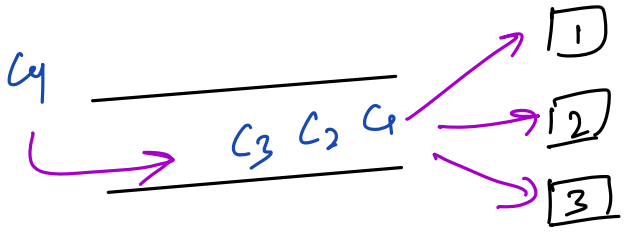
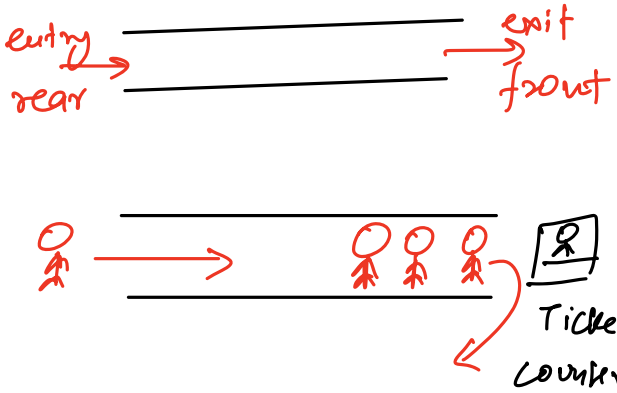
Queues : Implementation & Problems

Customer care





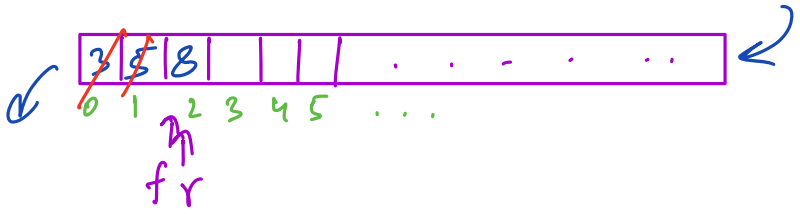
Operations

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 of front end
5. Rear() → Get the data of 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



Queue → from index f to r (subarray)

f = 0
r = -1 0 1 2

[-1, -1]
↓ insert
[-1, 0]

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

// Overflow → 1. Use dynamic array
2. Do not insert more than size

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

garbage

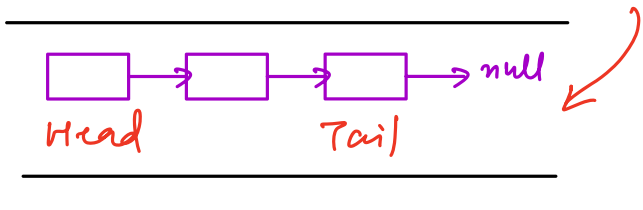
```
int front() {
    if(isEmpty()) return -1
    return A[f]
}
```

```
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

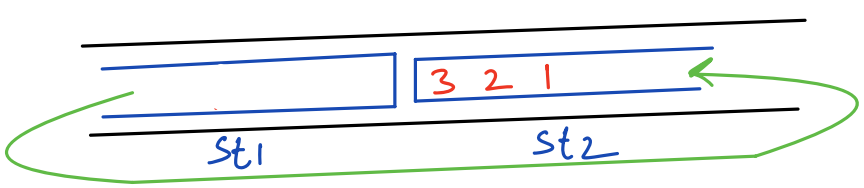
5. rear() → tail.data

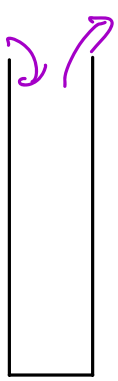
TC = O(1) ∀ operations

Ques → Implement queue using 2 stacks

→ enqueue(x)
dequeue()
isEmpty()

→ push(x)
→ pop()
→ isEmpty()





```
void enqueue(x) {
    st1.push()
3
```

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

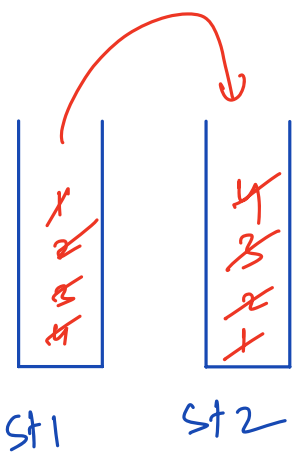
```
void move() {  // TC = O(st1.size())
    while(!st1.isEmpty())
        st2.push(st1.pop())
}
```

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

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

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

TC = O(2) = O(1)



dequeue() → O(4) + O(1)

dequeue() → O(1)

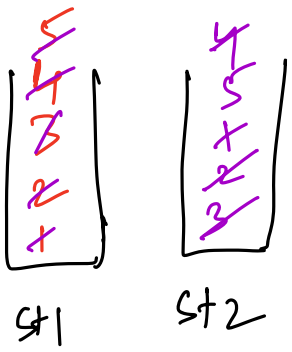
dequeue() → O(1)

dequeue() → O(1)

⇒ $\frac{8}{4} \approx 2$

1e, 2e, 3e, de, 4e, de, de, 5e, de

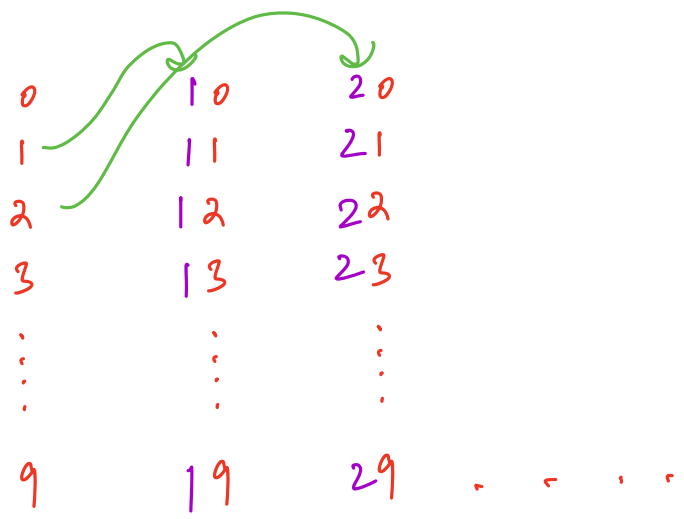
↓1 ↓2 ↓3 ↓4

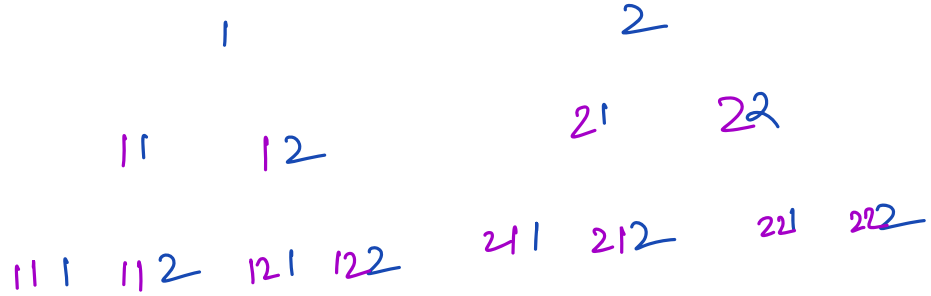


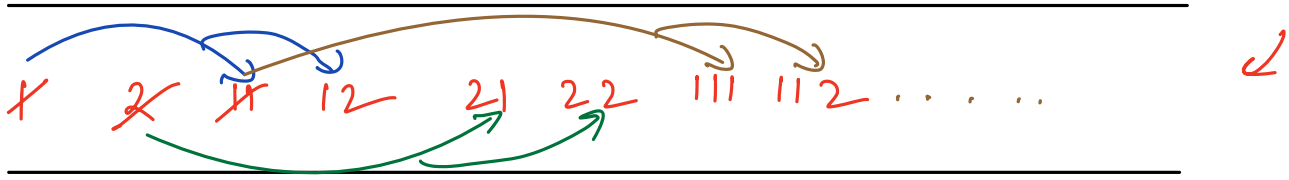
Ques → Given an integer N, find N^{th} number that can be formed by digits 1 & 2 only.

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

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







$x \rightarrow x*10+1$

$\hookrightarrow x*10+2$

```
if ( N<=2)  return N
q.enqueue (1)
q.enqueue (2)
i=3
while ( i<=N) {
    x = q.dequeue ()
    a = 10*x +1
    b = 10*x + 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 x=1 2 11 12

i=3 a=11 21 111 121

=5 b=12 22 112 122

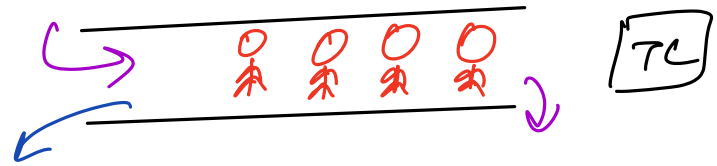
7

9

HW → find N^{th} number using only prime digits?

2, 3, 5, 7

Doubly Ended Queue



support UNDO also

Queue + Stack

1. enqueuefront(x) rear
2. enqueue Rear (x)
3. dequeuefront( )
4. dequeue Rear( )
5. isEmpty ( )

Implement Deque → doubly linked list

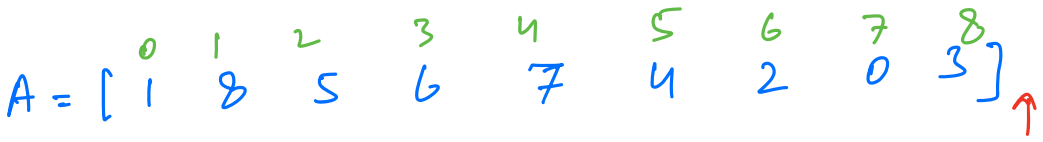
TC = O(1) ∀ operations

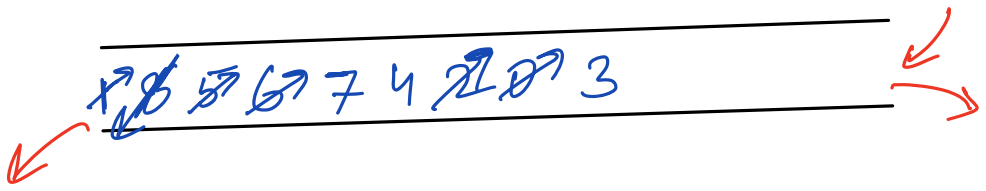
Qus → Given an integer array & an integer K.

find the max element ∀ subarray of size K. → sliding window

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

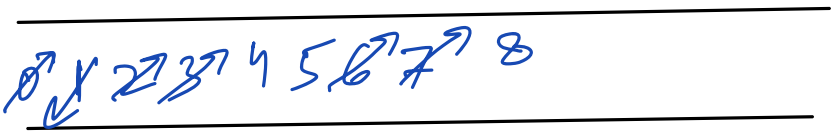
o/p → 8 8 7 7 7





8 8 7 7 7

store ~~data~~ index of data



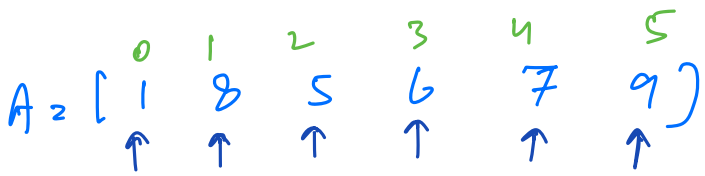
sliding window + deque

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

TC = O(N)

SC = O(K)



0 1 2 3 4 5

8 9