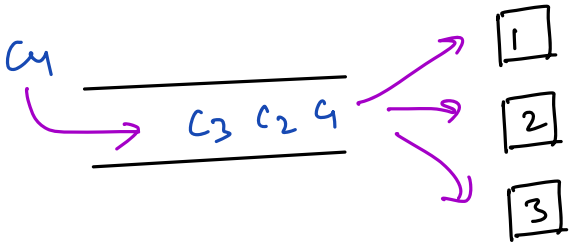
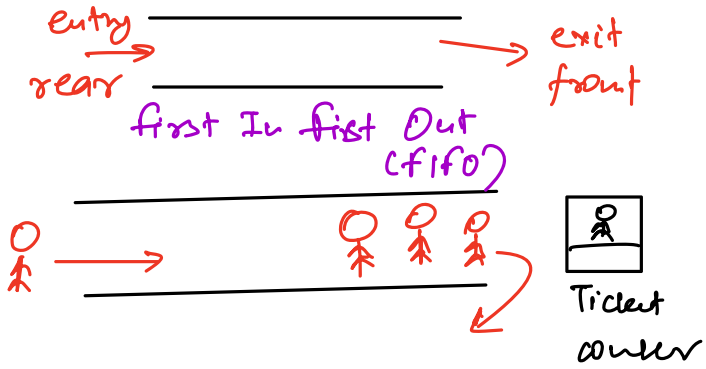
Queues: Implementation & Problems

Customer Care





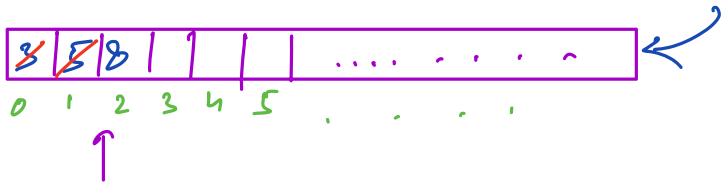
Operations

1. Enqueue (x) → Insert x from rear end
2. Dequeue () → Remove data from front end
3. isEmpty () → check if queue is empty
4. Front () → Get the data at front end
5. Rear () → Get the data at rear end

} TC = O(1)

Ques → Implement queue using array.

enqueue(3) ✓
enqueue(5) ✓
enqueue(8) ✓
dequeue() → 3
isEmpty() → false
front() → 5
dequeue() → 5
rear() → 8



f, r Queue → from index f to r (subarray)

f = 0 X 2
r = -1 0 1 2

```
void enqueue(x) {
    r++                          // Overflow → 1. use dynamic array
    A[r] = x                     //            2. Do not insert more than size.
}
```

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

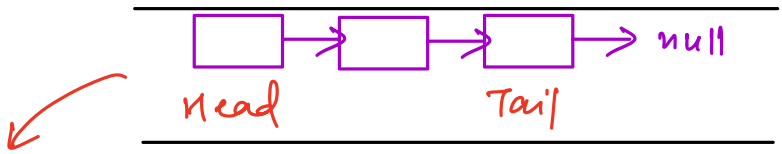
```
int front() {
    if (isEmpty()) return -1
    return A[f]
}
```

```
bool isEmpty() {
    return f > r ;
}
```

TC = O(1) # operations

```
int rear() {
    if (isEmpty()) return -1
    return A[r]
}
```

Ques → Implement Queue using linked list



1. enqueue(x) →
   insert at tail

2. dequeue() →
   remove from head

TC	Insertion	Removal
Head	O(1)	O(1) ✓
Tail	O(1) ✓	O(N)

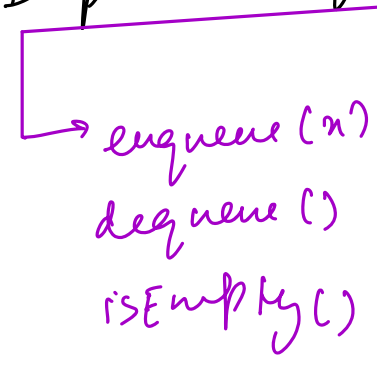
3. isEmpty() → (Head == null)

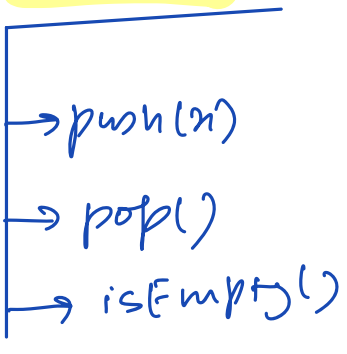
4. front() → Head.data

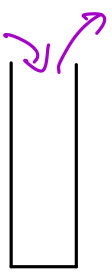
5. rear() → tail.data

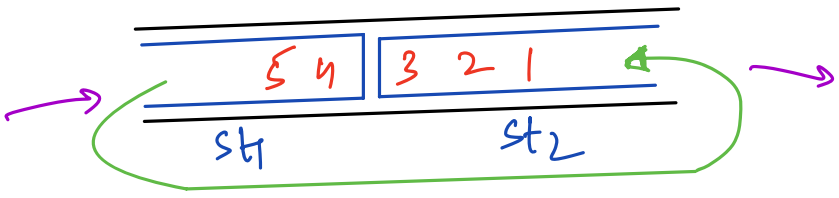
TC = O(1) ∀ operations

Ques → Implement queue using 2 stacks.









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

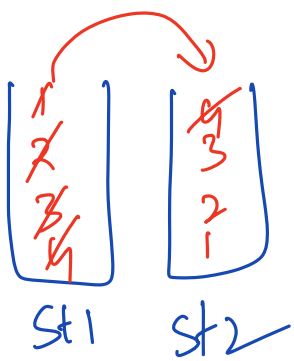
enqueue(1)
enqueue(2)
enqueue(3)
dequeue() → 1

```
void move(){ // TC = O(st1.size())
    while(!st1.isEmpty())
        st2.push(st1.pop())
}
```

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

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

If TC of move() = O(K) ⇒ next K dequeue() will
move TC = O(1)
TC = O(2) = O(1)



dequeue() → 4 (N)
dequeue() → 1
dequeue() → 1
dequeue() → 1

⇒ 7/4 ≈ 2

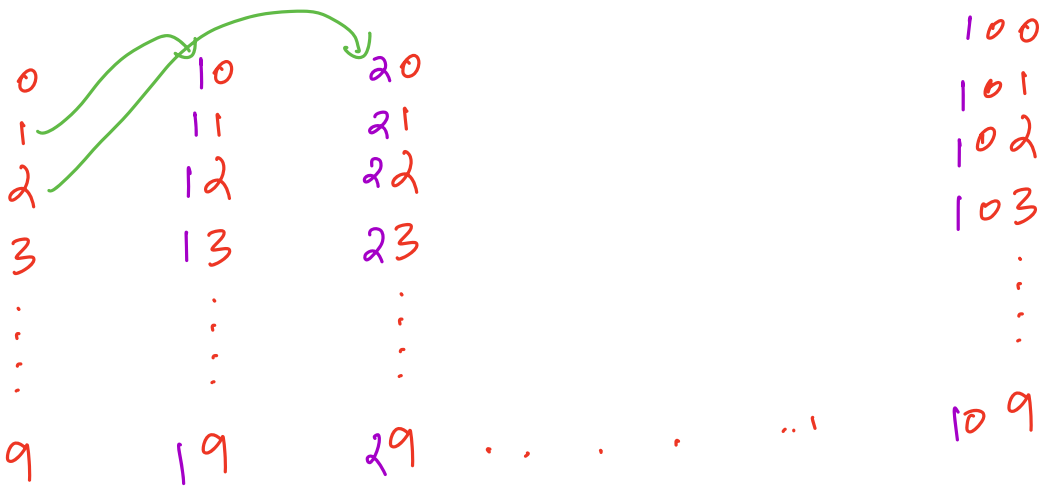
⇒ N + 1 × (N-1) / N

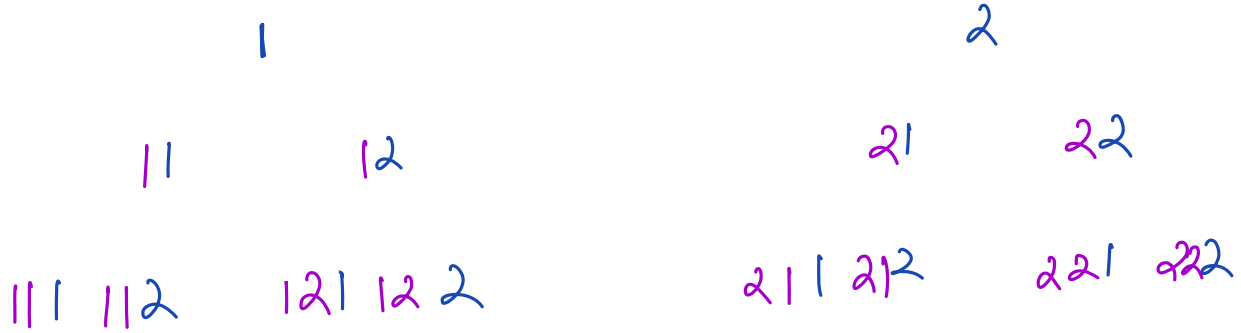
$\approx \frac{2N-1}{N} \approx 2$

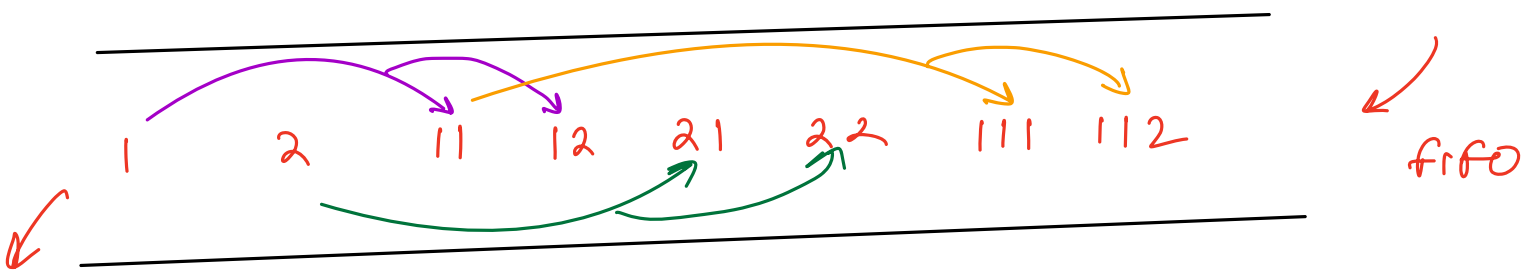
Ques → Given an integer N, find N^{th} number that can be formed by digits 1 & 2 only.

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

N = 1 2 3 4 5 6 7 8 9 10







x → x*10+1

↳ x*10+2

```
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)

1 2 11 12 21 22 111 112

N = 10

i = 3 5 7 9

x = 1 2 11 12

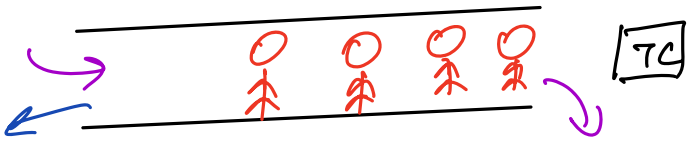
a = 11 21 111 121

b = 12 22 112 122

HW → find Nth number using only prime digits?

2, 3, 5 & 7

Doubly Ended Queue



Support UNDO also

Queue + Stack

1. enqueue front(x)
2. enqueue Rear(x)
3. dequeue front()
4. dequeue Rear()
5. isEmpty()

Implement Deque → doubly linked list

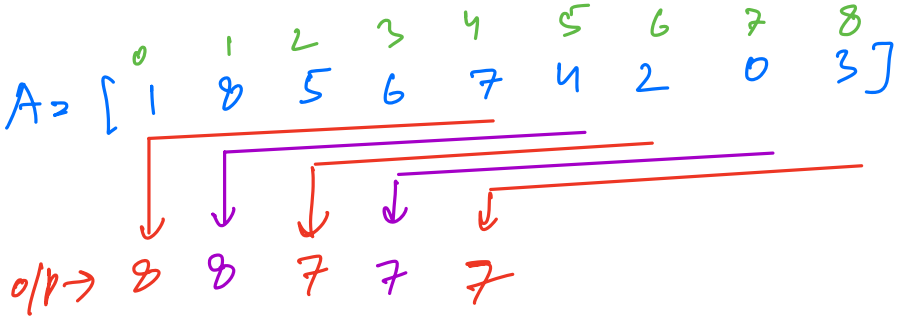
TC = O(1) ∀ operations

Ques → Given an integer array & an integer K.

find the max element of subarray of size K.

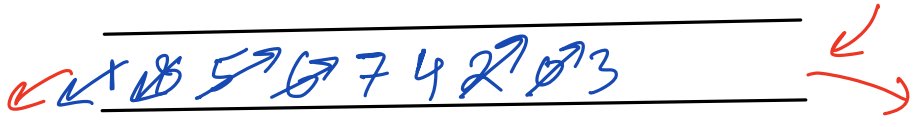
sliding window

A = [1 8 5 6 7 4 2 0 3] K=5



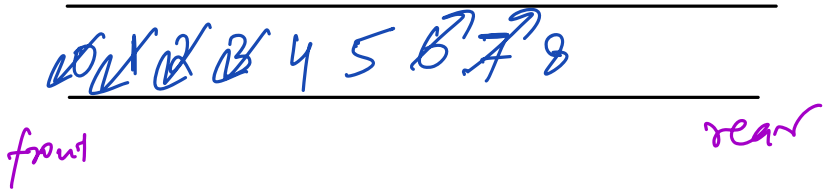
o/p → 8 8 7 7 7

A = [1 8 5 6 7 4 2 0 3] K=5



8 8 7 7 7

store ~~data~~ → index of data



Sliding window + dequeue

```
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) { // out of window
        q.dequeueFront()
    }
```

TC = O(N)

SC = O(K)

```
    print ( a [ q.front() ] )
}
```