Lecture 25

Wednesday, 27 May 2020 6:48 PM

heaps :- list
↳ { 3, 8, 1, 5, 6, 4, 2, 9, 7, 11 }

max-priority → time Complexity → O(n).
↳ add ↳ O(1).

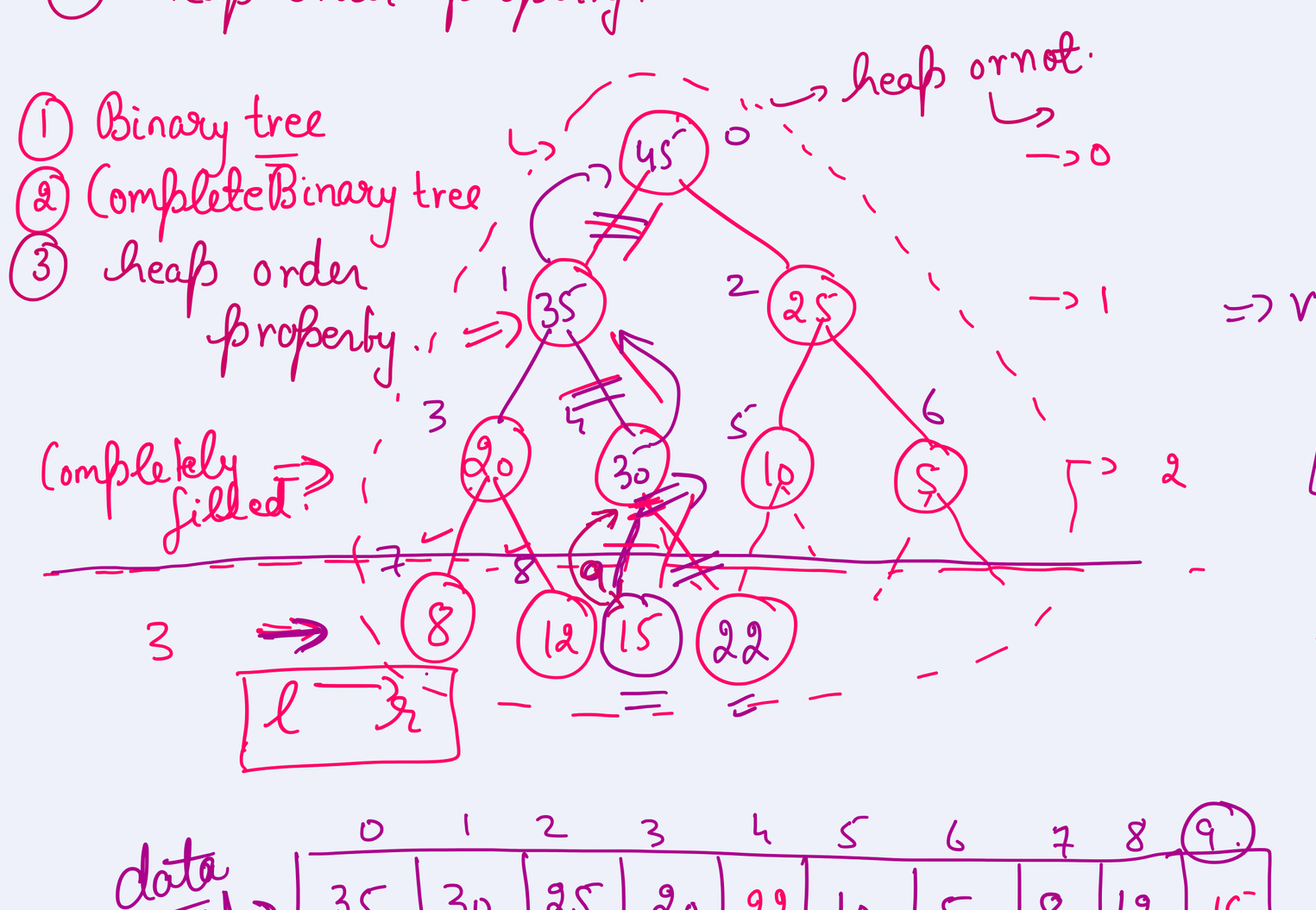
Sorted → { 1, 2, 3, 4, 5, 6, 7, 8, 9, 11. }
↳ max ↳ O(1) ↳ Const. (10)
↳ add → O(n).

add → O(1) , O(n).
max ↳ O(1) , O(n).

mid-way
↳ get max → O(1).
add → O(log n).

Heap ↳ ① Binary tree ↳ 0/1/2.
② Complete Binary tree ↳ except last level all levels are filled and last level is filled from left to right
③ Heap Order Property.

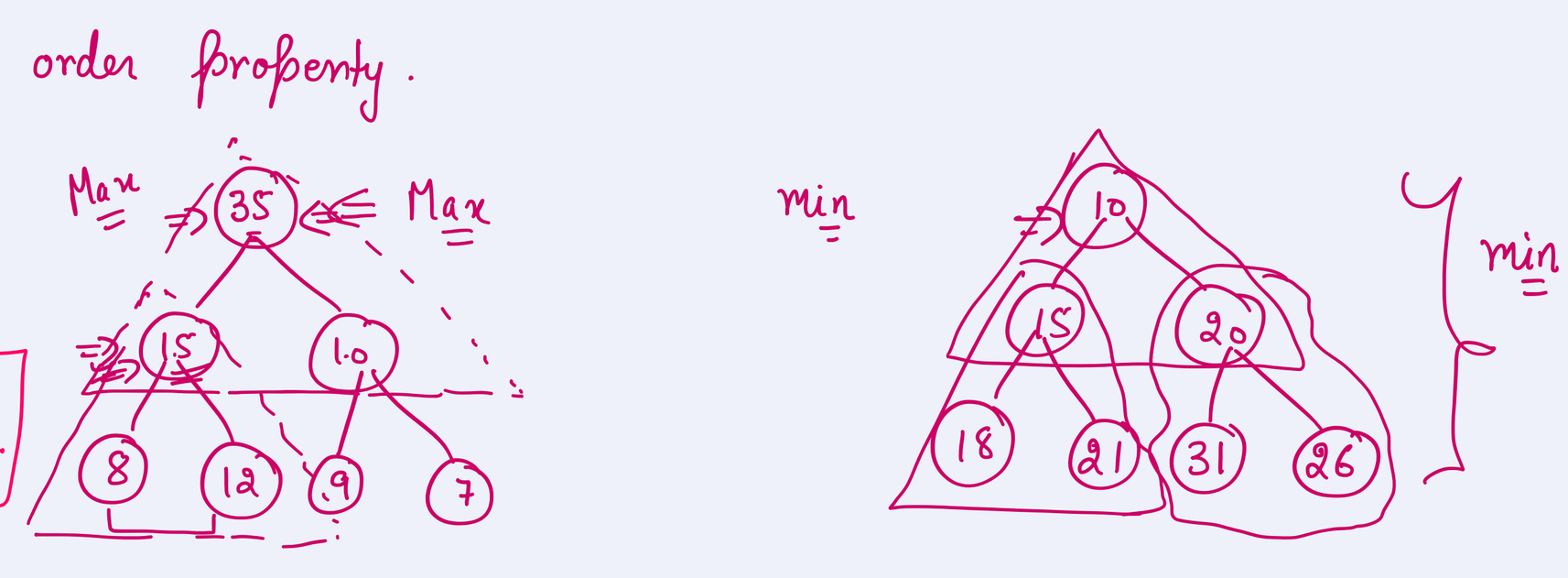
① Binary tree
② Complete Binary tree
③ heap order property



add(22) → visualise

max → height → O(log n).
add → (log n).

③ heap order property.



35

data ↳

0	1	2	3	4	5	6	7	8	9
35	30	25	20	22	10	5	8	12	15

height → O(log n).

```
public void add(int data){   // 22
    this.data.add(data);
    this.upheapify(this.data.size()-1);
}
```

0th index → max/min → O(1) ↳ Const.

add ⇒ max → O(1) (9 − 1) / 2 ⇒ 8/2 ⇒ (4)

22 > 15

22 < 30 22

⇒ (4 − 1) / 2
⇒ (1)

pi = (ci − 1)/2

worst case → O(log n)

35 ⇒ 45 ⇐ 25
20 ⇒ 35 ⇐ 30

```
private void upheapify(int ci){   // 9
    if(ci == 0){
        return;
    }
    int pi = (ci-1)/2;   // (4)
    if(islarger(pi, ci)){
        this.swap(pi, ci);
        this.upheapify(pi);   // ⇒ (4)
    }
}
```

false. ⇒ pi, ci
⇒ (pi)

CompareTo((int i) , (int j)){   [pi, child]
- i > j ⇒ +ve. > 0
- i < j ⇒ −ve < 0
- i = j ⇒ 0
}

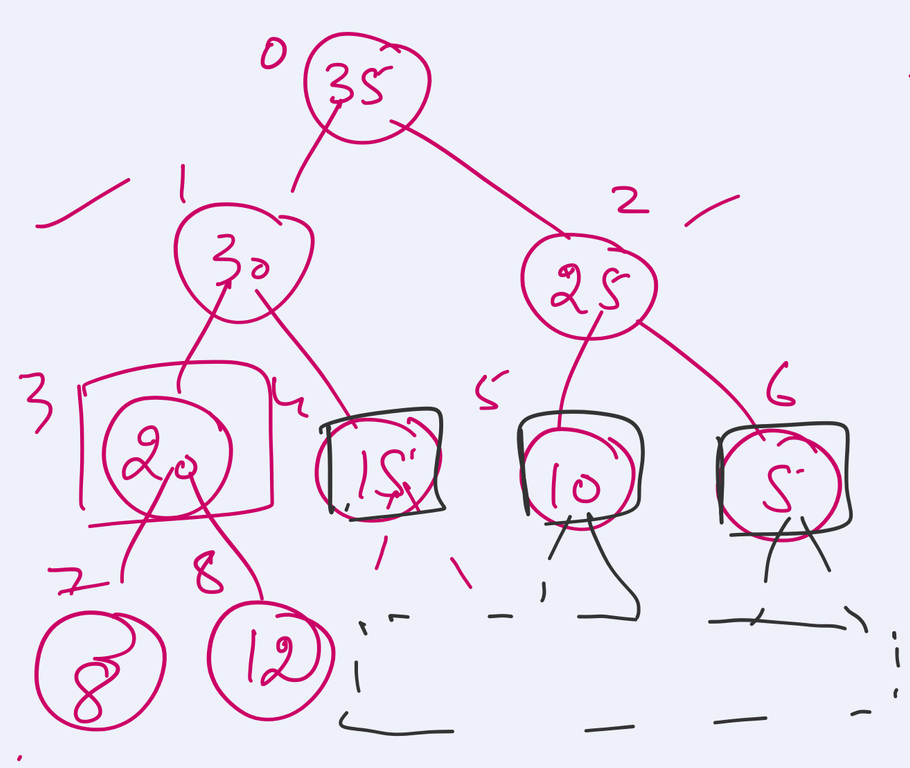
i − j > 0 ↳ int

⇒ Binary tree ⇐

swap(int pi, int ci){
}

Students ↳ marks ↳ name ↳ rollno → { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }

```
public boolean isLarger(int pi, int ci){
    int pvalue = data.get(pi);
    int cvalue = data.get(ci);
    if(this.ismin){
        pvalue.CompareTo(cvalue) < 0;
    } elses
        pvalue.CompareTo(cvalue) > 0;
    }
}
```



pi = 0.
lci = 2 * pi + 1 ; // 1
rci = 2 * pi + 2 // ②

lci 2 * 3 + 1 = 7
rci 2 * 3 + 2 = 8.

lci = 2 * 4 + 1 = (9)
rci = 2 * 4 + 2 = (10)

total elements → 9

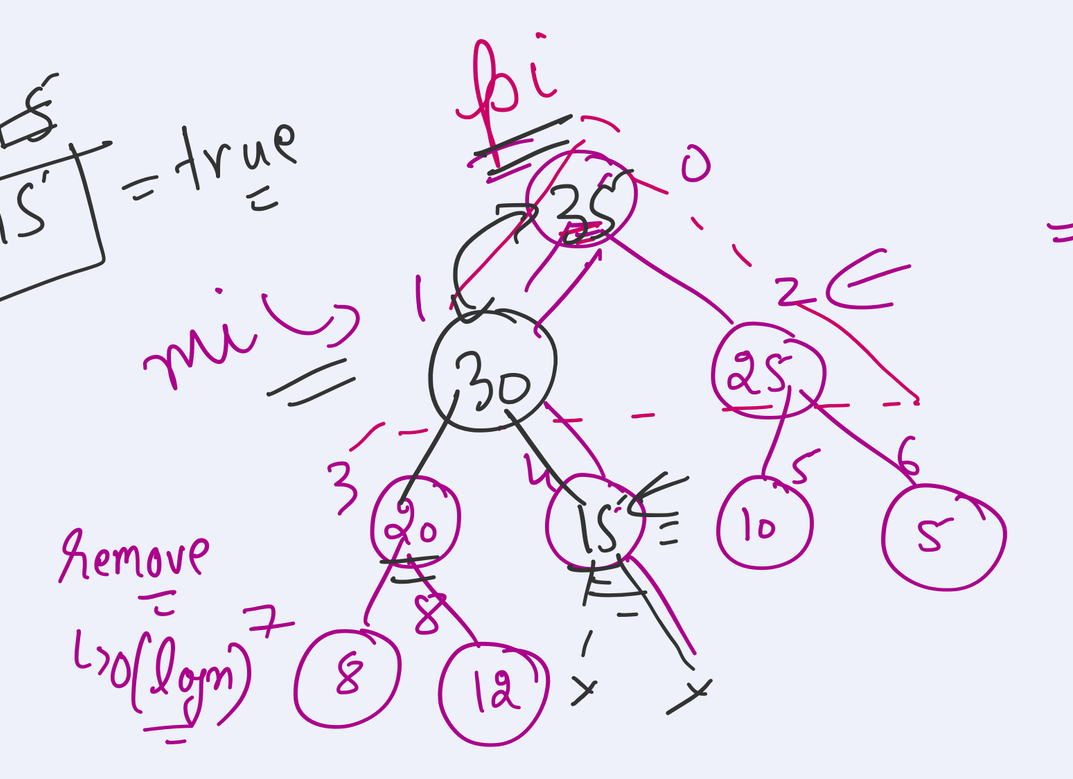
```
if (lci < data.size()){
    display(lci);
}
```

⇒ null pointer exception

25 > 35 ⇒ no.

| 10 > 15 < 9 |

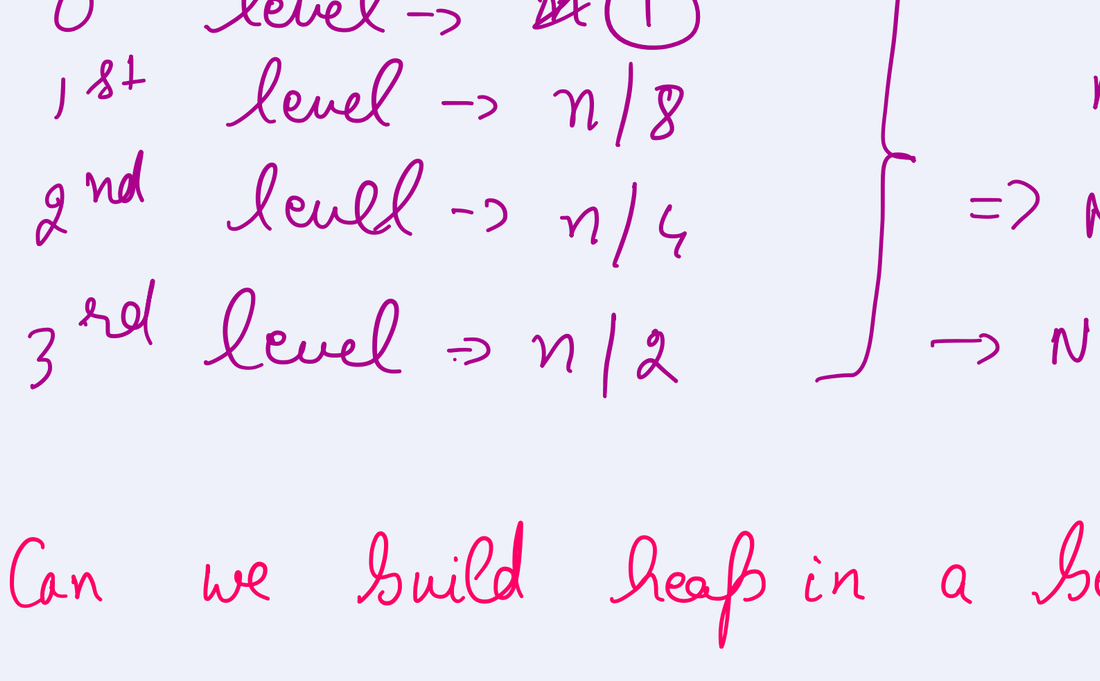
35 > 15 = true



⇒ Max heap.
↳ maxheapx
→ max-heap

```
remove(){
    T rv = this.data.get(0);
    swap(0, data.size()-1);
    this.data.remove(this.data.size()-1);
    this.downheapify(0);
}
```

```
private void downheapify(int pi){   // 0
    int mi = pi;   // ← (0)
    int lci = 2*pi+1;   // ①
    int rci = 2*pi+2;   // ②
    if(lci < this.data.size && this.islarger(lci, mi)){
        mi = lci;
    }
    if(rci < this.data.size && this.islarger(rci, mi)){
        mi = rci;
    }
    if(mi != pi){   // (1)
        this.swap(pi, mi);
        this.downheapify(mi);
    }
}
```



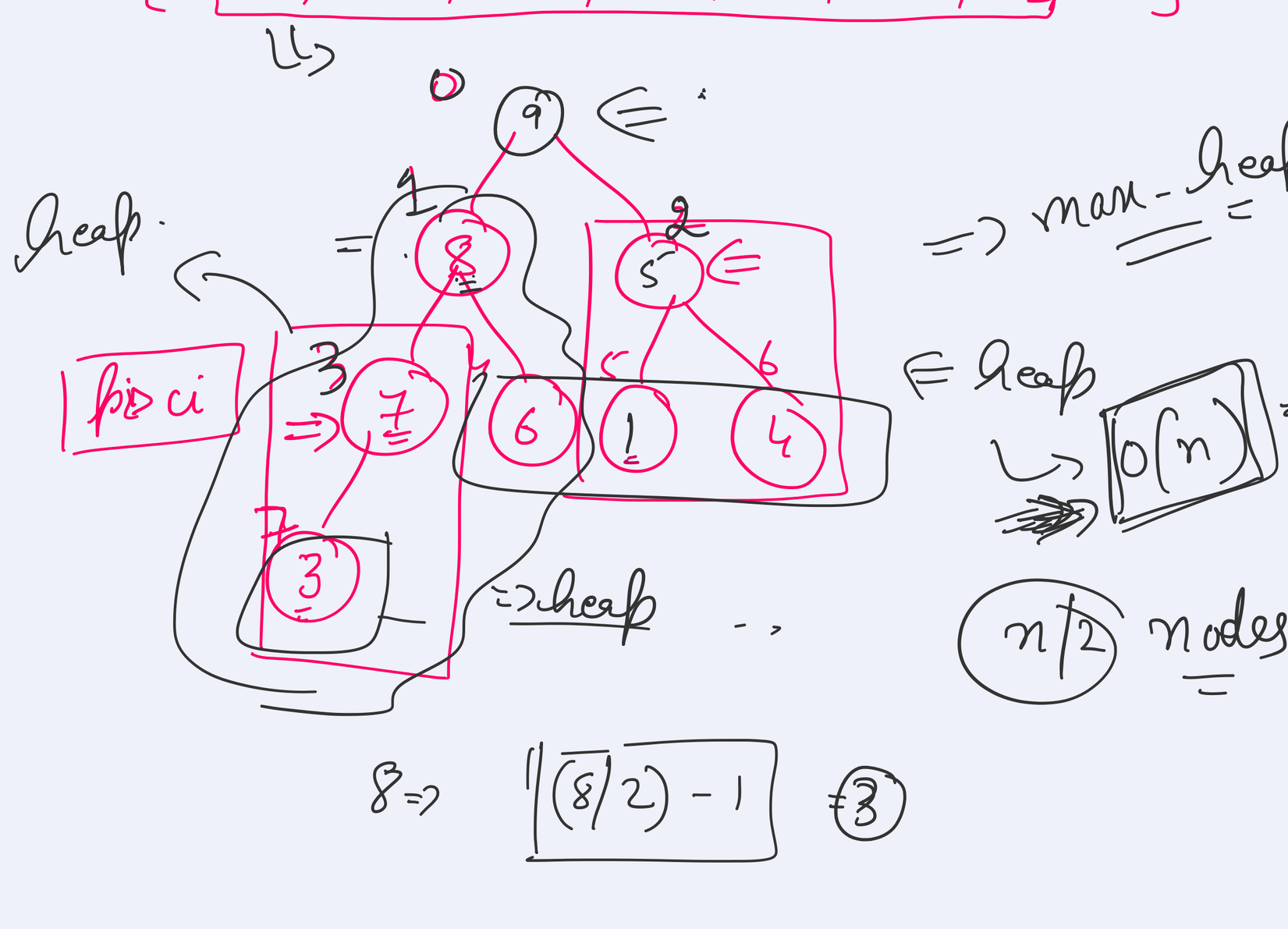
(15) elements
⇒ (15) N elements

add ⇒ n/2 log n.
↳ n/2 log n ⇒ n log n
n log n

0th level → ① N / N = 1
1st level → n/8 N/8
2nd level → n/4 ⇒ N/4 ⇒
3rd level → n/2 → N/2

Can we build heap in a better way?

arr: { 5, 8, 1, 7, 6, 9, 4, 3 }



⇒ max-heap

↳ O(n) ⇒

(n/2) nodes

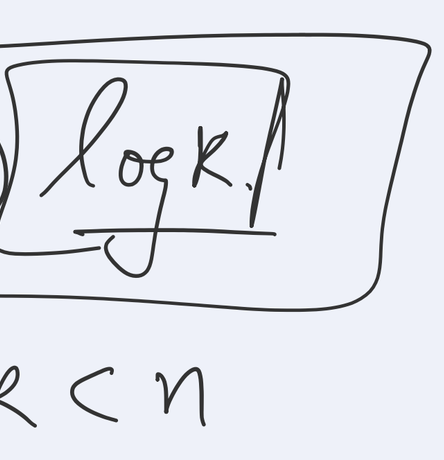
8 ⇒ (8/2) − 1 = (3)

Resume at 8:55 pm.

arr = { 3, 2, 11, 43, 12, 5, 13, 50. };
k ⇒ min. (n − k)
Kth largest Element ↳ ↳

min heap of k elements. ↳

(3)



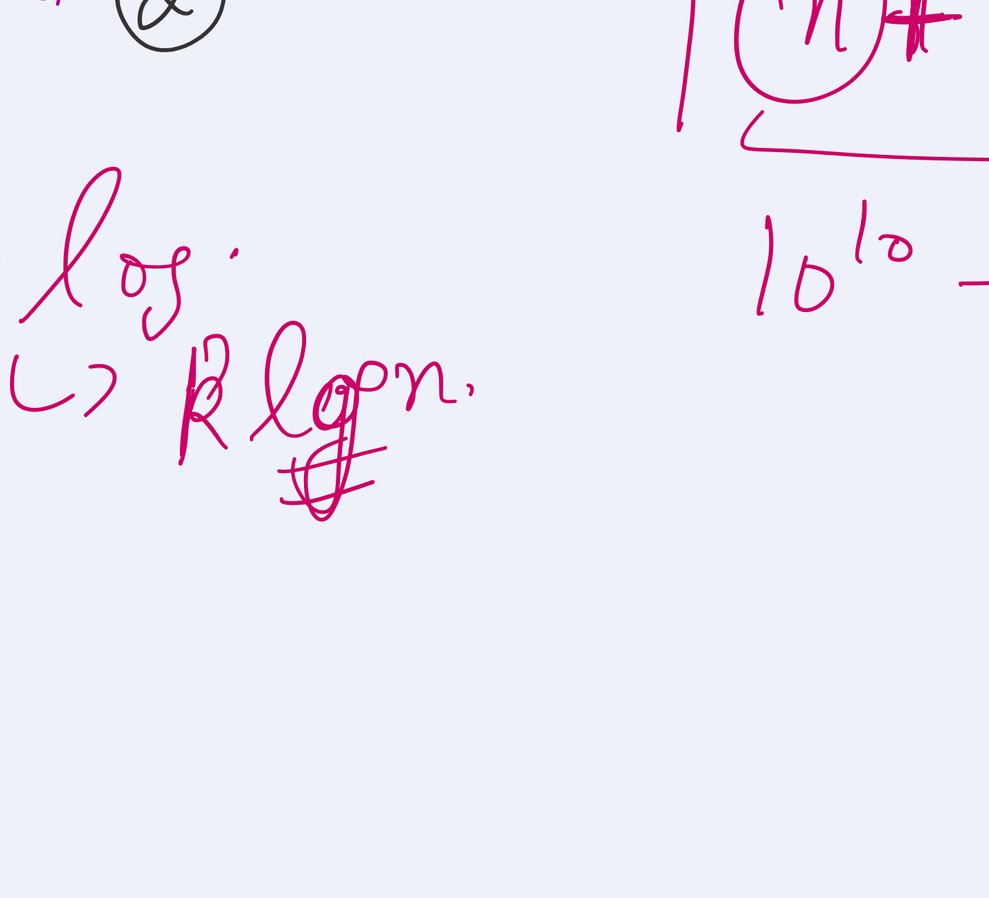
⇒ 50
43
13

(n − k) log k.

k < n

Sol 1 → Sort and return n−K.
↳ n log n.

Sol 2 →
max heap
↳



→ max heap

↳ n

(13) ↳

(K − 1) 3rd largest.
↳ remove ↳ 2 times.

(n + k log n) << n log n

10^{10} + k log 10^{10} < 10^{10} · log 10^{10}

(K − 1) log.
↳ k log n.

heaps
↳ Friday ↳ Tries ↳ Friday.