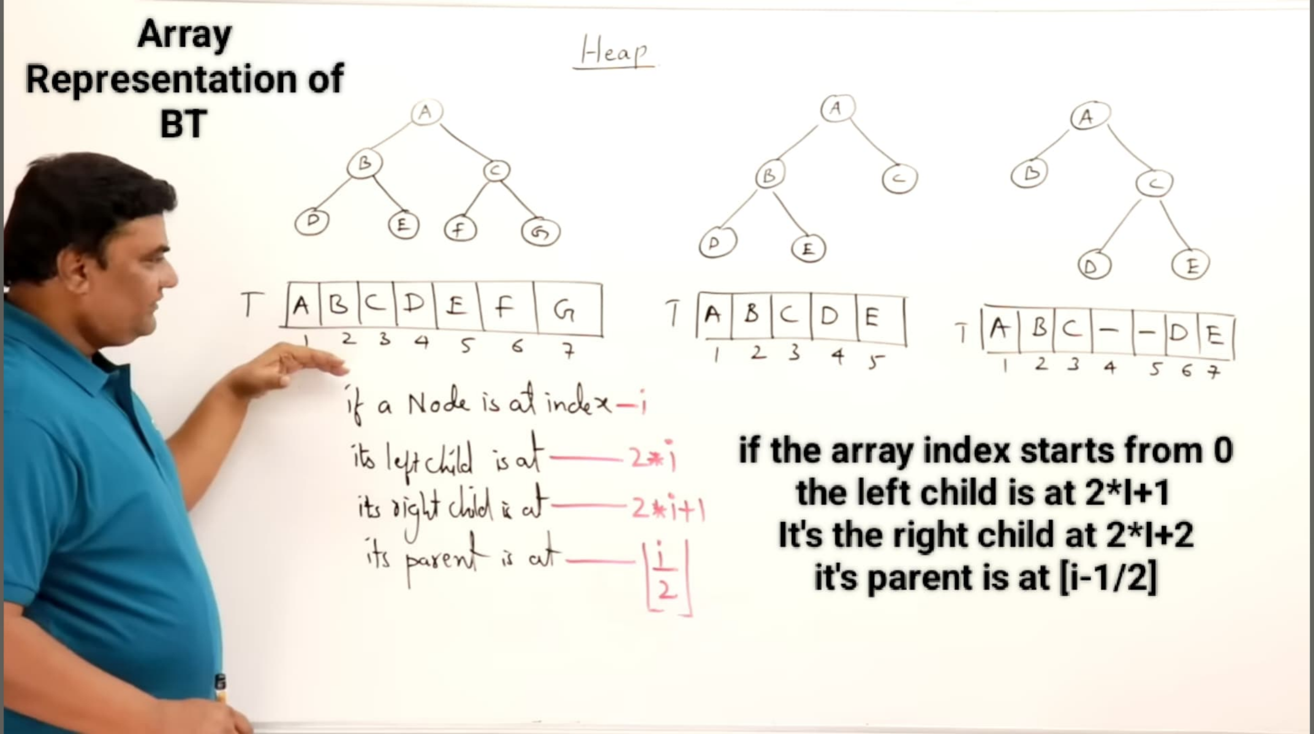
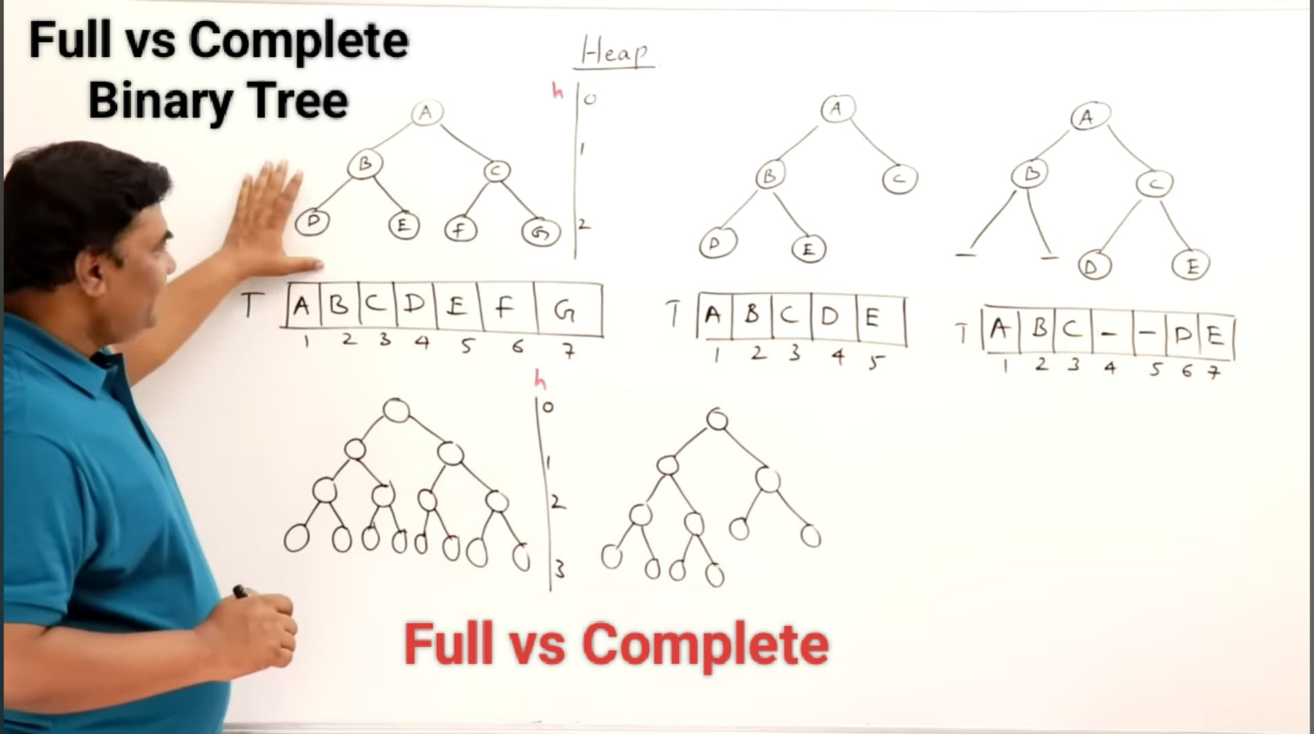
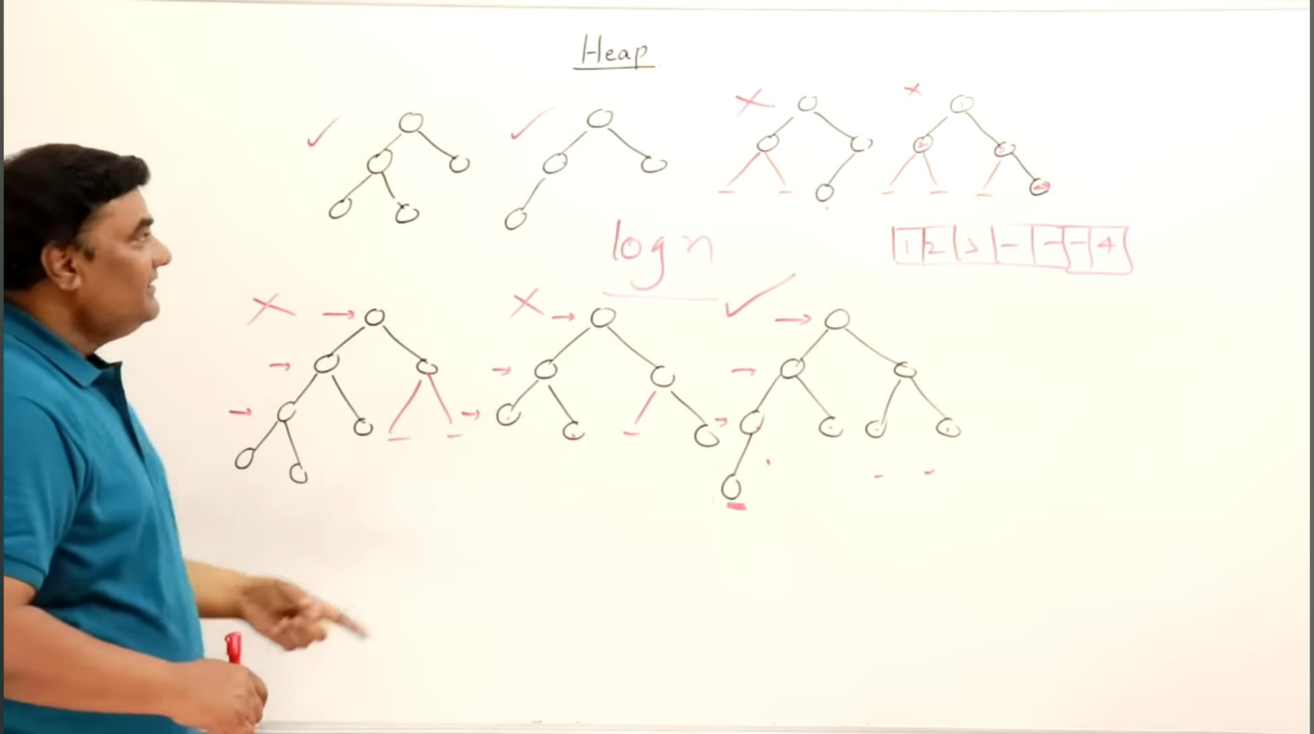
Heap

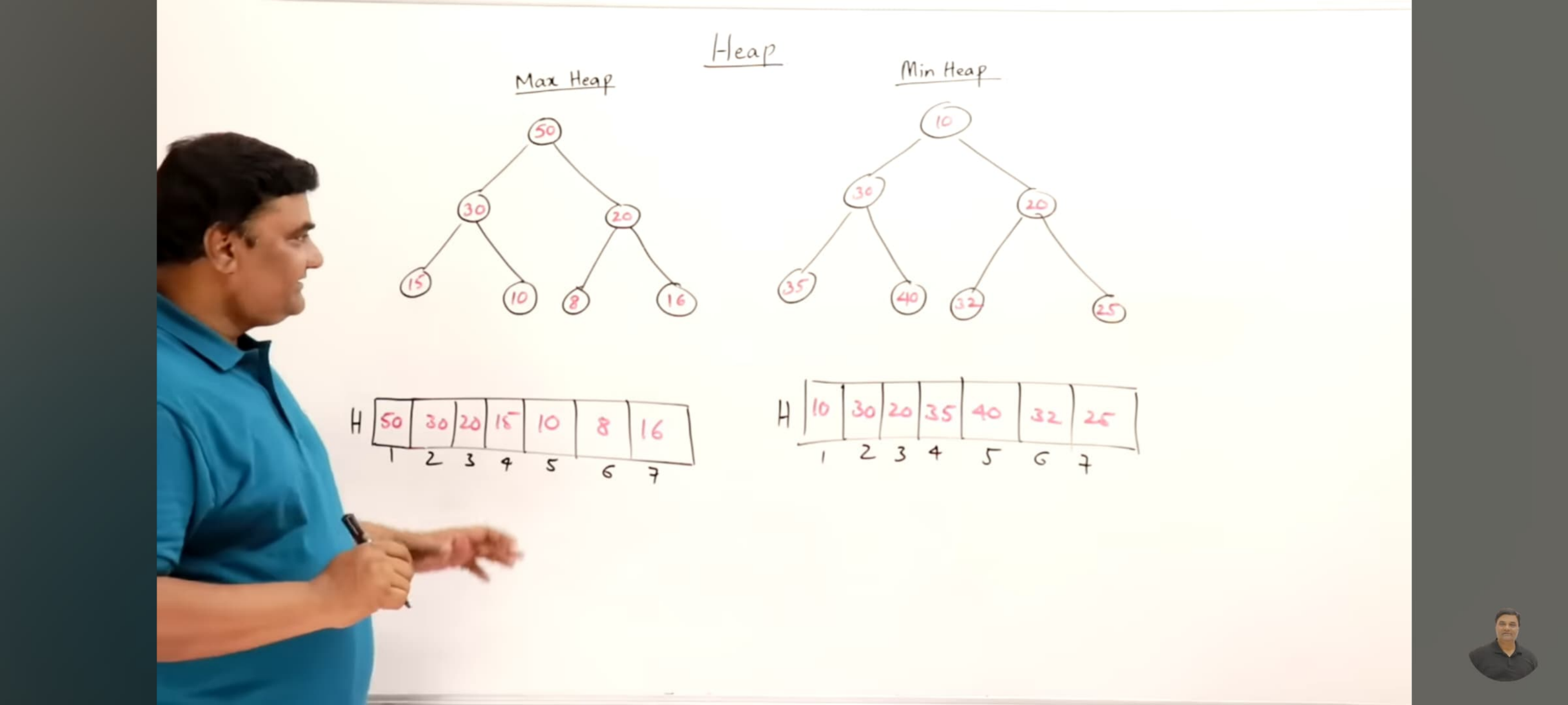
1. Array Representation of B.T
2. Complete Binary Tree
3. Heap
4. Insert & Delete
5. Heap Sort
6. Heapify
7. Priority Queue

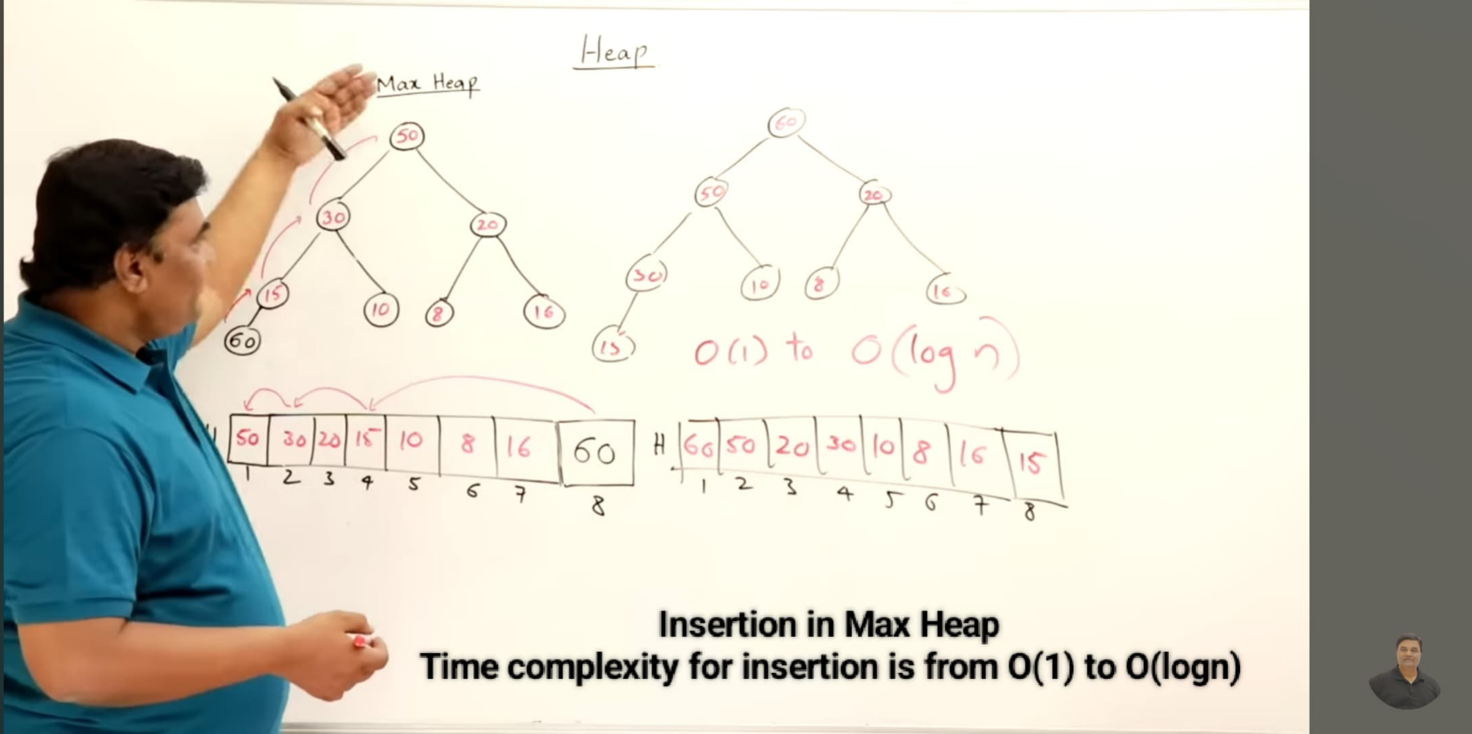


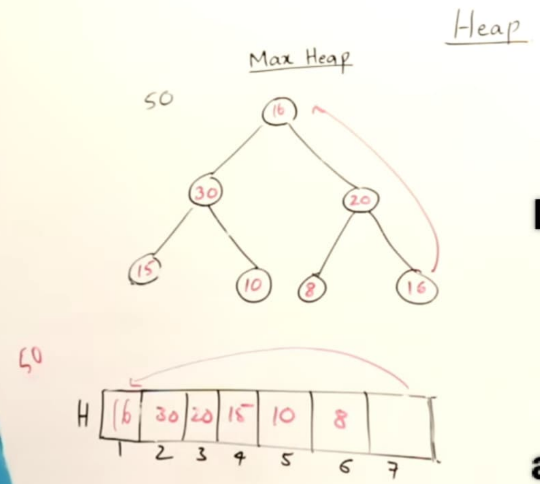
Array Representation of BT
Heap
A
B
C
D
E
F
G
T
A B C D E F G
1 2 3 4 5 6 7
A B C D E
1 2 3 4 5
A B C - - D E
1 2 3 4 5 6 7
if a Node is at index — i
its left child is at —— 2*i
its right child is at —— 2*i+1
its parent is at —— ⌊i/2⌋
if the array index starts from 0
the left child is at 2*I+1
It's the right child at 2*I+2
it's parent is at [i-1/2]


Full vs Complete
Binary Tree
Heap
h
0
1
2
A
B
C
D
E
F
G
T
A B C D E F G
1 2 3 4 5 6 7
A
B
C
D
E
T
A B C D E
1 2 3 4 5
A
B
C
D
E
T
A B C - - D E
1 2 3 4 5 6 7
h
0
1
2
3
Full vs Complete



Heap
log n
1 | 2 | 3 | – | – | – | 4

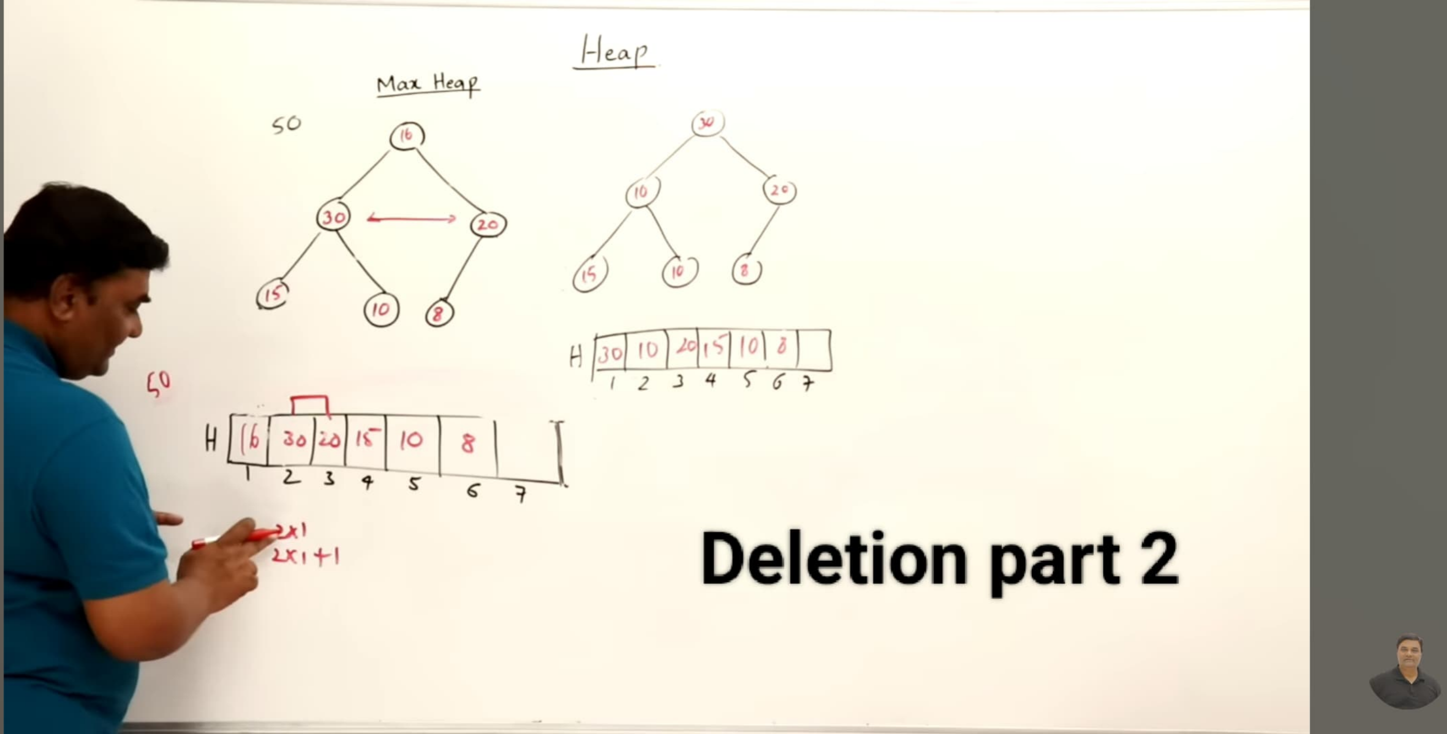

Heap
Max Heap
50
30
20
15
10
8
16
H
50 30 20 15 10 8 16
1 2 3 4 5 6 7
Min Heap
10
30
20
35
40
32
25
H
10 30 20 35 40 32 25
1 2 3 4 5 6 7


Heap
Max Heap
O(1) to O(log n)
H 50 30 20 15 10 8 16 60
1 2 3 4 5 6 7 8
H 60 50 20 30 10 8 16 15
1 2 3 4 5 6 7 8
Insertion in Max Heap
Time complexity for insertion is from O(1) to O(logn)

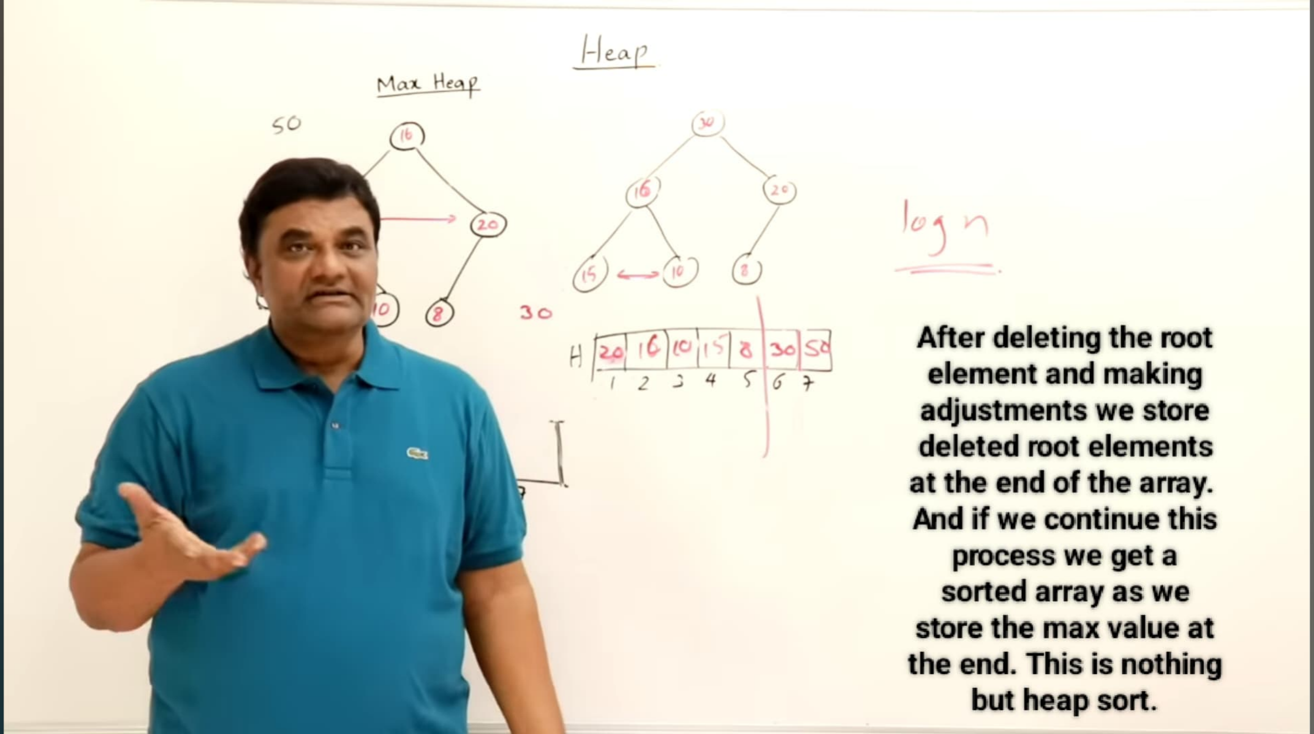


Deletion in Max Heap
Deletion is a top down approach
Time complexity is O(logn)
which is the height of the tree

In this example 50 is deleted and 16 is brought up as root and then root value is compared with its child and adjustments are made.



Heap
Max Heap
50
16
30
20
15
10
8
30
10
20
15
10
8
H | 30 | 10 | 20 | 15 | 10 | 8 |
1 2 3 4 5 6 7
50
H | 16 | 30 | 20 | 15 | 10 | 8 |
1 2 3 4 5 6 7
2×1
2×1+1
Deletion part 2


Heap
Max Heap
50
30
log n
H
20 16 10 15 8 30 50
1 2 3 4 5 6 7
After deleting the root element and making adjustments we store deleted root elements at the end of the array. And if we continue this process we get a sorted array as we store the max value at the end. This is nothing but heap sort.

You

In Max Heap we store the deleted root at the end of the array and if we continue this process we get a sorted array and this process is called heap sort

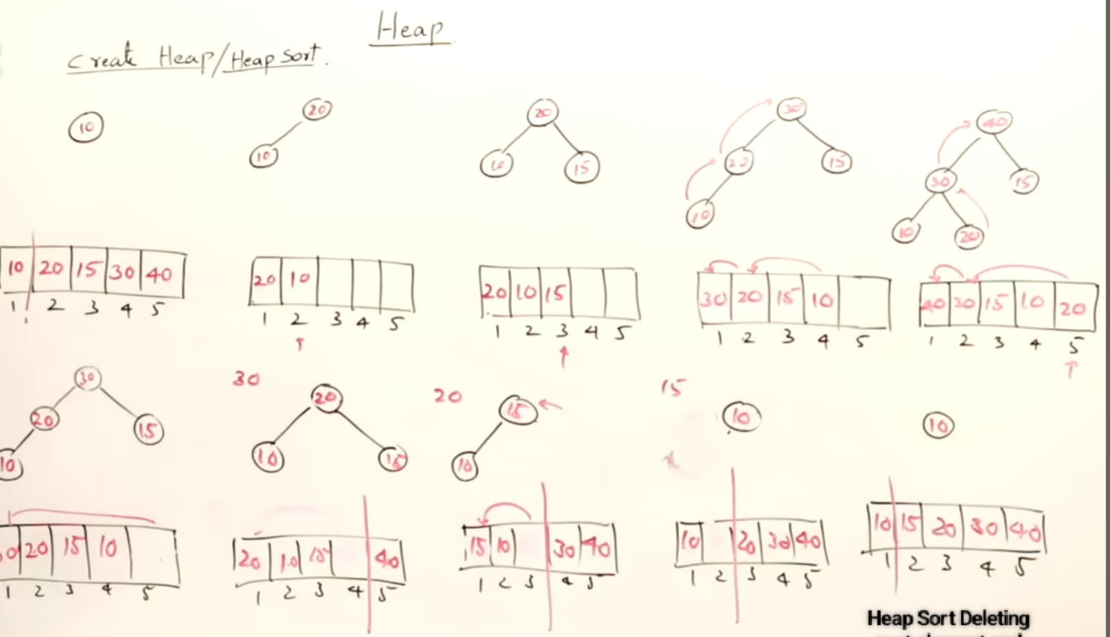
ChatGPT

Yes, exactly! In the heap sort algorithm, after each extraction of the maximum element from the max heap, which is always the root, we swap it with the last element of the array. By doing this repeatedly and reducing the size of the heap, we effectively partition the array into two parts: one sorted (at the end of the array) and one unsorted (remaining elements in the heap).

The sorted portion of the array grows from the end towards the beginning, while the unsorted portion shrinks. Eventually, the entire array becomes sorted as we continue extracting elements from the max heap until it is empty.

So, heap sort effectively leverages the heap data structure to achieve a sorted array in ascending order by repeatedly extracting the maximum element and placing it at the end of the array. This process is indeed called heap sort. Thank you for highlighting this key aspect!

Heap Creation by inserting an element at the leaf and adjusting it by comparing it with its parent element.



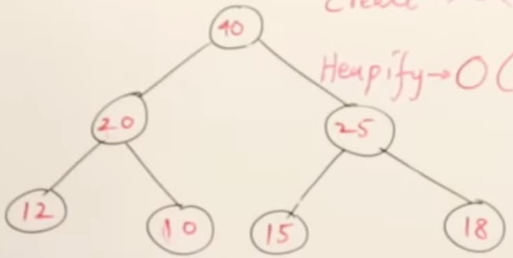
Heap Sort Deleting root element and storing it at the end of the array

Heap

Heapify

create → $O(n \log n)$

Heapify → $O(n)$



H	40	20	25	12	10	15	18
	1	2	3	4	5	6	7

Heap

Priority Queue

smaller number
Higher Priority

Large number
higher priority

$O(n)$

A

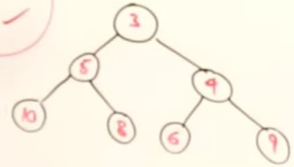
1	2	3	4	5	6	7
8	6	3	10	5	4	9

B

1	2	3	4	5	6	7
8	6	3	10	5	4	9

$\log n$
$\log n$

Min Heap



Max Heap