Heaps

→ K^{th} smallest element :-

i/p → { 7, 10, 4, 3, 20, 15 }

(III^rd, II^nd, 2^st)

K = 3 → ans = 3rd smallest element

ans = 7

ans → { 7, 10, 4, 20, 15 }

(II^nd, III^rd, I^st, IV^th)

arr[K-1] → { 4, 7, 10, 20, 23 }

(I^st, II^nd, III, IV, V^th)

K = 4

ans = 4th smallest element

ans = 15

0 1 2 3 ...

Approach :- Algo #1

→ sort array in inc order → O(n log n)

→ return ans = arr[K-1] → O(1)

T.C → O(n log n)

Approach #2 → Heap

→ first K element, 0-(K-1), create a max-Heap

→ for rest elements, K → (n-1)th, if element < heap.top()

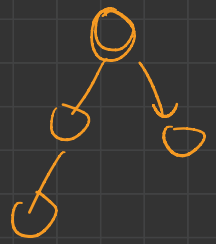
heap.pop()

heap.push(element)

Heap → K elements

n elements → K (smallest) element

ans = heap.top()



i/p → { 7, 10, 4, 20, 15 }

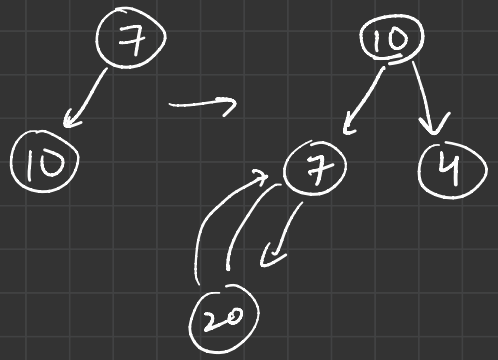
k = 4

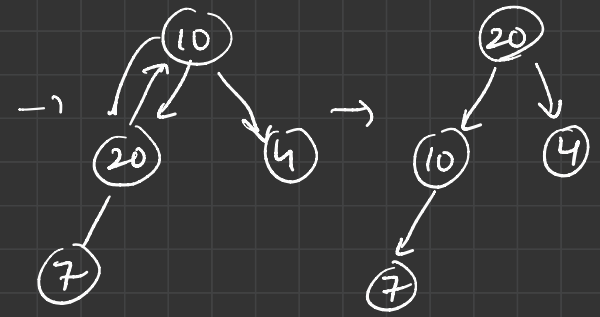
4 7 10 15 20

→ k size max-heap → for first k element 4th smallest

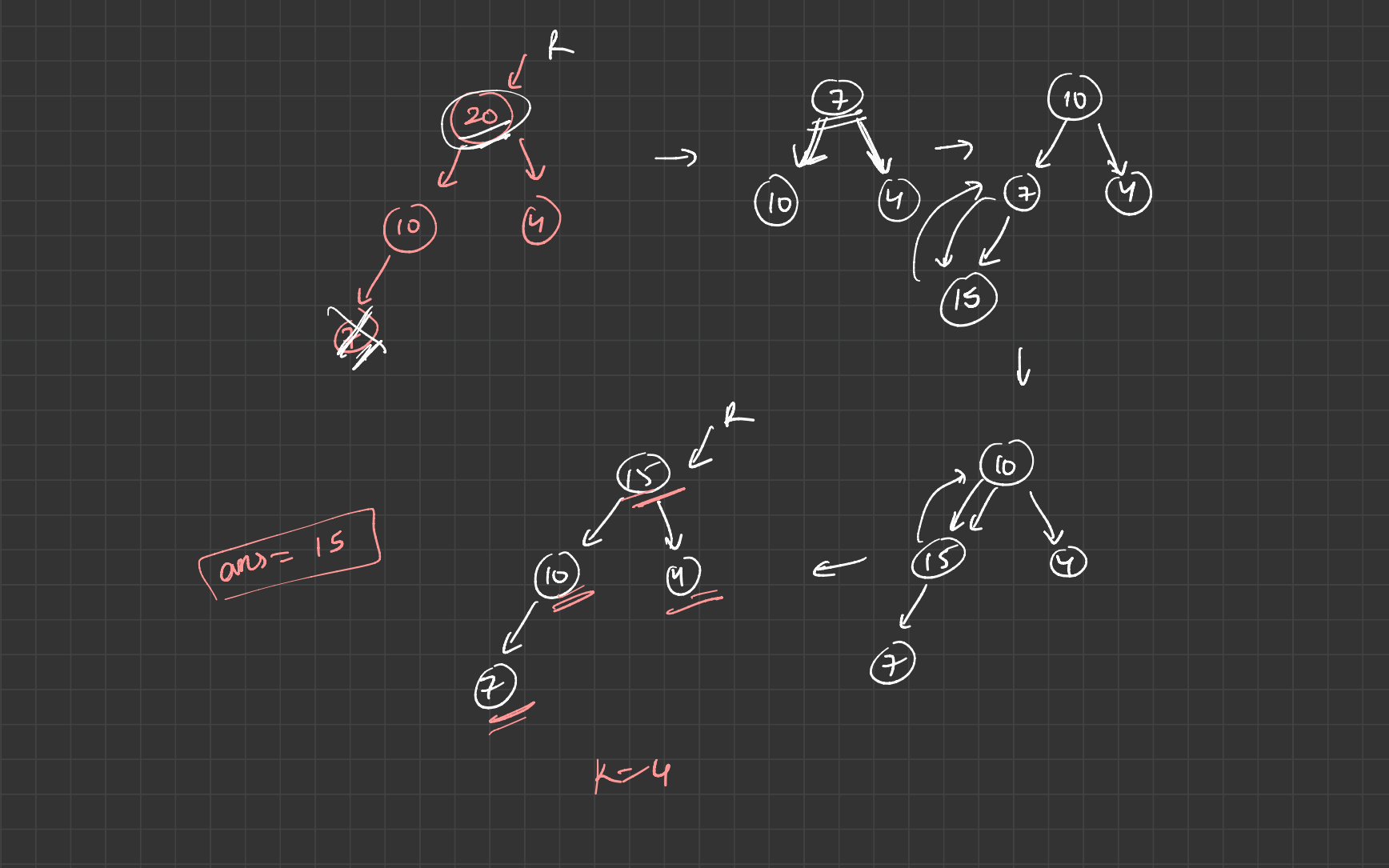
T.C → ? → H/W

S.C → O(k)



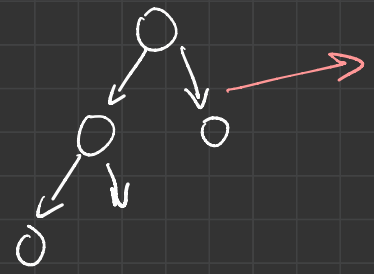


15 < 20 → True


R
20
10
4
7
10
4
7
10
4
7
15
R
15
10
4
7
10
15
4
7
ans = 15
K=4

→ is Binary Tree Heap →?

i/p → BT



Heap
- → CBT
- → satisfy Heap order
  - Max Heap
    - Root
    - childs

o/p → Heap → T/F

Algo:-

```
solve()
{
  if( isCBT() && isMaxOrder() )   // false
  {
    return true
  }
  else return false
}
```

isCBT (root , i , nodeCount)

{

if (root == NULL)

return True;

if (i > nodeCount)

return false

Use

{

left = isCBT (root → left , 2i+1 , nodeCount)

right ——————— (2i+2) ———

return (left && right)

}

0-based index

node → i

left → 2i+1

right → 2i+2

no left / right > totalcnt

↓

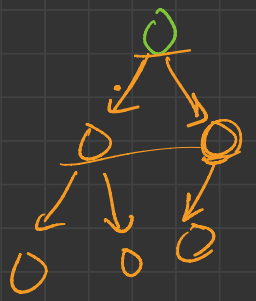
Not a CBT

CBT

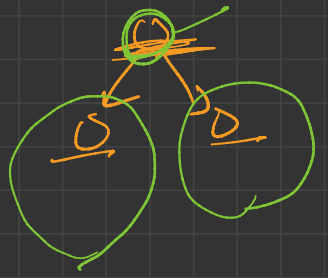
isMaxOrder (root)

{ //Leaf node
if leaf node
{ return true;

① Both child exist
② Leaf node
③ only left exist



if (right == NULL)
{ return root -> data >
root -> left -> data
}

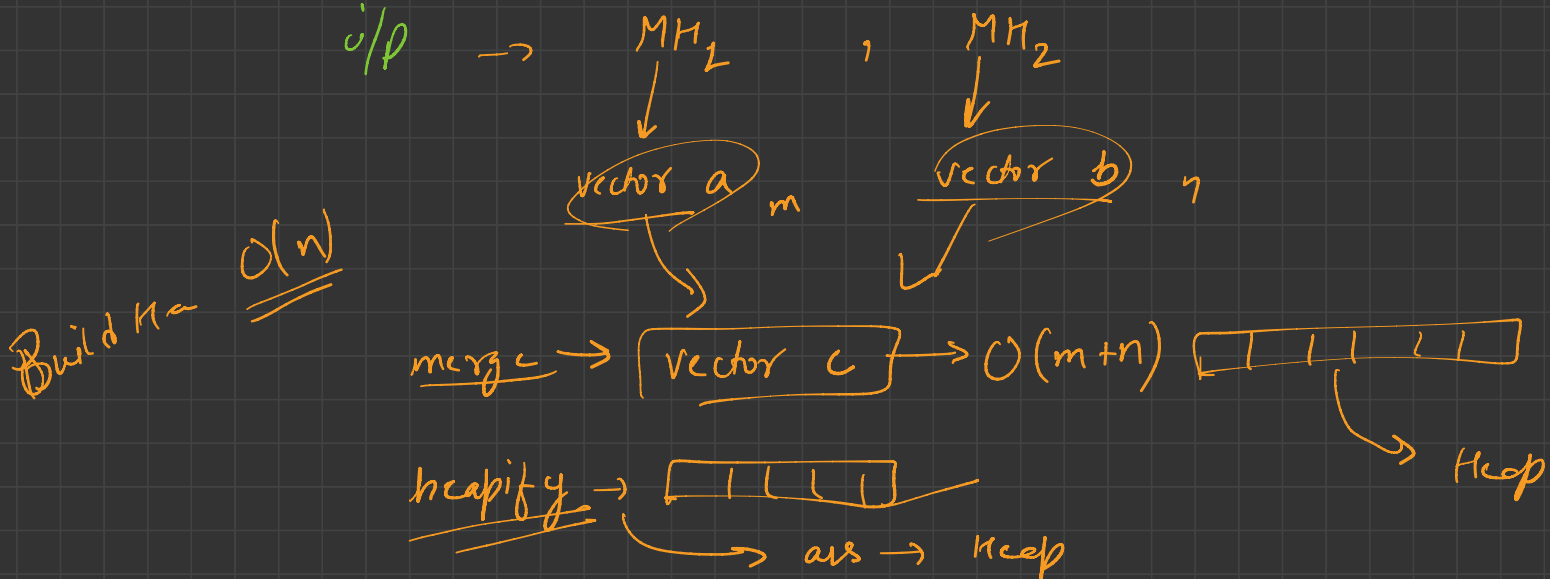


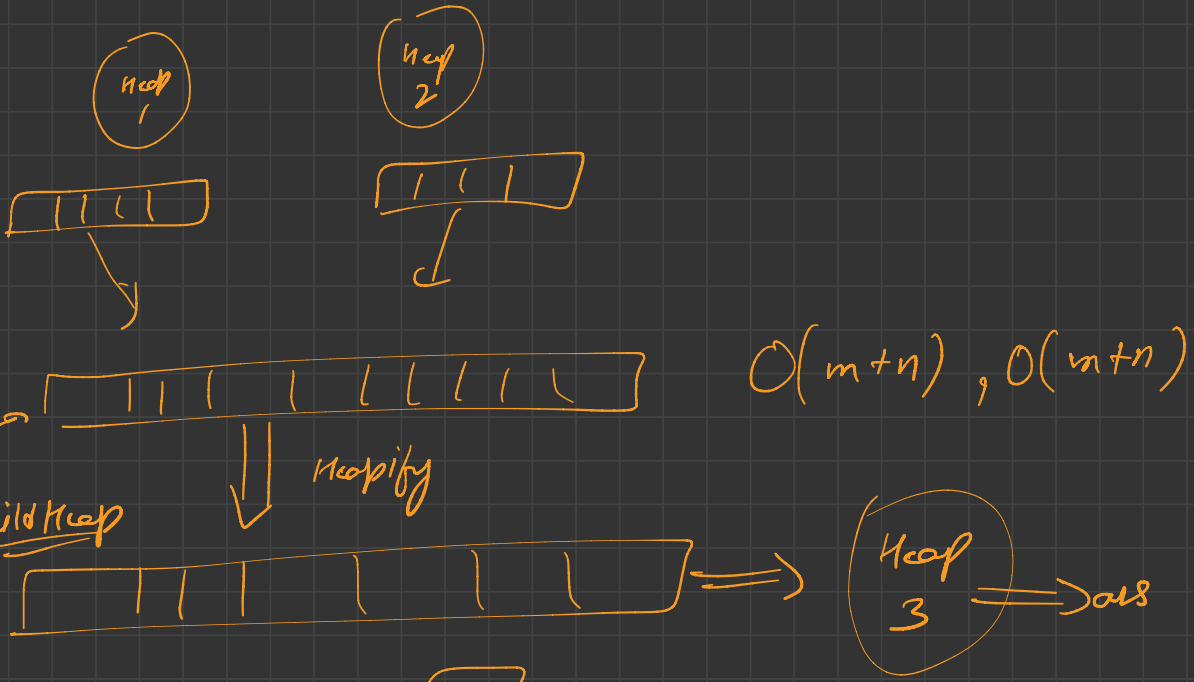
else
{
return root -> data > root -> left -> data
&&
root -> data > root -> right -> data
&&
isMaxOrder (left) && isMaxOrder (right);

S

T.C → $O(n) + O(n) + O(n) \rightarrow O(n)$

→ Merge 2 Binary Max-Heap





left = 2i+1
r = 2i+2

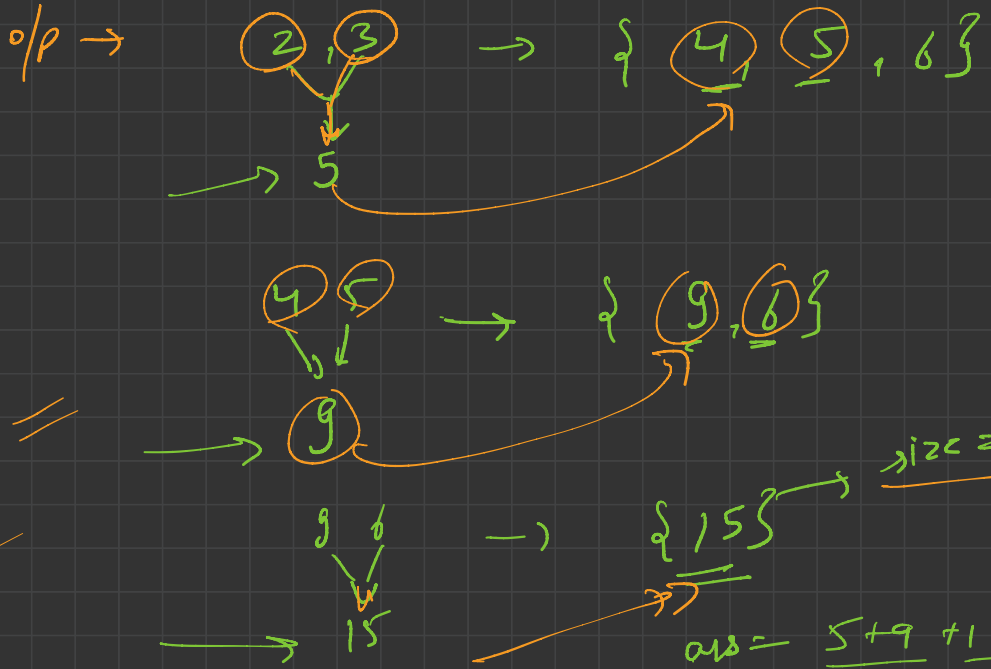
0 - based index → n/2 - 1 , >=0 → non-leaf node

1 - based index → n/2 → >0 → non-leaf nodes

left = 2i
r = 2i+1

→ Min Cost of ropes

i/p → {4, 3, 2, 6}



O(n log n)

O(n)

pq.

```
while (size > 1)
{
    int a = pq.top()
            pq.pop()
    int b = pq.top()
            pq.pop()
    int sum = a+b
    ans += sum
    pq.push(sum)
}
return ans
```

Min-Heap

↳ MinElem

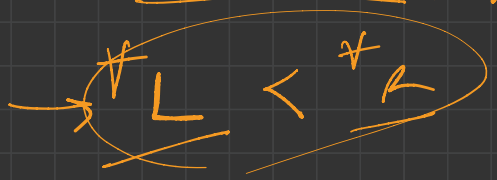
↓

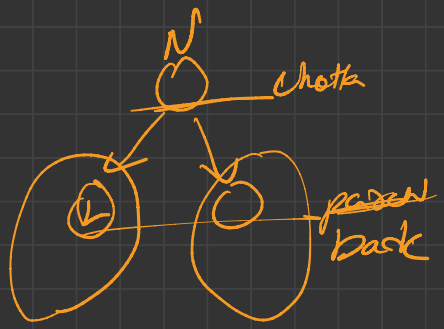
O(1)

BST → Convert BST to Min Heap

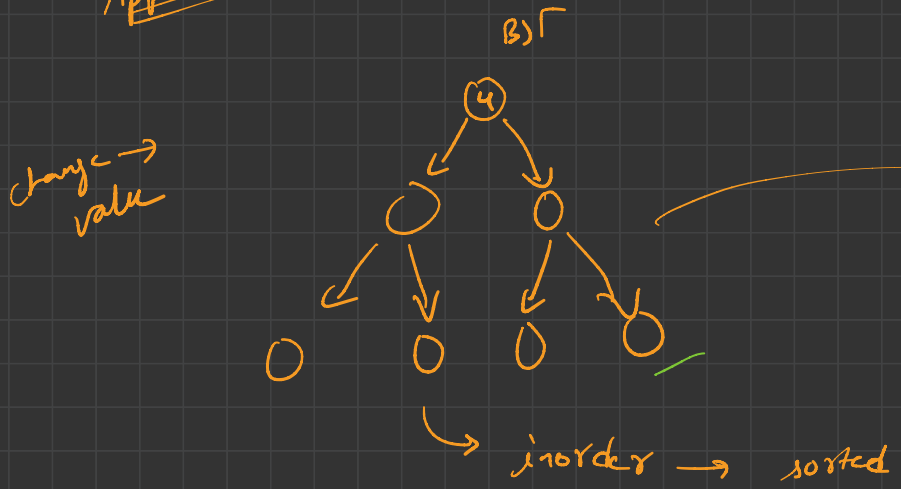
i/p → already CBT

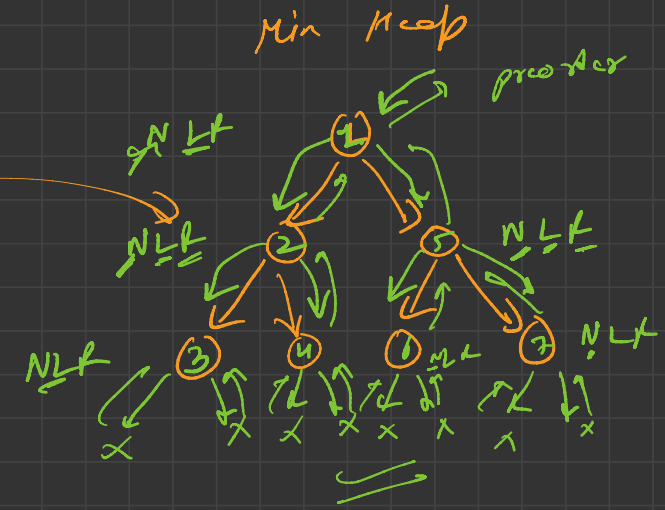
→ MinOrder property

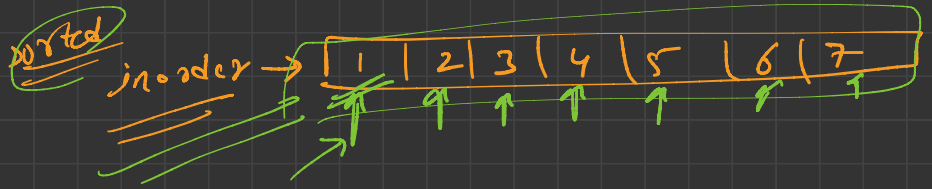


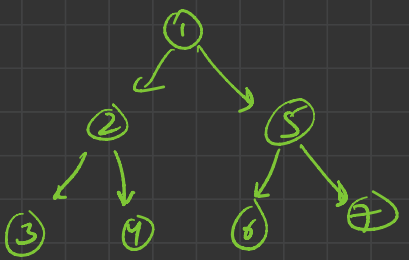


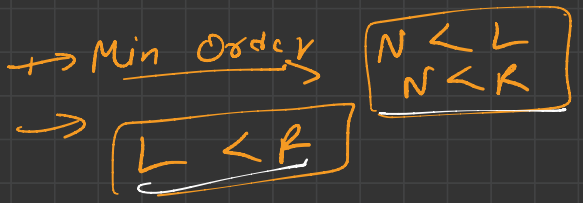
Approach :-

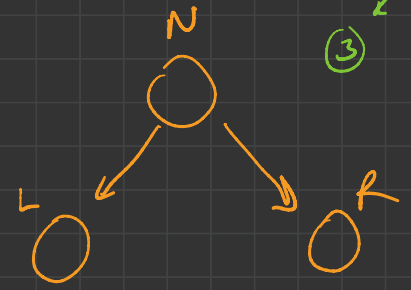


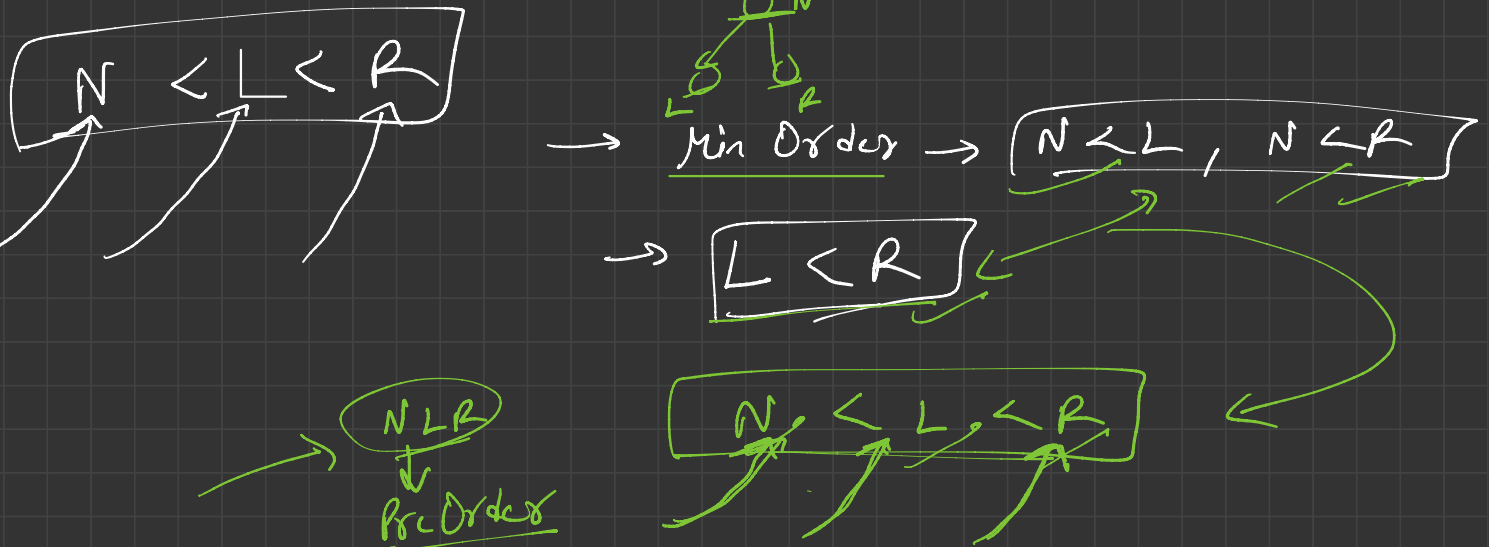



sorted
inorder
1
2
3
4
5
6
7


1
2
5
3
4
6
7


Min Order
N < L
N < R
L < R


N
L
R


N < L < R
N
L
R
Min Order
N < L , N < R
L < R
N L R
PreOrder
N < L < R

```
inorder ( root , vector in)
{
    // base case

    left call
    insert vector
    right call
}
```

```
fillPreOrder ( root , inorder , index
{
    // base case

    root -> data = inorder [index++];

    fillPreOrder (root -> left , inorder , index)
    ———— ( ———— right. ———— ———— )
}
```

N!

H/W

→ K^{th} Smallest / Largest →

MaxHeap MinHeap

→ Merge 2 Max/Min Heap

→ BuildHeap (Heapify)

0/1 based indexing

→ In BT → Heap → CBT + Tree

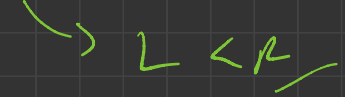
→ Min/Max Order

5

→ Min Cost of ropes → Min Heap

→ BST → Min Heap → already CBT → structure no change required

→ MinOrder prop


L < R