Heaps and heap sort.

D and C.

Divide and Conquer

(Previous Year Questions)

Q1. Find the solution to the following recurrences equation

[GATE : 2MARKS]

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

$T(1) = 1$

$f(n)$ $\quad n^{\log_b a}$

n $\quad n^{\log_2 1} = 1$

$O(n)$

Q2. Assume that the last element of the set is used as partition element in Quicksort. If n distinct elements from the set [1 ... n] are to be sorted, give an input for which Quicksort takes maximum time.

[GATE : 2MARKS]

increasing order

or

decreasing order

Q3. The recurrence relation that arises in relation with the complexity of binary search is:

[GATE : 2MARKS]

A. $T(n) = 2T\left(\frac{n}{2}\right) + k$, k is a constant

B. $T(n) = T\left(\frac{n}{2}\right) + k$, k is a constant

C. $T(n) = T\left(\frac{n}{2}\right) + \log n$

D. $T(n) = T\left(\frac{n}{2}\right) + n$

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

Q4. Merge sort uses:

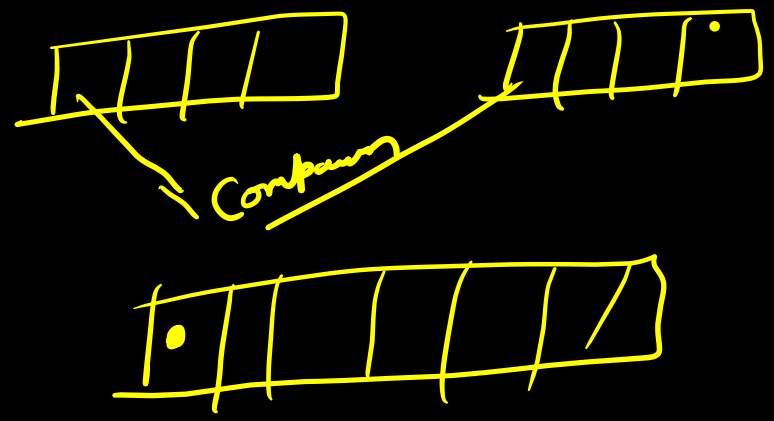
[GATE : 1MARK]

a) Divide and conquer strategy
b) Backtracking approach
c) Heuristic search
d) Greedy approach

Q5.For merging two sorted lists of sizes m and n into a sorted list of size m + n, we require comparisons of

[GATE : 1MARK]

a) O(m)
b) O(n)
c) O(m + n)
d) O(logm + logn0)



Q6.The recurrence relation

$T(1) = 2$

$T(n) = 3T(n/4) + n$

has the solution T(n) equal to

a) O(n)

b) O(logn)

c) $O(n^{3/4})$

d) none of these

[GATE : 2MARK]

$f(n)$ ↓ n

$O(n)$

$n^{\log_b a}$

$n^{\log_4 3} = n^{0.792}$

to diff by polynomial

$n^{1-0.792}$

Q7.Quicksort is run on two inputs shown below to sort in ascending order

(i) 1, 2, 3,....., n

(ii) n, n-1, n-2,....., 2, 1

Let C1 and C2 be the number of comparisons made for the inputs (i) and (ii) respectively. Then,

a) C1 < C2

b) C1 > C2

c) C1 = C2

d) We cannot say anything for arbitrary n

[GATE : 2MARKS]

Q8. Let T(n) be the function defined by $T(1) = 1,\ T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + \sqrt{n}$ for n>=2 **[GATE : 2MARKS]**

Which of the following statements is true?

A. $T(n) = O\sqrt{n}$

B. $T(n) = O(n)$

C. $T(n) = O(\log n)$

D. None of the above

$T(n) = 2T(n/2) + \sqrt{n}$

$f(n)$ → $\sqrt{n}$

$n^{\log_b a}$

$n^{\log 2^2}$

n ✓

$O(n)$.

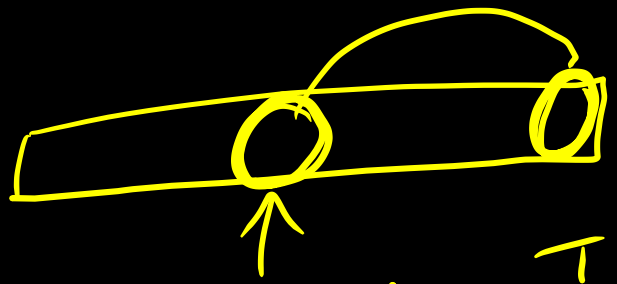
Q9.Let s be a sorted array of n integers. Let t(n) denote the time taken for the most efficient algorithm to determined if there are two elements with sum less than 1000 in s.Which of the following statement is true?

[GATE : 1MARK]

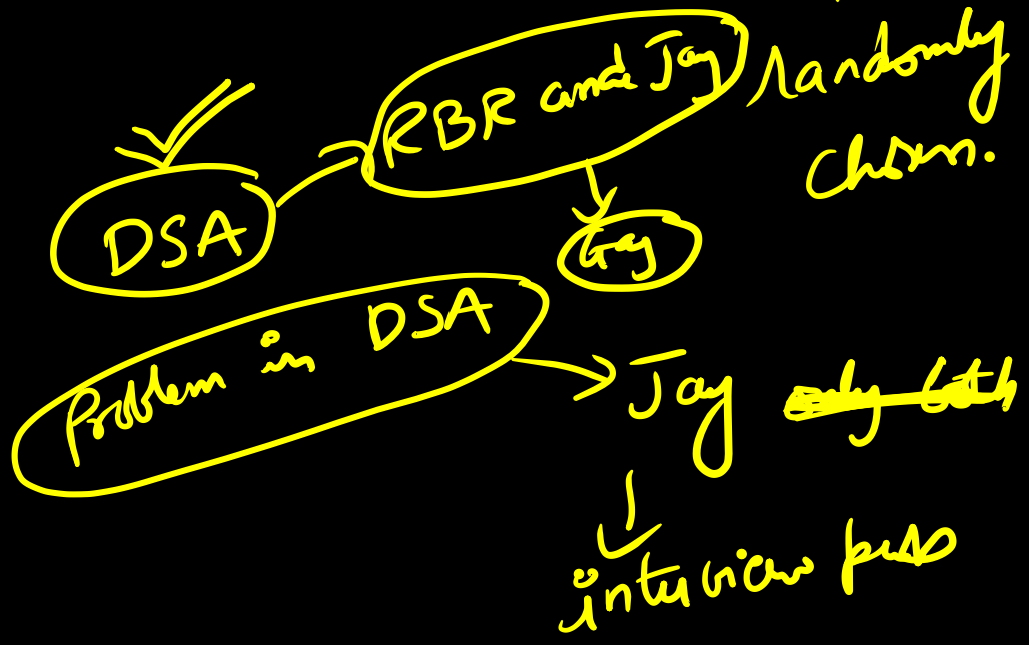
A. $T(n)$ is $O(1)$
B. $n \leq T(n) \leq n \log_2 n$
C. $n \log_2 n \leq T(n) < \frac{n}{2}$
D. $T(n) = \left(\frac{n}{2}\right)$

Q10.Randomized quicksort is an extension of quicksort where the pivot is chosen randomly. What is the worst case complexity of sorting n numbers using randomized quicksort? **[GATE: 1MARK]**

a) O(n)
b) O(n*log(n))
c) $O(n^2)$
d) O(n!)



$T(n) = T(0) + T(n-1) + O(n)$



Q11.Consider a list of recursive algorithms and a list of recurrence relations as shown below. Each recurrence relation corresponds to exactly one algorithm and is used to derive the time complexity of the algorithm.

[GATE : 2MARKS]

	Recursive Algorithm		Recurrence Relation
P	Binary search	I.	$T(n) = T(n-k) + T(k) + cn$
Q.	Merge sort	II.	$T(n) = 2T(n-1) + 1$
R.	Quick sort	III.	$T(n) = 2T(n/2) + cn$
S.	Tower of Hanoi	IV.	$T(n) = T(n/2) + 1$

Which of the following is the correct match between the algorithms and their recurrence relations?

A. P-II, Q-III, R-IV, S-I
B. P-IV, Q-III, R-I, S-II
C. P-III, Q-II, R-IV, S-I
D. P-IV, Q-II, R-I, S-III

P - IV
Q - III
R - I

Q12.Which of the following sorting algorithms has the lowest worst-case complexity?

[GATE : 1MARK]

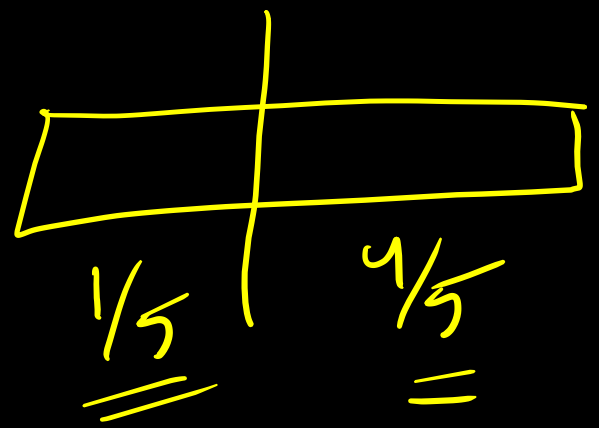
a) Merge sort
b) Bubble Sort
c) Quick Sort
d) Selection Sort

} Jay will discuss tomorrow

Q13.Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sub-lists each of which contains at least one-fifth of the elements. Let T(n) be the number of comparisons required to sort n elements. Then

[GATE : 2MARKS]

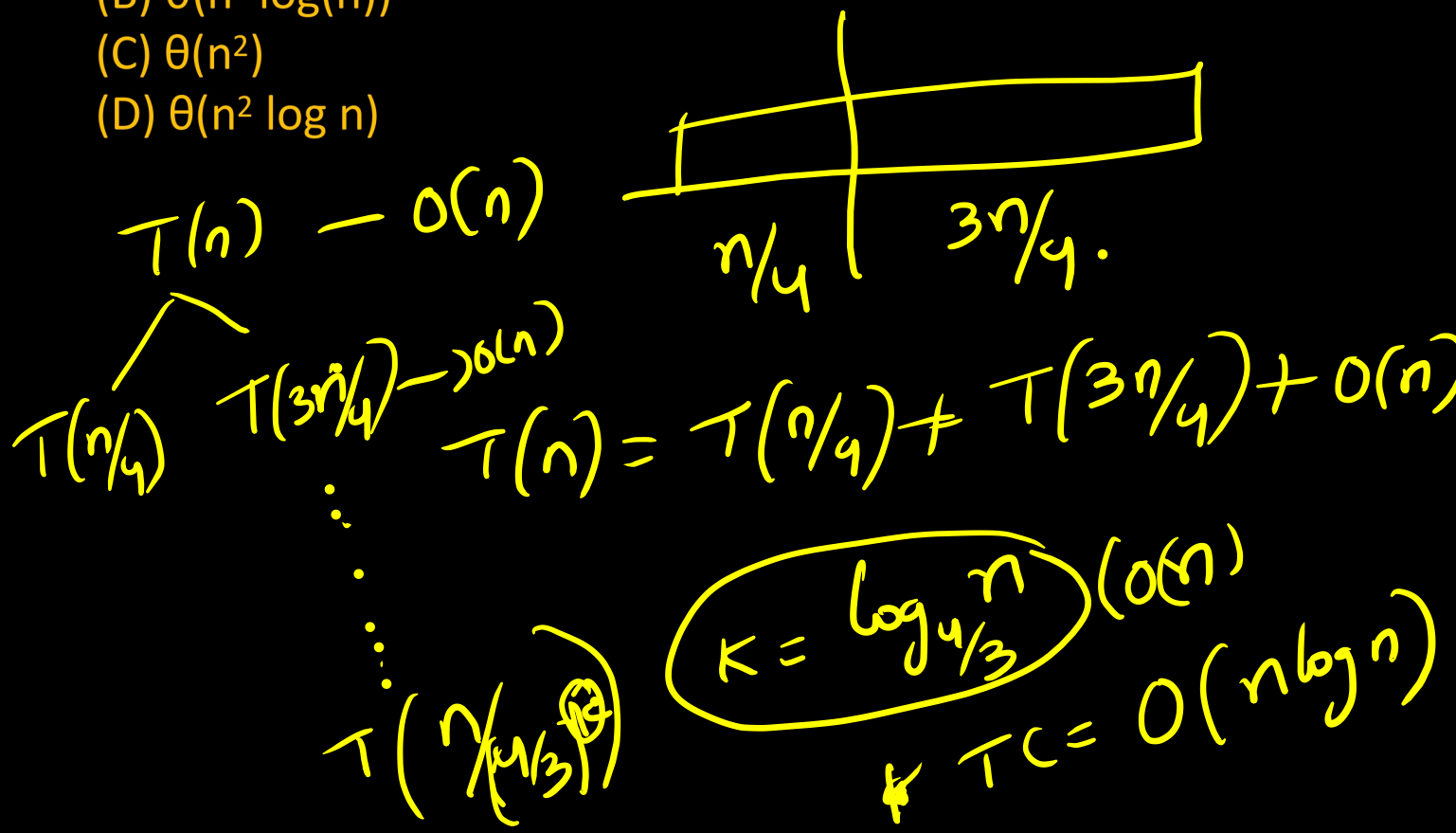
a) $T(n) <= 2T(n/5) + n$
b) $T(n) <= T(n/5) + T(4n/5) + n$
c) $T(n) <= 2T(4n/5) + n$
d) $T(n) <= 2T(n/2) + n$



Q14.In quick sort, for sorting n elements, the (n/4)th smallest element is selected as a pivot using an O(n) time algorithm. What is the worst-case time complexity of the quick sort?

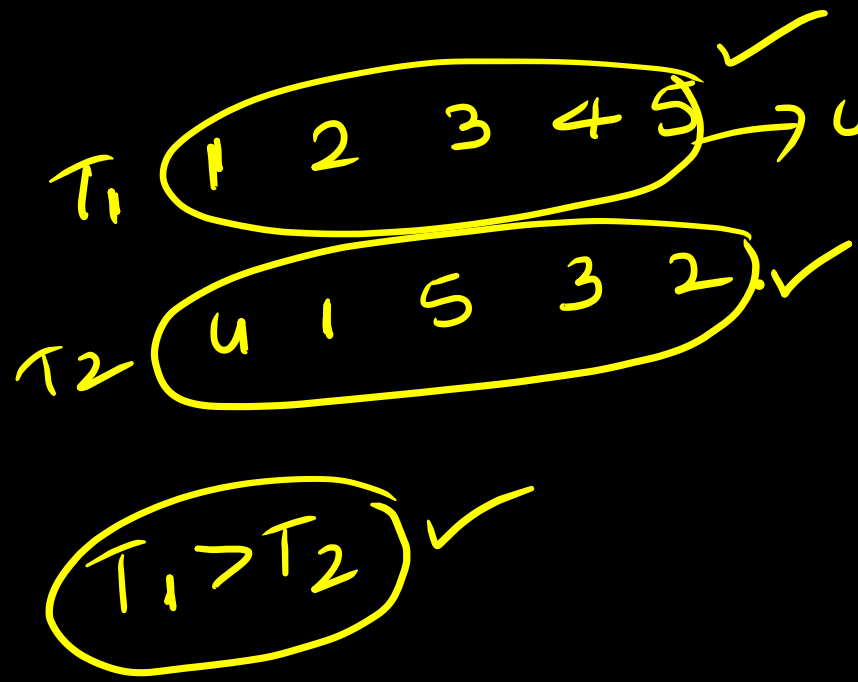
(A) θ(n)
(B) θ(n*log(n))
(C) θ(n^2)
(D) θ(n^2 log n)

[GATE : 2MARKS]



Q15.Let P be a QuickSort Program to sort numbers in ascending order using the first element as pivot. Let t1 and t2 be the number of comparisons made by P for the inputs {1, 2, 3, 4, 5} and {4, 1, 5, 3, 2} respectively. Which one of the following holds?

a) t1 = 5
b) t1 < t2
c) t1 > t2
d) t1 = t2



[GATE : 1MARK]

Q16.The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is _____.

[GATE : 2MARKS]

min max DAC $= \frac{3n}{2} - 2$

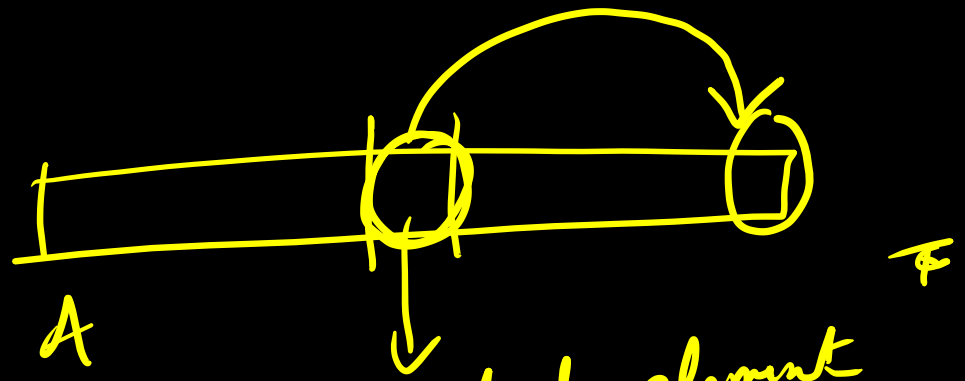
$3 \times \frac{\cancel{100}^{50}}{\cancel{2}} - 2$

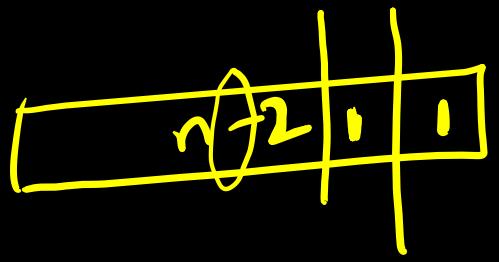
148

Q17.You have an array of n elements. Suppose you implement quick sort by always choosing the central element of the array as the pivot. Then the tightest upper bound for the worst case performance is

a) $O(n^2)$
b) O(n*log(n))
c) Theta(n*log(n))
d) $O(n^3)$

[GATE : 1MARK]



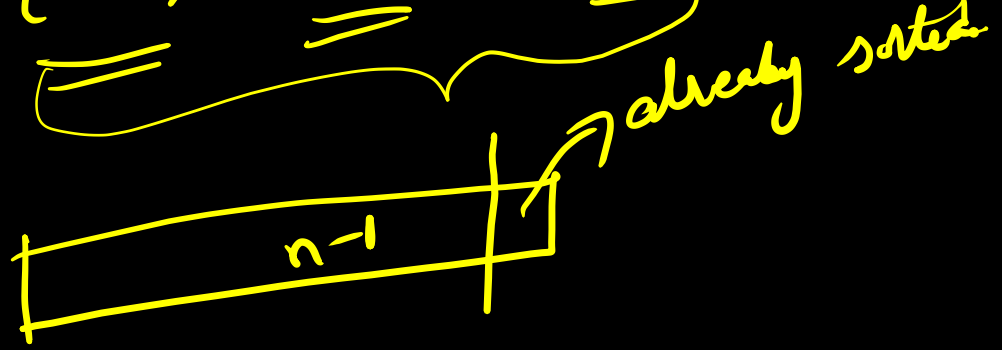


Q18. Which one of the following is the recurrence equation for the worst case time complexity of the Quicksort algorithm for sorting n(≥ 2) numbers? In the recurrence equations given in the options below, c is a constant.

a) $T(n) = 2T(n/2) + cn$
b) $T(n) = T(n-1) + T(1) + cn$
c) $T(n) = 2T(n-1) + cn$
a) $T(n) = T(n/2) + cn$

[GATE : 1MARK]

$T(n) = T(n-1) + T(0) + cn$.



Q19. Assume that a mergesort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes? [GATE: 2MARKS]

a) 256
b) 512
c) 1024
d) 2048

nlogn

TC ms = $O(n\log n)$

$= C \cdot n\log n$

$C \cdot n\log n = 6 \times 60$ sec.

$\frac{5}{64} \cdot n\log n = 6 \times 60$

$n = ?$ 512

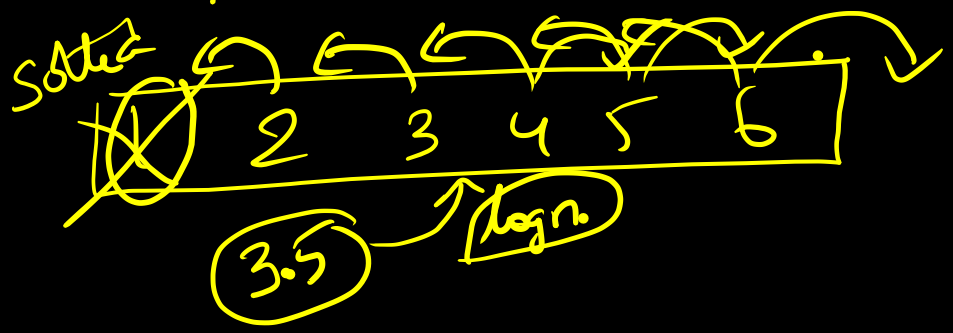
$C \cdot n\log n = 30$

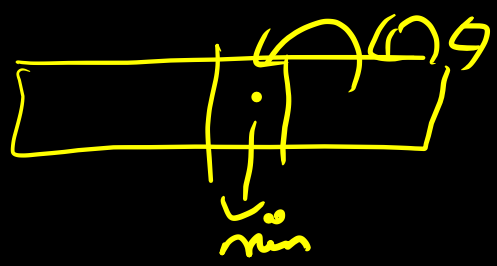
$C \cdot 64 \times \log 64 = 30$

$C = ?$ $5/64$

	insert	Search	Find min	Delete min
unsorted array	O(1)	O(n)	O(n)	O(n) + O(n) = O(n) (search, movement)
Sorted array	O(log n) + O(n) = O(n)	O(log n)	O(1)	O(1) + O(n) = O(n)
unsorted LL	O(1)	O(n)	O(n)	O(n)
min heap	log n	O(n)	O(1)	O(log n)

max heap





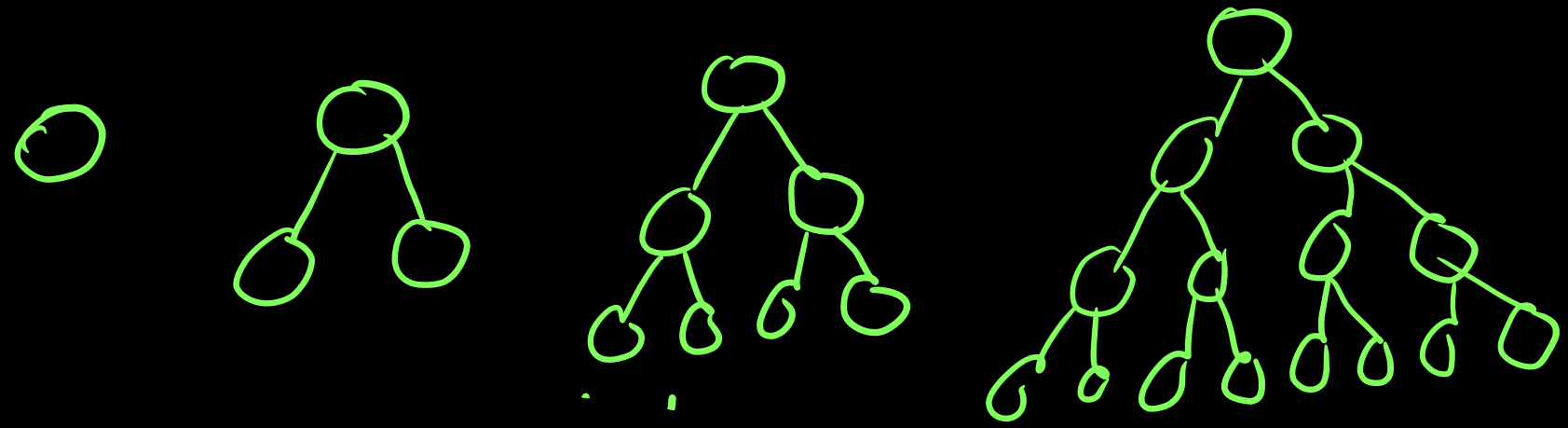
heap data struct is a priority queue

Ex: max heap → max element → priority is given to max elem

min heap → min elemt. → priority is given to min elemt

queue → FIFO

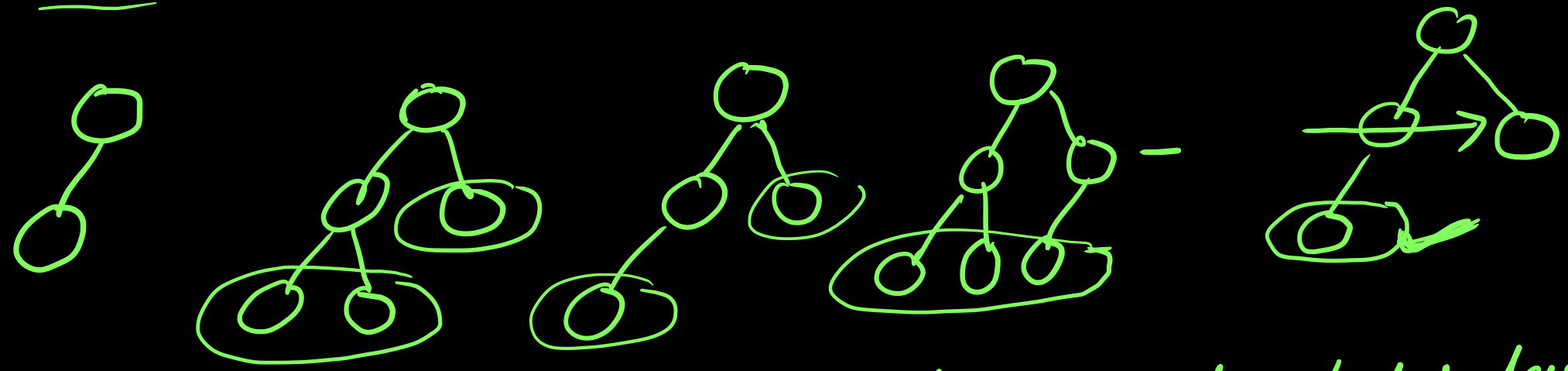
Complete binary tree:- (CLRS)



Complete

- All leafs are at last level
- all non leafs have two children

Almost Complete binary tree:



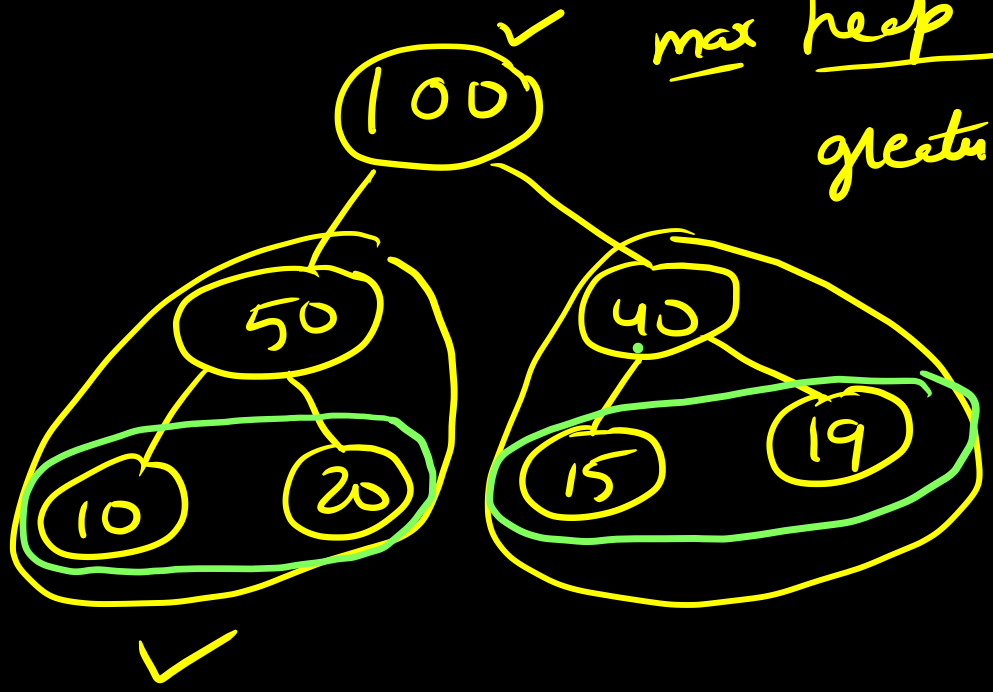
leafs can be present in last level and last but 1 level.

It should be filled from left to right.

You have to go to next level only if the current level is filled.

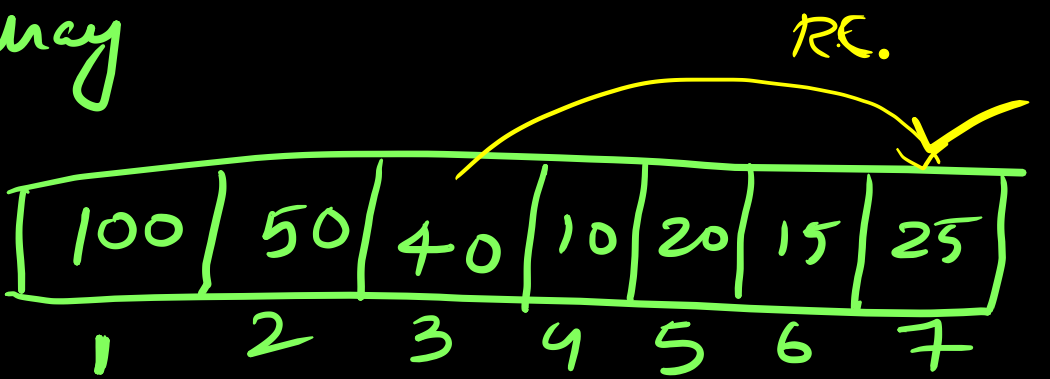
We will do only max heap related things and these are same for min heap also.

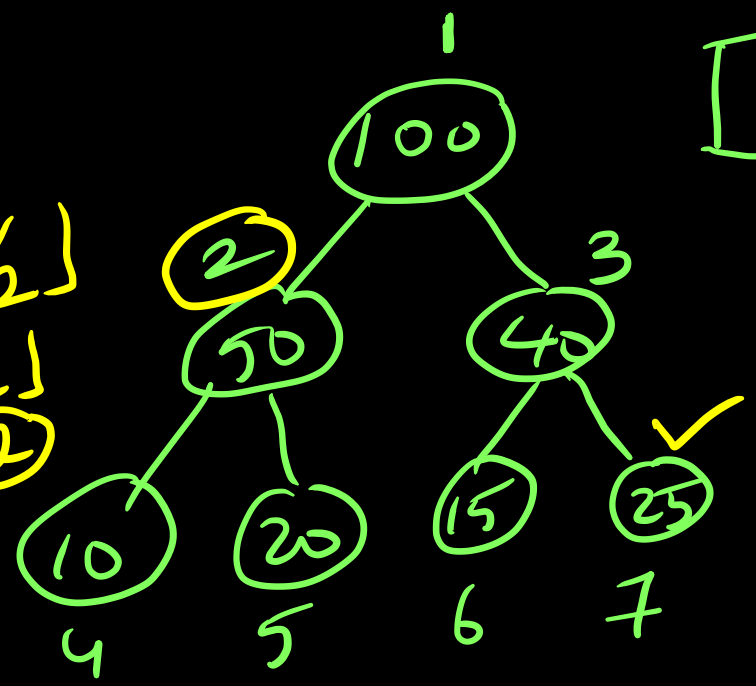
Ex:



max heap property: Every node is greater than or equal to all the nodes in its subtree.

Even though a max heap is interpreted as a tree, it is actually represented as an array





Parent (20)

Parent(5) = $\lfloor i/2 \rfloor$

= $\lfloor 5/2 \rfloor$

= 2

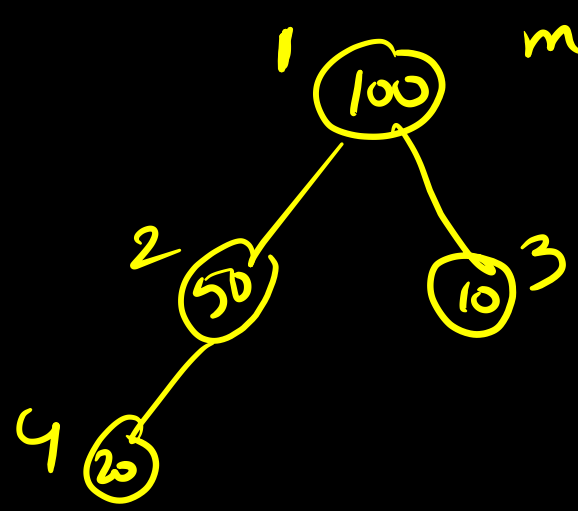
LC(1) = 2

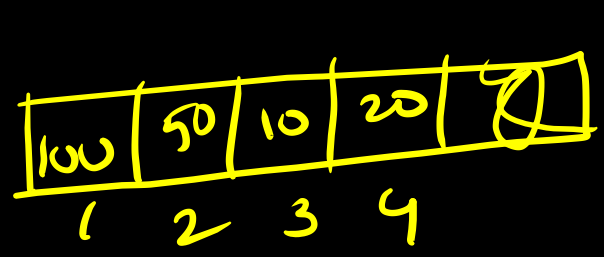
RC(1) = 3

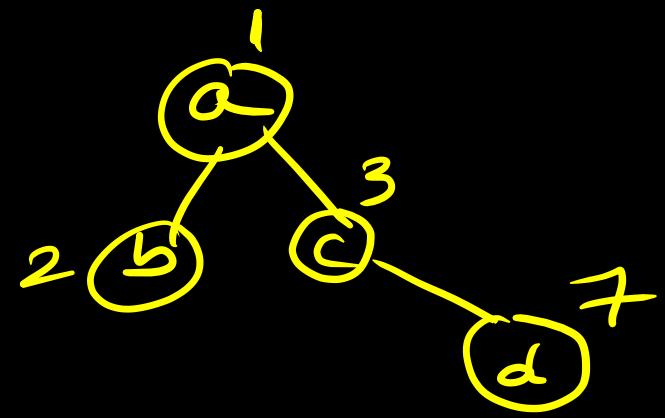
RC(3) = 2×3+1 = 7

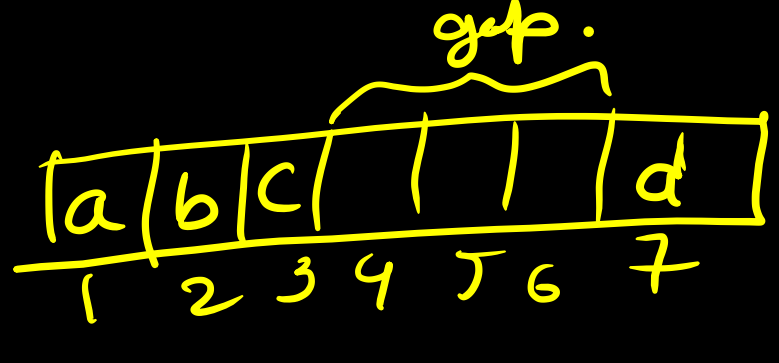
- LC(i) = $2i$
  RC(i) = $2i+1$
  Parent(i) = $\lfloor i/2 \rfloor$

Adv of almost Complete binary tree is, if we represent in array, there will be no gaps.



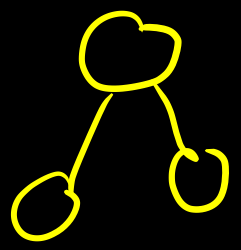


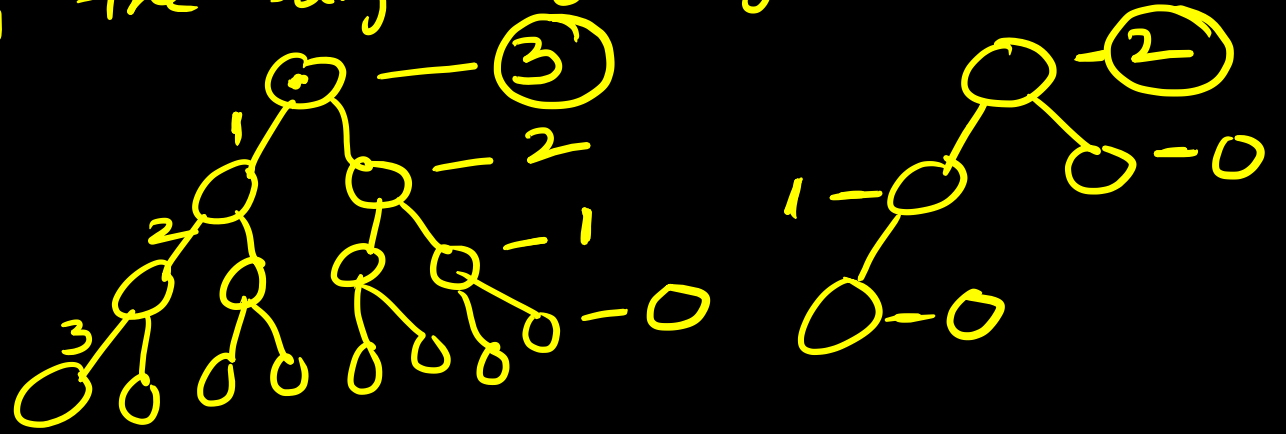




Heap is a Complete & almost Complete BT which follows heap property.

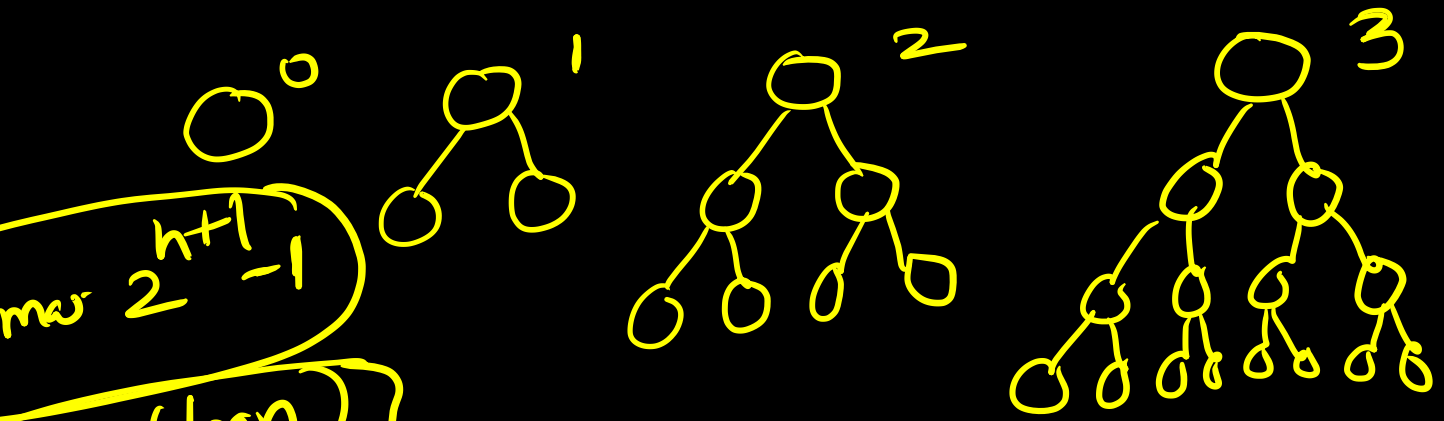
Height of a node is the height of longest number of edges to a leaf.





Height (tree) = Height (Root)

If height of a Complete BT is h, then max no of nodes in the tree is: $2^{h+1}-1$



5 nodes

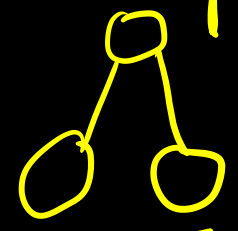
$h = \max 2^{h+1}-1$

$n = O(\log n)$

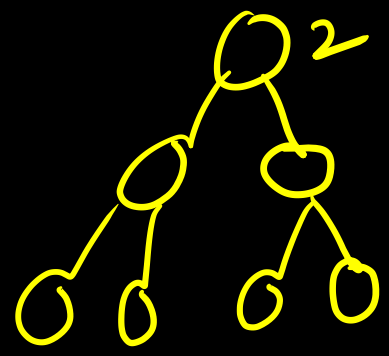
H of Complete BT	1	2	3	4
max no of nodes	3	7	15	31

$2^{h+1}-1$

If there are n nodes in an almost complete binary tree, then height of the tree is $\lfloor \log_2 n \rfloor$



$\lfloor \log_2 3 \rfloor = 1$



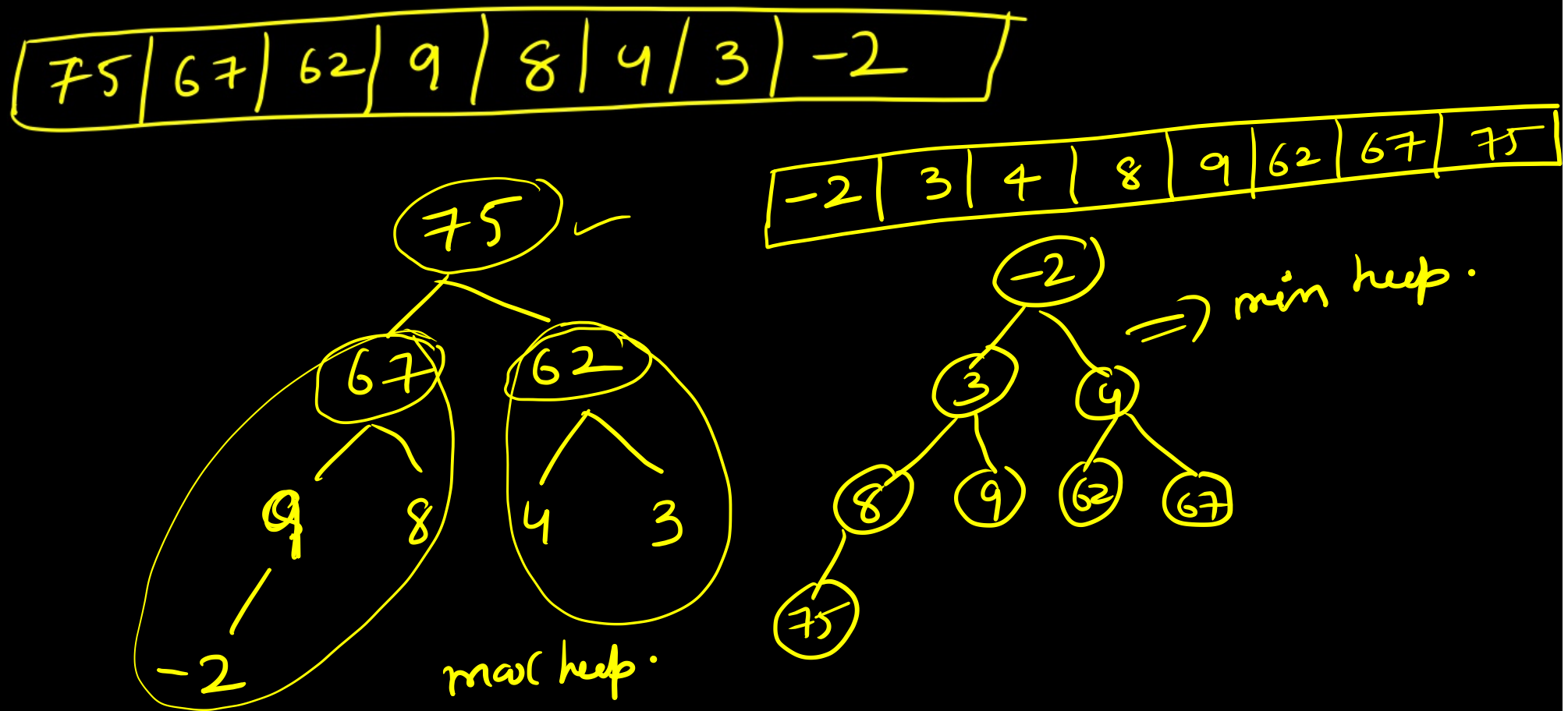
$\lfloor \log_2 7 \rfloor = 2$

In a almost complete / complete BT height is $O(\log n)$

9	8	75	3	4	67	-2	62

max heap ⇒
Sort it in decending order to get max heap.

75	67	62	9	8	4	3	-2

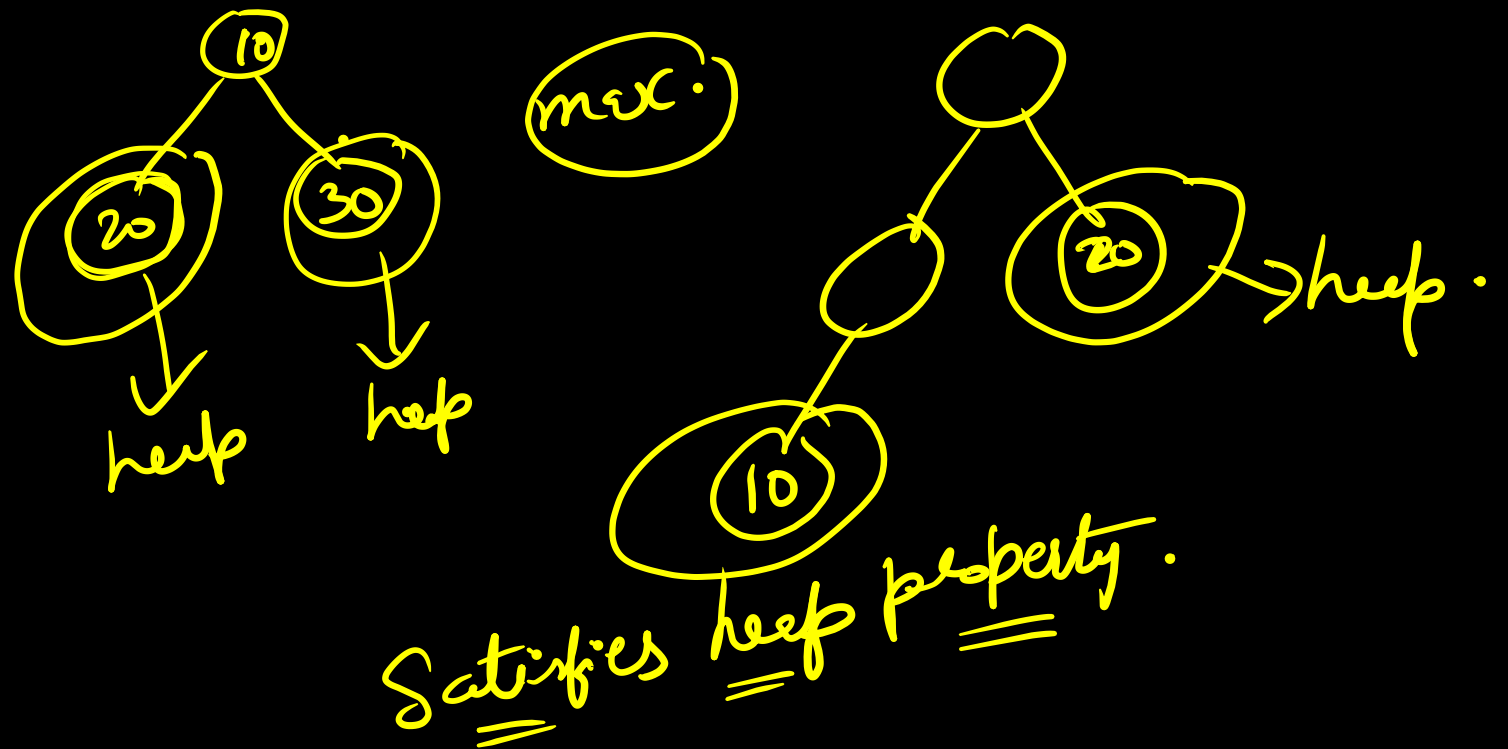

75 | 67 | 62 | 9 | 8 | 4 | 3 | -2
75
67
62
9
8
4
3
-2
max heap.
-2 | 3 | 4 | 8 | 9 | 62 | 67 | 75
-2
3
4
8
9
62
67
75
⇒ min heap.

From a given array we can get a heap by sorting it.

But it takes $O(n \log n)$ time.

So there is another way to build a heap in $O(n)$ time.

We need to make an observation, that every leaf is a heap.

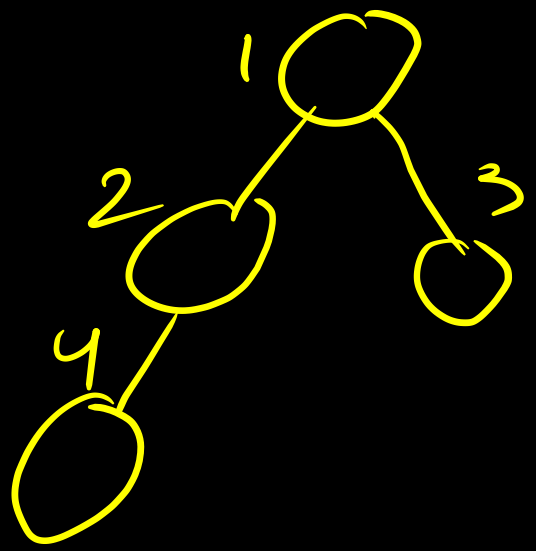


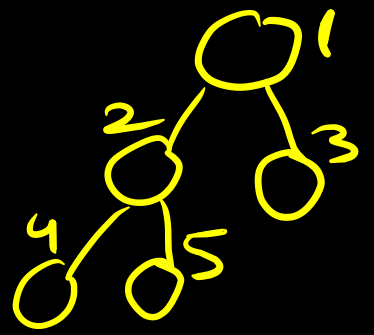
All leafs are already heaps, so we have to focus on non leafs.

We need to know where leafs starts and non leaves begin

1 2 non leaves

3, 4 leaves





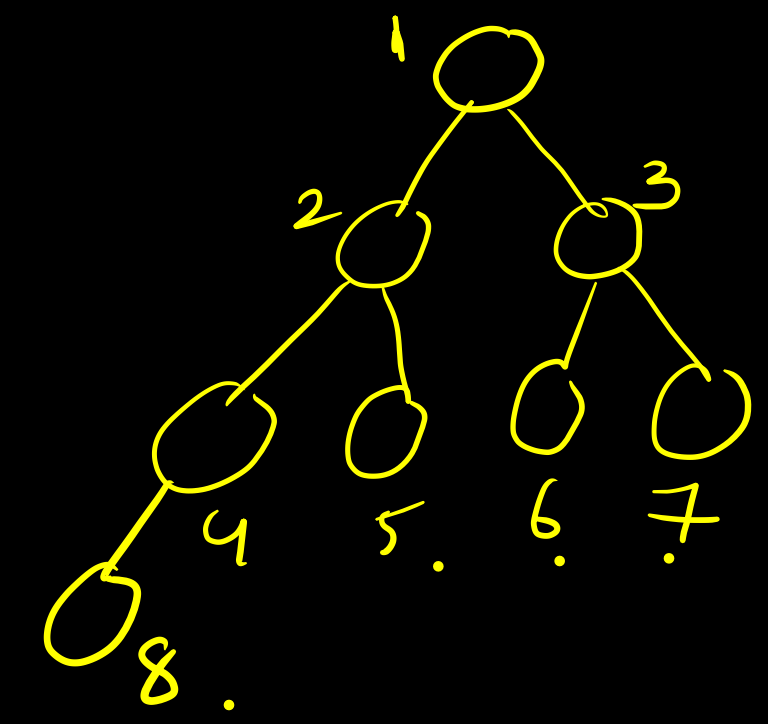
3, 4, 5 → leafs

1 2 non leafs.

In an almost complete BT,

$\left(\lfloor n/2 \rfloor + 1 \text{ to } n\right)$ are leafs

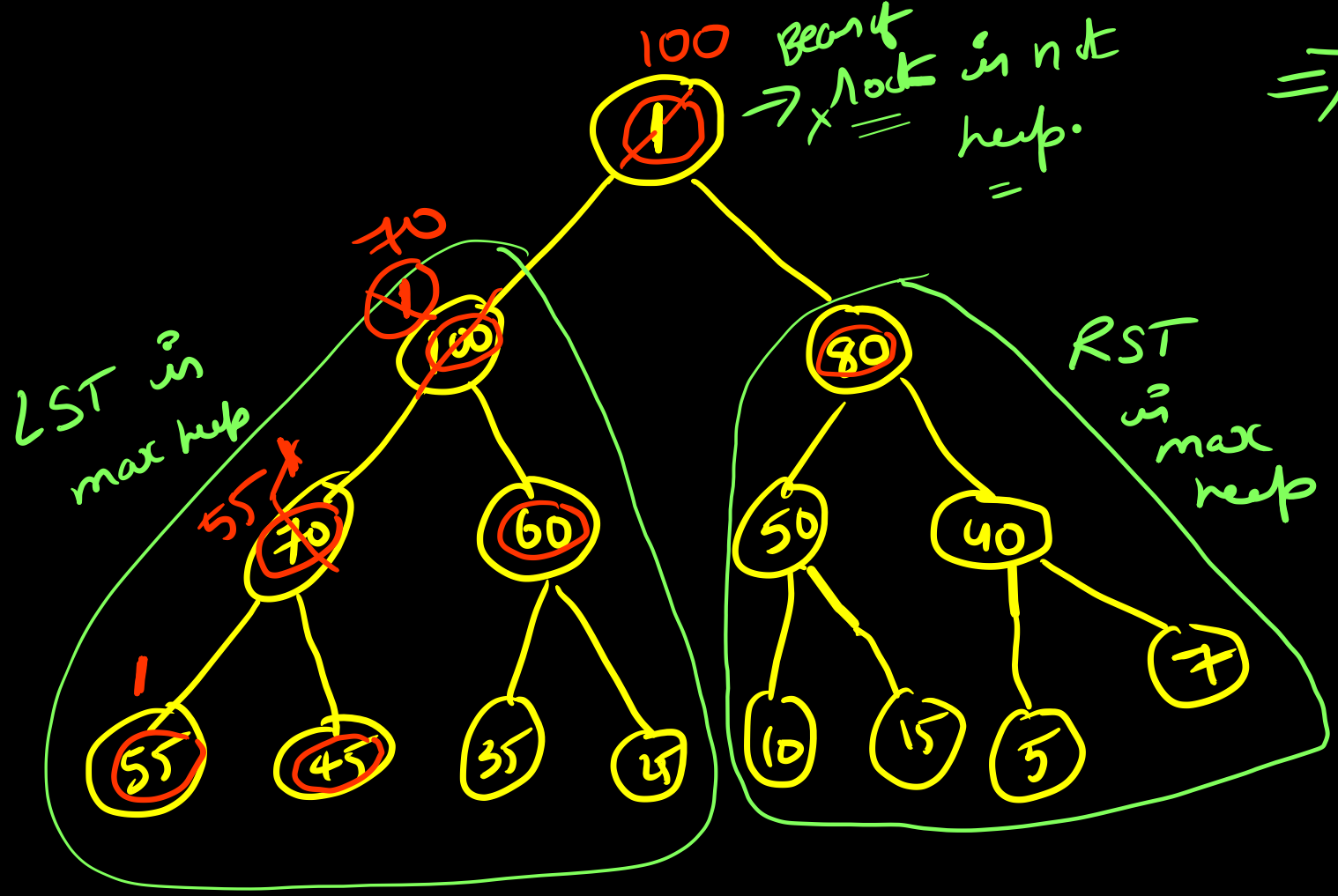
$\left(1 \text{ to } \lfloor \frac{n}{2} \rfloor\right)$ non leafs.



leafs $\rightarrow \lfloor \frac{n}{2} \rfloor + 1$ to n

5 to 8

non leafs $\rightarrow$ 1 to 4



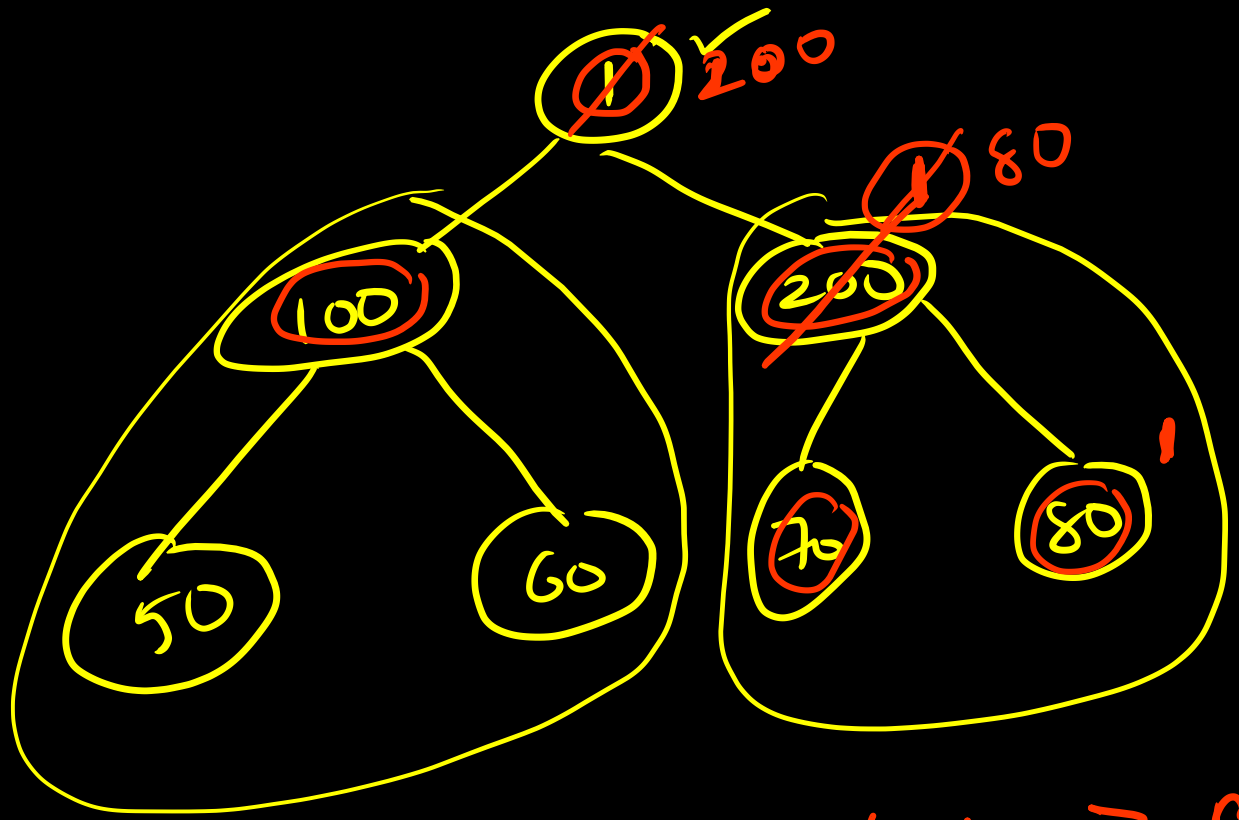
$\Rightarrow$ max-heapify.

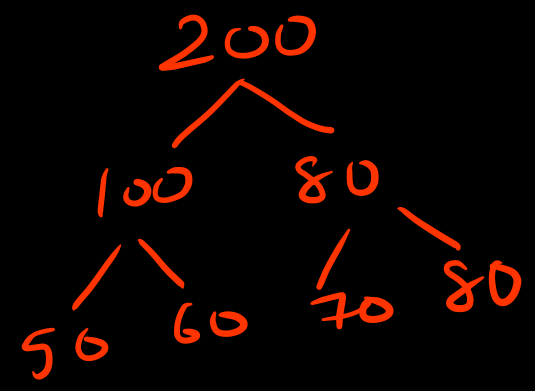
LST $\rightarrow$ max

RST $\rightarrow$ max.

Because of Root it is not a Heap. then we use max-heapify to make it a heap

max heapify

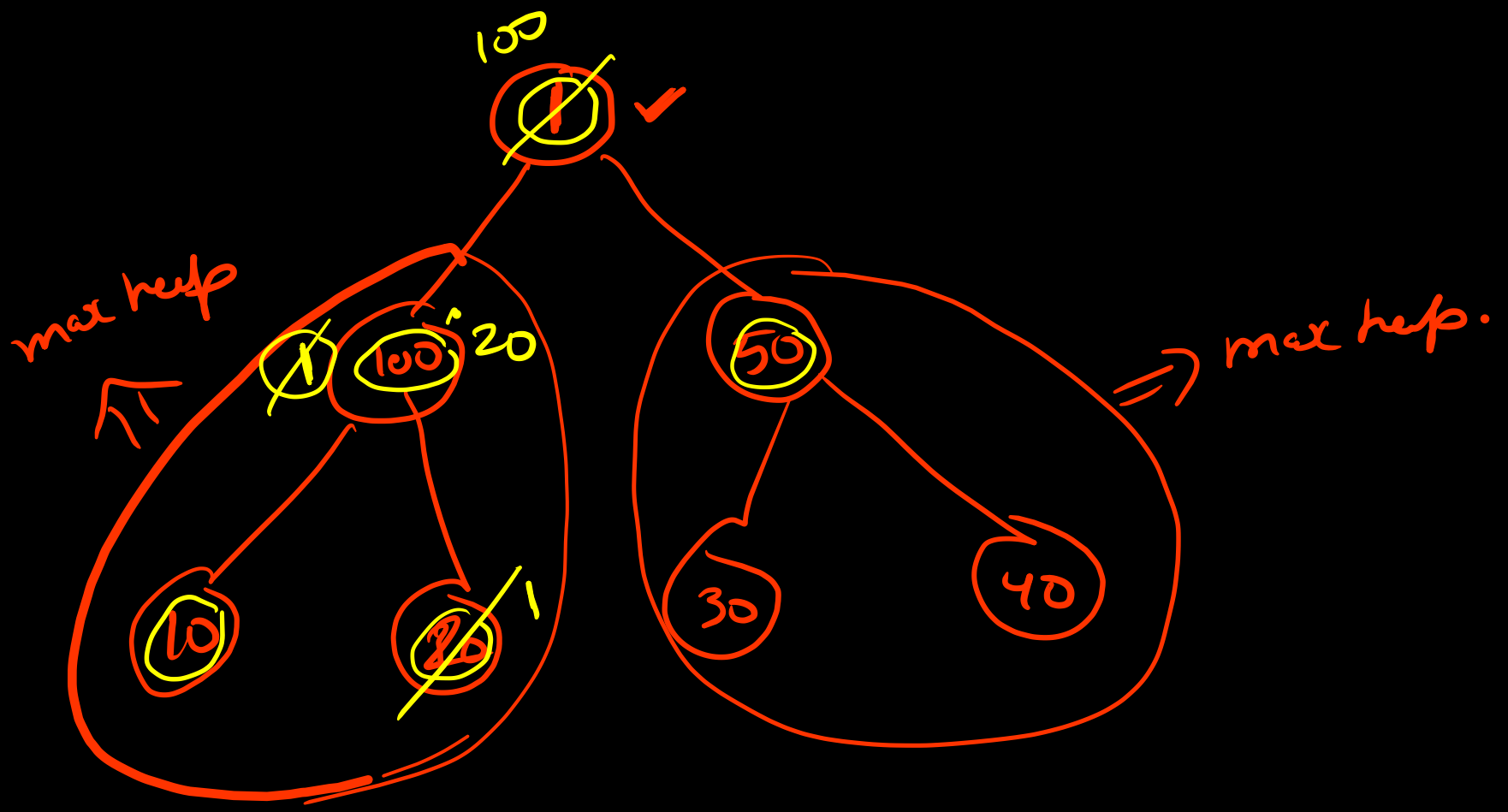




MS → merge

quick → partition

heaps → max & min heapify.


100
1
max heap
1
100
20
50
max heap.
10
20
1
30
40