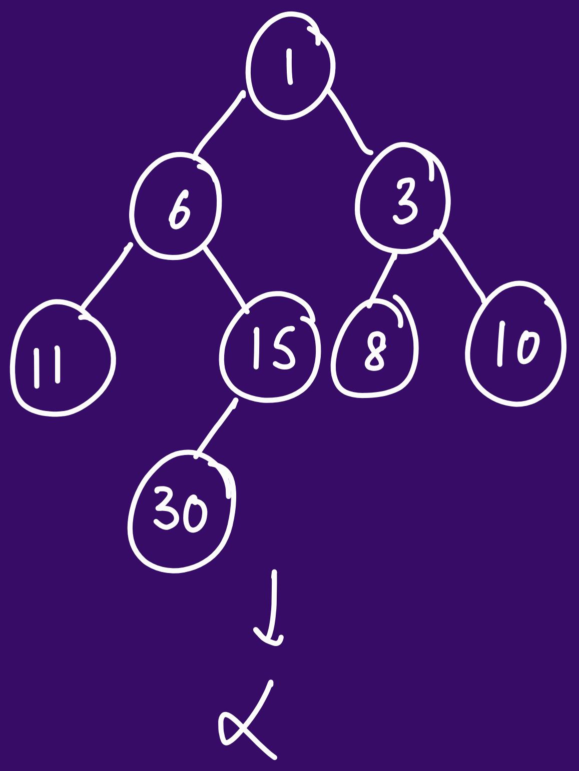
Heaps Visualisation → binary tree

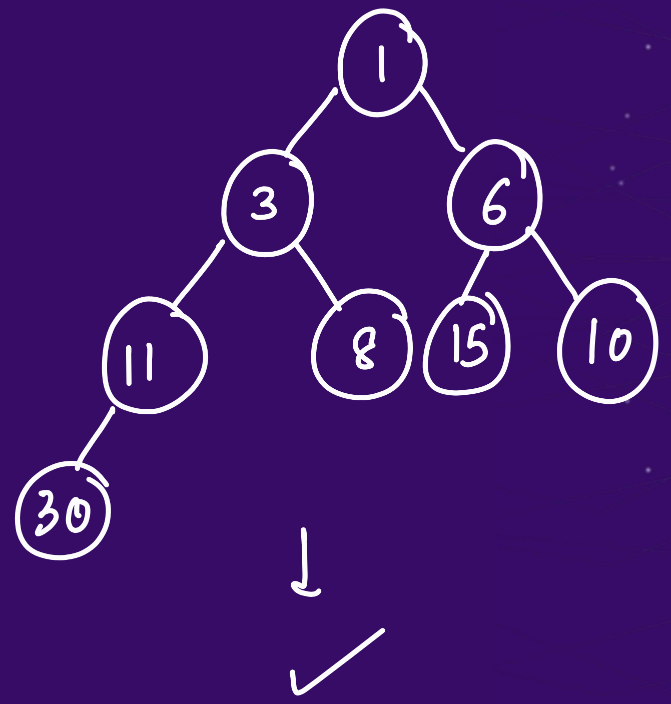
arr = { 10, 1, 3, 8, 11, 30, 15, 6 }

minheap :

↓

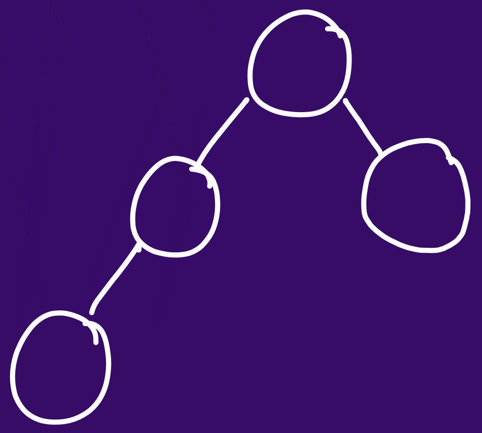
- ek esa binary tree jisme koi bhi node apne children se chhoti value rakhta hai
- CBT





Heaps Visualisation

arr = { 1, 4, 8, 2 }



CBT [Complete Binary Tree]

↓

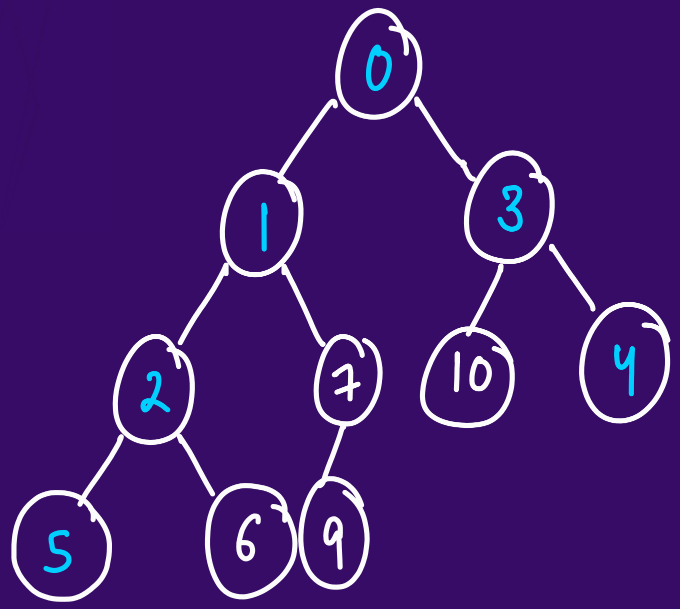
where 'n-1' levels are completely filled. & the last level may or may not be completely filled, but is filled from left to right

CBT is always balanced so height of CBT is always 'log n'

Heaps Visualisation

Rough Implementation : [Minheap]

arr = { 1, 2, 4, 5, 9, 10, 3, 0, 6



Addition / Insertion

↓

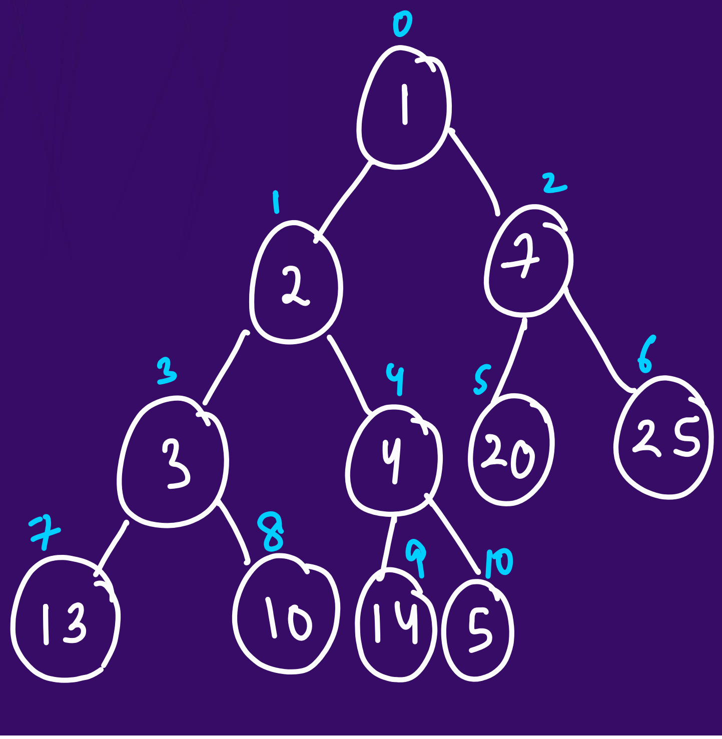
1) add the ele at last → O(1)

2) upheapify → O(log n)

Ques: Heap is implemented by array & visualised as a CBT with HOP

Q1 : Implement a MinHeap by Array

$$arr = \{ \begin{array}{ccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1, & 2, & 7, & 3, & 4, & 20, & 25, & 13, & 10, & 14, & 5 \end{array}$$



$lc = 2p+1$

$rc = 2p+2$

$p = \frac{c-1}{2}$ [c can be lc or rc]

add(13)

Ques:

Q1 : Implement a MinHeap by Array

```
class Minheap{
    int[] arr;
    int size;

    void add()
}
```

arr. length → capacity

add(num) → 1) arr[size++] = num;
2) upheapify

add(1)
add(6)
add(0)
add(2)

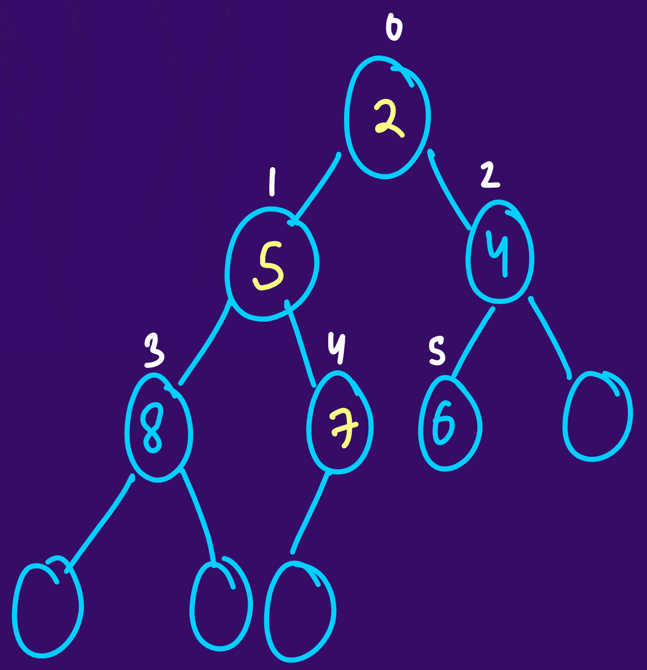
add(-1)

0	1	2	3	4	5	6	7	8	9
0	2	1	6	-1					

size
~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

Ques:

Q1 : Implement a MinHeap by Array 'Remove' in heap



pq. remove();

1) swap arr[0] & arr[size-1]

2) size--

3) DownHeapify

Ques:

Q1 : Implement a MinHeap by Array

```
private int[] arr;
private int size;
MinHeap(int capacity){
    arr = new int[capacity];
    size = 0;
}

public void swap(int i, int j){
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
public int peek() throws Exception {
    if(size==0) throw new
Exception("Heap is Empty!");
    return arr[0];
}
public int size(){
    return size;
}
```

```
public void add(int num) throws Exception{
    if(size==arr.length) throw new
Exception("Heap is Full!");
    arr[size++] = num;
    upheapify(size-1);
}
public void upheapify(int idx){
    if(idx==0) return; // base case
    int parent = (idx-1)/2;
    if(arr[idx]<arr[parent]){
        swap(idx,parent);
        upheapify(parent);
    }
}
public void swap(int i, int j){
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

Ques:

Q1 : Implement a MinHeap by Array

```
public int remove() throws Exception{
    if(size==0) throw new Exception("Heap is Empty!");
    int peek = arr[0];
    swap(0,size-1);
    size--;
    downHeapify(0);
    return peek;
}
public void downHeapify(int i){
    if(i>=size-1) return;
    int lc = 2*i + 1, rc = 2*i+2;
    int minIdx = i;
    if(lc<size && arr[lc]<arr[minIdx]) minIdx = lc;
    if(rc<size && arr[rc]<arr[minIdx]) minIdx = rc;
    if(i==minIdx) return;
    swap(i,minIdx);
    downHeapify(minIdx);
}
```

Homework :

Implement a MaxHeap using Array

Heapify Algorithm

↓

already done

upheapify & downheapify

↓ add

↓ remove

Note

- A sorted array is always a minheap. Vice-versa is not true
- A sorted array in decreasing order is always a Maxheap.

Heap Sort

↓

mazak

adding the 'n' elements to heap & then remove them one by one

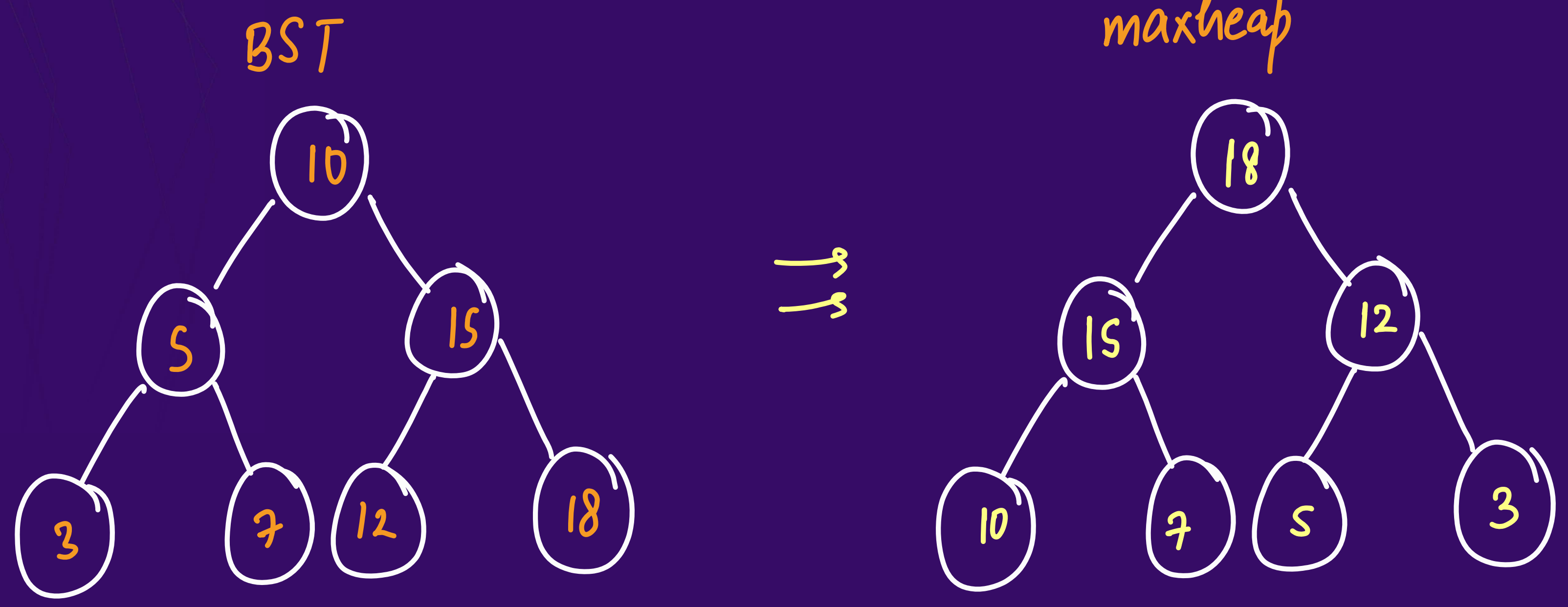
→ T.C. = $O(n \log n)$

S.C. = $O(n)$

Kind of like Merge Sort

Ques:

Q2 : Convert BST to MaxHeap



reverce inorder = {18, 15, 12, 10, 7, 5, 3}

& then level order

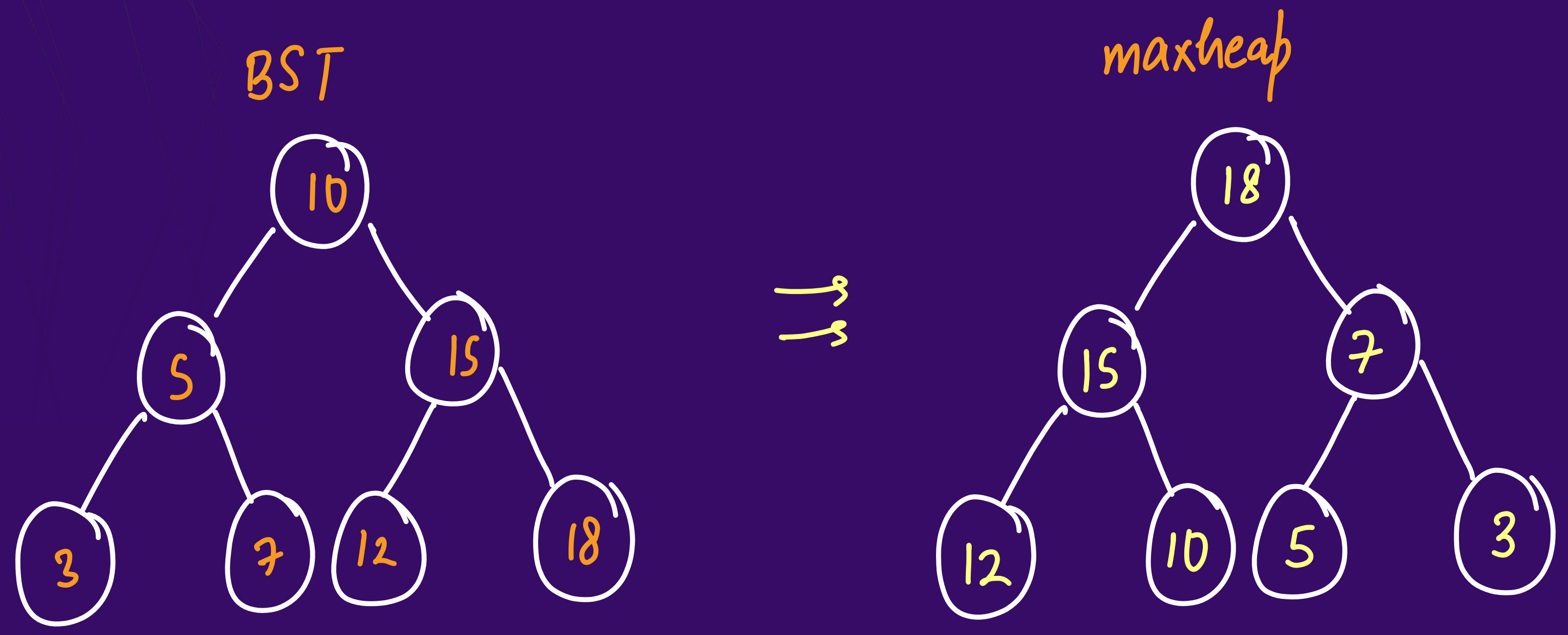
PW SKILLS

Ques:

Pre : Root Left Right

Rev In : Right Root Left

Q2 : Convert BST to MaxHeap



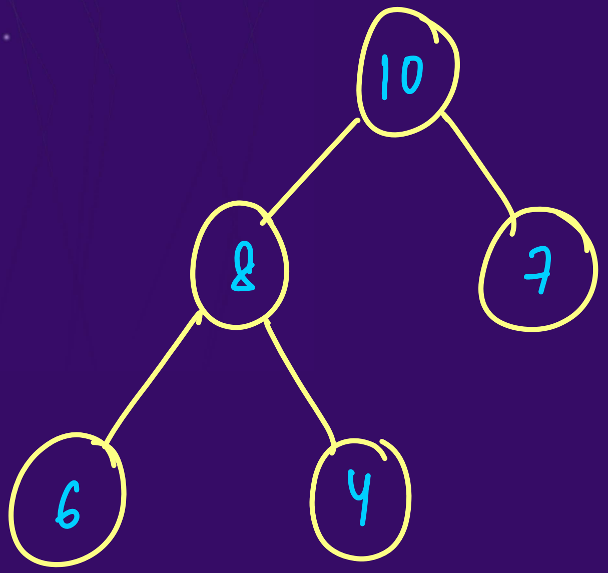
reverse Inorder : {18, 15, 12, 10, 7, 5, 3}

then preorder

Ques:

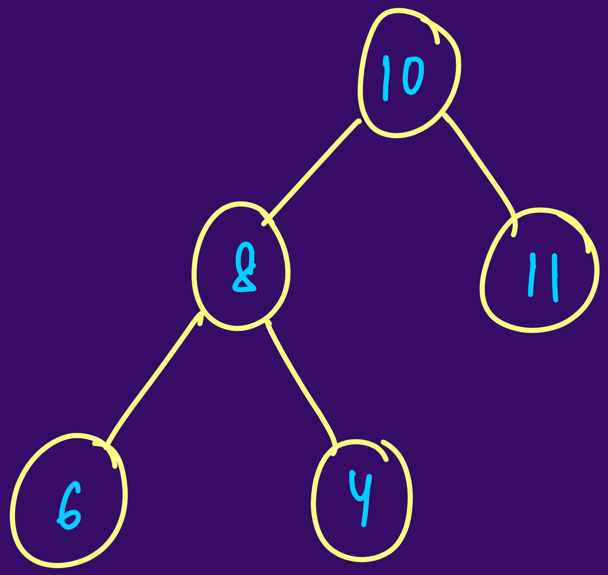
P > C ↑

Q3 : Check if given Binary Tree is a MaxHeap or not



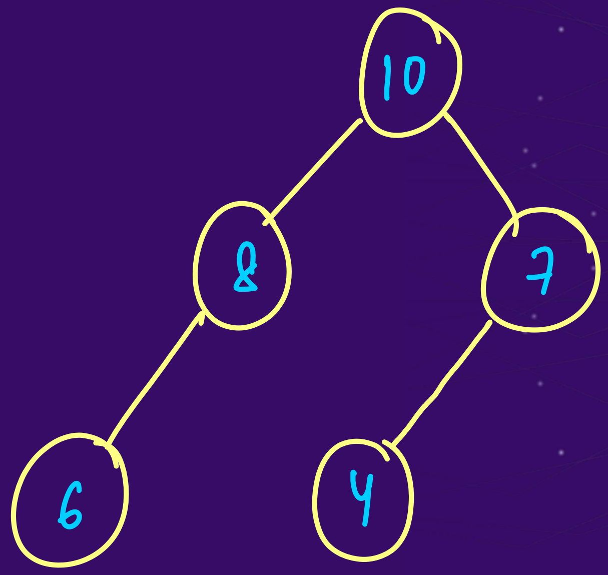
True

HOP ✓ CBT ✓



False

HOP ✗ CBT ✓



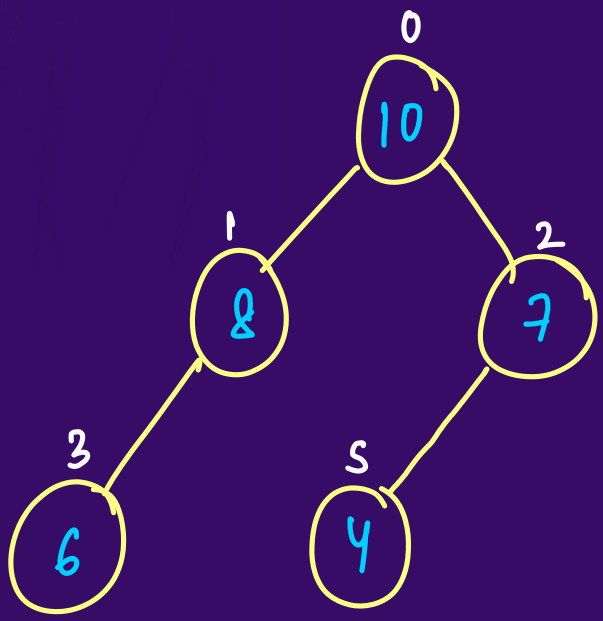
False

HOP ✓ CBT ✗

Ques:

Q3 : Check if given Binary Tree is a MaxHeap or not

How to check if any tree is CBT or not.



size = 5

$lc = 2i + 1$

$rc = 2i + 1$

Ques:

Q3 : Check if given Binary Tree is a MaxHeap or not

int , double, char, String → pass by value

↓ Integer ↓ Double ↓ Character

↓

pass by reference

Ques:

Q3 : Check if given Binary Tree is a MaxHeap or not

```java
private static boolean isMaxHeap(Node root){
    int n = size(root);
    return isHeap(root) && isCBT(root,0,n);
}

private static boolean isCBT(Node root, int i, Integer n) {
    if(root==null) return true;
    if(i>=n) return false;
    return isCBT(root.left,2*i+1,n) && isCBT(root.right,2*i+2,n);
}

private static int size(Node root) {
    if(root==null) return 0;
    return 1 + size(root.left) + size(root.right);
}

private static boolean isHeap(Node root) {
    if(root==null) return true;
    if(root.left!=null) if(root.val<root.left.val) return false;
    if(root.right!=null) if(root.val<root.right.val) return false;
    return isHeap(root.left) && isHeap(root.right);
}
```

Ques:

Q1 : Find Median from Data Stream.

8 6 1 3 13 18 -6

-6 1 3 6 8 13 18

↓

median

10 80 100 40

10 40 80 100

↓

60 median

[Leetcode 295]

Ques:

Q1 : Find Median from Data Stream.

8 6 1 3 13 18 -6

stream	median
8	8
8 ,6	7
8 ,6, 1	6
8 ,6, 1,3	4·5
8,6, 1,3, 13	6

$$\text{Total Time} = 1\log 1 + 2\log 2 + 3\log 3 + \ldots\ldots n\log n$$

$$= \quad n^2 < \sum_{r=1}^{n} r\log r \quad < n^2\log n$$

[Leetcode 295]

Ques:

Q1 : Find Median from Data Stream.

Improved Approach : Do not use builtIn Sort. Use insertion sorting algo

→ Time → $1+2+3+\dots n = \boxed{O(n^2)}$

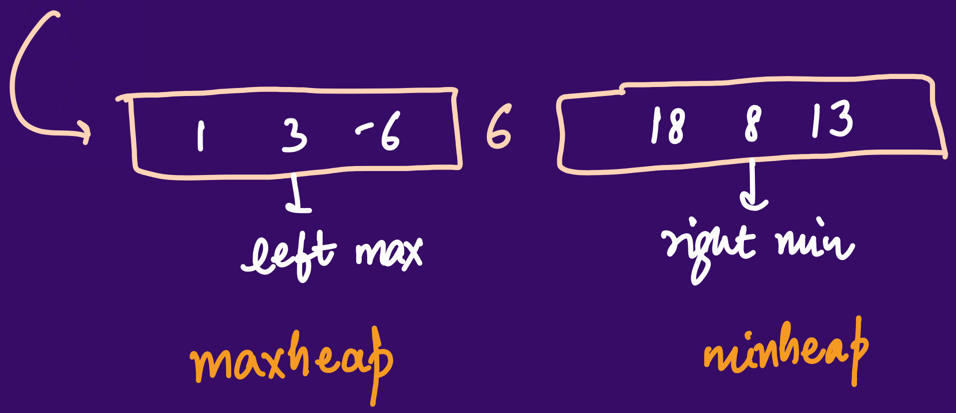
[Leetcode 295]

Ques:

Q1 : Find Median from Data Stream.

Best approach : Using Heaps

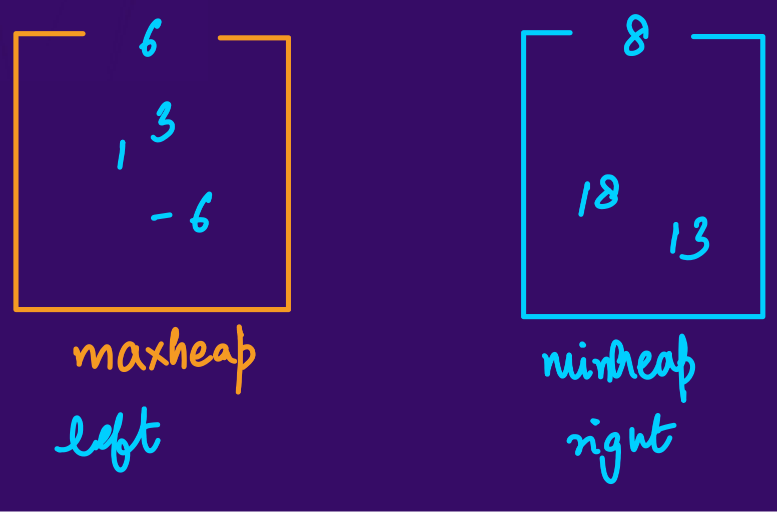
8 6 1 3 18 13 -6



[Leetcode 295]

Ques:

Q1 : Find Median from Data Stream.



[Leetcode 295]

Ques:

'n' calls

PW SKILLS

Q1 : Find Median from Data Stream.

```
public void addNum(int num) {
    if(maxHeap.size()==0) maxHeap.add(num);
    else{
        if(num<maxHeap.peek()) maxHeap.add(num);
        else minHeap.add(num);
    }

    // Balance the heaps
    if(maxHeap.size()==minHeap.size()+2){
        int top = maxHeap.remove();
        minHeap.add(top);
    }
    if(minHeap.size()==maxHeap.size()+2){
        int top = minHeap.remove();
        maxHeap.add(top);
    }
}
```

O(logn) for one call
so for 'n' calls = O(nlogn)

o(1)

```
public double findMedian() {
        if(maxHeap.size()==minHeap.size())
            return (maxHeap.peek()+minHeap.peek())/2.0;
        else if(maxHeap.size()>minHeap.size())
            return maxHeap.peek();
        else return minHeap.peek();
    }
```

O(n) total

[Leetcode 295]

Ques:

[a, b] ↑ b - a should be minimum

PW SKILLS

Q2 : Smallest Range covering elements from K Lists

K=3

[4 10 15 24 26]
[0 7 12 21]
[5 18 22 30]

heap < ele, row, col >

21
↓
(21, 1, 3)

22 24

minheap(K)

max
~~5~~
~~7~~
~~10~~
~~18~~
~~21~~
24

minRange = [0, IMax]

✓ [0,5] ✓ [4,7] ✗ [5,10] ✗ [7,18] ✗ [10,18] ✗ [12,18] ✗ [15,21]

✗ [18,24] ✗ [21,24]

[Leetcode 632]

Ques:

Q2 : Smallest Range covering elements from K Lists

[Leetcode 632]

```
public int[] smallestRange(List<List<Integer>> nums) {
        int[] ans = {0,Integer.MAX_VALUE};
        // MinHeap
        PriorityQueue<Triplet> pq = new PriorityQueue<>();
        int k = nums.size();
        int max = Integer.MIN_VALUE;
        for(int i=0;i<k;i++){
            int ele = nums.get(i).get(0);
            pq.add(new Triplet(ele,i,0));
            max = Math.max(max,ele);
        }
        while(true){
            Triplet top = pq.remove();
            int ele = top.ele, row = top.row, col = top.col;
            // Update the minimum range
            if(max-ele < ans[1]-ans[0]){
                ans[0] = ele;
                ans[1] = max;
            }
            if(col==nums.get(row).size()-1) break;
            int next = nums.get(row).get(col+1);
            max = Math.max(max,next);
            pq.add(new Triplet(next,row,col+1));
        }
        return ans;
    }
```

If there are total 'n' elements in nums

Extra Space = $O(k)$

Time Complexity = $O(n \log k)$

```
public class Triplet implements Comparable<Triplet>{
        int ele;
        int row;
        int col;
        Triplet(int ele, int row, int col){
            this.ele = ele;
            this.row = row;
            this.col = col;
        }
        public int compareTo(Triplet t){
            return this.ele - t.ele;
        }
    }
```

Time ⇒

THANK YOU