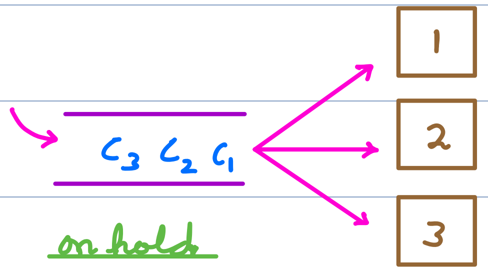
SCALER

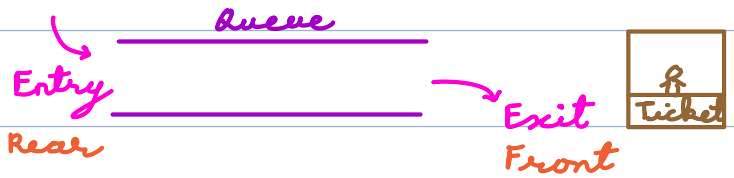
Queues

TABLE OF CONTENTS

Notes

Queue





FIFO → First in first out

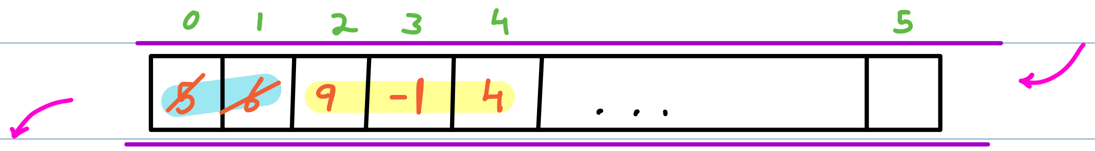
Operations in Queues

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~~ ~~1~~ 2

r = ~~-1~~ ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

Queue → [f —— r]

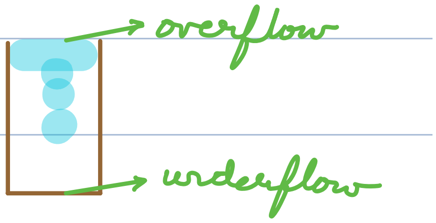
```
void enqueue (x) {
    r++
    A [r] = x
}
```

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

overflow → 1) Limit insertion.

2) Use dynamic array. ✓

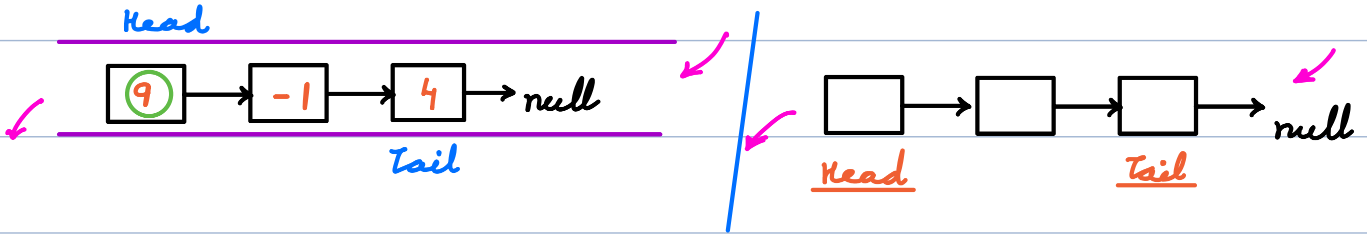
```
boolean isEmpty () {
    len = r - f + 1
    return (len == 0)   // f > r
}
```



∀ operations, TC = O(1)

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.

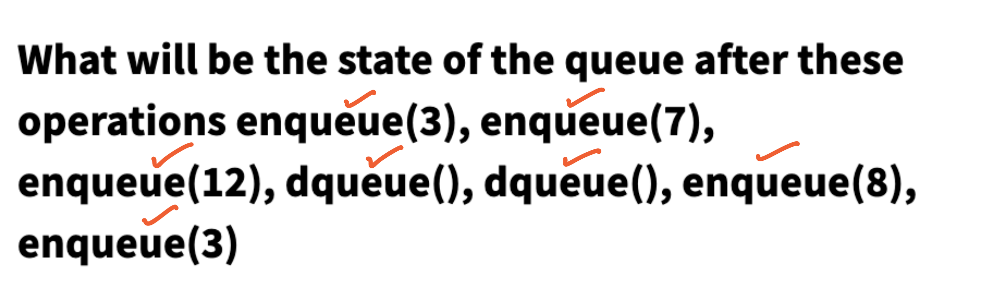
3) front () → Head . data

4) rear () → Tail . data

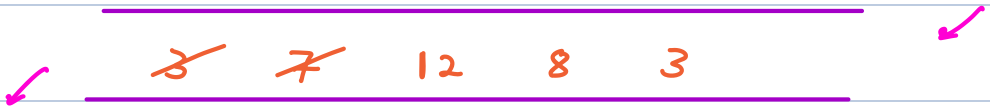
5) isEmpty () → Head == null

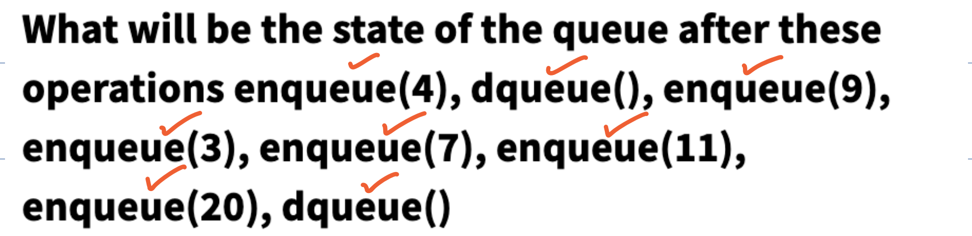
underflow ✓

TC = O(1)

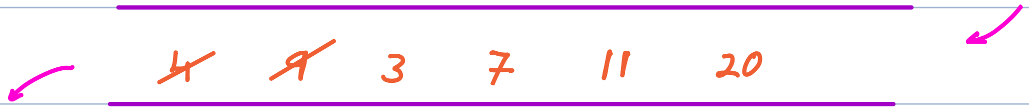


What will be the state of the queue after these operations enqueue(3), enqueue(7), enqueue(12), dqueue(), dqueue(), enqueue(8), enqueue(3)





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()



Implementation of Queue using 2 Stacks - [Directi]

→ Dequeue

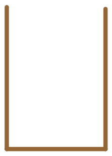
→ Enqueue

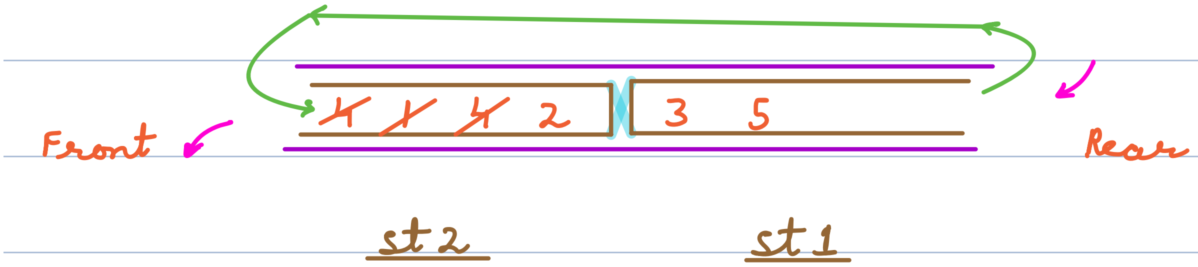
→ front( )/rear()

→ isEmpty( )

behind the scene only these functions can be used

→ push
→ pop
→ top / peek
→ isEmpty
→ size





```
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)
```

```
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.

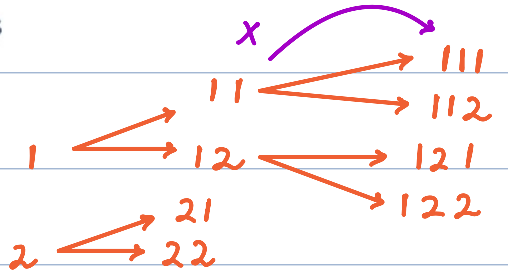
O(2) ⟶ O(1)

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

	↓	↓
0	10	20
1	11	21
2	12	22
⋮	⋮	⋮
9	19	29



$10 * x + 1$

$10 * x + 2$

x is generated before y

⇒ x will generate next numbers before y.

FIFO

N = 5

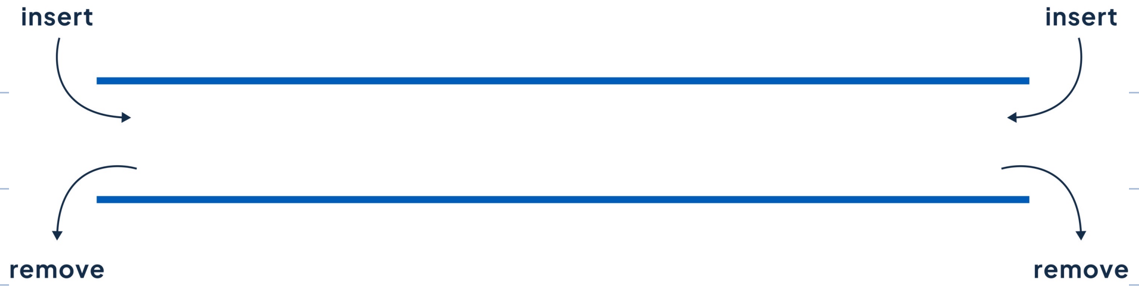
~~1~~ ~~2~~ 11 12

i = ~~3~~ 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)

Doubly Ended Queue



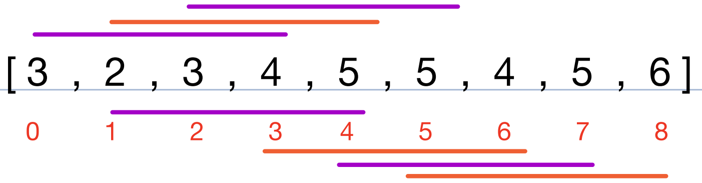
T.C → O(1)

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.



K = 4

$1 \leq N \leq 10^5$

[0 - 3] → 4

[1 - 4] → 5

[2 - 5] → 5

[3 - 6] → 5

[4 - 7] → 5

[5 - 8] → 6

Bruteforce → $TC = O(N^3) \longrightarrow \underline{O(N^2)}$

$SC = \underline{O(1)}$

[3 , 2 , 3 , ~~4~~ 1 , ~~5~~ 1 , 5 , 4 , 5 , 6]

H.W → Sliding Window + Deque .

3	2	9	4	-1	16	1	7	-2	5	-5
0	1	2	3	4	5	6	7	8	9	10

K = 4