9/20/18

PRIORITY QUEUES (HEAPS)

→ Want a data structure with complexity to insert or delete element as $\Theta(\log n)$ (n → # of elements)

find minimum in $\Theta(1)$.

→ There are min-heaps and max-heaps

find min in $\Theta(1)$ — find max in $\Theta(1)$

We discuss min-heaps.

What is a heap?

- Conceptually a heap is a binary tree such that for every node v (other than the root) :

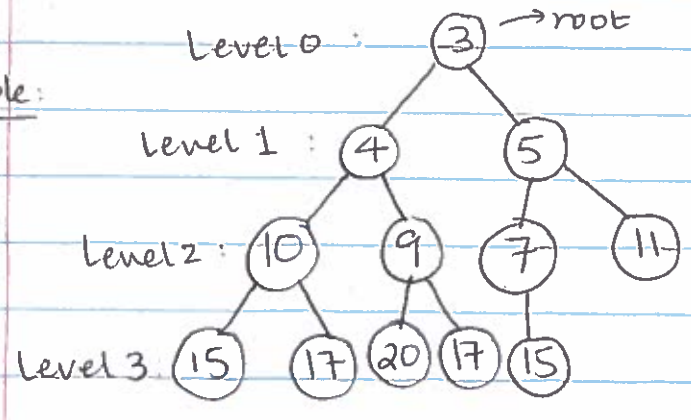
value (parent (v)) ≤ value (v)

"key" = "priority"

"HEAP PROPERTY"

(For max heaps, value (parent(v)) ≥ value (v))

Example:



Heap property satisfied!

→ Few things to notice:

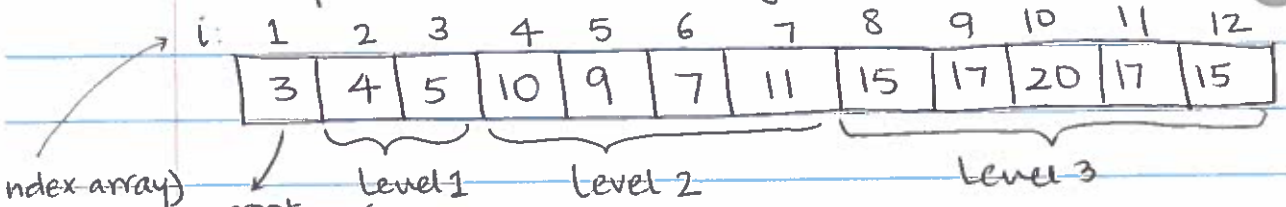
- Duplicates are okay!
- Many valid heaps for same set of elements.
  (In the previous example, you can switch ④ and ⑤)
- "Balanced" Binary Tree

  - Complete Binary Tree is one with all nodes
    Heap need not be one

  - Missing elements can only be on the side.

How are heaps represented?

- Implemented as an array:



That's why we have a requirement of balanced tree in order to represent heap as this sort of an array.

→ Node at index 'i' → has children at indices: '2i' and '2i+1'

→ has parent at index: $\left\lfloor \frac{i}{2} \right\rfloor$

- In constant time, for a given node you can find the children and parent of that node.

- Minimum is always the root! → So getting minimum has complexity $\Theta(1)$

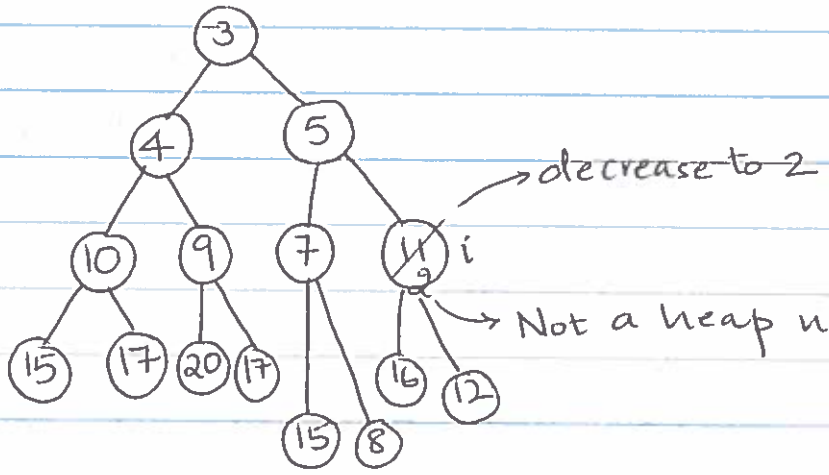
  ↓

  Finding maximum in min-heap? $\Theta(n)$

⟹ Insertions And Deletions:

→ Change: $\Theta(\log n)$

Decrease Element

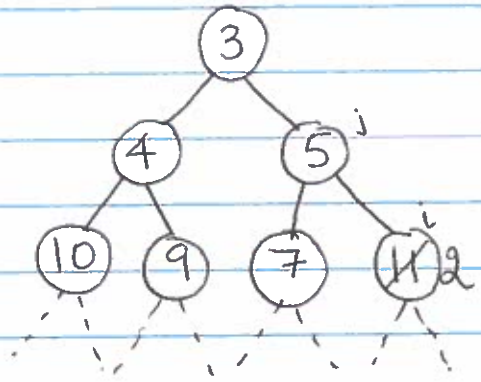


- No problem with children of 'i' but issue with parent
- How do we make this a heap now? Switch with it's parent!

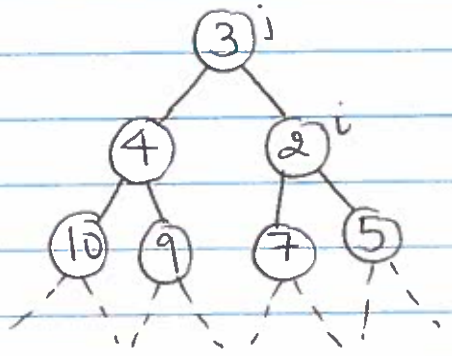
  ↓

  Algorithm to do so called Heapify-up.

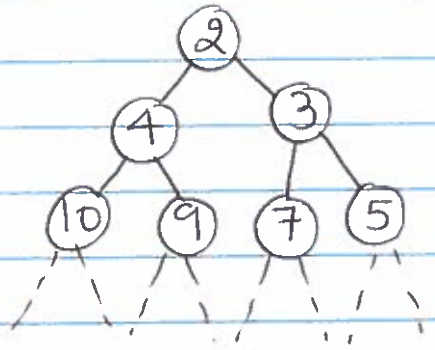
→ Heapify-up(i) : If the parent of i (j) is larger than i, swap the value of i and j, and call Heapify-up(j).



i=2 , j=5
Swap!



i=2 , j=3
Swap!



Reached Root so Stop!

Now we have a heap! ☺

→ Proof for Correct Working of Heapify-Up

Claim: If we start with a valid heap, decreas the value of any node i, and run Heapify-u then we get a valid heap.

Proof: By induction on level of node i → Recursive so by induction makes s

- Base case: If i is root, then decreasing i is still a heap.
- Inductive Hypothesis:

  Assume that if you first start with a valid hea decrease any level l node i and run Heapify-up(i), then you get a valid heap.

- Want to Prove: That if you first start with a valid heap, decrease any level l+1 node i and run Heapify-up(i), then you get a valid heap.

We can think of "swap the value of i and j" as

temp = value(i)

value(i) = value(j)

value(j) = temp ← At this point we have a val heap

j → parent of i

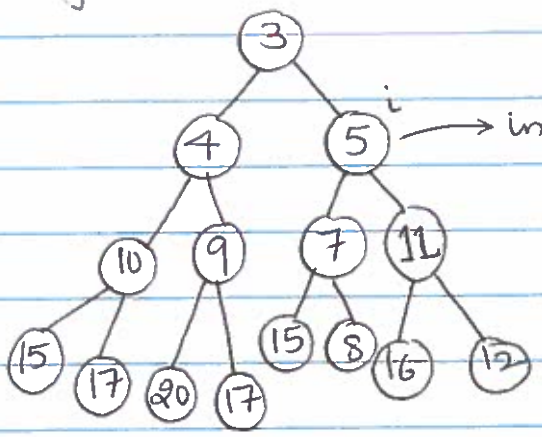
decrease level l node as j is pare of i

So by inductive assumption, running Heapify-up(j) gives a valid heap.

→ Running Time of Heapify-up:

Worst case → $\Theta(\log n)$ (h has to go through all log(n) levels)

→ Now say we increase an element:



- Problem with children and not parents here.
  ↓
  Swap with children!

→ Heapify-down(i): If i is larger than its smallest child, swap their values and call Heapify-Down(j)

j ←

Recursive.

Proof similar to Heapify-Up

Claim: If start with a valid heap, increase any node i, and run Heapify-Down(i), then we get a valid heap.

Run-time is $\Theta(\log n)$