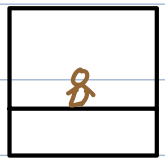
1) 

Exit / Front

Entry / Rear

Ticket Counter Queue

FIFO → First In First Out

2) Printer commands

3) Toll Plaza

4) call center

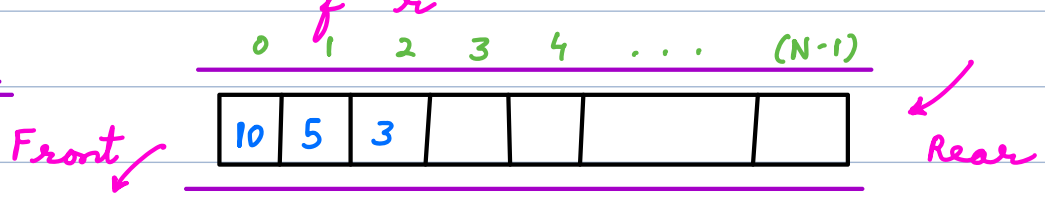
Operations in queue

1) enqueue (x) → insert x at rear end
2) dequeue () → removing element at front end
3) isEmpty () → check if queue is empty
4) front () → get data at front end
5) rear () → get data at rear end

} TC = O(1)

Implementation of Queues

1) Arrays



Queue → f — r

```
enqueue (10)
enqueue (5)
enqueue (3)
dequeue ()
isEmpty ()
```

initially, r = -1 f = 0

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

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

```
boolean isEmpty () {
    return f > r
}
```

Underflow → Empty DS

Overflow → completely full DS

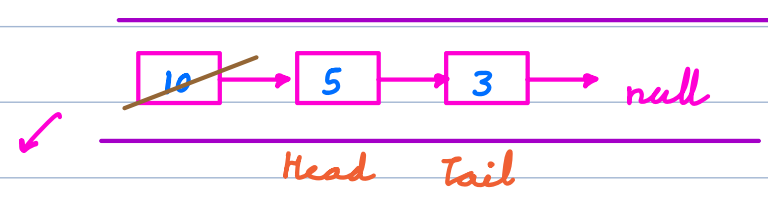
→ Do not allow more elements to insert.

→ Use dynamic array ✓

---

2) Using linked list (effective use of memory wrt arrays)

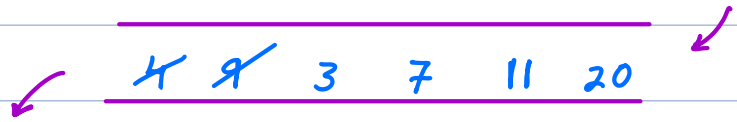
```
enqueue (10)
enqueue (5)
enqueue (3)
dequeue ()
isEmpty ()
```



initially, Head = Tail = null

1) enqueue(x) → insert at tail node
2) dequeue() → delete head
3) isEmpty() → return (Head == null)
4) front() → return Head.data
5) rear() → return Tail.data

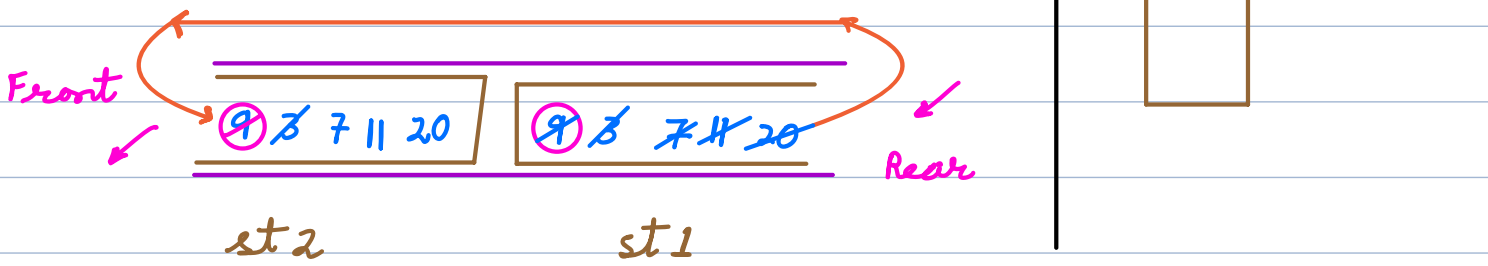
→ check underflow case



enqueue (4) ✓ dequeue () ✓ enqueue (9) ✓
enqueue (3) ✓ enqueue (7) ✓ enqueue (11) ✓
enqueue (20) ✓ dequeue () ✓

Q → Implement queue using **2 stacks**.

push (x) isEmpty ()
pop () peek () / top ()



enqueue (4) ✓ dequeue () ✓ enqueue (9) ✓
enqueue (3) ✓ enqueue (7) ✓ enqueue (11) ✓
enqueue (20) ✓ dequeue () ← 5 extra steps
dequeue () * 5 in TC = O(1)

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

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

```
int dequeue() {
    if (isEmpty())
        return -1
    if (st2.isEmpty())
        move()
    return st2.pop()
}
```

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

move K elements → TC = O(K)

dequeue K items → TC = O(K)

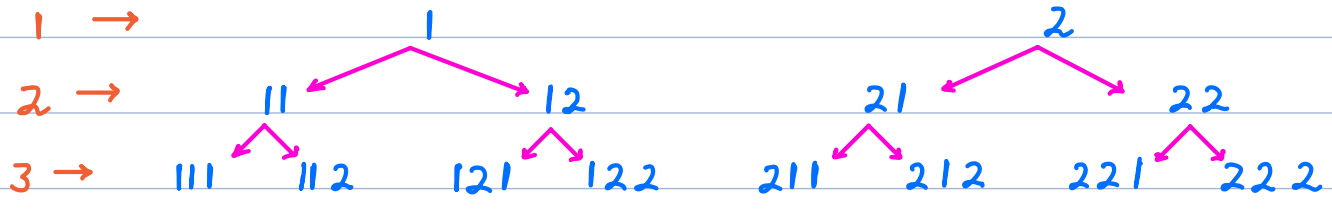
removal → (move + dequeue)

⇒ TC = O(2) = O(1)

Q → Given an integer N, find N^{th} number formed by only digits 1 & 2. (Perfect number)

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

digits



N = 10

Queue

~~1~~ ~~2~~ ~~11~~ ~~12~~ 21 22 111 112 121 122

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

```
res = ""
N--
while (N >= 0) {
    d = (N % 2) + 1
    res = d + res   // append
    N = N/2 - 1
}
```

TC = O(log(N)) SC = O(1)

Doubly Ended Queue

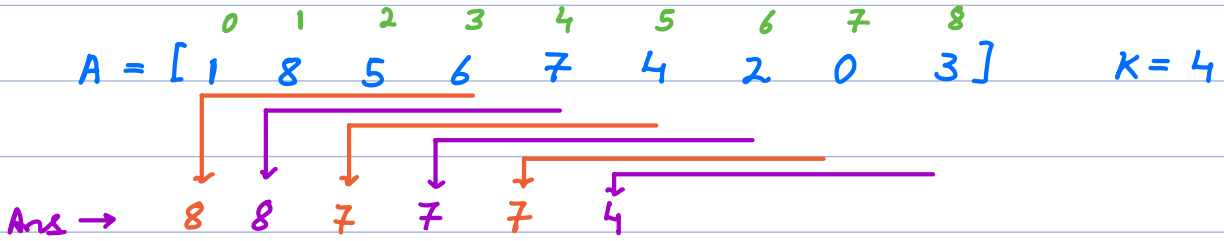
Operations

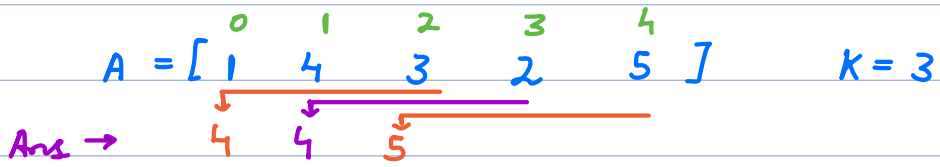
1) enqueueFront(x)
2) enqueueRear(x)
3) dequeueFront()
4) dequeueRear()
5) isEmpty()
6) front()
7) rear()

DS suited to implement doubly ended queue → doubly linked list

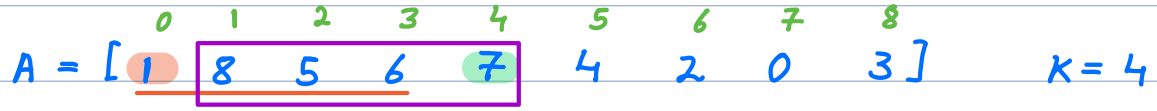
HW → implement

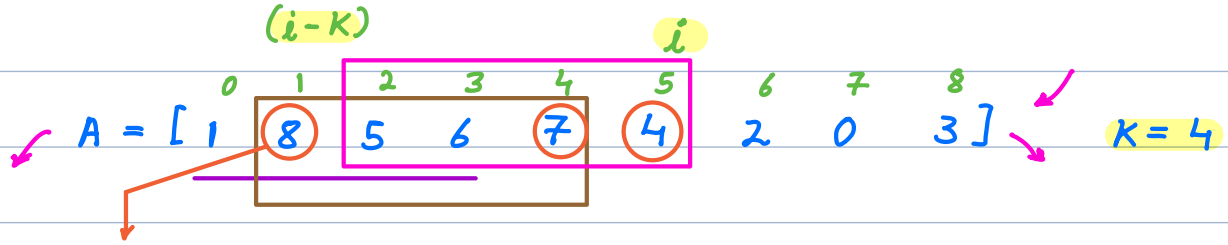
Q → Given an integer array & an integer K.
Find max element of subarrays of size K.





Fixed size subarray ⇒ sliding window





✓ outside window ⇒ remove

✓ if greater on right ⇒ current cannot be answer.

```
for i → 0 to (K-1) {
    while ( ! q.isEmpty && A[q.rear()] < A[i])
        q.dequeueRear()
    }
    q.enqueueRear(i)
}
```

```
print ( A[q.front()] )
for i → K to (N-1) {
    while ( !q.isEmpty && A[q.rear()] < A[i])
        q.dequeueRear()
    }
    q.enqueueRear(i)
    if (q.front() == i-K)
        q.dequeueFront()
    print ( A[q.front()] )
}
```

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