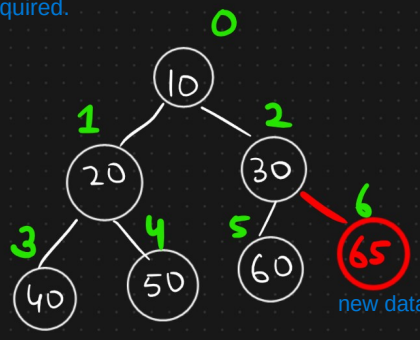
Deletion in Minheap

Insertion
- worst & average = $O(\log n)$ $\longleftrightarrow$ n elements
- Best case = $O(1)$ = constant

Best case occurs when only 1 comparison and 0 swap is required.
Eg;



$30 < 65 \Rightarrow$ valid Minheap

1 comparison & 0 swap

Question: What is time complexity for inserting an element in Min heap in case we are provided with n*2^n elements?

Ans; If not provided we will consider that the time complexity is asked for the worst or average case scenario which is equivalent to the number of levels. that is;
for n elements number of levels is given as;
k=log(n)base2 where we will get the TC of order O(n)

for n*2^n elements number levels will be
k = log(n*2^n)base2 which is order of O(n). See elaboration on he right hand side

$$\# \text{elements} = n \times 2^n \quad (\text{Insertion})$$

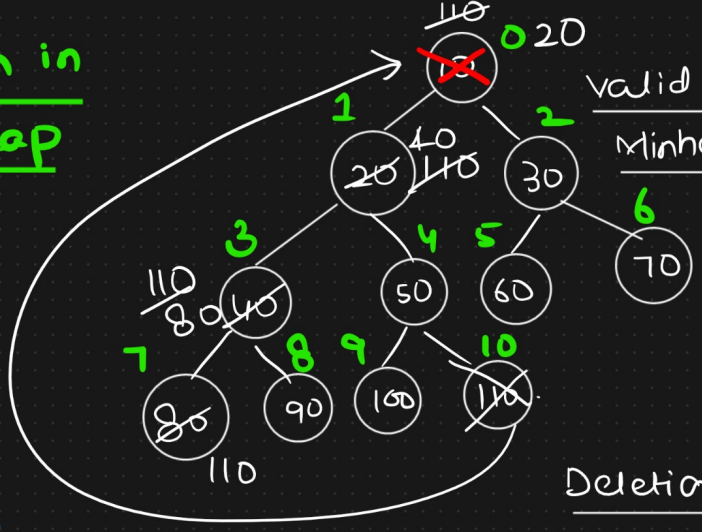
$$= \log_2(n \times 2^n)$$

$$= \log_2 n + \log_2 2^n \qquad (n * \log_2 2 = n * 1)$$

$$= \log_2 n + n$$

$$= O(n) \Longleftarrow$$ (Since n is greater then log n)

Deletion in minheap



Valid Minheap { 1) CBT 2) Parent < child node data data

⇒ 10

Delete

Deletion in MinHeap happens in the following 2 steps:
1. Deletion happens at the root level. Therefore, element at the 0th index will be Popped out.
2. Element at the last index of the MinHeap tree will be moved to the 0th index followed by performing continuous comparison and swapping until MinHeap property is again restored in case it gets destroyed

Deletion → # comparisons

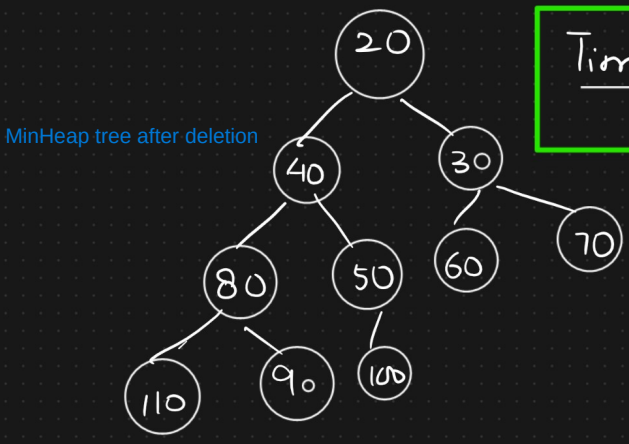
↳ 2 × logN

no. of levels

Root Node to Leaf Node { Left child & Right child

swaps → logN

Swaps will happen in each level and either left child or right child will be involved



MinHeap tree after deletion

Time complexity → O(logN)

comparison based sorting ↑ Heapsort

If we perform continuous deletion and store the respective deleted element in a new Array then we will get the sorted array that will be containing all the elements of the MinHeap tree in the sorted fashion. This is called Heap Sort which is comparison based sorting algorithm

Deletion - 1st time ⟶ 1st smallest ⌐logN

2nd time ⟶ 2nd smallest ⌐logN

3rd time ⟶ 3rd smallest

⋮

n time ⟶ nth smallest ⌐logN

elemen-

Array ⇓ Sorted array ⇓ increasing order

Overall time complexity = O(nlogn) (Since n times deletion is performed)

Maxheap → Sorted array
Deletion
n number
of times

→ Decreasing
order

Heapsort → $O(n \log n)$