Lecture 8 : Heap sort

~~Dictionary~~ → (k, v)

Priority Queue :

$\boxed{\text{Operation}}$: Identify or remove an entry whose key is the $\boxed{\text{lowest.}}$ → $\Theta(1)$

(5 : e1)

* Insert any key at any time (Flexibility)

Eg Event queue

5 : e1
8 : e2

→ Insert (9 : e3) →

5 : e1
8 : e2
9 : e3

$\boxed{\text{removeMin( )}}$

8 : e2
9 : e3

$\boxed{\text{min( )}}$ → (8 : e2)

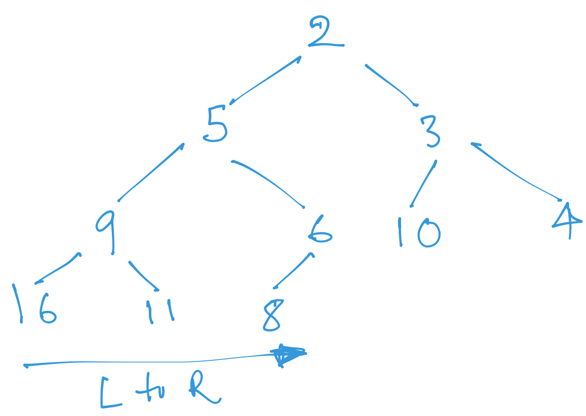
How to implement this efficiently ?

⟹ $\boxed{\text{Binary Heap}}$: An implementation of a priority queue.

Binary Heap : $\boxed{\text{Complete}}$ Binary Tree.

* Every level is FULL, except possibly the bottom level, which is filled up from left to right.

Eg



2

5 3

9 6 10 4

16 11 8

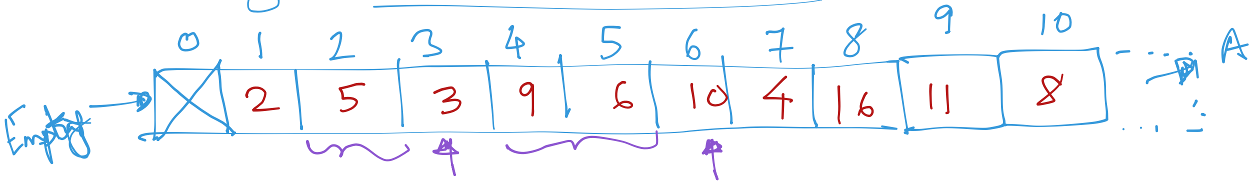
* Entries in a binary heap satisfy the

HEAP - ORDER PROPERTY

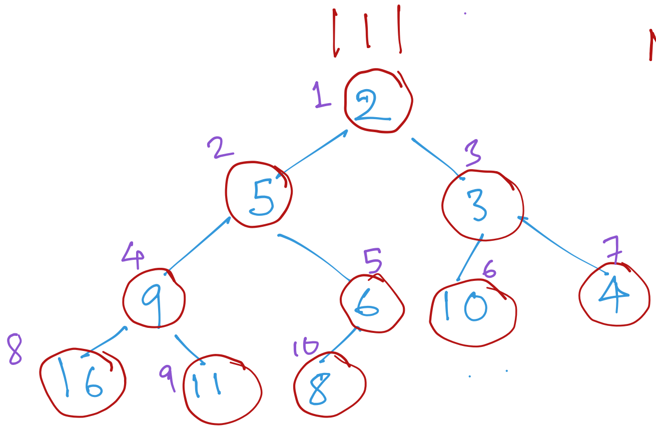
⟹ No child has a key less than its

parent's key.

* Often stored as ARRAYS of entries by level-order traversal.



Insert()
Min()
RemoveMin()



Node i's children: $\boxed{2i}$ and $\boxed{2i+1}$

Node i's parent: $\left\lfloor \frac{i}{2} \right\rfloor$

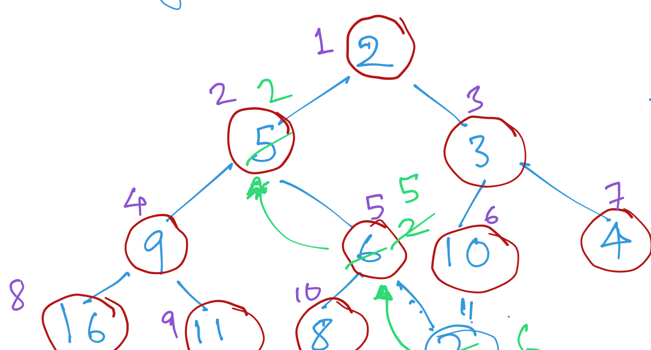
1) int min() → $\boxed{A[1]}$ $\boxed{\Theta(1)}$

2) insert (k, v)

* Let n be the new entry (k, v).
Place n in the bottom level of the tree.
at the first open slot from the left.
OR start a new level with the leftmost entry.

⇒ First free location in the ARRAY.
↳ May violate the heap-order property.

$\boxed{O(\lg n)}$



Bubble up the entry until the heap order property is satisfied.

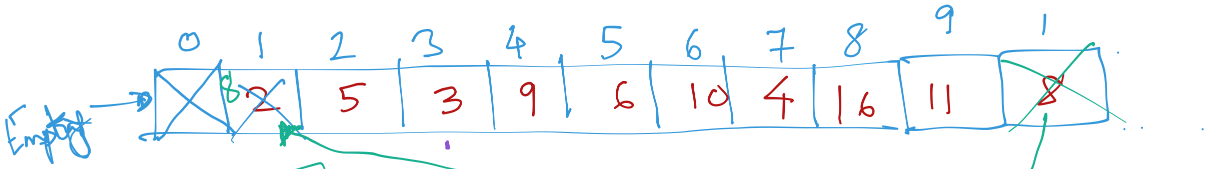
Repeat

Compare n's key with its parent's key.
If n's key is less, then exchange.

$\boxed{t = A[1];}$

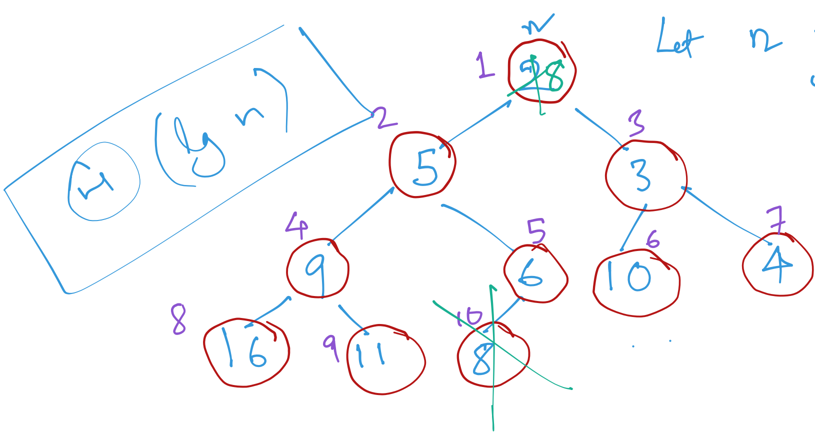
3. `int removeMin()`

- Remove the entry at the root.
- Save the value for return.



$\boxed{t := 2;}$

- Fill the hole with the last entry in the tree.

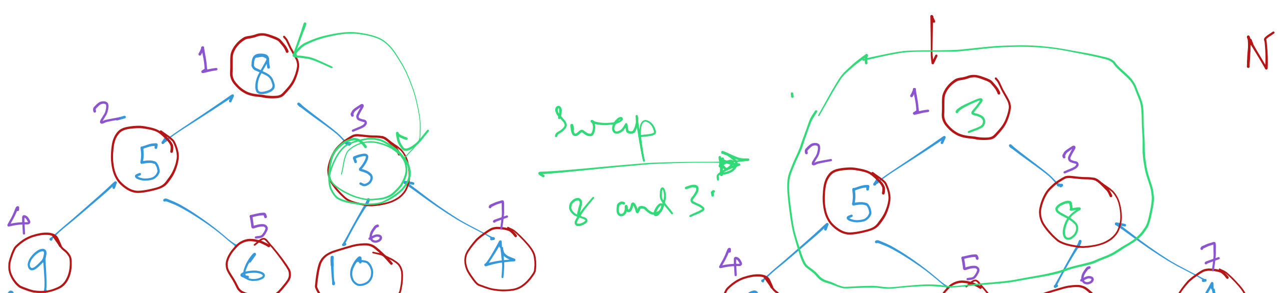


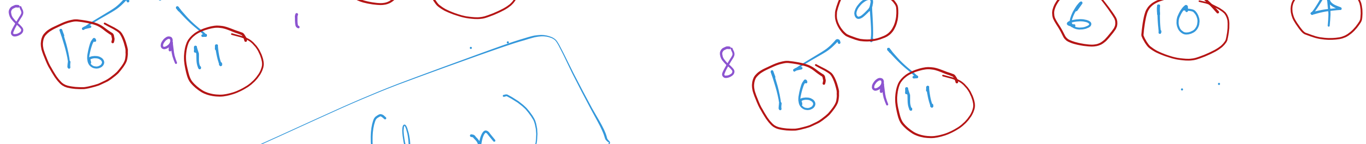
Let n → node which moved up the tree.

- Bubble down n so that Heap-order property is restored.

Repeat

If n > one or both of its children, swap n with the MIN child.





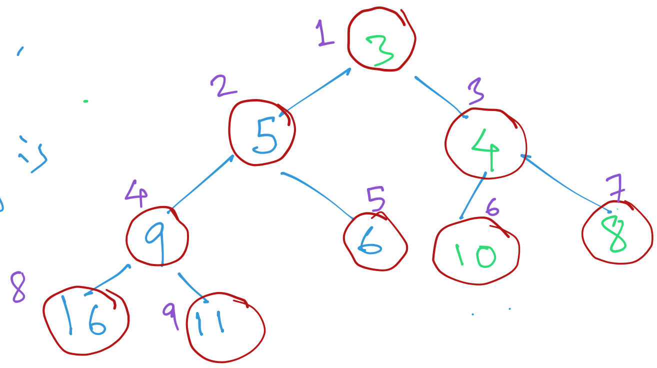
$\boxed{\Theta(\lg n)}$

Swap 8 and 4

Min-Heap is

$\rightarrow$ Min-key at the root,

Max-heap $\rightarrow$ Max key is at the root.



Every subtree of a binary heap is a binary heap.

$\Rightarrow$ Min-heap

N $\rightarrow$ # of entries in the Heap, (Complete Binary tree.)

$\Rightarrow$ $\Theta(\lg n)$ worst case time

(Bubble-up / down ops are involved)

Bottom-up Heap construction

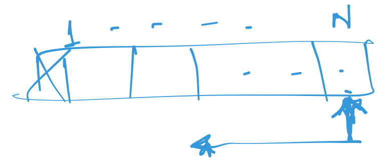
Given a set of entries, make a heap out of them.

4 bottomUpHeap() : Insert the entries one by one $\rightarrow$ $\Theta(n \lg n)$

Heap-order property.

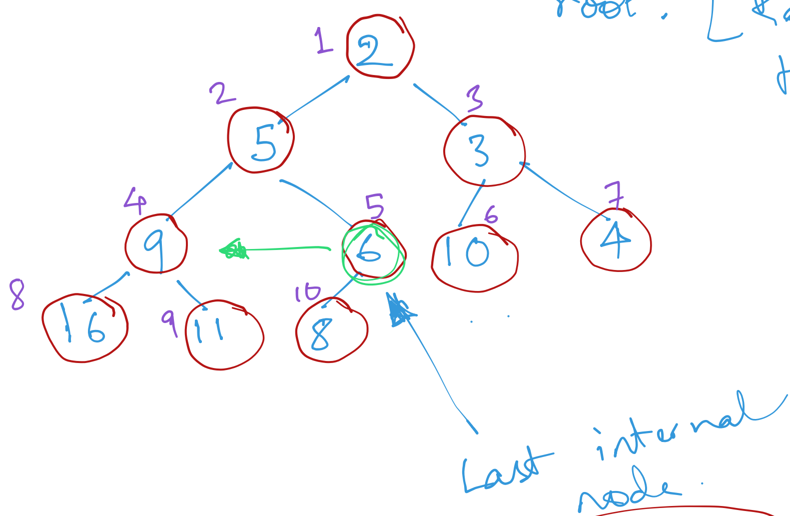
Alternative

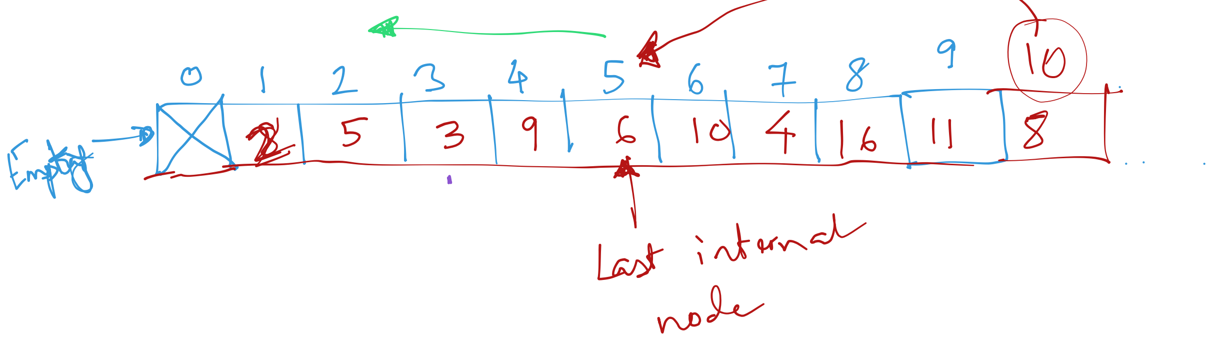
- Make a complete binary tree out of the entries in any order.



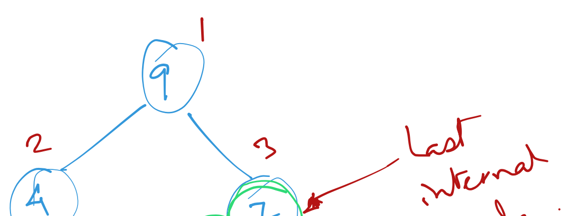
⇒ Broken Heap.

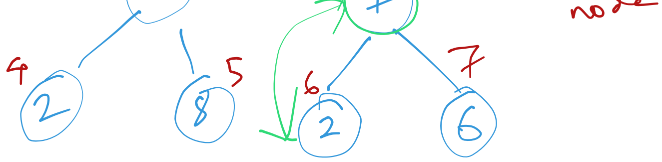
- Walk backward through the array from the last internal node (Not a leaf) to the root. [Reverse order in the ARRAY]

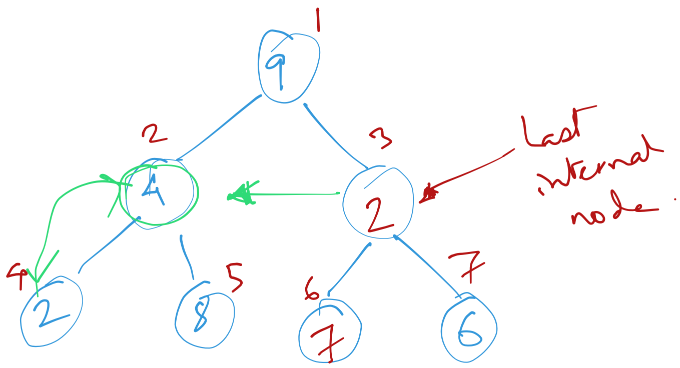


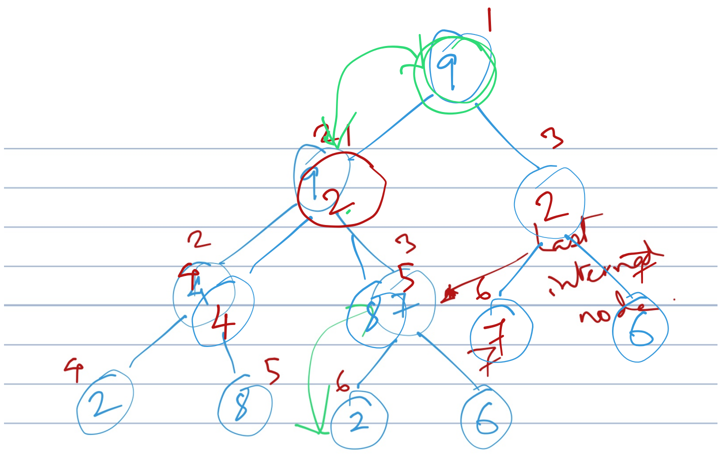


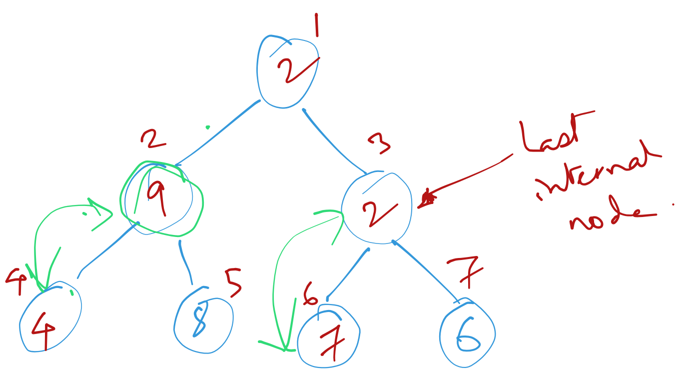
* When we visit a node, bubble it down as in removeMin().



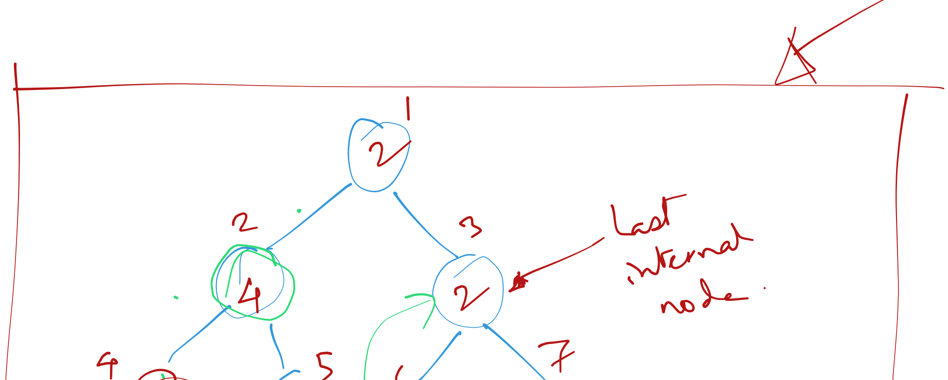

node

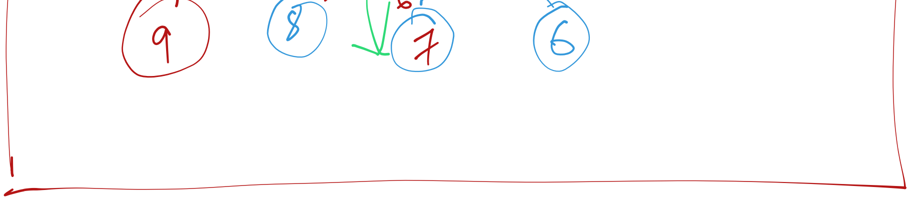

Last internal node.


Last internal node.


Last internal node.

Min-Heap.


Last internal node.


9
8
7
6