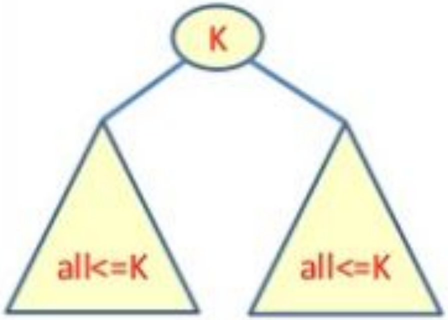
Binary Heaps

Binary Heaps

DEFINITION: A max-heap is a binary tree structure with the following properties:

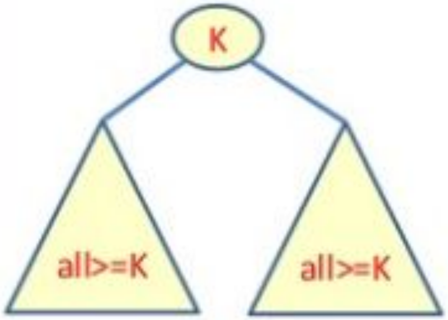
- The tree is complete or nearly complete.
- The key value of each node is greater than or equal to the key value



max-heap

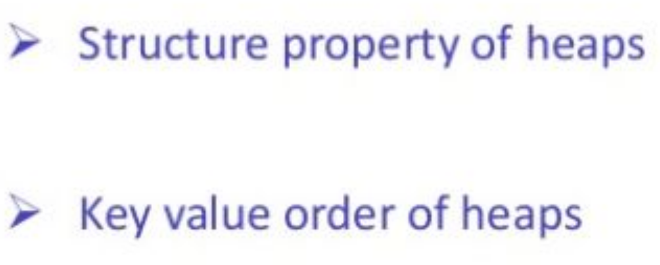
DEFINITION: A min-heap is a binary tree structure with the following properties:

- The tree is complete or nearly complete.
- The key value of each node is less than or equal to the key value in each of its descendents.



min-heap

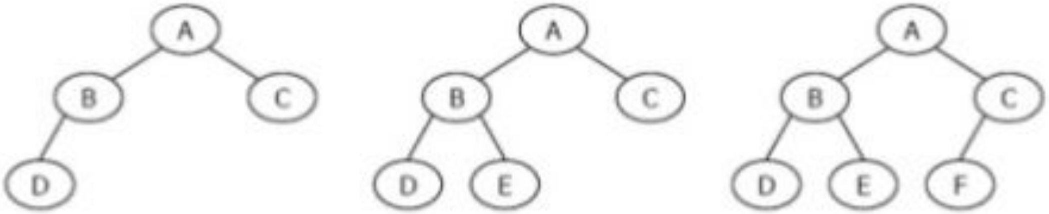
Properties of Binary Heap



- Structure property of heaps
- Key value order of heaps

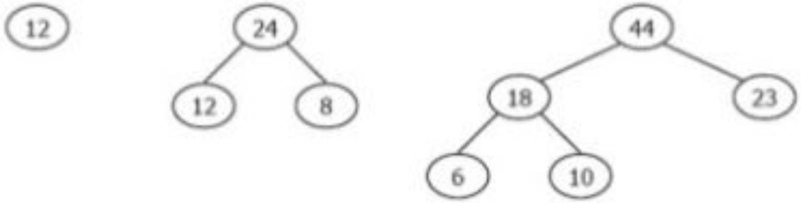
Structure property of heaps:

- A complete or nearly complete binary tree.
- If the height is h, the number of nodes n is between 2^{h-1} and $(2^h - 1)$
- Complete tree: $n = 2^h - 1$ when last level is full.
- Nearly complete: All nodes in the last level are on the left.

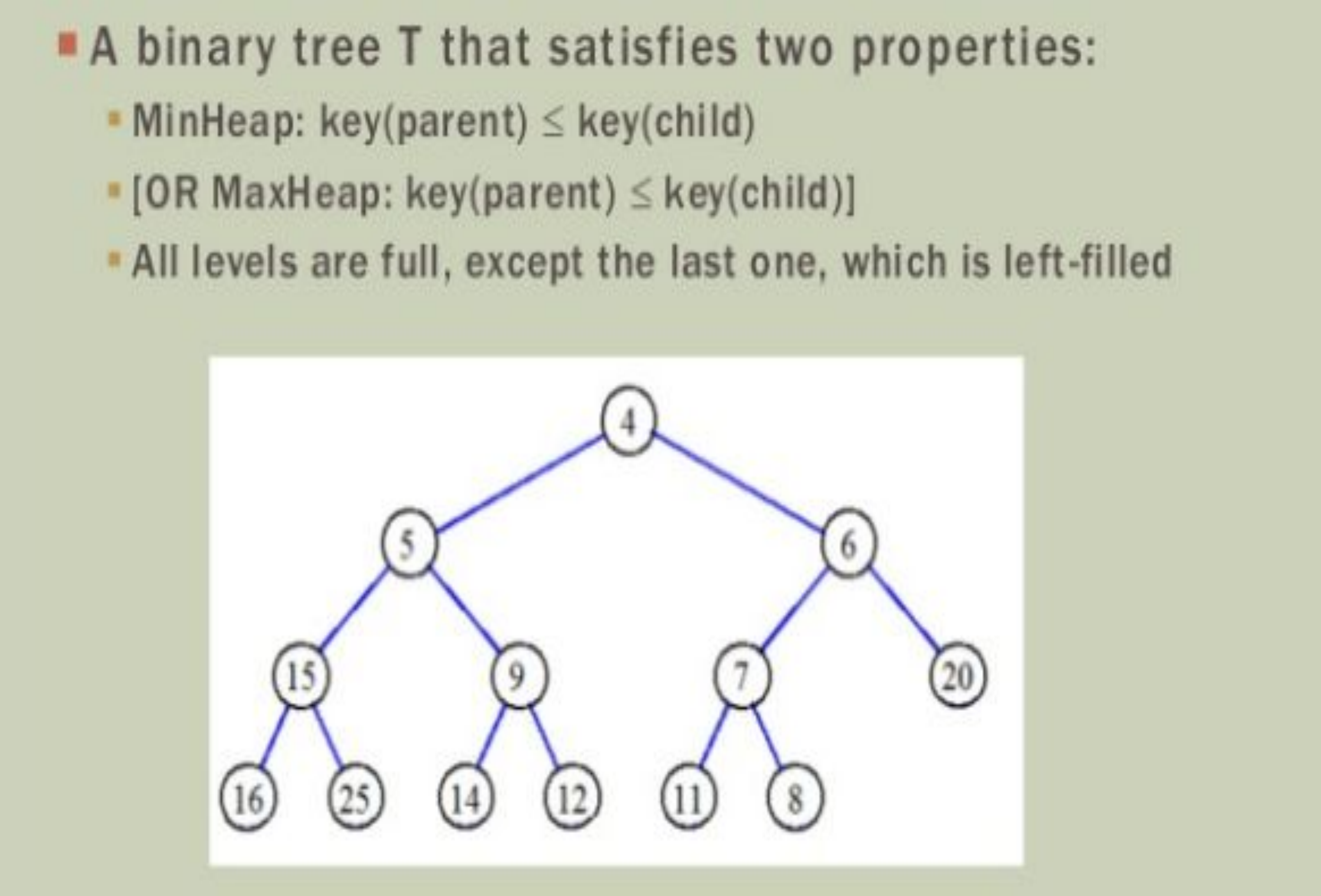


- $h = \lfloor \log_2 n \rfloor + 1$
- Can be represented in an array and no pointers are necessary.

Key value order of max-heap:

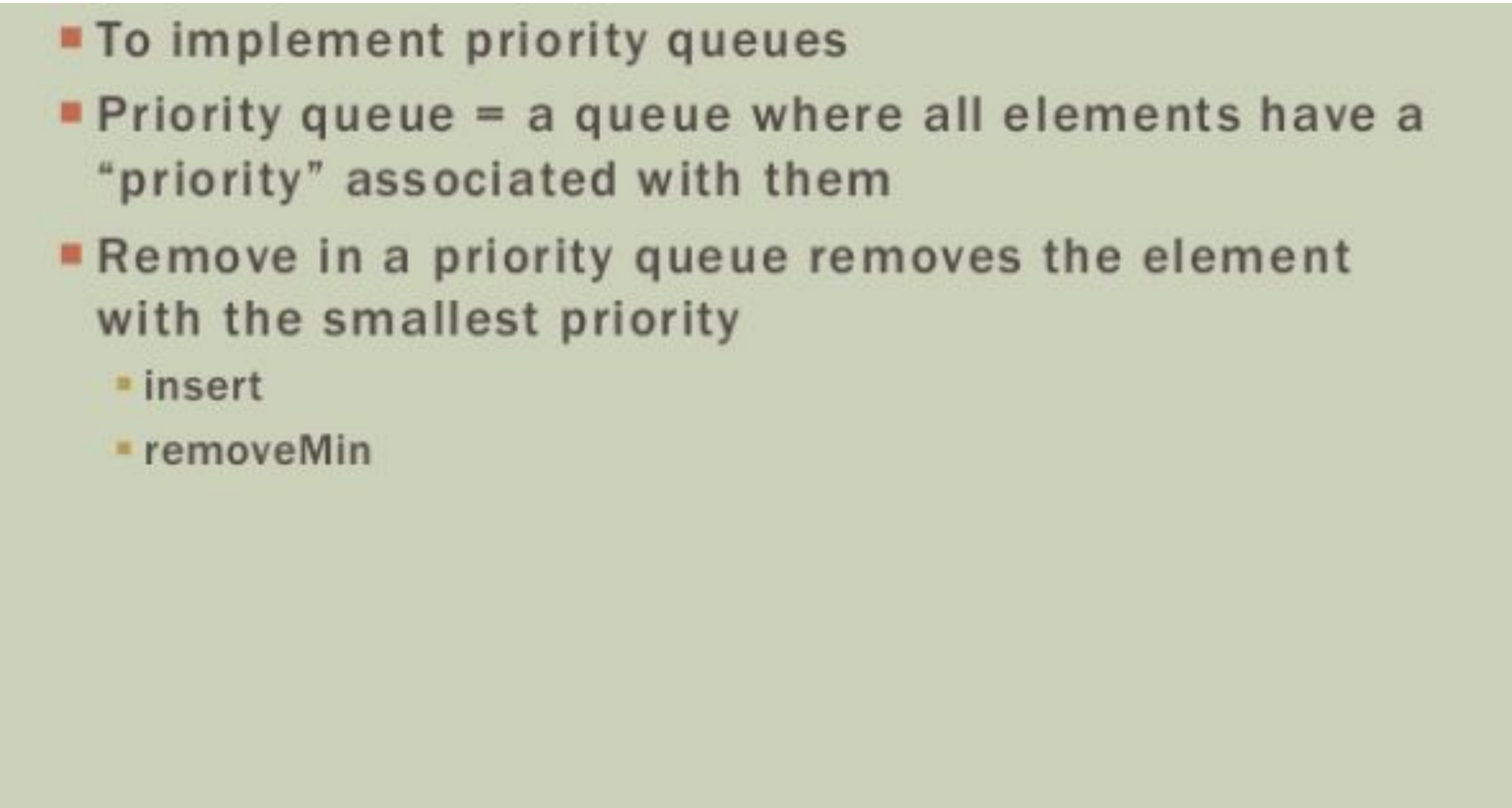


(max-heap is often called as *heap*)



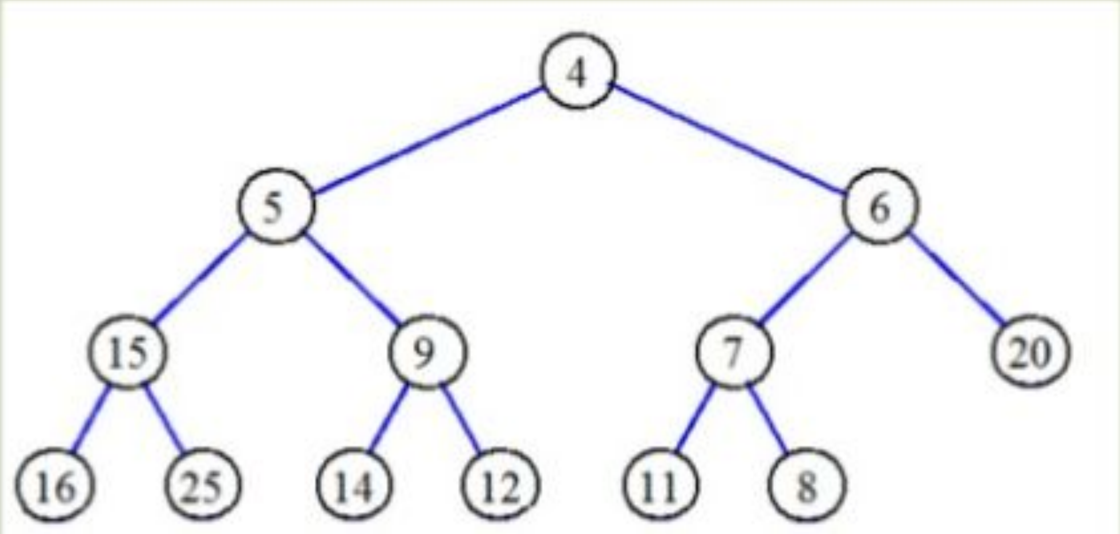
- A binary tree T that satisfies two properties:
  - MinHeap: key(parent) ≤ key(child)
  - [OR MaxHeap: key(parent) ≤ key(child)]
  - All levels are full, except the last one, which is left-filled

4
5
6
15
9
7
20
16
25
14
12
11
8

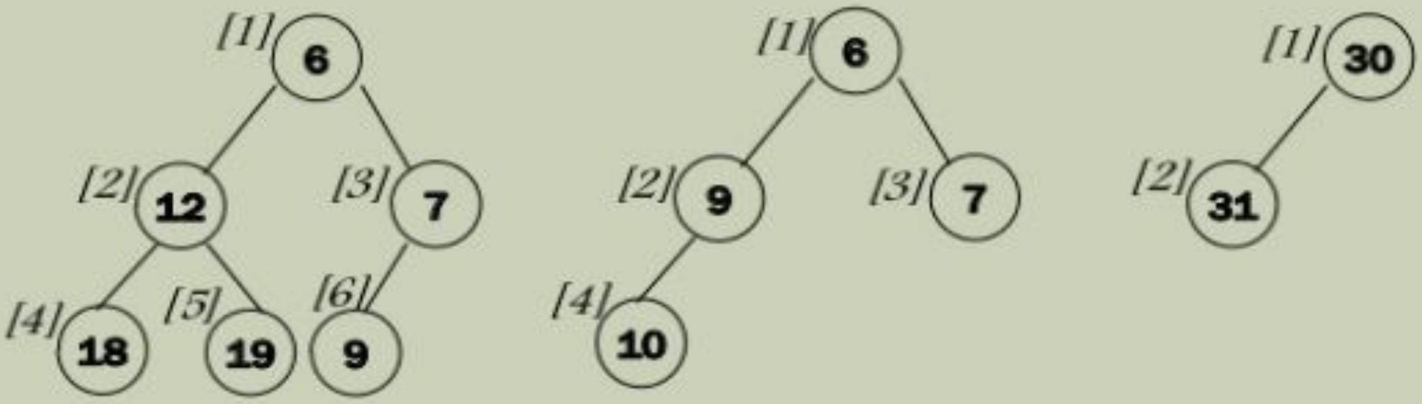


- To implement priority queues
- Priority queue = a queue where all elements have a “priority” associated with them
- Remove in a priority queue removes the element with the smallest priority
  - insert
  - removeMin

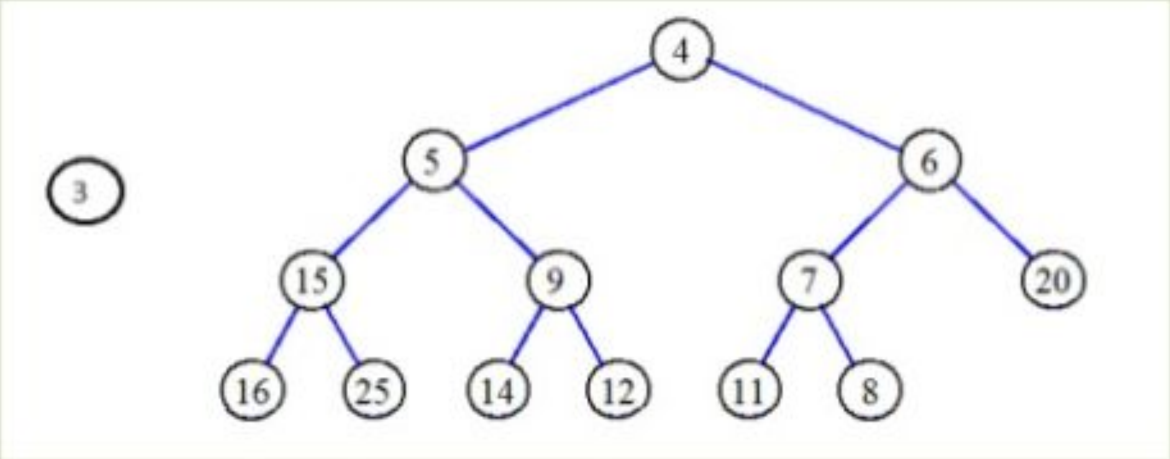
- A heap T storing n keys has height h = $\lfloor \log(n) \rfloor$, which is O(log n). *[The statement is true for* ***almost complete binary trees*** *in general.]*



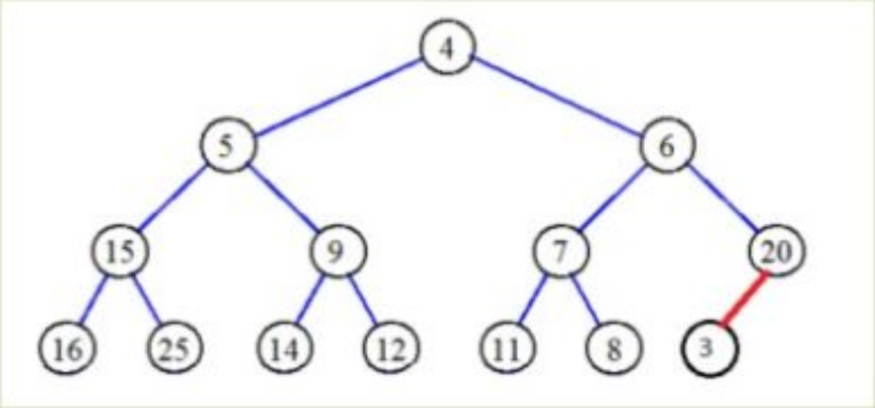
- Using arrays.
- If indexed from 1: Parent = k ; Children = 2k , 2k+1
- Efficient! *[No pointers. Just array indexes.]*

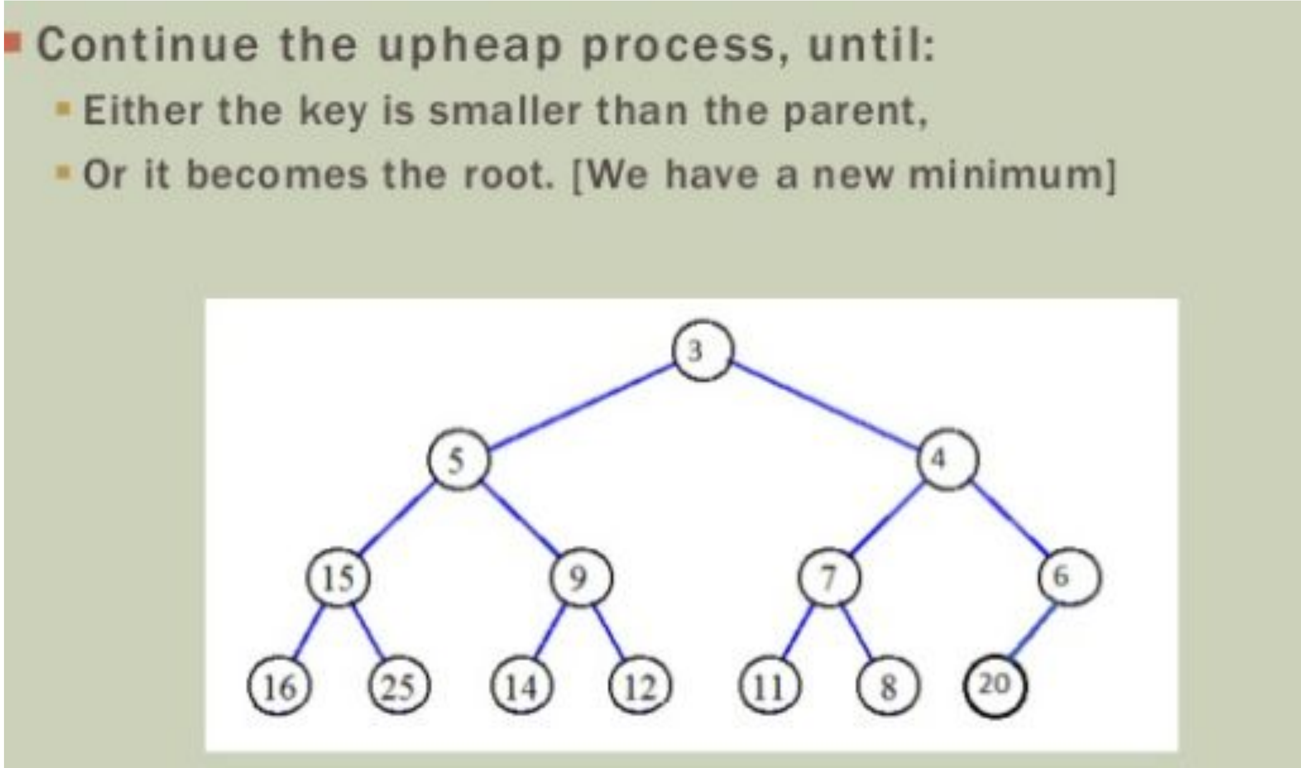


- Insert 3

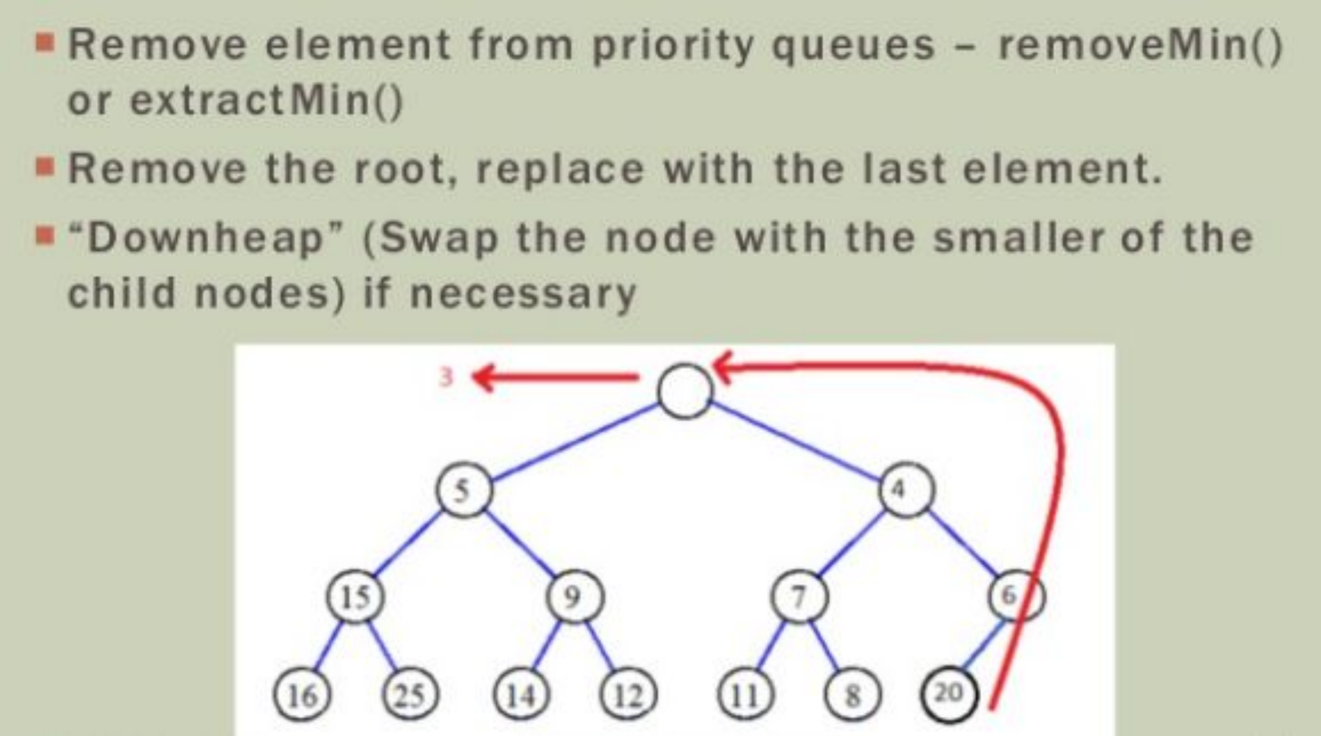


- Just insert it at the first empty slot.
- Upheap (Shift the key up), if necessary



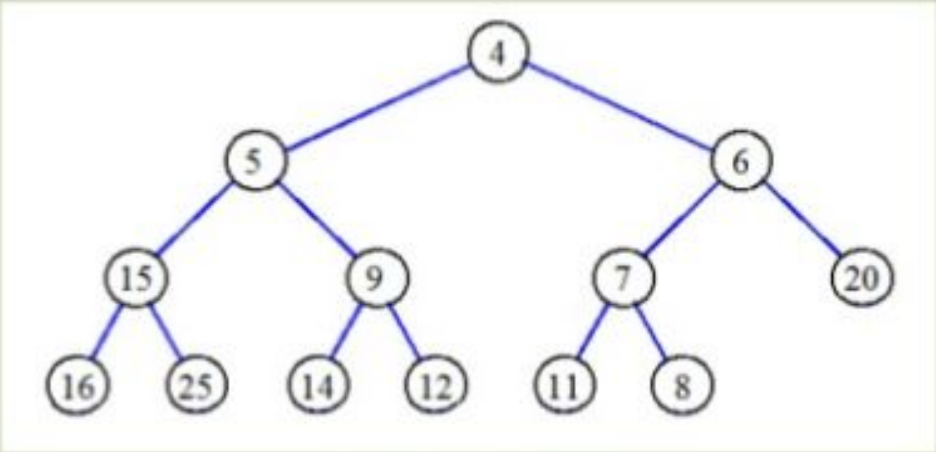


- Continue the upheap process, until:
  - Either the key is smaller than the parent,
  - Or it becomes the root. [We have a new minimum]



- Remove element from priority queues – removeMin() or extractMin()
- Remove the root, replace with the last element.
- “Downheap” (Swap the node with the smaller of the child nodes) if necessary

- Terminate downheap when
  - reach leaf level
  - key parent is greater than key child



Thank You!!!