K Largest Element

Sunday, 27 June 2021 5:41 PM

/ Quick Select (worst case → $O(n^2)$)

k^{th} largest

$k = 4$.

arr → { 10, 9, 20, 7, 21, 26, 15, 19, 18 }

① Brute force → Fill all the elements in priority Queue (max).
remove k Elements.

Time: $O(n \log n)$

complexity for single insertion → $\log n$.

n insertion → $n \log n$.

k removals → $n \log n$.

Total = $2n \log n \Rightarrow O(n \log n)$

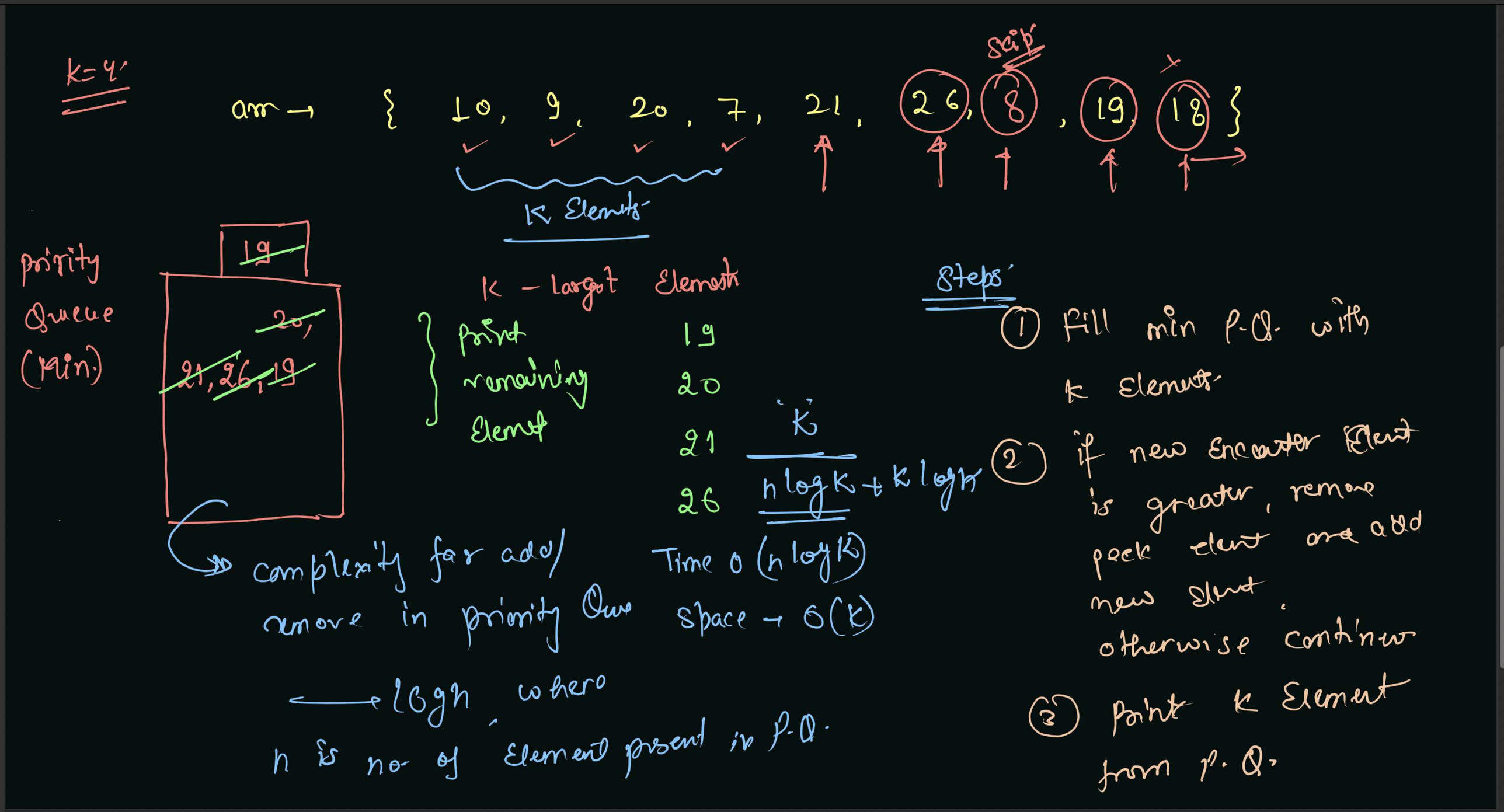
First remove → 26

Second remove → 21

⋮

k^{th} remove → k^{th} largest Element

optimise from $n \log n$ to $n \log k$


k=4
arr → { 10, 9, 20, 7, 21, 26, 8, 19, 18 }
skip
K Elements
priority Queue (Min.)
19
20,
21, 26, 19
K - Largest Elements
print remaining Element
19
20
21
26
Steps
(1) Fill min P.Q. with K Elements
(2) if new Encounter Element is greater, remove peek elem and add new elem. otherwise continue
(3) Print K Element from P.Q.
K
n log K + K log K
Time O(n log K)
Space → O(K)
complexity for add/ remove in priority Que
→ log n, where
n is no of Element present in P.Q.

Sort K-Sorted Array

Friday, 2 July 2021 7:18 PM

$k=3$. Elements are shifted from $-k$ to k distance from its actual position.

Input → 20 40 30 10 80 70 50 60 ↓

min-

10 20 20 40 50 60 70 80 →

Complexity → $n \log k$ } → k-shifted elements

Brute $n \log n$.

Allowed → $n \log k$

Sorted → 10 20 30 40 50 60 70 80

Median Priority Queue

Friday, 2 July 2021 7:45 PM

peek = median

add 10
add 20
add 30
add 40
peek
add 50
peek
remove
peek
remove
peek
remove
peek
remove
peek
quit

20
30
40
50
10

Max P.Q.

Left

50

Min PQ

Right

Median →

left > right → left = median

<right> left → right = median.

Even →

priority → Median.

↳ increasing order → middle

→ Even → First Half Last

10 | 20 | 30 | 40

median.

odd → left

median

S_1 S_1

S_{i-1} median S_1

add-50

add → 80

add → 45

add → 40

peek → 45

add → 38

peek → 45

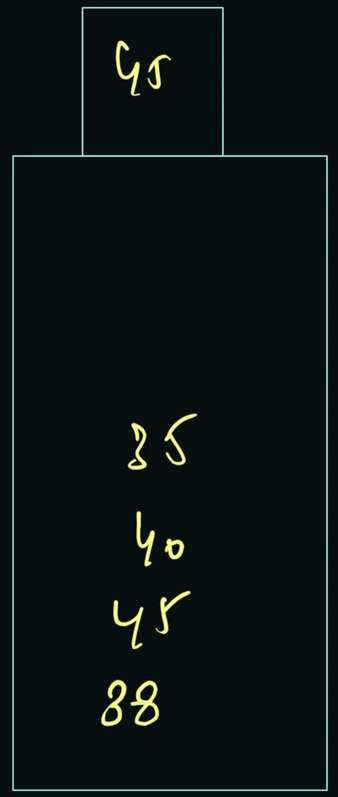
add → 35

peek → 40

add → 60

add → 70

peek → 45



left P.Q

(max)

right P.Q

(min)

peek

```
if(this.size() == 0){
    que underflow.
}
if(right.size() > left.size)
```

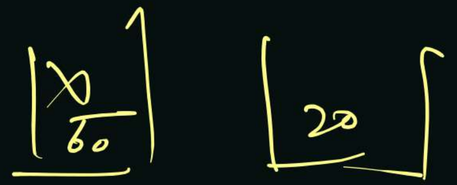
right.peek();

```
} else {
    // left.size == right.size
    // left.size() > right.size()
    left.peek();
}
```

add → 10

add → 20

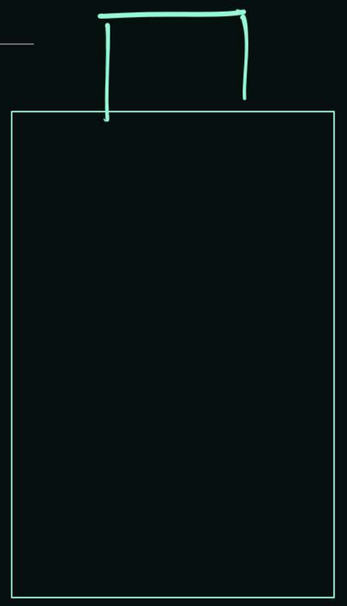
remove add → 50

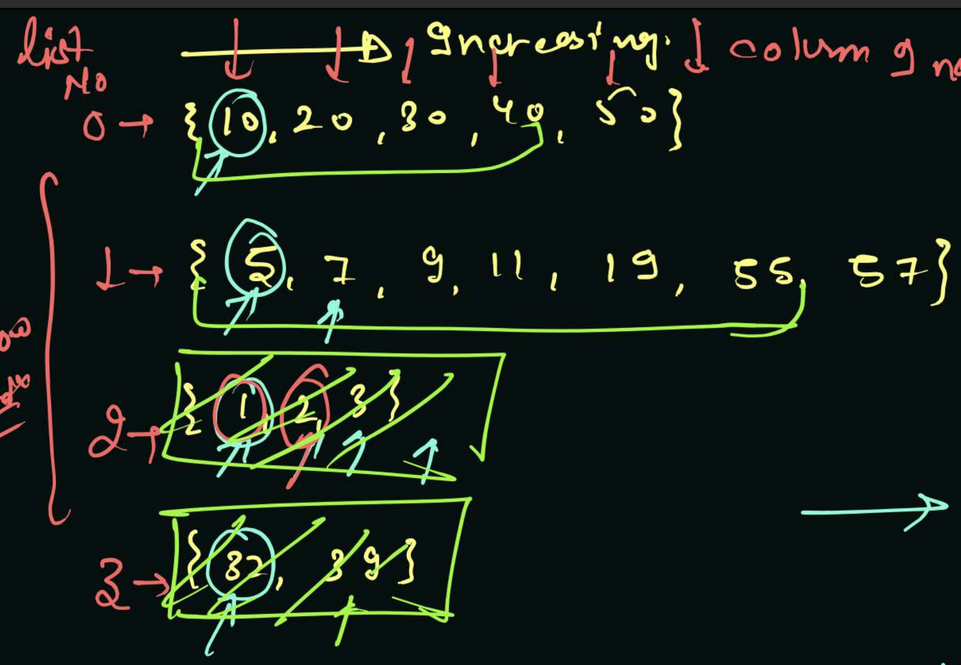


Merge K Sorted List

Friday, 2 July 2021 8:50 PM

5
10 20 30 40 50
7
5 7 9 11 19 55 57
3
1 2 3
2
32 39





Removal → ✓

→ Insertion of next Element from list which is just Now removed ?

Final Result → 1, 2, 3, 5, 7, 9, 10, 11, 19, 20, 30, 32
35, 40, 50, 55, 57

wrapper class →

priority Helper class
↳ data ⇒ compareTo on data.

Heap Creation

Friday, 2 July 2021 8:50 PM

Function-

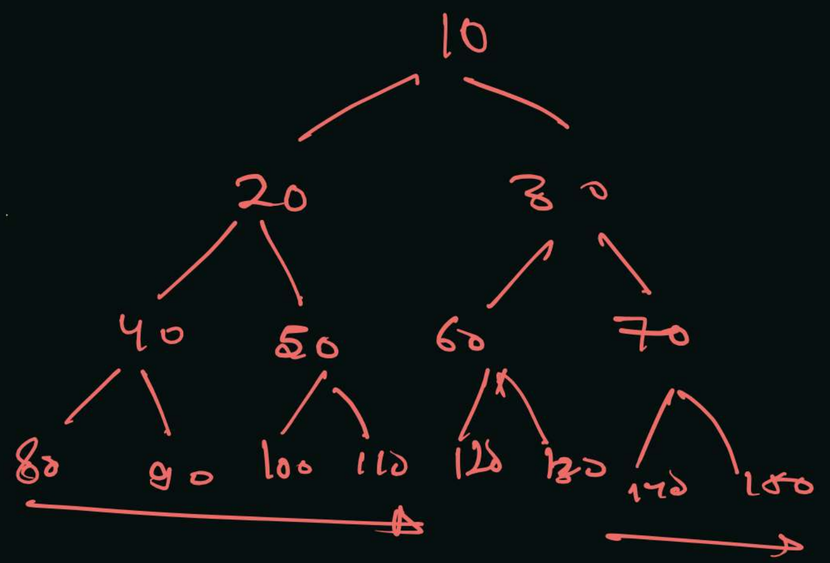
1. add → Insertion of new Element according to priority
2. remove → remove priority Elemet, return its value-
3. peek → return value of priority Element
4. size → size
5. isEmpty → size == 0] True
   otherwise → false

1. Complete Binary Tree
2. Heap order priority

Complete Binary Tree

① A level is completely Filled by nodes. Except last level.

② In the last level Addition of nodes is from left to Right



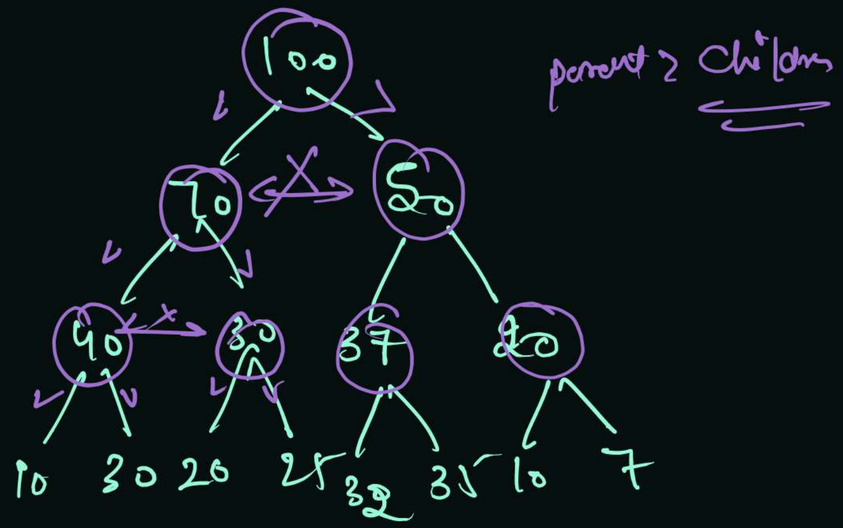
Heap order property

① Parent have more priority than children

NOTE: There is no connection between siblings.

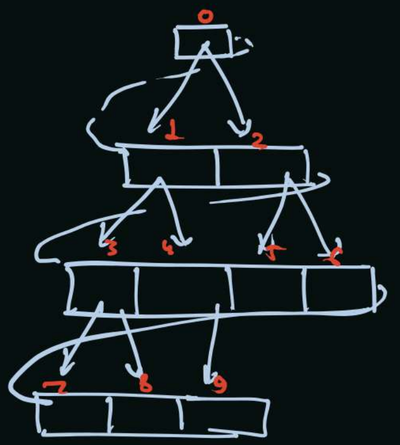
priority (parent) $\geq$ (child)

max → priority.

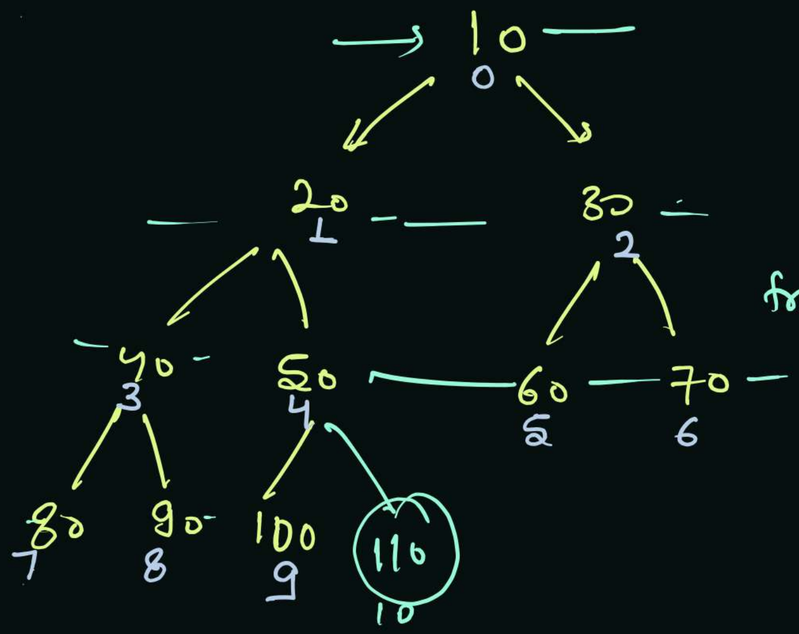


data → 10, 20, 30, 40, 50, 60, 70, 80, 90, 100

(Mon)



Advantage of array over Tree → Why we are using array rather than tree ??



(1) traversal in array – parent index = i

left child index = 2*i + 1

right child index = 2*i + 2

child index = i

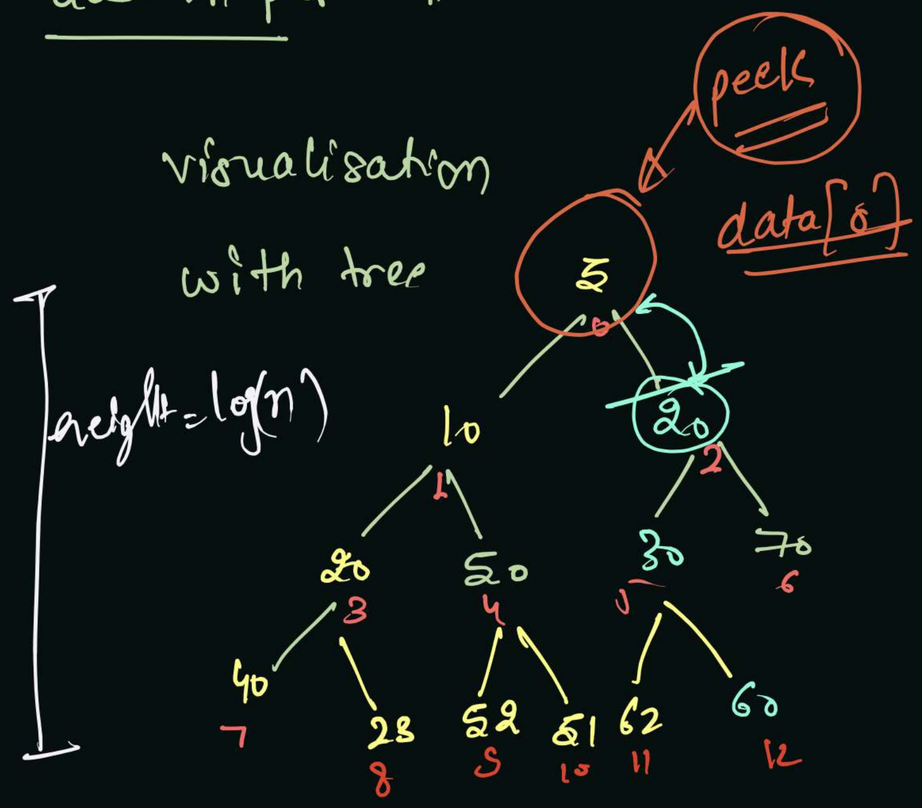
parent index = $\left(\frac{i-1}{2}\right)$

Traversal from child to parent is feasible in array rather than tree.

(2) add new element → O(n) in tree

But O(1) in array

add in pq → 10



data = {5, 10, 20, 20, 50, 30, 70, 40, 23, 52, 51, 62, 60}

priority on parent (min)

priority(parent) > priority(child)

function of add

```
add(int val){
    data.add(val);
    upheapify(data.size() - 1);
}
```

traversal of correction

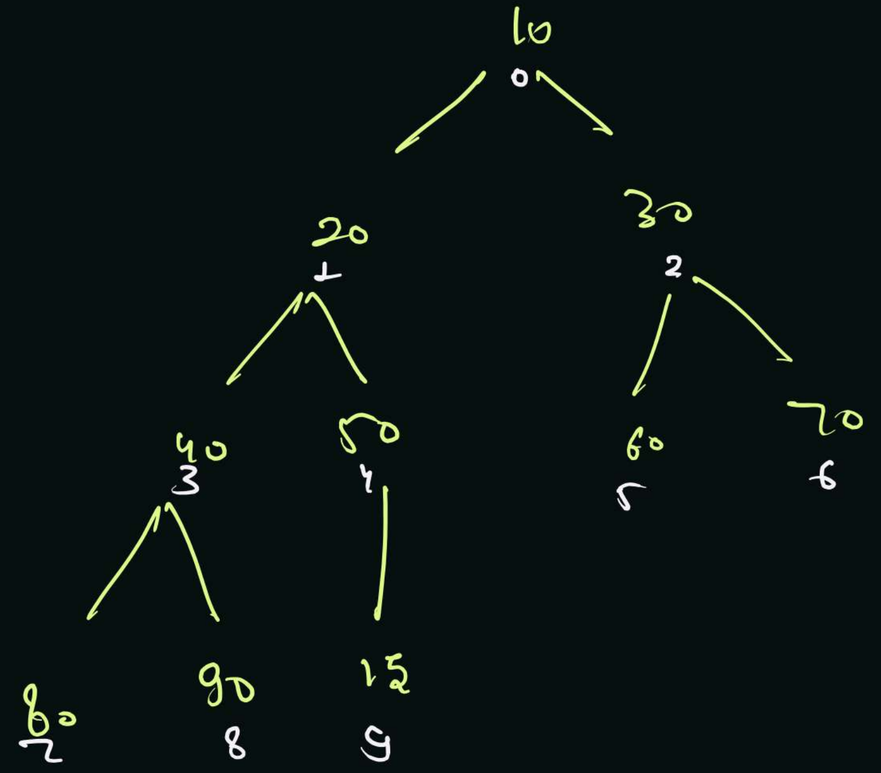
↳ Upheapify. } Recursive

parent Idx = $\left(\frac{i-1}{2}\right)$

complexity → add → log(n)

due to upheapify.

Upheapify →



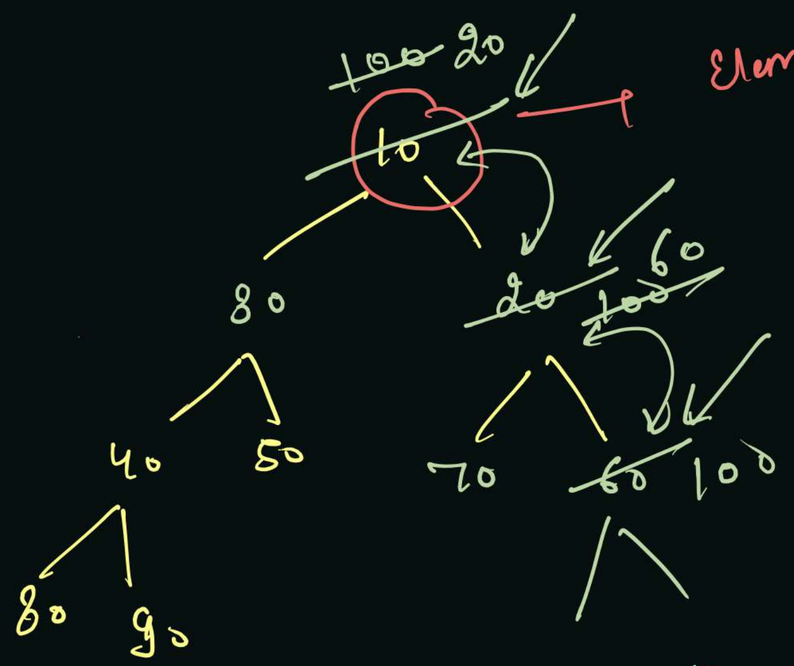

```
upheapify ( int indx ) {

    int pi = (i-1)/2

    if( pi >= 0 && data[i] < data[pi] ) {
        swap( i, pi );
        upheapify (pi);
    }

}
```

remove in PQ →



Element which is going to remove

remove from 0th index on data →

→ data.remove(0)

↳ complexity → O(n)

↳ Heap order property ✗

① swap(0, data.size() - 1) → O(1)

② remove last Element from data array. (O(1))

③ down heapify() || or can maintain

remove() {

```
int val = data.get(0);
swap(0, data.size()-1);
data.remove(data.size()-1);
downheapify(0);
return val;
}
```

heap order

downheapify property.

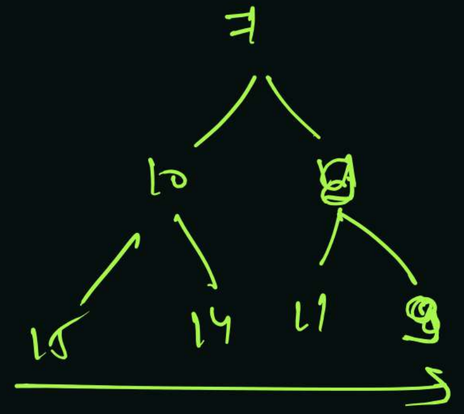
complexity → O(1) + O(log n) → O(log n)

Efficient Heap Constructor

Friday, 2 July 2021 10:45 PM

pass an array ⟶ {10, 7, 11, 15, 14, 9, 8}

in constructr. → data tree



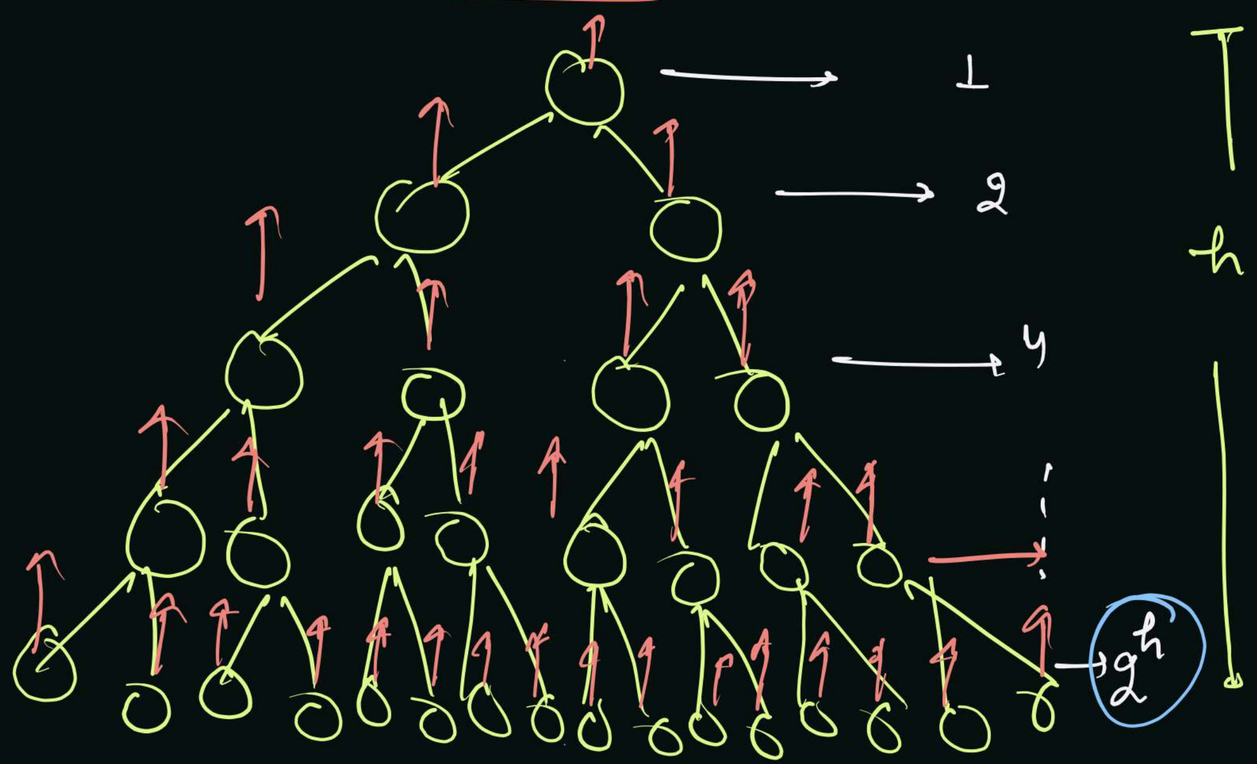

```
for(int i=0; i<arr.length; i++){
    add(arr[i]);
}
```

Steps – (1) Recieve an array in constructr

(2) add call for Every Element

Complexity Analysis →

add → upheapify → worst → $\log n$.



$h = \log n$

Time. $T(n) = 2^h \times h + 2^{h-1} \times h-1 + 2^{h-2} \times h-2 + \cdots$

$\cdots\cdots\cdots + 4 \times 2 + 2 \times 1 + 1 \times 0$

$T(n) = 2^h * h \rightarrow \cdots\cdots$

$2^{\log n} * \log n \rightarrow \cdots\cdots$

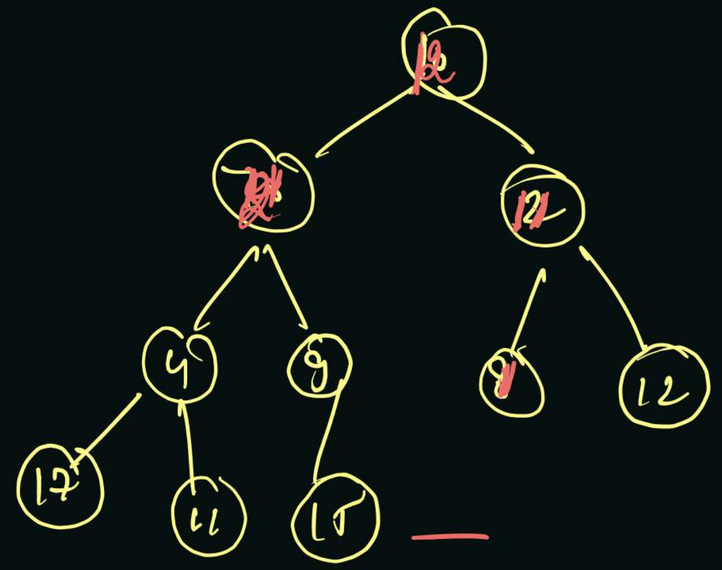
$n * \log n \rightarrow \cdots\cdots$

$a^{\log_a b} = b$

How to reduce Comp.?? $T(n) > O(n \log n)$

arr [] → {10, 70, 2, 4, 9, 8, 12, 17, 11, 15}

Visualise data list as tree →



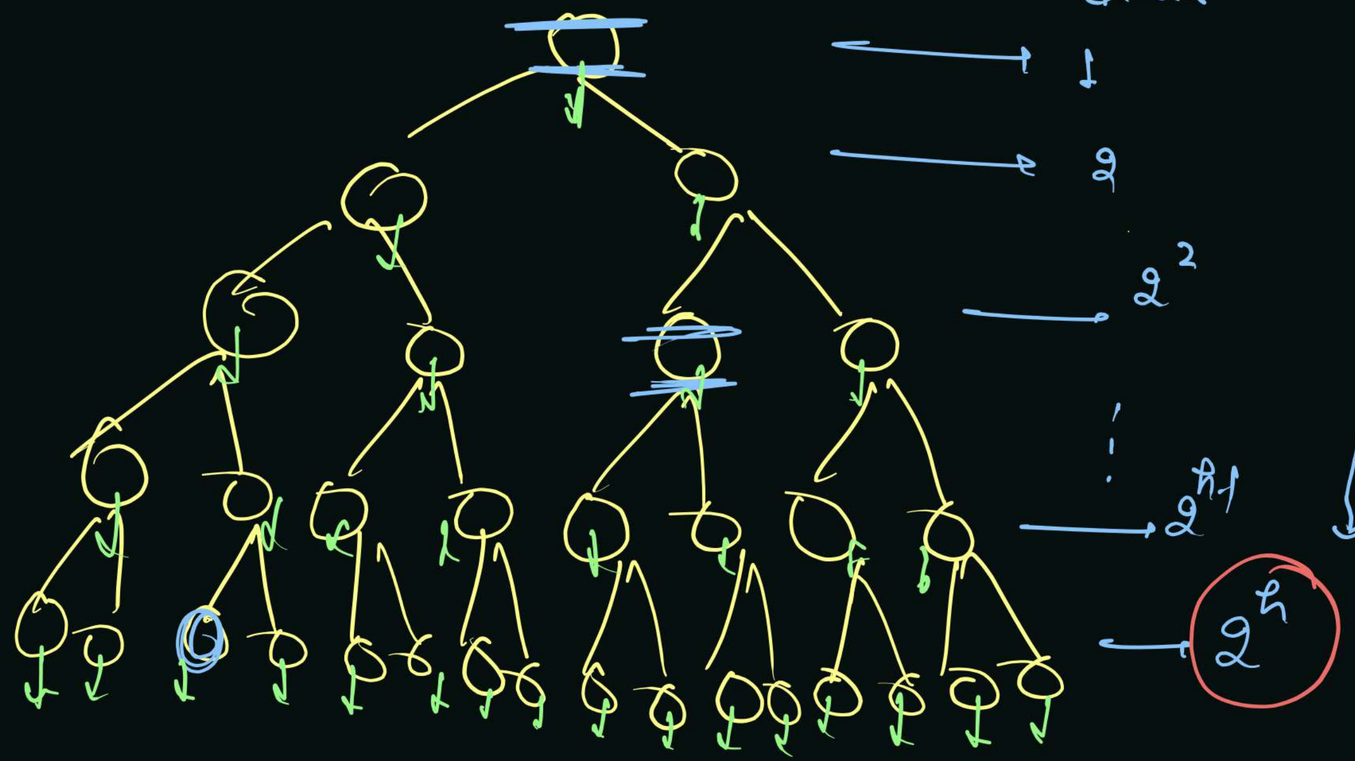
Step → ① add the element of array in data list

② Down heapify from End of data list.

③ Final Result is achieved.

Time Complexity Analysis

↓



complexity of downheap :-

$\log(m)$ · where 'm' is no. of nodes present down the current node.

Elements & Time complexity below that node.

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

$$= 2^h \times 1 + \frac{1}{2}\left[2^h \sum_{i=0}^{h} (2^i * (h-i))\right] = h \sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i 2^i$$

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

$$= h2^h - h - (2^h \cdot h - 2^h + h - 1) + 2$$

$$= \cancel{h2^h} - h - \cancel{h2^h} + 2^h - h + 1 + 2$$

$$T(n) = 2^h - 2h + 3 \qquad h = \log n.$$

$$T(n) = 2^{\log n} - 2\log n + 3$$

$$T(n) = n - 2\log n + 3$$

$$\boxed{T(n) = \Theta(n)}$$