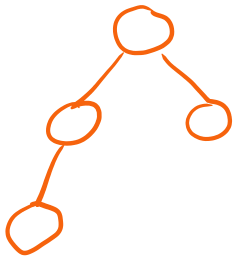
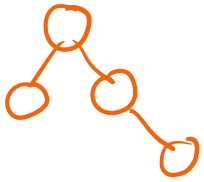
→ Heap → D.S.
↳ complet Binary Tree.
↳ that holds a Heap propurity

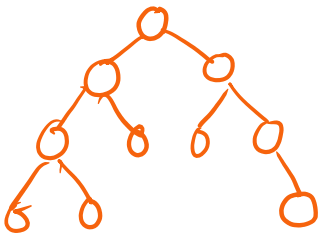
CBT
↳ all levels are completly filled (except last one)
(leaf → right build)



CBT



not CBT

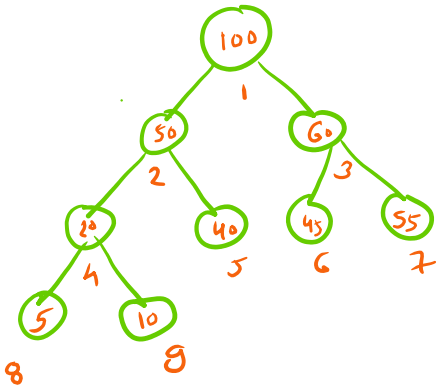


not CBT

Creation in array from.
but visulation is tree from.

Max-Heap → Parent are both child che max

Min-Heap → Parent are both child che min.

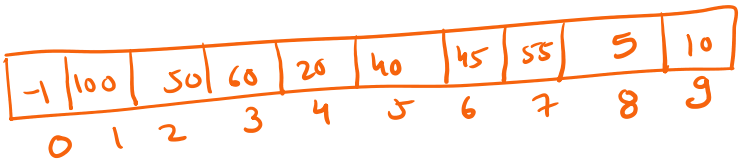


1 se start.
parent = i
left child index = 2*i
Right child index = 2*i+1
child → parent.
index = i/2

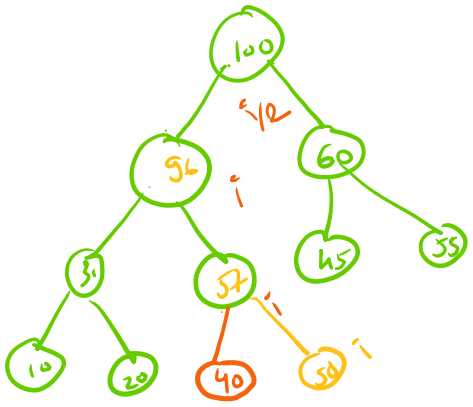
0 se start
i
left = 2*i+1
right = 2*i+2

using Arrays

Using Arrays



Insertion



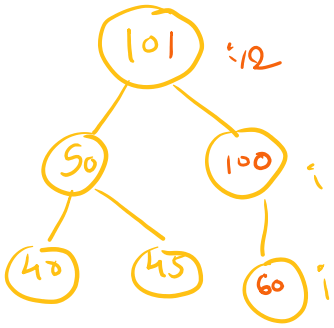
new Node = 57

new Node = 96

T.C = O(logn)

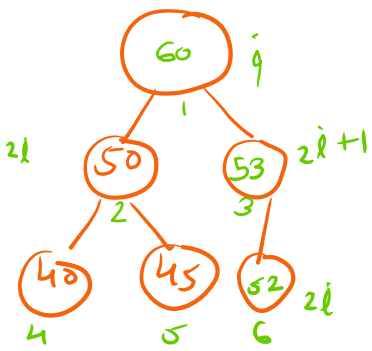
(1) Add node at the end of array

(2) Sahi place pah lekar jao



-1, 100, 50, 60, 40, 45

Deletion



Root node only delete

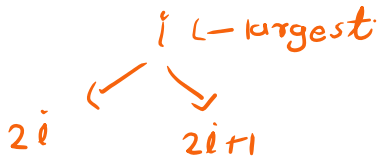
T.C = O(2logn)

1. Replace last value with root node.
2. Root node ko right (correct) place par replace.

heapify :- Node ko sahi place par place karna

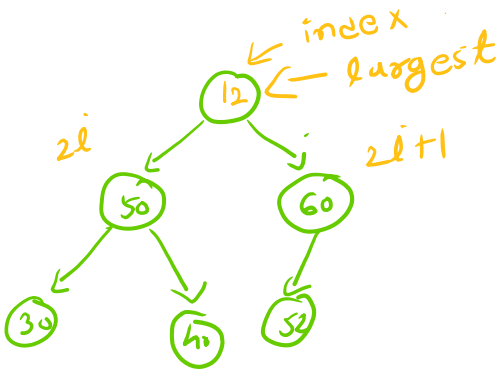
any error use convert to Heap this function using

Hepify Hepify(arr, size, index)



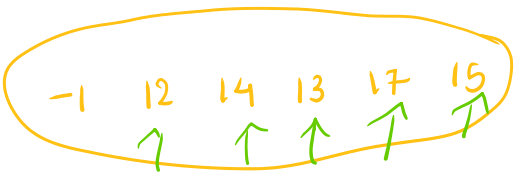
i => largest

i != largest
swap
i = largest
↳ hepify



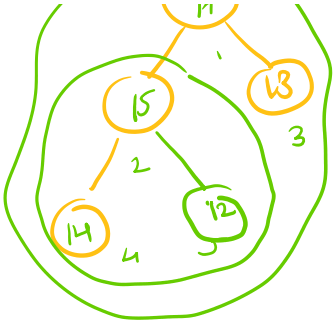
T.C = O(logn)

Build Heap



↳ Heap convert

leaf Node



leaf Node
↳ hepify.
↳ No Need.

$\left(\frac{n}{2}+1\right) \to n$ → all node are leaf node.

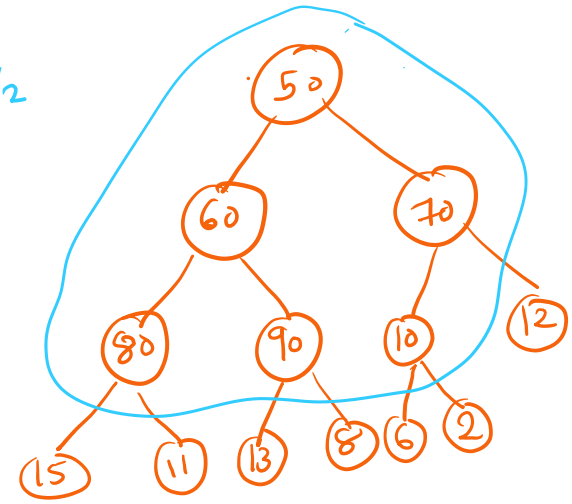
$\left(\frac{5}{2}+1\right) \to 5$

$3 \to 5$

$\frac{n}{2} \to > 0$ hepify

no need to hepify ↓
7 → 13 leaf node

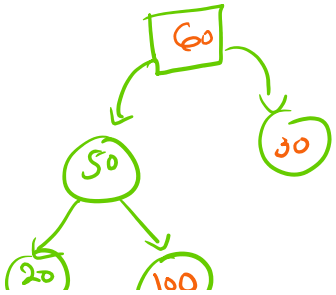
$1 \to n/2$

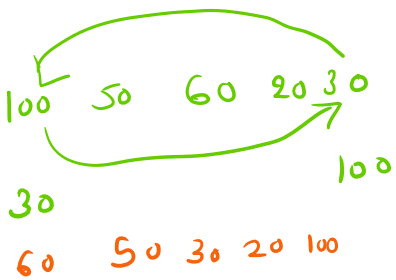


T.C = O(n)

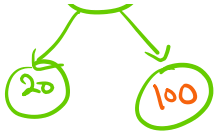
buildheap
for ($\frac{n}{2} \to > 0$)
{
hepify( )
}

Heap Sort T.C = O(nlogn)





30
60 50 30 20 100



(1) First element / last element:
swap

(2) root node ko sahi place par replace
karna (heapify)

Priority-queue
↳ Heap behaviour

→ k^{th} smallest no :-

i/p arr → | 3 | 6 | 7 | 12 | 11 | 4 | 2 | 8 | — n size.

#1 arr sort

sort(arr, arr+1)
↳ k^{th} ele → arr[k-1]

T.C = n log n

#2 min-Heap

n size → heap create
(n-k) time pop

S.C = O(n)

T.C = O(n)

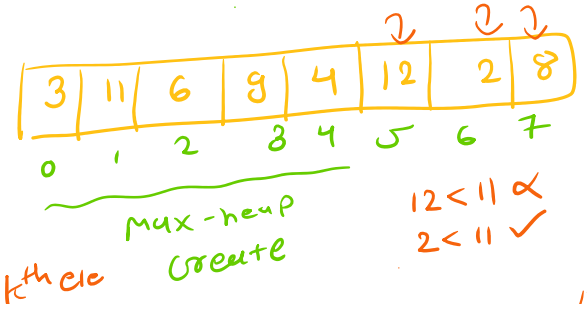
1^{st} pop → 1^{st} small ele
2^{st} pop → 2^{st} small ele
⋮
k^{th} pop → k^{th} small ele.

#3 max Heap

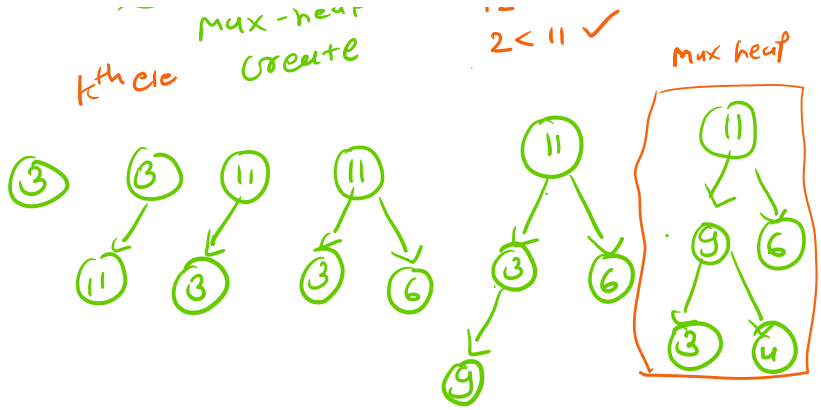
first k ele → max heap

insert →
new element < heap.top()
heap.pop()
heap.insert(new element)

max heap
↳ k-size
↳ k small element
↳ top → k^{th} small element

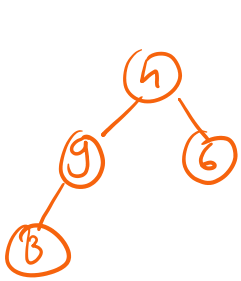


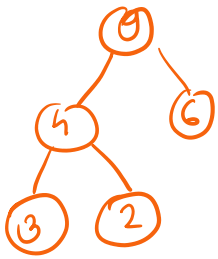
k^{th} ele



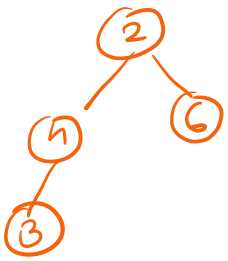
2 < 11

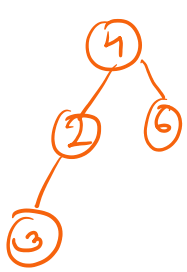
11 delete 11 pop

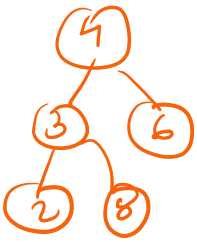




8 < 9 ✓





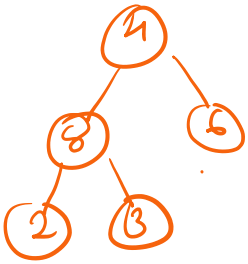


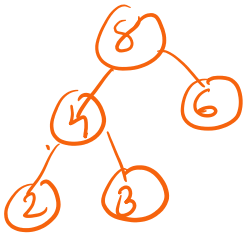
algo :-

(1) k^{th} element create a max heap

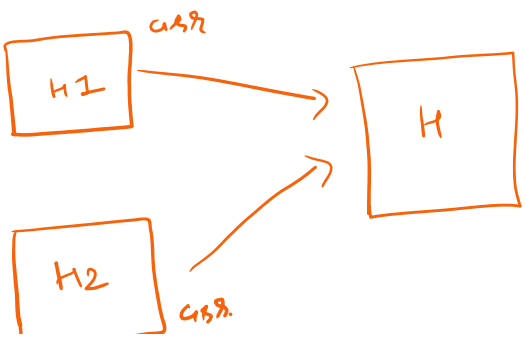
(2) another element compare the heap top element

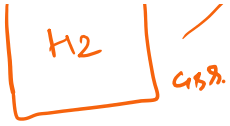
```
if ( element < heap.top() )
 {   delete heap.top()
     insert (element)
 }
```





→ Merge 2 max-heap





H1 heap

T.C O(nlogn) H2 insert in H1

(1) arr → Merge(h1, h2) → O(m+n)

Build heap O(n)

T.C O(m+n)

(2) h1 ✓

h2 each element insert h1

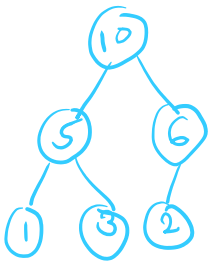
T.C O(nlogn)

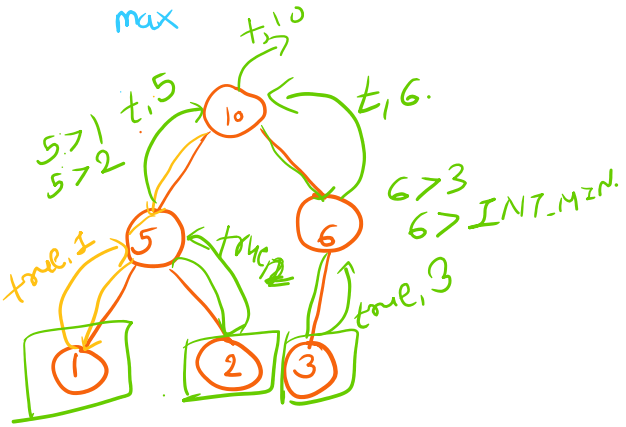
Complete.

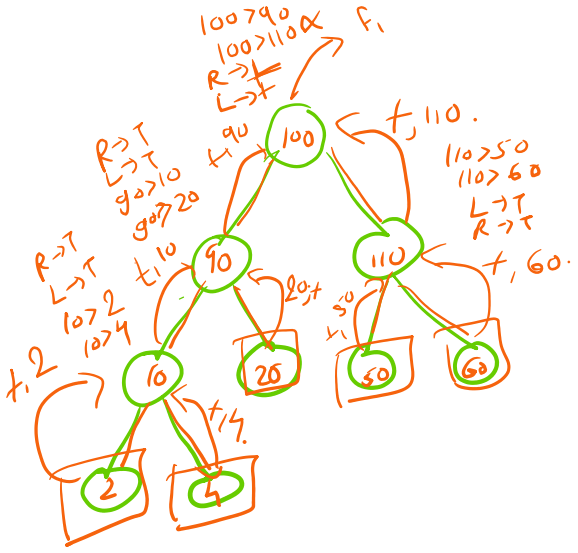
→ Binary Tree

↳ Heap or not

max







for C)

{ //B.C

if(L.N)

return { true, root->data).

if(== null)

return {true, INT_MIN)

leftAns -> Rec

Right Ans ->Rec

(9) CBT

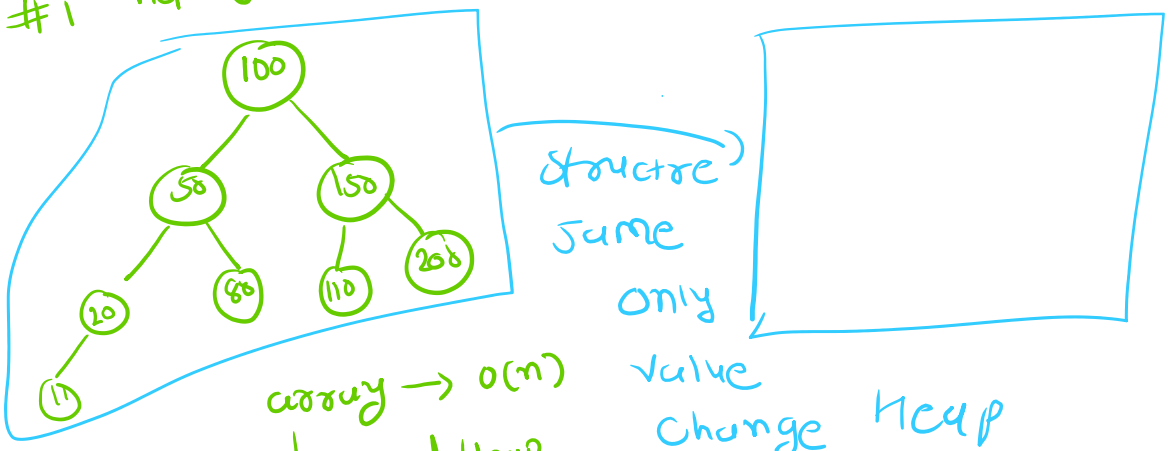
-> BST ⟶ convert Max Heap

L> CBT

L> Max heap

property.

#1 hepify

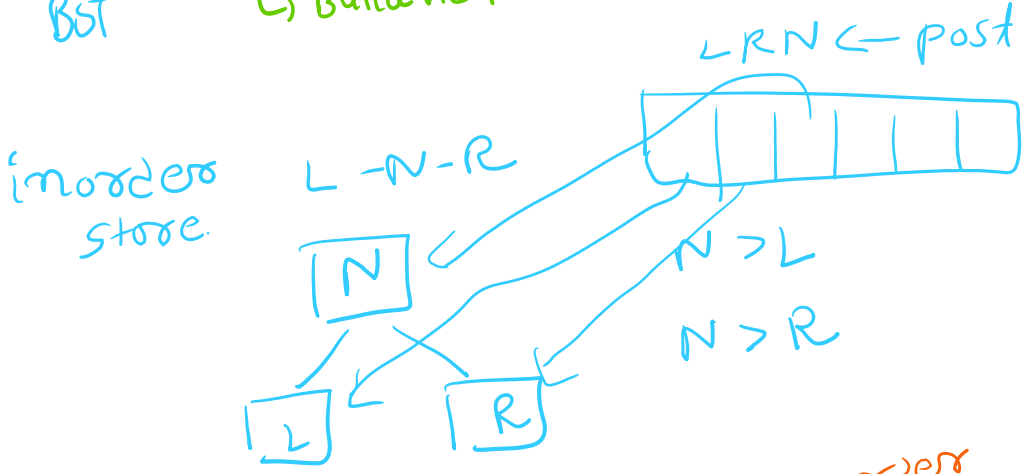


BST

array → O(n)

L> Build Heap

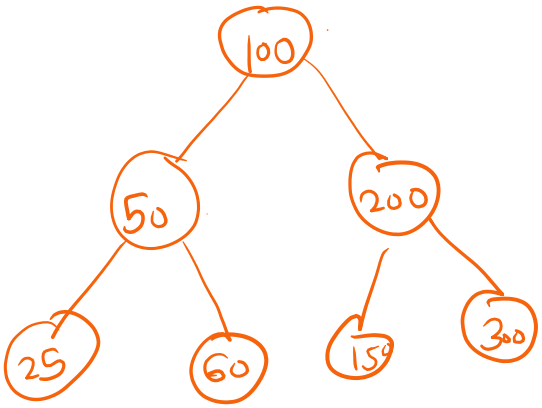
Heap

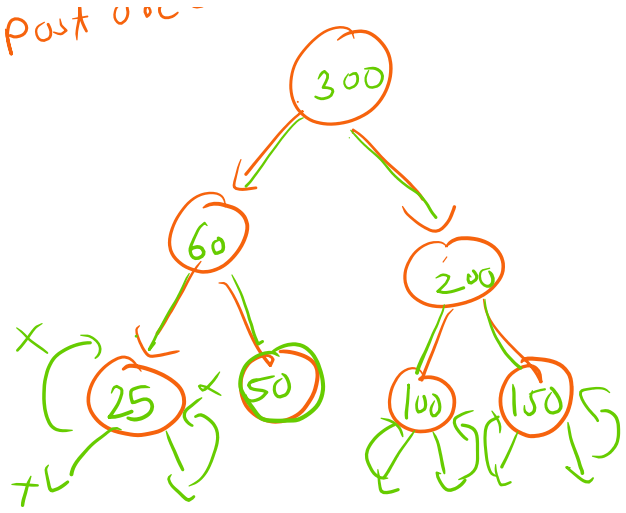


insert

post order

300

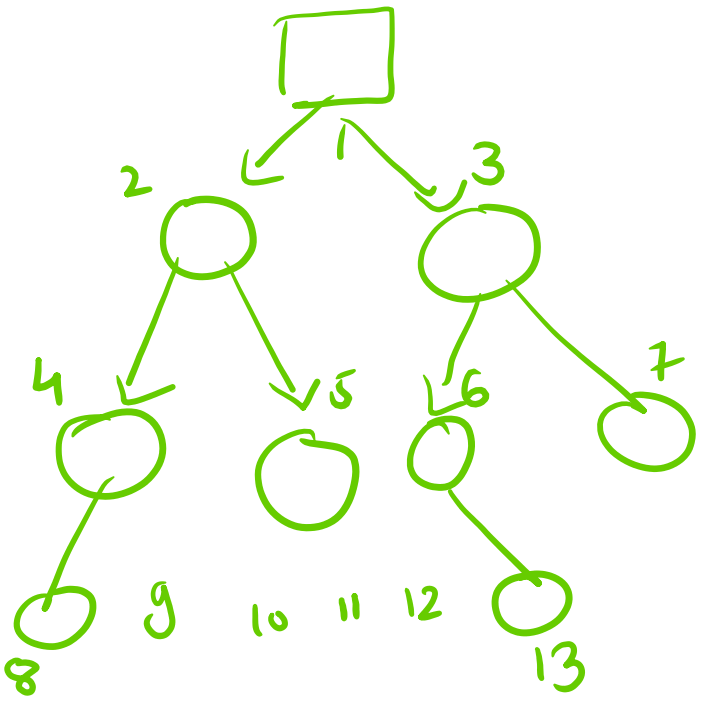




inorder

25 50 60 100 150 200 300

Check tree → is CBT or not



#1 level order

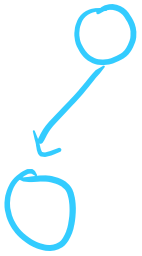
#2 Total node = 9

13 > 9

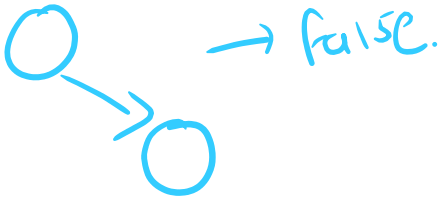
∴ not CBT.

I

II



III

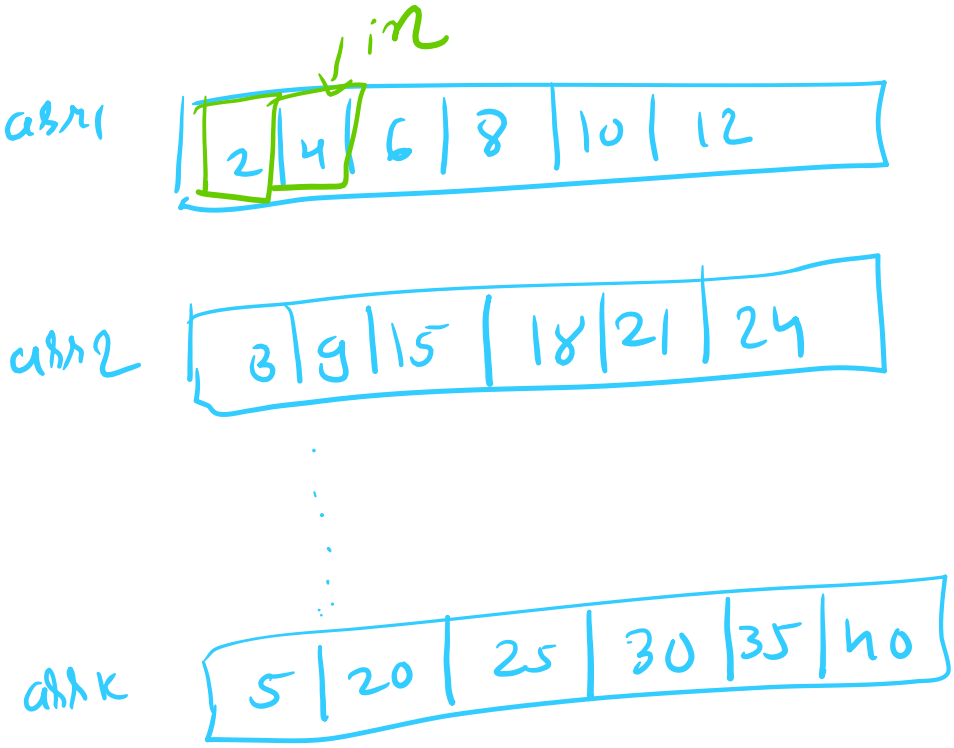


IV

04 April 2024 14:48

***

Merge k-sorted array.



Move singal arr. T.C = nk log(nk)

sort

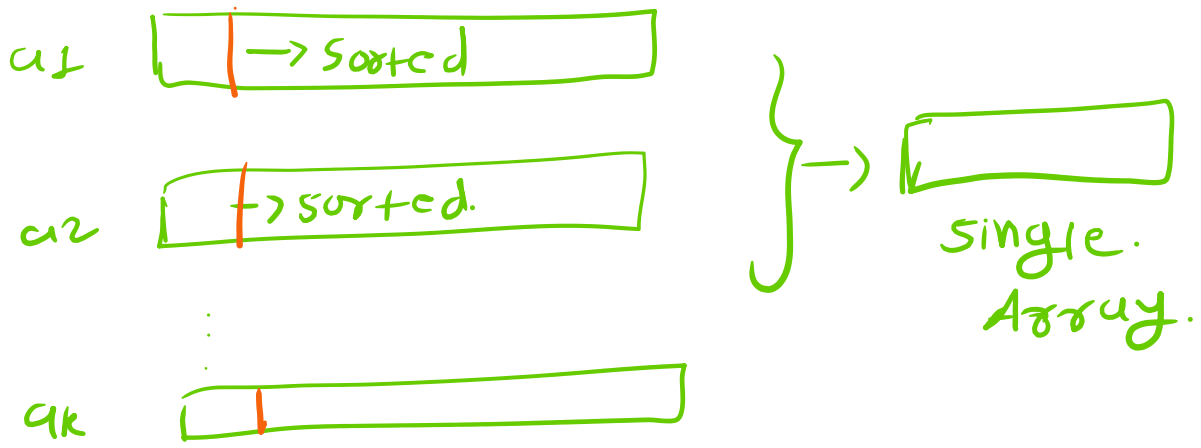
#

Min-Heap-create → all arr in first element

ð top पर इस को insert then delete. then that arr next

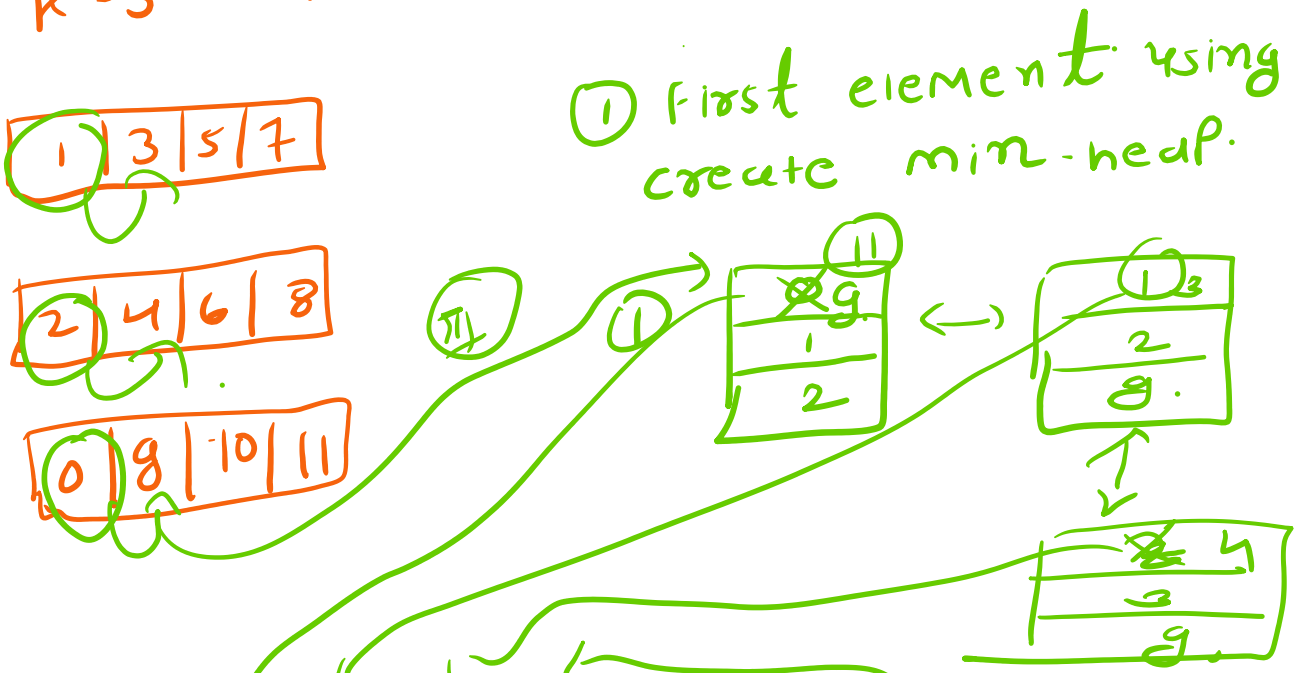
delete. then that arr next
element insert.

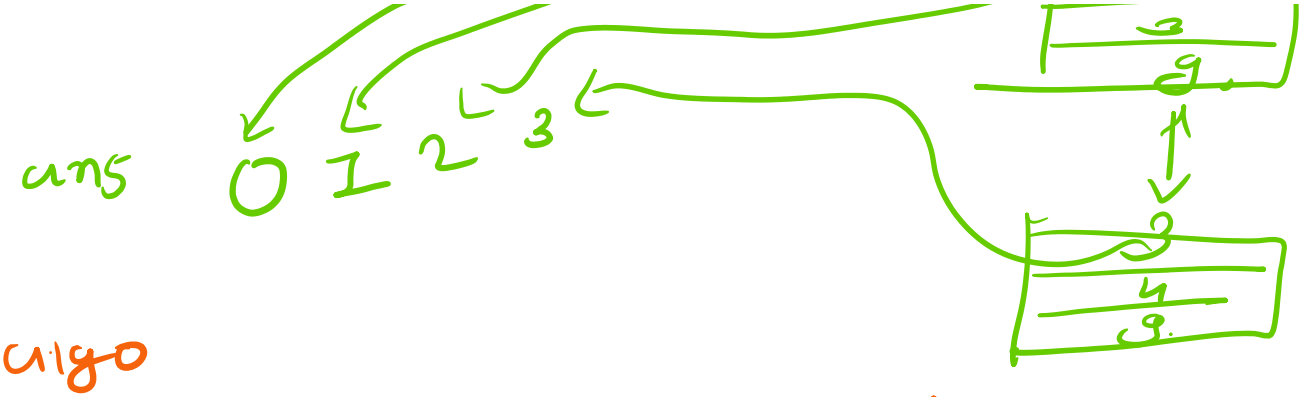
Merge k Sorted Array.



k-size min-heap -> min number.
min number - push - single array.
min number next number insert in heap

k = 3 N = 4





algo

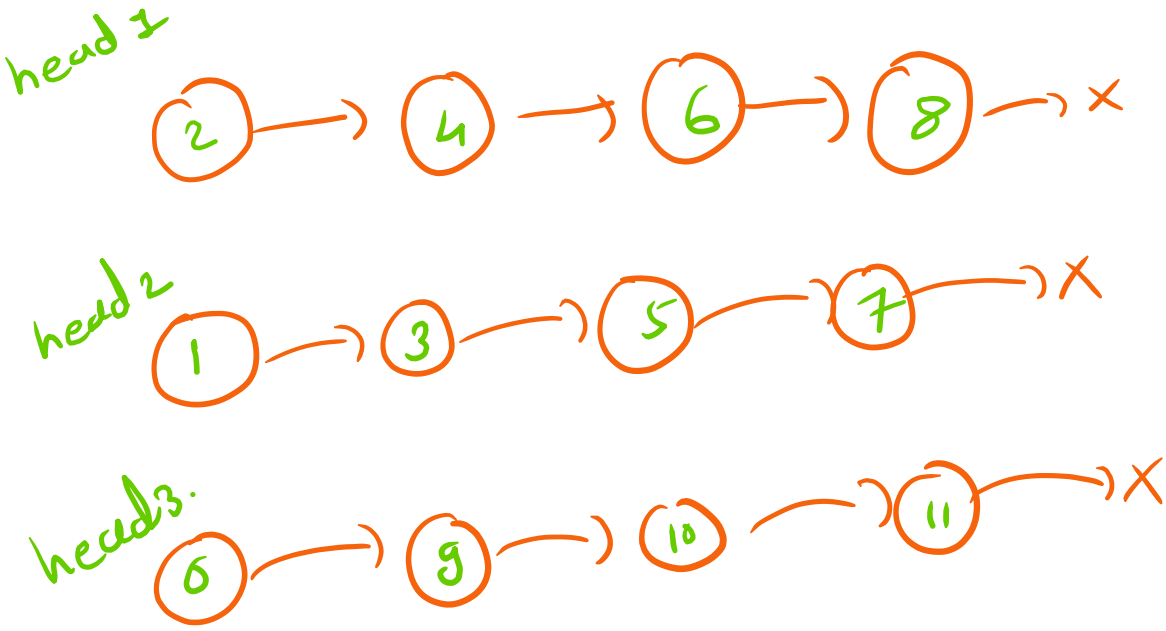
(1) all array first element create min-heap

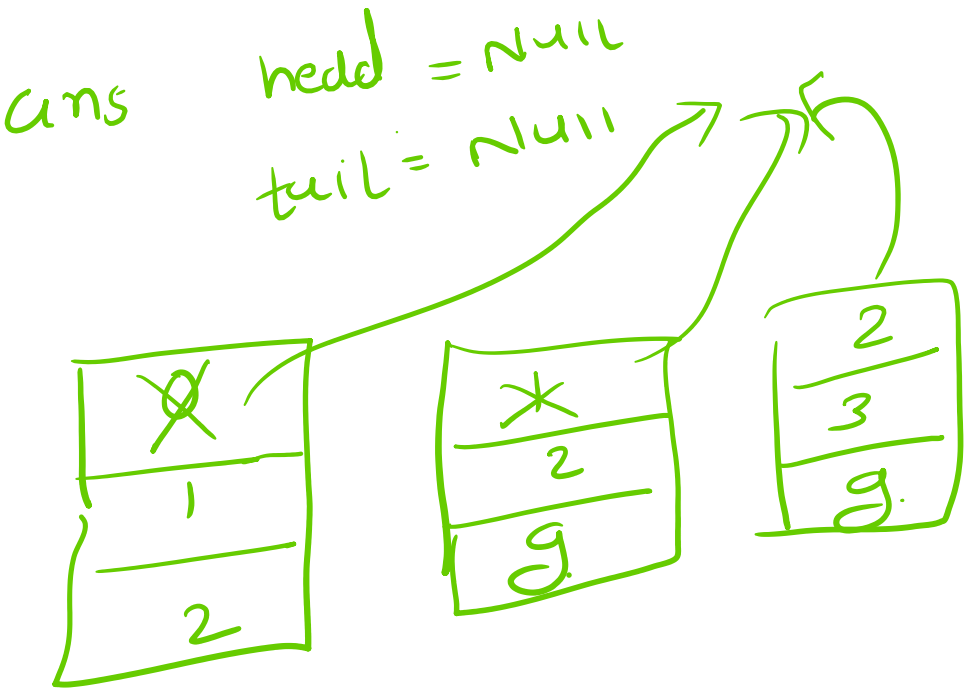
(2) min-heap top ele push ans array. and delete the. top ele.

(3) push element next element push in heap.

T.C = O(nk log k)

28. → Merge k Sorted linked list

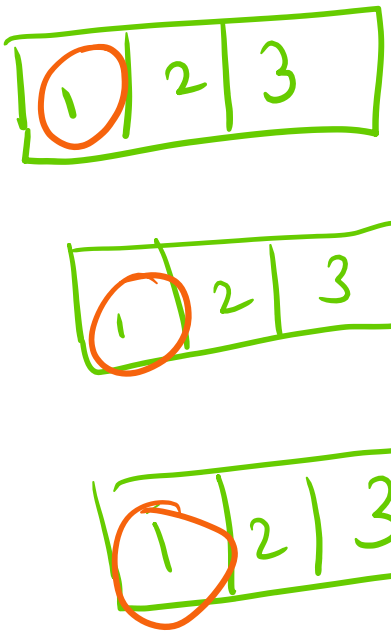




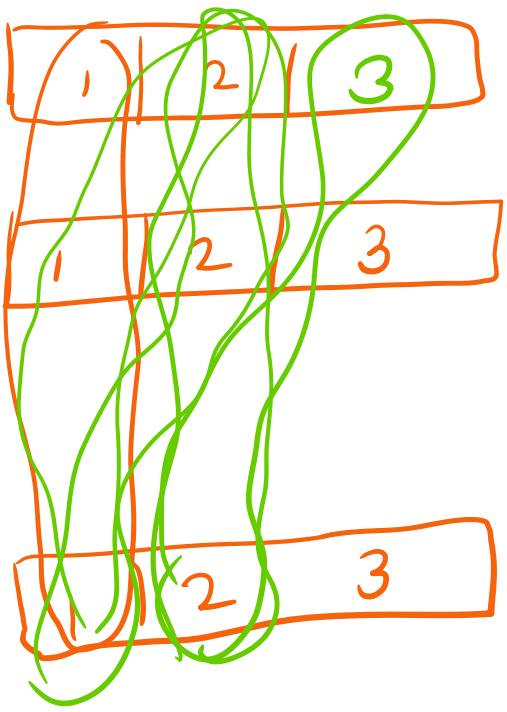
05 April 2024 11:36

632 Imp

→ Smallest range in k List



Max - Min
↳ small



Min = 1
Max = 2 — 1

Min = 1
Max = 2 = 1

Min = 2 = 0
Max = 2

Min = 2
= 1

min - -

max = 3 = 1

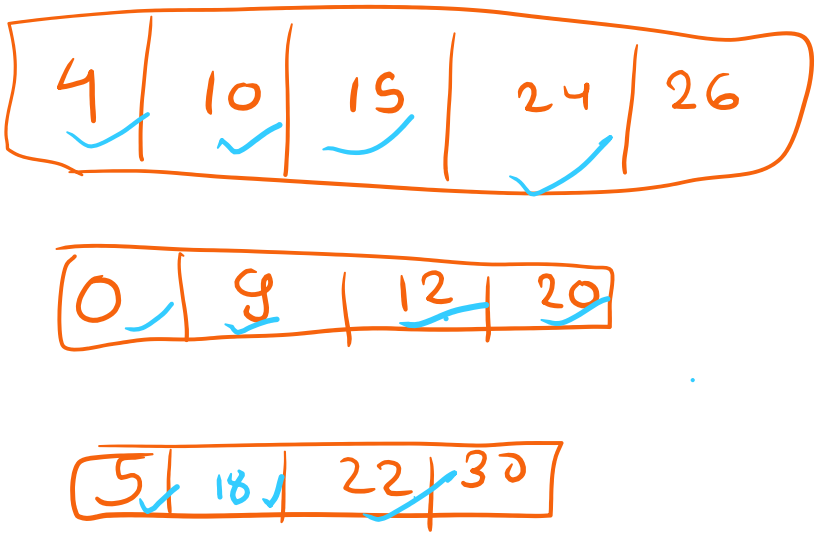
if any list finised then took day

smallest range.

↳ min - max.

min ↑ min Heap used

Max ↓



0 → 5

4 → 9

5 → 16

9 → 18

10 → 18

12 → 18.

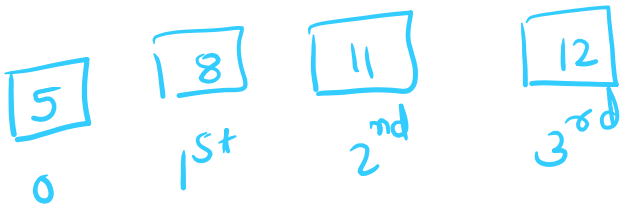
15 → 20.

18 → 24.

20 → 24.

1962 Remove stones to minimize the total

(K) → operation



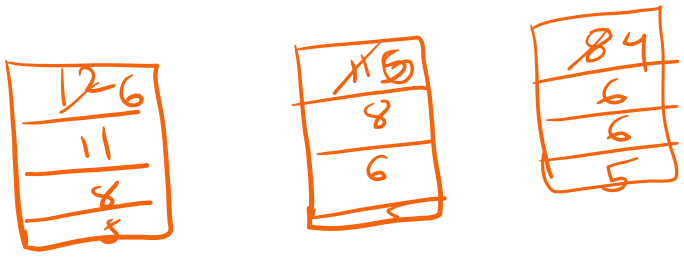
total - floor(total/2)

Min stones after k options

Max heap

↳ k operation

k=3

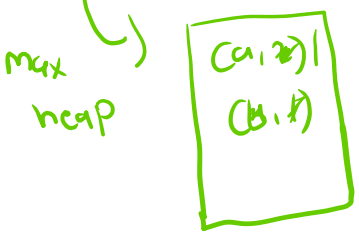


STL: heapify()

767 ReOrgenize String

Min cost to cut ropes.

"aab" → a-2, b-1

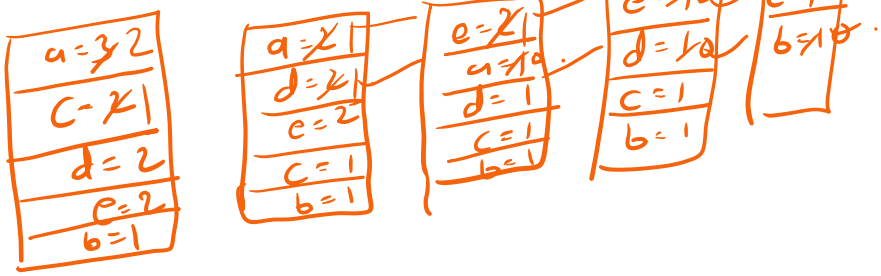


ans = "" + a + b + a

aaabcc dd ee

two element together ans Nikalna.

a-3
b-1
c-2
d-2
e-2



ans = "" + a + c + a + d + e + a + e + d + c + b

1405 Longest Happy String

a=0 b=8 c=11

c=11 7 5 3 2 1
b=8 6 4 2 0

ans = cc + bb + cc + bb + cc + bb + cc + bb + cc

cc + b + cc + bb + cc + bb + cc + bb + cc + b c

Median in a Stream

data stream

2, 3, 5, 1, 4, 9, 8 . . . . .

↓ ↓ ↓

Med Med Med.

Median

2 3 5 10 11 12 13 odd

↑

Median.

2 3 [4 5] 6 7 even

↑

Avg.

2, 20, 11, 4, 8, 25 . . .

X #1 Vector → Push → sort → Middle ele

$O(n^2 \log n)$ ↳ Median

X #2 BST → no - BST create → logn.

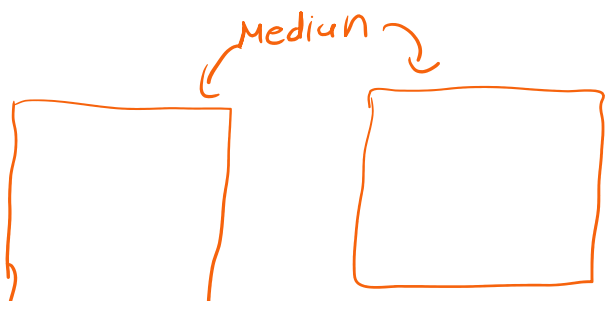
$O(n^2)$ mid. ← inorder.

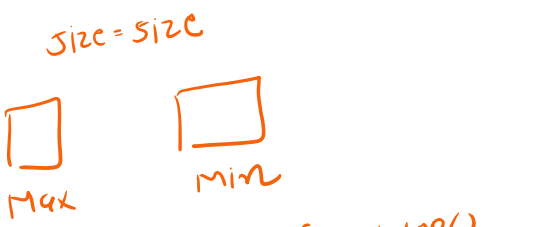
#3 Heap

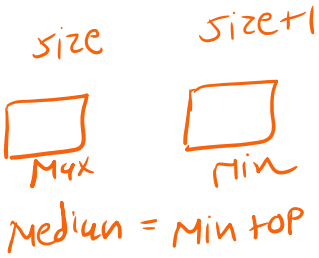
↳ Max-Heap

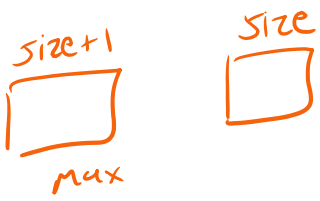
↳ n/2 pop. X

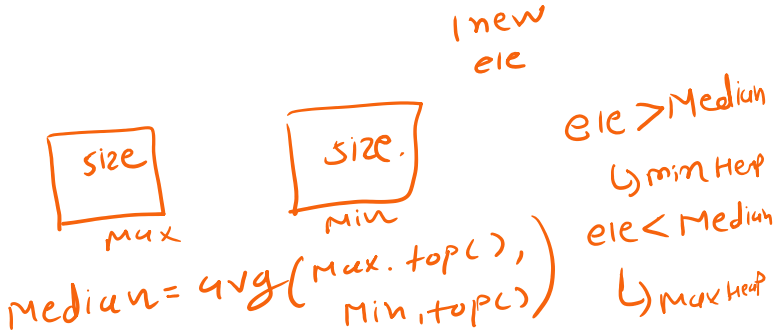
Min + Max Heap popX

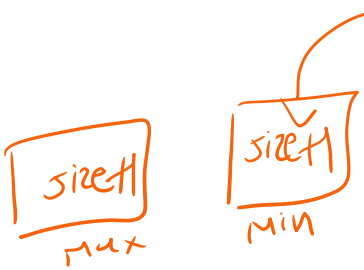


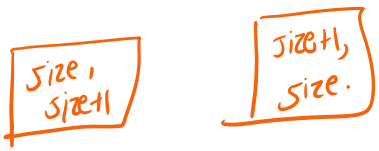








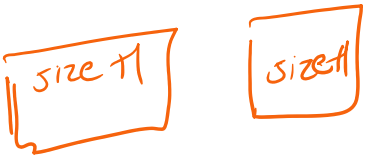




one element pop on min and that element push in Max Heap

ele < Median.

↳ left.



ele

ele > Medi

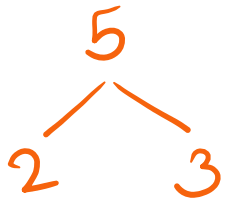
↳ right.

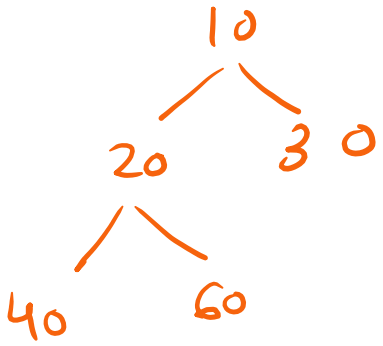
ele < Median

↳ left.

pop element of Maxheap pop and that element push on Minheap

gfg check if Binary Tree is Heap



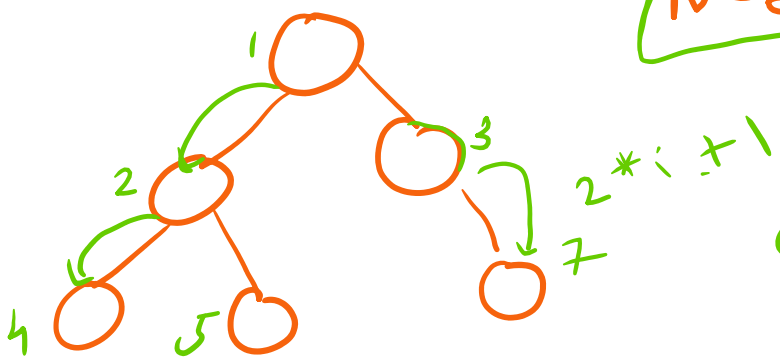


(1) is complete BT

(2) does it satisfy max heap order

How to check CBT?

- total No. of Nodes



child/node index > N total No of Node.

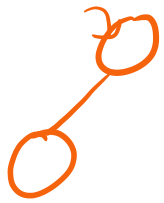
7 > 6

for every no. check Max order.

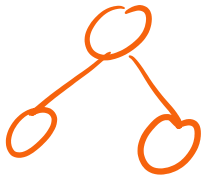
parent value > children

parent value > children value.

case 1 : leaf node ✓

case 2 :  left child only

parent > child ✓

case 3 :  both child available

parent > LL

> RL

05 April 2024 18:29

gfg Merge two Binary Max Heap

a → 10, 5, 6, 2

b → 12, 7, 9

① Merge two heap vector

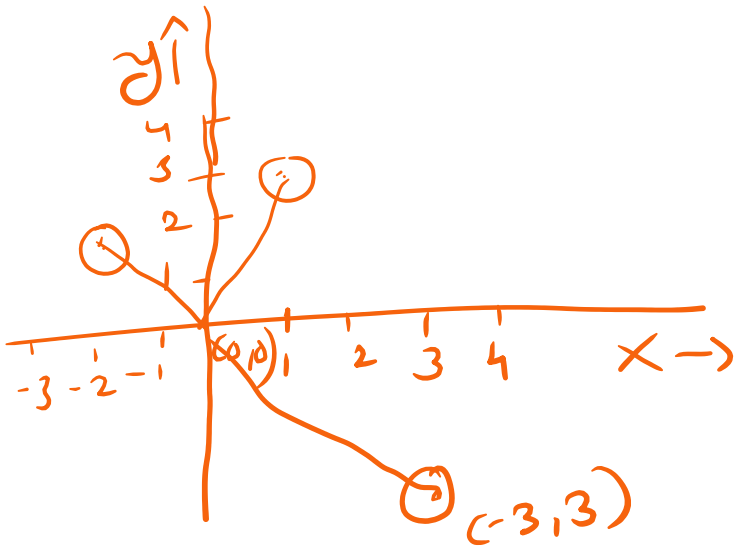
10, 5, 6, 2, 12, 7, 9

② hepifly function convet into heap

③ c → heap

05 April 2024 18:42

973 K- closest points to origin



$$\Rightarrow \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

$$\sqrt{x_i^2 + y_i^2}$$

$$d^2 = x_i^2 + y_i^2$$

$d = 2$ $\quad 2 < 3$

$d = 3$ ↑ More closer

① 1, 3

$d^2 = 1^2 + 3^2 = 10$

$d = 4$

$d = 9$

② -2, 2

$1^2 - 4 + 4 = 8$

$k = 1$

$d^2 = 4 + 4 = 8$

(3) $-3, 3$

$d^2 = 9 + 9 = 18$

$k = 1$

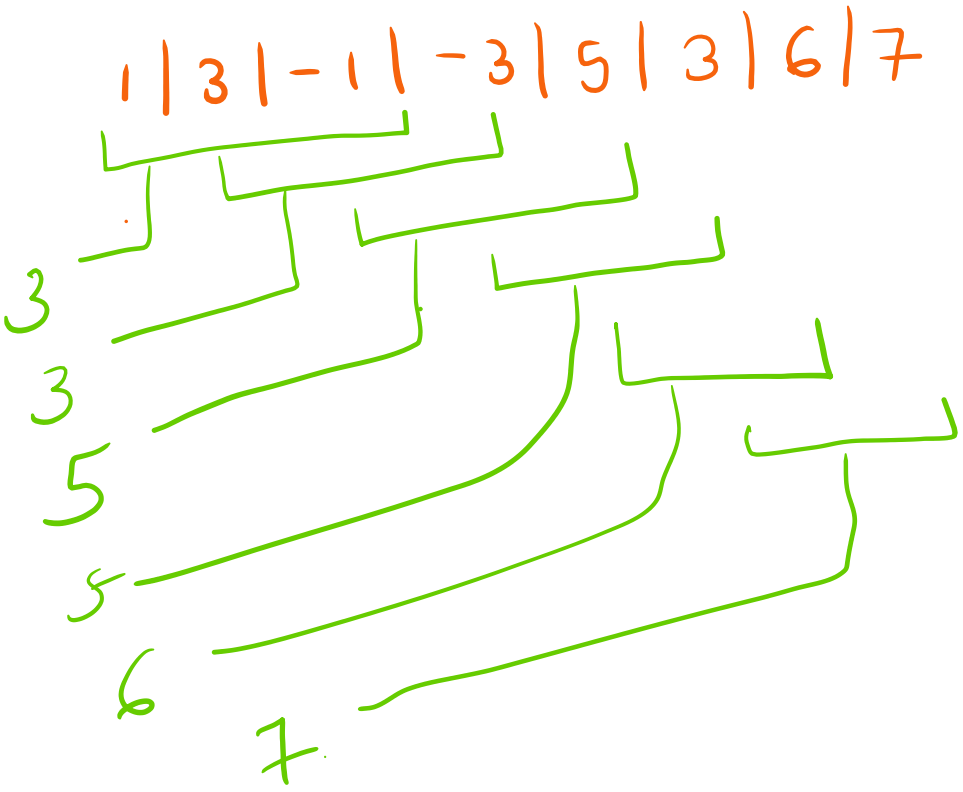
ans :- $(-2, 2)$

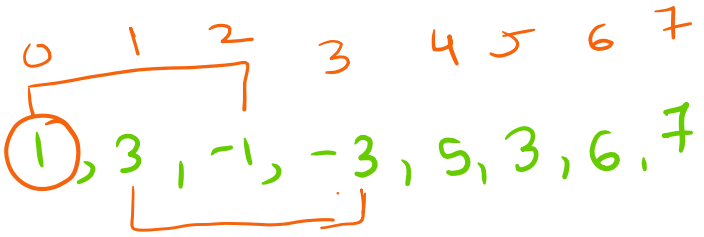
$k = 2$

ans :- $(-2, 2)$

$(1, 3)$

239 sliding window max





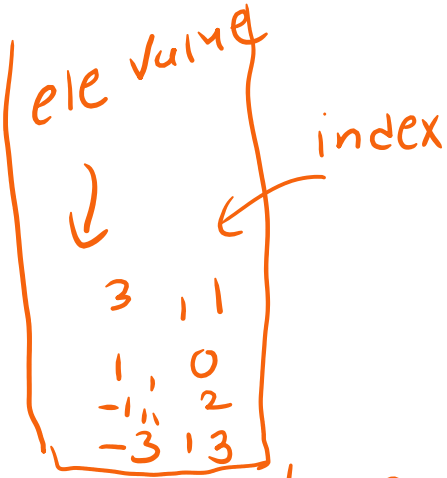
loop = 0 -> k-1

Next window

i = k

lets remove, old window max

while (top().second <= i-k){

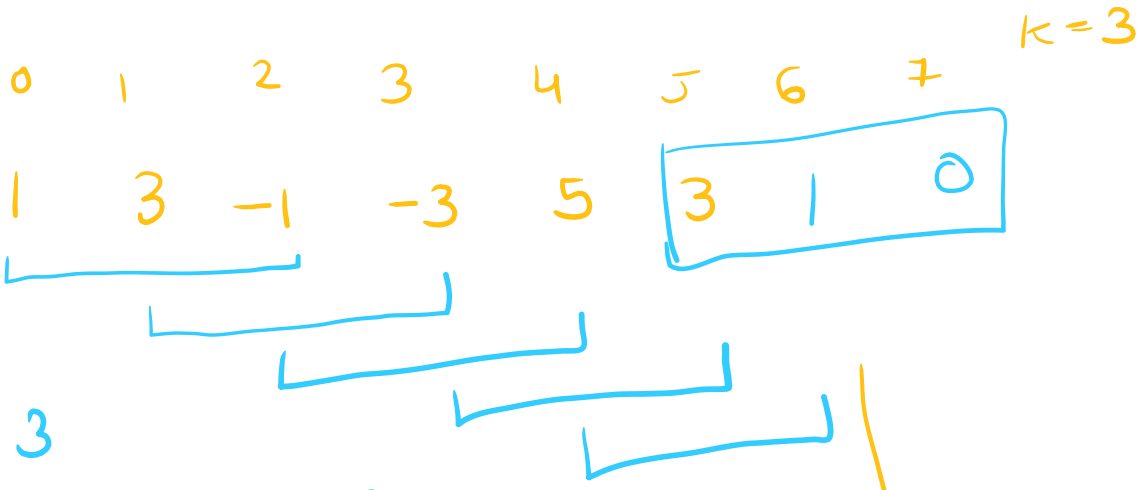


3 -3 = 0

pq.pop()

}

ans → 3



① 3

② i = k => i = 3.

③ i = 4

④ i = 5

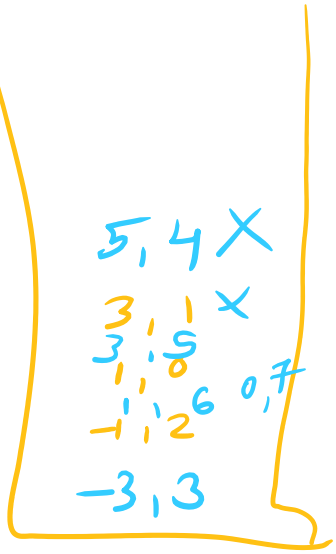
⑤ i = 6.

⑥ i = 1

while(top().second <= i-k)

pop.

i = 7-3 = 4



ans => 3 3 5 5 5

→ for every window push into heap
& check if the max top() belongs
to pervious window → pop()

07 April 2024 12:42

1878 Get Biggest Three Rhombus Sums In A Grid

(1) All grid -> cell [i,j] => needs to be considered

(2) Make Rhombus using each cell

2163 Minimum Difference in sums After Removal of Elements

7 9 5 8 1 3

$N = 6$

$3 \times n = 6$

$n = 2$

7 9 ~~8~~ ~~5~~ 1 3

$2 \times n = 4$

7 9 | 1 3

n n

Sum first = 16

Sum second = 4

16 - 4 = 12

7 ~~9~~ 5 8 ~~1~~ 3

7 5 | 8 3

12 11

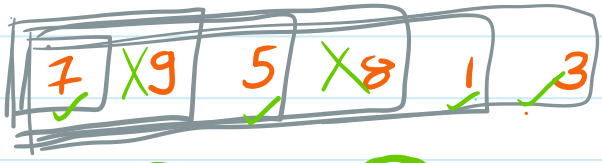
12 - 11 = 1

(M1) Bruteforce → Try and remove all n subsequence

① $S_f - S_s \rightarrow$ minimize

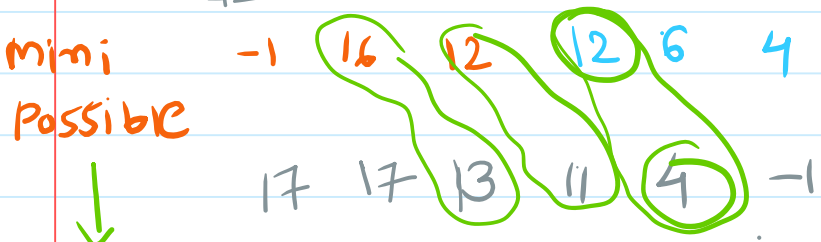
S_f ↓ Mini possible

S_s ↓ Max possible



$3 \times n = 6$

$n = 2$



Prefix ↓

sum of first n min-element from left.

3, 1, 8 → ans

Max possible ↓

Suffix ↓

sum of first n max. element from right.

7 9 5 8 1 3

0 1 2 3 4 5

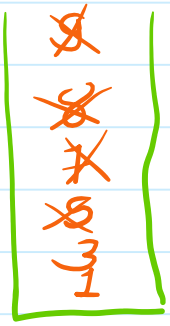
[-1, 16, 12, 12, 6, 4]

minimum possible sum of n element

i=0, sum=7

i=1, sum= 7+9 = 16

```
if(heap.size == n)
{
    Prefix[i] = sum;
}
```



max heap

21-9 = 12

i=2 sum = 16+5 = 21

```
if ( heap.size > n)
{
    heap.top element ko pop kasana.
    sum -= heap.top();
    pop();
}
```

i=3 sum = 12+8 = 20

20-8 = 12

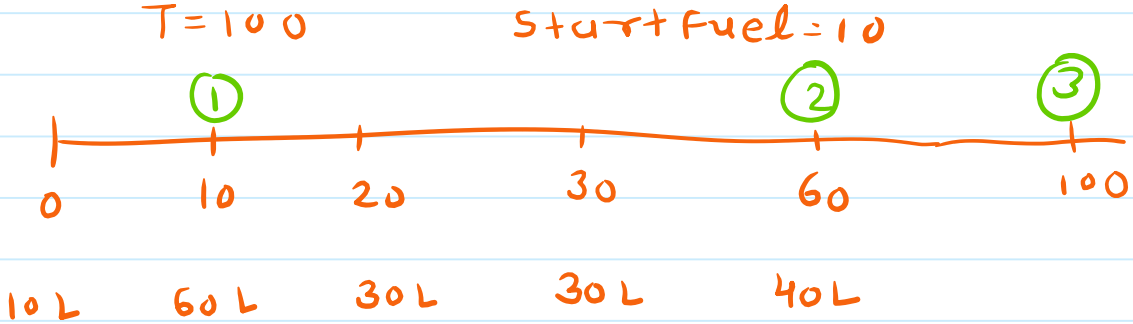
i=4 sum = 12+1 = 13

13-7 = 6

i=5 sum= 6+3 = 9

9-5 = 4

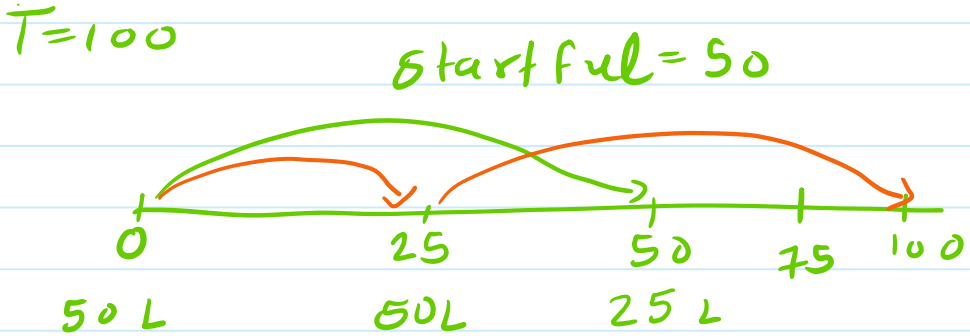
871 Minimum Number of Refueling stops



F=60

F=10
+40
50

F=10L



25+50
=75L

F=0+25L

stop=1