Q K smaller elements.

I/p:- A:[8, 3, 10, 4, 11, 2, 7, 6, 5, 1] , K=4

O/p:- [1, 2, 3, 4]

Sorting Approach.

T.C= O(nlogn)

S.C= O(N)/O(logN)

Min heap Approach.

- Create a min heap → O(N)
- Keep extracting min. → K times.
  → KlogN

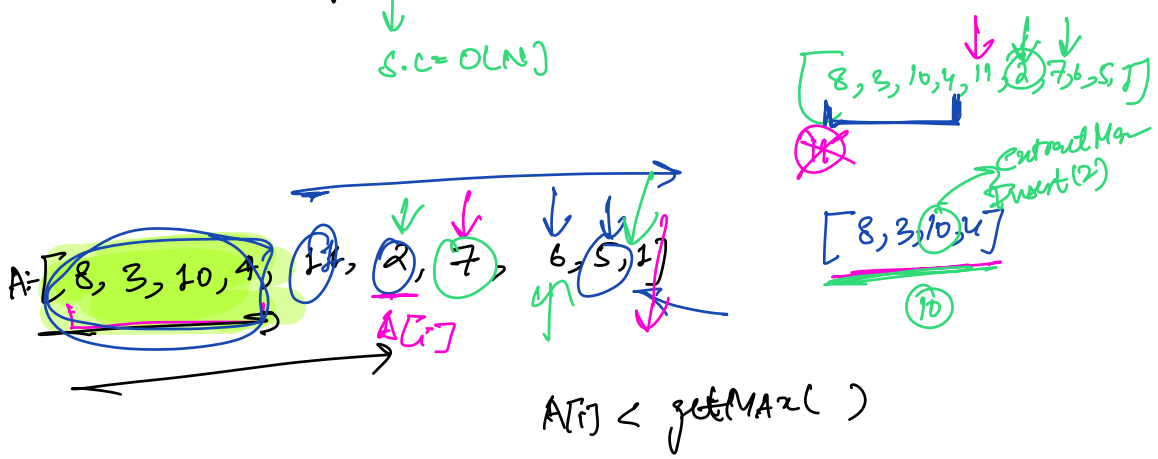
T.C= O(N) + O(KlogN)

S.C= O(1)

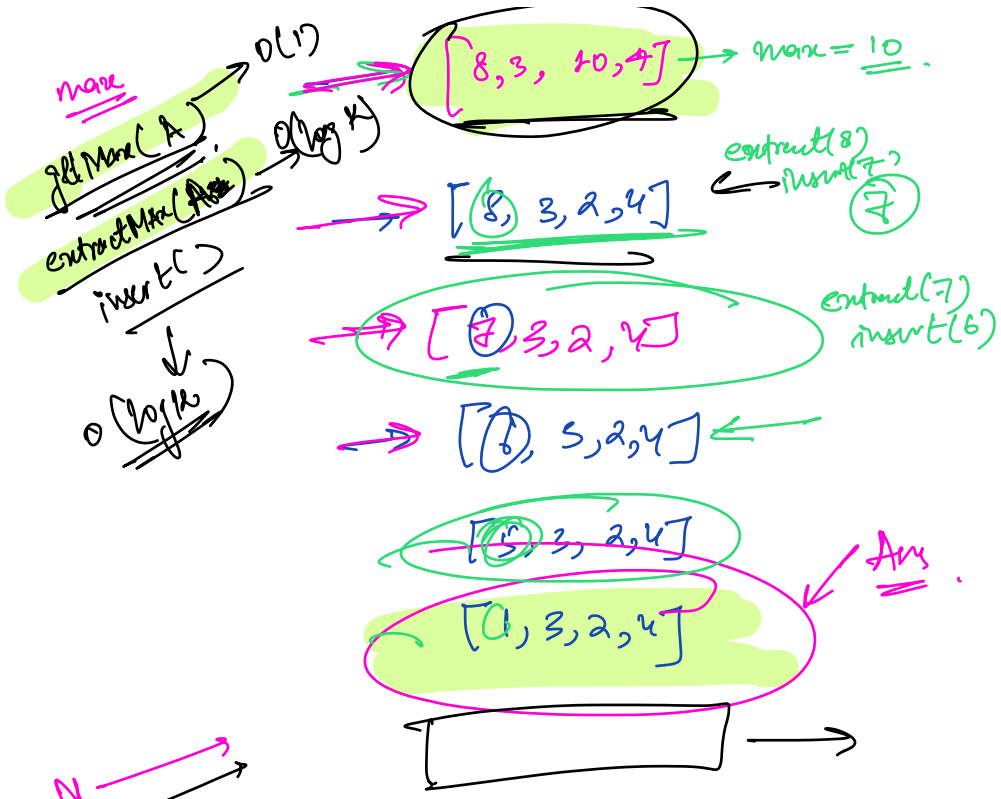
Cannot modify the given array.

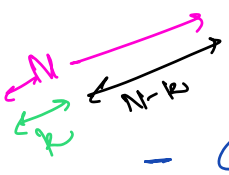
↓

S.C= O(N)



A[i] < getMax( )





- Create a max heap of first K elements from the given array.
  → O(K)
- Iterate over remaining (N-K) elements.
  - if (A[i] < heap.top()) { → O(1)
    - extractMax() → O(log K)
    - insert(A[i]) → O(log K)
  - }

O((N-K) * log K)

K = N/2

T.C = O(K) + O((N-K) * log K)

S.C = O(K) / O(1)

Q.

Given an array.

Nearly Sorted, K-Sorted array

* every element is atmost K positions away from its sorted position.

Goal:- Sort the array.

→ A: [6, 5, 3, 2, 8, 10, 9], K=3

Sorted A: [2, 3, 5, 6, 8, 9, 10]

|actual Index − sorted Index| ≤ K

(6 − x) ≤ K

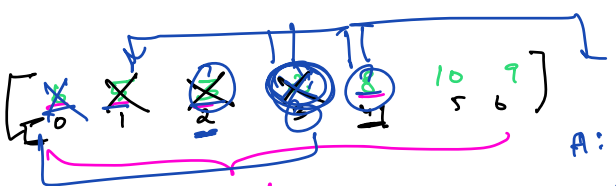
x ≤ K ∴ x = K

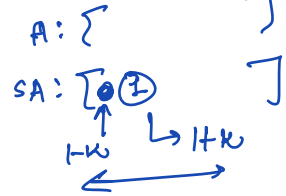
012 = 3

0123 = 4

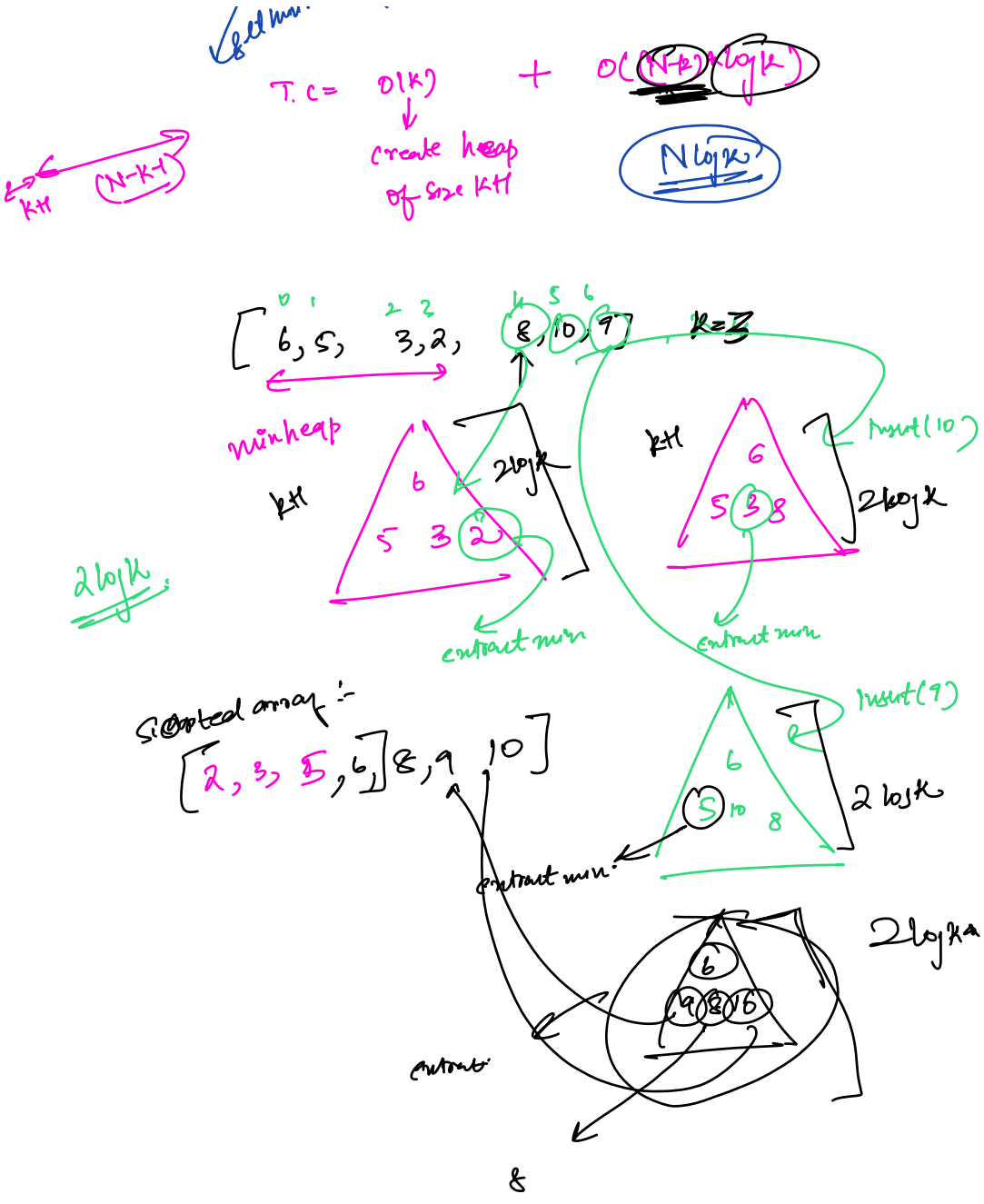
(012...K) = K+1

→ min of first (K+1) elements will be a[0] (sorted array)





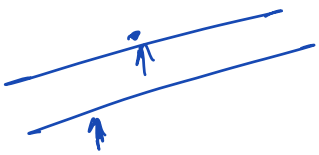
[6 5 3 2]


T.C = O(k) + O((N-k)log k)
create heap of size k+1
k+1
(N-k-1)
N log k
minheap
k+1
2log k
insert(10)
insert(9)
extract min
extract min
extract min
sorted array :-
2log k
2log k

Q. Given an array. Sort it inplace.

↳ S.C. O(1)

Bubble Sort
Insertion Sort
Selection Sort
→ O(N²) T.C.
S.C = O(1)



Hint:- Selection Sort + How a heap is created.

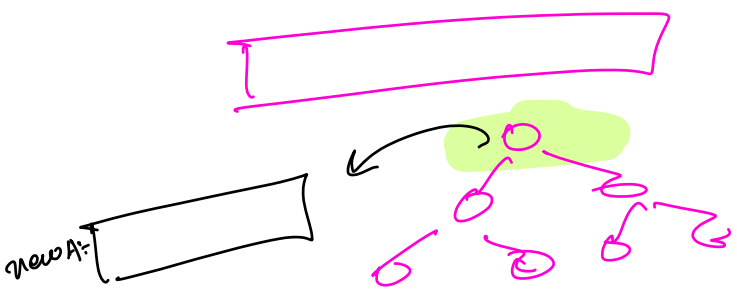
→ [8, 3, 7, 6, 1, 5, 10, 4, 9]

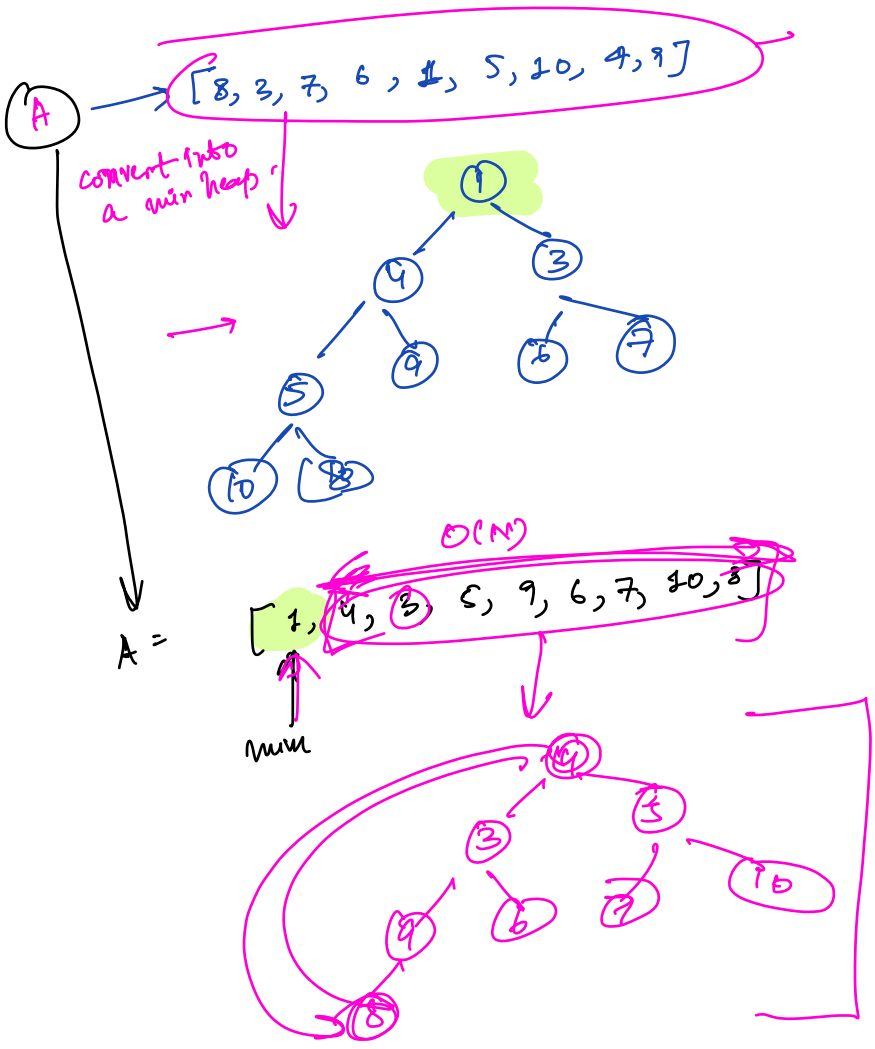
- Create heap
- extract max.

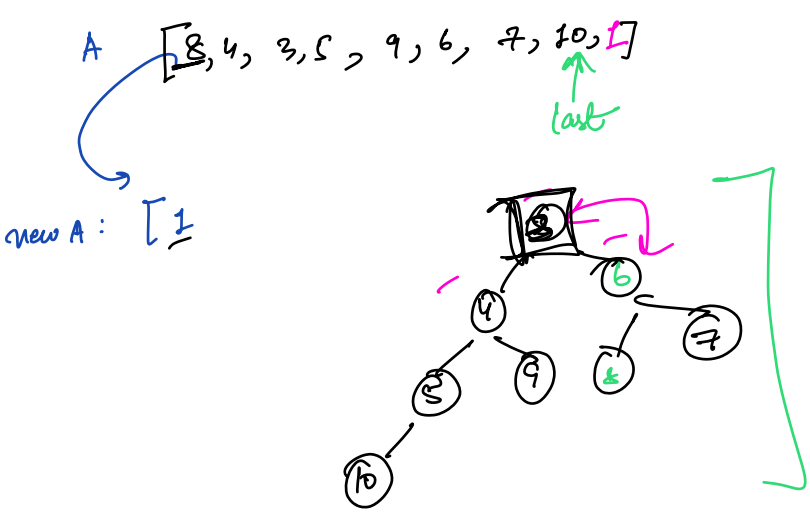
Selection Sort.

* get min : O(N) → logN : min heap
* put in correct position : O(1)

N → O(N²)



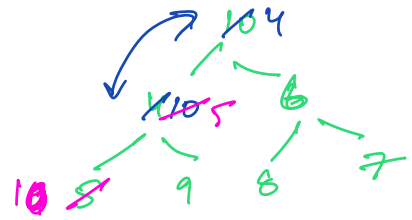

A
[8, 3, 7, 6, 1, 5, 10, 4, 9]
convert into a min heap
O(N)
A =
[1, 4, 3, 5, 9, 6, 7, 10, 8]
min


A
[8, 4, 3, 5, 9, 6, 7, 10, 1]
last
new A : [1

A: [10, 4, 6, 5, 9, 8, 7, 3, 1]

new A: [1, 3

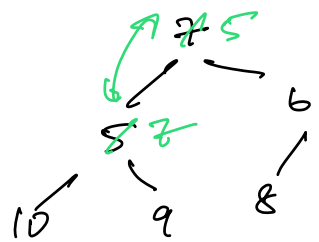
last ptr.



A: [7, 5, 6, 10, 9, 8, 4, 3, 1]

new A: [1, 3, 4

last



A: [8, 7, 6, 10, 9, 5, 4, 3, 1]

last

Sorted A: [1, 3, 4, 5

1, 3, 4, 5

A: [10, 9, 8, 7, 6, 5, 4, 3, 1]

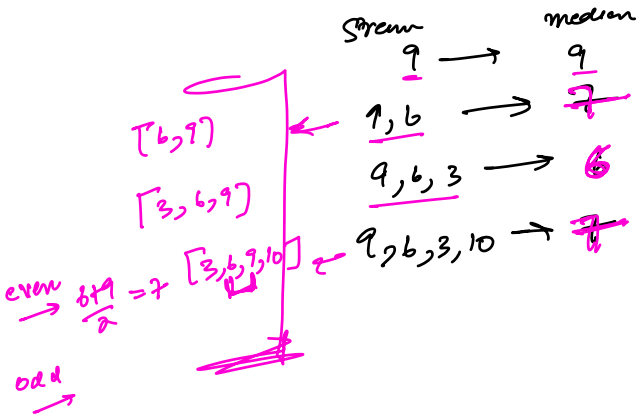
Sorted A [1, 3, 4, 5, 6, 7, 8, 9, 10]

HeapSort

T.C = O(N) + O(N log N) : O(N log N)

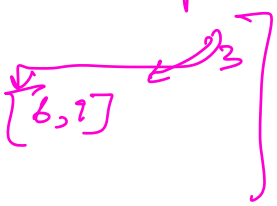
S.C = O(1)

Q. Given a stream of integers. Find the median with every insertion.

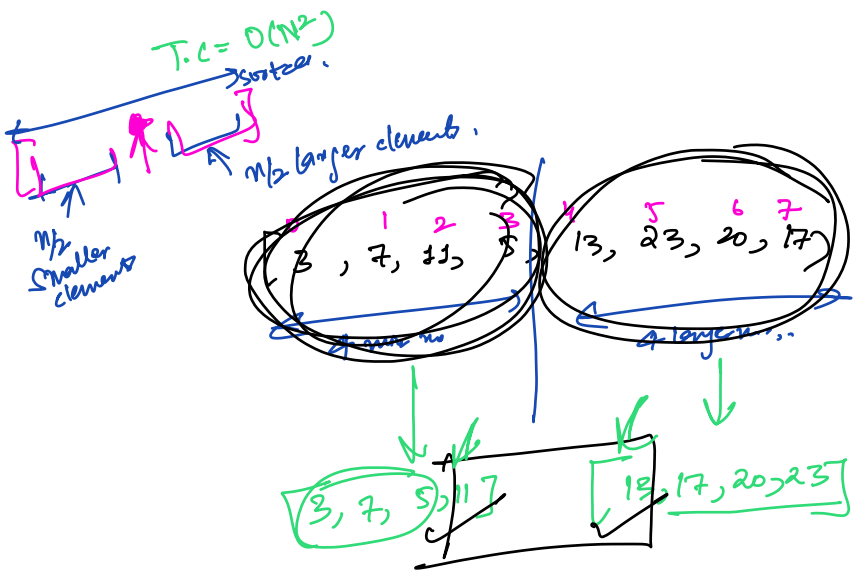


→ Sort Elements every time.
if odd → return middle
if even → return avg(mid2)

T.C = (n log n * n)
= (n² log n)

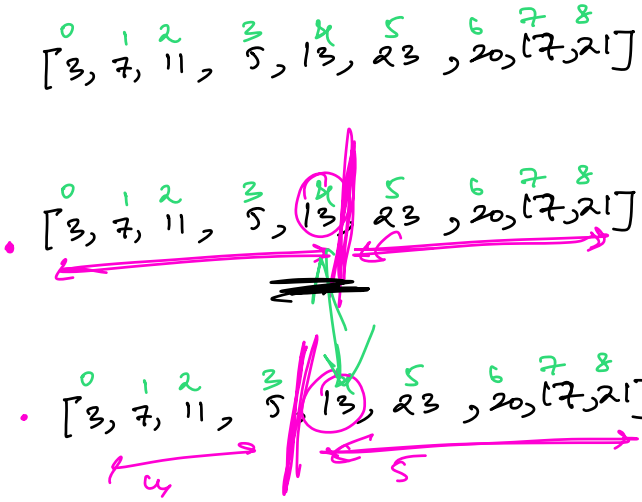


for every element
↳ O(N)



n/2 smaller → need max → maxheap

n/2 larger → need min → minheap.



Steps:

1) Divide the elements into 2 groups.

2) If no. of elements = even

return avg(max(left) , min(right))

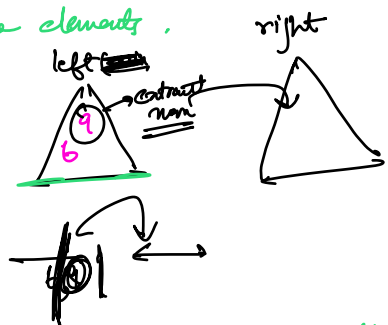
If no. of elements = odd

if left group has 1 extra element

return max(left)

else return min(right)

How to divide the elements.



median (9)

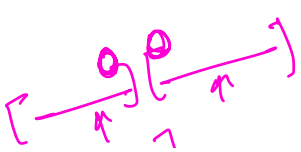
9, 6

if (new_num > median)

go to right

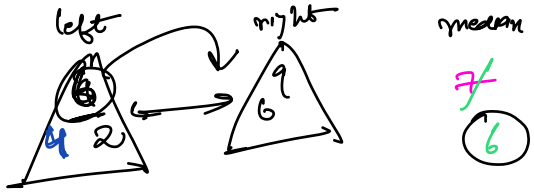
else

go to left.



[ ]

9, 6, 4, 2.



median

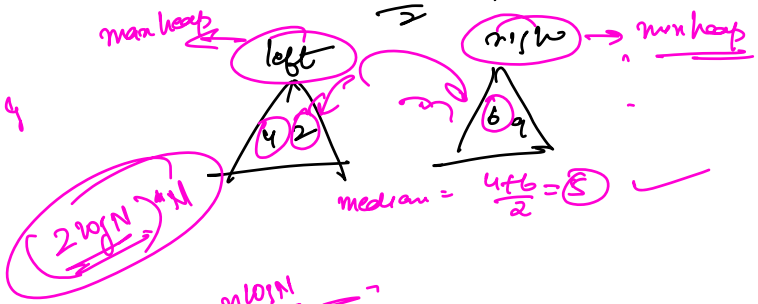
~~7~~

6

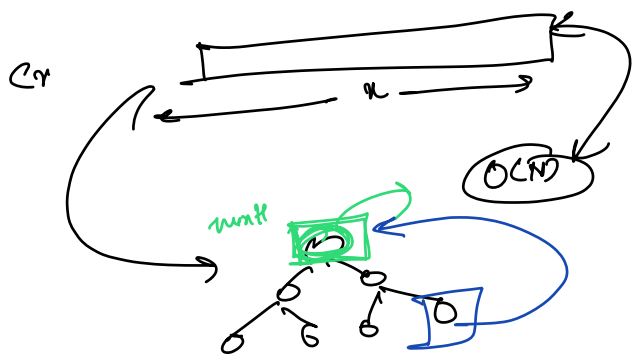
max(left) = 6

min (right) = 9

$\frac{6+9}{2} = 7$



T.C = N log N



List / array

priority - queue → heap