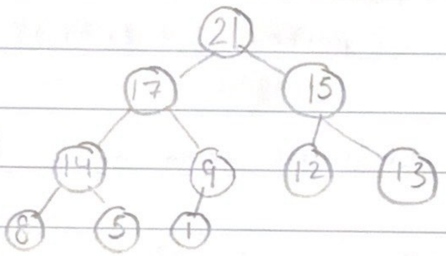
[Review Quiz #1 questions on these]

Heaps (Inserd/Delete/Heap build)

Heap = data structure to manage information

- sometimes called binery Heaps
- nearly/complete binary trees

Example of Heap:



- levels filled except the lowest
- lowest level filled to a certain point starting from the left.

Uses:

- heap sort
- priority quese

types:

- max-heap
- min-heaps

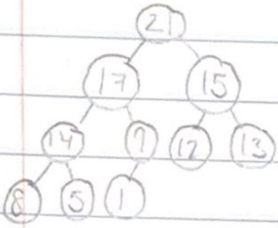
This is a max-heap, conditions for a max-heap is value of i ≤ value of parent, Max-heaps are used for heap sort

min-heap: value of i ≥ value of parent, work great for priority queues

height of a heap is O(log N)

i = index

Max-Heap Represented As an array:



→

21	17	15	14	9	12	13	8	5	1
0	1	2	3	4	5	6	7	8	9

root = A[1]	to get left child = $left(i) = 2i + 1$
(index 1 of the array)	right child = $right(i) = 2i + 2$
	$parent(i) = (i-1)/2$

example toget 15, left child we would do:

$left(i) = 2i + 1$

$left(2) = 2(2)+1 = 5$ ← so this is the index we look at

[index 5 we have 12]

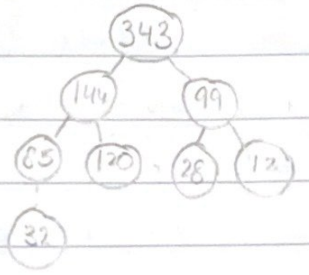
↑

We put 3 for i, as index of 15 is 3

Sub-Heaps are part of a heap itself, smaller portions of it.

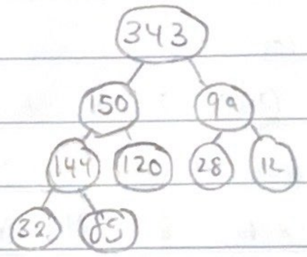
(Max) Heaps : Inserting Values.

We Add new values to a Heap, the same way we add values to a normal BST. From left to right!



So adding 32 to this would just look like adding it to 85

Now lets say we want to add 150 though, since the incoming value is greater than the parent, we need to move nodes around!



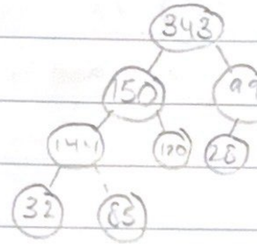
Time Complexity For Inserting Heap:

- Best: $\Theta(1)$
- Average: $\Theta(\log N)$
- Worst: $\Theta(\log N)$

Insertion for Min-Heaps are similar, the only difference is, we will do swaps to ensure a different condition is true for all nodes in the heap:

- The root is lesser than (or equal to) its children

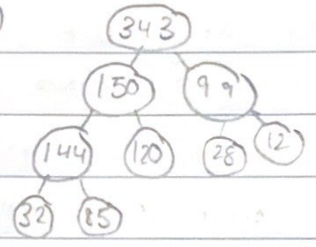
Max (Heaps): Deleting Values



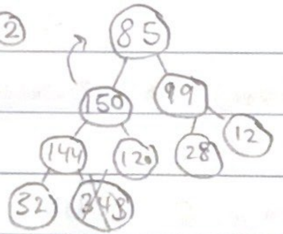
We want to delete 85 Since it is just a leaf Node its super easy, we can just get rid of it!

Now lets say we want to delete 343. To do this we will first swap it with a leaf Node, and delete 343. Next we will swap around the new Root Node, til it is in the correct spot!

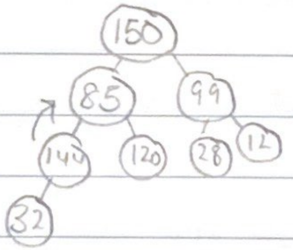
①



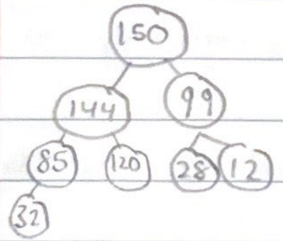
②



③



④



Done! Since parent 85 is greater than Child Nodes 32

Time Complexity:

Best - $\Theta(1)$

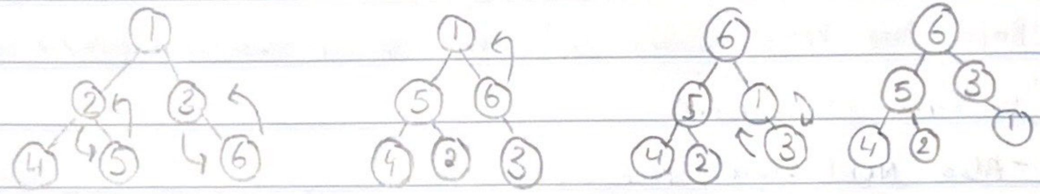
Average - $\Theta(\log N)$

Worst - $\Theta(\log N)$

BuildHeap

1	2	3	4	5	6

↑
start from middle! with heapify



BuildHeap Time Complexity: O(n)

Space Complexity (All Heap Types): O(1)