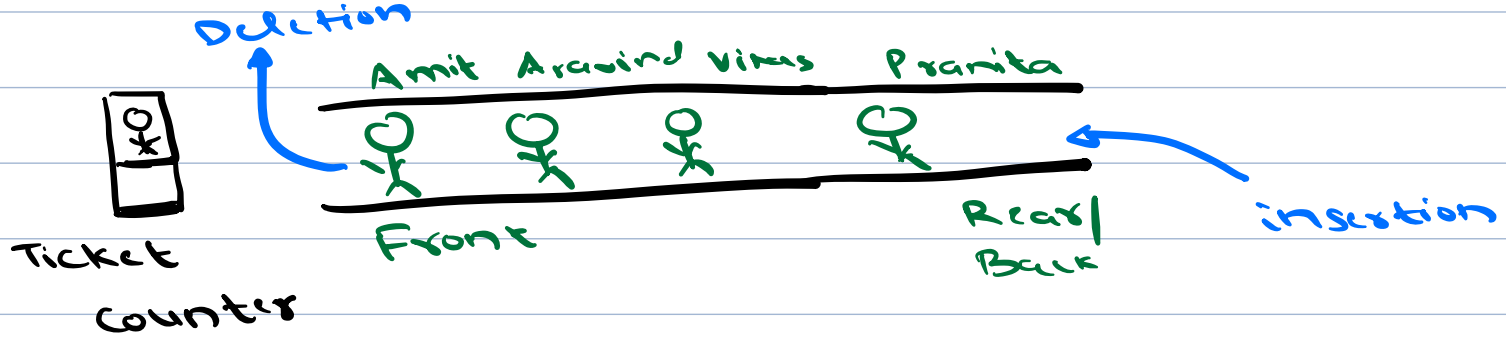
Agenda

1. Queue
2. Implementation of Queue using array & LL
3. Implementation of Stack
4. Perfect Number Question
5. Double Ended Queue (Deque)
6. Sliding Window Maximum

Queue

1. Linear Data Structure where items are added at one end and removed from the other end.
2. First In First Out Principle
   Item that has been inserted first in queue will be first one to removed.



FIFO → First In First out

Common Operations in Queue:

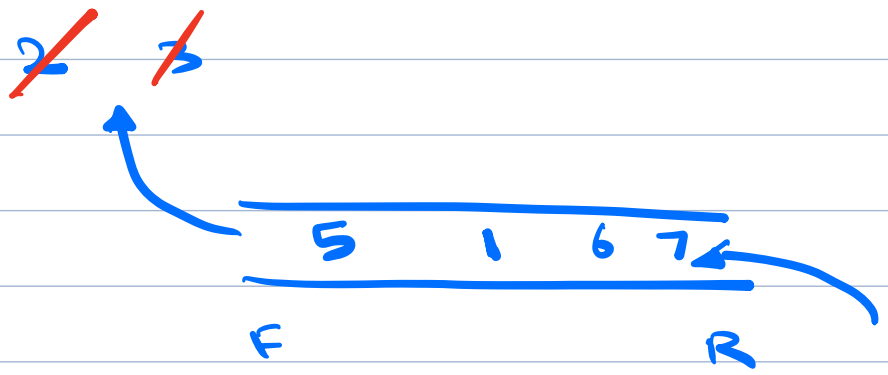
1. Enqueue: Adds element to back/rear of queue. Also called 'push' in some languages.

2. Deque: Removes element from front of queue. Also called 'pop' in some languages.

3. Peek or Front: Returns front element of queue (without removing).

4. isEmpty: Checks if queue is empty.

5. Size/length: Returns no. of elements in queue.

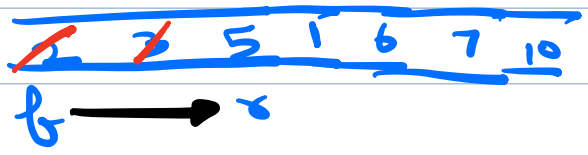


Enqueue(2) ✓
Enqueue(3) ✓
Enqueue(5) ✓
Deque() ✓
Front() ✓ 3
Enqueue(1) ✓
Deque() ✓
Front() ✓ 5
Enqueue(6) ✓
Enqueue(7) ✓

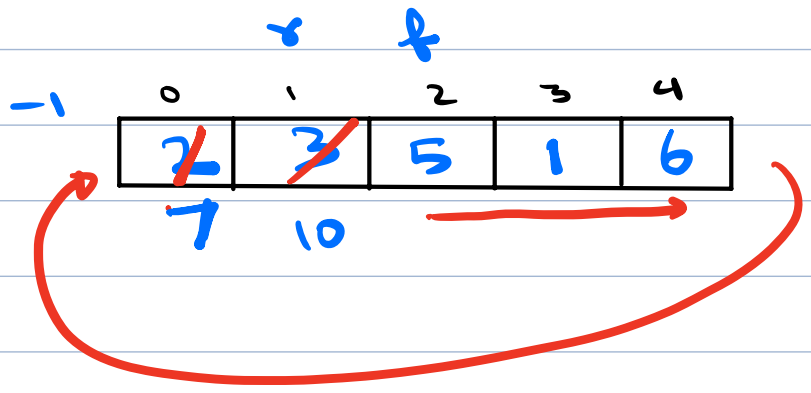
TC of every operation:
O(1)

Implementation of Queue

Queue



1. Using fixed size array



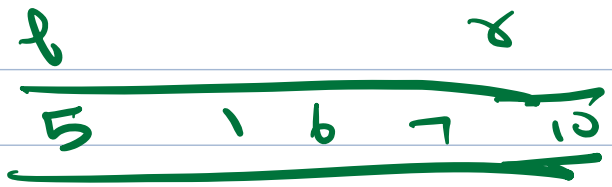
Enqueue(2) ✓
Enqueue(3) ✓
Enqueue(5 ✓
Dequeue() ✓
Front() ✓ 3
Enqueue(1) ✓
Dequeue() ✓
Front() ✓ 5
Enqueue (6) ✓

enqueue() {
  r++
  A[r] = new de
}

dequeue() {
  f++
}

$r = (r+1) \% capacity$

$r = (r+1) \% 5$



Enqueue (7)
Enqueue (10)
Enqueue (12)

↓
overflow

Note → Solving the issue of space wasted by dequeue() → circular array
But it cannot solve the capacity problem.
overflow

Dynamic array:

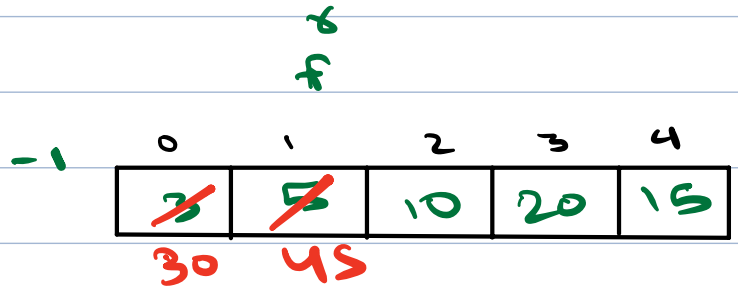
Circular Fixed Length Queue

```
class Queue {
    int [] ar
    int size, f, r, capacity   → max no. of elements a queue can hold

    Queue ( limit ) {
        capacity = limit
        ar = new Array (limit)
        size = 0,  r = -1 , f = 0
    }

    void enqueue (int ele) {
        if (size == capacity)
            throw overflow error
        r = (r + 1) % capacity
        ar [r] = ele
```

```
    size++
}

void dequeue() {
    if (size == 0)
        throw underflow error
    f = (f+1) % capacity
    size--
}

bool isEmpty() {
    return size == 0
}

int peek / front() {
    if (size == 0)
        throw underflow error
    return ar[f]
}

bool isFull() {
    return size == capacity
}
```

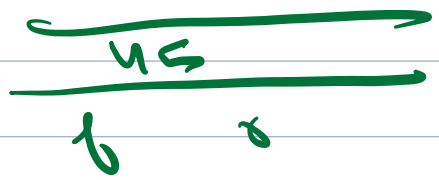
Print Queue: If (size != 0)

```
for (i = f; i <= r; i = (i+1) % capacity)
        print (ar[i])
```



Cap = 5
size = 3
f = 0
r = -1

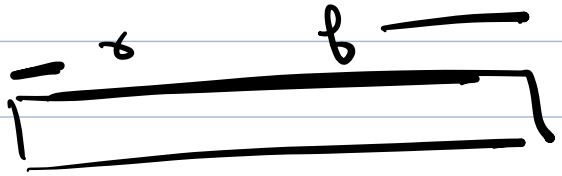
E(3)
E(5)
E(10)
F() → 3
E(20)
E(15)
D()
D()
E(30)
D()
D()
D()
D()
E(45)



① If f ≤ r

Queue → f ⟶ r

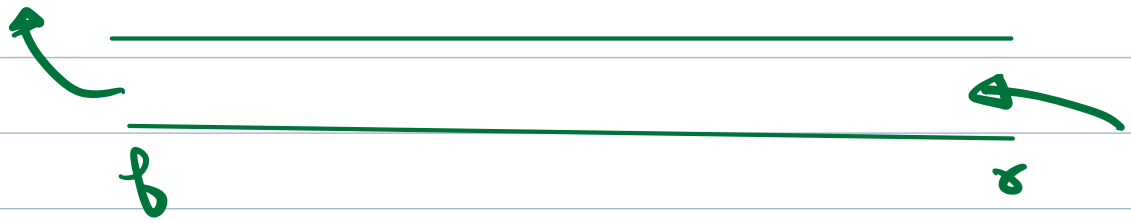
② If f > r



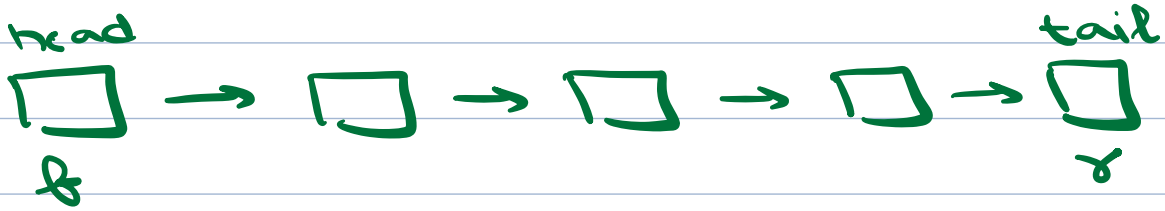
Queue → (f till last) + (0 till r)

TC of every operation is O(1)

Implementation of queue using LL



(1)



Deletion at f (head) → O(1)

Insertion at r (tail) → O(1)

provided we maintain tail

(2)

Deletion at f (tail) → O(n)

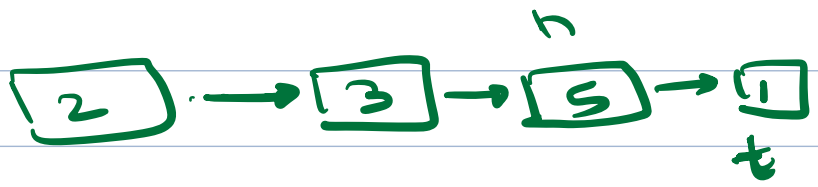
Insertion at r (head) → O(1)

Approach 1 is better as per TC

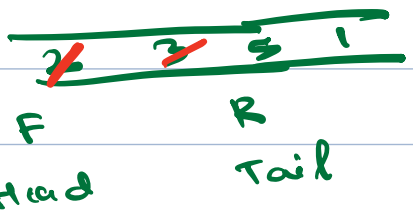
head → f , tail → r

Enqueue(2) ✓
Enqueue(3) ✓
Enqueue(5 ✓
Dequeue() ✓
Front() ✓ 3
Enqueue(1) ✓
Dequeue() ✓
Front() ✓ 5

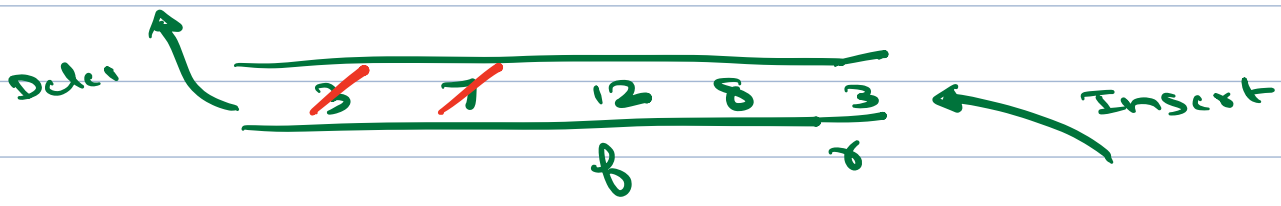
h = t = NULL



t.next = newNode
t = newNode

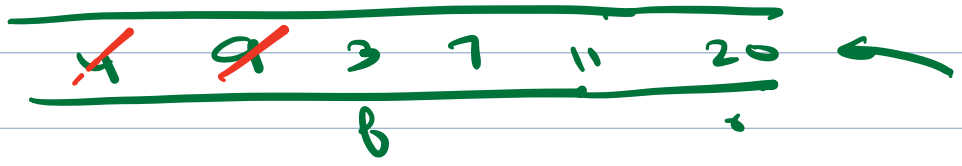


Quiz 1: E(3), E(7), E(12), D(), D(), E(8), E(3)



12 8 3

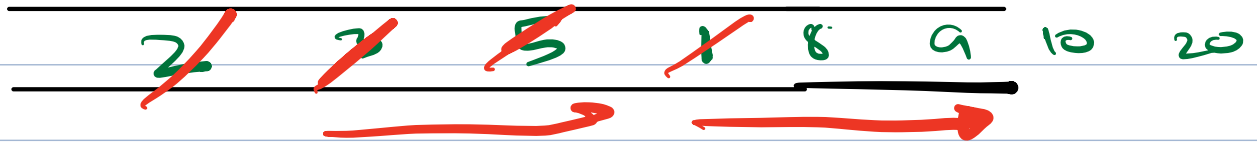
Quiz 2: E(4), D(), E(9), E(3), E(7), E(11), E(20), D()



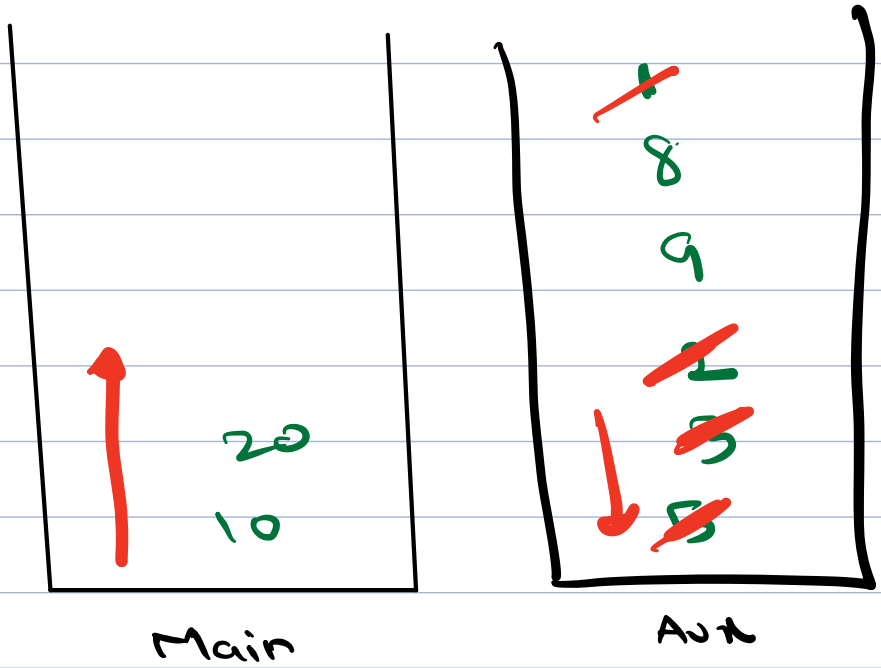
3 7 11 20

10:20

Implimentation of queue using stack

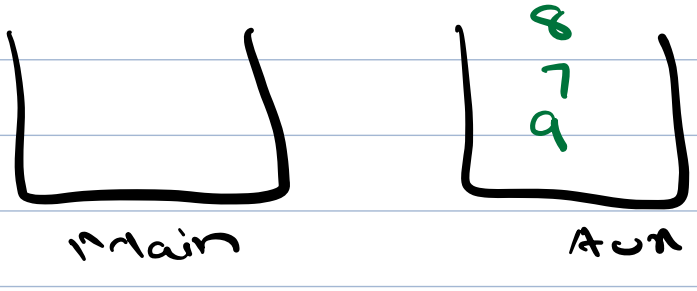


Enqueue(2) ✓
Enqueue(3) ✓
Enqueue(5) ✓
Dequeue() ✓
Front() ✓ 3
Enqueue(1) ✓
Enqueue(8) ✓
Enqueue(9) ✓
Dequeue() ✓
Dequeue() ✓
Dequeue() ✓
Enqueue(10) ✓
Enqueue(20) ✓



Queue → Aux (T → B) + Main (B → T)

E(8)
E(7)
E(9)
f() → 8



Top of aux stack

```
stack <int> main, aux

void enqueue (int el) {
    main.push (el)
}

void dequeue () {
    if (isEmpty()) return UNDERFLOW
    if (aux.isEmpty()) {
        while (!main.isEmpty()) {
            int x = main.top()
            main.pop()      aux.push(x)
        }
    }
    aux.pop()
}

bool isEmpty() {
    return main.isEmpty() &&
           aux.isEmpty()
}
```

```
int front() {
    if (isEmpty()) return UNDERFLOW
    if (aux.isEmpty()) {
        while (!main.isEmpty()) {
            int x = main.top()
            main.pop()      aux.push(x)
        }
    }
    return aux.top()
}
```

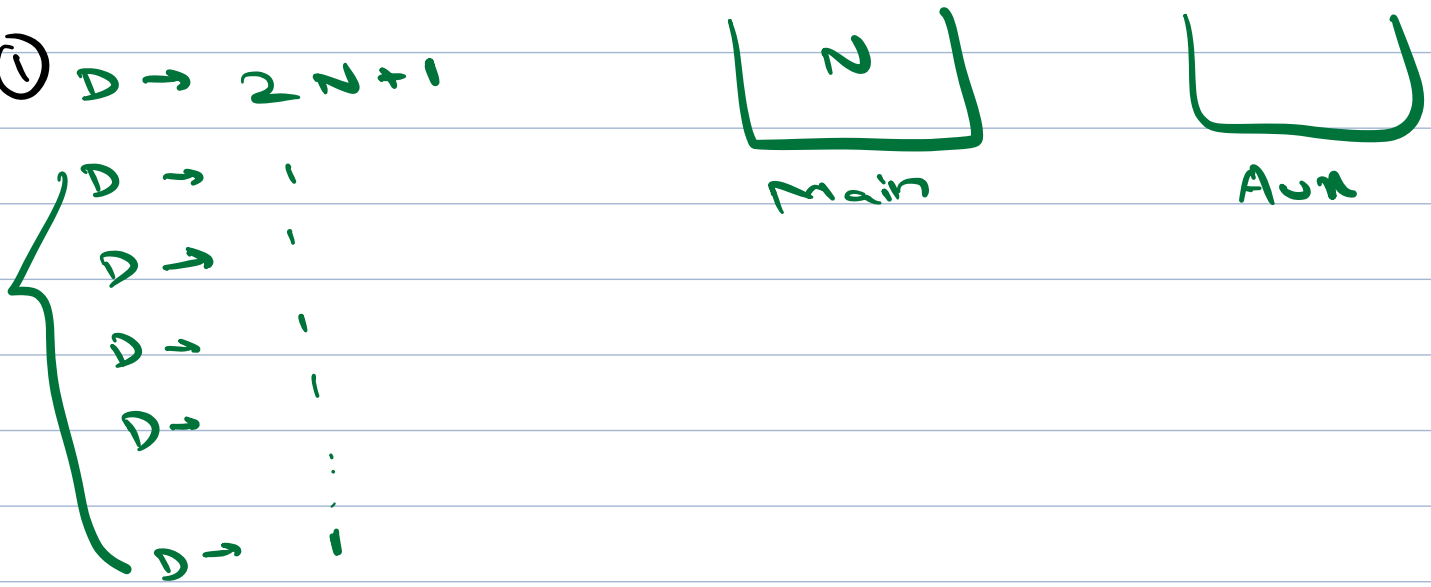
E(7) → 1 opr
E(8) → 1 opr
E(6) → 1 opr

D() → 7 opr
D() → 1 opr
D() → 1 opr

3 D → 9 opr 1 D → 3 opr

On avg. TC of dequeue() / front() → O(1)
Amortized TC → O(1)

① $D \to 2N+1$



N deletions → $2N+1 + N-1$

$= 3N$ opr

1 deletion → $\frac{3N}{N} = 3$ opr → O(1)

Q1. Perfect numbers are numbers with only digit as 1 and 2. Given N, return N^{th} perfect no.

1	2	3	4	5	6	7	8	9	
1	2	11	12	21	22	111	112	121	....

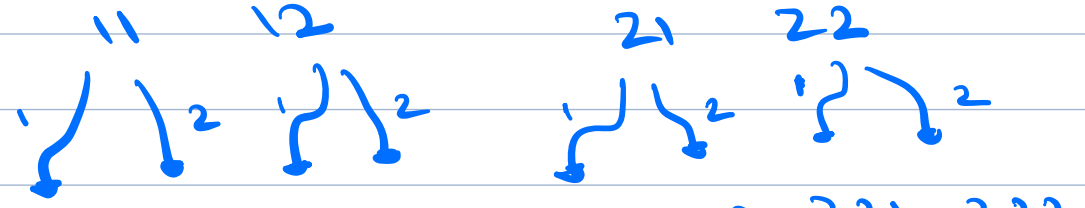
N = 6 ans = 22

Level Order Traversal

BFS (Breadth First Search)

1 digit

2. digit



3 digit

111 112 121 122 211 212 221 222

N = 10

cnt = 2

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

x = 1 a = 11 cnt = 3 b = 12 cnt = 4

x = 2 a = 21 cnt = 5 b = 22 cnt = 6

x = 11 a = 111 cnt = 7 b = 112 cnt = 8

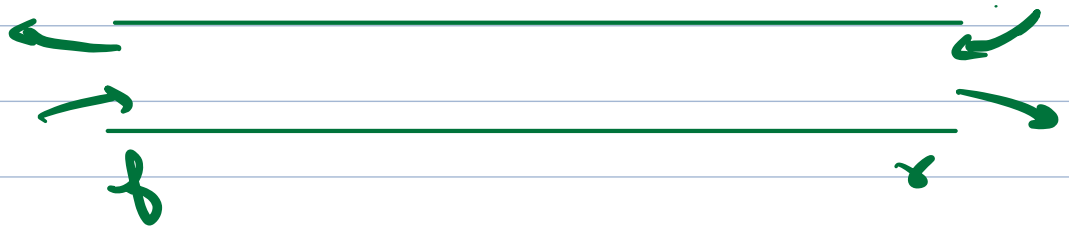
x = 12 a = 121 9 b = 122 10

```
String NthperfectNo (int N) {
    if (N == 1)  return "1"
    if (N == 2)  return "2"
    Queue <String> q
    q.enqueue ("1")
    q.enqueue ("2")
    int cnt = 2
    while (cnt < N) {
        String num = q.front()
        q.dequeue()
        String a = num + "1"
        cnt++      q.enqueue(a)
        if (cnt == N)
            return a
        String b = num + "2"
        cnt++      q.enqueue(b)
        if (cnt == N)
            return b
    }
}
```

TC : O(N)

SC : O(N)

Double Ended Queue (Deque)



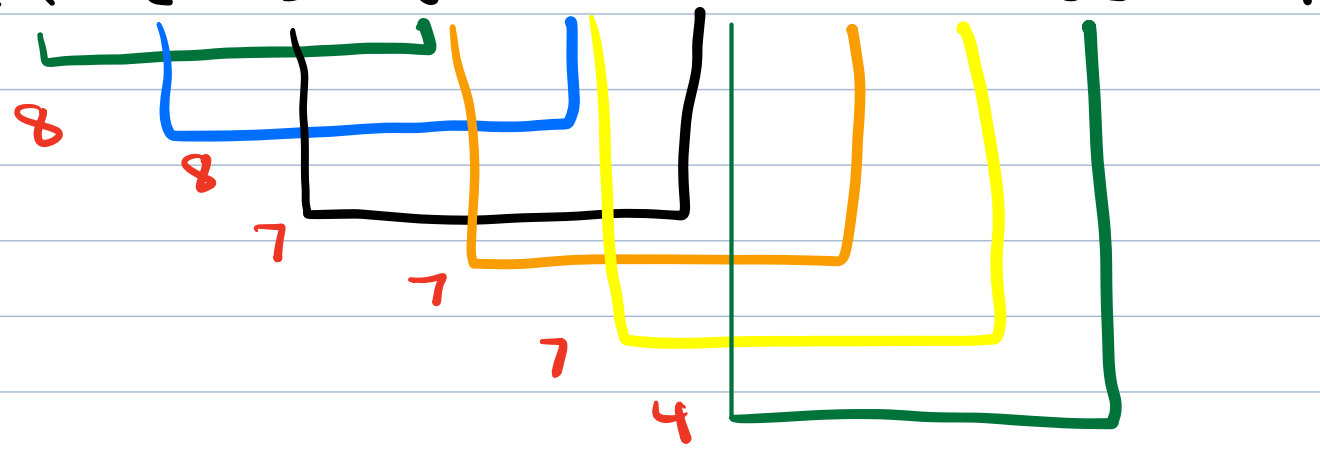
① Insertion and deletion at both ends
② Implemented using a doubly LL

push_front()
push_back()
pop_front()
pop_back()
front()
back()

TC : O(1)

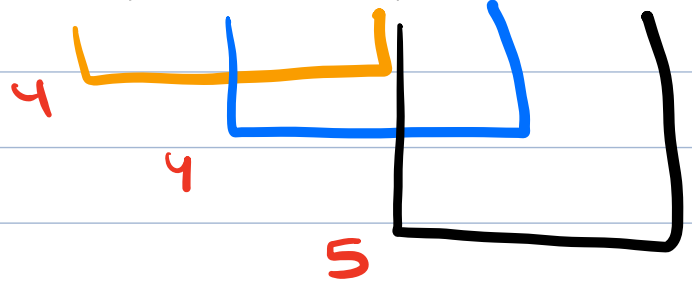
Q2. Given integer array A, and size K. Find max element for each window of size K.

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



ans = [8, 8, 7, 7, 7, 4]

A = [1, 4, 3, 2, 5] K = 3



ans → [4, 4, 5]

BF : For every subarray of size K, iterate and get max

TC : O ((N-K+1) * K) K = N/2

↓

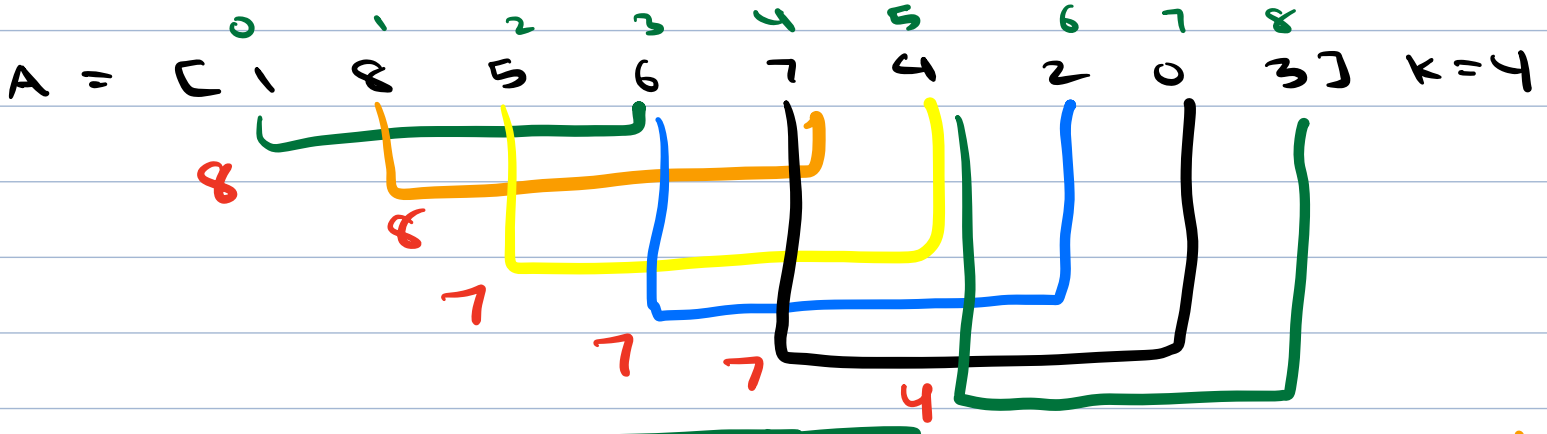
O (N²)

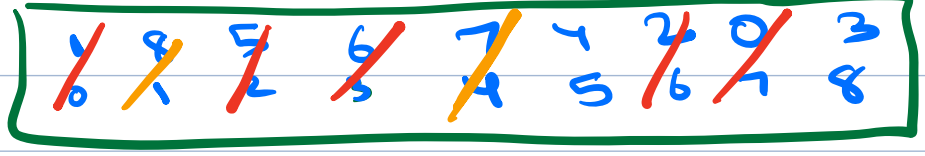
SC : O(1)

Optimized : Size of subarray is fixed

↓

sliding window





— → outgoing de
— → smaller de

Good candidates → elements who can be max de at some time

① Max → Front

② Queue → Decreasing order (f → r)

③ Insert → Del all the de smaller from back

```
list <int> maxInWindow (int A[], int N,
                        int k) {

    list <int> ans
    deque <int> q

for (i=0 ; i<k ; i++) {
    while (!q.empty() && A[q.back()]<A[i])
                q.pop_back()

    q.push_back(i)
}

ans.add(A[q.front()])
```

0 → k-1
1 → k

```
i = 1        j = k
while ( j < n ) {
    // add (i-1)th ele
    if (q.front() == i-1)
        q.pop_front()

    while (!q.empty() && A[q.back()] < A[j])
        q.pop_back()

    q.push_back(j)
    ans.add(A[q.front()])
    i++    j++
}
return ans
```

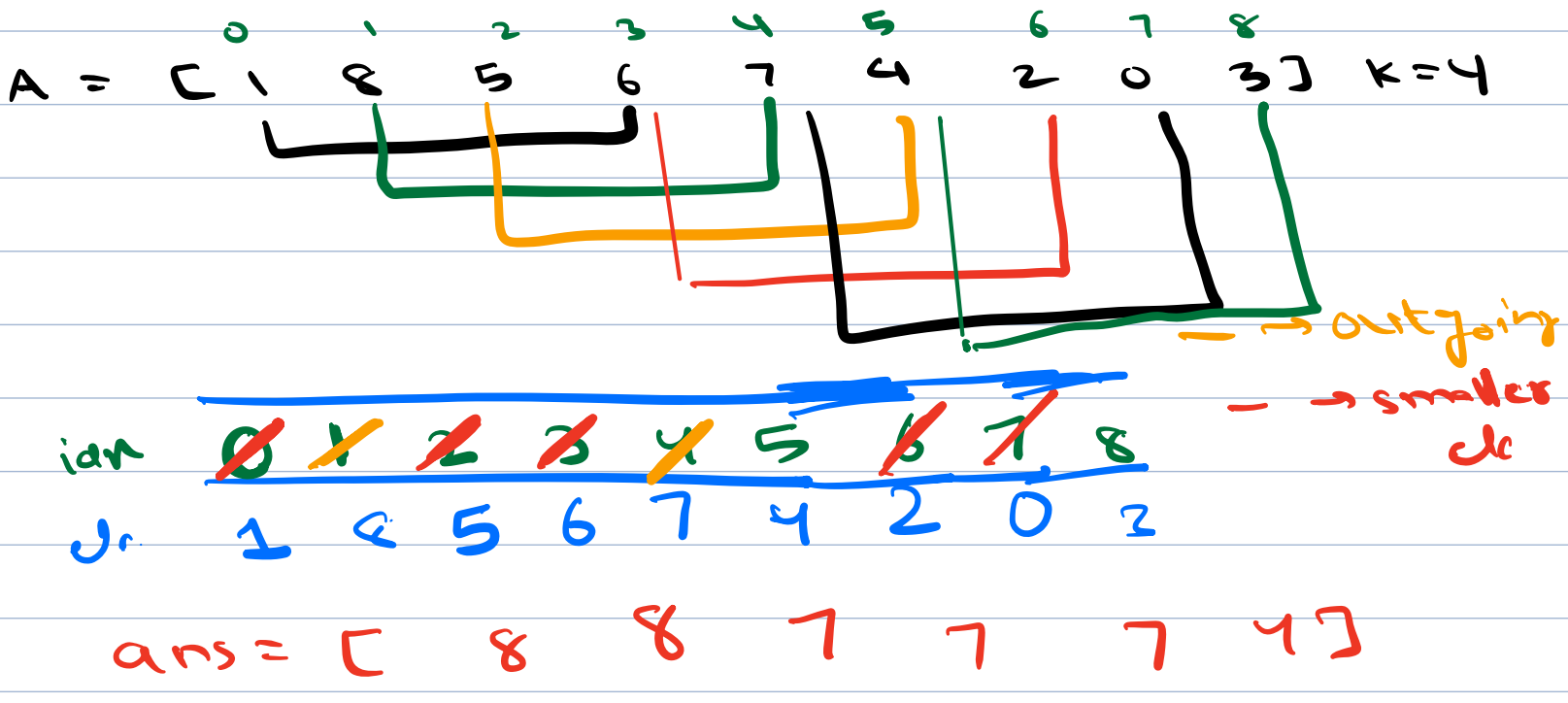
TC : O(N)

SC : O(k)

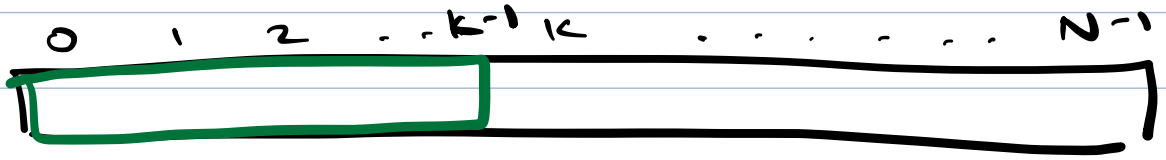
↓

deque

7



ans = [8 8 7 7 7 4]



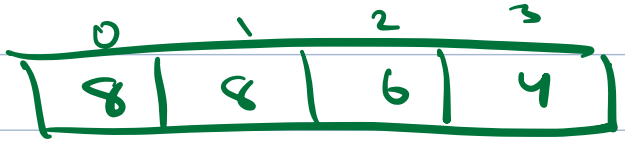
		s	e
1st	subarray →	[0	k-1]
2nd		1	k
3rd		2	k+1
⋮			
Last	subarray →	[	N-1]

$k-1 \rightarrow N-1$

No. of ending pts → $N-1-(k-1)+1$

$= N-1-k+1+1$

$= N-k+1$



$k=3$