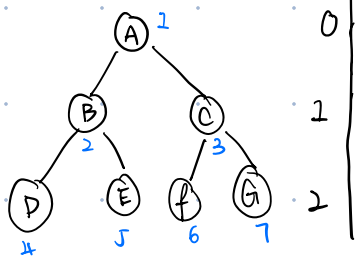
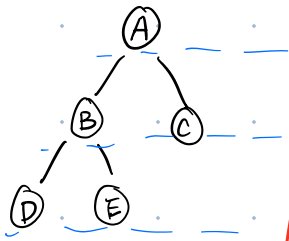
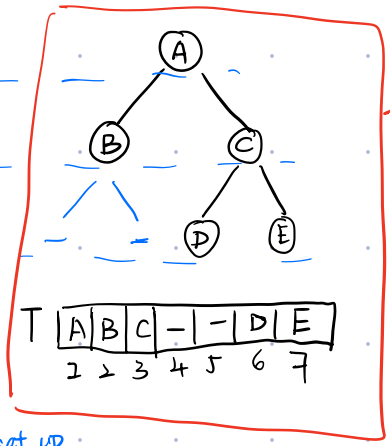
Heap : Introduction



T | A | B | C | D | E | F | G |
1 2 3 4 5 6 7



T | A | B | C | D | E |
1 2 3 4 5



T | A | B | C | - | - | D | E |
1 2 3 4 5 6 7

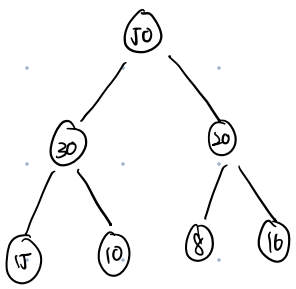
→ This is not a complete binary tree

exploration: The order is set up from side by side, from left to right.

(1) Node at the index — i
(2) left child — $2 \times i$
(3) right child — $2 \times i + 1$
(4) its parent — $\lfloor \frac{i}{2} \rfloor$

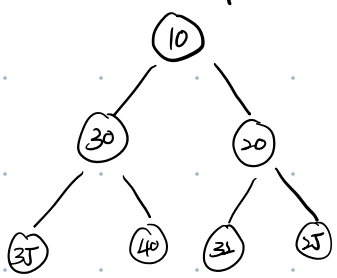
2. Structure

max Heap



H | 50 | 30 | 20 | 15 | 10 | 8 | 16 |
1 2 3 4 5 6 7

Min Heap

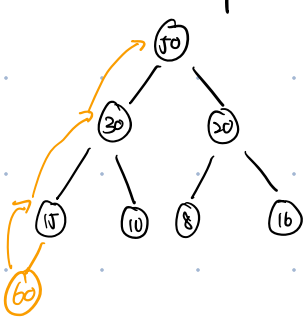


H | 10 | 30 | 20 | 35 | 40 | 32 | 25 |
1 2 3 4 5 6 7

3. Insert items into Heap

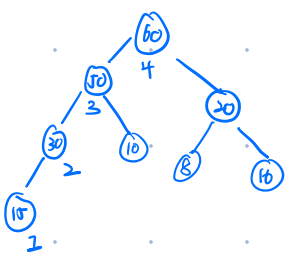
Given the scenario:

Max Heap



H | 50 | 30 | 20 | 15 | 10 | 8 | 16 | 60 |
1 2 3 4 5 6 7 8

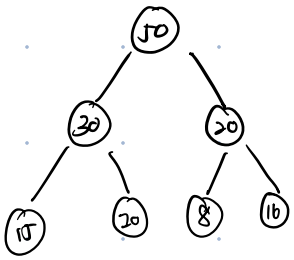
(1) Insert the node 60 compare to each parents, and it will reach to right place → let's see the order:



Big O: O(logn)

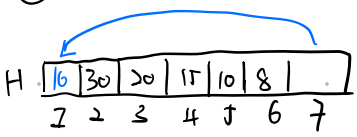
4. Delete

Given the senario:



(1) we can only remove the root element.

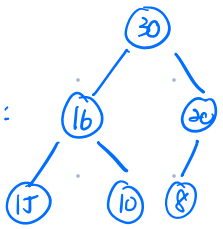
Because we need to keep the completeness of Binary tree, but it's truly some ways to remove other nodes, we discuss it later.



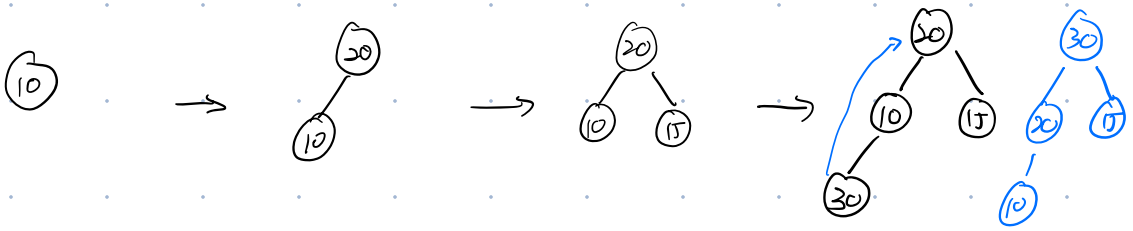
O(logn)

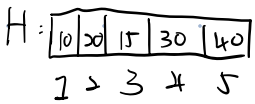
(2) Delete 30

1. and the last element will replace the root element
2. then compare with root's children, and the greater children will take place of it

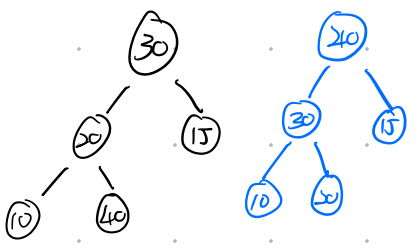


5. Create Heap

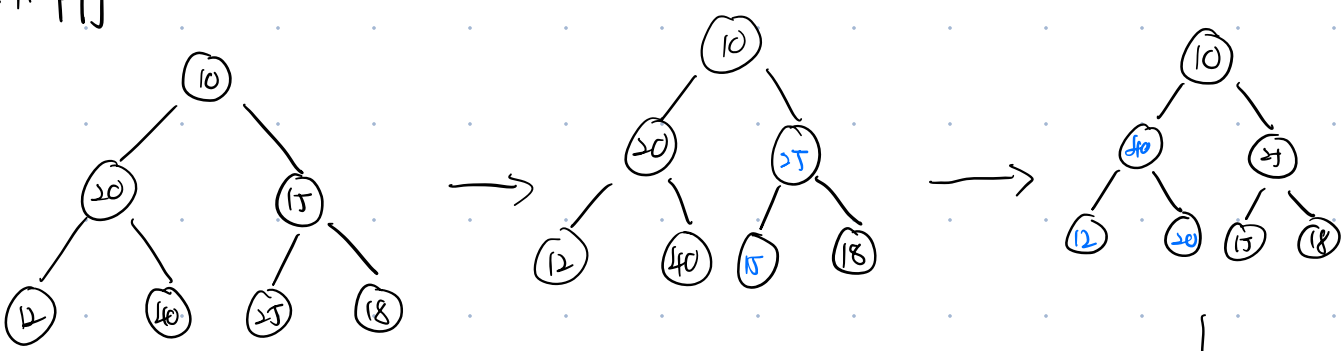


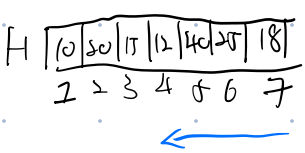


O(nlogn)



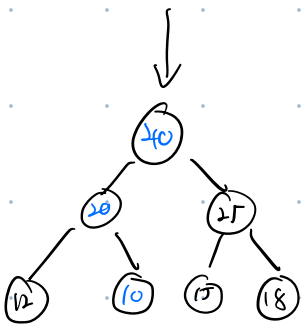
6. Heapify





O(n)

(1) look the order Ended element (Without descendants)



Priority Queue

1. Small number has high priority
   Large number has low priority

- Priority - Based Access

Max - priority queue, the element with highest priority is dequeued first.

Min - priority queue, the element with lowest priority is dequeued first.