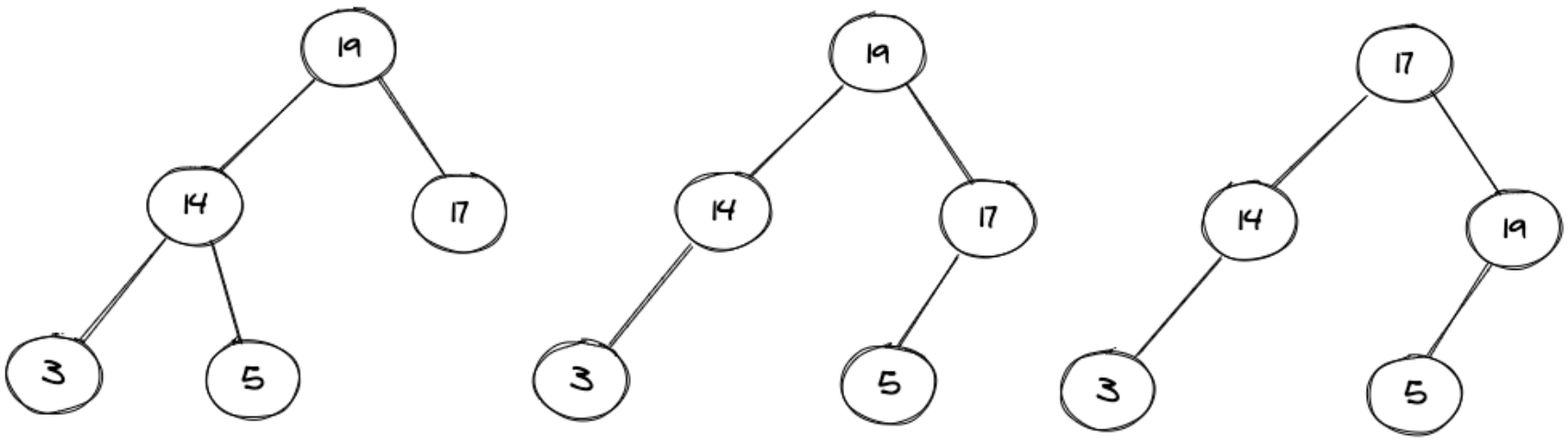
# Heap Data Structure

A Heap is a special type of tree that follows two properties. These properties are :

- All leaves must be at `h` or `h-1` levels for some `h > 0`(complete binary tree property).
- The value of the node must be `>=` (or `<=`) the values of its children nodes, known as the heap property.

Consider the pictorial representation shown below:




In the pictures shown above, the leftmost tree denotes a heap (Max Heap) and the two tree to its right aren't heap as the middle tree violates the first heap property(not a complete binary tree) and the last tree from the left violates the second heap property(17 < 19).

## Types of Heap

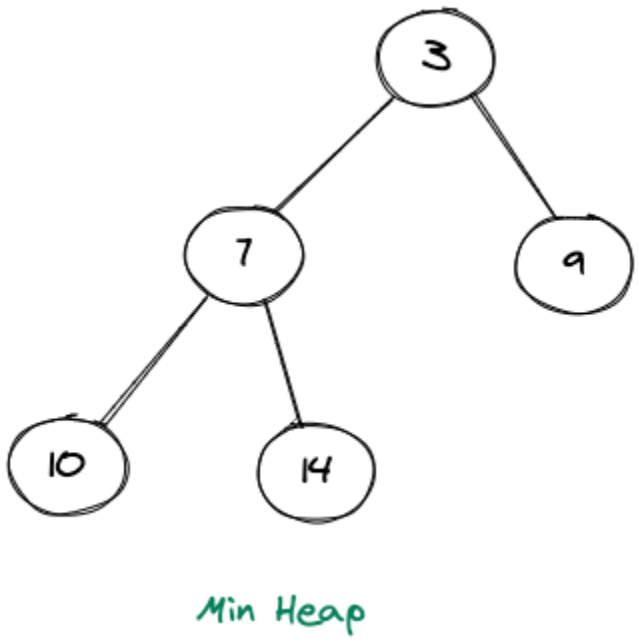
If we consider the properties of a heap, then we can have two types of heaps. These mainly are:

- Min Heap
- Max Heap

### Min Heap

In this heap, the value of a node must be less than or equal to the values of its children nodes.

Consider the pictorial representation of a Min Heap below:

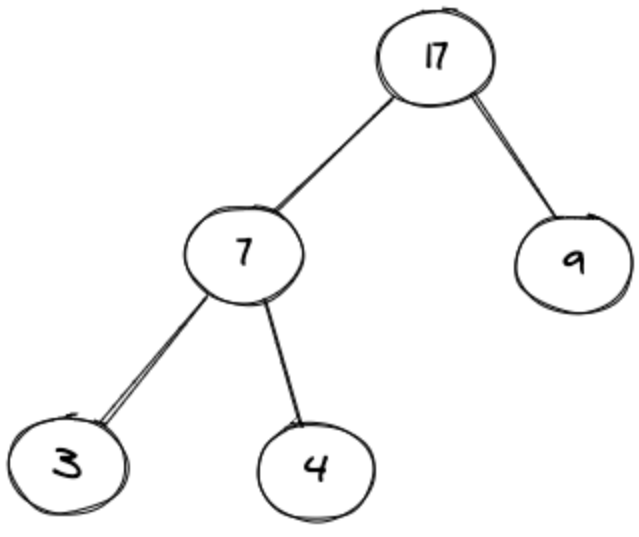



It can be clearly seen that the value of any node in the above heap is always less than the value of its children nodes.

### Max Heap

In this heap, the value of a node must be greater than or equal to the values of its children nodes.

Consider the pictorial representation of a Max Heap below:




Max Heap

It can be clearly seen that the value of any node in the above heap is always greater than the value of its children nodes.

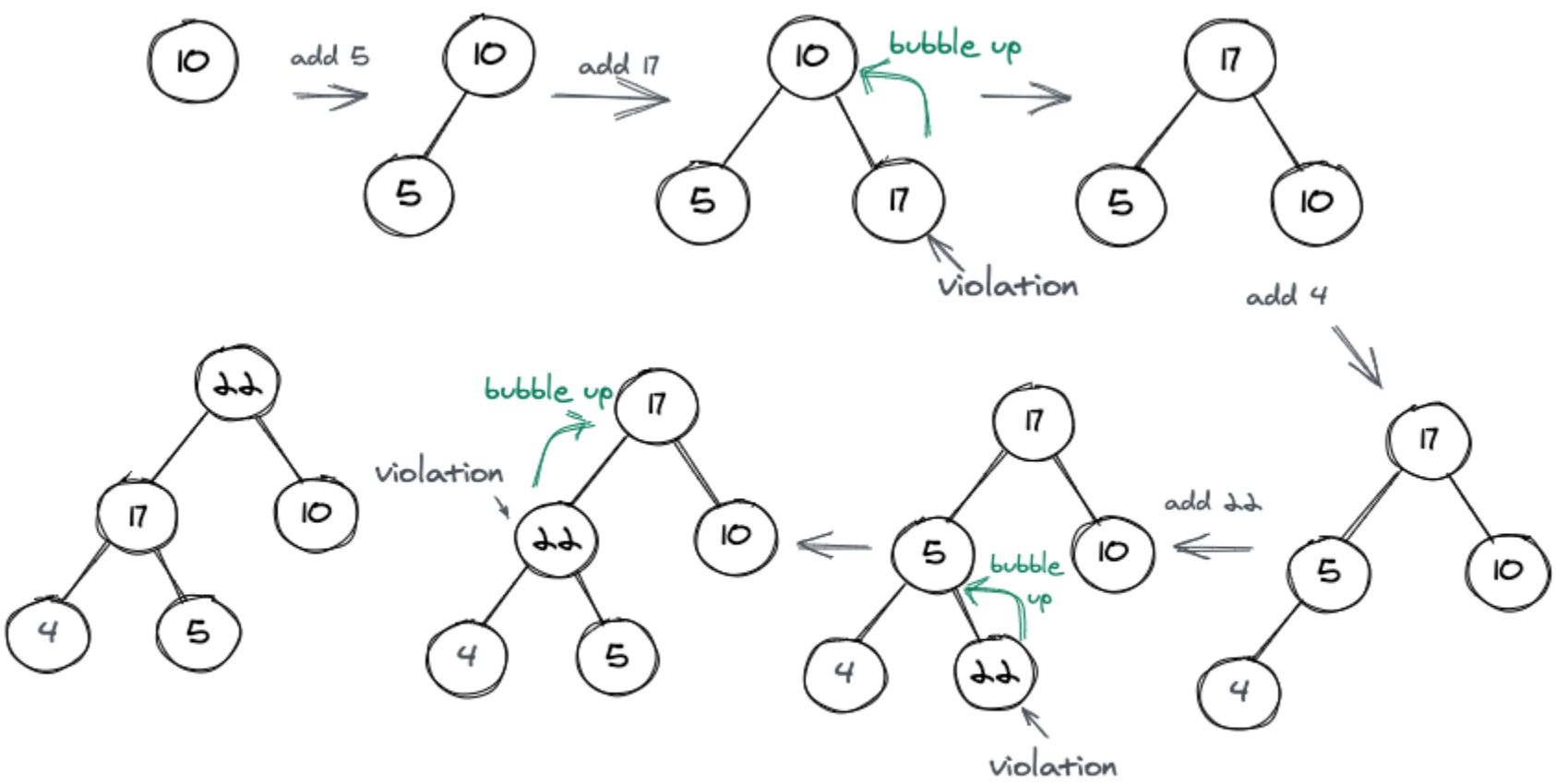
## Building a Heap

Let's look at some operations like inserting an element in a heap or deleting an element from a heap.

### 1. Inserting an Element :

- First increase the heap size by 1.
- Then insert the new element at the available position(leftmost position ? Last level).
- Heapify the element from the bottom to the top(bubble up).

Building a heap includes adding elements into the heap. Let us consider an array of elements, namely nums = [10,5,17,4,22]. We want to make a Max Heap out of these elements, and the way we do that is shown in the pictorial representation below.

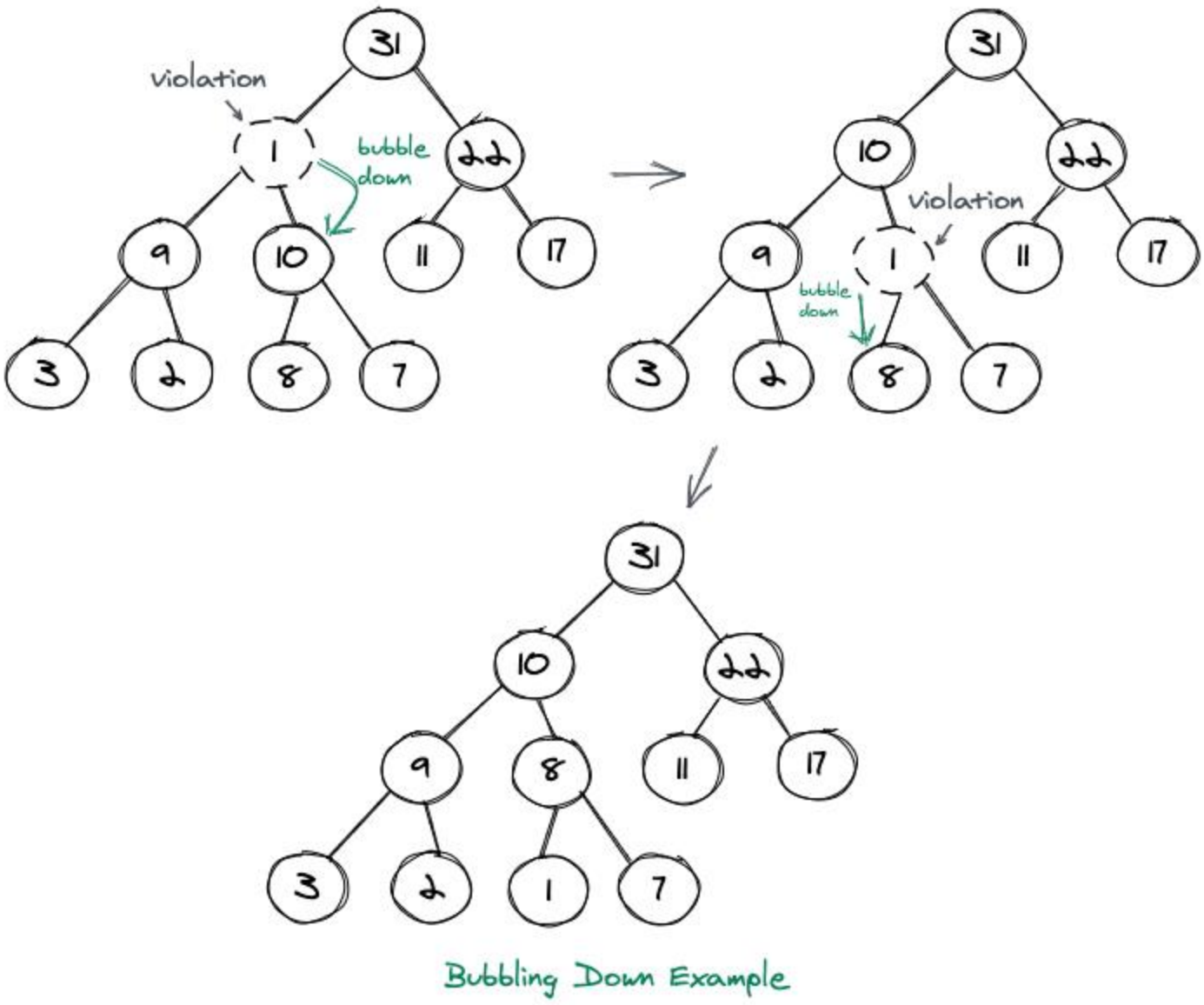



Building a Heap

Whenever we add an element while building a heap, it is quite possible that it will violate one of the properties. We take care of the first heap property by simply filling each level with maximum nodes and then when we move on to the next level, we fill the left child first and then the right child. But it is still possible that we have a violation regarding the second heap property, in that case we simply bubble up the current element that we have inserted and try to find the right position of it in the heap. Bubbling up includes swapping the current element with its parent till the heap is a max heap. In case of a min heap, we do the same procedure while adding an element to the heap.

The above representation shows three violations in total(17, 22, 22) and in all these cases we have basically swapped the current node with the parent node, hence bubbling up. It can also be noted that this process of bubbling up is also known as sift up.

Now let us look at another example, where we bubble down. Consider the pictorial representation of a tree(not heap) shown below:



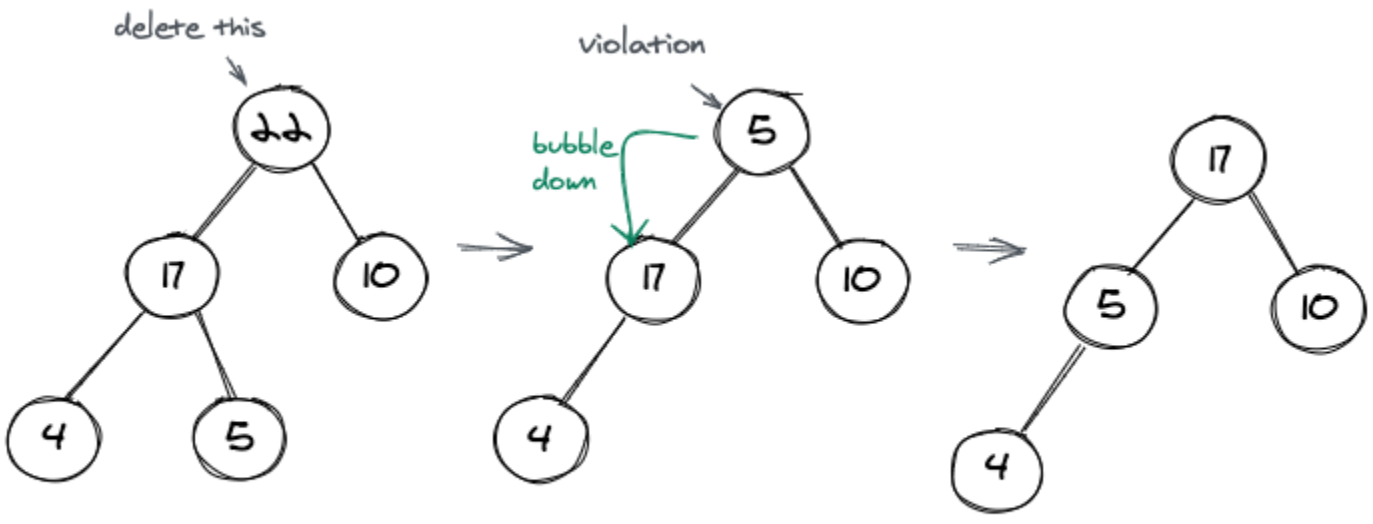

Bubbling Down Example

2. Deleting an Element :

- Copy the first element of the heap(root) into some variable
- Place the last element of the heap in the root's position
- Bubble Down to make it a valid heap

Whenever we are deleting an element, we simply delete the root element and replace it with the last element of the heap(the rightmost child of last level). We do that as we simply want to maintain the first property, as if we take any other element out of the heap we won't have a valid complete binary tree. We then place this node at the root, and it might be possible that this node will not satisfy the second property of the heap, and hence we bubble down to make it a valid heap. It can also be noted that this process of bubbling down is also known as sift down.

Consider the pictorial representation shown below:




Deletion in Heap

## Applications of Binary Heaps

- Binary heaps are used in a famous sorting algorithm known as Heap sort.
- Binary heaps are also the main reason of implementing priority queues, as because of them the several priority queue operations like add(), remove() etc gets a time complexity of O(n).
- They are also the most preferred choice for solving Kth smallest / Kth Largest element questions.

## Heap: Time Complexity Analysis

Let's see the time complexity for various operations of heap.

### 1. Inserting an Element:

Inserting an element in heap includes inserting it at the leaf level and then bubbling it up if it is somehow violating any property of the heap. We know that the heap is a complete binary tree, and the height of a complete binary tree is ($\log N$) where N represents the number of elements in the tree. So, if we consider the worst case scenario where we might have to swap this newly inserted node to the very top, we will have 1 swap at each level of the tree, hence we will require log N swaps. Hence, the worst case time complexity of Inserting an element in a binary heap is: O(log N)

### 2. Deleting an Element:

Deleting an element from a heap includes removing the root node and then swapping it with the last node of the last level, and then if this new root node violates any heap property, we need to swap it with the child node, until the tree is a valid binary heap. Since, in worst case scenarios we might have to swap this new root node with node at the lower levels to the very bottom(leaf level), which in turn means the height of the tree, the time complexity of deleting the node from the binary heap thus in turn is: $O(\log N)$.

### 3. Get Min/Max Element:

Getting the max(or min) element in a binary heap is simply a constant time operation, as we know that if it is a min heap the minimum will be the root node, and similarly in case of a max heap the maximum element will also be the root node. So, time complexity of extracting Min/Max is: $O(1)$.

## Conclusions

- We learned what a heap is, followed by the explanation of what min/max heaps are.
- Then we learned how to do insertion into a heap, followed by deletion of an element from the heap.
- Then we talked about the applications of binary heaps.
- Finally, we discussed the time complexity of heaps.

---