Deque:

→ Deque basics

→ Double ended Queue : Doubts

→ Infix → Postfix :

→ Water logging : 2 pointers

→ Doubts

Deque: Double ended queue

Operations: → Implemented using double linked list

push_front() } S_1

pop_front()

Q_2 Q_1 {

push_rear() } S_2

pop_rear()

front() rear()

→ TC for each operation : O(1)

Thursday

→ June Morining: 2 : 7AM : ✓

→ June Morning: 1
VS June
Aug Advanced :

7:30am → 10:am 710 →

Friday: 7:30am

1Q) Given ar[N] & k, find max element in every window of size k

         0 1 2 3 4 5 6 7 8

Eg1: ar[] = 10 1 9 3 7 6 5 11 8

k = 4

Output: 10 9 9 7 11 11

Idea1: For every subarray of len = k, iterate & get max

TC: $[N-k+1] * k$ SC: $O(1)$

↳ $k \approx N/2$ TC: $O(N^2)$ SC: $O(1)$

Idea2:

         0 1 2 3 4 5 6 7 8

ar[] = $\cancel{10}$ $\cancel{2}$ $\cancel{9}$ 3 1 6 5 11 8

k = 4

data

1

$\cancel{2}$

3

5

6

$\cancel{9}$

$\cancel{10}$

output: 10 9 9 6 . . .

Idea2: Sliding window + Treemap

TC: $N * \{\log k + \log k\}$ SC: $O(k)$

↳ $k \approx N/2$ TC $\approx N \log N$ SC: $O(N)$

         0 1 2 3 4 5

ar[] : $\cancel{10}$ 6 10 3 9 11

k = 4

Treeset

$\cancel{10}$

3

6

9

output: 10 9*

Issue: Delete an element, indirectly all occurrences are getting deleted

Ex2:

	0	1	2	3	4	5	6	7	8	9	10	11	12
ar[] =	3	15	6	12	4	2	10	9	13	7	2	5	3

k=4

front ↳ ~~3~~ ~~15~~ ~~6~~ ~~12~~ ~~4~~ ~~2~~ ~~10~~ ~~9~~ ~~13~~ 7 ~~2~~ 5 3 ↲ rear

3

output: 15 15 12 12 10 13 13 13 13 7

operations: deque

- pop_front()
- push_rear()
- pop_rear()
- front()
- rear()

	0	1	2	3	4	5
ar[] :	10	6	10	3	9	11

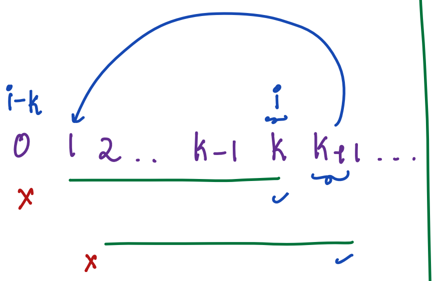
k=4

~~10~~ ~~6~~ ~~10~~ ~~3~~ ~~9~~ 11

rear < ar[i] : delete

output: 10 10 11

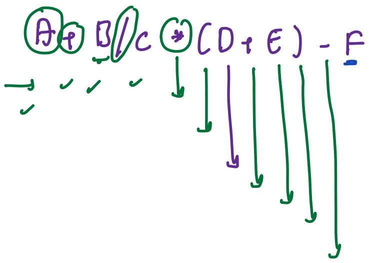
```
void   Submax(int ar[n], int k){   TC: O(N)  SC: O(k) ≈ O(N)
    deque<int> dq; // TODO, inbuil library in your language
    //Step1: Insert first k elements
    i=0; i<k; i++){
        while( dq.size()>0 && dq.rear() < ar[i] ){
            dq.pop_rear()                    dq.rear() > ar[i]
        }
        dq.push_rear(ar[i])
    }
    print(dq.front())
    i=k; i<n; i++){
        // insert ar[i], delete ar[i-k]
        while( dq.size()>0 && dq.rear() < ar[i]){
            dq.pop_rear()                dq.rear() > ar[i]
        }
        dq.push_rear(ar[i])
        if( dq.front() == ar[i-k]){
            dq.pop_front()
        }
        print( dq.front()
    }
}
```



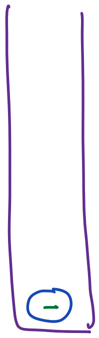
Subarray min

Infix → Postfix :

Infix:

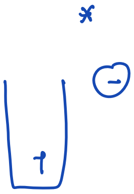


Stack



Postfix

A B C / D E + * + F -



{ top() > operat
top() = operato

→ Iterating on Infix: TC: O(N) SC: O(N)

: operand : send to postfix

: '(' : inside stack

: ')' : pop all items in stack till
we get a '(' & add them postfix

: (+) : operator

```
while(st.size() > 0 && precedence(st.top()) >= precendence(operator)){
    add st.top() to postfix
    st.pop()
}
st.push(operator)
```

:
pop all remaining elements of stack & add them to postfix

```
int precendence(char ch)
    if (ch == '+' || ch == '-') return 0
    else if (ch == '/' || ch == '*') return 1
}   else return 2
```

+, -	/ *	^
0	1	2

Part:4 : Trees : 11

BT BST Tries Heaps Greedy

→ google/microsoft :

Part5: 14

Backtracking Dp Graphs

Coming Monday : 7am : 14th Nov

I will taking ✓

Problems

→ Stacks/Queu/Deque

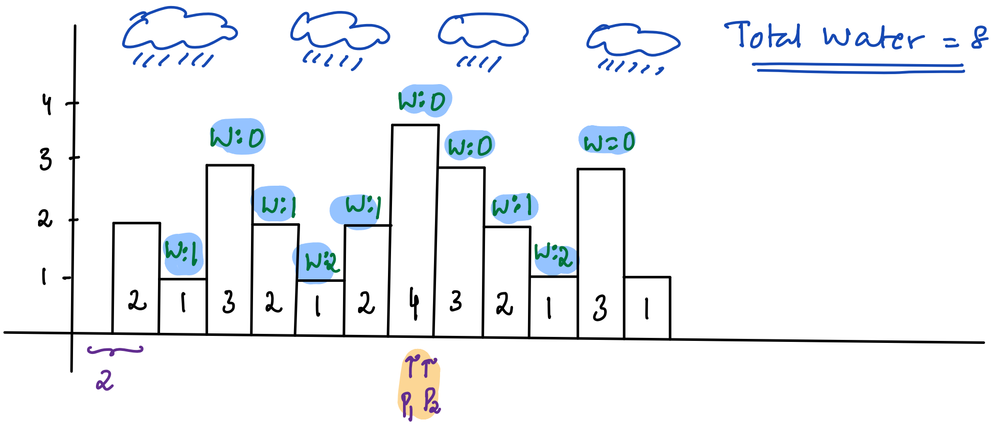
9:10 → 9:20

4Q) Rain water trapped ?

Given ar[N] elements, where ar[i] represents height of the building, retun amount of water trapped in all buildings

Note: Width of Each building is 1

	0	1	2	3	4	5	6	7	8	9	10	11
ar[] =	{2	1	3	2	1	2	4	3	2	1	3	1}



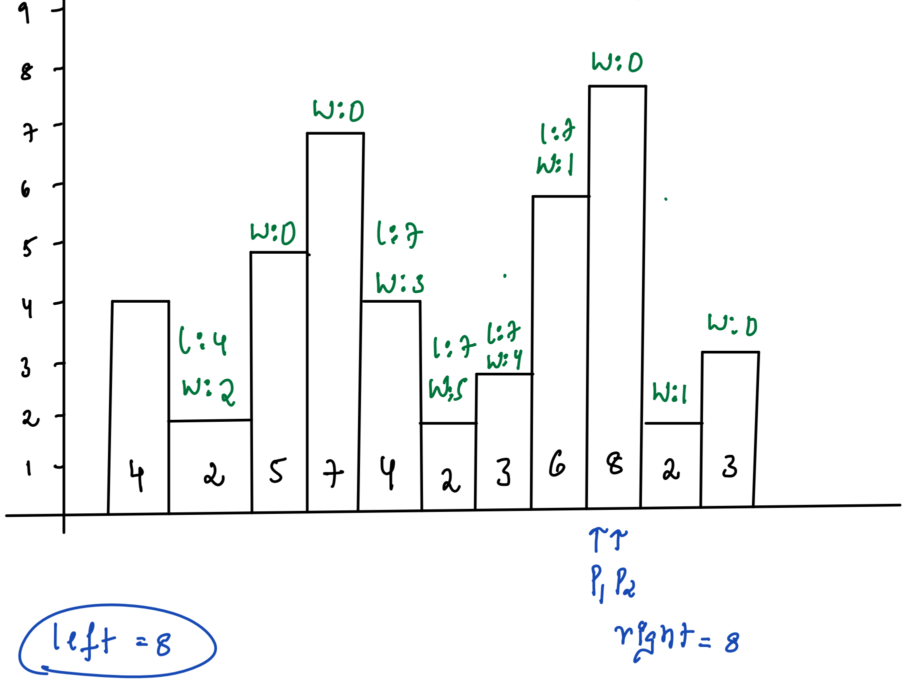
Total water = 8

left = 4

right = 3

Ex:

Obs: For a building we only need, min of (Lmax, Rmax)



Idea: For every building get it's limiting building height