heap sort

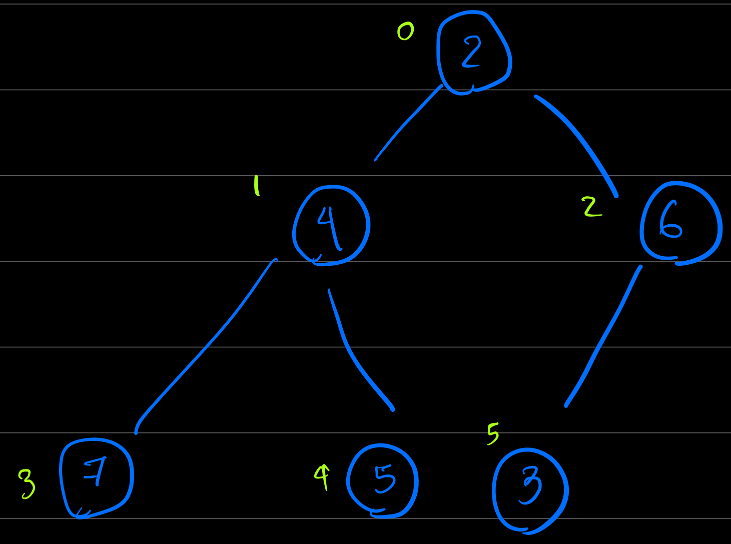
- It works by using a data structure called "Complete binary trees."

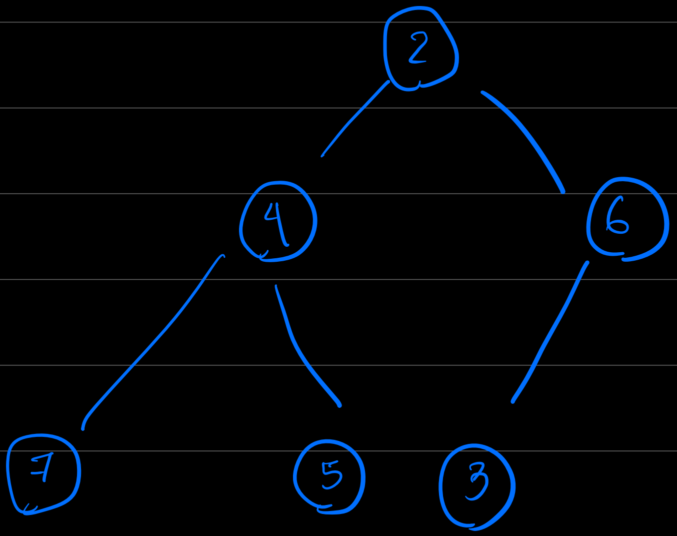
- It is a "recursive function".

Suppose you have a list

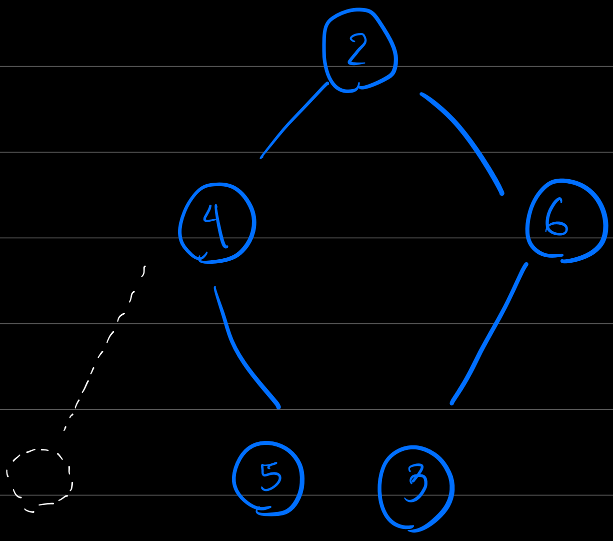
0	1	2	3	4	5
2	4	6	7	5	3

Turn that into complete binary tree.



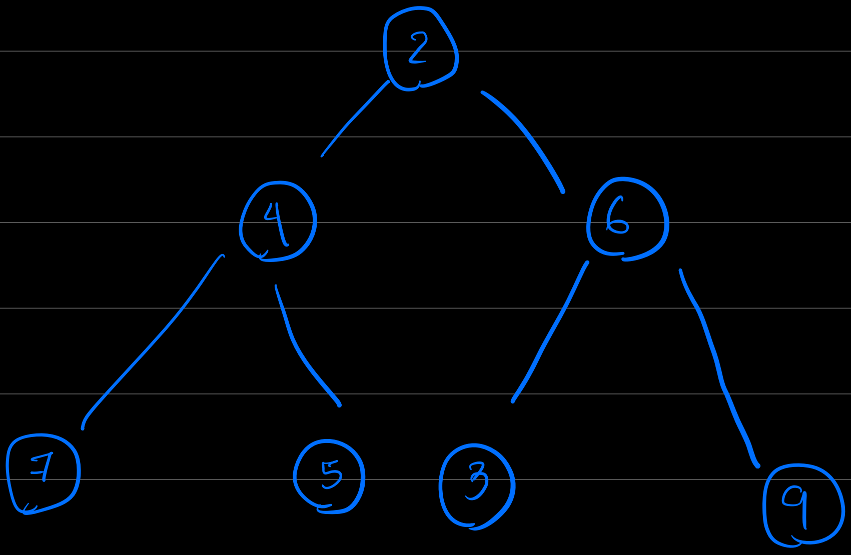


This is a complete binary tree it goes from top to bottom, left to right.



This is a binary tree but not "complete"

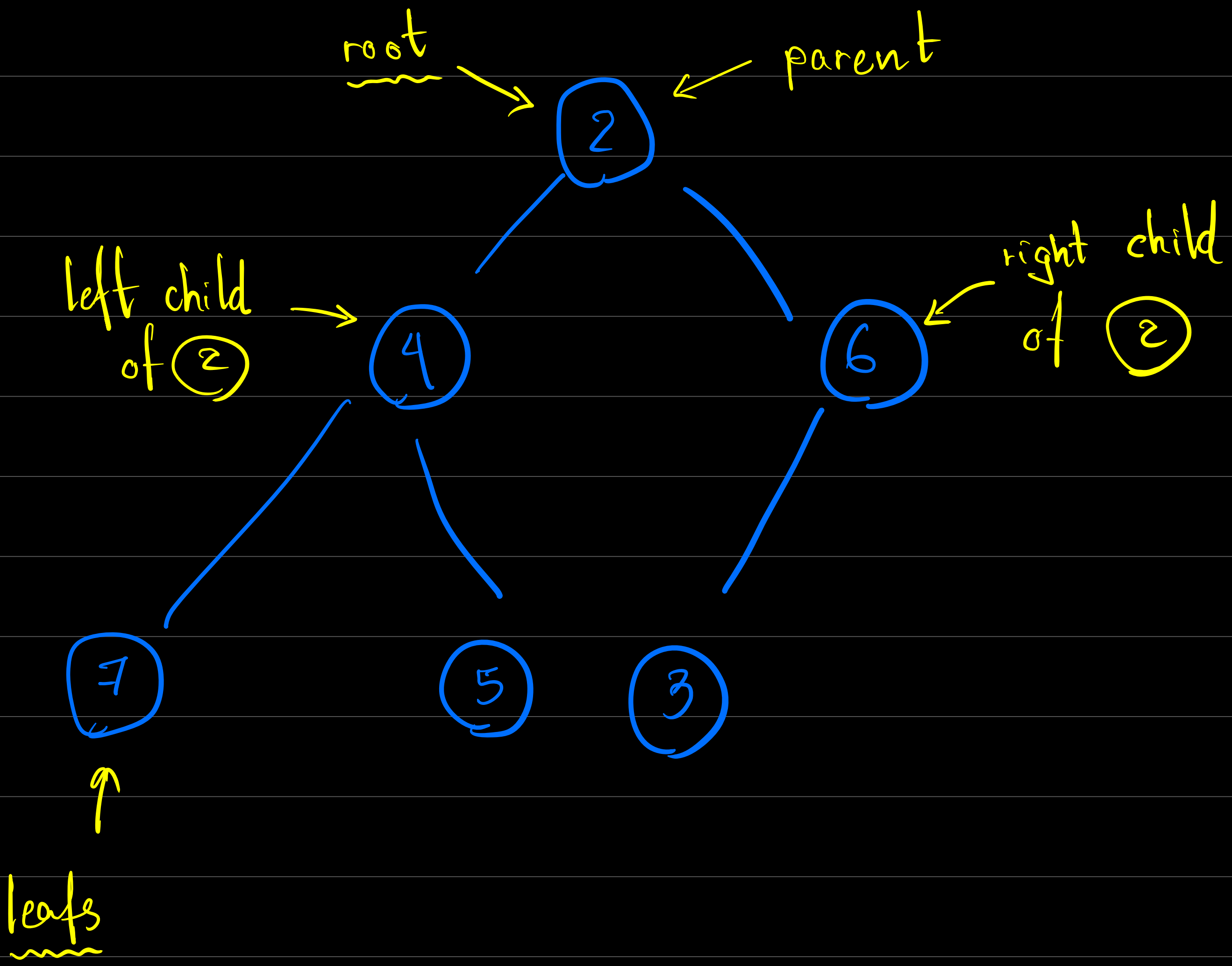
it has blank element at index 3.



This is a "full binary tree" in oder to add an element, you must add a new line or "height" of the tree.

Heap sort works with complete binary trees.

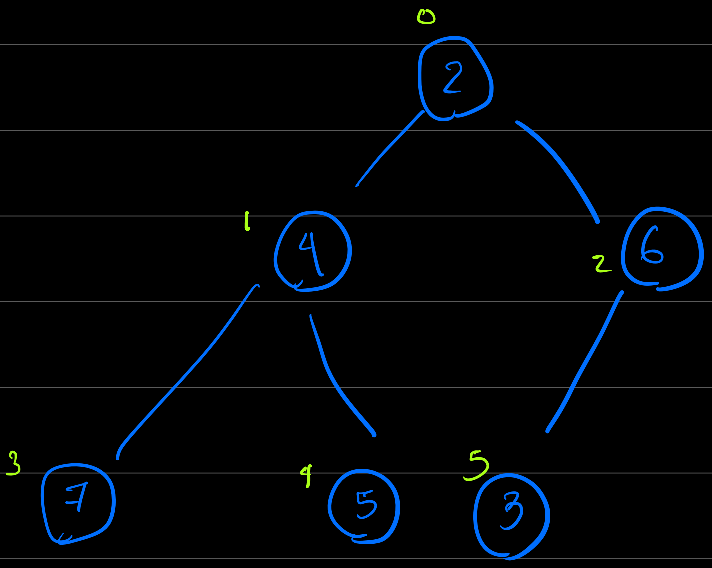
Glossary

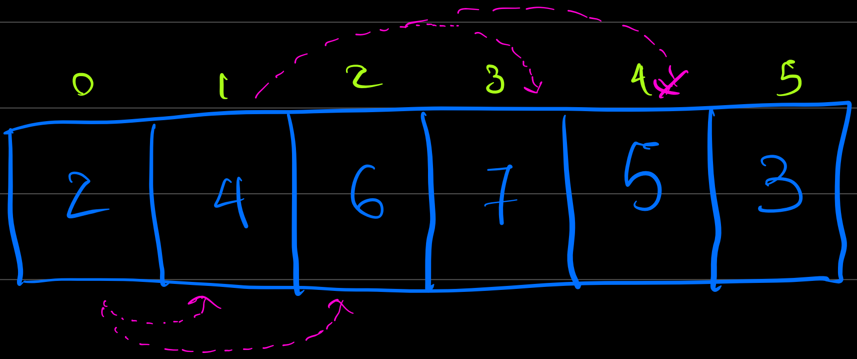


Min Heap - Parent must have lower value than child.

Max Heap - Parent must have high value than child.

Indexing





0	1	2	3	4	5
2	4	6	7	5	3

left child = $i * 2 + 1$

right child = $i * 2 + 2$

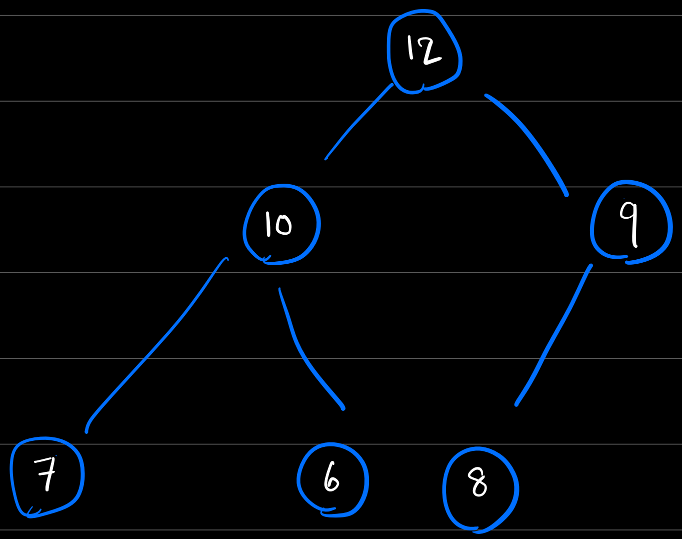
parent = $i // 2 - 1$

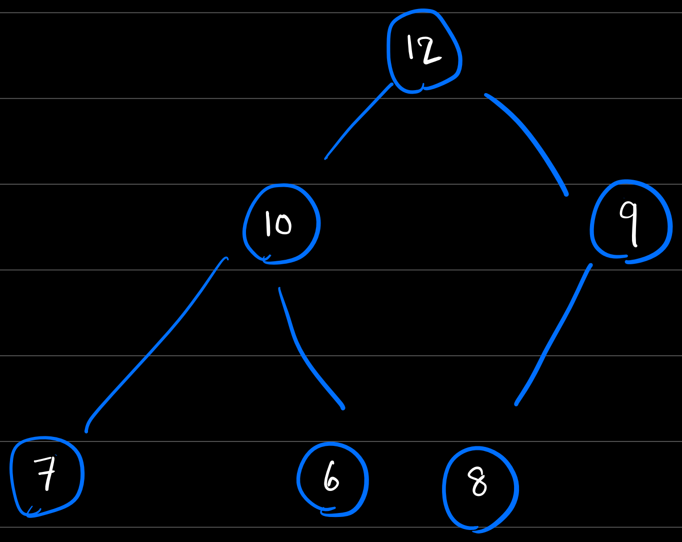
The Actual sorting

Let's go with max heap as an example....

The rule is the parent must always be bigger than the child.

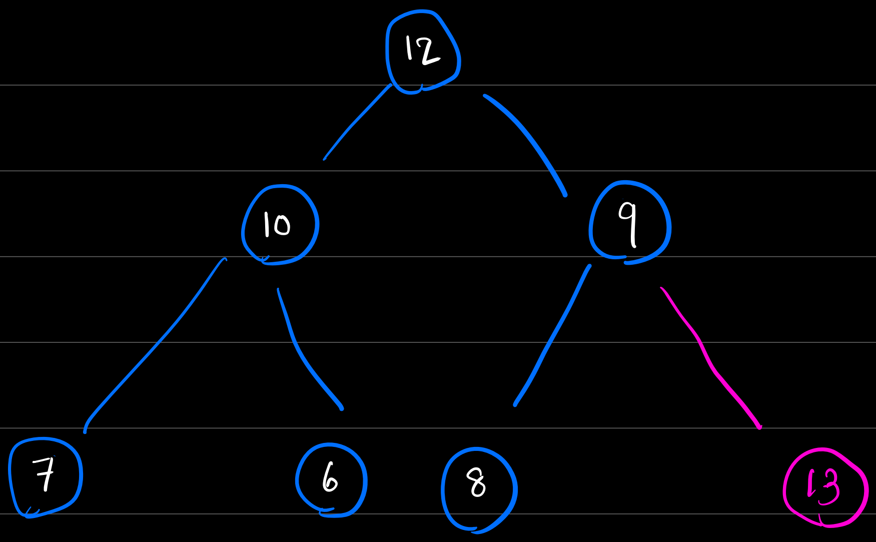
Looks something like this.





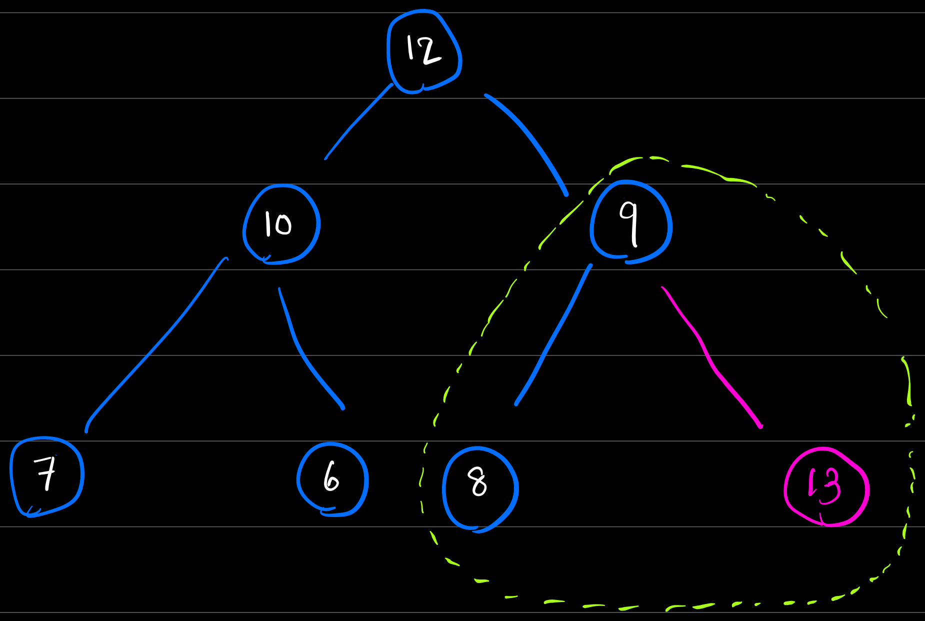
Let's say we want to add "13" in it.

We first insert it to the next index.



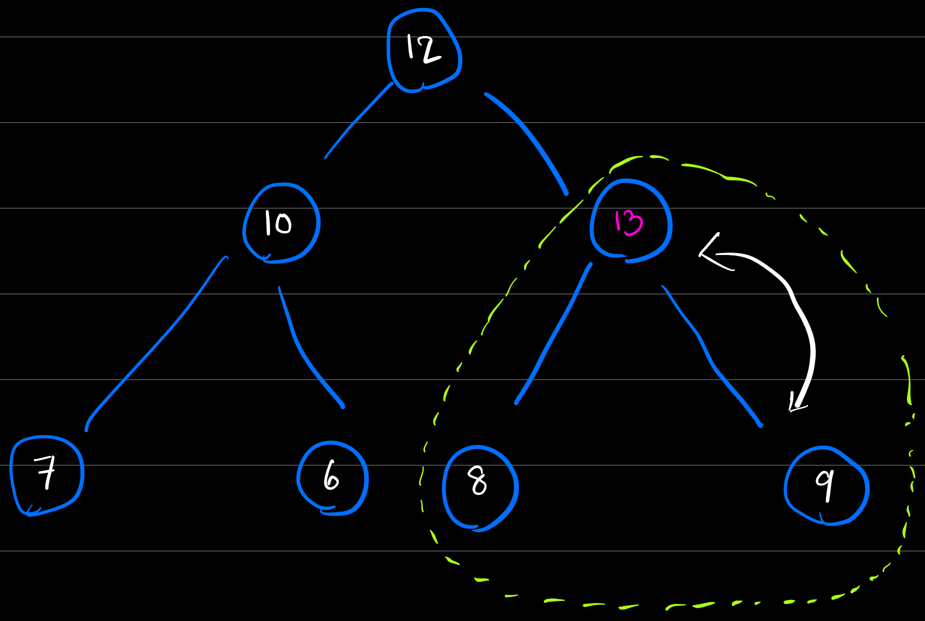
Max heap rule is not satisfy & we need to fix

Step by step.

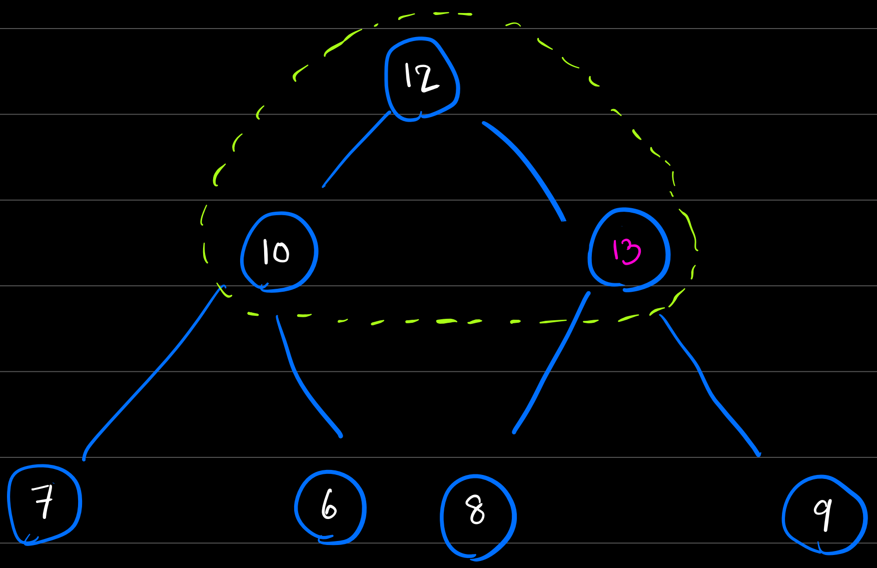


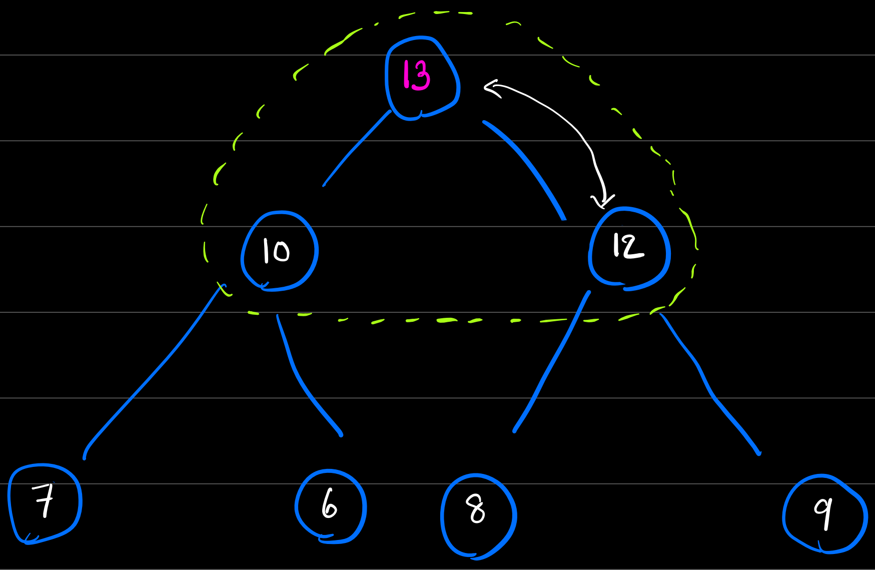
Out of those 3, the largest value must be swapped to parent index.

STEP 1



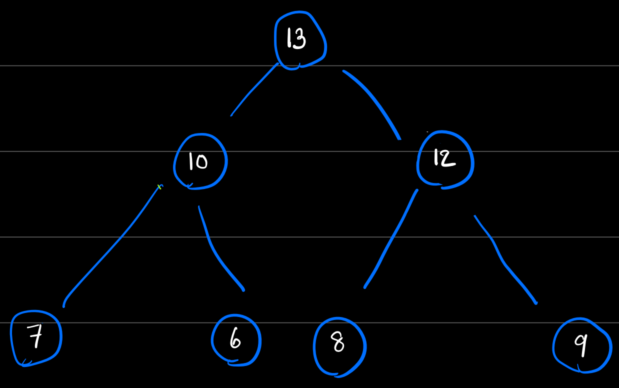
STEP - 2



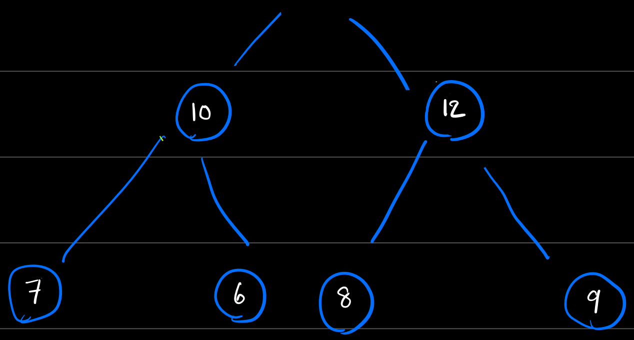


It took two steps which is worst case scenario and it is the "height of the binary tree"

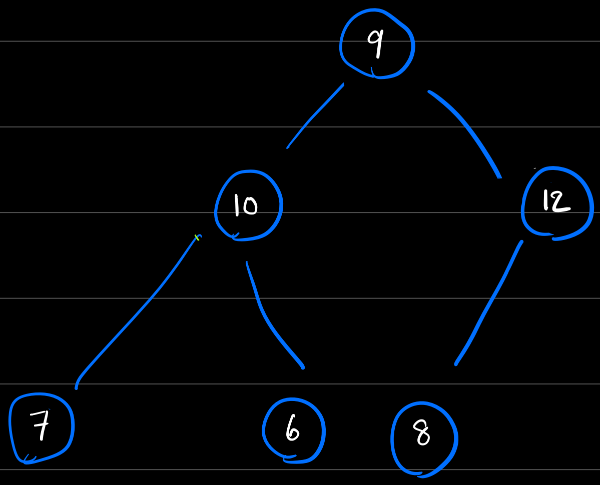
Delete

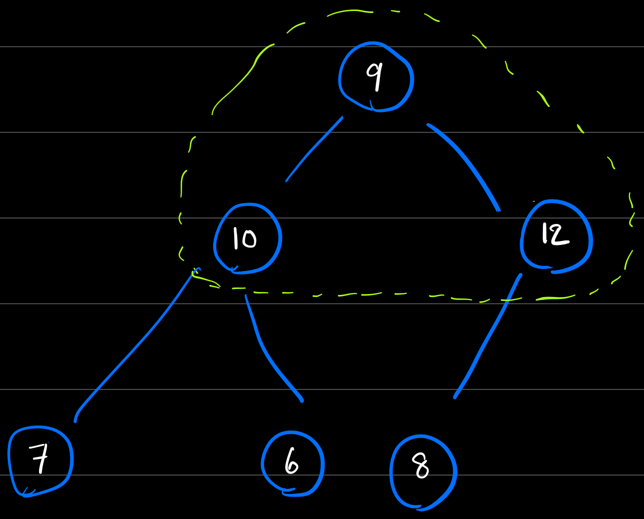


in heapsort, we always delete the "root" first.

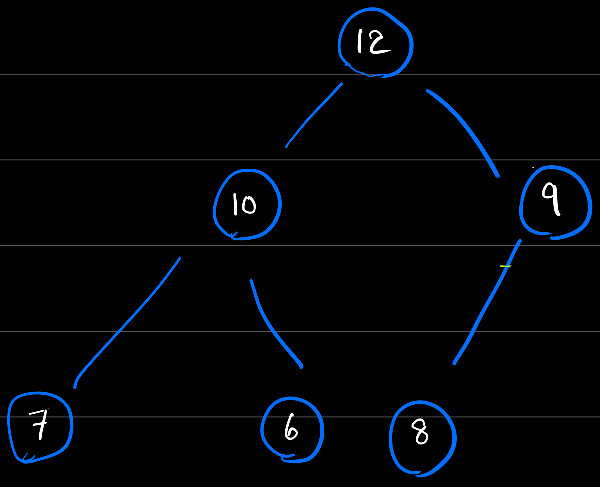


The element at " last index " takes its place.





Then fix the value from root to leaves.

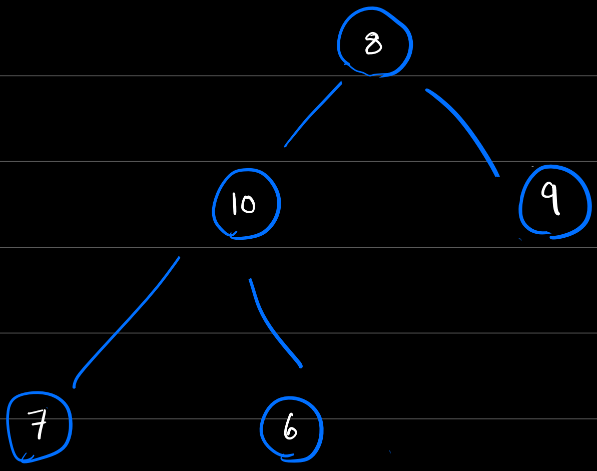


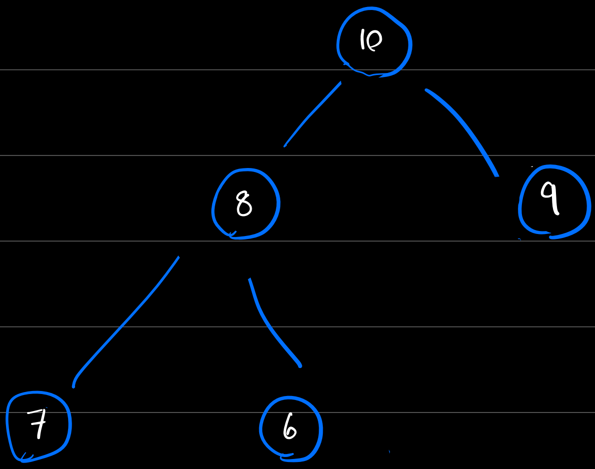
Put that deleted element "13" outside.

So the list works like that

12	10	9	7	6	8	13

13 is sorted & outside of heap.





10	8	9	7	6	13	12

sorted.

After all deletion are done, the list end up like this

13	12	10	9	8	7	6

Delete takes "height time".

heapify

" Same procedure as deletion, direction is opposit

Starts from "last element" & looks downward. If no chid, it's already a heap.

Time complexity

Best case (parent already in the right spot) = O(1)

Worst case (height of tree steps) = O(nlog n)

Avg case = O(n log n)

Good to know

- heapsort is efficient, accurate, highly consistent with low RAM usage than others like quicksort.
- Ideal to use in priority queues, elements in heapstructures are ppl waiting in queue with the stem being the highest priority. Deleting the stem means Serving that person and putting it in "served list".
- Time complexity is noted as $O(n\log(n))$ but actual time taken is less since it takes less time as the process progresses. Remember, Big O ignore constant values line $(n-1)(n-2)\dots\dots(1)$