Start : 7:04 AM IST

Agenda

Morris Inorder traversal

→ Inorder

→ PreOrder

→ Post order

Right view of Binary Tree

→ level Order traversal

Running Median

→ Idea of heap

→ Min and Max Heap

Add next pointer in Binary Tree

Any additional Questions (based on time)

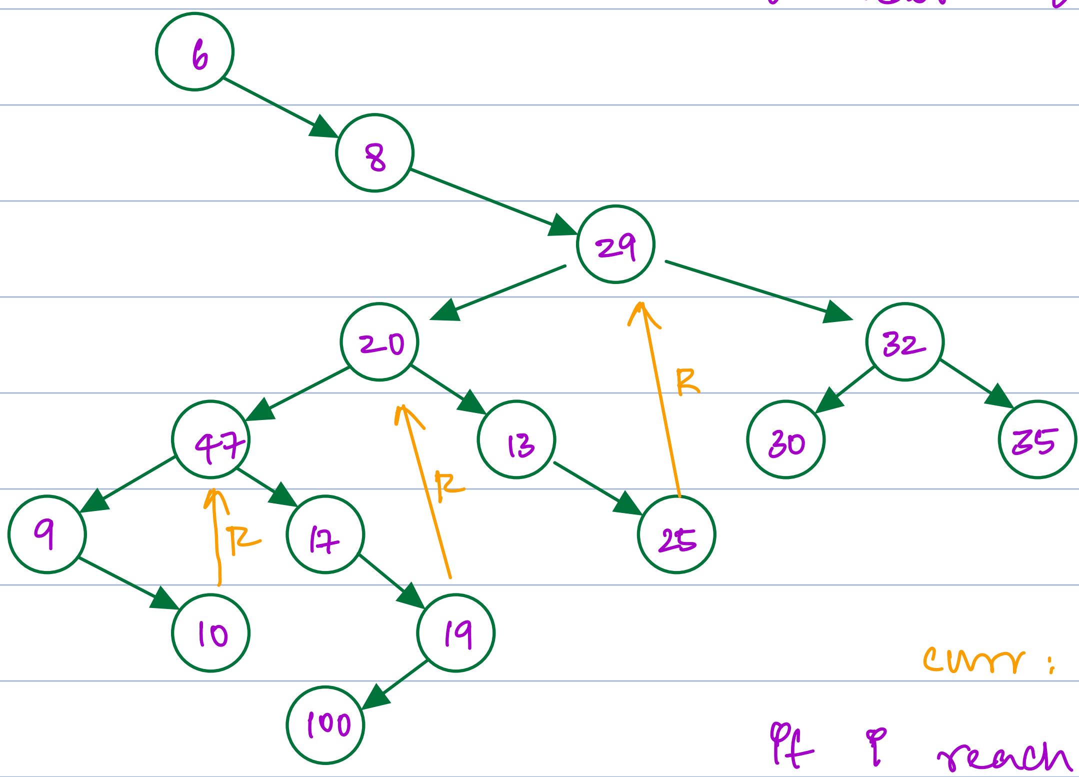
Q2 → Morris Inorder Traversal

Without using any space, output the inorder traversal

Recursion → stack space

Iteration →

Inorder : L N R



curr : 47

if i reach the node itself

Inorder traversal

6	8	9	10	47	17	100	19	20	13	25	29	30	32	35
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Observation

1. if node.left == null print node.data and go right
2. if node.left != null

   Find the right most node in LST and point its right reference to node itself

while finding RMN if we encounter node

itself ① print node.data

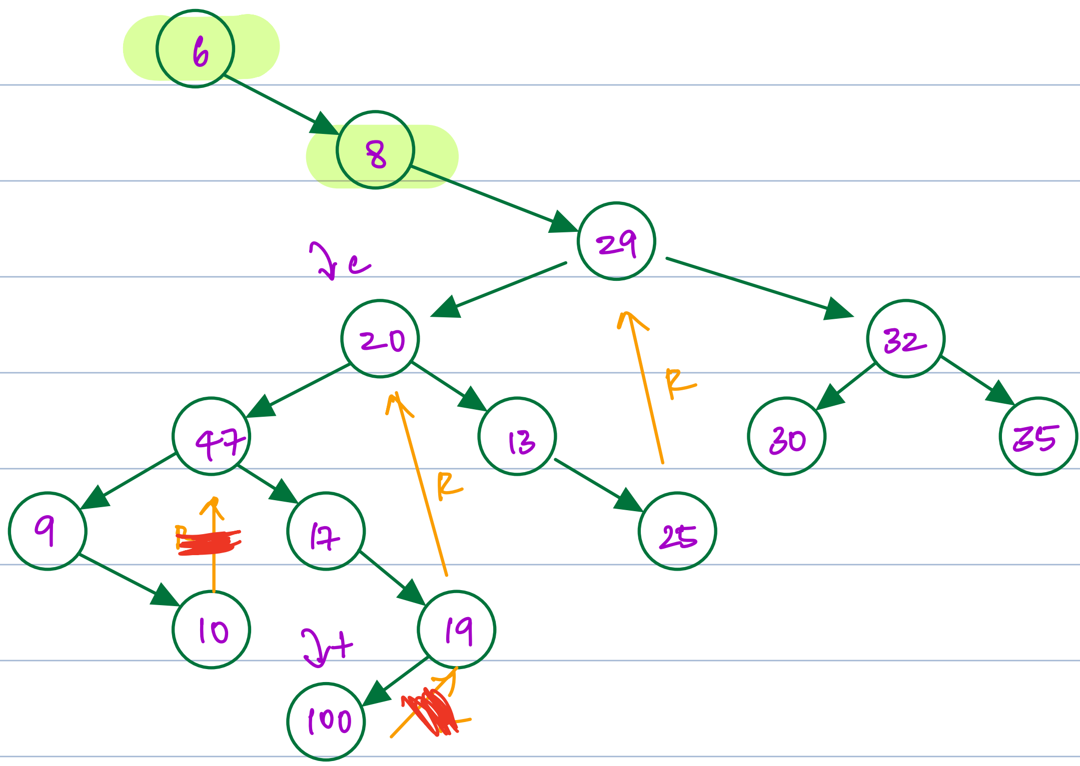
② Remove reference move right

pseudo code (iteration)

```
Node Morris Inorder (root) {
    if (root == null) return Null;
    Node curr = root;
    while (curr != null) {
        // LST is null
        if (curr.left == null) {
            print (curr.data);
            curr = curr.right;
        }
        // LST is not null
        else { // Find Right most node in LST
            temp = curr.left;
            while (temp.right != null &&
                    temp.right != curr)
                temp = temp.right;
            // create link
            if (temp.right == null) {
                temp.right = curr;
```

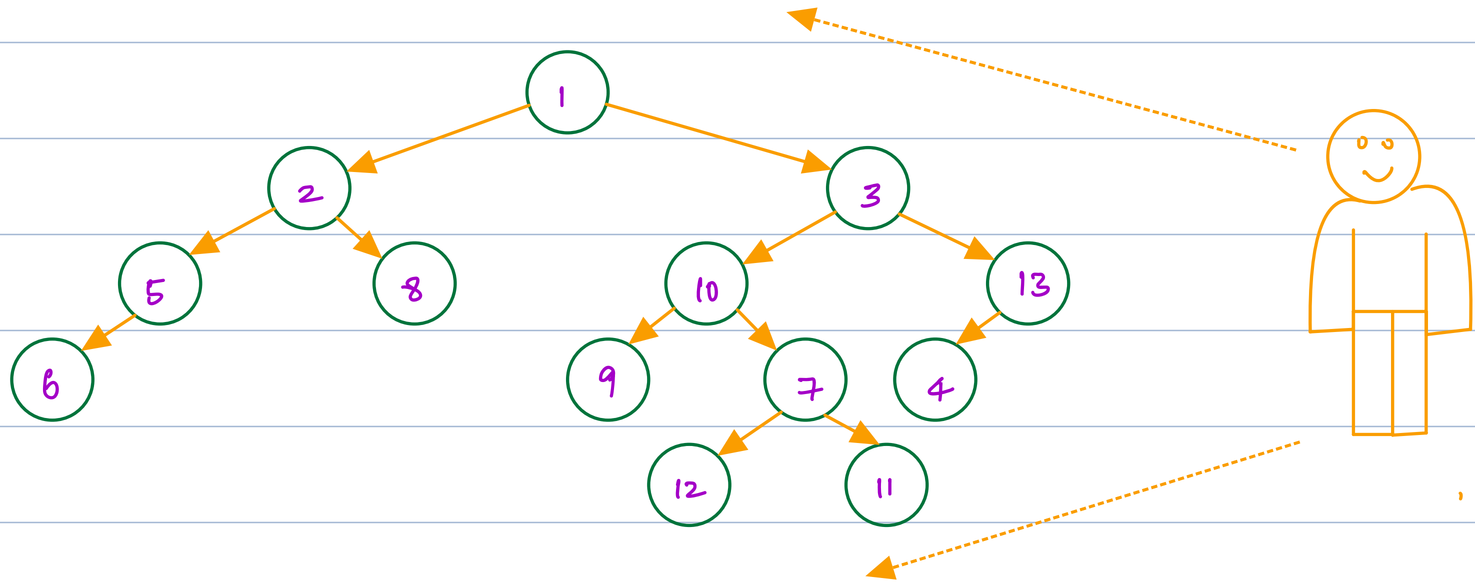
```
                    curr = curr.left;
                }

                // destroy link
                else {
                    temp.right = null;
                    print (curr.data);
                    curr = curr.right;
                }
            }
        }
}
```

$T.C = O(n)$ $S.C = O(1)$



print : 6 8 9 10 47 17 100 19

Given a binary tree, print its right view.



output =

1	3	13	4	11
0	1	2	3	4

~~1~~ ~~2~~ ~~3~~ ~~5~~ ~~8~~ ~~10~~ ~~13~~ ~~6~~ ~~9~~ ~~7~~ ~~4~~ ~~12~~ ~~11~~

Queue

last ↑

pseudo code

```
if (root == null) return null;
// Initialize Queue
q.enqueue (root); last = root;
while (! q. isEmpty()) {
        x = q. dequeue;
        if (x.left) q.enqueue (x.left);
        if (x.right) q.enqueue (x.right);
        if (x == last) print (x.data);
        if (x == last && ! q.isEmpty()) {
```

}   }   last = q.rear();

T.C = O(n)   S.C = O(n)

Break: 8:20 AM → 8:26 AM

Heaps

structure → a complete B.Tree

Types of Heaps

Min Heap (node.data <= its children)

Max Heap (node.data >= its children)

what DS will we use for heap.

array / arraylist

T.C for insert in heap = O(log n)

T.C for removing min in min Heap = O(log n)

T.C for creating heap from array = O(n)

Q → Given an infinite stream of integers. Find the median of the curr set of elements.

Median is the middle element in sorted array

The median of

1	2	5	4	3	6

① sort the array

1	2	3	4	5	6

② median for even length array $\frac{3+4}{2}$

3.5

Median of

1	2	4	3

Sort the array

1	2	3	4

median $= \frac{2+3}{2} = 2.5$

Median of

1	2	3

odd length array only one median

median = 2

Brute Force

I/P → 9 8 4 6 7 12 15 ...

Median → 9 8.5 8

For every input add it to end of array and sort it.

T.C = $N^2 \log n$ S.C = $O(n)$

Idea 2

For every input insert it in correct position and calculate median (Insertion sort)

T.C = $O(n^2)$

Idea 3

I/P → 9 8 4 6 7 12 15 ...

7 4 6	8 9
smaller elements Max Heap	larger elements Min Heap

largest ... smallest

Observation

In case of odd length largest element of Max Heap

In case of even length largest (max) smallest (min Heap) $\frac{\text{largest} + \text{smallest}}{2}$

Maintain equal sizes

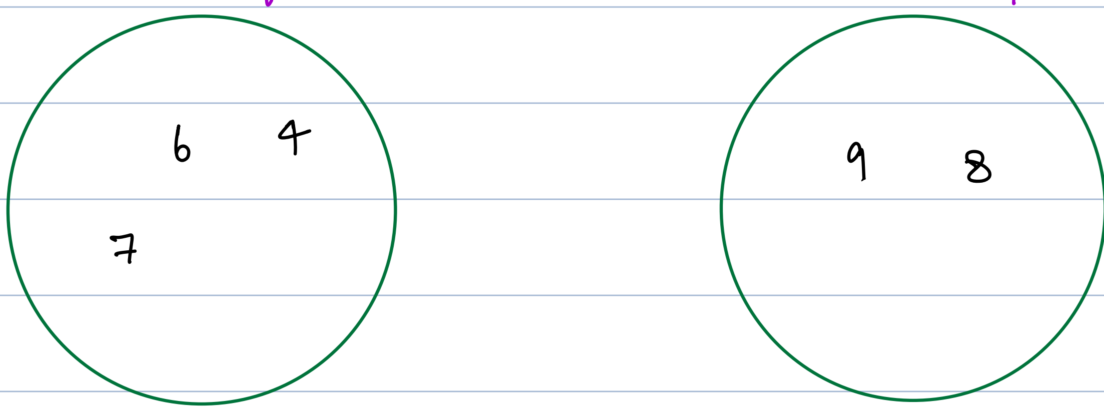
size of max Heap − size of min Heap = {0, 1}

if not met reshuffle heaps

I/P → 9 8 4 6 7 12 15 ...

9 8.5 8 7 7

Max Heap Min Heap



 # pseudo code

```
for ( i=0 ; i< ∞ ; i++) {
    x = input from user ;
    if ( x <= maxHeap[0] )
```

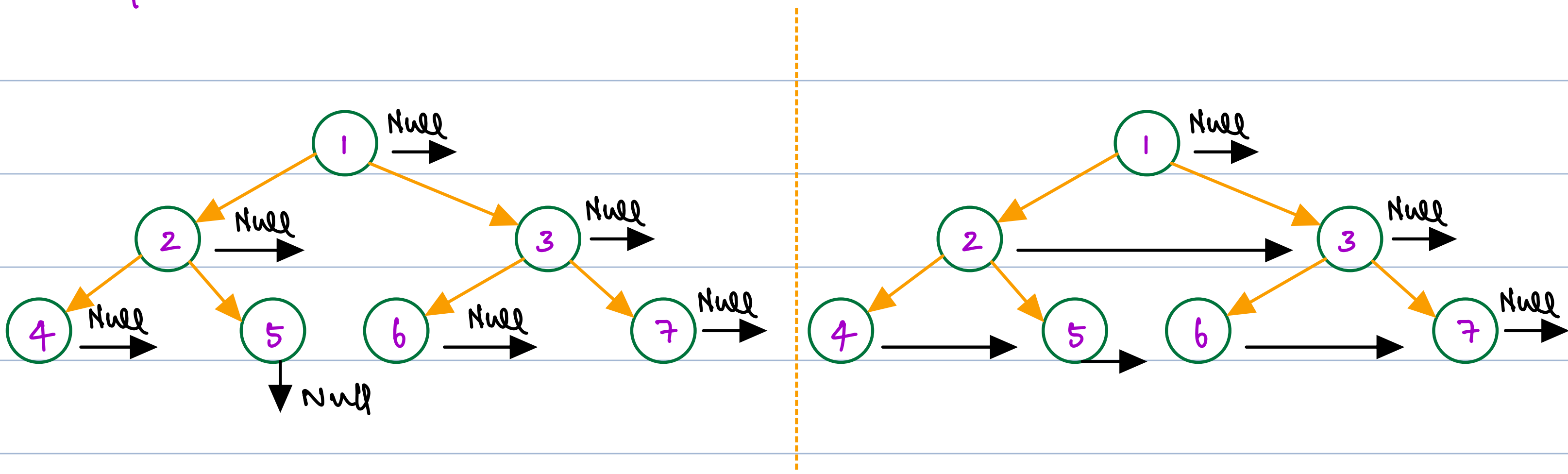
```
        maxHeap.insert(x);
    else
        minHeap.insert(x);
    size_diff = size(maxHeap) - size(minH)
    if (size > 1)
        l = maxHeap.extractMax();
        minHeap.insert(l)
    else if (size < 0)
        s = minHeap.extractMin();
        maxHeap.insert(s);
    // compute median.
    n = size(maxHeap) + size(minHeap)
    if (n % 2 != 0) {
        median = maxHeap[0];
    }
    else {
        median = (maxHeap[0] + minHeap[0]) / 2
    }
}
```

$$T.C = O(n \log n)$$

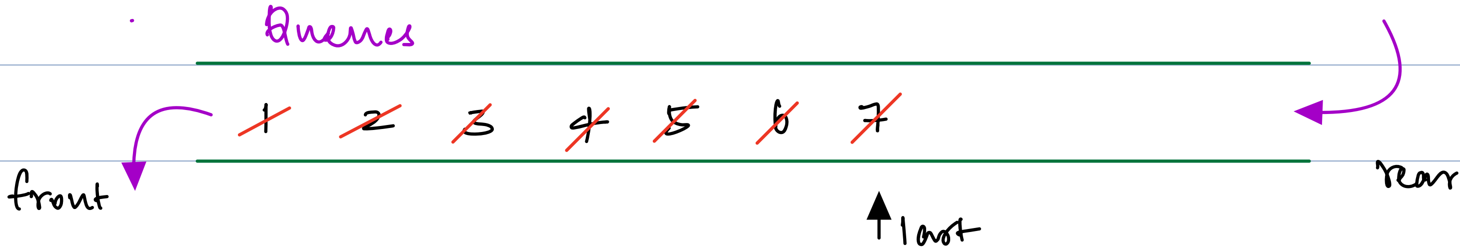
Q → Given a perfect binary tree with next pointers in all nodes, initially pointing to null.

Update the next pointer to point to next node in same level ∀ nodes.

Input **Output**



Queues



Observation

if (node == last) node.next = null

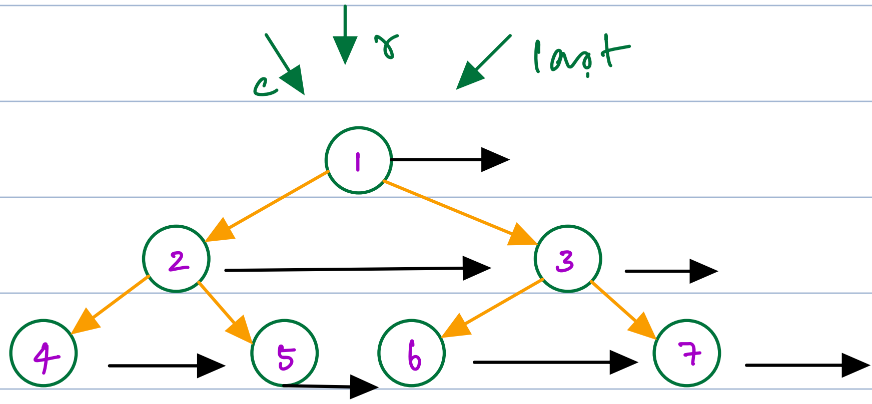
else node.next = q.front()

pseudo code

```
// initialize a queue
q. enqueue (root);   last = root;
while (! q. isEmpty()) {
        curr = q. dequeue ();
        if (curr. left) { q. enqueue (curr. left) }
        if (curr. right) { q. enqueue (curr.right) }
        if (curr != last)
                curr.next = q. front();
        else {
                // update last
                if (! q. isEmpty ()) last = q.rear() ;
        }
}
```

T.C = O(n)   S.C = O(n)

Observation



3 pointer

curr – which node I have added

r = root node

last = last node in given level

pseudo code

```
r = root;    c = root;    last = root;
while (r != null) {
    if (r.left) {
        curr.next = r.left;
        curr = curr.next;
    }
    if (r.right) {
        curr.next = r.right;
        curr = curr.next;
    }
    if (r == last) {
        r = r.next;
```

T.C = O(n)

S.C = O(1)

```
        last.next = null;
        last = curr;
    }
    else {
        r = r.next;
    }
}
```

Converting to Min Heap

arr =

7	6	5	4	3	2	1
0	1	2	3	4	5	6

↓ sort the array

arr =

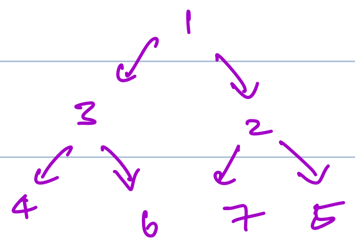
1	2	3	4	5	6	7
0	1	2	3	4	5	6

arr =

7	6	5	4	3	2	1
0	1	2	3	4	5	6

Observation 1 : at any given heap we have $\frac{n}{2}$ leaf nodes

⋮



→ 2 swaps per node [n/8]

→ 1 swap per node [n/4]

→ 0 swaps [n/2]

$$\frac{n}{2} * 0 + \frac{n}{4} * 1 + \frac{n}{8} * 2$$

```
for (i = n/2 - 1 ; i >= 0; i--) {
    heapify (heap, i)
}
```

```
void heapify (heap [], i) {
    while (2i+1 < N) {
        // handle if right child exist or
        // not
        x = min (heap [i], heap [2i+1], heap [2i+2])
        if (x == heap [i])
            break;
        else if (x == heap [2i+1])
            swap (heap, i, 2i+1)
```

```
            i = 2i+1
        }

        else {
            swap (heap, i, 2i+2)
            i = 2i+2 ;
        }
    }
}
```