Heaps

① Smallest Range in K-Lists (sorted)

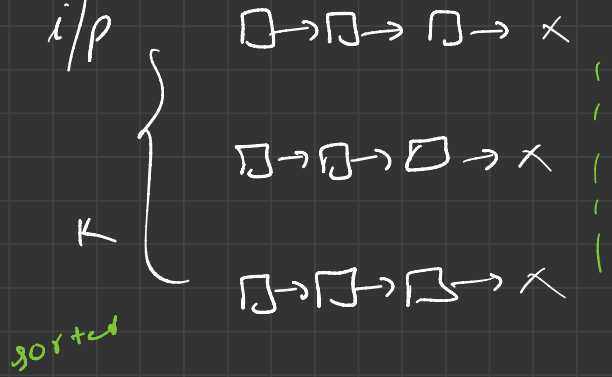
K → int

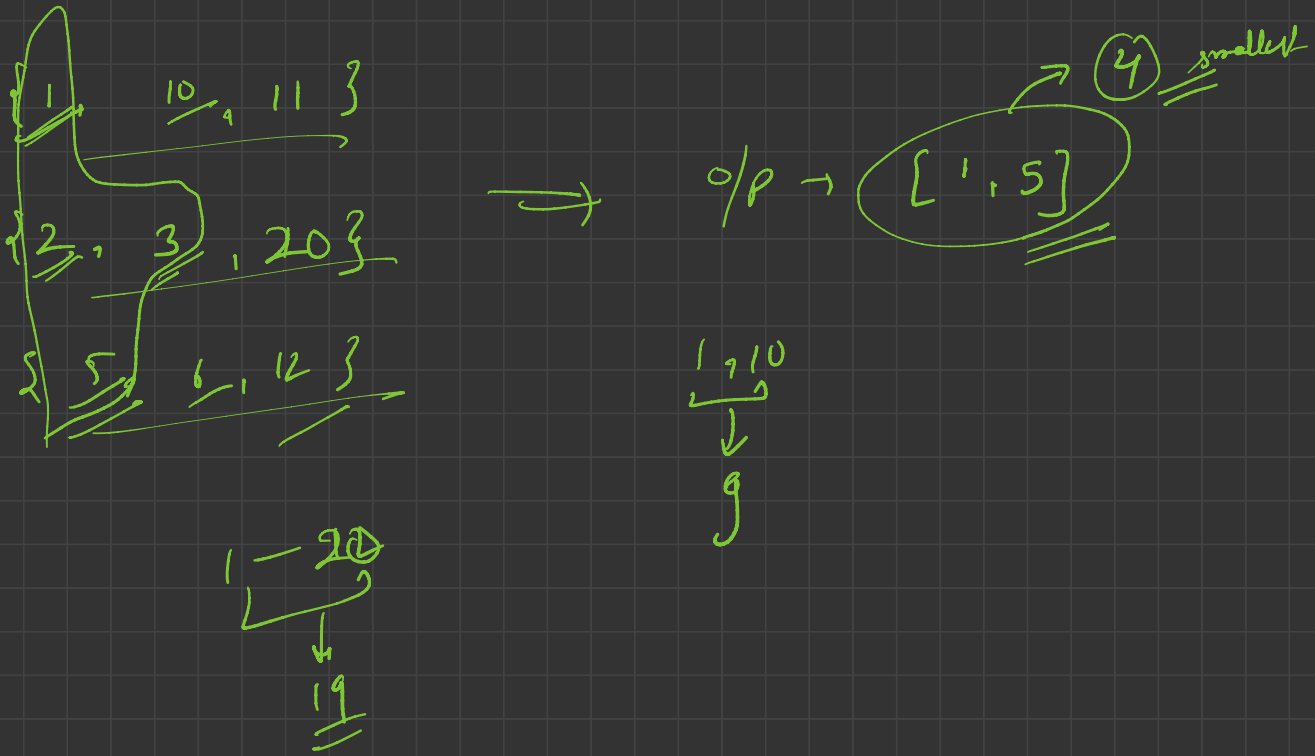
⇒ o/p → smallest range

[x , y]

↓

ha list ka atleast ek element to aa hi jaye




1 10, 11
2, 3, 20
5 5, 6, 12
o/p → [1, 5]
4
smaller
1, 10
9
1 — 20
19

approach:- (Brute force)

#1

$k = 3$

$k \Rightarrow$ { 1, 10, 11 }

$\Rightarrow$ { 2, 3, 20 }

$\Rightarrow$ { 5, 6, 12 }

n

smallest saare $\Rightarrow$ diff $\Rightarrow$ smallest $\Rightarrow$ store

L.E called 1 time

total no of diff = $n+k \Rightarrow N$

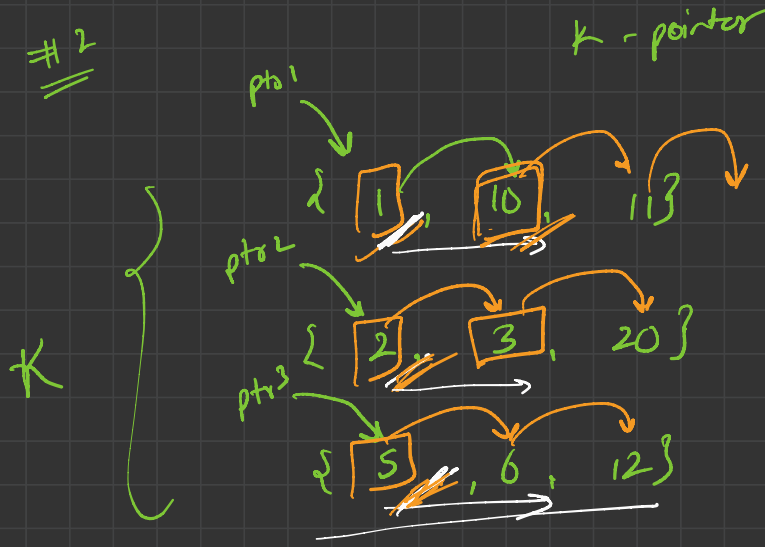
$O(N^2) = O(n^2k^2)$

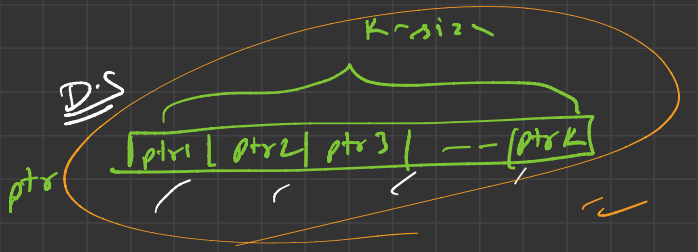
$O(n^2k^2)$

{ 1, 2, 3, 4 } $\rightarrow O(n^2)$

#2

K - pointer





→ min / max

min = 1 , max = 5

satisfy

[1, 5]

→ smallest range & effect ition

10, 2, 5

ans

[1, 5] → 4

option

min → increase

max → decrease → ✗

3, 10, 5

min = 3

max = 10

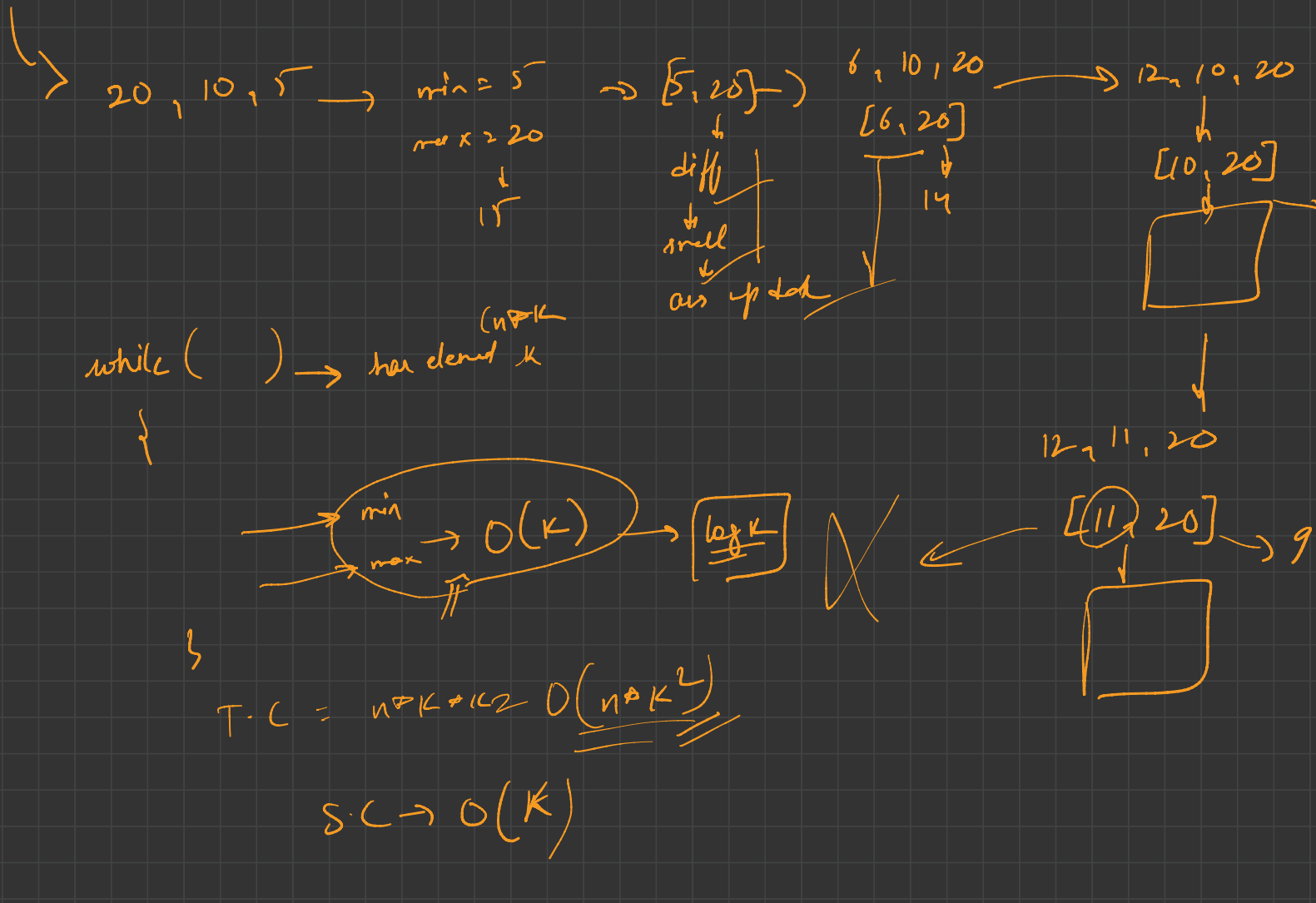
[3, 10] → diff → small → store

min increase

min = 2

max = 10

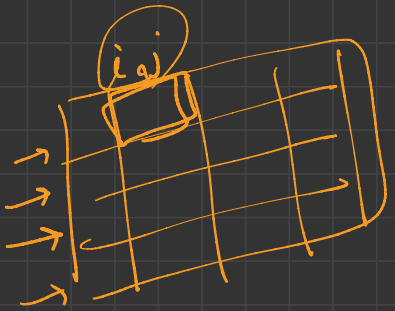
[2, 10] → diff → 8 → smallest → stop

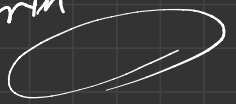

20, 10, 5
min = 5
max = 20
15
[5, 20]
diff
small
as update
6, 10, 20
[6, 20]
14
12, 10, 20
[10, 20]
10
12, 11, 20
[11, 20]
9
while ( ) → har element
min
max
O(K)
log K
T.C = n*K*log2 O(n*K^2)
S.C → O(K)

#3 Heaps

i/p ⇒ { 2, 3, 20 }

⇒ { 5, 6, 12 }





Algo:-

0 → min / max →

1 → min-Heap → K size

↳ list → starting element

2 ⇒ doubt hua

↳ aapne yaar track kiyaa

int arrStart, arrEnd;

range = arrEnd - arrStart

while (! pq.empty)

{

min = pq.top() → element, row, col

pq.pop();

ans update → if (max - min < ansEnd - ansStart)

{ update ans

}

max - update → if (min.col < n)

next element exist

{

maxi = max (maxi , ans [mini.row] [mini.colH])

pq.push (new node (==== min row ====));

}

[5, 5) → 1

5 - 5 + 1 = 1

[3 - 9] → (9 - 3 + 1)

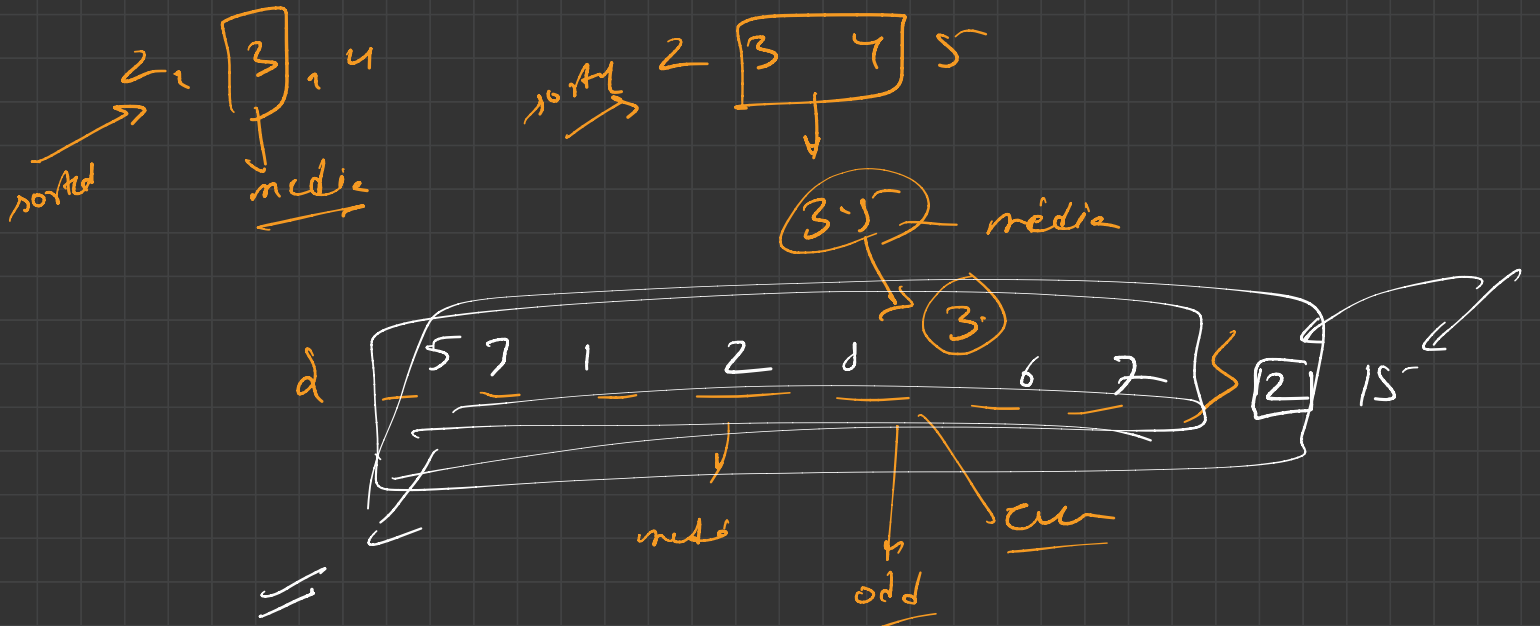
(7) (7)

dir

{ break;

}

→ Median in a Stream



Approach #1 B/I/S/Q/M/H → 6 algo

→ Insertion sort → $O(n^2)$

Insertion sort → $O(n^2)$

$O(n^2)$

→ $O(n^2)$

→ median → odd → middle → $ans\left(\frac{n}{2}\right)$

→ $O(1)$

even → $\frac{ans\left(\frac{n-1}{2}\right) + ans\left(\frac{n}{2}\right)}{2}$

$n=6$

2 | 3 | 1 | 7 | 9 | 6

2 3

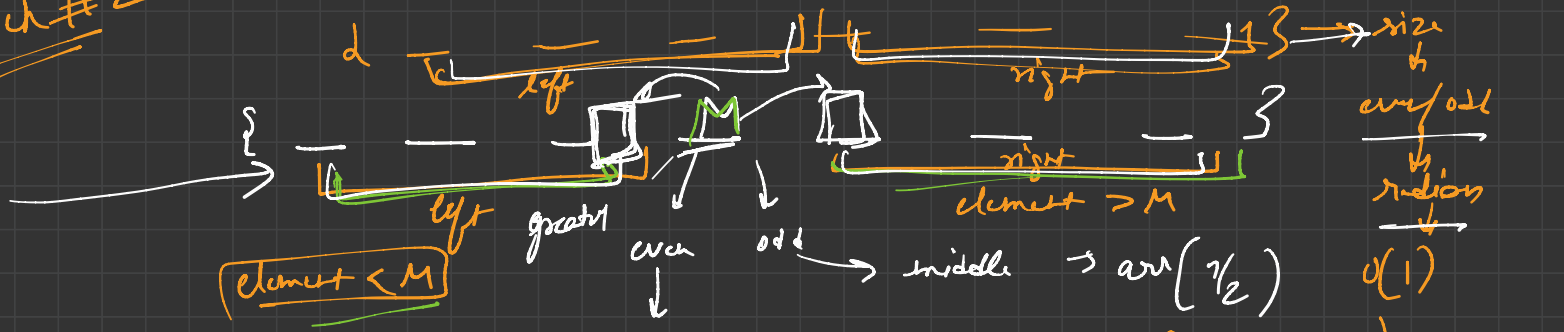
{1, 2, [4], 5, 6, 7, 9}

4

B.S → right → $\log n$

→ $O(n)$

Approach #2



$$\frac{arr\left(\frac{n-1}{2}\right) + arr\left(\frac{n}{2}\right)}{2}$$

Odd size → Case 0: left $n-1$, right n or Case 1: left n, right $n-1$

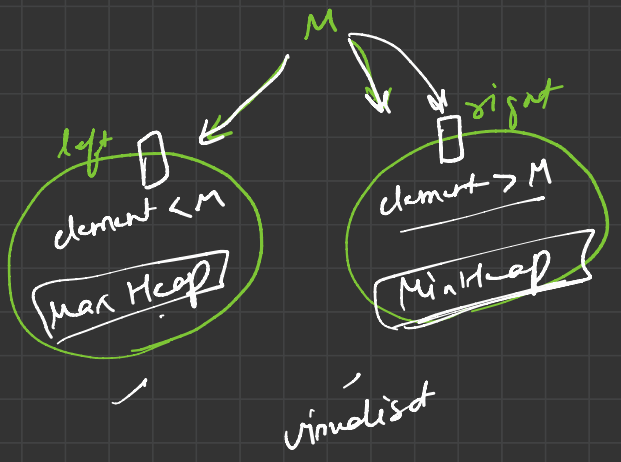
Even size → Case 2: left n size, right n size

signum (a , b)

a == b , → 0

a > b , → 1

a < b → -1



visualise

Algo:-

For (i=0 ⟶ n)

{

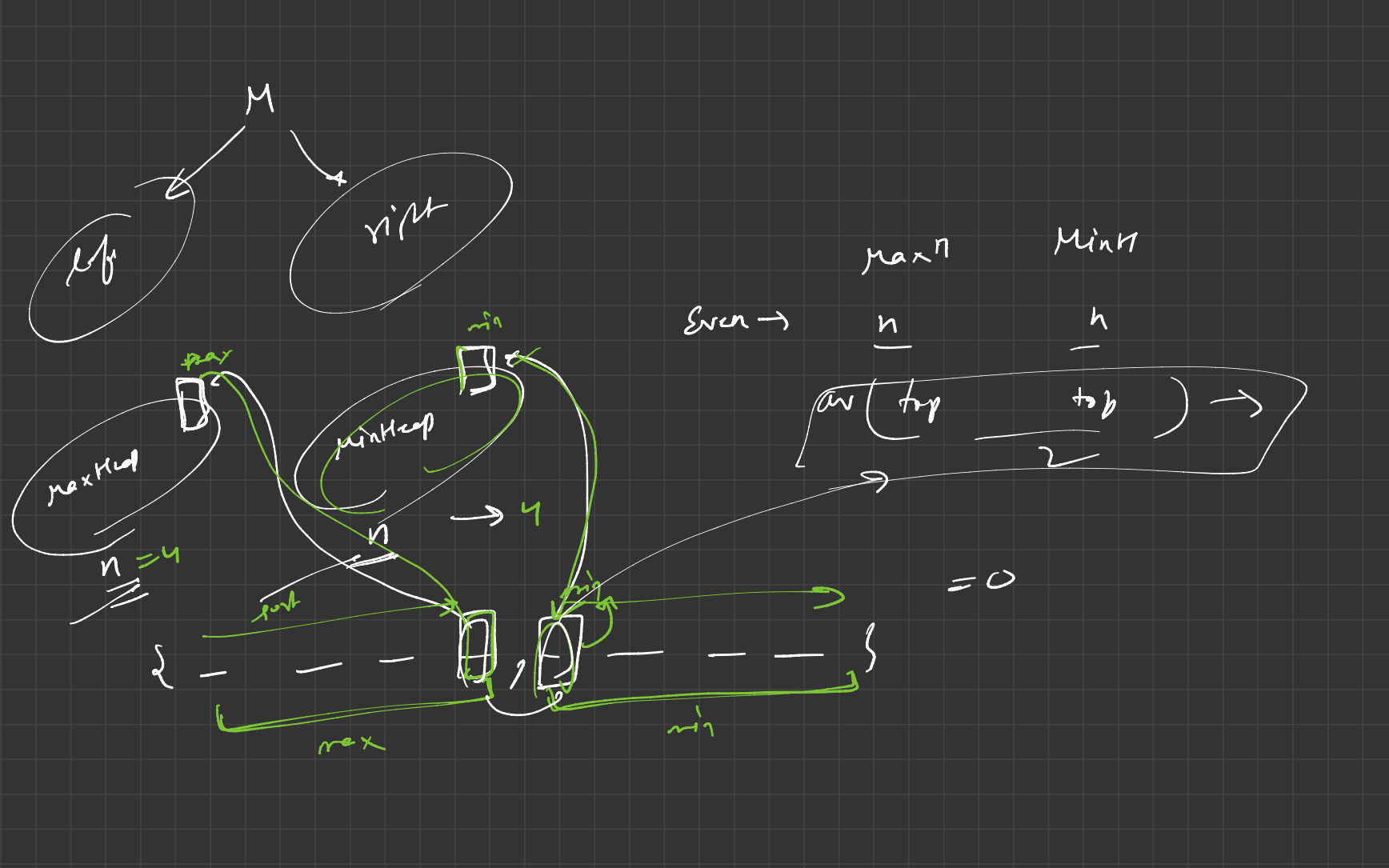
element = arr[i]

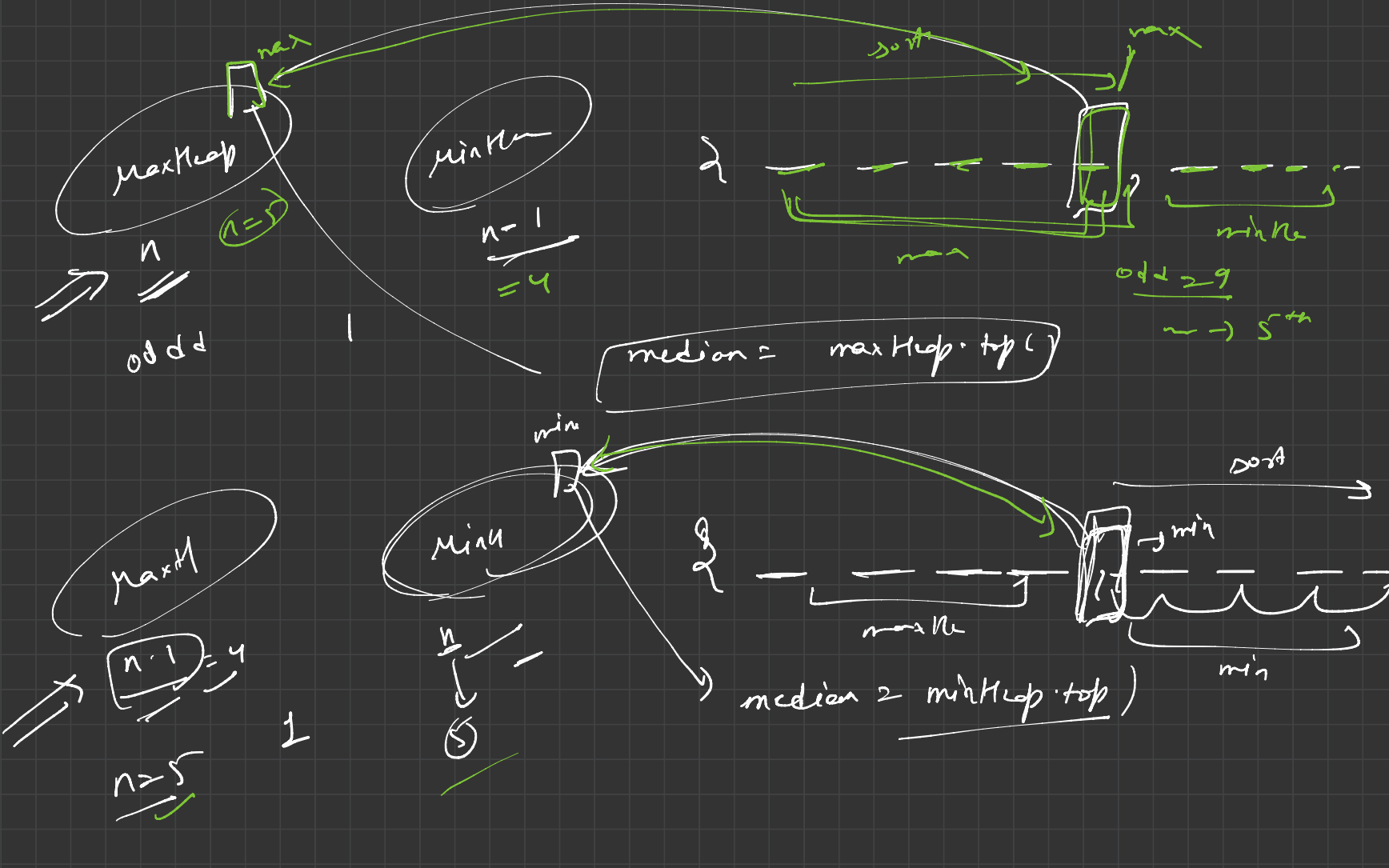
ans = call_median (element , maxHeap , minHeap)

vector . push (ans)

}

return vector;


M
left
right
maxHeap
minHeap
max
min
n
4
n = 4
root
max
min
maxH
MinH
Even →
n/2
h/2
av(top, top)
= 0


max
maxHeap
minHeap
n=5
n
odd
1
n-1
=4
sort
max
2
maxHe
minHe
odd=9
→ 5th
median = maxHeap.top()
min
minH
maxH
n
5
median = minHeap.top()
sort
min
maxHe
minHe

call _median (elem, _max, min, &median) by ref

{

switch (signum (maxHeap.size(), minHeap.size()))

{

case 0: if (element > M)

{ minHeap.push (element)

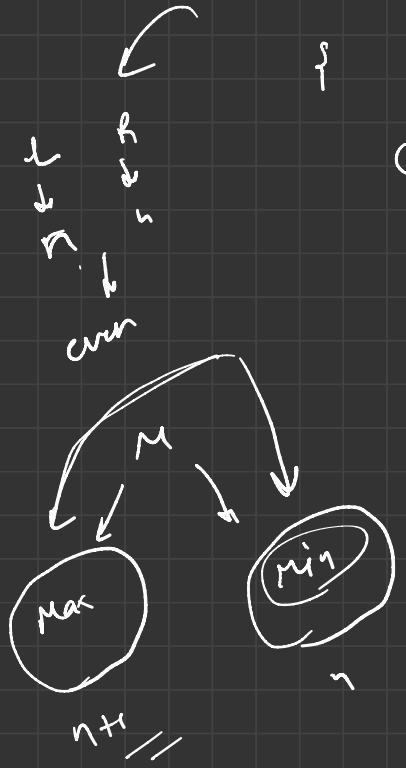
median = minHeap.top() }

else

{ maxHeap.push (element

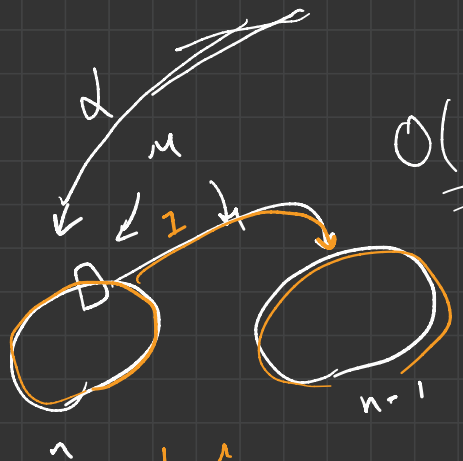
median = maxHeap.top()

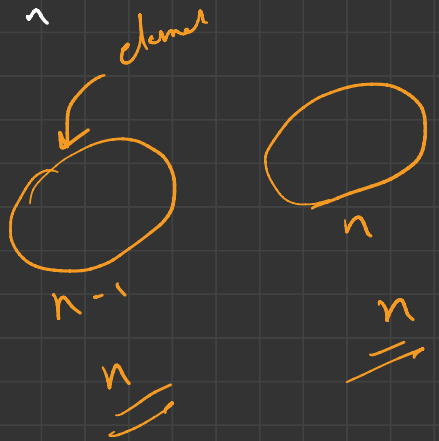
}



case 1:

$O(n \log n)$






```
if (element > M)
{
    minHeap.push(element)
    median = avg(maxH.top(), minHeap.top())
}
else
{
    minHeap.push(maxHeap.top())
    maxHeap.pop()
    maxHeap.push(element
    // copy
}
```

a < b

case -1 :

if (element > M)

{

maxHeap.push(minHeap.top())

minHeap.pop()

minHeap.push(element)

Median = avg(minTop, maxTop);

}

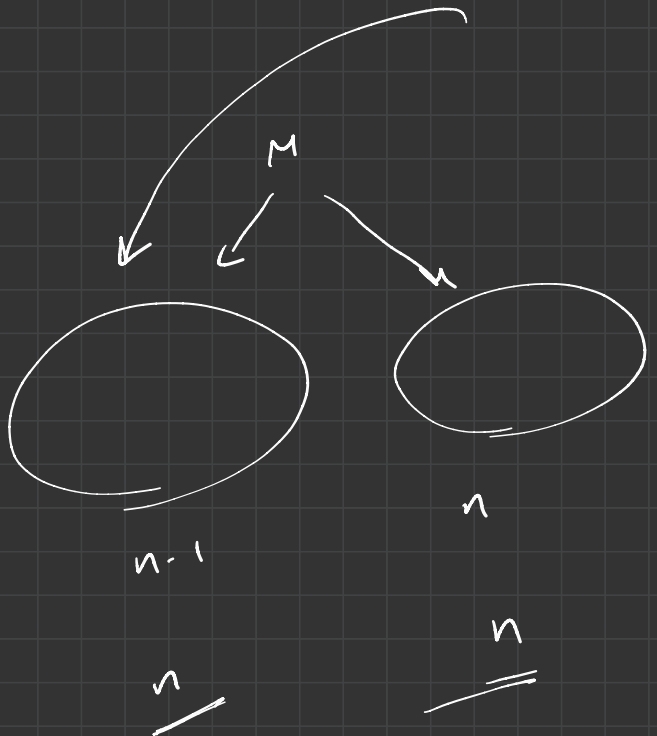
else

{

maxHeap.push(element)

// com

}



T.C → O(N log N)

Section 1