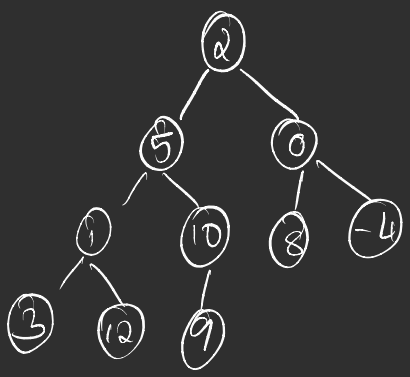
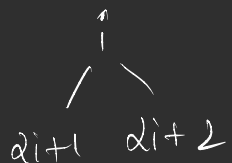
Heap & Priority Queue

Notes

Heap & Priority Queue:

* A particular type of binary tree
* Eg:



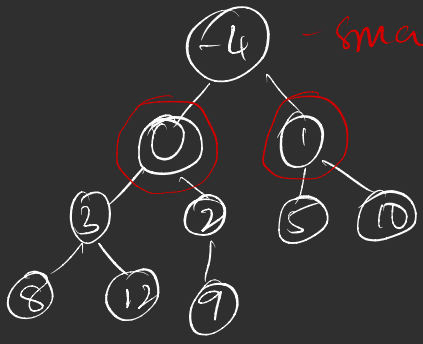


A = [2 5 0 1 10 8 -4 3 12 9]
0 1 2 3 4 5 6 7 8 9

This is a binary tree & not a heap.

* Heap has 2 versions: min heap & max heap

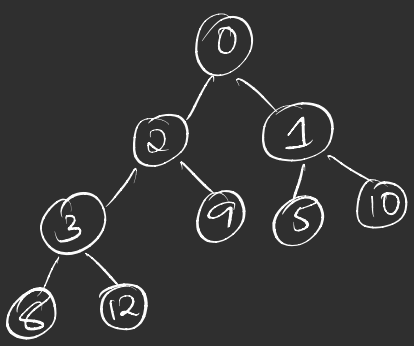
Specifically used to get min & max elements using min heapify.



- Smaller than 0 & 1

Heap POP : -4

time complexity = $O(\log n)$ } (Extract min)



operation going down the tree is $O(\log n)$

Insert :

Heap push : $O(\log n)$

why log n?

If you had 'n' nodes in the tree total and if we start we start with the root node, we go left we are basically saying I'm not doing half of the total tree and further we choose left or right and again we do only half of the total subtree.

we repeatedly keep dividing n by 2

and so whenever you do something involving a branch assuming the tree is roughly height balanced then it is going to be $\log_2 n$

Heap Peak : ask what the minimum is

$\Downarrow$

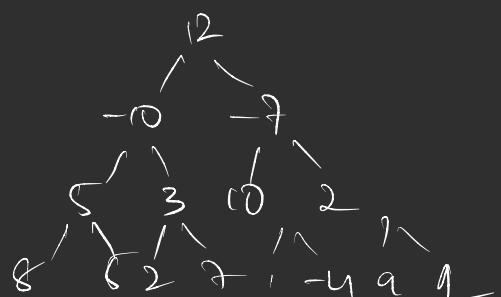
$O(1)$

Heap Sort : repeatedly pop off the minimum means n elements pop in $\log n$ time and hence it is $O(n \log n)$

and space is $O(1)$: rearranging the elements in that array representation

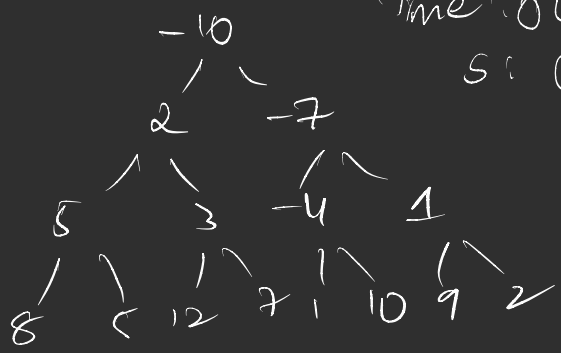
Heapify :

initially it is a binary tree



sift down:

min heap:



Time: O(n) ⇒ same for max heap
S: O(1)

Operation	Time	Space
• Build min heap heapq.heapify(A)	O(n)	O(1)
• Insert Element heapq.heappush(A, 1)	O(log n)	
• Extract Min Element heapq.heappop(A)	O(log n)	
• Heap sort custom heapsort function	O(nlogn)	O(n)
• heapq.heappushpop(A, 99)	O(logn)	