~~ b d → a b d
4. a ~~b b~~ c ~~b b~~ c a c x → a ~~c c~~ a c x → a ~~a~~ c x → c x

a b c ~~d d~~

store data but use latest data first ⇒ LIFO

⇒ stack



SCALER

```
// stack → st
for i → 0 to (N-1) {
    ch = s[i]
    if (st.isEmpty() || (st.peek() != ch))
        st.push(ch)
    else    st.pop()
}

ans = ""
while (!st.isEmpty()) {
    ans = st.pop() + ans
}

return ans
```

TC = O(N) SC = O(N)

SCALER

infix	postfix
2 + 3	2 3 +
2 + (5*3)	2 5 3 * +
operand1 operator operand2	operand1 operand2 operator

< *Question* > : Evaluate given valid post fix expression.

arr [ ] -

3 + 5 = 8

2 * 5 = 10

8 - 2 = 6

6 - 10 = -4 (Ans)

Quiz : 5 , 2 , * , 3 , -

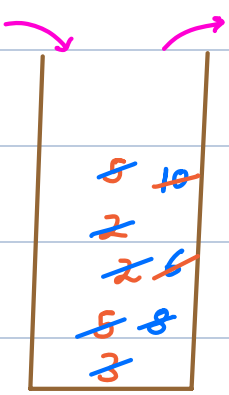
operator → select latest 2 operands & evaluate.

5 * 2 = 10

10 - 3 = 7 (Ans)

✓ ✓ ↓ ✓ ↓ ✓ ✓ ↓ ↓

3 5 + 2 - 2 5 * -



y = st.pop()

x = st.pop()

x operator y

3 + 5 = 8

8 - 2 = 6

2 * 5 = 10

6 - 10 = -4 (Ans)

SCALER

H.W → code using stack.

TC = O(N) SC = O(N)

SCALER

Stacks - 2

TABLE OF CONTENTS

SCALER

Nearest Smaller Element

Q1→ Given an integer array find the index of nearest smaller element on left of i ∀ A[i].

	0	1	2	3	4	5	6	7
A =	8	2	4	9	7	5	3	10
	-1	-1	1	2	2	2	1	6

∀i, find max j s.t A[j] < A[i] & j < i.

	0	1	2	3	4	5	6	7
A =	4	6	10	11	7	8	3	5
	-1	0	1	2	1	4	-1	6

	0	1	2	3	4	5
A =	4	5	2	10	8	2
	-1	0	-1	2	2	-1

Bruteforce → TC = O(N²) SC = O(1)

	0	1	2	3	4	5	6	7
A =	(8)	_	_	_	_	5	_	_

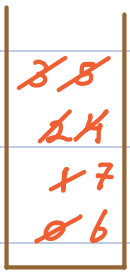
For any i > 5, can index 0 be the ans ?

No, ∀ i > 5, if 8 < A[i] ⇒ 5 < A[i]

& closer index will be 5.

SCALER

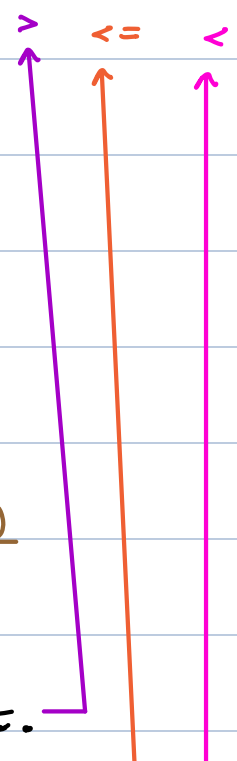
	0	1	2	3	4	5	6	7 (i)
A =	4	6	10	11	7	8	3	5
	-1	0	1	2	1	4	-1	6



checking → latest index first ⇒ LIFO ∴ use stack.

```
// stack st
for i → 0 to (N-1) {
    while (!st.isEmpty() && A[st.peek()] >= A[i]) {
        st.pop()
    }
    if (st.isEmpty())   ans[i] = -1
    else                ans[i] = st.peek()
    st.push(i)
}
```

TC = O(N) SC = O(N)



Q2 → Find nearest smaller or equal on left.

Q3 → Find nearest greater element on left.

Q4 → Find nearest greater or equal on left.

SCALER

Q5 → Find nearest smaller element on right.

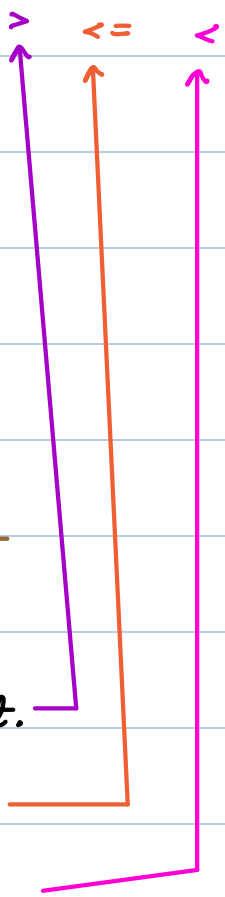
```
// stack st
for i → (N-1) to 0 {
    while (!st.isEmpty() && A[st.peek()] >= A[i]) {
        st.pop()
    }
    if (st.isEmpty())   ans[i] = -1
    else                ans[i] = st.peek()
    st.push(i)
}
```

TC = O(N) SC = O(N)

Q6 → Find nearest smaller or equal on right.

Q7 → Find nearest greater element on right.

Q8 → Find nearest greater or equal on right.



- **A person uses Google Maps to find the nearest restaurants and picks one based on it's proximity. Unfortunately, after visiting, they realised that the restaurant didn't meet their expectations.**

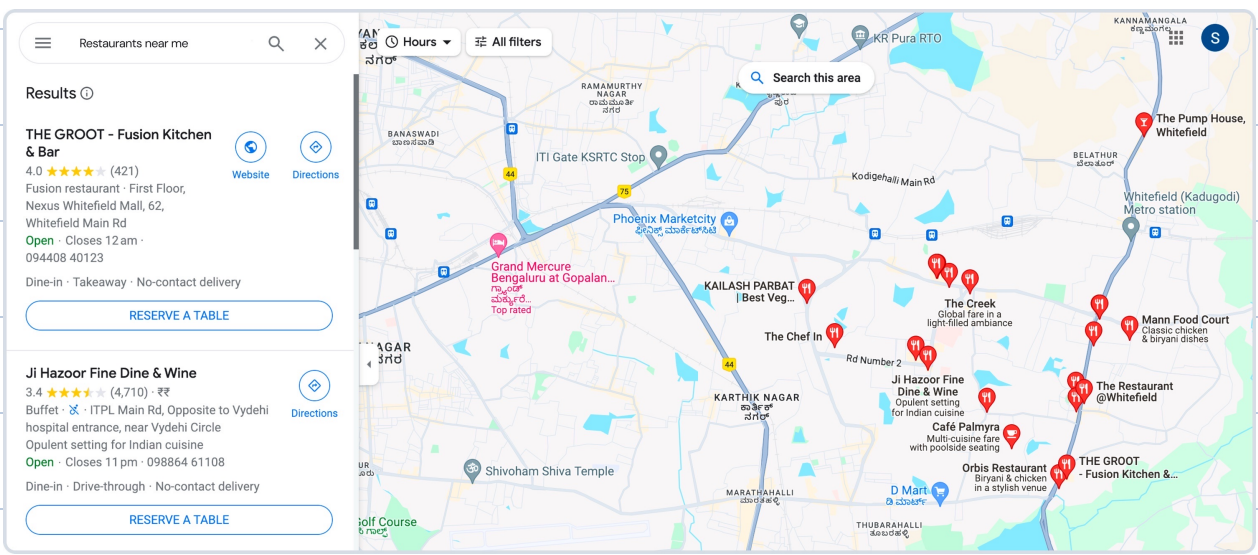
Task

Let's break it down with a simple example. You have a list of restaurants and their ratings. For each restaurant, we're going to find the next restaurant to the right on the list that's not just close but also has a higher rating than the current one. If there's no better option on the list, we'll say there's none available.

Problem

Given a sequence of restaurants listed on Google Maps with their ratings, create a tool that helps users discover the rating of the next higher-rated restaurant to the right for each listed establishment.

Nearest greater on right.



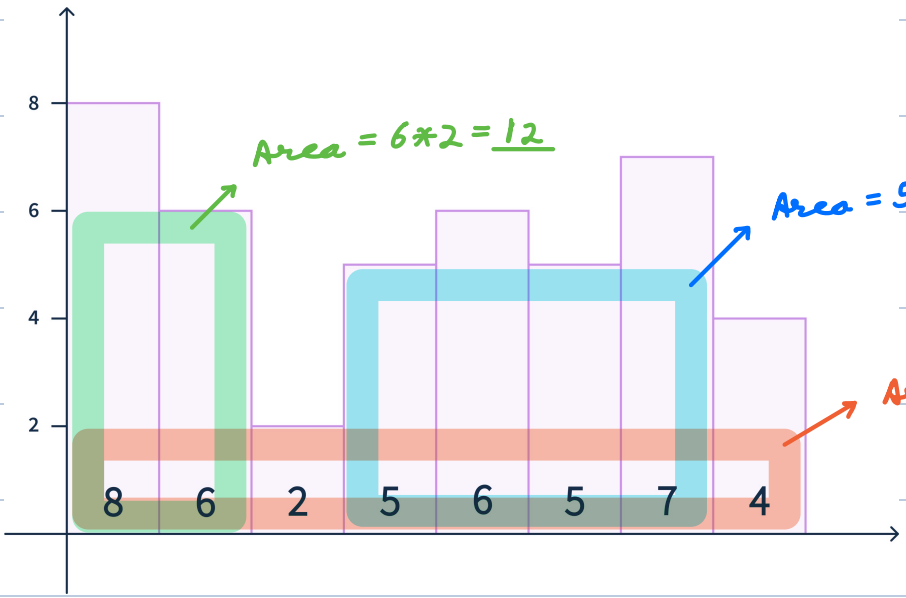
SCALER

< Question > : Find the largest area of rectangle (formed by continuous bars) in histogram.

H * W

A[i] → Height of i^{th} bar

Width = 1 ∀ bars



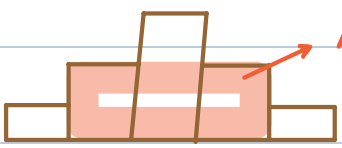
Area = 6*2 = 12

Area = 5*4 = 20 ✓

Area = 2*8 = 16

0 1 2 3 4

A = [1 2 3 2 1]



Area = 2*3 = 6 ✓

Idea → Find continuous bars that give largest rectangle.

Bruteforce → ∀ subarray, calculate

area = (min A[i]) * length of subarray.

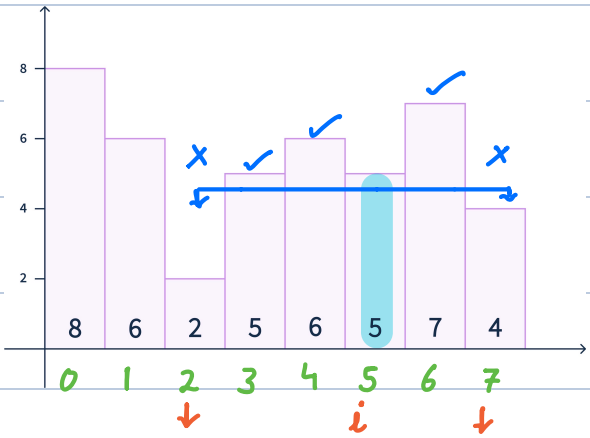
carry forward l — r ⇒ r - l + 1

TC = $O(N^3)$ → $O(N^2)$ SC = $O(1)$

SCALER

TC < O(N²) ⇒ we cannot travel all subarrays i.e width.

Idea 2 → ∀ height find max width.



[j+1 k-1]

width = k - 1 - (j+1) + 1

= k - j - 1

Nearest smaller on left = j

Nearest smaller on right = k

$$Ans = \max_{\forall i} (A[i] * (nearestSmallerRight[i] - nearestSmallerLeft[i] - 1))$$

TC = O(N + N + N) = O(N)

SC = O(N)

< Question > : Find sum of (max-min) for all subarrays. $1 \le N \le 10^5$

arr : [1 2 3] (0 1 2)

	max	min	max-min
[1]	1	1	0
[1,2]	2	1	1
[1,2,3]	3	1	2
[2]	2	2	0
[2,3]	3	2	1
[3]	3	3	0
			4 (Ans)

Bruteforce → TC = O(N^3) ⟶ O(N^2) (carry forward)

SC = O(1)

A = [2 5 3] (0 1 2)

2 → 2 - 2 = 0

2 5 → 5 - 2 = 3

2 5 3 → 5 - 2 = 3

5 → 5 - 5 = 0

5 3 → 5 - 3 = 2

3 → 3 - 3 = 0

8 (Ans)

2*(1-3) = -4

5*(4-1) = 15

3*(1-2) = -3

8

3 7 8 9 11

X X ✓ X X

SCALER

Contribution Technique

$$Ans = \sum_{\forall i} \text{contribution of } A[i]$$

$$A[i] * \left(\#\text{subarrays } A[i] \text{ is max} - \#\text{subarrays } A[i] \text{ is min} \right)$$

< Question > : In how many subarrays, ith element is the maximum element?

arr[ ] : [1 8 3 5 4 2 11 7 2]

0 1 2 3 4 5 6 7 8

i = 3, j = 1, k = 6

[3 5] [5]

[3 5 4] [5 4]

[3 5 4 2] [5 4 2] 6 ✓

Nearest greater on left — Nearest greater on right

subarrays → (# start) * (# end) = (i - j) * (k - i)

(# start) → [j+1 i] → i - (j+1) + 1 = i - j

(# end) → [i k-1] → k - 1 - i + 1 = k - i

subarrays A[i] is min ?

j → Nearest smaller on left

k → Nearest smaller on right

SCALER

$$Ans = \sum_{\forall i} A[i] * ((i - nearestGreaterLeft[i]) * (nearestGreaterRight[i] - i) - (i - nearestSmallerLeft[i]) * (nearestSmallerRight[i] - i))$$

$TC = O(N + N + N + N + N) = \underline{O(N)}$ $SC = \underline{O(N)}$

H.W → How to solve if duplicates are present.

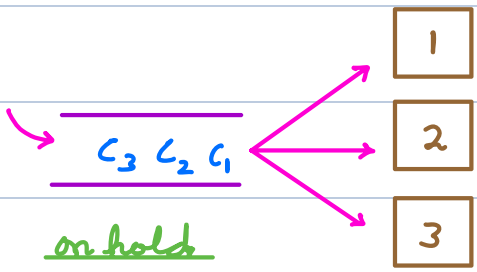
SCALER

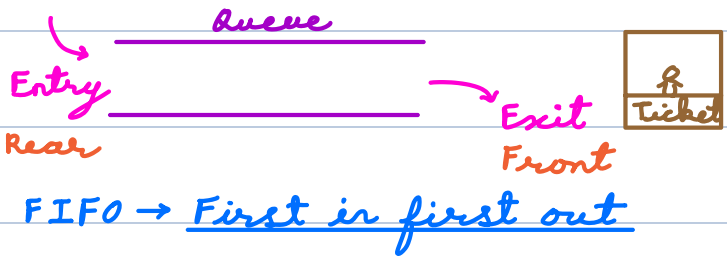
Queues

TABLE OF CONTENTS

SCALER

Queue





FIFO → First in first out

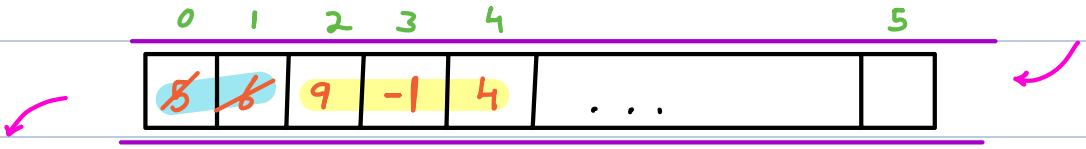
Operations in Queues

1. Enqueue(x) → Insert x at rear end.
2. Dequeue( ) → Remove element from front end.
3. front( ) / rear( ) → Get element of front/rear end.
4. isEmpty( ) → check if the queue is empty.

TC = O(1)

Array Implementation of Queues

Example : Eq(5) , Eq(6) , Eq(9) , Eq(-1) , Dq( ) , Dq( ) , front , Eq(4)



f = 0 → 2

r = -1 → 0 → 2 → 3 → 4

Queue → [f — r]

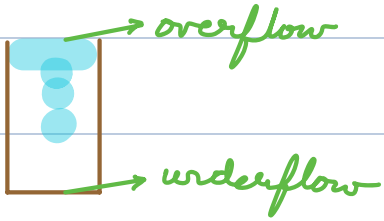
```
void enqueue (x) {
    r++
    A[r] = x
}
```

```
int dequeue () {
    if (isEmpty()) return -1
    f++
    return A[f-1]
}
```

overflow → 1) Limit insertion.

2) Use dynamic array. ✓

```
boolean isEmpty () {
    len = r - f + 1
    return (len == 0)   // f > r
}
```

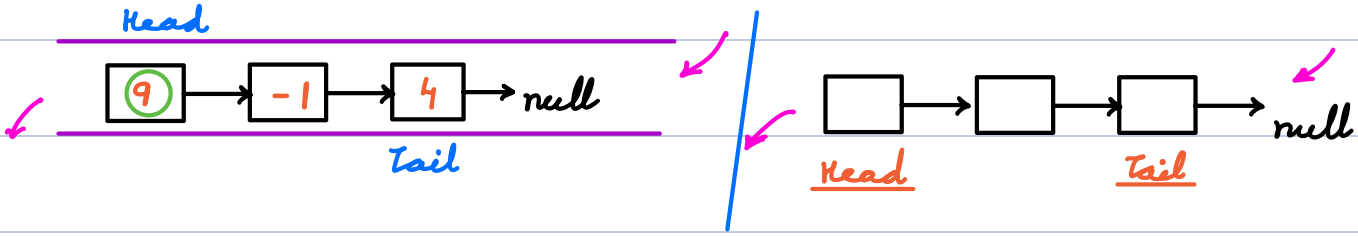


∀ operations, TC = O(1)

SCALER

Implementation of Queue using LinkedList

Example : Eq(5) , Eq(6) , Eq(9) , Eq(-1) , Dq( ) , Dq( ) , front , Eq(4)



`Tail.next = new Node(x)`

`Tail = Tail.next`

`Head = null`

`Tail = null`

1) Enqueue (x) → Insert at tail
2) Dequeue ( ) → Remove head.
3) front ( ) → Head.data
4) rear ( ) → Tail.data
5) isEmpty ( ) → Head == null

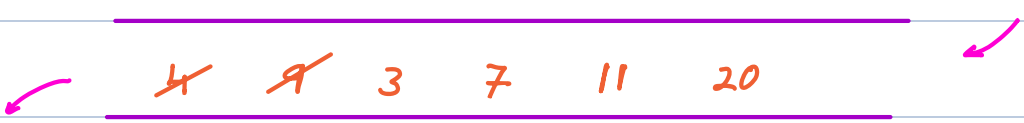
underflow ✓

TC = O(1)

SCALER

What will be the state of the queue after these operations enqueue(3), enqueue(7), enqueue(12), dqueue(), dqueue(), enqueue(8), enqueue(3)

What will be the state of the queue after these operations enqueue(4), dqueue(), enqueue(9), enqueue(3), enqueue(7), enqueue(11), enqueue(20), dqueue()



SCALER

Implementation of Queue using ② Stacks - [Directi]

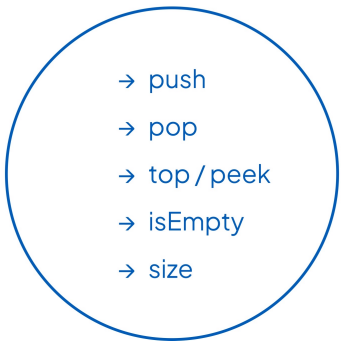
→ Dequeue

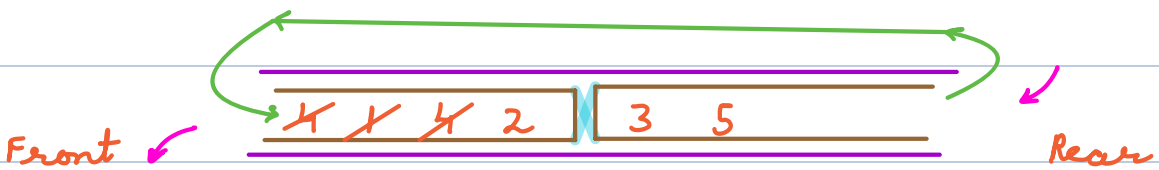
→ Enqueue

→ front( )/rear()

→ isEmpty( )

behind the scene only these functions can be used





st 2 st 1

```
void enqueue (x) {
    st1.push (x)
}
```

✓

```
boolean isEmpty () {
    return st1.isEmpty ()
        && st2.isEmpty ()
}
```

✓

```
void move () {
    while (! st1.isEmpty ()) {
        st2.push (st1.pop ())
    }
}
```

TC = O(K)

SCALER

```
int dequeue () {
    if (isEmpty ()) return -1
    if (st2.isEmpty ()) {
        move ()    // st1 ⟶ st2   K elements
    }
    return st2.pop ()
}
```

⇒ for next K dequeue operations → TC = O(1)

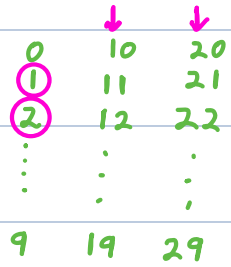
⇒ 2 steps per element.

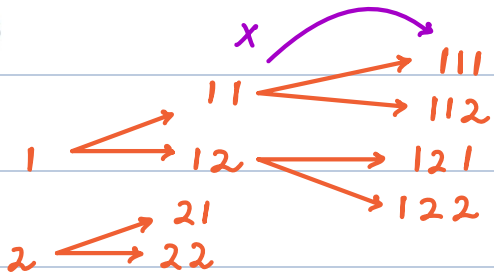
O(2) → O(1)

Perfect Numbers

Q → Find N^{th} number formed using digits 1 & 2.

N →	1	2	3	4	5	6	7	8
	1	2	11	12	21	22	111	112





$10 * x + 1$
$10 * x + 2$

SCALER

x is generated before y

⇒ x will generate next numbers before y.

FIFO

N = 5

1 x 11 12

i = ~~2~~ 5 x = ~~1~~ 2 a = 21 b = 22

```
if (N <= 2) return N
q.enqueue(1) ✓   q.enqueue(2) ✓
i = 3
while (i <= N) {
    x = q.dequeue()
    a = x * 10 + 1 ✓    b = x * 10 + 2 ✓
    if (i == N) return a ←
    if (i+1 == N) return b
    q.enqueue(a)    q.enqueue(b)
    i += 2
}
```

TC = O(N) SC = O(N)

SCALER

Doubly Ended Queue

T.C → O(1)

1. **Insertion at Front (push_front)**: Add an element to the front (head) of the deque. ✓
2. **Insertion at Back (push_back)**: Add an element to the back (tail) of the deque. ✓
3. **Removal from Front (pop_front)**: Remove and return the element from the front of the deque. ✓
4. **Removal from Back (pop_back)**: Remove and return the element from the back of the deque. ✓
5. **Front Element Access (front)**: Get the element at the front of the deque without removing it. ✓
6. **Back Element Access (back)**: Get the element at the back of the deque without removing it. ✓

Stack + Queue → Doubly Ended Queue

Sliding Window Maximum

< *Question* > : Find max of every subarray of size K.

[3 , 2 , 3 , 4 , 5 , 5 , 4 , 5 , 6]

0 1 2 3 4 5 6 7 8

K = 4

$1 \le N \le 10^5$

SCALER

[0 - 3] → 4

[1 - 4] → 5

[2 - 5] → 5

[3 - 6] → 5

[4 - 7] → 5

[5 - 8] → 6

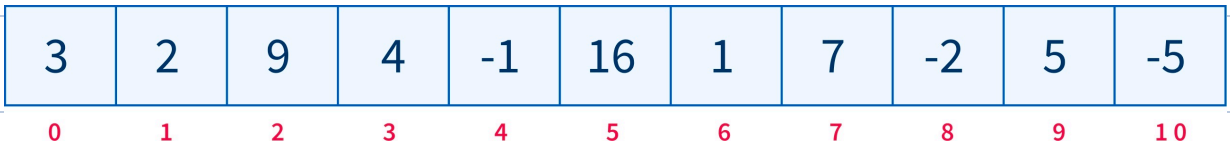
Bruteforce → TC = $O(N^3)$ → $O(N^2)$

SC = $O(1)$

[3, 2, 3, ~~4~~, ~~5~~, 5, 4, 5, 6]

H.W → Sliding Window + Deque.

SCALER



K = 4