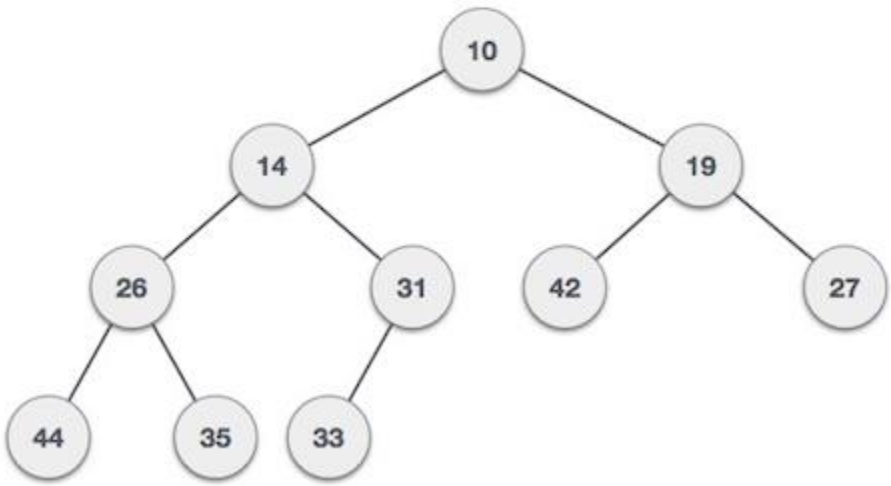
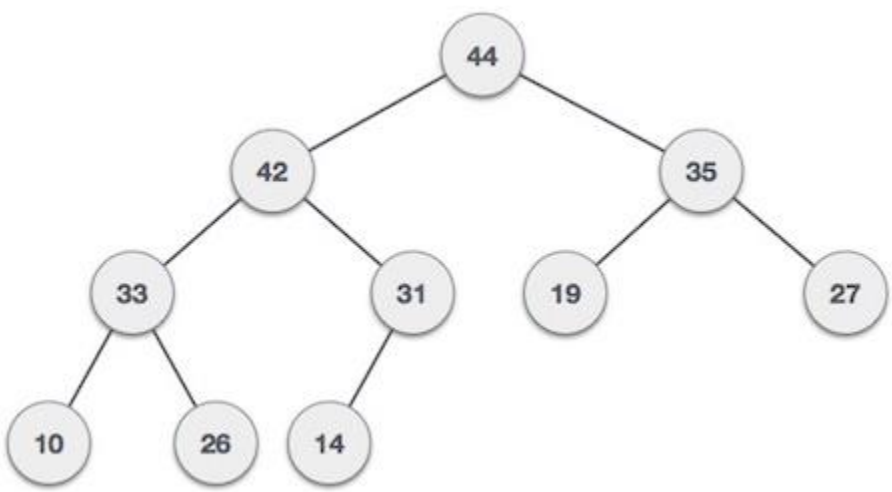
A heap is a specialized tree-based data structure that satisfied the heap property:

- If B is a child node of A, then key(A) ≥ key(B). This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a max-heap. Of course, there's also a min-heap.

Min-Heap − where the value of root node is less than or equal to either of its children.

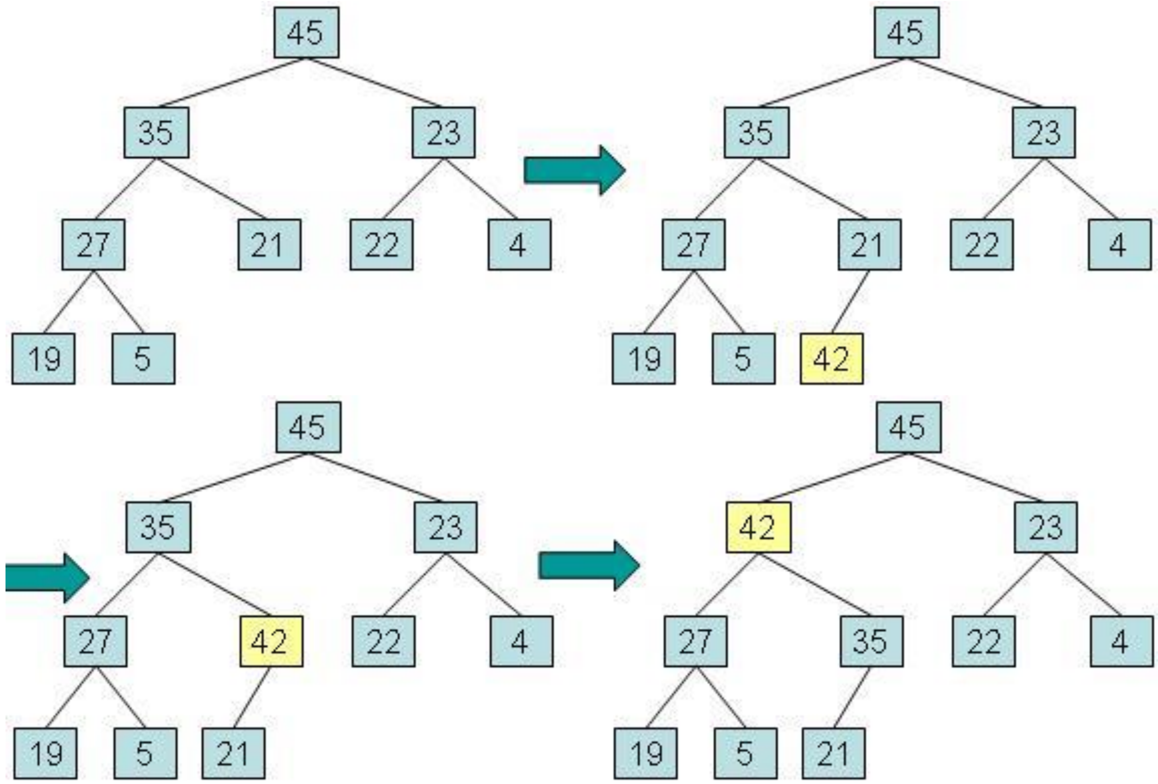


Max-Heap − where the value of root node is greater than or equal to either of its children.



Adding an Element to a Heap

Example: We want to insert a node with value 42 to the heap on the left.

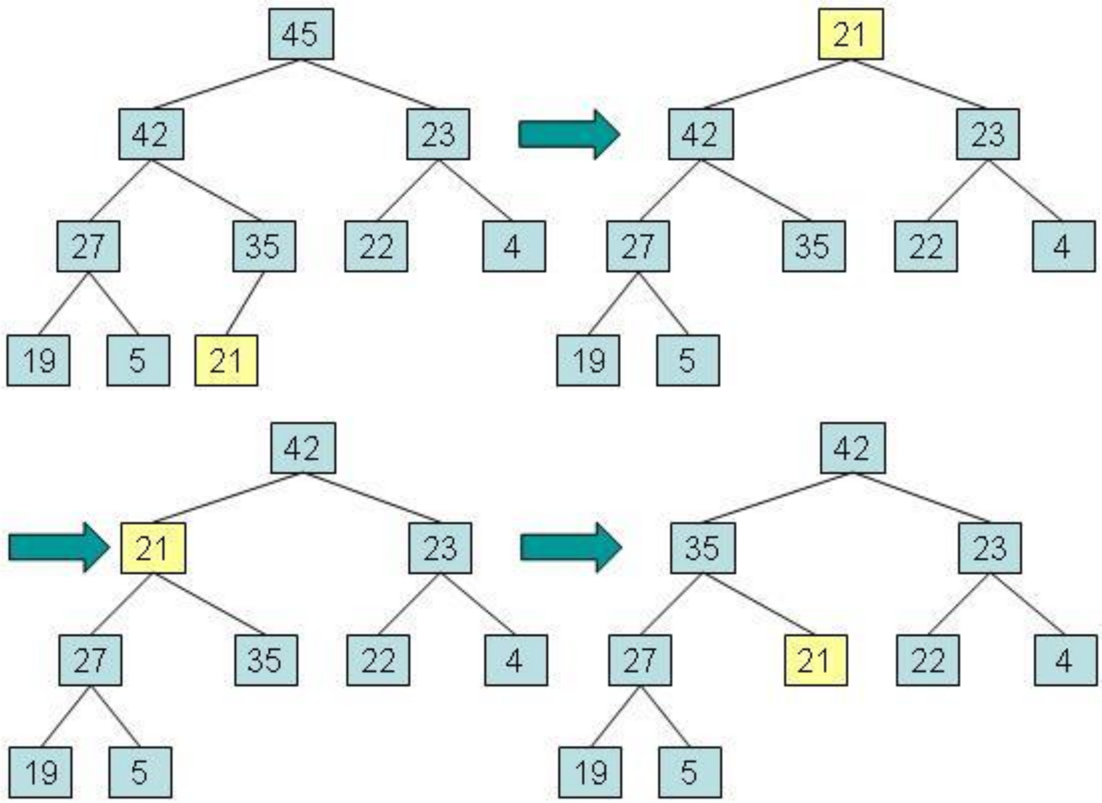


The above process is called **reheapification upward**.

<u>Pseudocode for Adding an Element</u>:

```
Step 1 – Create a new node at the end of heap.
Step 2 – Assign new value to the node.
Step 3 – Compare the value of this child node with its parent.
Step 4 – If value of parent is less than child, then swap them.
Step 5 – Repeat step 3 & 4 until Heap property holds.
```

Removing the Root of a Heap



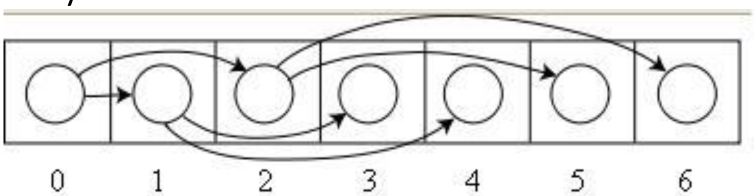
The above process is called **reheapification downward**.

Psuedocode for Removing the Root:

```
Step 1 − Remove root node.
Step 2 − Move the last element of last level to root.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.
```

Heap Implementation:

A more common approach is to store the heap in an array. Since heap is always a complete binary tree, it can be stored compactly. No space is required for pointers; instead, the parent and children of each node can be found by simple arithmetic on array indices.



The rules (assume the root is stored in *arr*[0]):

- For each index *i*, element *arr*[*i*] has children at *arr*[2*i* + 1] and *arr*[2*i* + 2], and the parent at *arr*[floor((*i* - 1)/2)].

Time Complexity:

Function reheapUp() has time complexity O(logn), when you add n elements to the heap, each takes O(logn), so total heap sort complexity O(nlogn)

Similarly, reheapDown() has time complexity O(logn), when you delete n elements to the heap, each takes O(logn), so total heap sort complexity O(nlogn)

In general, heap sort is O(nlogn) for Best, Average and worst case.

Heap/PriorityQueue (min/max):

- **Find Min/Find Max**: **O(1)**
- **Insert**: **O(log n)**
- **Delete Min/Delete Max**: **O(log n)**
- **Lookup, Delete** (if at all provided): **O(n)**, we will have to scan all the elements as they are not ordered like BST