unacademy

Trees - Part X

Course on Data Structure

Pankaj Sharma • Lesson 30 • Aug 12, 2024

unacademy

CS & IT Engineering

Data Structure

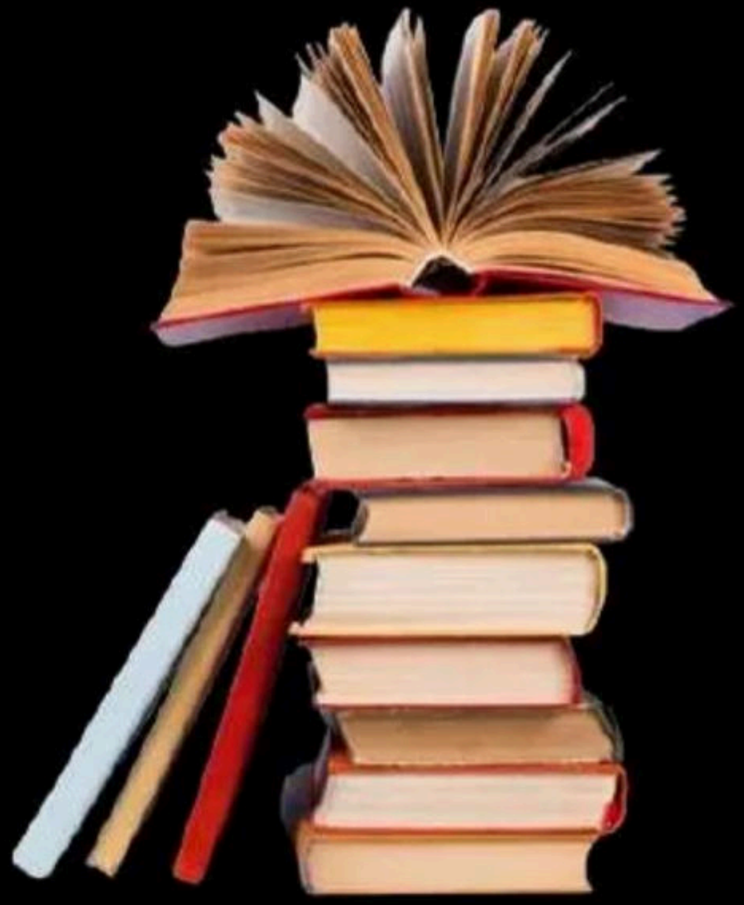
Tree

Lecture Number- 30

By- Pankaj Sir

Topics

to be covered



1. Tree-IX

Const. of heap using build heap method

Given an array representation of a CBT, convert it into a max-heap.

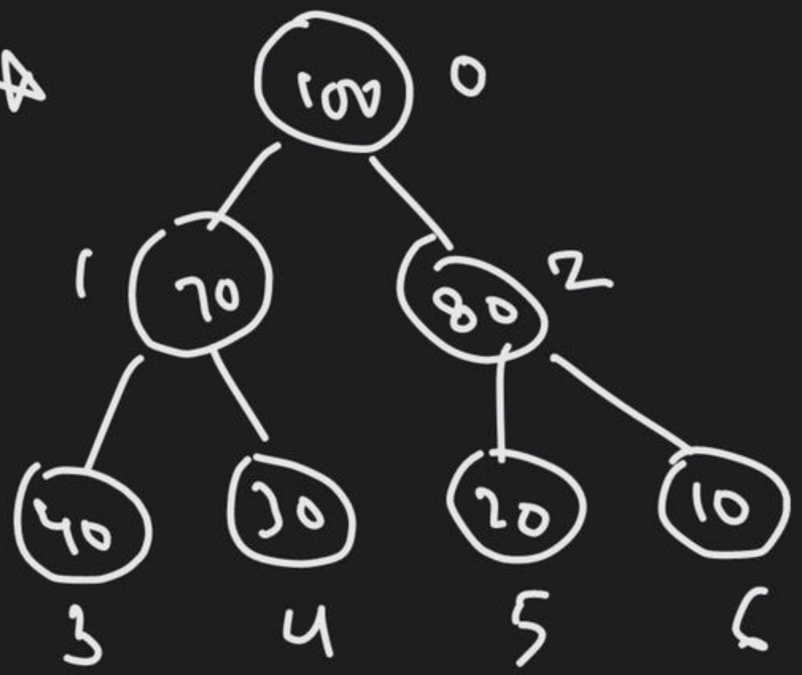
Heapify method / Algo.

unacademy

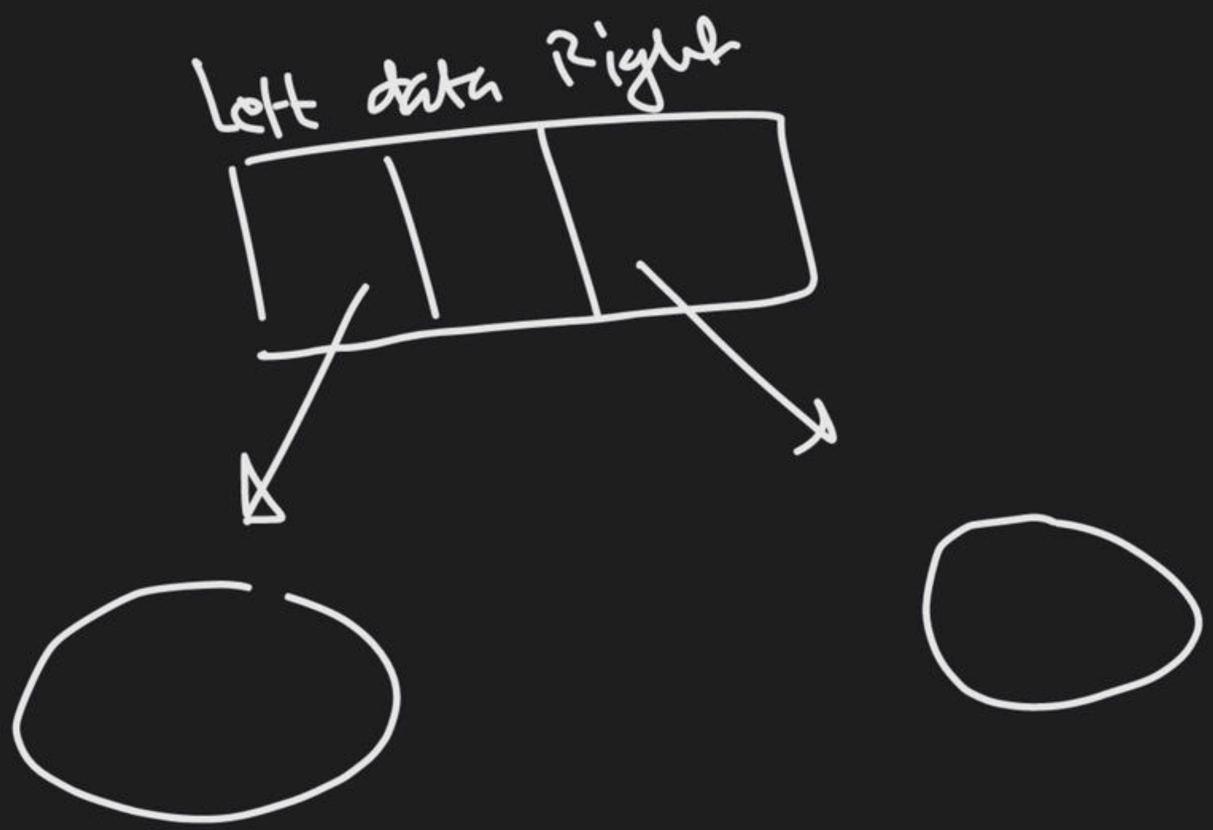
Heap → CBT

→ array representation

100	70	80	40	30	20	10
0	1	2	3	4	5	6

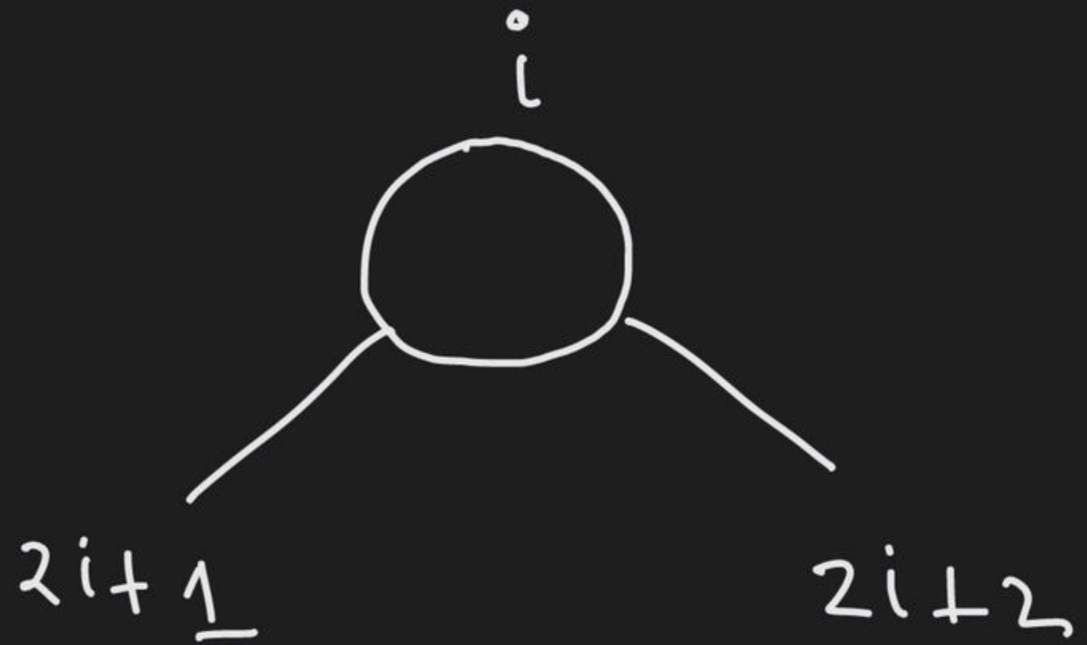


L.L. representation

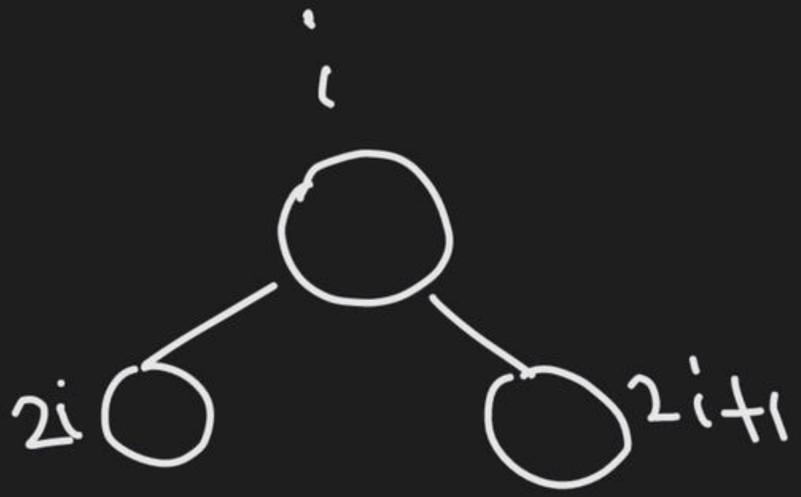


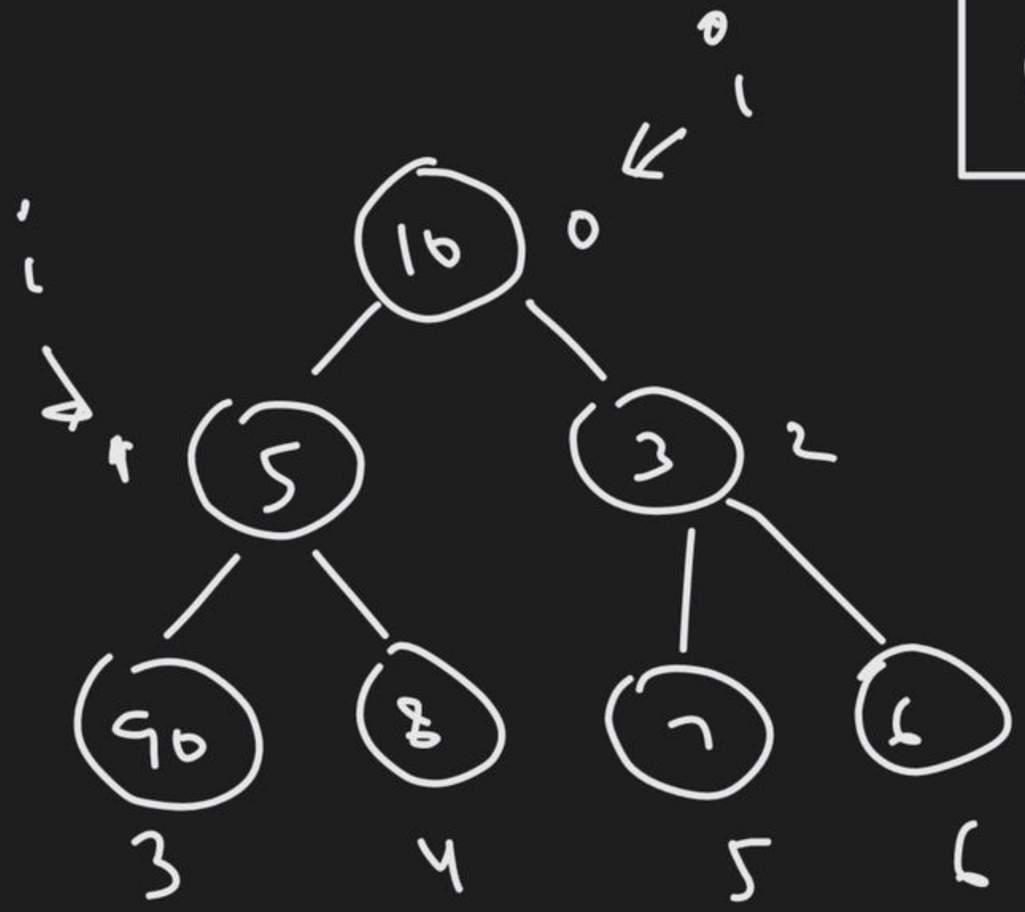
index

Code

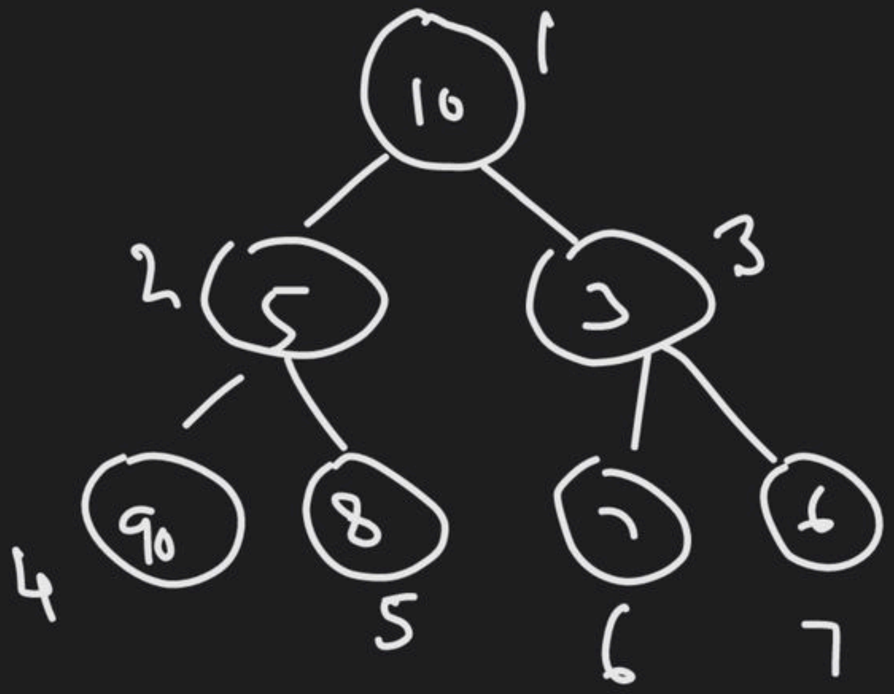


Theory





10	5	3	90	8	7	6
0	1	2	3	4	5	6

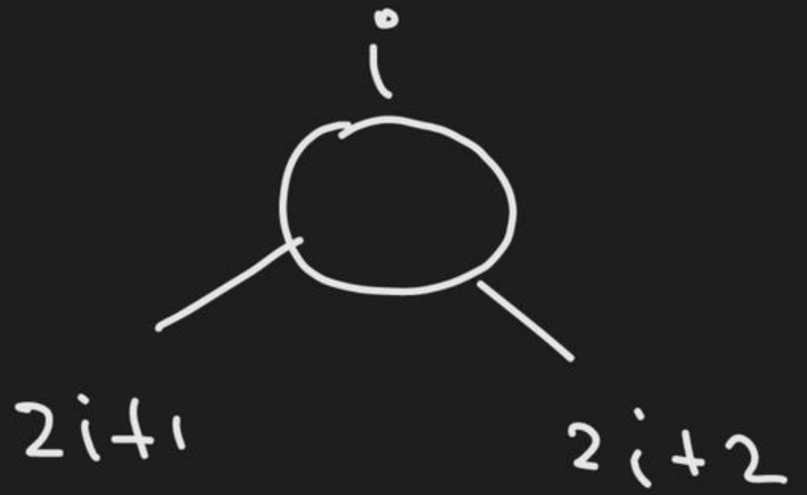


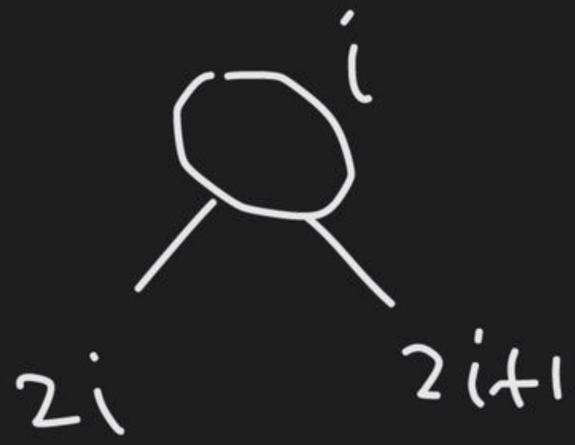
10	5	3	90	8	7	6
1	2	3	4	5	6	7

unacademy

Why not for

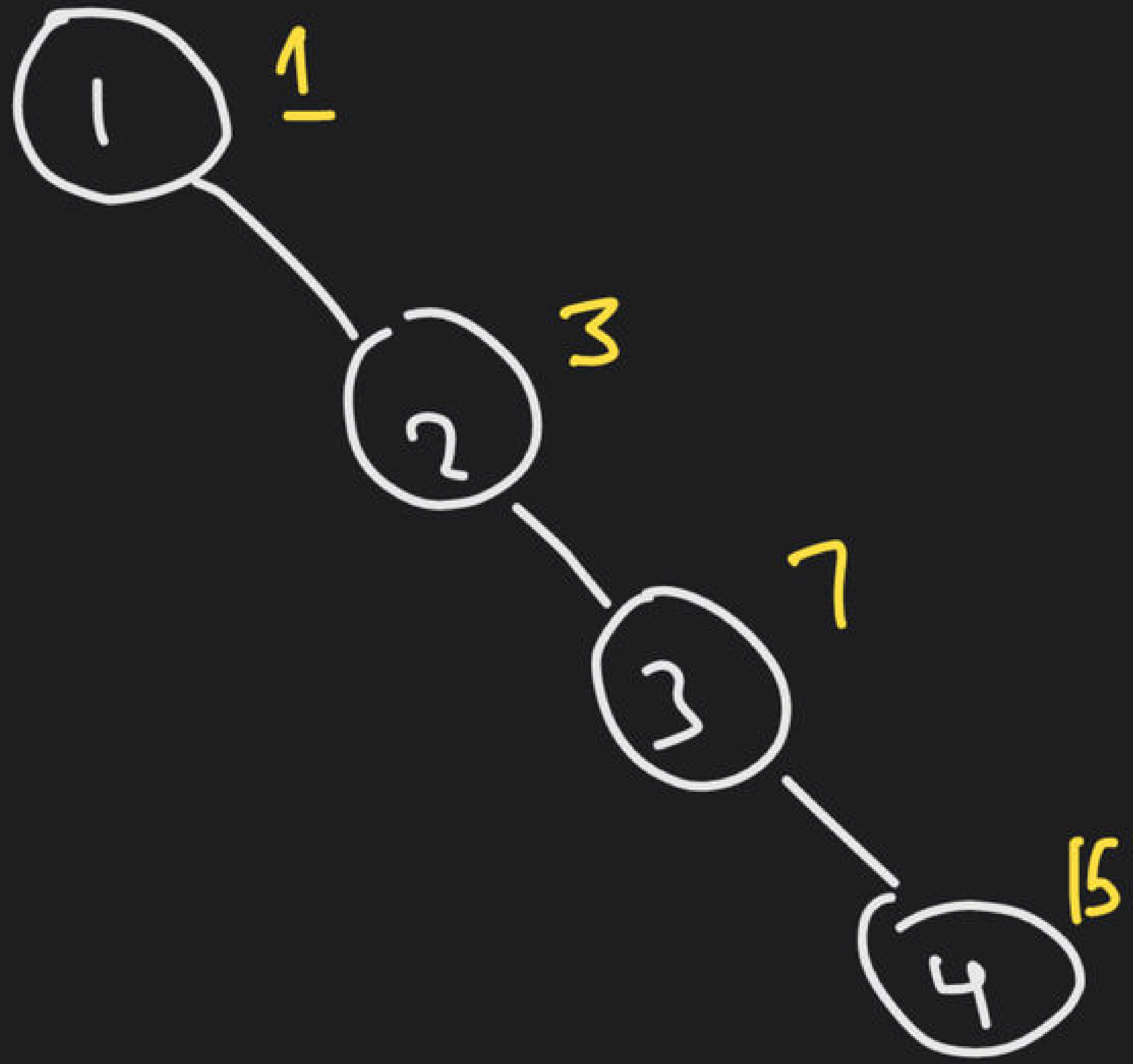
Br ?





$4 \rightarrow \approx 16$ size

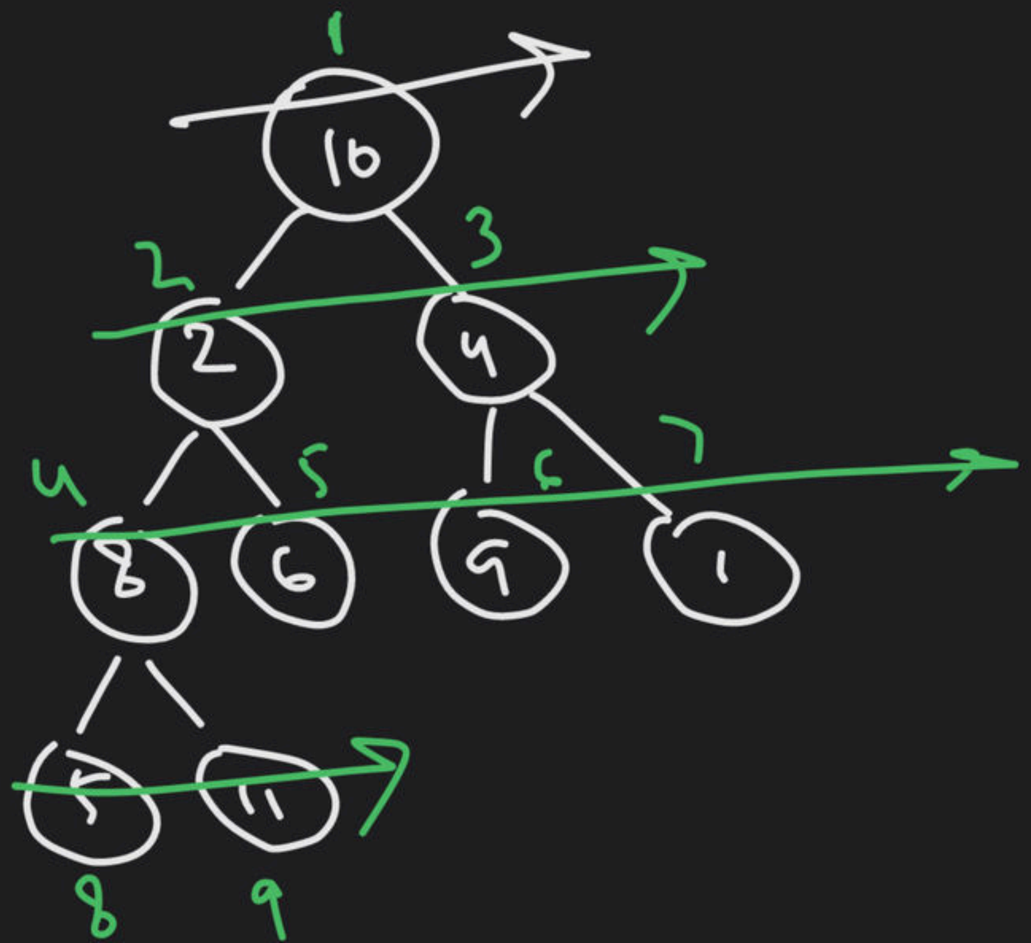
$n \rightarrow O(2^n)$



unacademy

10	2	4	8	6	9	1	5	11
1	2	3	4	5	6	7	8	9

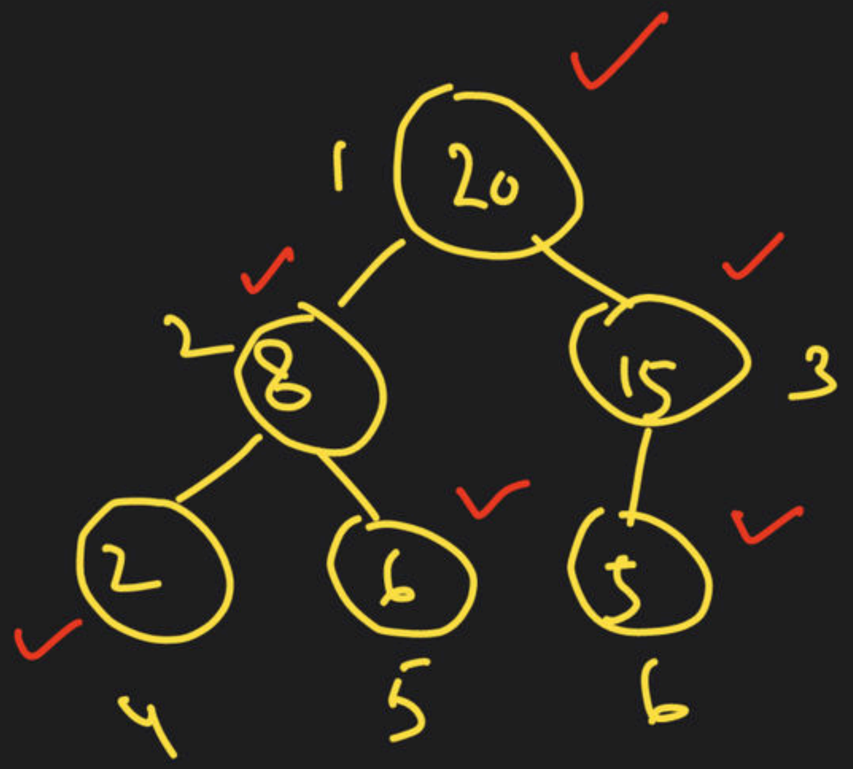
Array represent. of a CBT



Max-heap?
NO

Given an array rep. a CBT 20, 8, 15, 2, 6, 5

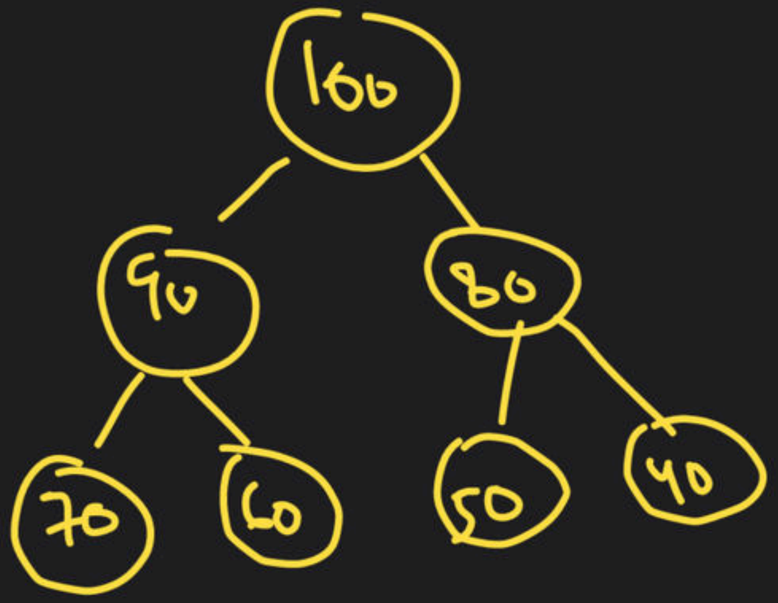
Is it rep. a max heap?



20	8	15	2	6	5
1	2	3	4	5	6

Given an array rep. a CBT 100, 90, 80, 70, 60, 50, 40

Is it rep. a max-heap?



100	90	80	70	60	50	40
1	2	3	4	5	6	7

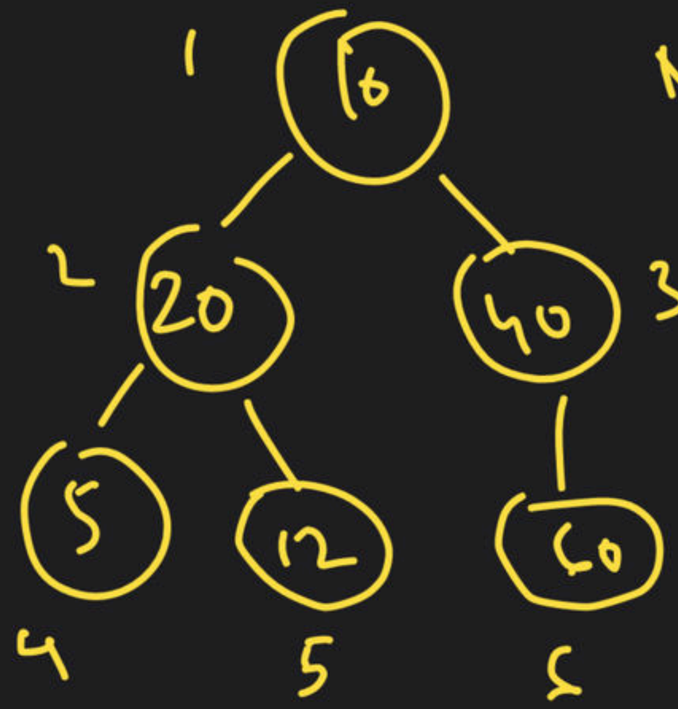
array dec. sorted ⇒ always rep a max-heap

array inc. sorted ⇒ always rep a

Given an array representing a CBT:

10, 20, 40, 5, 12, 60

Is it rep. a max-heap?



Not a max heap

10	20	40	5	12	60
1	2	3	4	5	6

Given an Array representing a CBT : 10, 20, 30, 40, 50, 60, 70

Convert it into a max-heap.

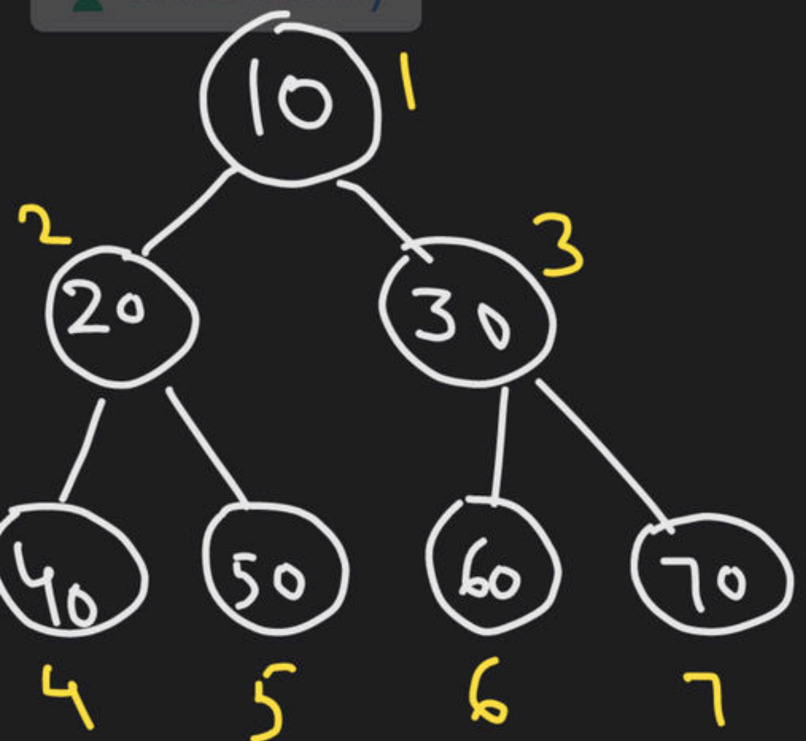
10	20	30	40	50	60	70

Sort → 70, 60, 50, 40, 30, 20, 10

O(nlogn)

Better ?

Build-heap method
↳ Heapify Algo

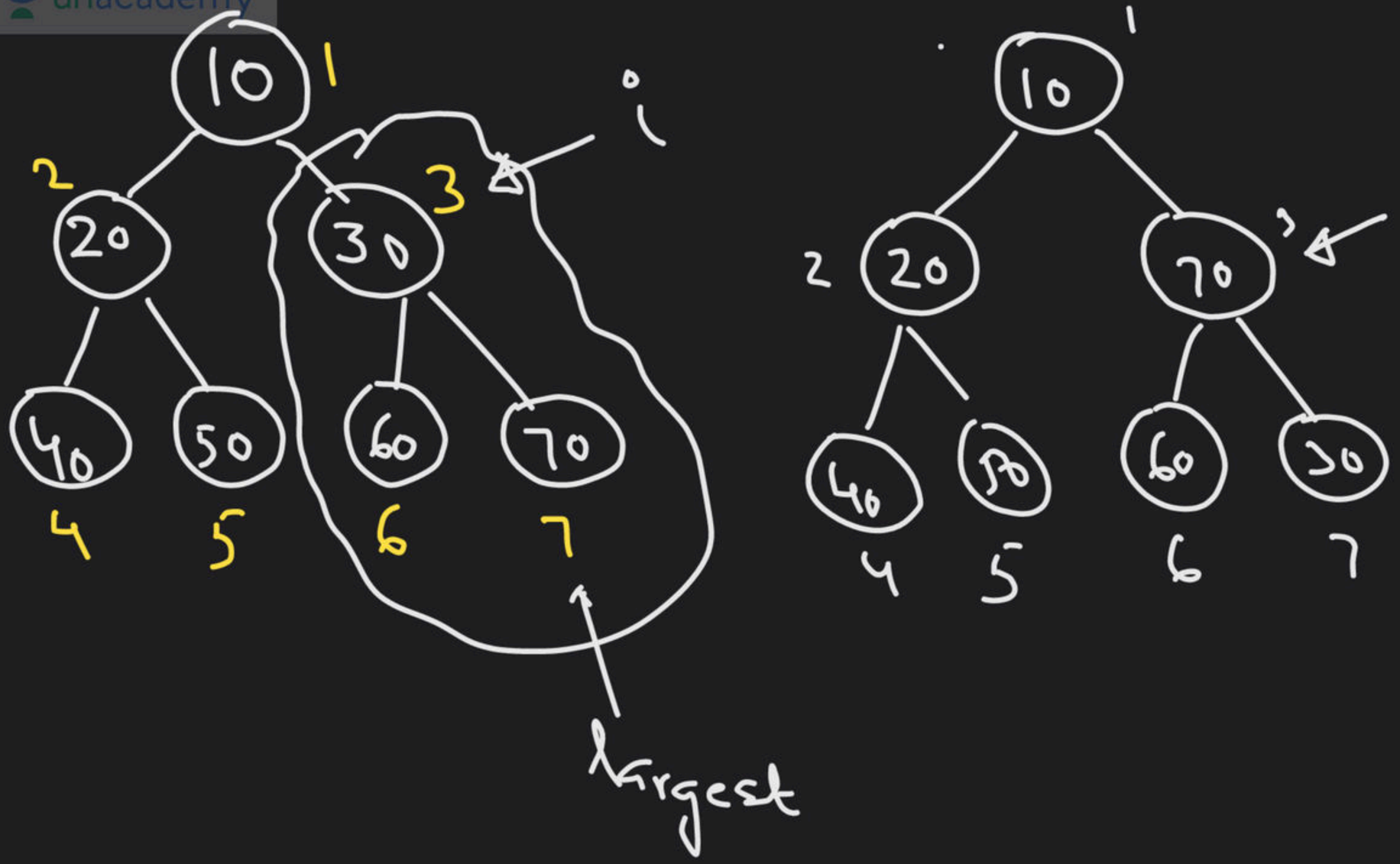


index of internal nodes

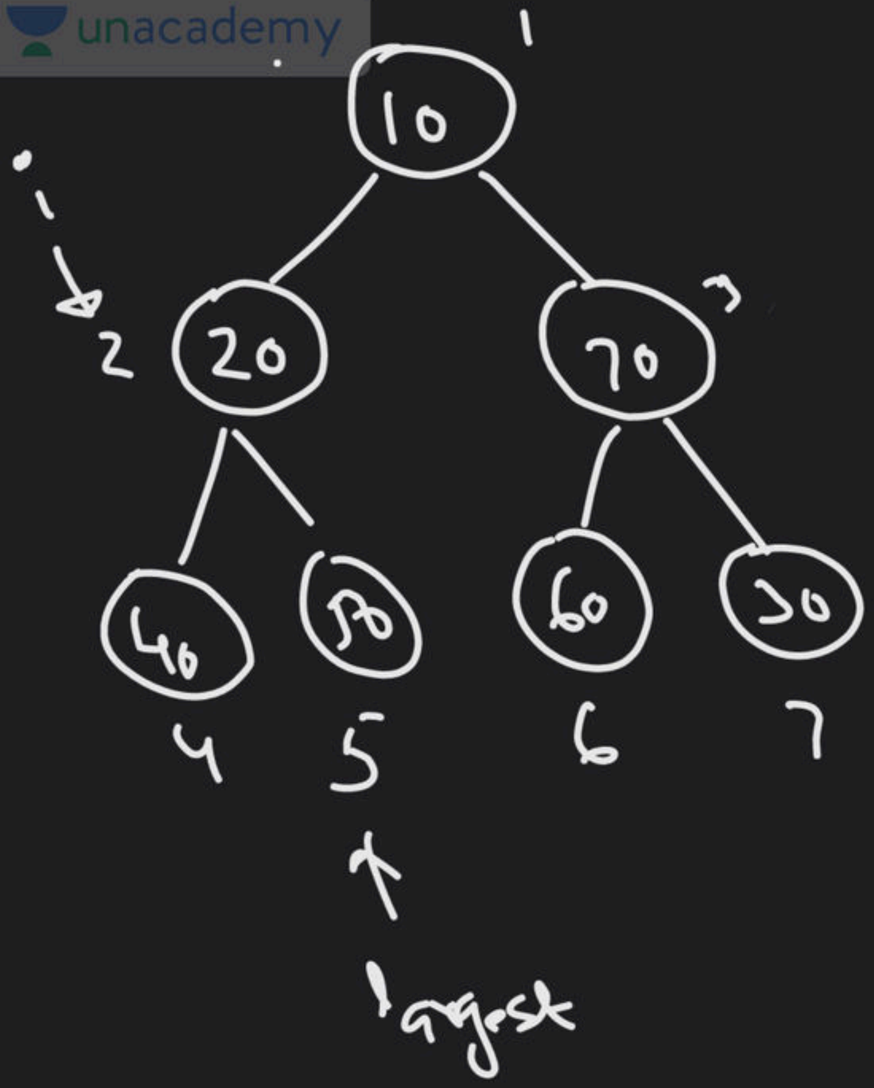
$= 1$ to $\lfloor \frac{n}{2} \rfloor \Rightarrow 1, 2, 3$

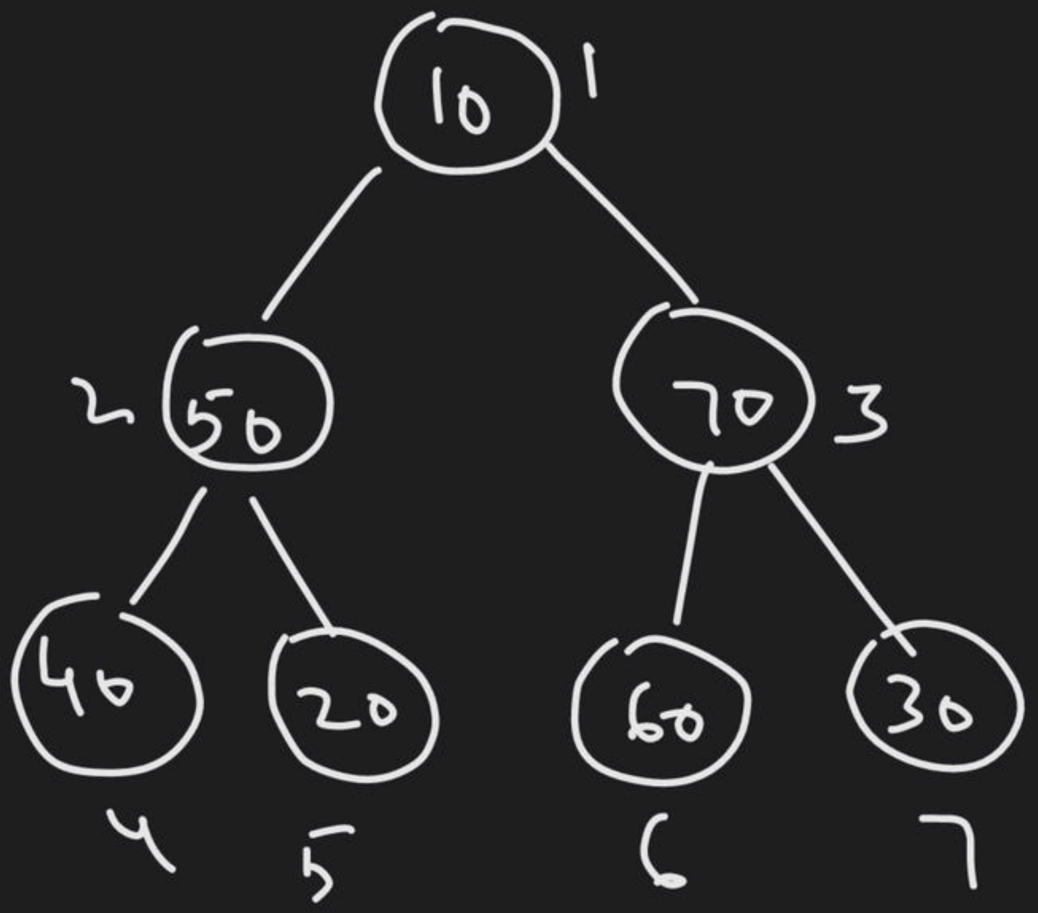
Every leaf node always satisfy max-heap property.

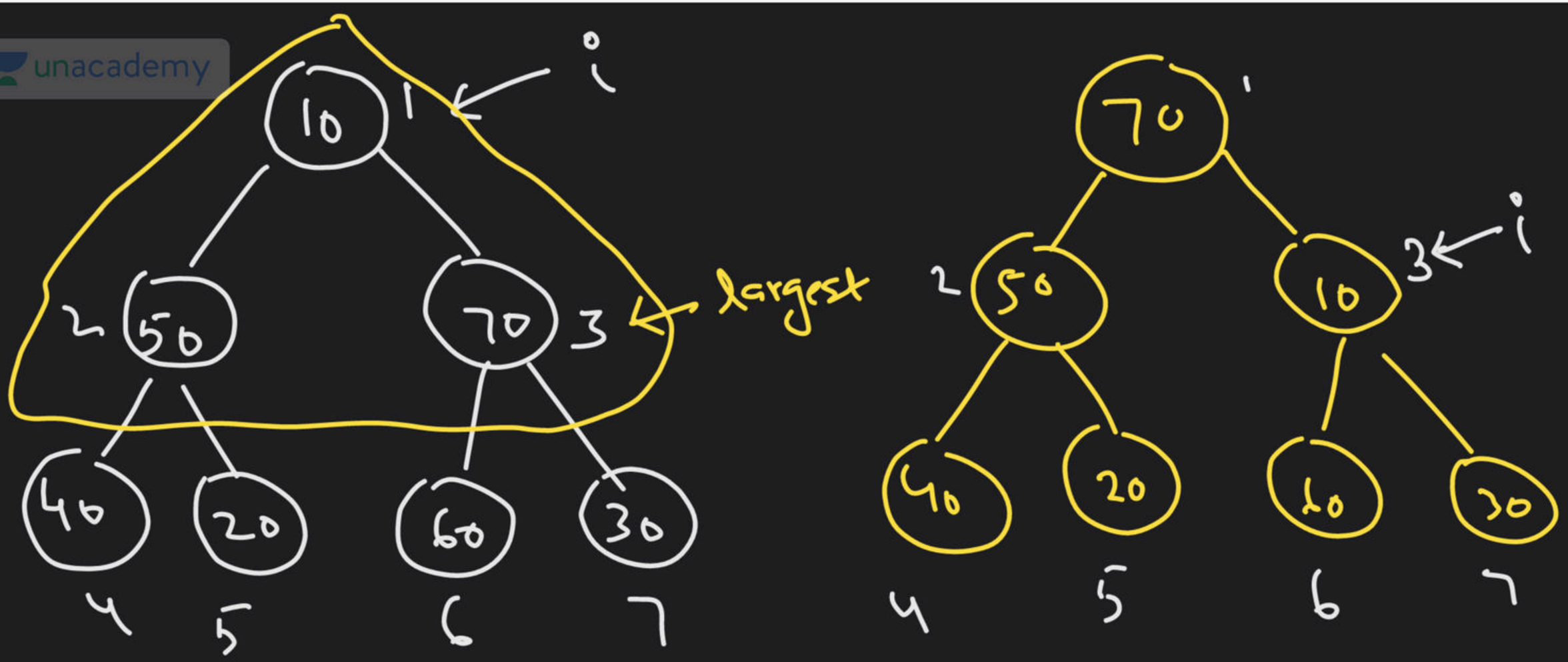
index $\rightarrow 4, 5, 6, 7$

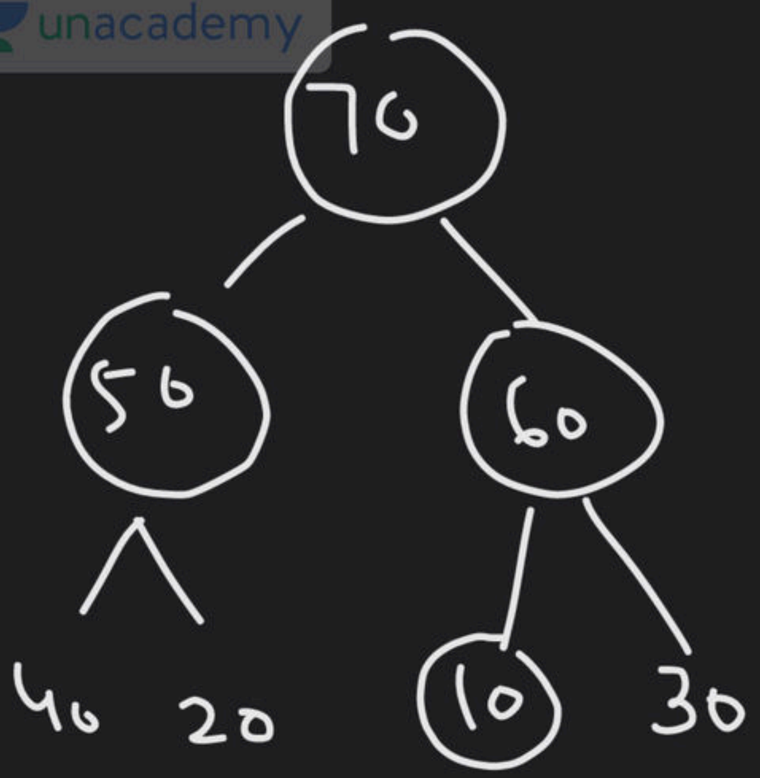


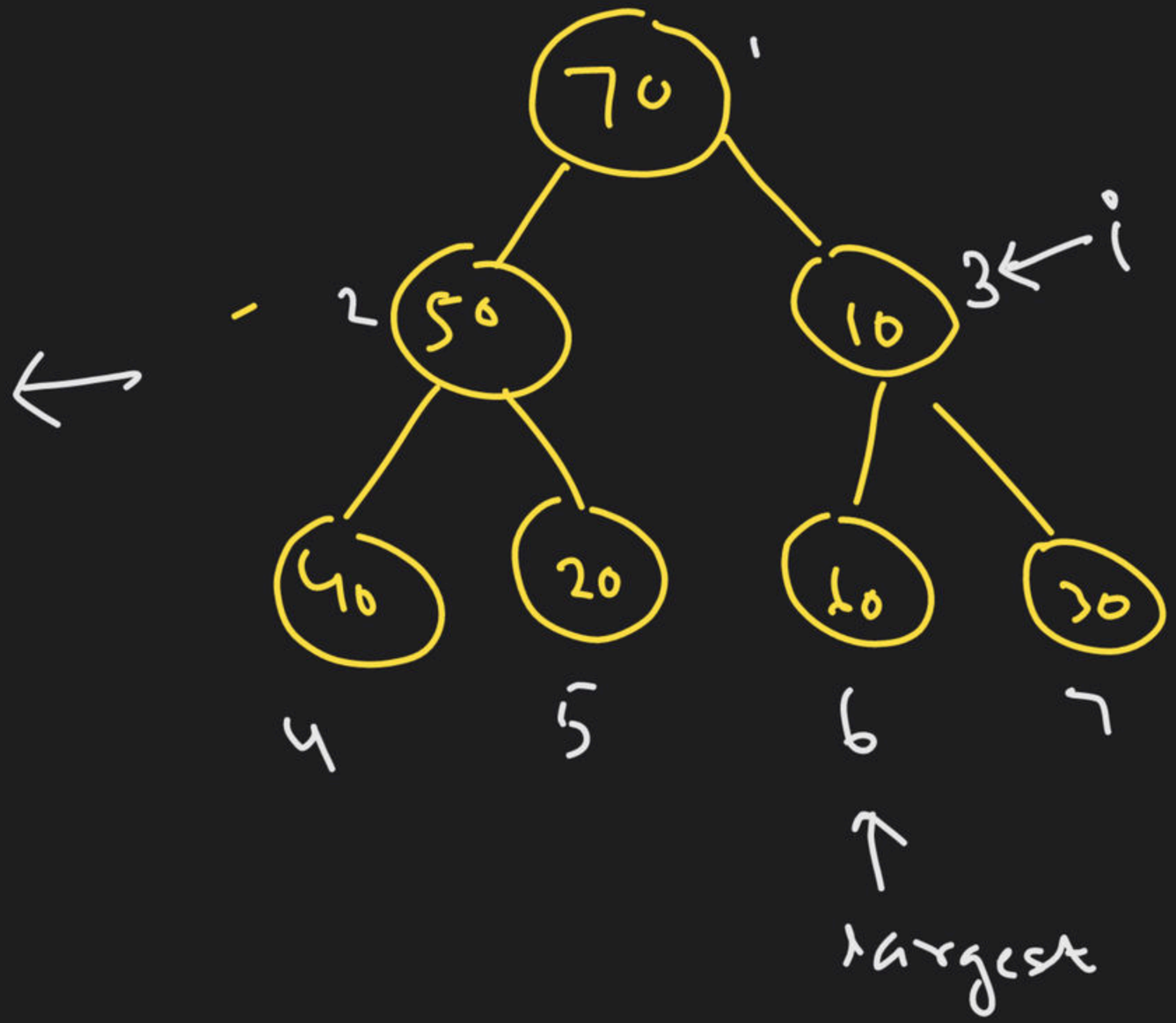
1, 2, 3



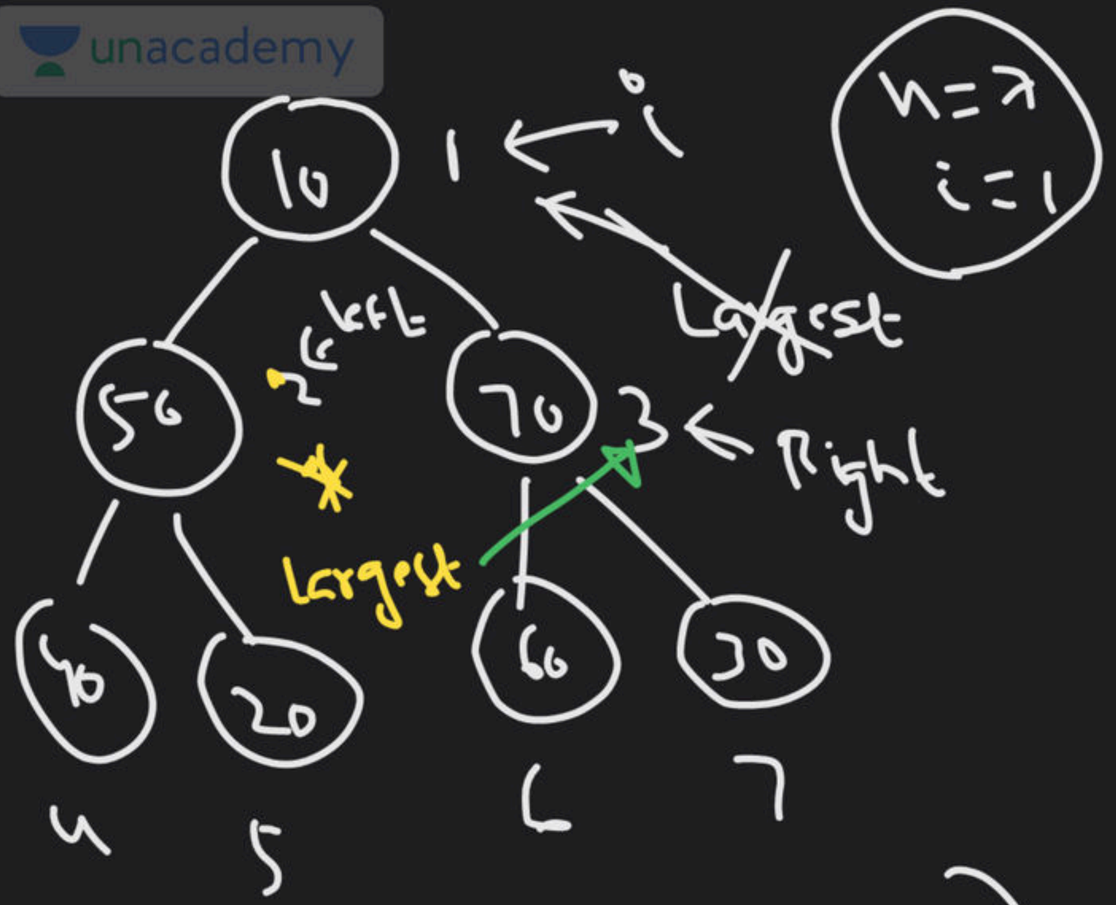



unacademy
i
1
10
2
50
70
3
largest
40
20
60
30
4
5
6
7
1
70
2
50
3
i
10
40
20
10
30
4
5
6
7





Max-heap



In Complete code

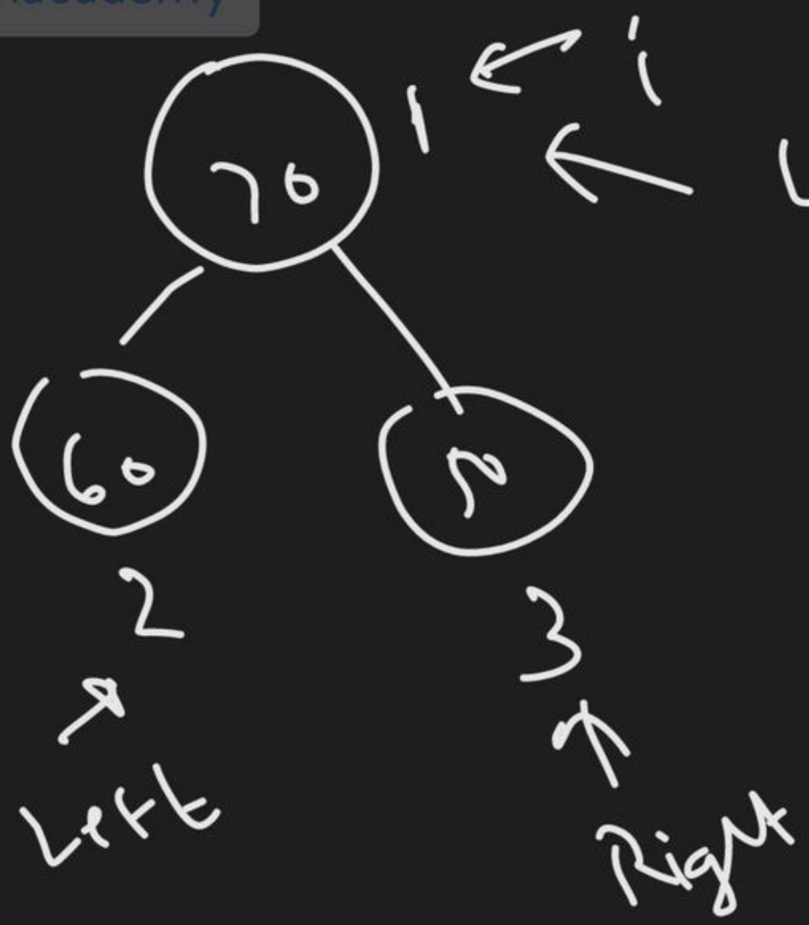
Heapify (A, i, n)

1.) Left = 2*i ; Right = 2*i+1 ;
Largest = i ;

2.) If (A[Left] > A[Largest])
Largest = Left ;

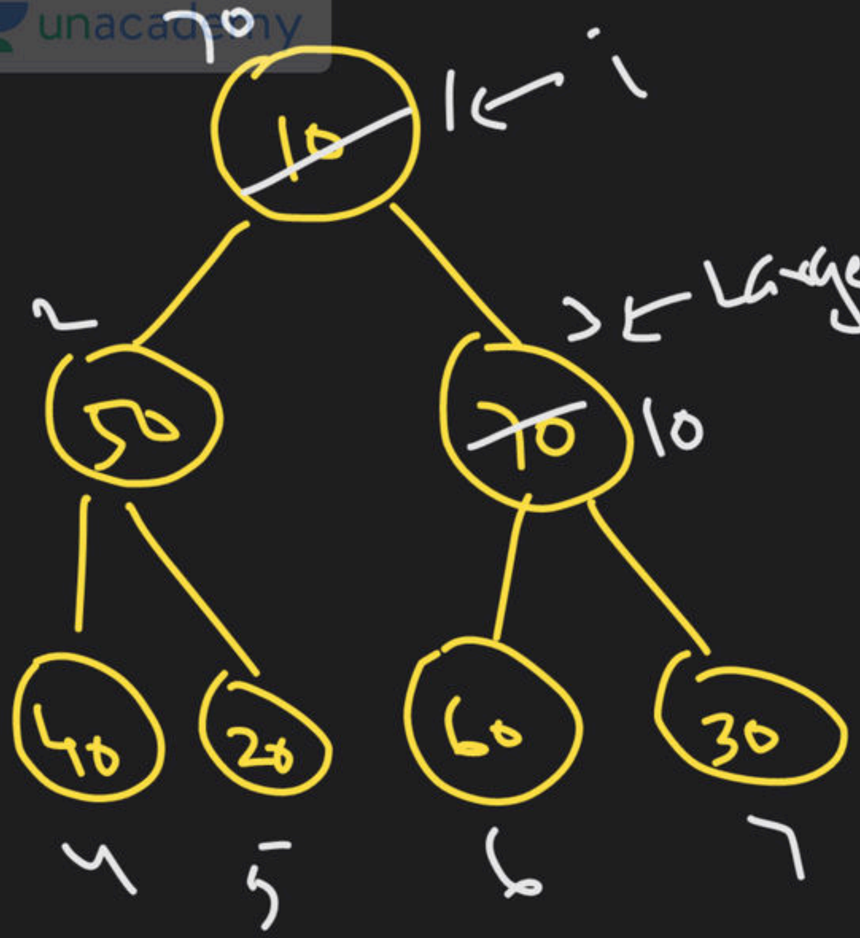
3) If (A[Right] > A[Largest])
Largest = Right ;

In this example ⇒ swap.



When to swap

```
if ( i != Largest )
swap (A[i], A[Largest])
```



Apply same algo

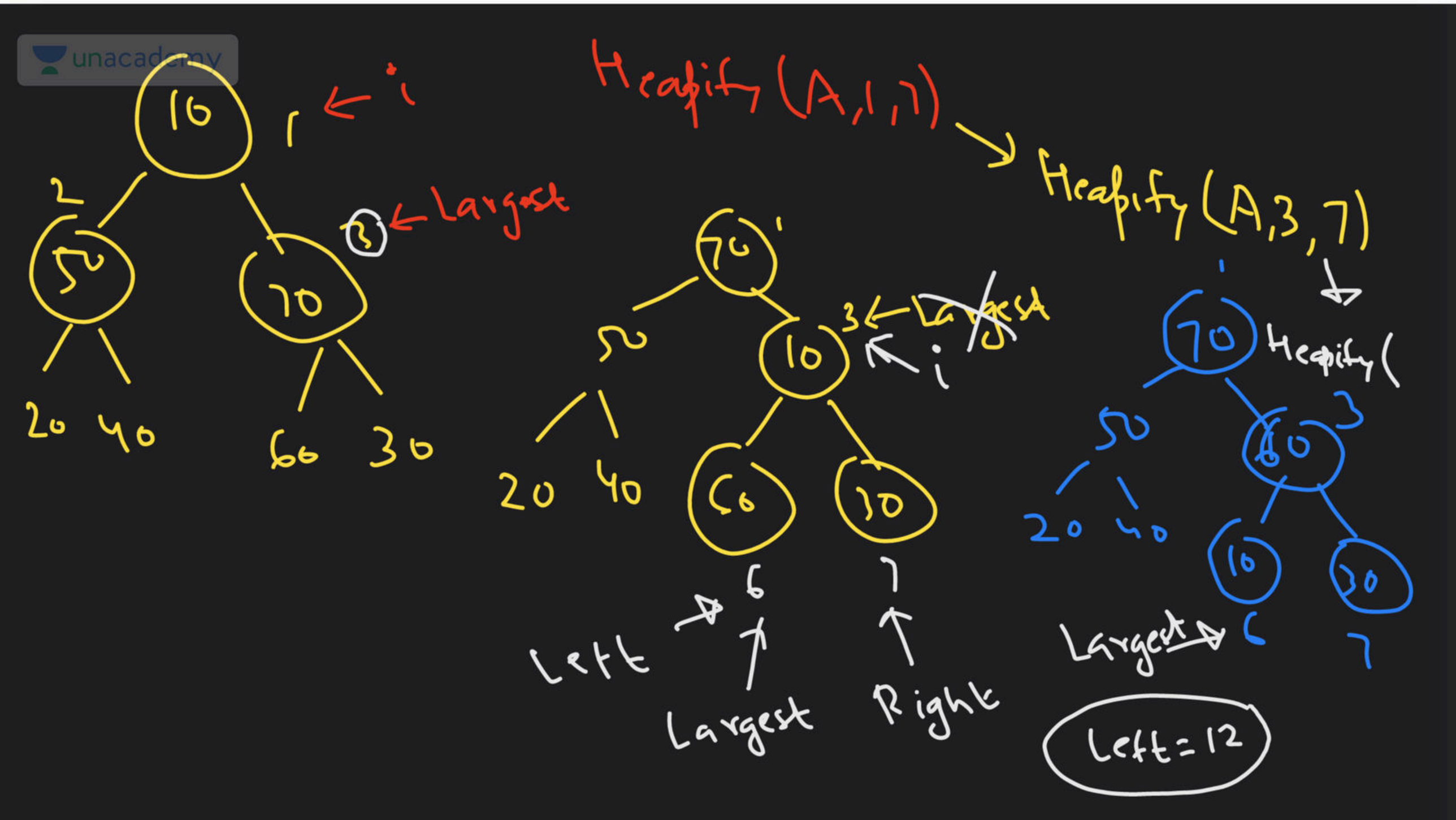
at index

Largest

unacademy

This is not the exact algo.

```
Heapify(A,i,n)
1) Left = 2*i; Right = 2*i+1;
   Largest = i;
2) if (A[Left] > A[Largest])
       Largest = Left;
3) if (A[Right] > A[Largest])
       Largest = Right;
   if (i != Largest) {
       swap(A[i], A[Largest]);
       Heapify(A, Largest, n);
   }
```


unacademy
10
i
2
Largest
3
50
70
20
40
60
30
Heapify (A,1,7)
70
50
10
3
Largest
i
20
40
60
10
Left
6
Largest
7
Right
Heapify (A,3,7)
70
Heapify (
50
60
20
40
10
30
Largest
Left=12

Heapify (A, i, n)

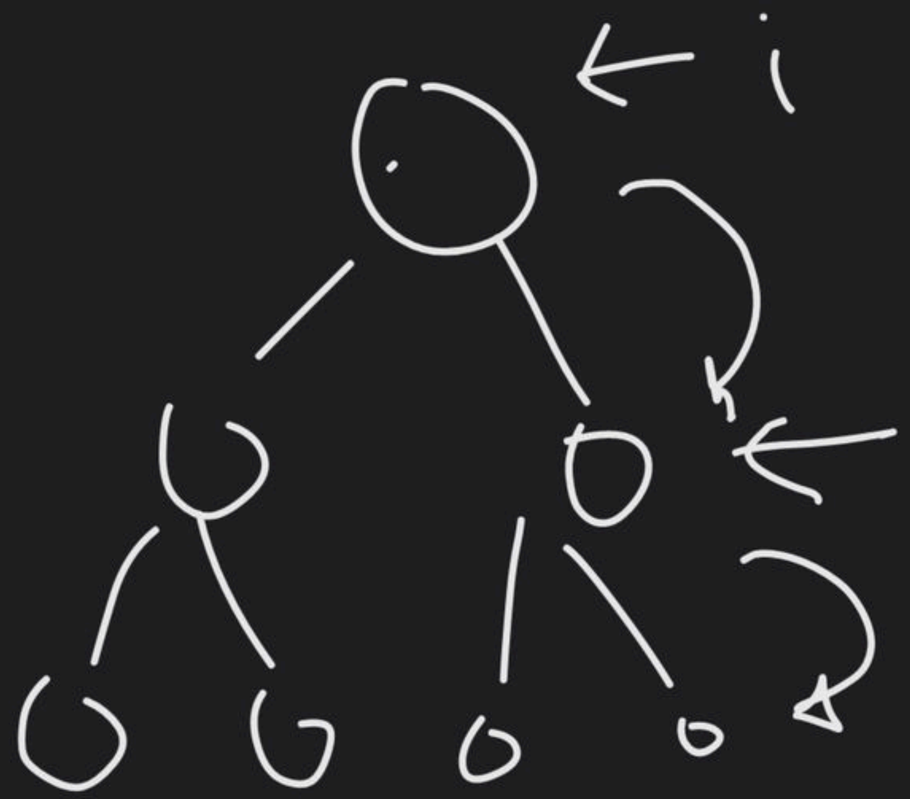
1) Left = 2*i ; Right = 2*i + 1 ; Largest = i ;

2) if (Left <= n && A[Left] > A[Largest])
   Largest = Left ;

3) if (Right <= n && A[Right] > A[Largest])
   Largest = Right ;

4) if (i != Largest) {
   swap(A[i], A[Largest]);
   Heapify(A, Largest, n);
   }

Analysis



$h \Rightarrow O(\log_2 n)$

$O(n)$

internal nodes $\Rightarrow$ Heapify

unacademy

Max-heap

Build-heap ⇒ O(n)

100	90	95	80	85	70		84
1	2	3	4	5	6		7

① Find Max ⇒ return A[1]

O(1)

② Find Min ⇒ O(n)

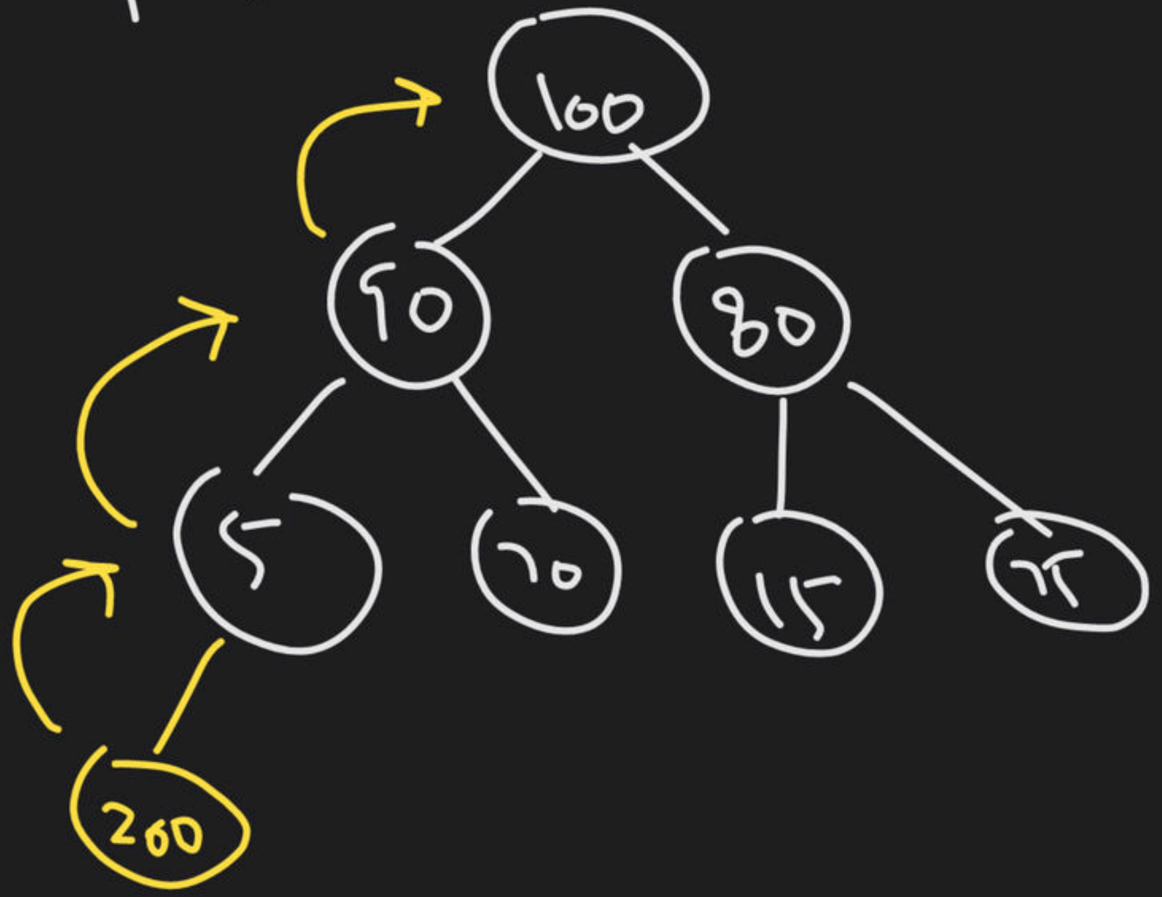
leafs = $\lceil \frac{n}{2} \rceil = \lceil \frac{7}{2} \rceil = \lceil 3.5 \rceil = 4$

$10 \Rightarrow 9$

$\left\lceil \frac{n}{2} \right\rceil \Rightarrow O(n)$

(3) Search $\Rightarrow O(n)$

Insert a key



$O(\log_2 n)$

$Par(i)$

$= \lfloor \frac{i}{2} \rfloor$

Max-heap

1) Find Max - $O(1)$
2) Find Min - $O(n)$
3) Insert key - $O(\log_2 n)$
4) Search - $O(n)$
5) Extract Max : $O(\log_2 n)$

Min-heap

1) Find Min - $O(1)$
2) Find Max - $O(n)$
3) Insert → $O(\log n)$
4) Search - $O(n)$
5) Extract-min → $O(\log_2 n)$

unacademy

THANK YOU!

Here's to a cracking journey ahead!