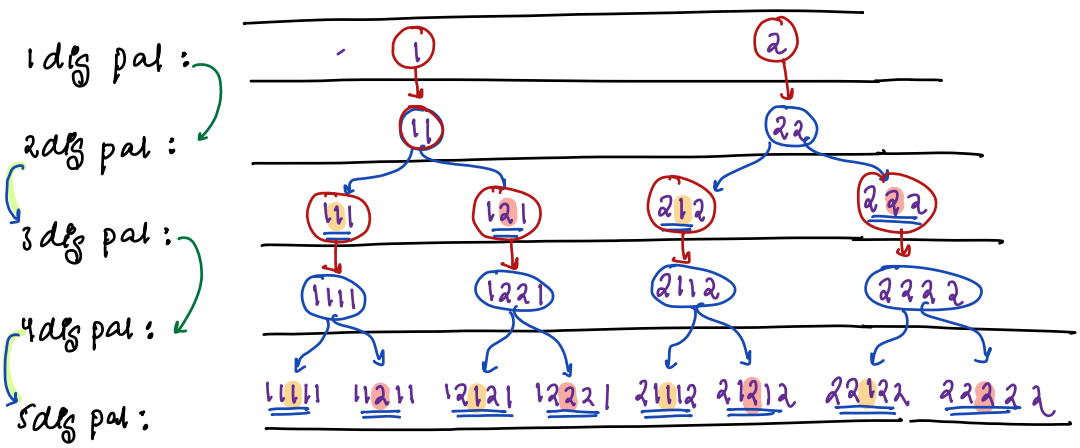
Todays Content:

- → Palindrome generation ✓
- → De-que() ✓
- → Max of every window ✓

Q: Print k^{th} pal number: generated by only 1 & 2

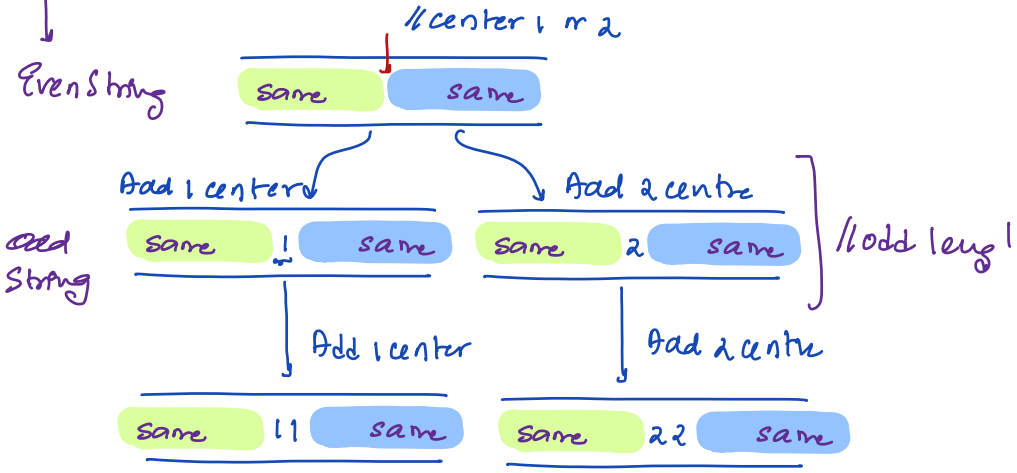
k=5: 1 2 11 22 111 → ans

k=7: 1 2 11 22 111 121 212 → ans



obs1: For even string: at center add 1 → get palindrome string

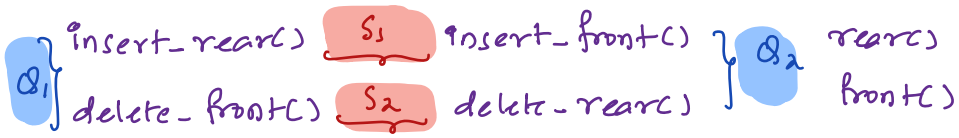
at center add 2 → get palindrome string



obs2: for odd string, add character at center once again at center to get → even length palindrome

TODO :

Deque: double ended queue



Using linked:

1) insert at start
2) insert at back
3) delet at front

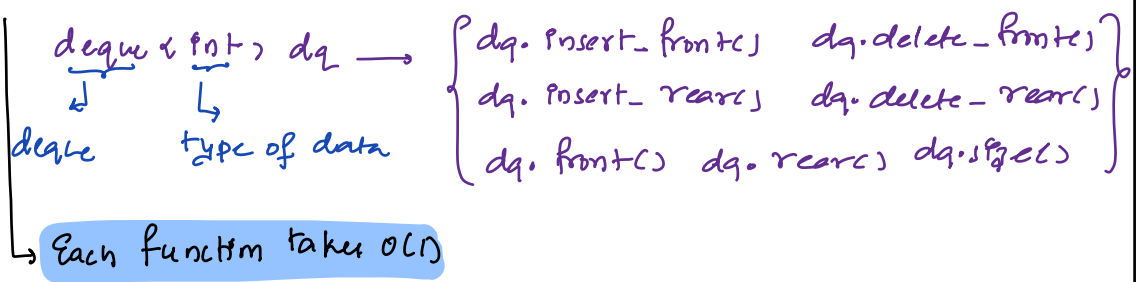
} double linked list

Code : TODO

4) delete at end

↳ LRU Cache

Syntax: { if not there implement it } Functional



↳ Each function takes O(1)

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

↳ subarray of : len k

↳ continous part of array

```
# N = 9     0   1   2   3   4   5   6   7   8
Ex1: ar[9] = 10  1   9   3   7   6   5  11   8
```

k = 4

} # N - k + 1 = 9 - 4 + 1 = 6

Output : 10 9 9 7 11 11

Idea1: For every subarrays of len = k iterate & print max

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

↳ k ≈ N/2 = (N - N/2 + 1)(N/2) = (N/2 + 1)(N/2) ≈ O(N²)

Idea2:

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

DataStructure → max / delete ele() / insert() // TreeSet : Hashmap : 2

```
[ 10   1 ]
[  8   6 ]      10 9 9 6 ....
[  9   5 ]
[  3     ]
```

store all data in sorted order

→ min : log N

→ max : log N

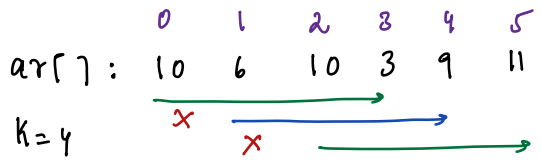
```
          0   1   2   3   4   5
ar[] :   10   6  10   3   9  11
```

k = 4

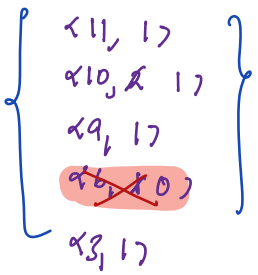
TreeSet *{ duplicates get deleted } ⇒ Treemap < key, value > : arr[] & it's frequency

Data sorted based on keys

```
{ 10 }      10 9
{  6 }
{  3 }
{  9 }
```



Treemap



Idea: Treemap + Sliding Window : TODO

↳ : Every element goes inside Treemap once
gets deleted once

TC: N * log k SC: O(k)

TC: N

Eg2:

	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 → rear

Output: 15 15 12 12 10 13 13 13 13 7 // cross verify

operations: obsl: rear ele & new element : rear ele del

→ rear_element | rear_insert() | rear_delete() } deque
→ front_element | | front() - delete()

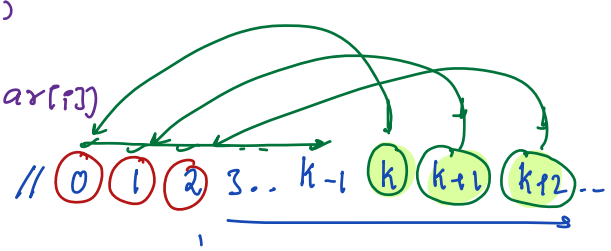
	0	1	2	3	4	5
ar[] :	10	6	10	3	9	11

k=4

front → #new=11 → rear

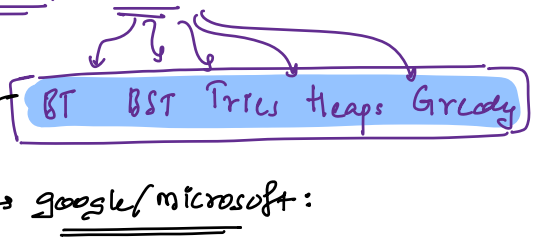
Output: 10 10 11 // cross check

```
void subman(int ar[n], int k){ TC:O(N) SC:O(k)
    deque<int> dq;
    Step1: Insert first k ele in window [0, k-1]
    i=0; i<k; i++){
        //new element insert = ar[i]
        while(dq.size()>0 && dq.rear() < ar[i]){
            dq.delete_rear()
        }
        dq.insert_rear(ar[i])
    }
    print(dq.front())
```




```
    i=k; i<n; i++){
        // new element insert = ar[i]  // delete = ar[i-k] ?
        while(dq.size()>0 && dq.rear() < ar[i]){
            dq.delete_rear()
        }
        dq.insert_rear(ar[i])
        if(dq.front() == ar[i-k]){
            dq.delete_front()
        }
        print(dq.front())
    }
}
```

Part:4 : Trees : 11 : 4 weeks



→ google/microsoft:

Part:5 : 14 :

Backtracking Dp Graphs

28th | 4th → 1st

9 days

Coming Sunday : 10:30am

Rahul yadav

→ Stacks / Queue / Deque / linked list

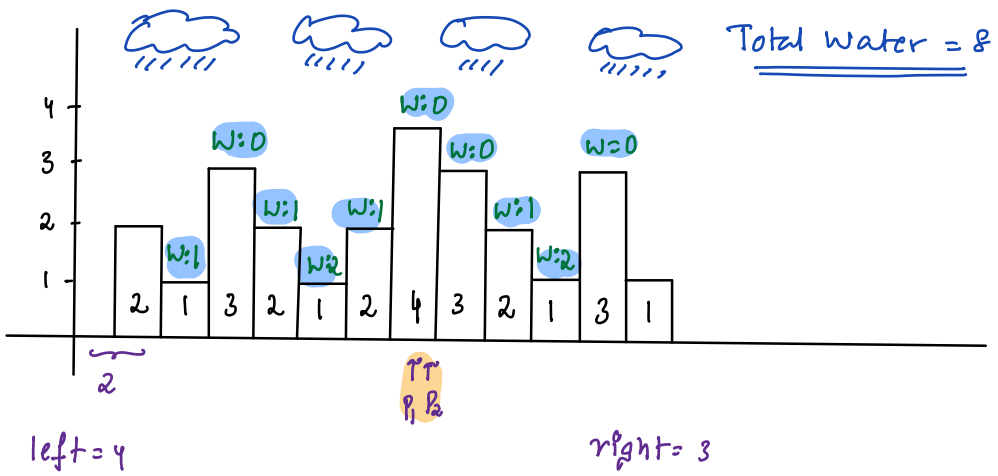
9:10 ⟶ 9:20

40) Rain water trapped ?

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

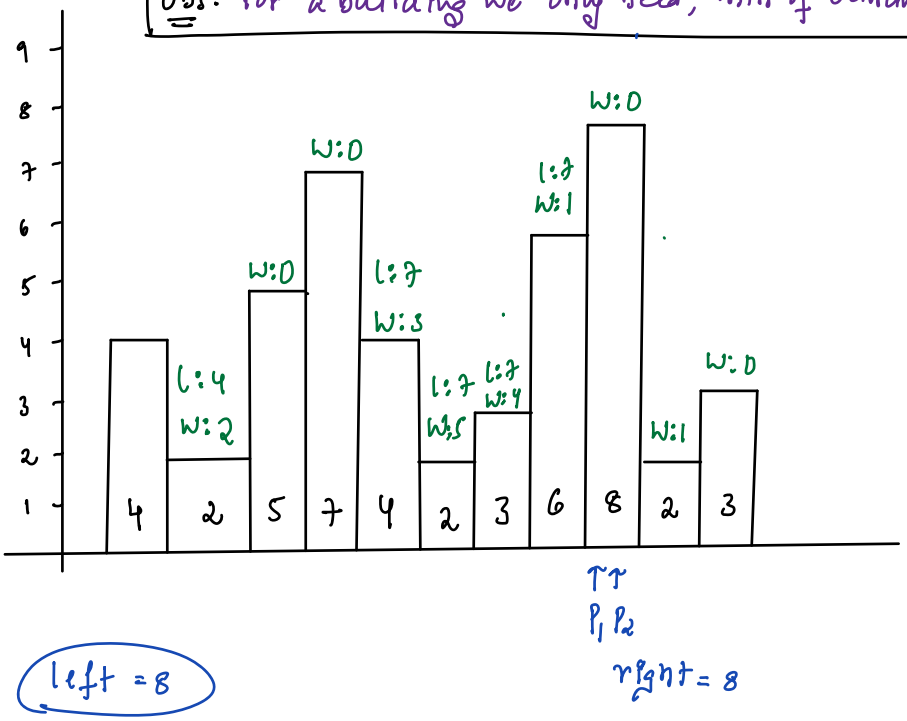
Note: Width of each building is 1

	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 }



Ex:

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



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