one of the most imp DS ⟵ Priority Queue ⟶ whatever is the order of insertion, you will access the element with highest priority first.

Basic queue ⟶ FIFO

⟶ will be accessed first

el1 ⟶ el2 ⟶ lower the timestamp higher is the priority

t1 < t2

qx+by -c

Car1 ⟶ Car2 ⟶ higher the price higher is the priority

p1 < p2

Priority Queue → | — push
— get → will return highest priority el.
— pop → remove the highest pri. element, & reorder the queue.

How can we design it??

1) Arrays → O(1) → push
O(n) → pop
O(n) → get

2) LL → O(1) → push
O(n) → pop
O(n) → get

3) BBST → Insert → O(log n)
pop → O(log n)
get → O(log n)

if let's say implemented via AVL tree, then issue is rotations

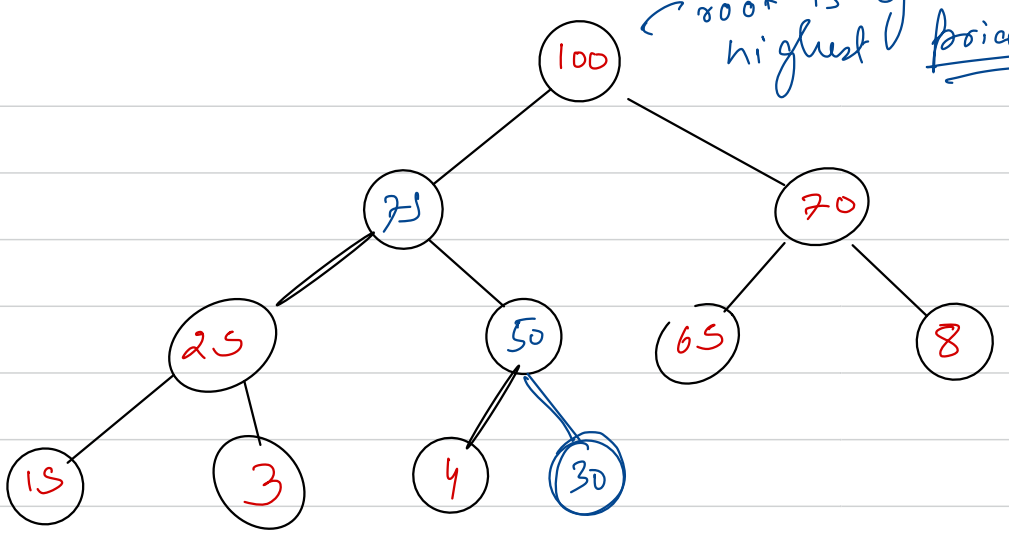
4) Heaps (Binary) →

Binary Heaps

- push → O(log n)
- pop → O(log n)
- get → O(1)

→ It is a binary tree.
→ It is always a complete binary tree (CBT)
→ In a binary heap, priority of parent is always greater than priority of children.

NOTE → It is a hierarchy based DS which can be implemented linearly.



root is highest priority

priority → max element

max heap

insert(75) → upheapify → $O(h)$ → $O(\log n)$

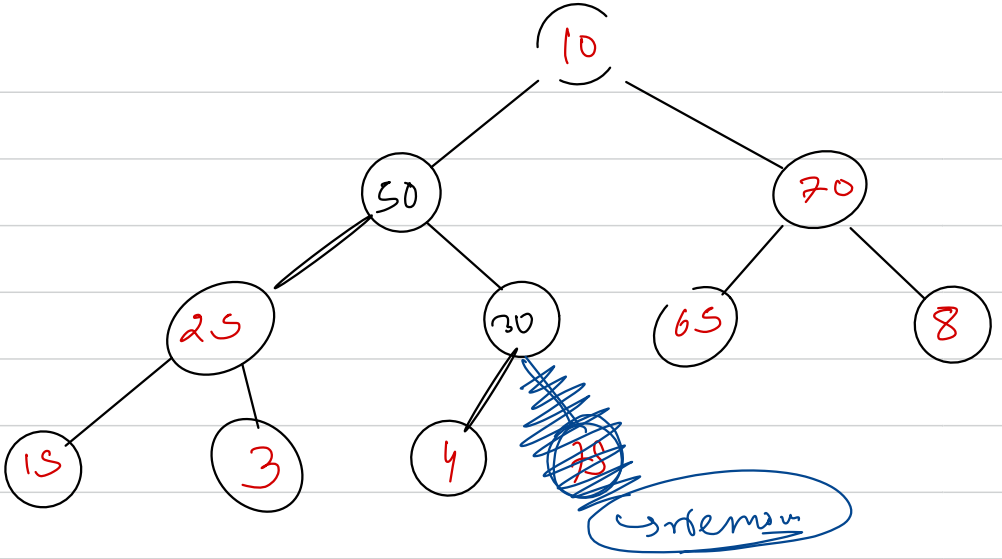
total nodes

$2^0 + 2^1 + 2^2 \cdots\cdots 2^{h-1} = n$

$2^h - 1 = n$

$h = \log_2(n+1)$

How to Remove the highest-priority element



deleteHeapify → O(log n)

$T(n) = T(n/2) + 2$

$T(n/2) = T(n/4) + 2$

$T(n/4) = T(n/8) + 2$

$T(n/2^k) = T(1) + 2$

$T(n) = T(1) + 2k$

k

$\hookrightarrow O(\log n)$

$T(n) = T(n/3) + 3$

$T(n/3) = T(n/9) + 3$

$T(n/9) = T(n/27) + 3$

$T(n/3^k) = T(1) + 3$

$T(n) = T(1) + 3k$

k'

$\hookrightarrow O(\log n)$

$n \to \frac{n}{2} \to \frac{n}{4} \to \frac{n}{8} \text{ ------ } \frac{n}{2^k}$ $\qquad \frac{n}{2^k} = 1$

$k = \log_2 n$

$k' = \log_3 n$

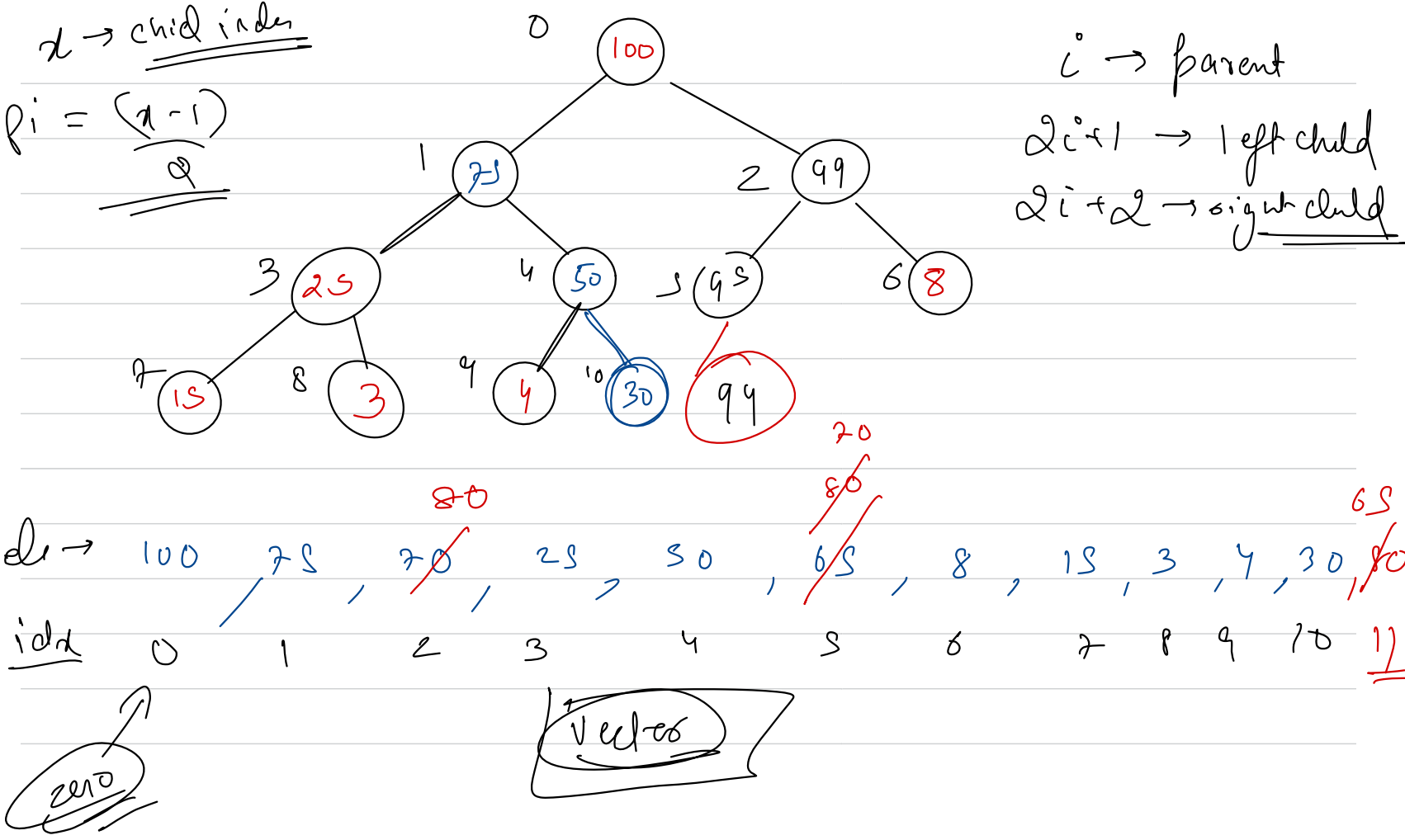
$2\log_2 n$

Comparisons

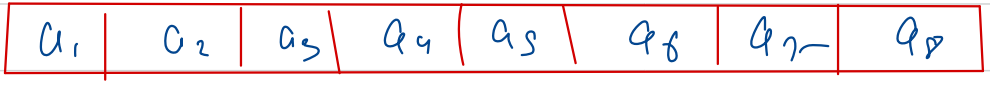
$3\log_3 n$

Comparisons

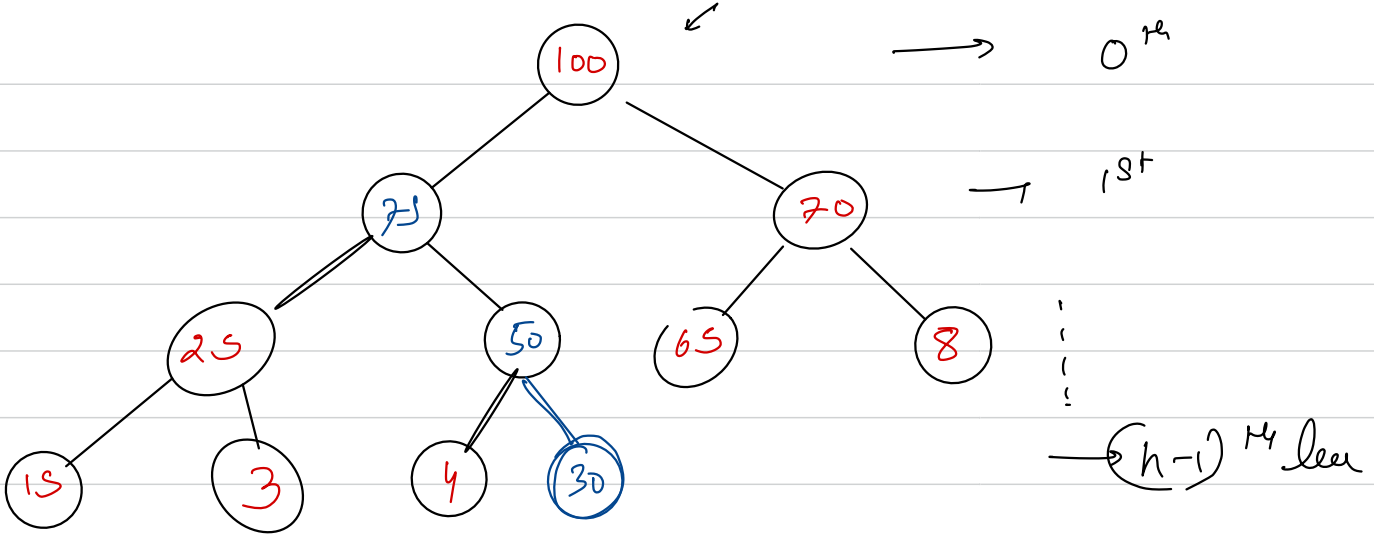
$$2\log_2 n < \frac{3}{\log_2 3} \times \log_2 n$$


x → child index
i = (x-1)/2
i → parent
2i+1 → left child
2i+2 → right child
0
100
1
75
2
99
3
25
4
50
5
65
6
8
7
15
8
3
9
4
10
30
99
20
80
80
65
arr →
100
75
20
25
50
65
8
15
3
4
30
80
idx
0
1
2
3
4
5
6
7
8
9
10
11
zero
Vector

Q. Given an array of integers convert this array into a binary max-heap in place.



Starting from index 1, do upheapify.



$$S = \underbrace{2^0 \times 0}_{x} + 2^1 \times 1 + 2^2 \times 2 \cdots\cdots\cdots 2^{h-2} \times (h-2) + 2^{h-1} \times (h-1)$$

$$S = 2^1 \times 1 + 2^2 \times 2 + 2^3 \times 3 \cdots\cdots 2^{h-2} \times (h-2) + 2^{h-1}(h-1)$$

$$2S = 2^2 \times 1 + 2^3 \times 2 + 2^4 \times 3 \cdots\cdots 2^{h-1}(h-2) + 2^h(h-1)$$

$$2S - S = 2^1 \times 1 + 2^2(-1) + 2^3(-1) \cdots\cdots 2^{h-1}(-1) + 2^h(h-1)$$

$$S = 2 - (2^2 + 2^3 + 2^4 \cdots\cdots 2^{h-1}) + 2^h(h-1)$$

$$S = 2 - 2^2 \times (2^{h-2} - 1) + 2^h h - 2$$

$$S = -2^h + 2^2 + 2^h h$$

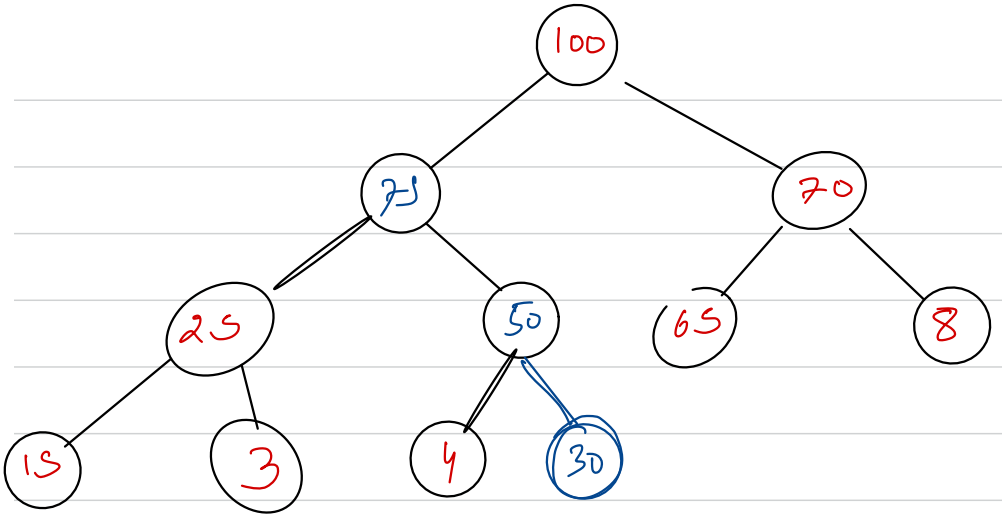
$$S = -n + 4 + n\log n$$

$\longrightarrow O(n\log n)$

$h \to \log_2 n$

$2^h \to 2^{\log_2 n} = n$

greedy



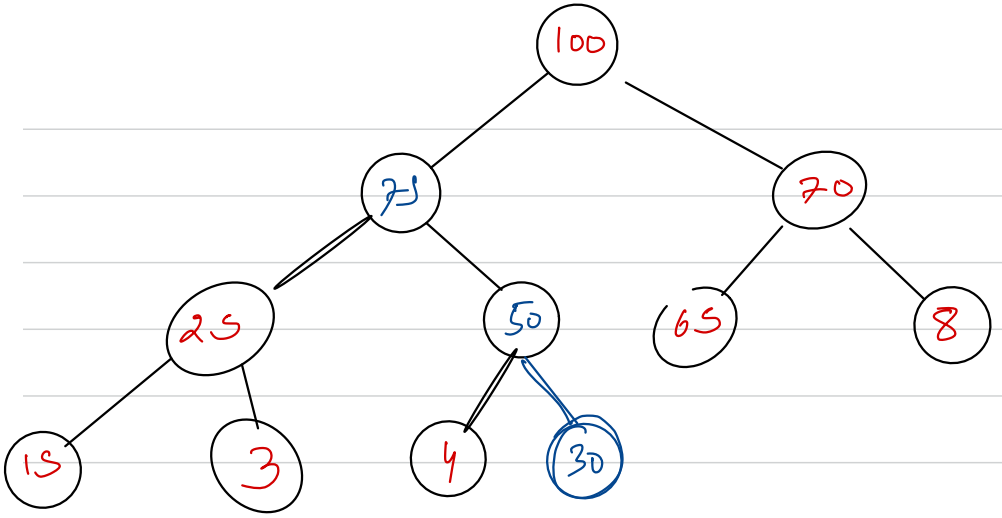
#last level nodes

$$\underbrace{2^0 + 2^1 + 2^2 + 2^3 - \cdots - 2^{h-2}}_{} + 2^{h-1} = n$$

$$(2^{h-1} - 1) + 2^{h-1} = n \Rightarrow 2 \times 2^{h-1} = n+1$$

$$2^{h-1} = \frac{(n+1)}{2}$$

nodes at last level



$$S = 2^{h-1} \times 0 + 2^{h-2} \times 1 + 2^{h-3} \times 2 - - - - - - 2^{1}(h-2) + 2^{0}(h-1)$$

$$S = 2^{h-1} \times 0 + 2^{h-2} \times 1 + 2^{h-3} \times 2 \cdots\cdots 2^1 (h-2) + 2^0 (h-1)$$

$$S = 2^{h-2} \times 1 + 2^{h-3} \times 2 \cdots\cdots 2^1 (h-2) + 2^0 (h-1)$$

$$2S = 2^{h-1} \times 1 + 2^{h-2} \times 2 + 2^{h-3} \times 3 \cdots 2^2 (h-2) + 2^1 (h-1)$$

$$S = 2^{h-1} + \underbrace{2^{h-2} + 2^{h-3} \cdots\cdots 2^2 + 2^1}_{GP} + 2^0 (h-1)$$

$$S = 2^{h-1} + 2^2 \left(2^{h-2} - 1\right) + (h-1)$$

$$S = 2^{h-1} + 2^{h-1} - 2 + h - 1$$

$$S = 2^h + h - 3$$

$$S = n + \log n - 3 \longrightarrow O(n)$$

$h \to \log n$

$2^h = n$