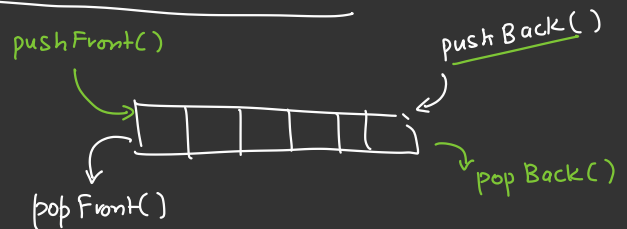
"Deque"

("deck" pronunciation)

"Doubly Ended Queue"

Queue
- enqueue
- dequeue

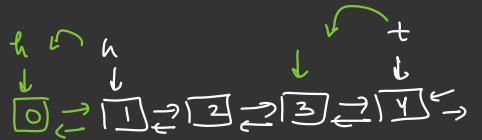


Own class

```
class Node {
    ...
}

class Deque {
    Node head
    Node tail
    ...
}
```

Queue



DLL

Names

push-Back()	O(1)
pop-back()	O(1)
push-front()	O(1)
pop-front()	O(1)

Insert/Del on a Doubly Linked List

```
Deque q = new Deque();
```

Deque (interface in java)

↓ methods are not implemented

```
import java.util.LinkedList
Deque<Integer> dq = new LinkedList<Integer>();
```

↓

dll

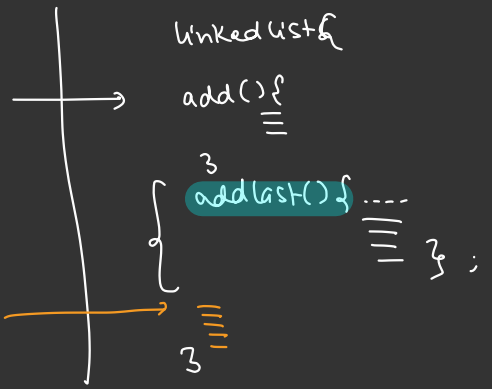
Deque {

→ addLast(){...}

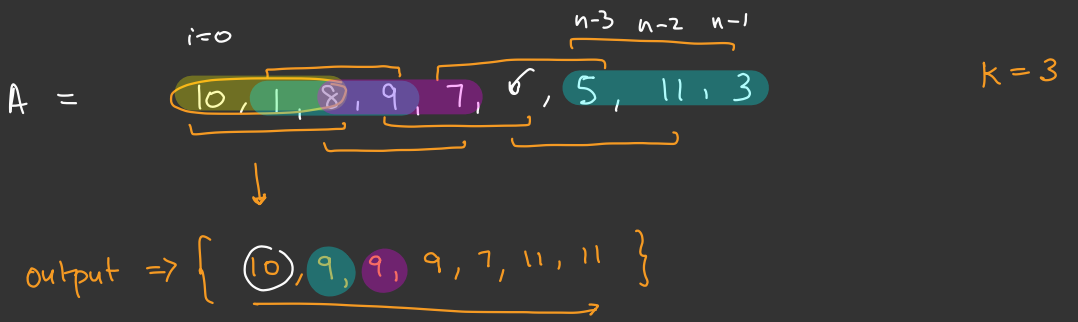
→ peekLast() {...}

→ removeLast() {...}

}



⑨ Array containing N elements, find out max val of every window of size K



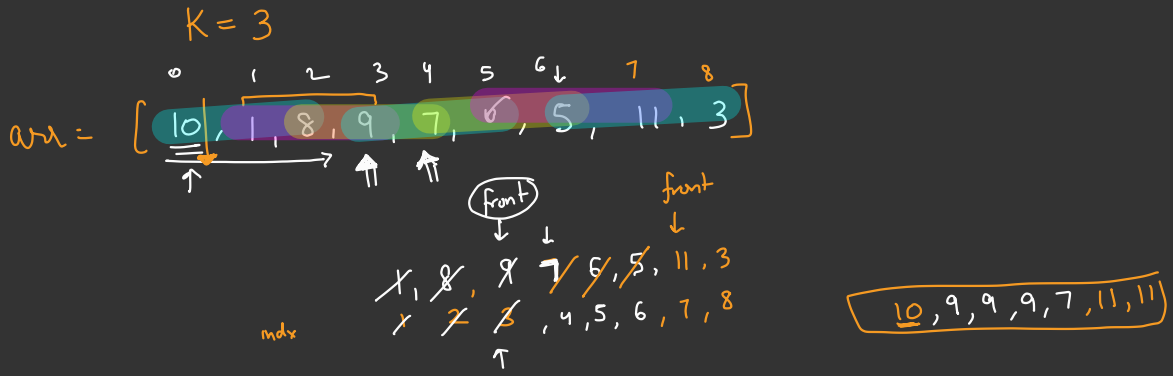
Brute force

```
for(i=0 ; i<=n-k; i++){
    max = -∞
    for(j=i; j<= i+k-1 ; j++){
        max = Math.max(a[j], max);      → O(1)
    }
print(max)
```

O(N·K)

3

Deque Based Algorithm (Unintutive , but its popular because of "deck")



⇒ Before Insert any element, remove all element at the back which smaller than 9

⇒ Max element is always front.

Comp. a[indx]

6 - 3 ≤ (3)

Code

```
Deque<Integer> dq = new LinkedList<>();
```

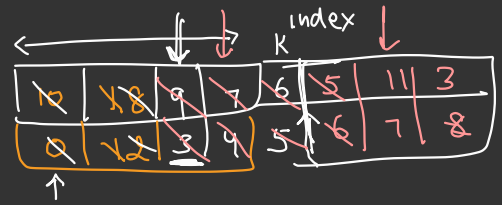
dq → indices

O(N)

Exclusive →

```
for (i=0, i<K; i++){
    // Remove smaller element
    while(!dq.isEmpty() && arr[i] >= arr[dq.peekLast()]){
        dq.removeLast();
    }
    dq.addLast(i);
}
```

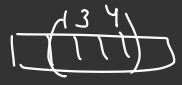
10, 11, 8




```
for (i=K; i<=N-1; i++){
    output = arr[dq.peekFirst()];
    Print(output)
    // Remove elements from front whose indices are the window
```

10,

0 <= 3-K
<= 3-3



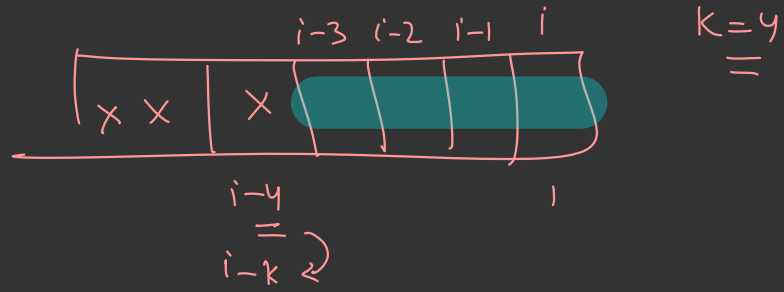
10, 9, 9, 9, 7, 11, 11

```
x → [ while( !dq.isEmpty() && dq.peekFirst() <= i-k ){
        dq.removeFirst()
    }

// Remove smaller elements from the back before pushing current element
=> [ while( !dq.isEmpty() && arr[dq.peekLast()] <= arr[i]){
        dq.removeLast();
    }
// Add
dq.addLast(i);
}
```

i = 3

```
print( arr[dq.peekFirst()] );
```



Break :)

Q1

Generate valid Brackets

N = 3

N pairs of brackets s.t all are balanced

Print

open = closing

5 are valid

((( )))

( ) ( ) ( )

(( )) ( )

( ) (( ))

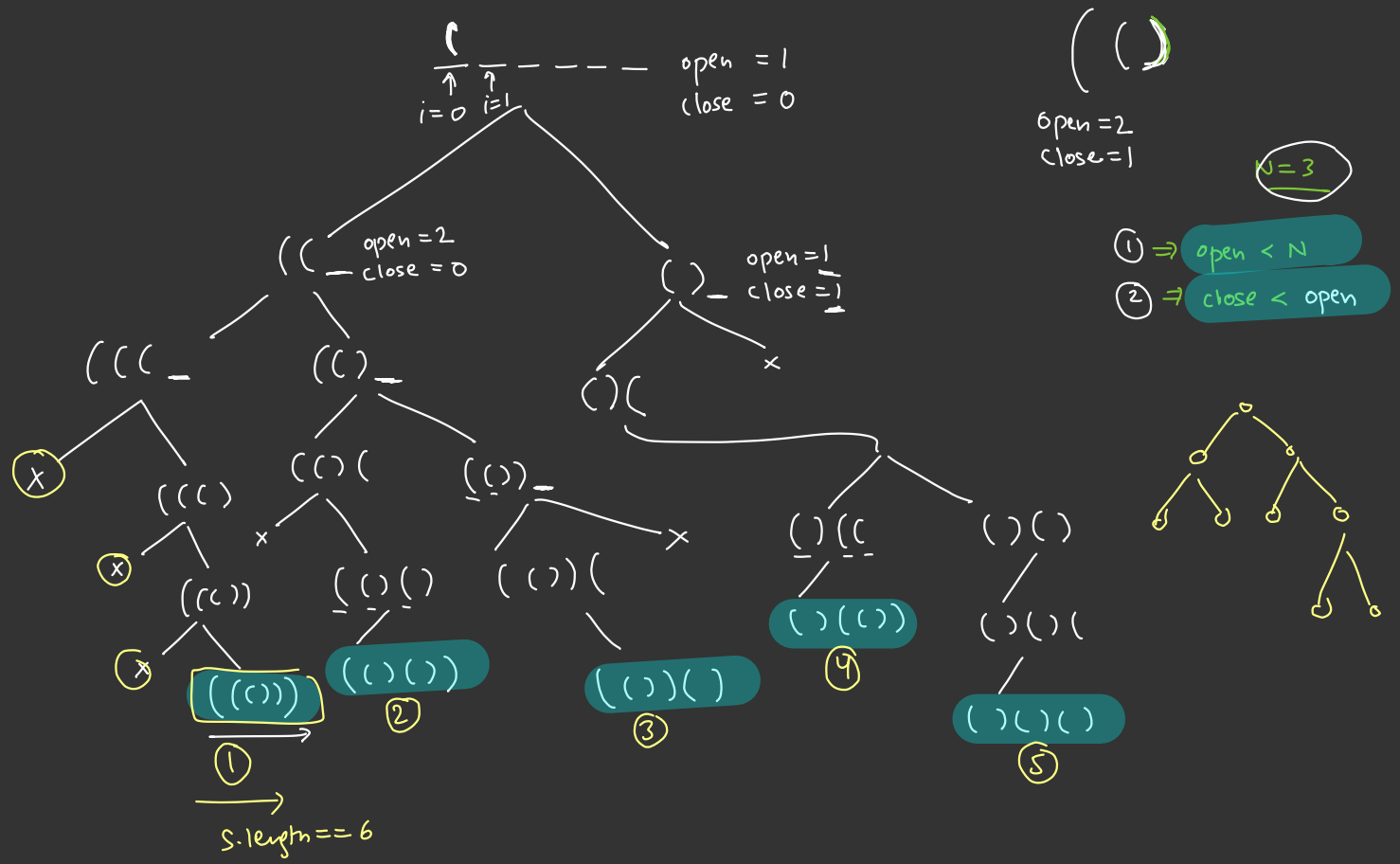
(( ) ( ))

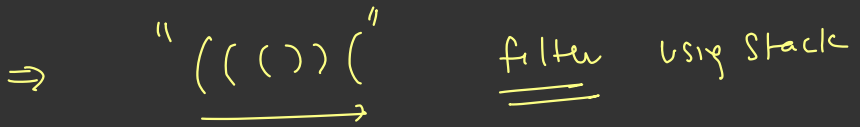
( ) ( ) ( ) = 2^6 possibilities

Str len = 6

( ) ( ) ( ) = ${}^6C_3 \; {}^3C_3$

))) (((= $\frac{6.5.4}{1.2.3}$ = 20


open = 1
close = 0
i = 0
i = 1
open = 2
close = 0
open = 1
close = 1
open = 2
close = 1
N = 3
1 ⇒ open < N
2 ⇒ close < open
S.length == 6
1
2
3
4
5




```java
public class generateBrackets {

    static void generateBrackets(String s,int N,int open,int close){
        //base case
        if(s.length()==2*N){
            System.out.println(s);
            return;
        }
        //rec case
        if(open<N){
            generateBrackets(s + "(",N,open+1,close);
        }
        if(close<open){
            generateBrackets(s+")",N,open,close+1);
        }
    }

    public static void main(String[] args) {
        generateBrackets("",3,0,0);
    }
}
```