Introduction to Heaps

Special class

Rohit Mazumder • Dec 30, 2020

Introduction to Heaps

Agenda:

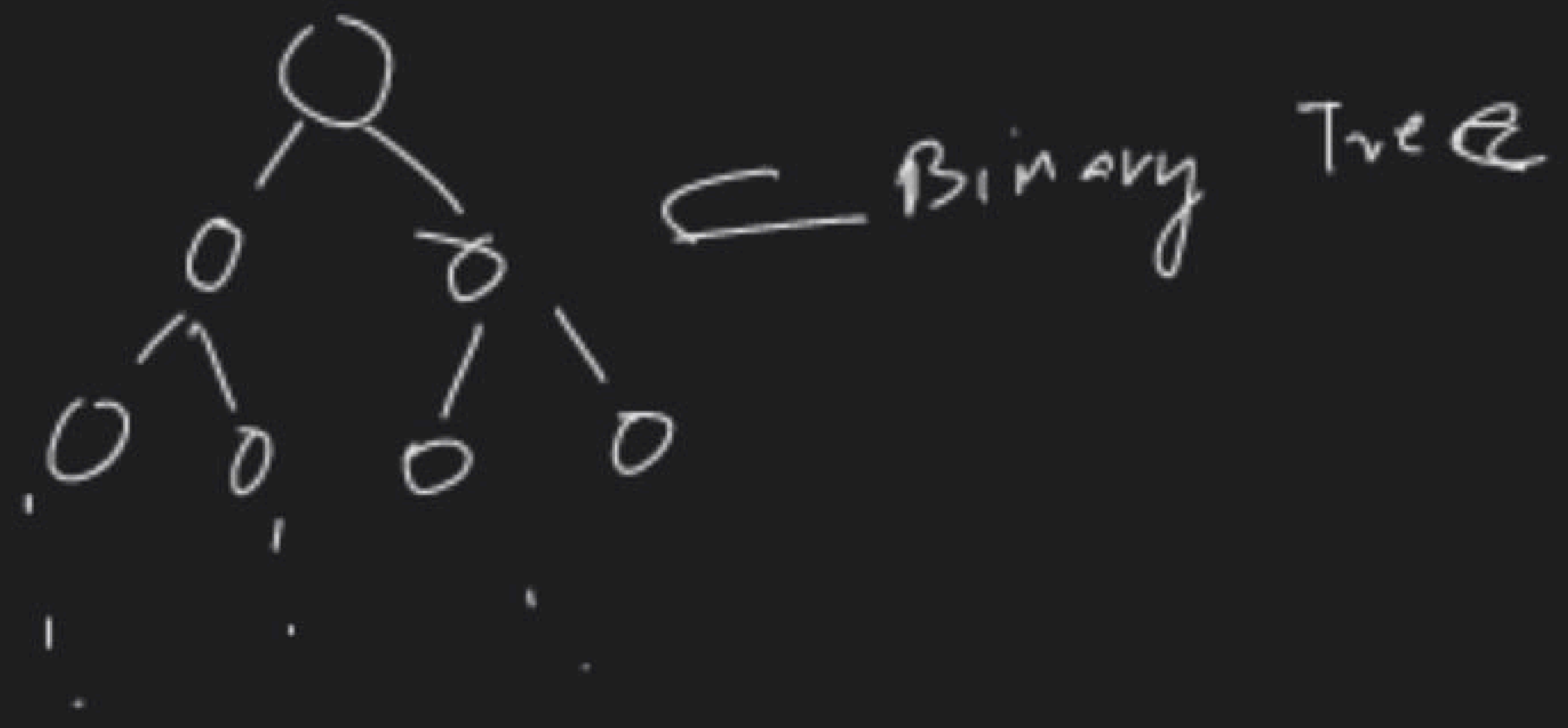
1. What are heaps? What is a binary Heap?
2. Representation of Heaps.
3. Operations on Heaps.
4. Implementing Heaps from scratch.
5. Building a heap from an array.
6. HeapSort.

Heaps - "Just another Tree with some specific properties"

1. Binary Heaps
2. k-ary Heaps
3. Fibonacci Heaps
4. Leftist Heaps
5. Binomial Heaps
6. Brodal Heaps, etc..

Binary Heaps:

1. A binary heap is a complete binary tree.
2. It is either a max-heap or a min-heap.

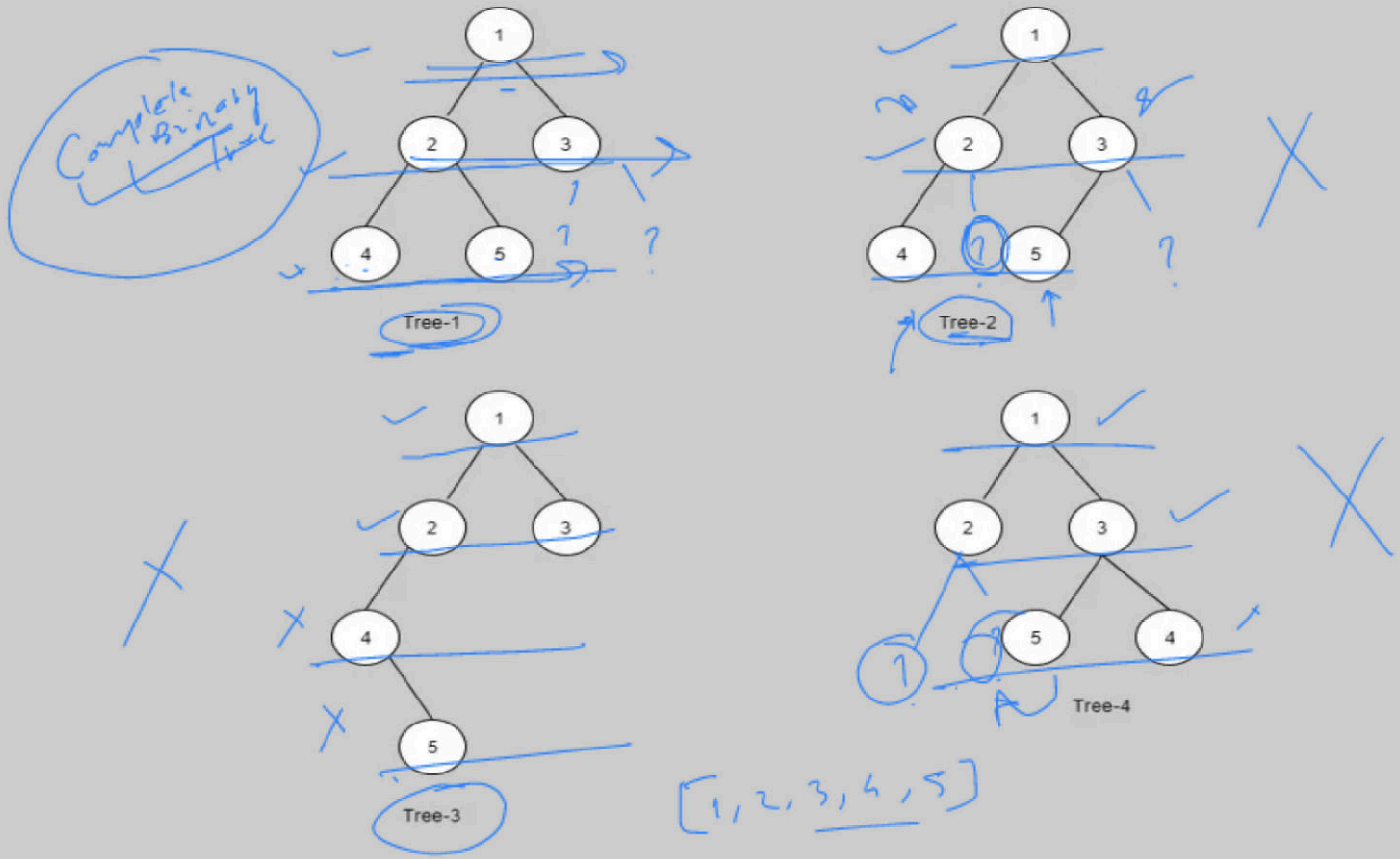

Binary Tree

What is a complete binary tree?

- All levels are completely filled except possibly the last level and the last level has all keys as to the left as possible.

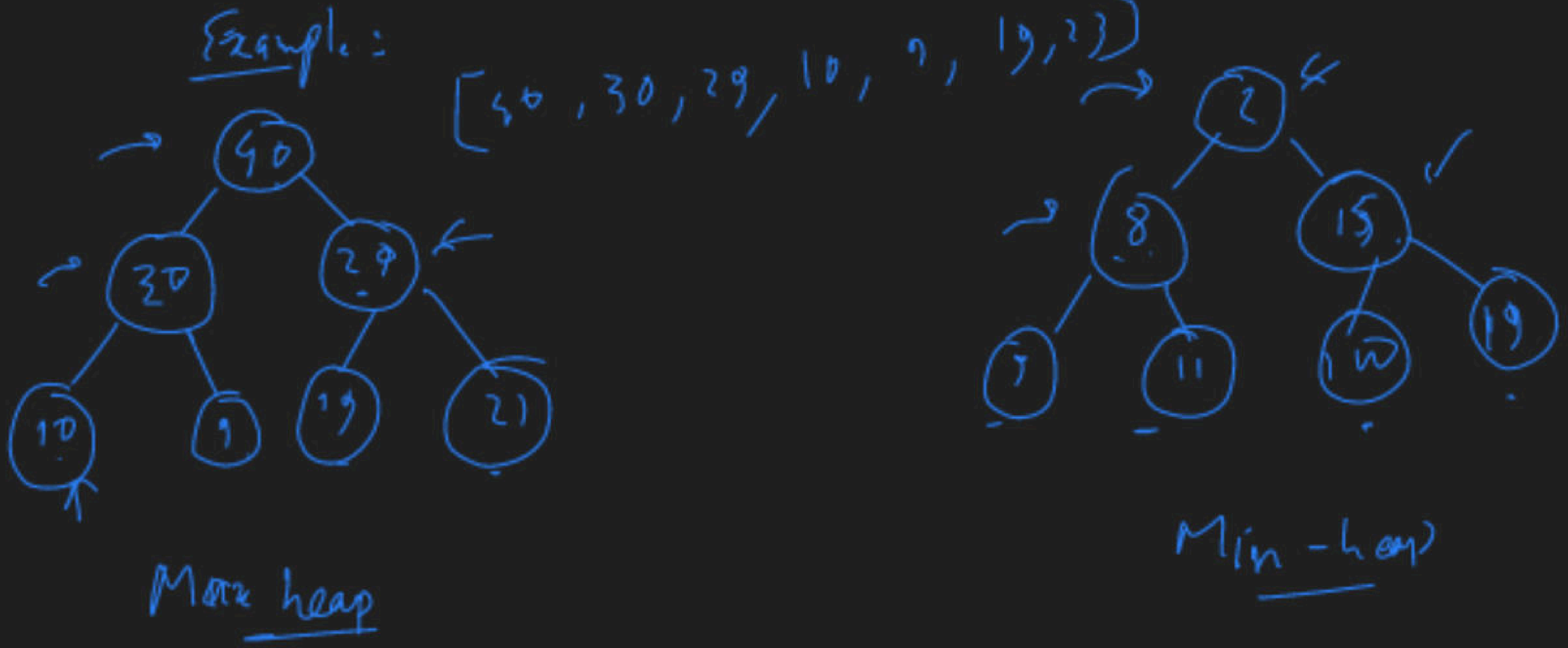
- Completely filled levels → No more vacancies to add another node to the level

- In the last level, there is no vacancy present before any of the nodes


Complete Binary Tree
Tree-1
Tree-2
Tree-3
Tree-4
[1, 2, 3, 4, 5]

Max-heap: The value of the root is greater than or equal to the values of the either children. This is recursively followed.

Min-heap: The value of the root is lesser than or equal the values of the either children. This is recursively followed.



Representation of Heaps

Do we create TreeNodes?

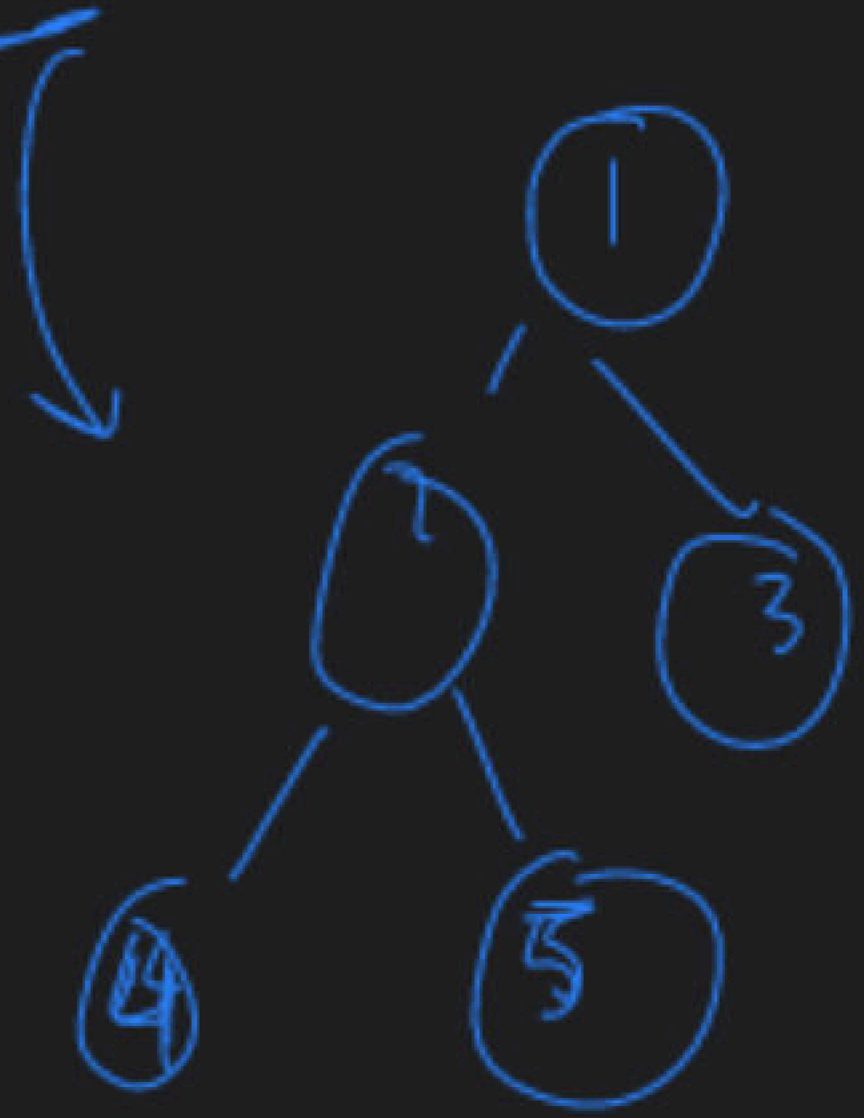
```
class TreeNode {

    int value;

    TreeNode left, right;

}
```

→ Objects of this class to represent the nodes in the tree.

node.left

A complete binary tree can be represented using JUST the Level order Traversal of the tree!

L.O.T = [1, 2, 3, 4, 5]

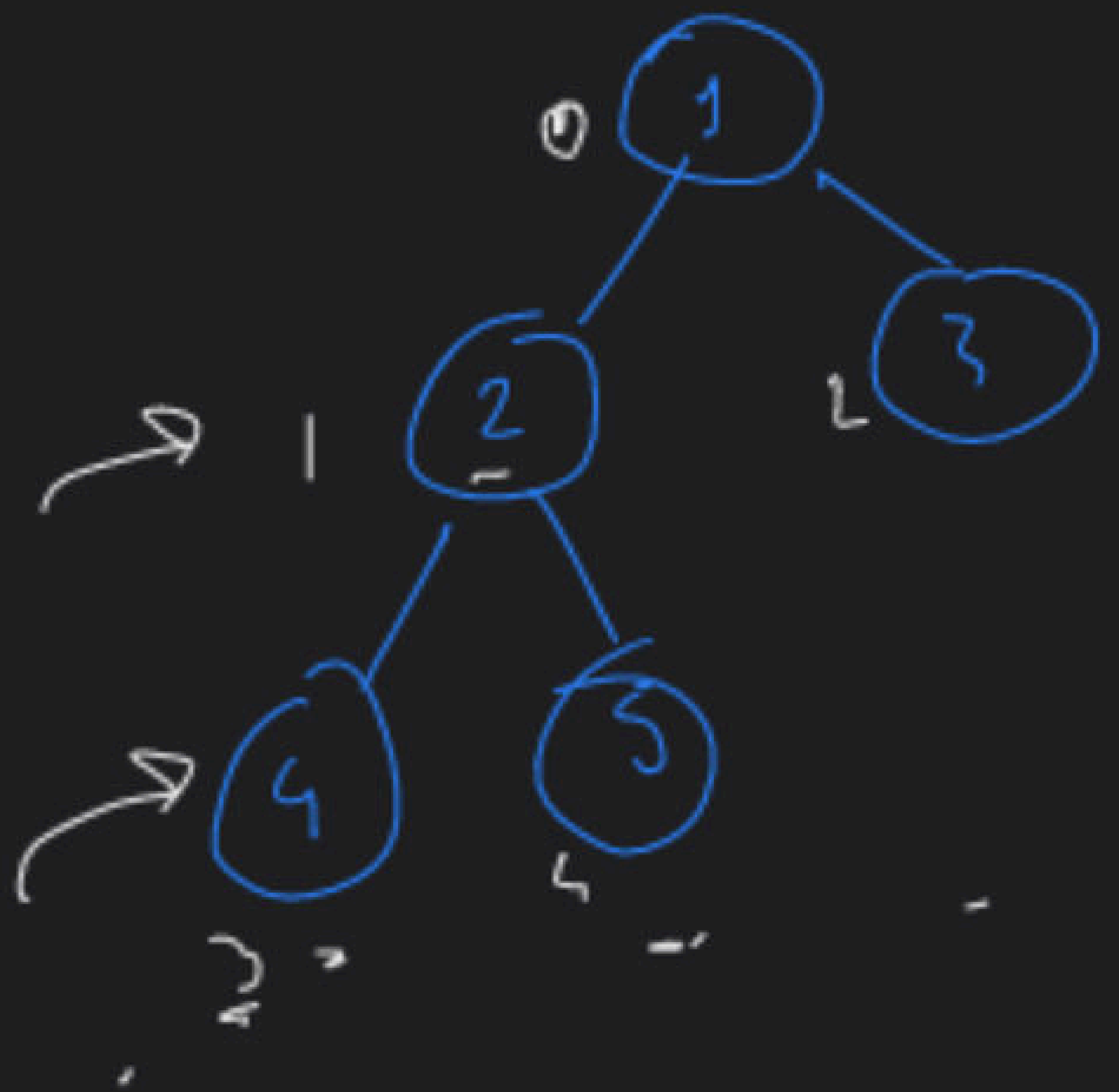


$\boxed{\text{Heap} \rightarrow \text{L.O.T}}$

Node → i

L.O.T [1, 2, 3, 4, 5]

$$\begin{array}{ccccc} 1, & 2, & 3, & 4, & 5 \\ 0 & 1 & 2 & 3 & 4 \end{array}$$



Node i

$$\boxed{\begin{aligned} left(i) &= 2 * i + 1 \\ right(i) &= 2i + 2 \end{aligned}}$$

$$left(1) = 2(1) + 1 = 3$$

$$right(1) = 2(1) + 2 = 4$$

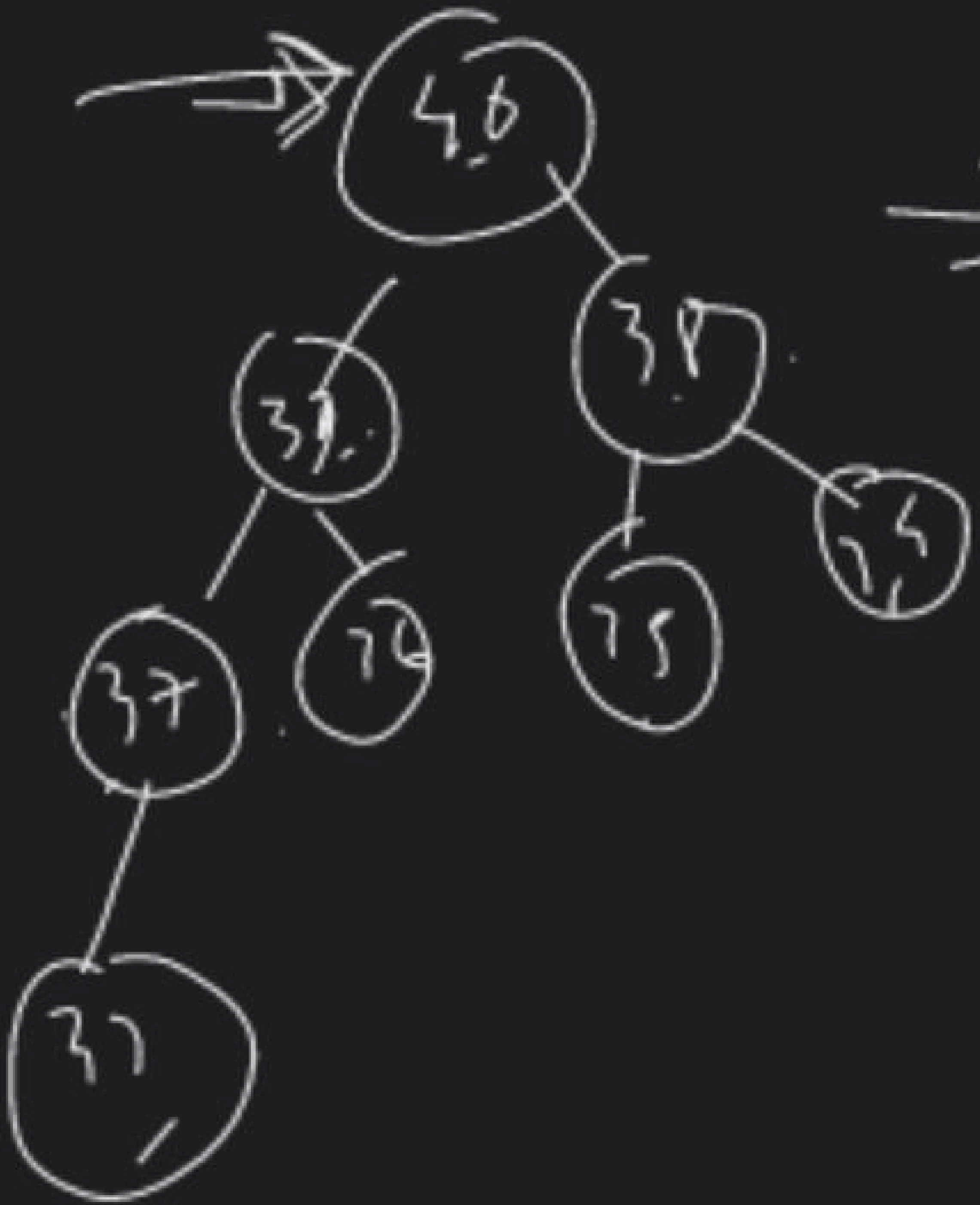
$$\boxed{parent(i) = \frac{(i-1)}{2}} \quad \left\lfloor \left(\frac{i-1}{2}\right) \right\rfloor$$

$$parent(3) = \frac{3-1}{2} = 1$$

Operations on Heaps: (Max-Heaps)

1. getMax() → arr[0] → $O(1)$
2. extractMax() $O(\log N)$
3. insert(value) $O(\log N)$

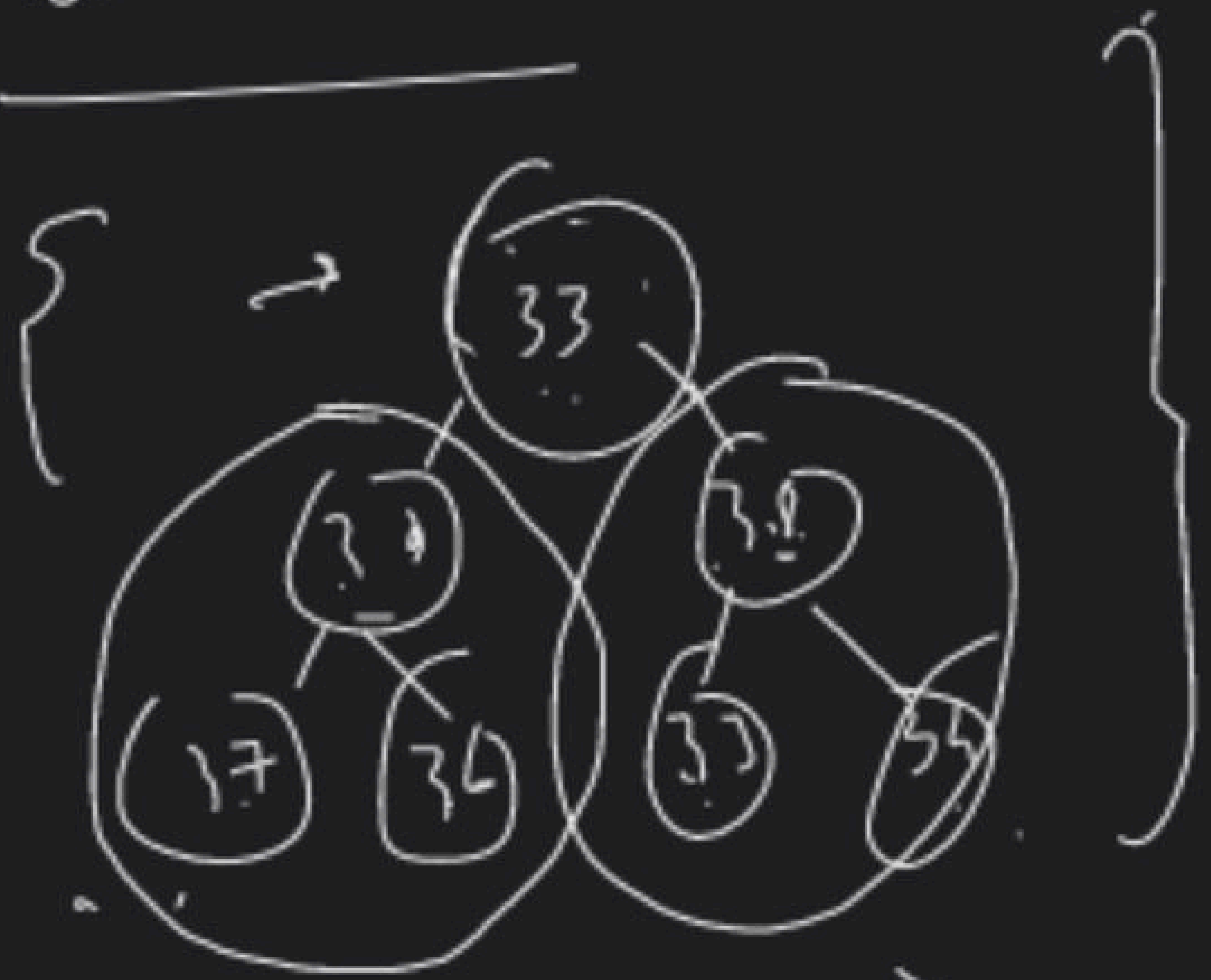
1. getMax():



[40, 39, 38, 37, 36, 35, 34, 33]

LOT[0]

2. extractMax():



$\rightarrow$ heapify() $\longrightarrow$ heap

Heapify!

Heapify(rootIndex):

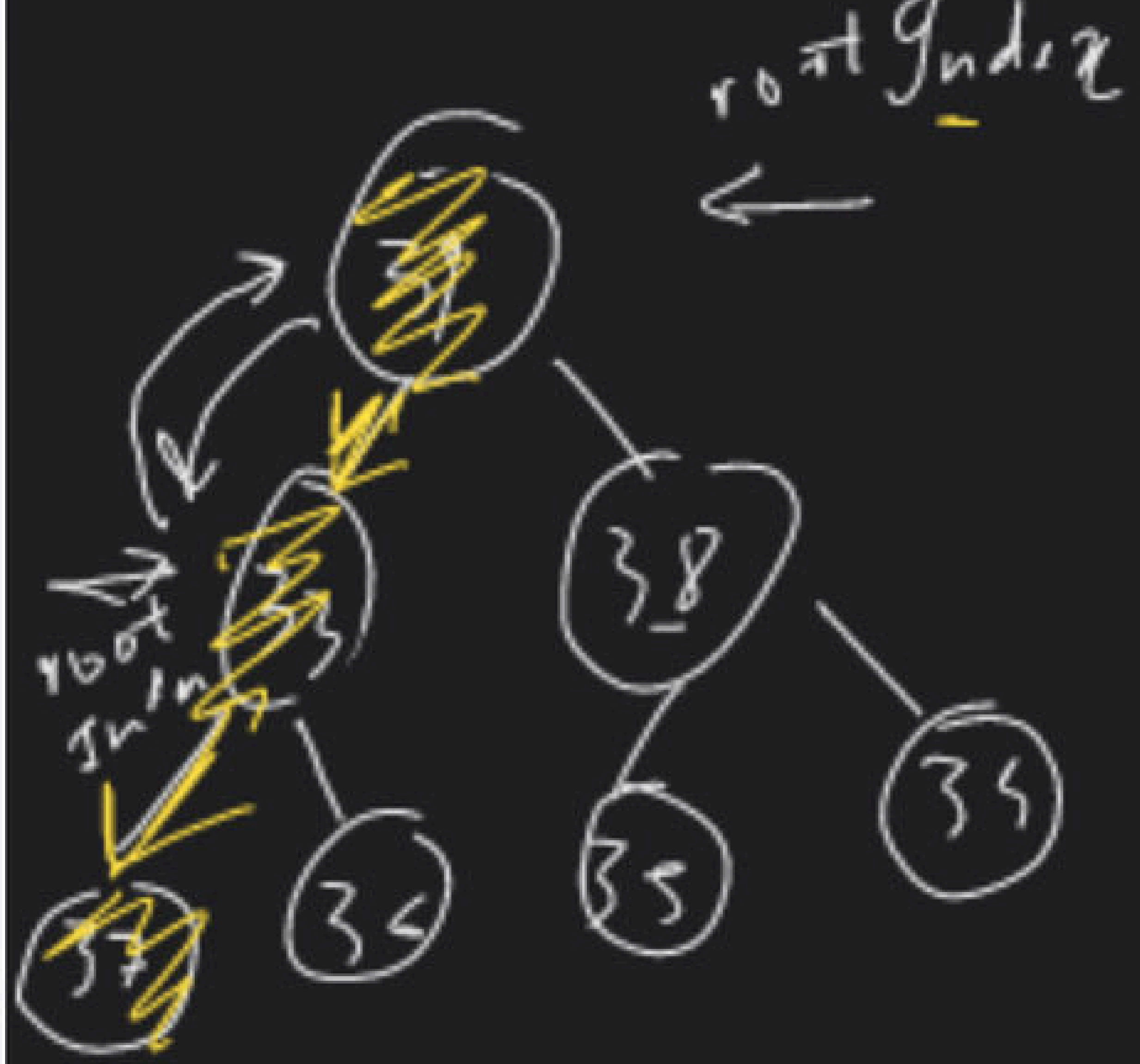
[Note: The following steps work iff all the subtrees below the root are heaps]

1. Select the largest of the 3 values, A[rootIndex], A[left(rootIndex)] and A[right(rootIndex)]

2. If the largest value is the root itself then return.

3. Swap the largest value with the root value.

4. Now, call recursively heapify on subtree whose root was swapped.



1. largest (33, 39, 38) = 39

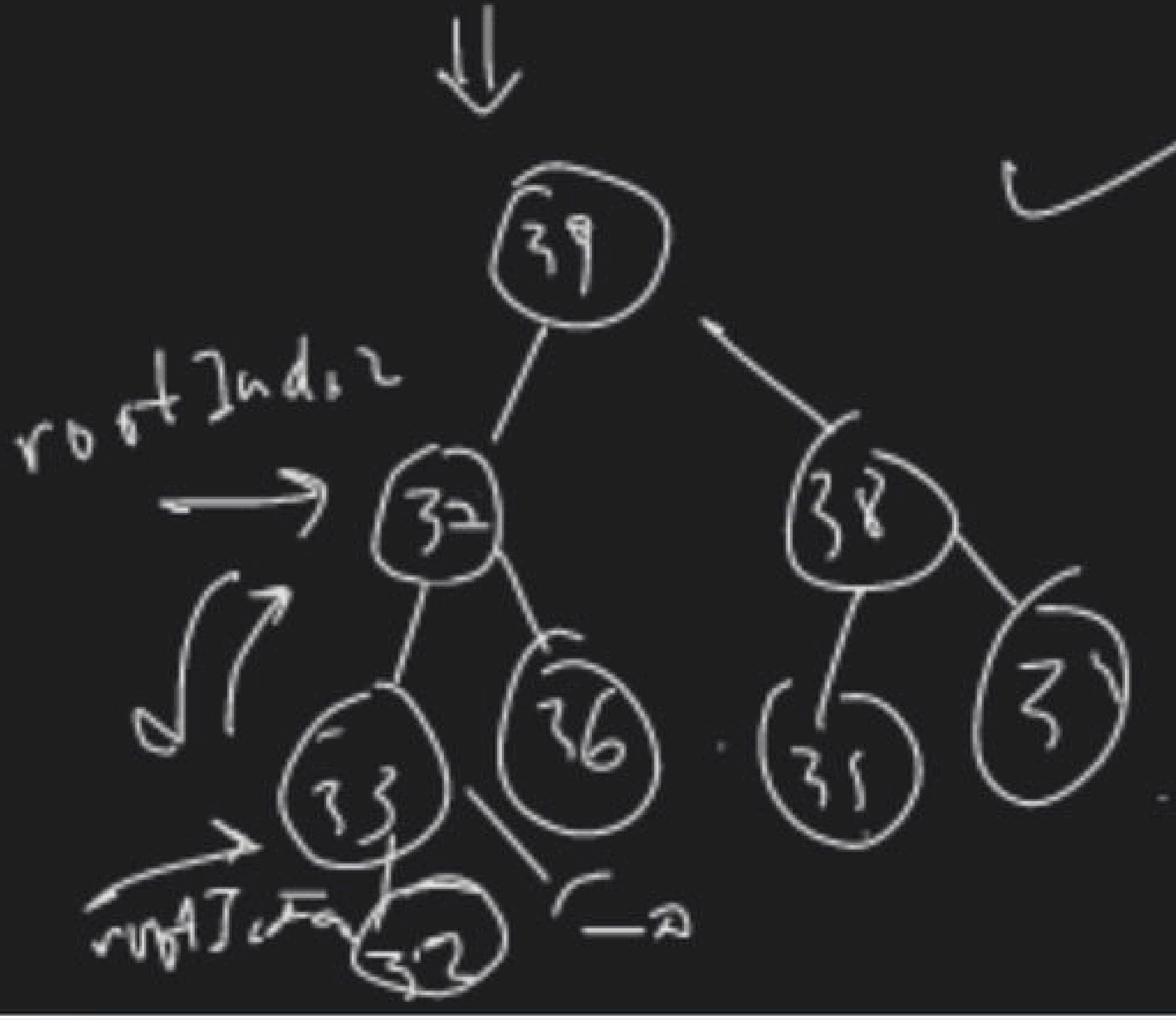
2.

3.

4.

heapify() → along a path from root to leaf

O(h) : h → Height of the tree



1. largest (33, 37, 36) = 37

2.

3. swap (37, 33)

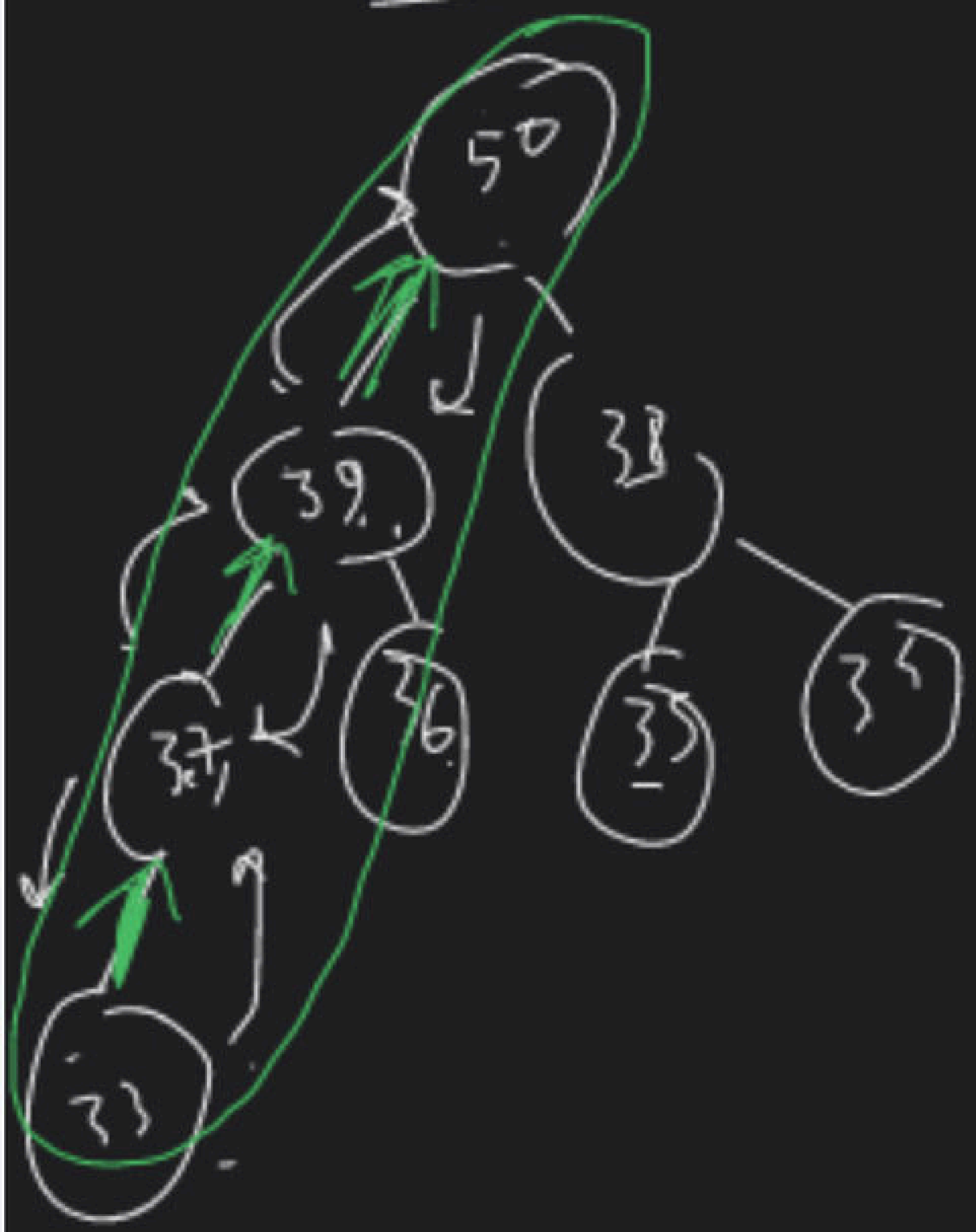
$h = \log_2 N$

$O(\log_2 N)$

1. largest (33, 32, _) = 33

3. Insert: insert(50)

[39, 37, 38, 33, 36, 35, 34, 50]



1. compare i, parent(i)

$O(h) \rightarrow O(\log_2 N)$

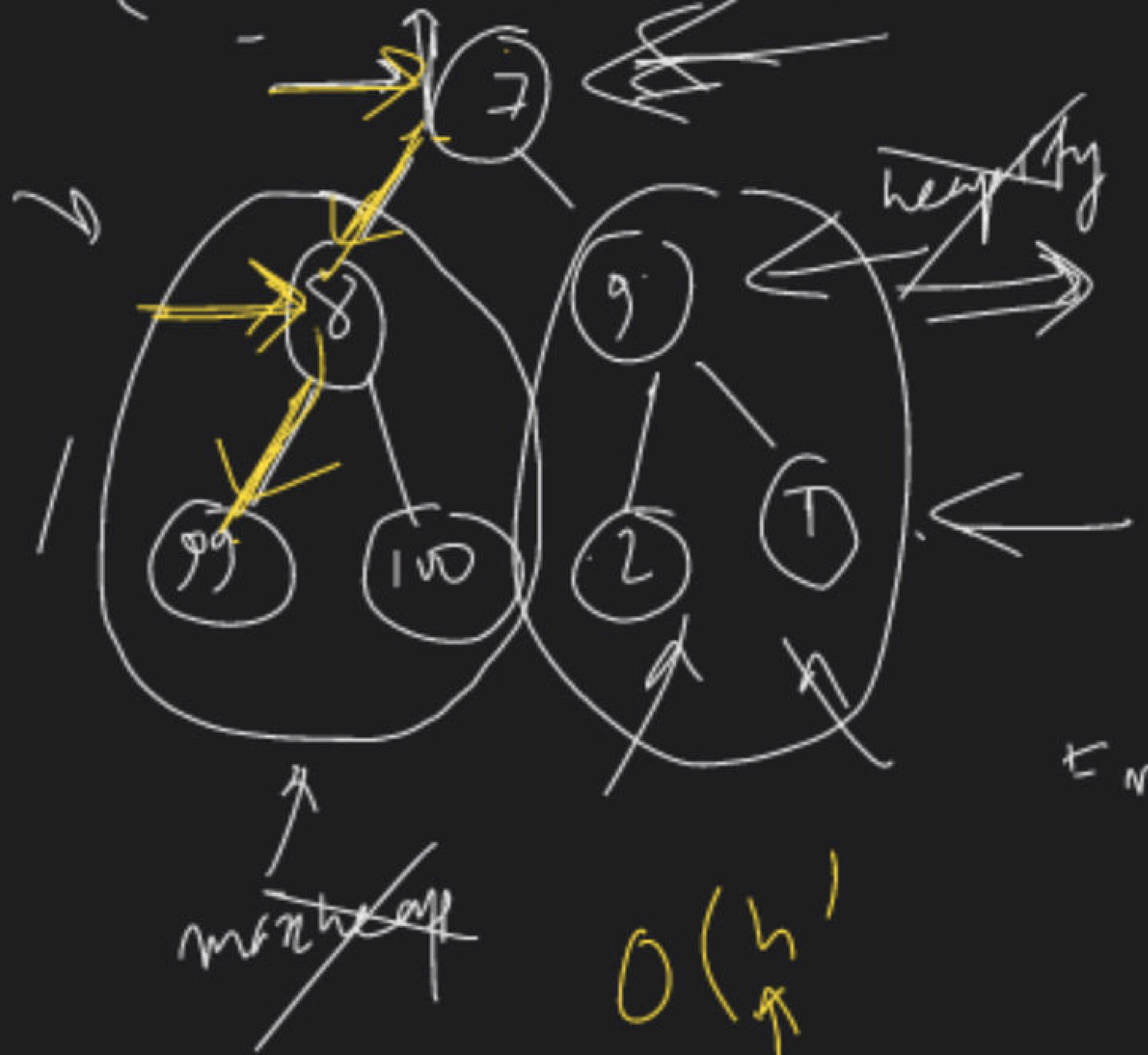
Implementing heaps from scratch

Building Heaps from an array

$$\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 7, & 8, & 9, & 99, & 100, & 2, & 1 \end{array}$$

$\Rightarrow N = 7$

$\left\lfloor \frac{N}{2} - 1 \right\rfloor = \frac{7}{2} - 1 = 2$



heap

$O(h)$

- Convert subtrees into heaps

from $\frac{N}{2}$ th element → leaves

The buildHeap() function: → $O(N)$

```
buildHeap(int[] arr, int n){
    for (int i = n / 2 - 1; i >= 0; i--){
        heapify(arr, n, i);
    }
}
```

$O(\log_2 N)$

Oh! why $\frac{N}{2} - 1$?

why not N ?

Ignore the leaves

Time Complexity of buildHeap() :

Trivial → $O(N \log N)$

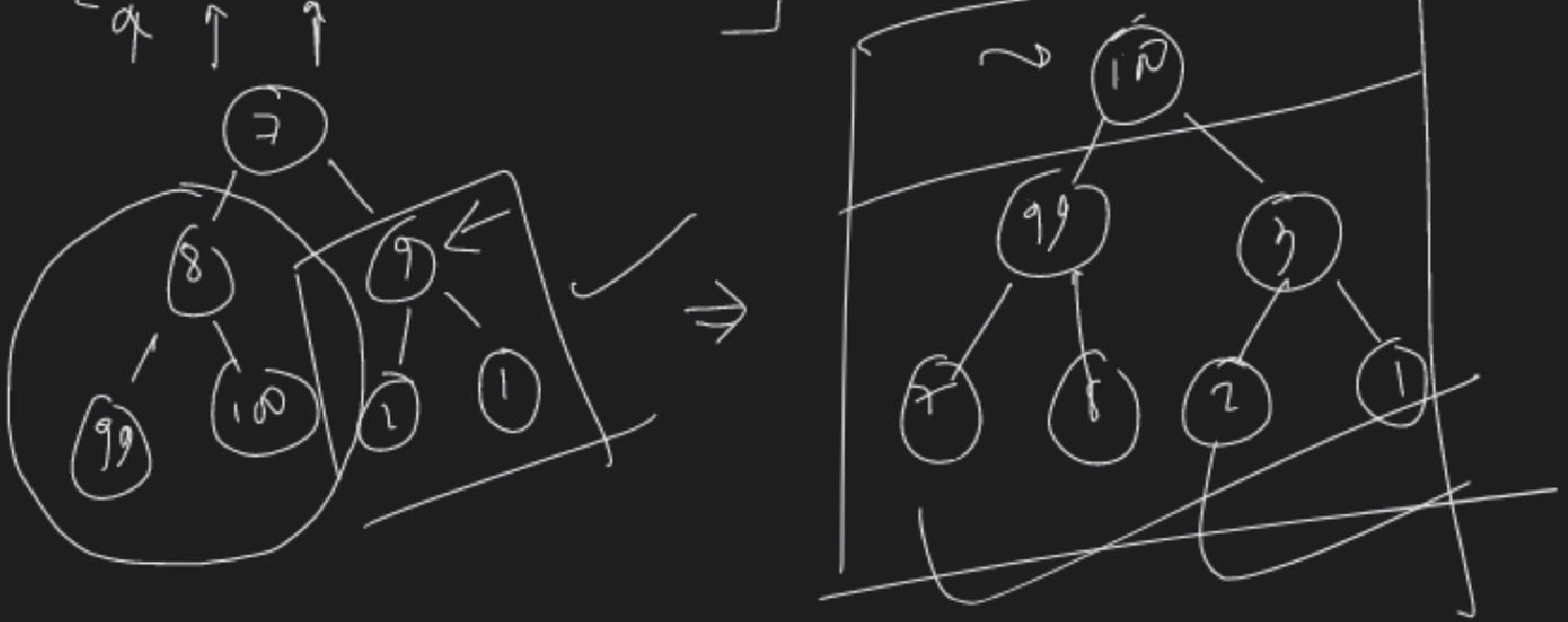
We can find a much tighter bound than this

Actual → $\boxed{O(N)}$

$$\sum_{i=0}^{\frac{N}{2}-1} \text{height}(i) = \sum_{h=0}^{\log N} \left\lceil \frac{N}{2^{h+1}} \right\rceil * h$$

In a complete binary tree the number of nodes at a height h $= \left\lceil \frac{N}{2^{h+1}} \right\rceil$

[7, 8, 9, 99, 100, 2, 1]



$$\sum_{h=0}^{\log N} \left\lceil \frac{N}{2^{h+1}} \right\rceil \times h$$

$$= O\left(\sum_{h=0}^{\log N} \frac{N}{2^{h+1}} \cdot h \right)$$

$$= O\left(N \sum_{h=0}^{\log N} \frac{h}{2^h \cdot 2} \right)$$

$$= O\left(N \sum_{h=0}^{\log N} \frac{h}{2^h} \right)$$

$$= O\left(N \sum_{h=0}^{\log N} \frac{h}{2^h} \right)$$

$$= O\left(N \sum_{h=0}^{\infty} \frac{h}{2^h} \right) = O(N \times 2)$$

$$= O(2N)$$

$$= \boxed{O(N)}$$

$$\sum_{n=0}^{\infty} x^n \, ; \quad x < 1$$

$$= \left(\frac{1}{1 - x} \right)$$

Actual time complexity

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$

Diff. both sides,

$$\sum_{n=0}^{\infty} n x^{n-1} = \frac{x}{(1-x)^2}$$

$$\left.\begin{aligned} n &\to n \\ x &\to \left(\tfrac{1}{2}\right) \end{aligned}\right\} \quad \sum_{n=0}^{\infty} n\left(\frac{1}{2}\right)^{n-1} = \frac{\frac{1}{2}\left(\frac{1}{2}\right)}{\left(1-\frac{1}{2}\right)^2}$$

$$= 2$$

Heap-Sort

1. We convert the given unsorted array to a heap array using the buildheap( ) function $\rightarrow O(N)$

2. Call extractMax() N times. $\rightarrow O(N \log N)$

Space $O(1)$

$A = [9, 99, 1, 2] \rightarrow [1, 2, 9, 99]$

$[99, 9, 2, 1]$

Time $O(N + N\log N) = O(N \log N)$