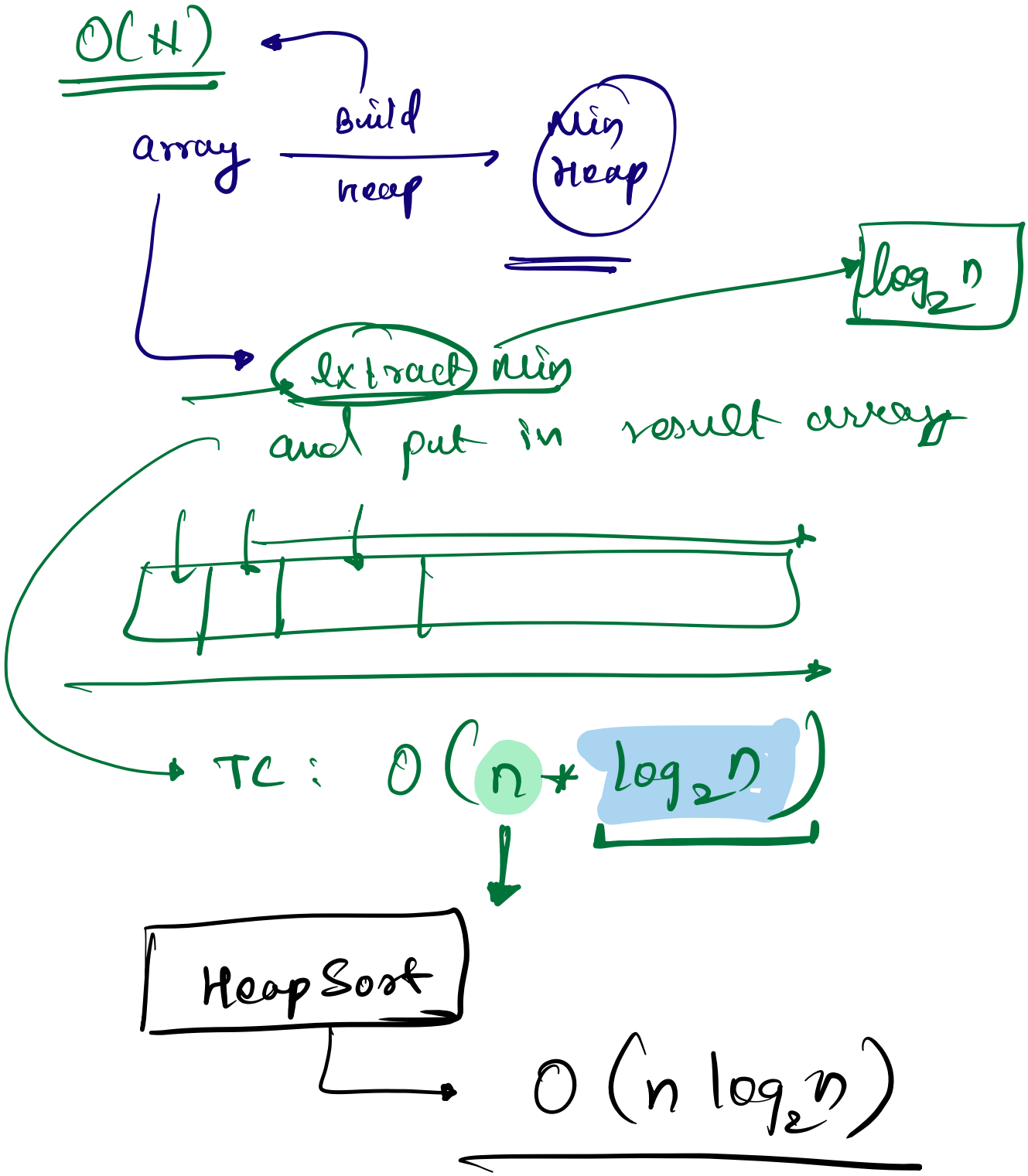
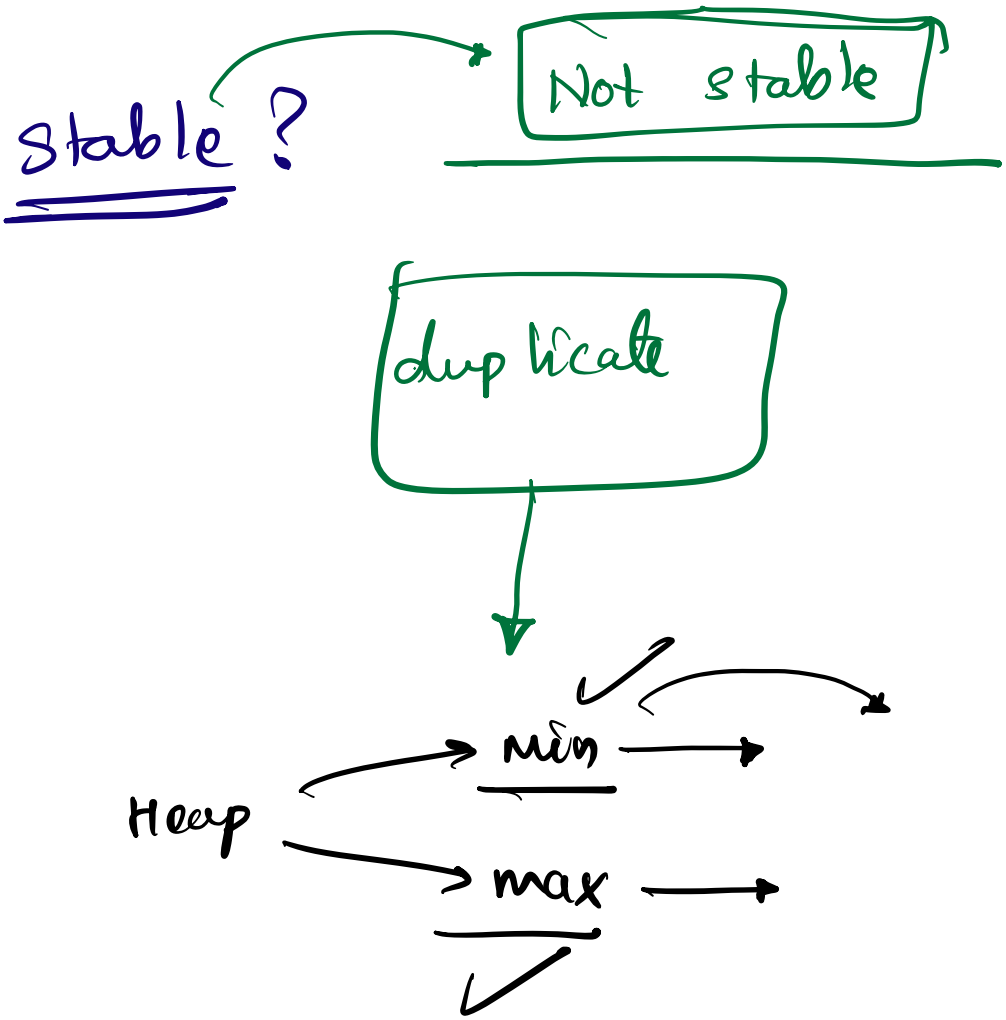
"Hello Everyone!"



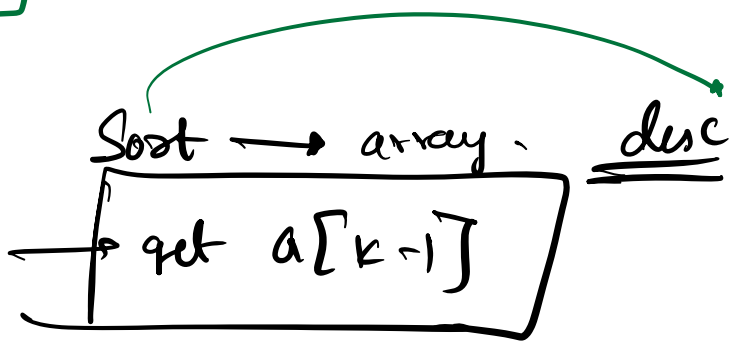

Stable ?
Not stable
duplicate
Heap
min
max

(Q) Find k^{th} largest element in array.

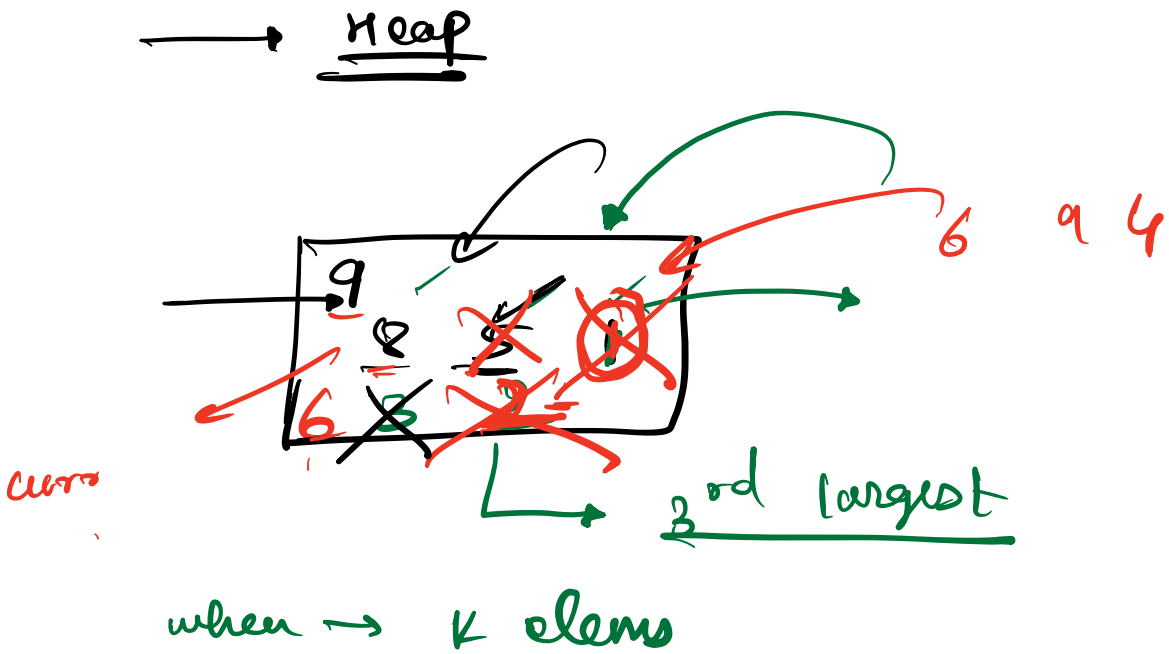
eg:- a = 8 5 1 2 3 6 9 4

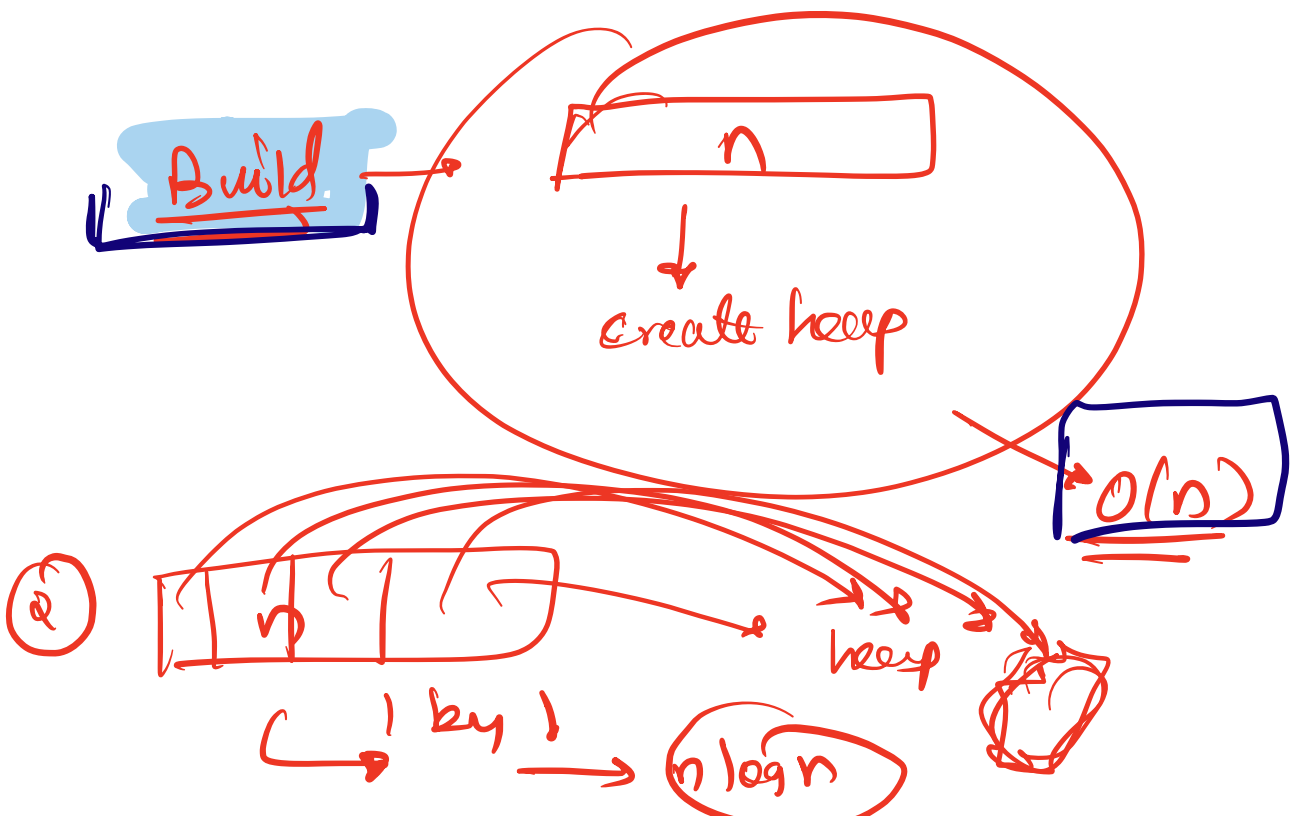
k = 3

Apl :-



→ $O(n \log_2 n)$


Heap
9
8
6
6
9 4
curr
3rd largest
when → K elems


Build
n
create heap
O(n)
2
n
heap
1 by 1
n log n

steps :-

① Build a minheap of first K elements.

② for remaining (n-K) elements

if (a > 7) {
  if min of heap is smaller than the current elem
    → extractMin() → Remove
    Insert current
}

→ size of heap → K

TC : O(K + (n-K) * $\log_2 K$)

size of heap

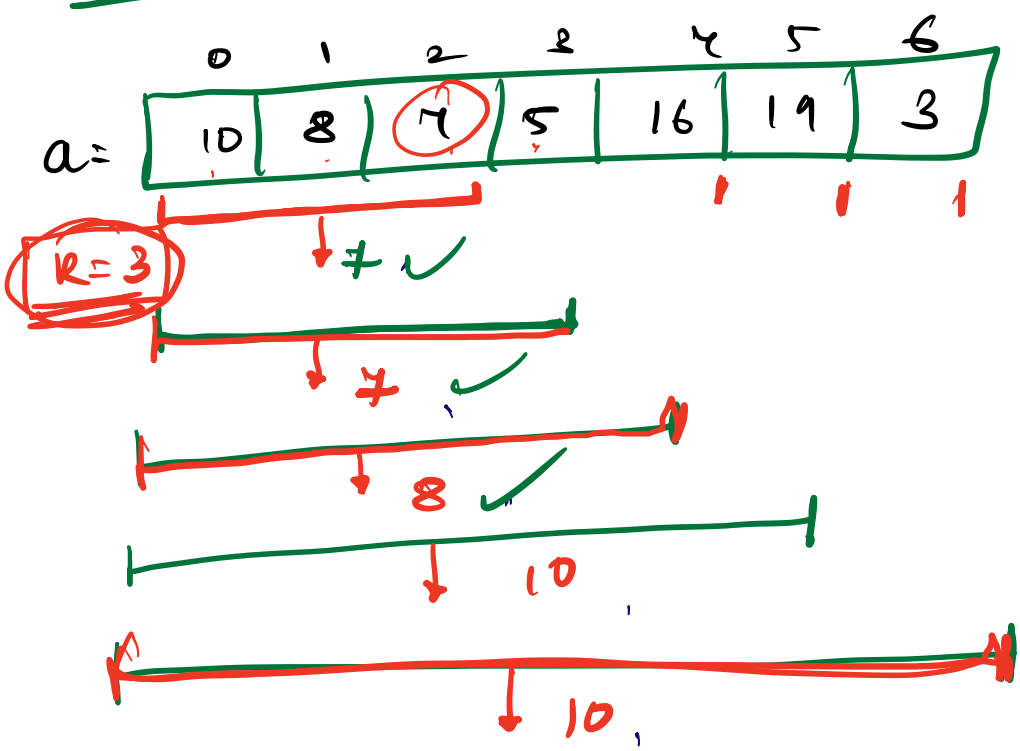
$n \log n$

$n > k$

(Q.2) k^{th} largest elem for every window $(0 - i)$

$i >= k-1$

eg1



O/P: 7 7 8 10 10

99% → same as prev

↳ just print min of heap for every iteration.

→ Rest everything exactly same.

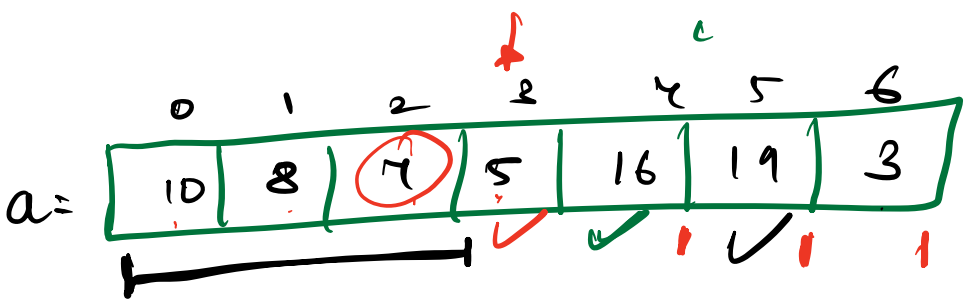
Pseudo Code

a ⟶ n

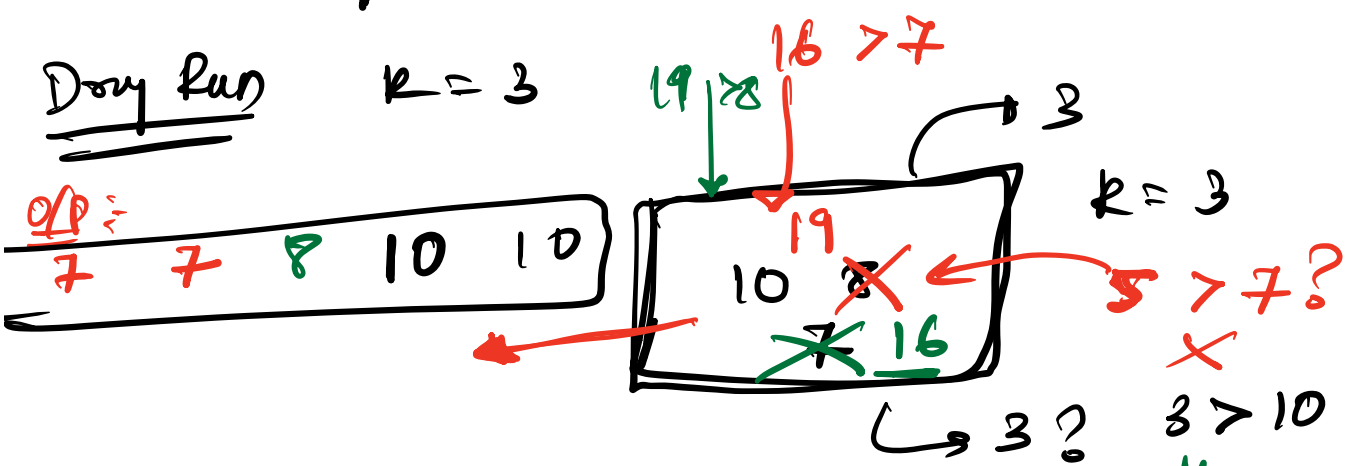
{

for i = 0 ; i < k ; i++

↳ build a heap → min



Dry Run k = 3



Imp :- Among k element, k^{th} largest elem is the min elem.

TC: $O(k + (n-k) * \log_2 k)$

$n > k$

9:59 → 10:09

⭐3 K-sorted array

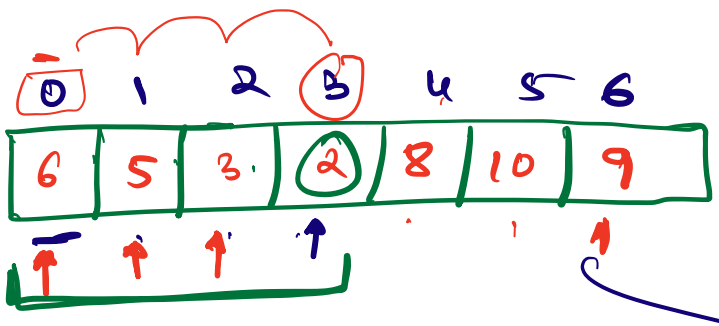
↳ every elem in array is

atmax K- distance apart

from its **sorted** location.

→ return the sorted array.

eg :-

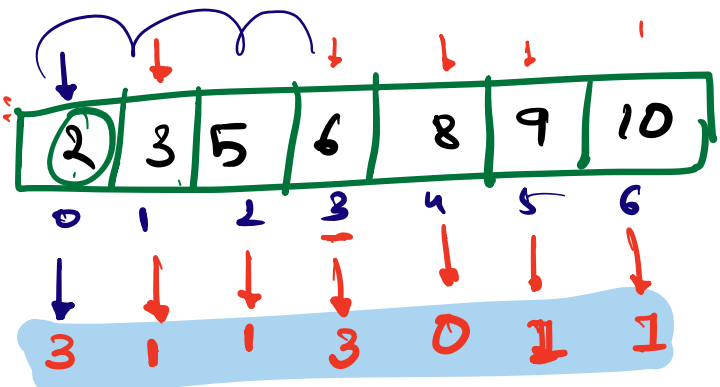


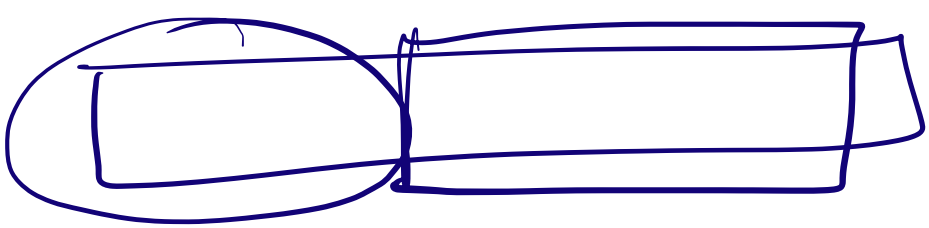
$0 \le k \le n-1$

$k = 3$

valid I/P

o/p :

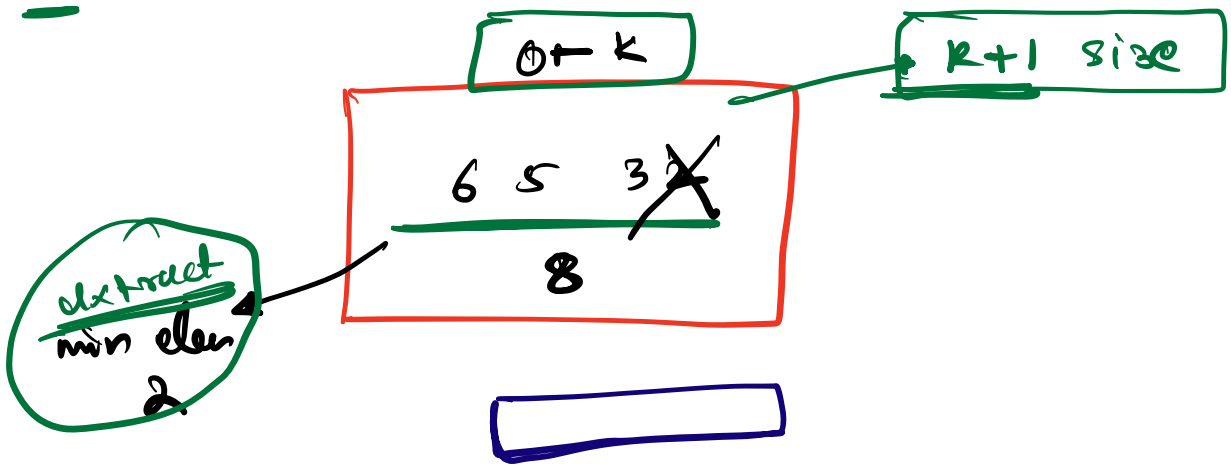




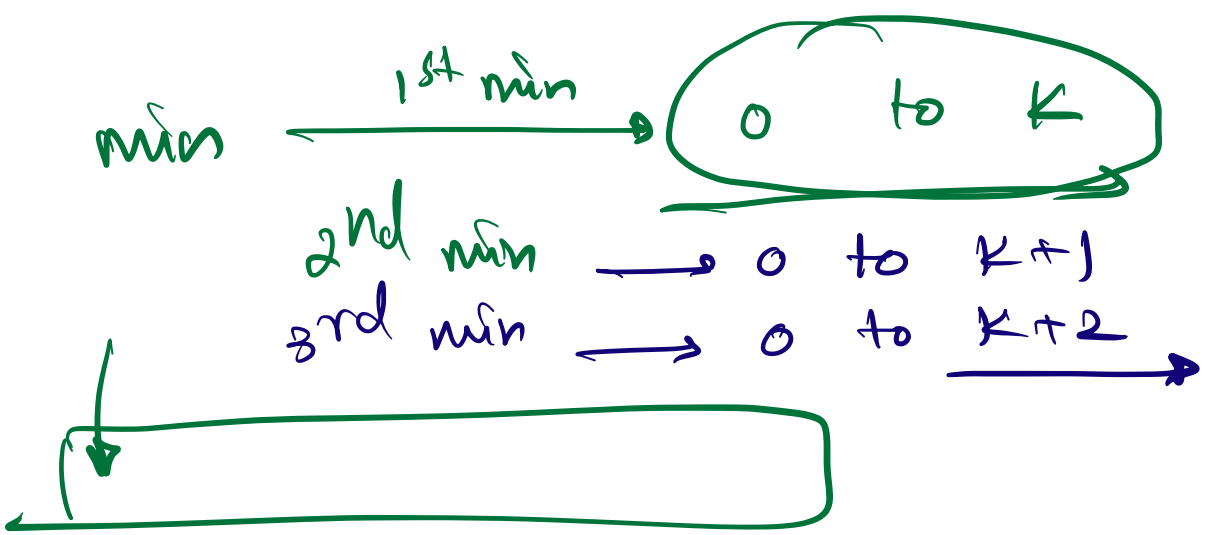
App 1 :- Sort ⟶ $O(n \log_2 n)$ ✓

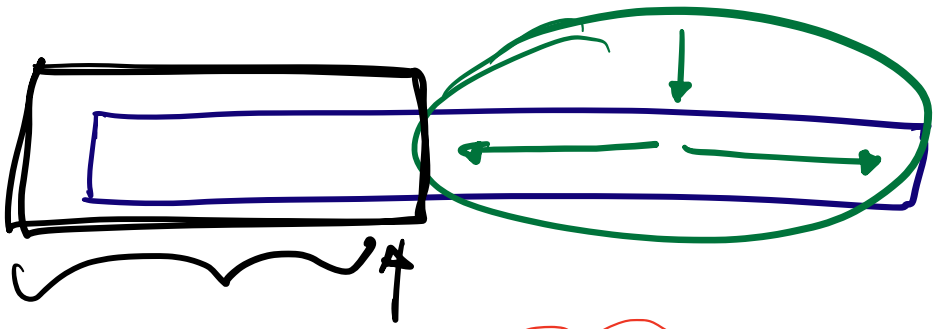
App 2 ⟶ use the given info:

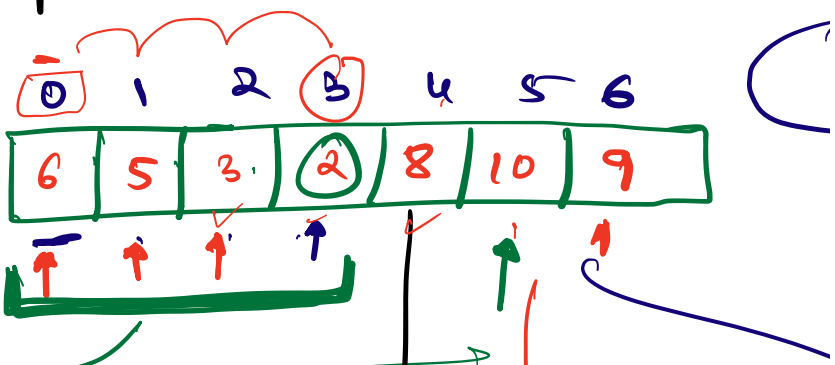
0th ⟶



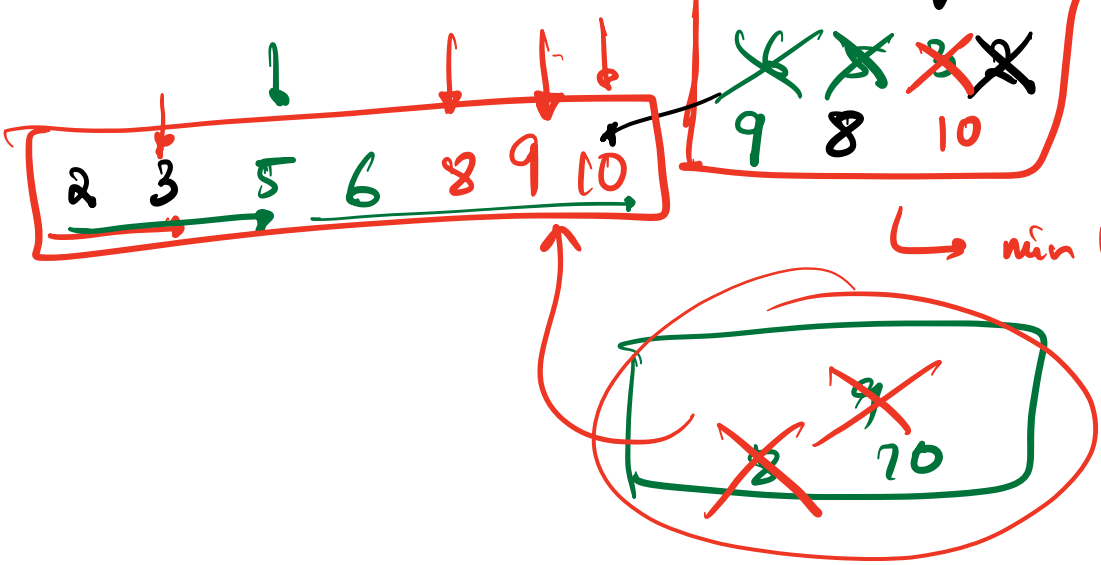
2 3







Dry Run :-



↳ min heap.

Pseudocode

① put 0-k index elems to min heap

0 - k

Build heap of k+1 elems ②

```
for ( i=k+1 ; i<n ; i++)
{
    extractMin() → put it in the ans array
    insert(a[i])
}

while ( minheap is not empty)
{
    extractMin()
}
```

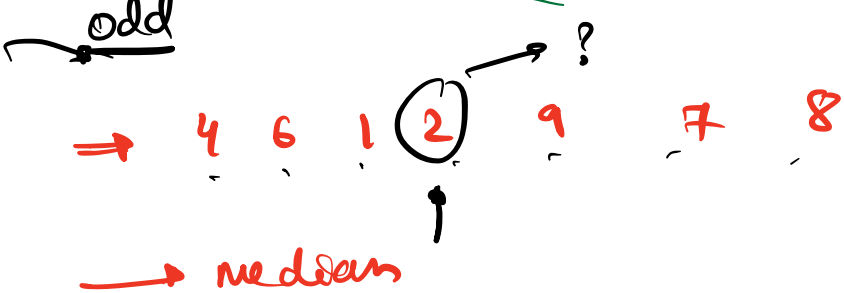
n

TC: $O\left((k+1) + (n-(k+1)) * \log_2^{(k+1)}\right)$

(Q) In a Running stream of integers. find the median of the elements present so far, after every new element joined.

Median → middle element in sorted order.

odd



1 2 4 6 7 8 9

median.

even :-

4 6 1 2 9 7

median → 1 2 4 6 7 9

avg → (4+6)/2

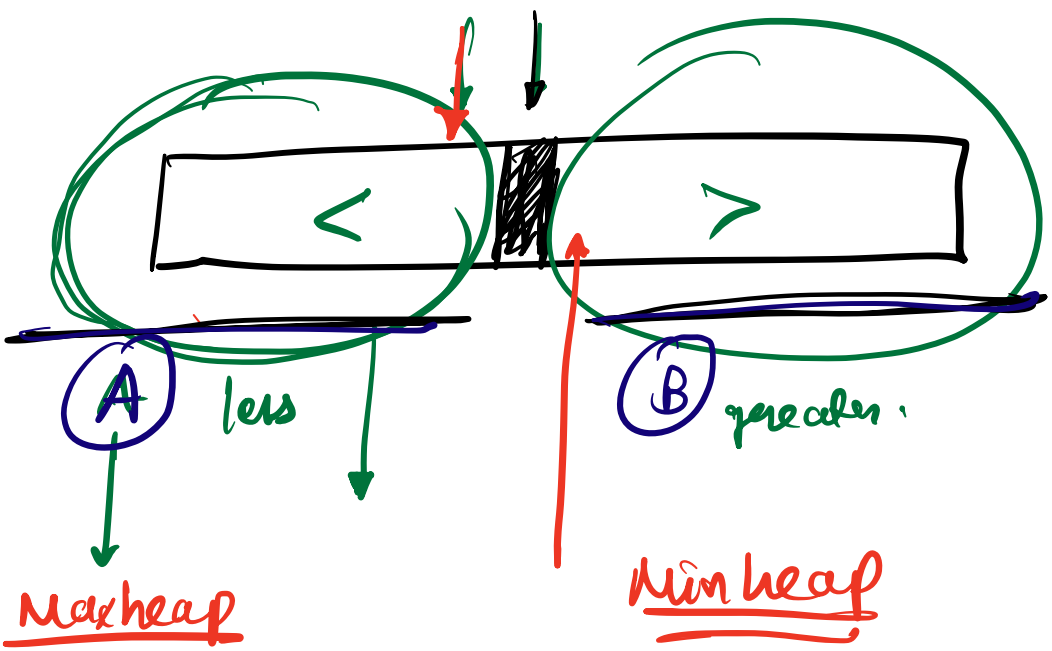
↳ 10/2 → 5

Running Stream

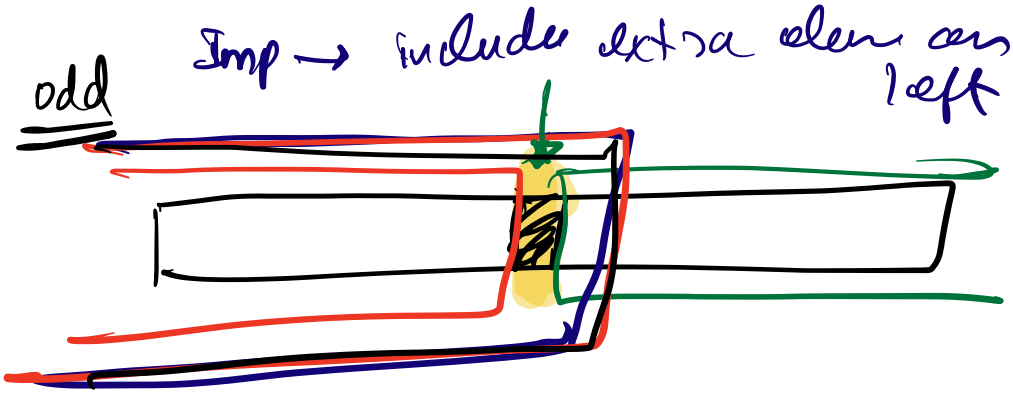
(9+8)/2 → 8

Stream: 9 8 7 3 6 5 . . . →

o/p: 9 8 8 7 7 6



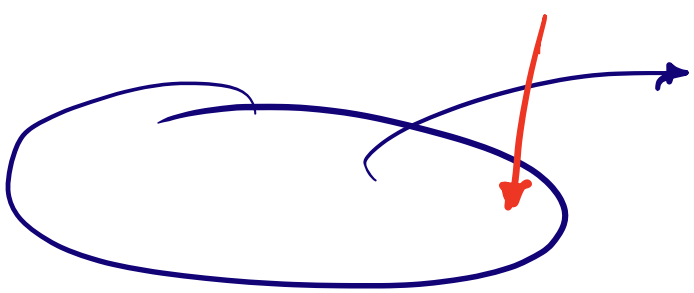
Size (A) − size (B) <= 1



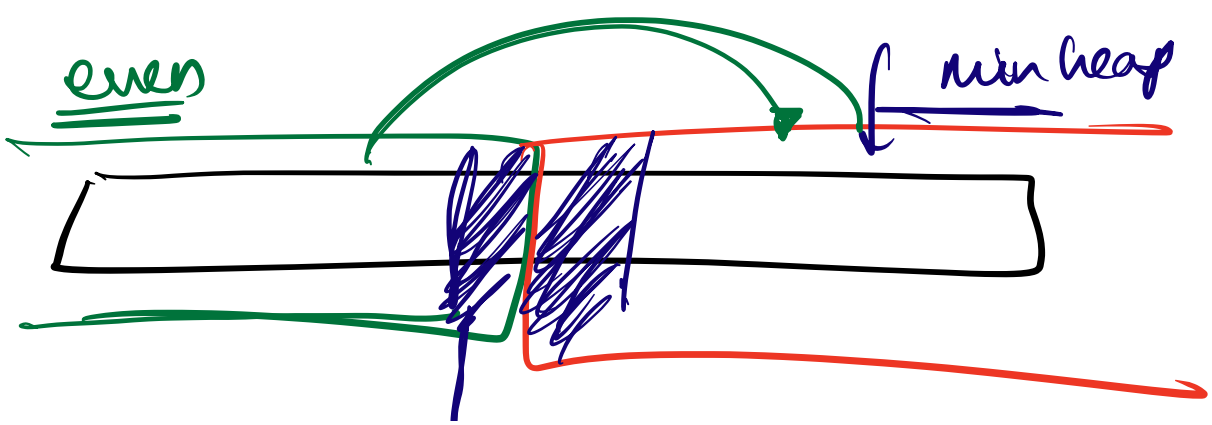
eg: 15 odd

8 7

median exits



maxheap → max elem



median = $\frac{\text{max of A} + \text{min of B}}{2}$

while(new elem) → x

{

x > max of A → in bucket B.

if (size B > size A)

{ extract Min from B

insert in A.

}

x < max of A → BucketA

if (size(A) - size(B) > 1)

{ extract Max from A

insert in B

}

ans → dependency on even/odd.

odd → max of A

even → $\frac{\text{max of A} + \text{min of B}}{2}$