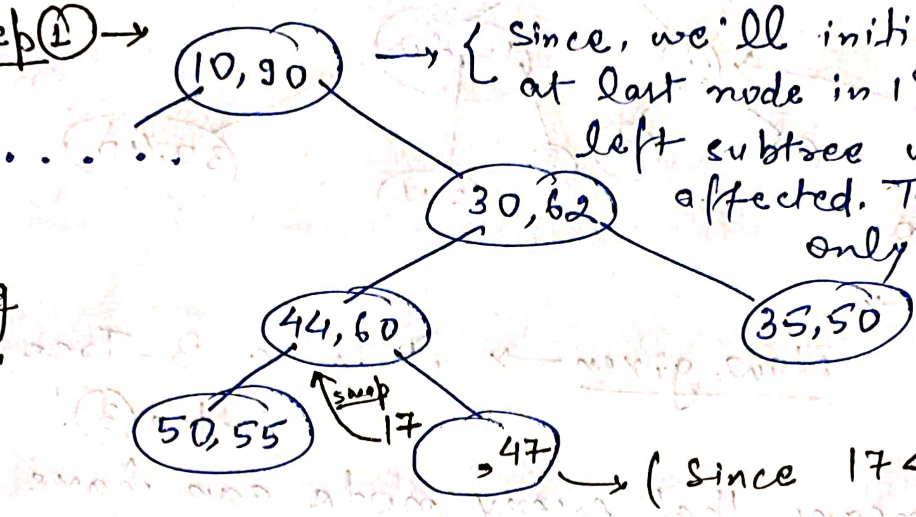
→ DOR → 2021PCS1017

→ AMBUJ MISHRA

Ans ②. → step ① →



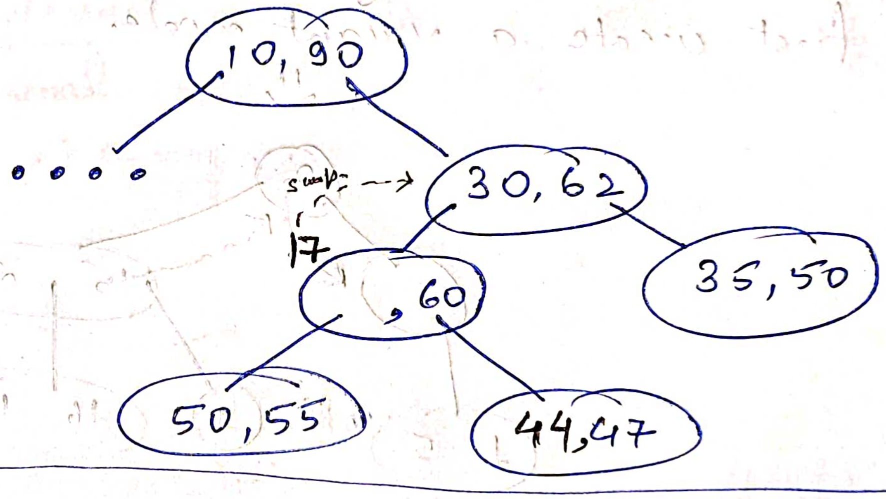
{ Since, we'll initially insert at last node in 1st step. So, left subtree will not be affected. Thus showing only right subtree }

inserting ⑰ →

→ { Since 17 < 47, so, ⑰ is inserted in the min heap & '47' will move to max heap }.

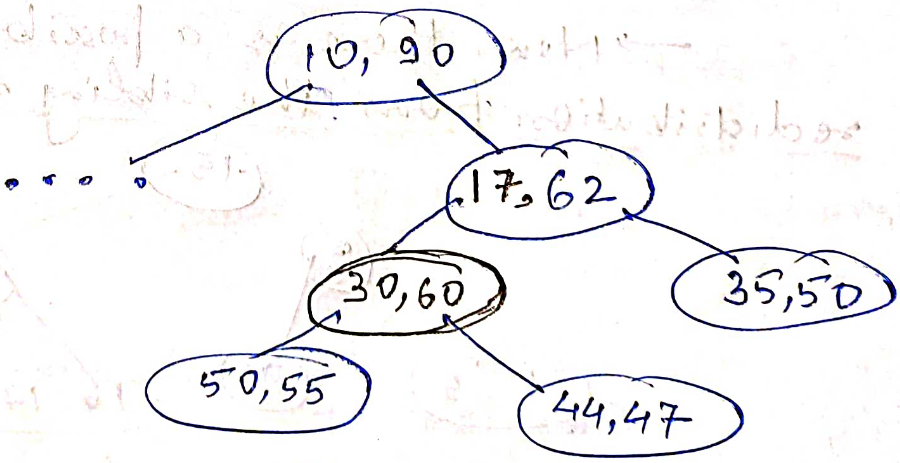
step ② →

Since (17 < 44.), so, swap →



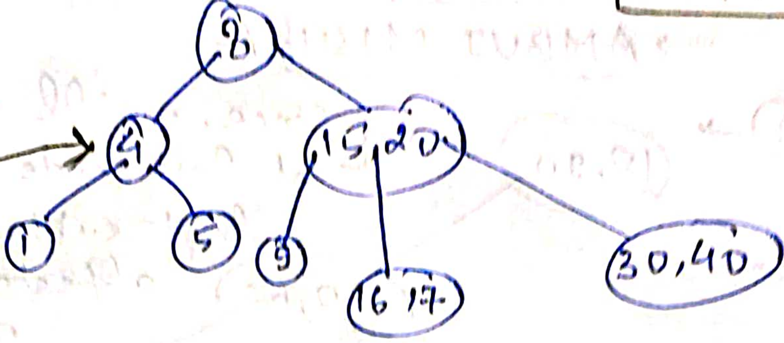
step ③ →

Since (17 < 30), so swap →



⇒ No, further swapping is done as ⑰ has reached it's optimum position.

Ans. 3 →



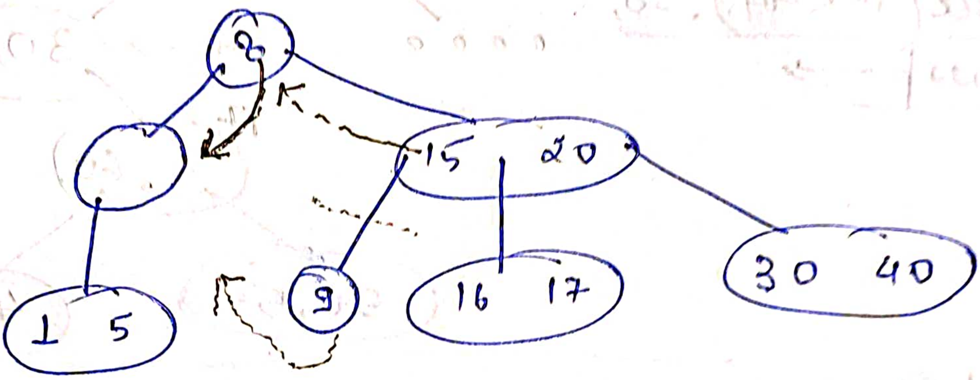
we have to delete 4 →

Now, given → it is a B-Tree of order 3.

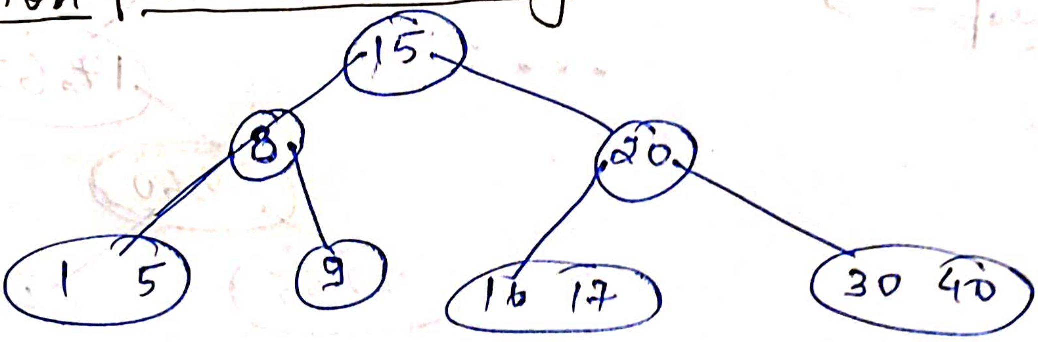
So, it means that every node can have only 1, or 2 pair inside it.

→ Deleting 4 from an internal would first create a vacant node.

⇓



→ Now, there is a possibility for redistribution from the sibling node. So →



→ it is final Tree after Deleting 4.

2021PCS1017

Que. 4 → (i). Contiguous → Arrays are contiguous data structures, in which continuous memory is allocated.

Because of the same reason, we are able to directly access data from array.

(ii). linked → linked list follows linked data structure type. In this, memory allocation is not continuous. One memory location maybe connected to it's adjacent location in one way (or) in both ways.

(iii). indexed → indexed binary search trees, as it keeps an additional index value (of no. of nodes in it's left subtree) along with each node

Que 1 → T.C. to 'find' 2nd largest → ~~O(logn)~~ $\boxed{O(1)}$ ~~O(l[illegible])~~, as it keeps both min heap & max heap both. So, we'll find 2nd largest on 2nd level of max heap.

T.C. to 'remove' 2nd largest → $\boxed{O(\log n)}$: Removal in worst case we'll have to traverse through the height of the complete interval heap. In that case →

$\boxed{O(\log n)}$