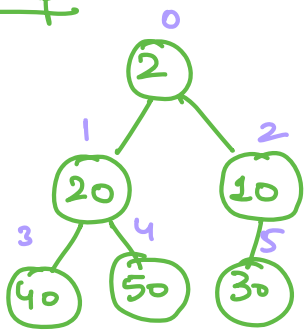
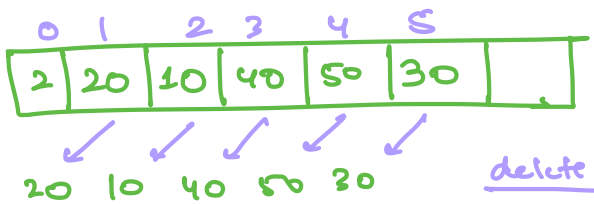
Removal:

Heap: Highest Priority

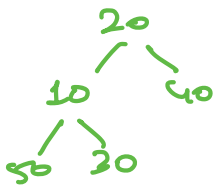
min heap



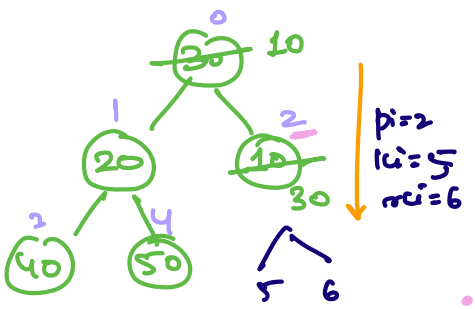


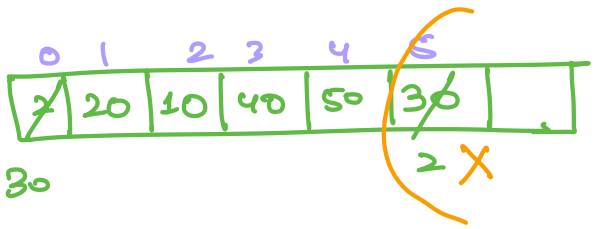
delete 0 : heap priority violate

O(n) X



mini = 0 1 2





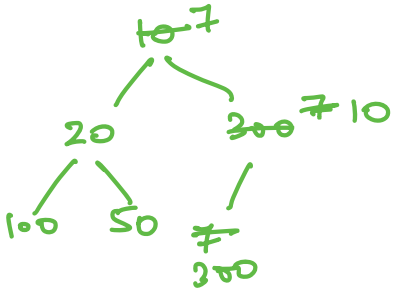
- 0th index & last index swap
- delete last value
- downheapify

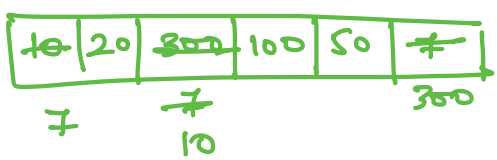
find index & minimum value

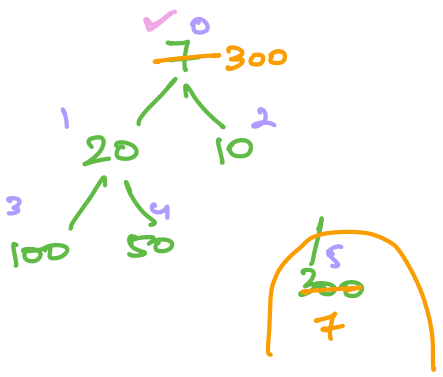
mini = 2

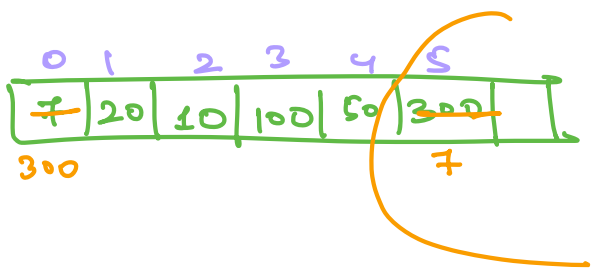
swap (0index, 2index)

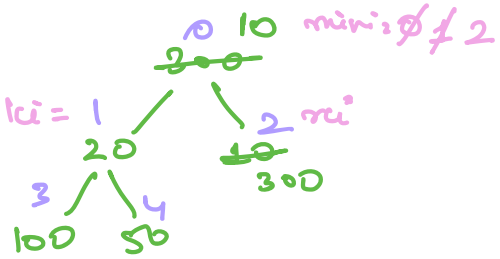
mini = pi ri

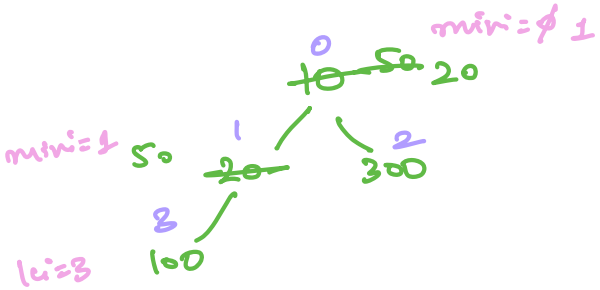


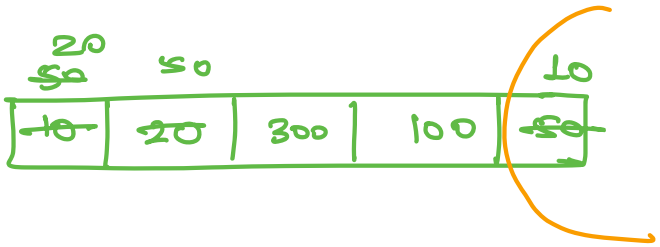


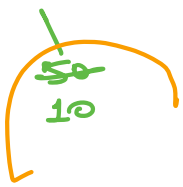






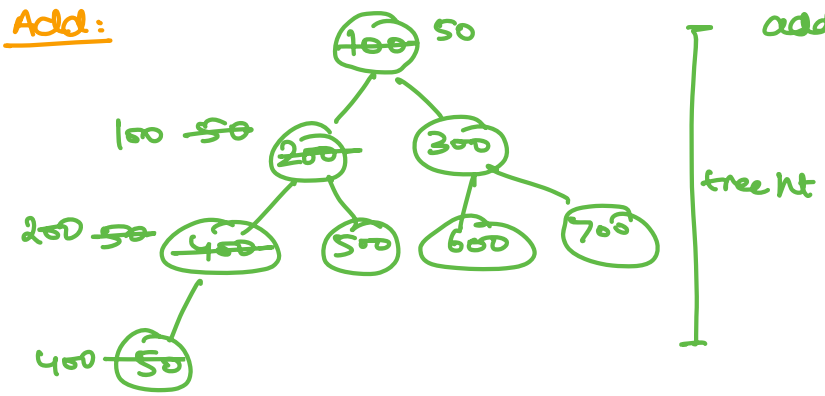






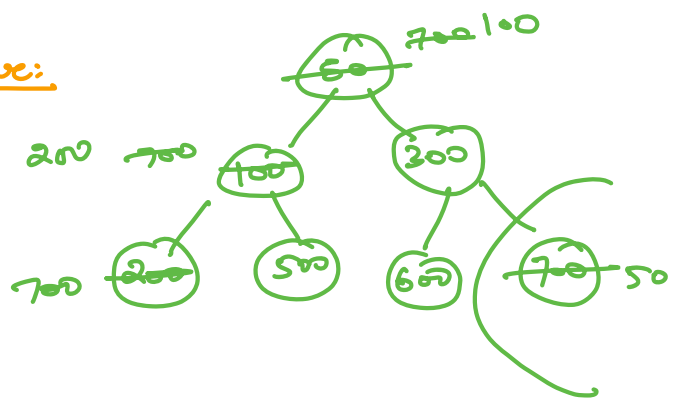
Time Complexity:

Add:



O(h)

Remove:



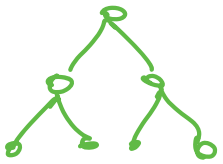
swap: O(1)
remove: O(1)
downheapify: O(h)

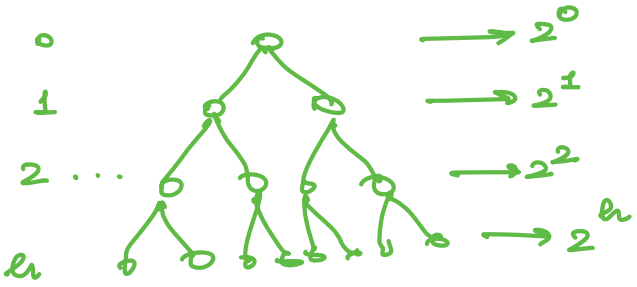
O(h)

Get: data[0] O(1)

Height ? n terms

- Maximum no. of elements required to have ht h in CBT ?





$$n = 2^0 + 2^1 + 2^2 + \cdots 2^h$$

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

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

$$n+1 = 2^{h+1}$$

$$2^{h+1} = n+1$$

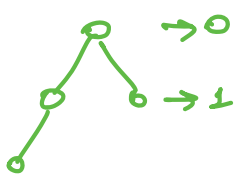
$$h+1 = \log_2(n+1)$$

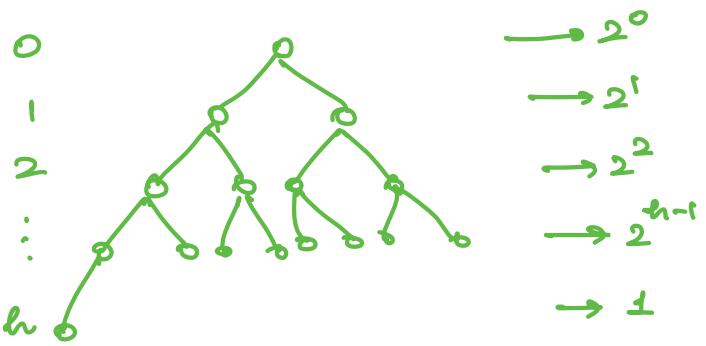
$$h = \log_2(n+1) - 1$$

$$\boxed{h = O(\log_2 n)}$$

- minimum no. of elements required to have ht h in CBT ?

h = 2





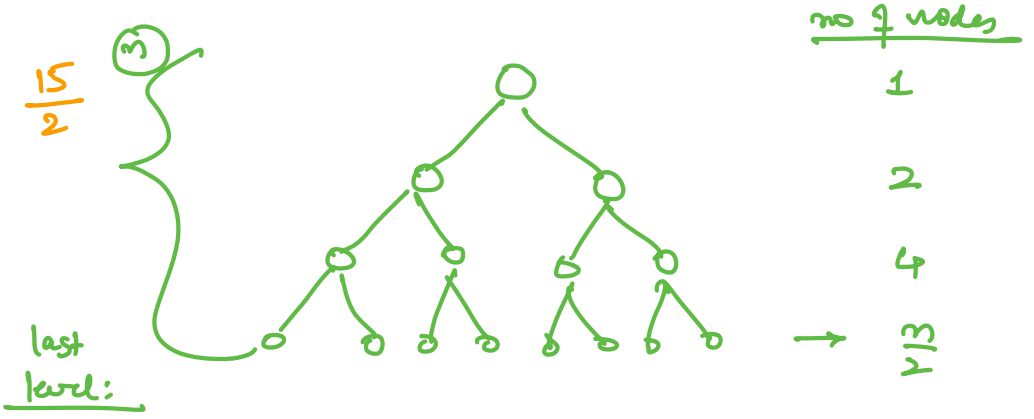
$$n = 2^0 + 2^1 + 2^2 + \cdots 2^{h-1} + 1$$

$$n = 2^h - 1 + 1$$

$$\boxed{h = \log_2 n}$$

1 element { add / remove } $O(h) = O(\log_2 n)$

n elements : $O(n \log_2 n)$



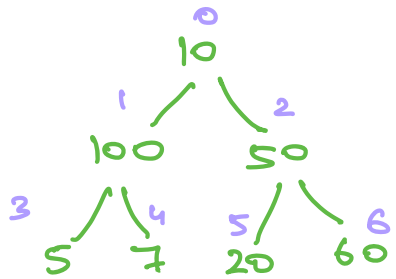
add(10)
(20)
(30)
(40)
⋮
} add n elements. O(nlogn)

$\frac{n}{2}(\log_2 n) + \cdots + 4(2) + 2(1) + 1(0)$

$O(n \log n)$

Step 2: All n elements are given at once

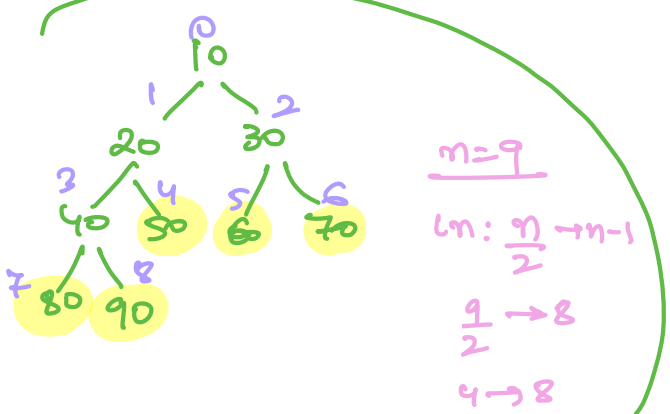
array: [10 100 50 5 7 20 60] (0 1 2 3 4 5 6)

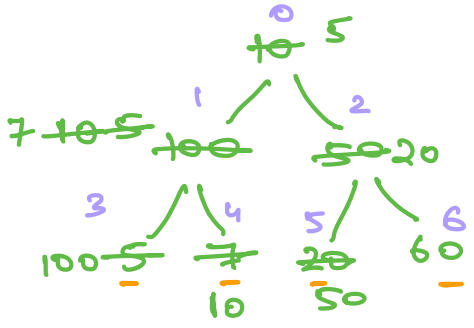


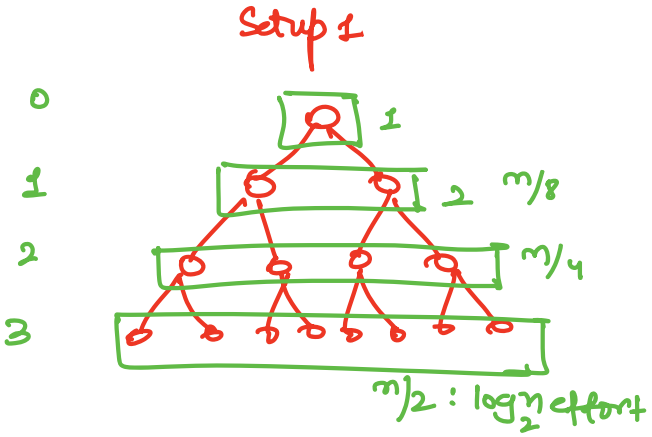
Imp point: n elements (n = 7)
leaf nodes CBT: $\frac{n}{2} \rightarrow n-1$

$\frac{7}{2} \rightarrow 6$

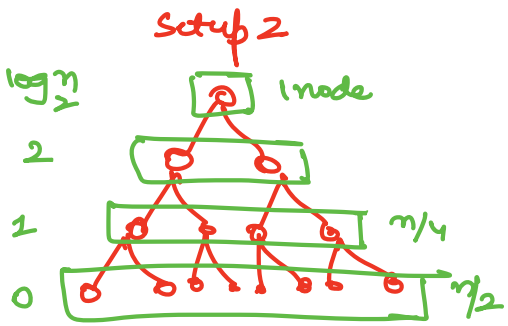
$3 \rightarrow 6$



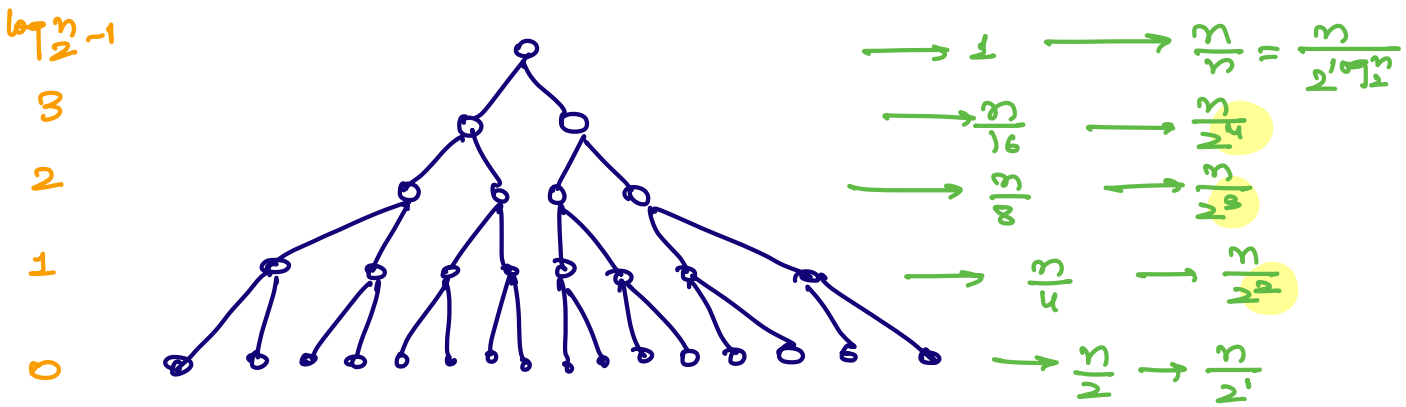




TC: n log n



TC?



$$S = \frac{n}{2}\times 0 + \frac{n}{2^2}\times 1 + \frac{n}{2^3}\times 2 + \frac{n}{2^4}\times 3 \cdots \frac{n}{2^{\log_2 n-1}}(\log_2 n - 2) + \frac{n}{2^{\log_2 n}}(\log_2 n - 1)$$

$$2S = n\times 0 + \frac{n}{2}\times 1 + \frac{n}{2^2}\times 2 + \frac{n}{2^3}\times 3 + \frac{n}{2^{\log_2 n-1}}(\log_2 n - 1)$$

$$S = \frac{n}{2}(1-0) + \frac{n}{2^2}(2-1) + \frac{n}{2^3}(3-2) + \cdots \frac{n}{2^{\log_2 n-1}}(-1+2) - \frac{n}{2^{\log_2 n}}(\log_2 n - 1)$$

$$S = \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots \frac{n}{2^{\log_2 n-1}} - \frac{n}{2^{\log_2 n}}(\log_2 n - 1)$$

$$= n\left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \frac{1}{2^{\log_2 n - 1}}\right)$$

$$= n\left(\frac{\frac{1}{2}\left(1 - \left(\frac{1}{2}\right)^{\log_2 n - 1}\right)}{\left(1 - \frac{1}{2}\right)}\right) = n\left(1 - \left(\frac{1}{2}\right)^{\log_2 n - 1}\right)$$

$$1 - \frac{\left(\frac{1}{2}\right)^{\log_2 n}}{\left(\frac{1}{2}\right)}$$

$$1 - \frac{2^{-\log_2 n}}{\left(\frac{1}{2}\right)}$$

$$1 - \frac{n^{-1}}{\frac{1}{2}} = 1 - \frac{1}{n}\left(\frac{1}{\frac{1}{2}}\right)$$

$$= 1 - \frac{2}{n} = \frac{n-2}{n} \simeq 1$$

$$S = n(1) - \frac{n}{2^{\log_2 n}}(\log_2 n - 1)$$

$$S = n - \frac{n}{n}(\log_2 n - 1) = n - \log_2 n - 1$$

$$O(n)$$

Step 2: All n elements are given at once: heap : $O(n)$

Heap Sort:

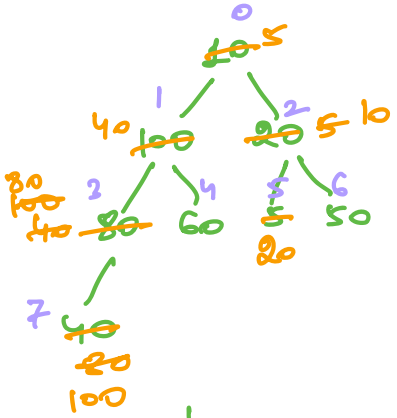
BS, SS, IS, MS, QS

↓

DEC

Descending:

[10 100 20 80 60 5 50 40]



- Heap Convert:

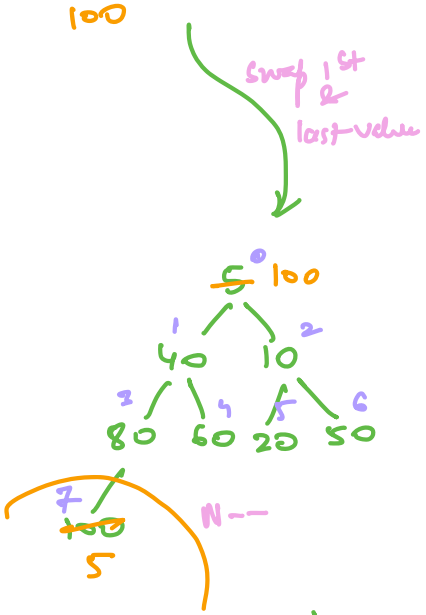
n=8

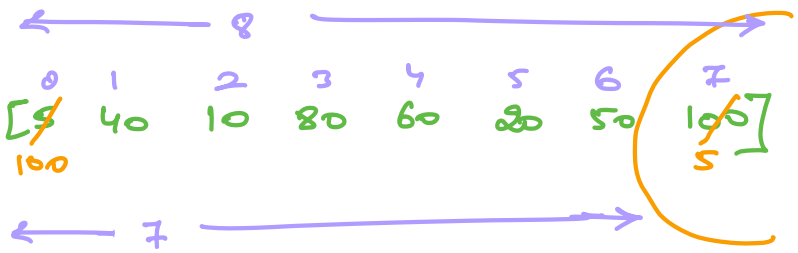
$\frac{n}{2} \rightarrow n-1$: LN 4 → 7 : LN

$\frac{n}{2}-1 \rightarrow 0$

2 → 0

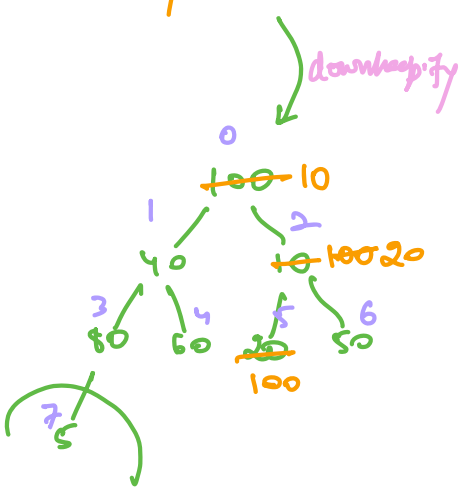
2, 2, 1, 0

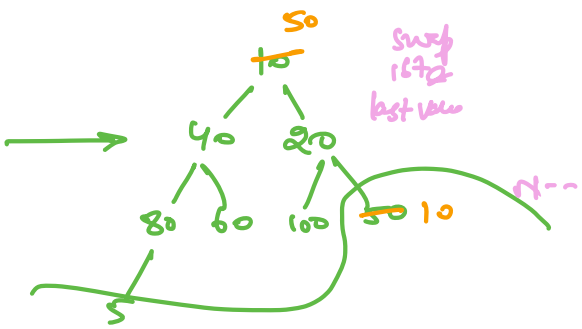


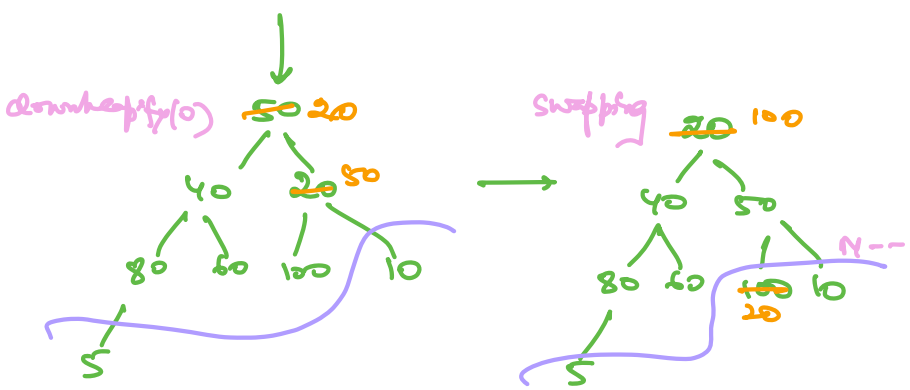


- swap 1st & last = since last index (N-1)
- N--
- downheapify(0)

- Vec.pop_back()
- N--







N-1 times

[Create heap } n

[N-1 → 1 Runs for N-1 times

  swap: O(1)
  down (0): O(log n)] } O(n log n)

Q: k largest elements

k = 3

10 ✓ 100 ✓ 20 ✓ 15 ↓ 3 ↓ 35 ↓ 200 ↓ 60 ↓ 85 ↓ 9 ↓

Simple strategy: Sort all Elmets; last k pick.

O(n log n) + k

O(n log n)