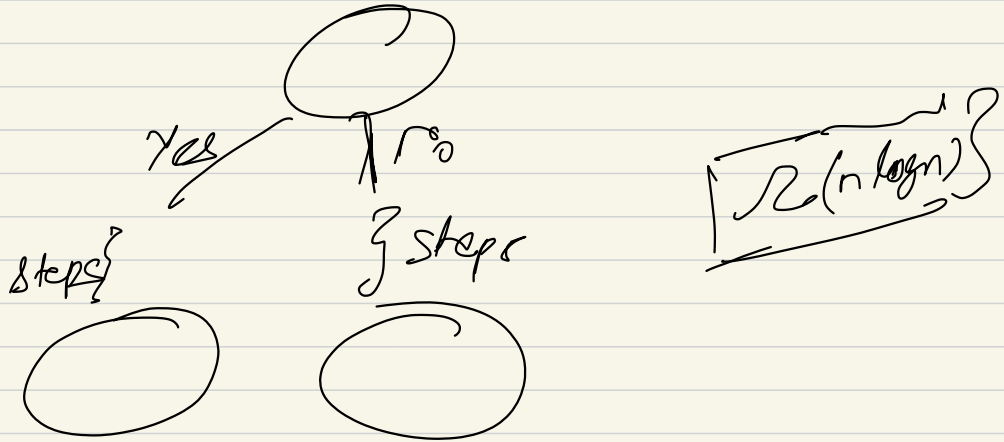
Comparison based sorting Model

items as black box

Only comparison is allowed



$O(n \log n)$ $\Theta(n \log n)$

Linear-time Sorting

Counting Sort — $O(n)$ — Cond^n: range $K = O(n)$

Radix Sort — $O(n)$ — $K = O(n^d)$

Bucket Sort — expected $O(n)$ (assume about the distribution)

Integers

Heaps

almost complete binary tree

array to represent trees

Oper^n:
- → DeleteMax
- → FindMax
- → Insert

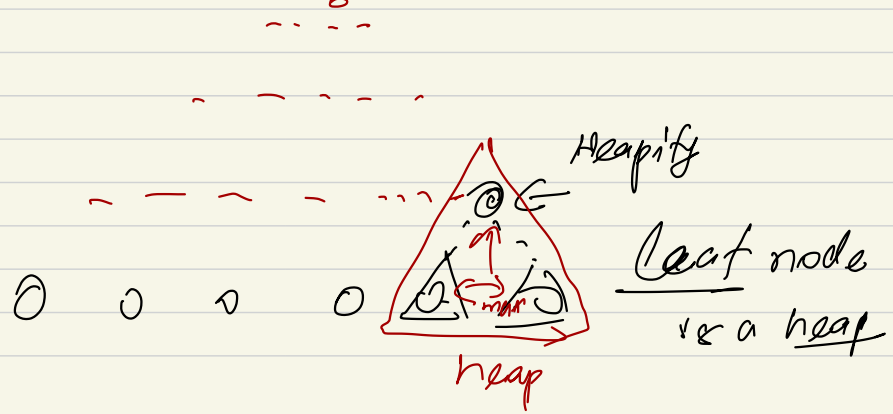
Search? $O(n)$

Given a problem → see the requirements

always — find the max of all the elements

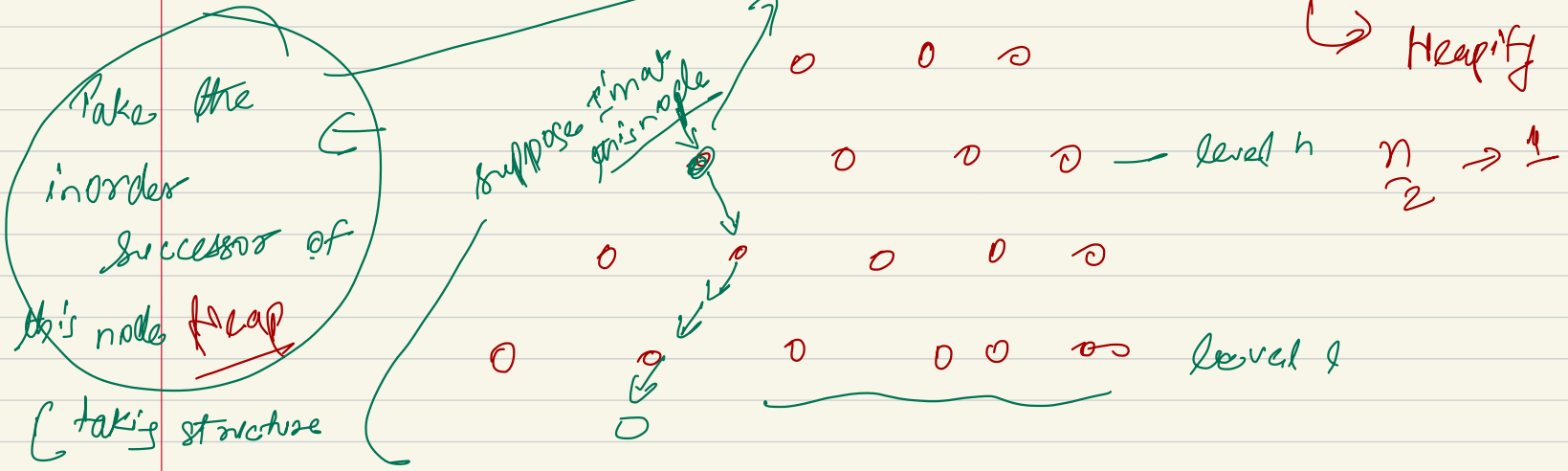
Build Heap Make Heap

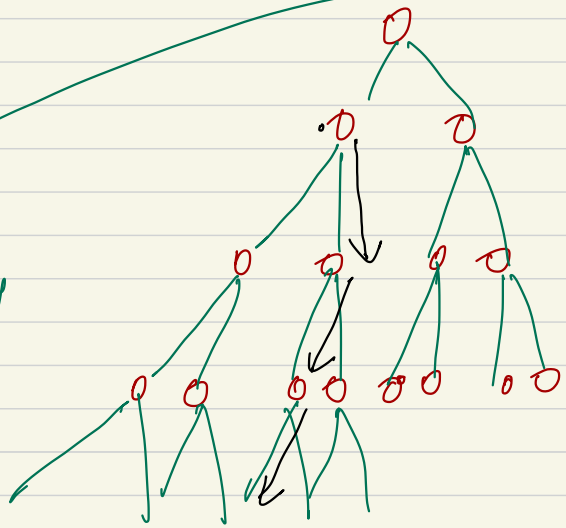
↳ bottom-up construction → O(n)



Claim:→ The length of this path (to go to inorder successor)
= $O(l-h)$

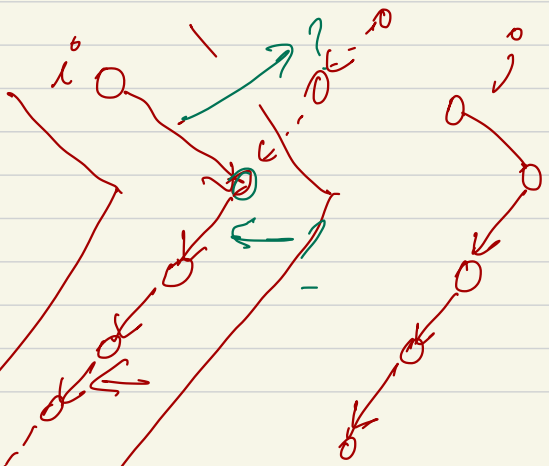
Take a right, keep taking left





for which I've to run heapify (non-leaf)

Claim 2: For each node, take the path to its inorder successor (as per BST structure)



Unique path

no edge can intersect

The paths do not have intersection for any two arbitrary node,

(no edge intersection)

Combine Claim 1 + Claim 2

makeHeap = $\sum_{\text{non-leaf nodes}}$ [complexity (Heapify) at each node]

$\Downarrow$

$\sum_{\text{non-leaf nodes}}$ length of path of inorder successor

$= O(\#\text{ of edges})$

$= O(n)$

* What is the min. time complexity for constructing a BST? worst case

(Can you do in $O(n)$?)

Given n integers, can you create a BST in $O(n)$?

↓

done by comparison

Or [illegible]

Proof by contradiction: Suppose I could do this. Once I've created a BST in $O(n)$ time using only comparisons, I can use

inorder traversal & get the sorted order.

$\Rightarrow$ Sorting algo in $O(n)$ time in comparison-based model

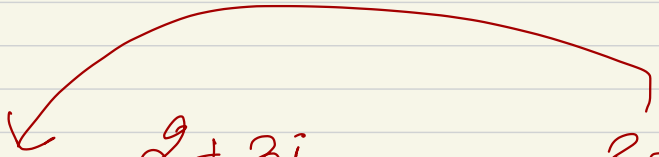
Given n integers $\rightarrow$ Construct of a BST takes $\Omega(n \log n)$ time

Q:- A complex number $z = x + iy$ $\rightarrow$ (x, y) pair of real numbers

$$|z| = \sqrt{x^2 + y^2}$$

You are given n complex numbers with each x, y being an integer from $[-n, n]$

Propose an efficient algo to sort this array of complex numbers wrt. the magnitude of the elements.



$2+3i$ $\quad$ $3+i$ $\quad$ $10+2i$

$\sqrt{13}$ $\quad$ $\sqrt{10}$ $\quad$ $\sqrt{104}$

Sort wrt $x^2+y^2 \in [0, 2n^2]$ $\rightarrow$ radix Sort

$x_i^2 + y_i^2$

x_0, y_0; $x_1^2 + y_1^2$

x_2, y_2; $x_2^2 + y_2^2$

Sort on the magnitude^2

Counting Sort

Complexity = $O(n)$

Cond^n: range $= O(n)$

range = $[0, 2n^2]$

$\lceil \sqrt{800} \rceil = 29$

$n = 20$

range $[0, 800]$

Or I use 2 digits?

x k-ary digits

$a_1, a_0 \in [0, 28]$

0 to 800

$a_1 \cdot 29 + a_0$

$O_2 2n^2$

$x = O(n)$

Use only 2 digits

$a = a_1 x + a_0$

x ?

$$x = \left\lceil \sqrt{2n^2 + 1} \right\rceil$$

Every element can be written as

$a = a_1 x + a_0$

Sort wrt a_0 then a_1

Radix Sort

$a = (a_1, a_0)$

You're given a Heap (max-Heap)

What's the complexity to find 2nd largest element?

Sol"1.. Delete the max element X

(exchange first & last)

↓ call maxHeapify

↓

findMax

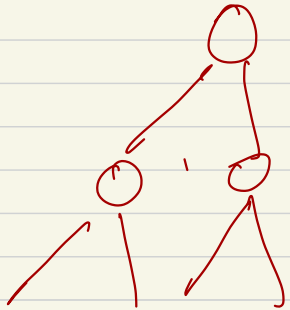
$O(\log n)$ algo

Can you do better?

Sol"2. Max (second, third)

max(second, third) = 2nd largest element

elements are unique



O(K log n) :→

maxDelete : K times

K-1 times

+ findMax

O(K²) :-

Find Kth largest element :→

Sol¹ : O → log K

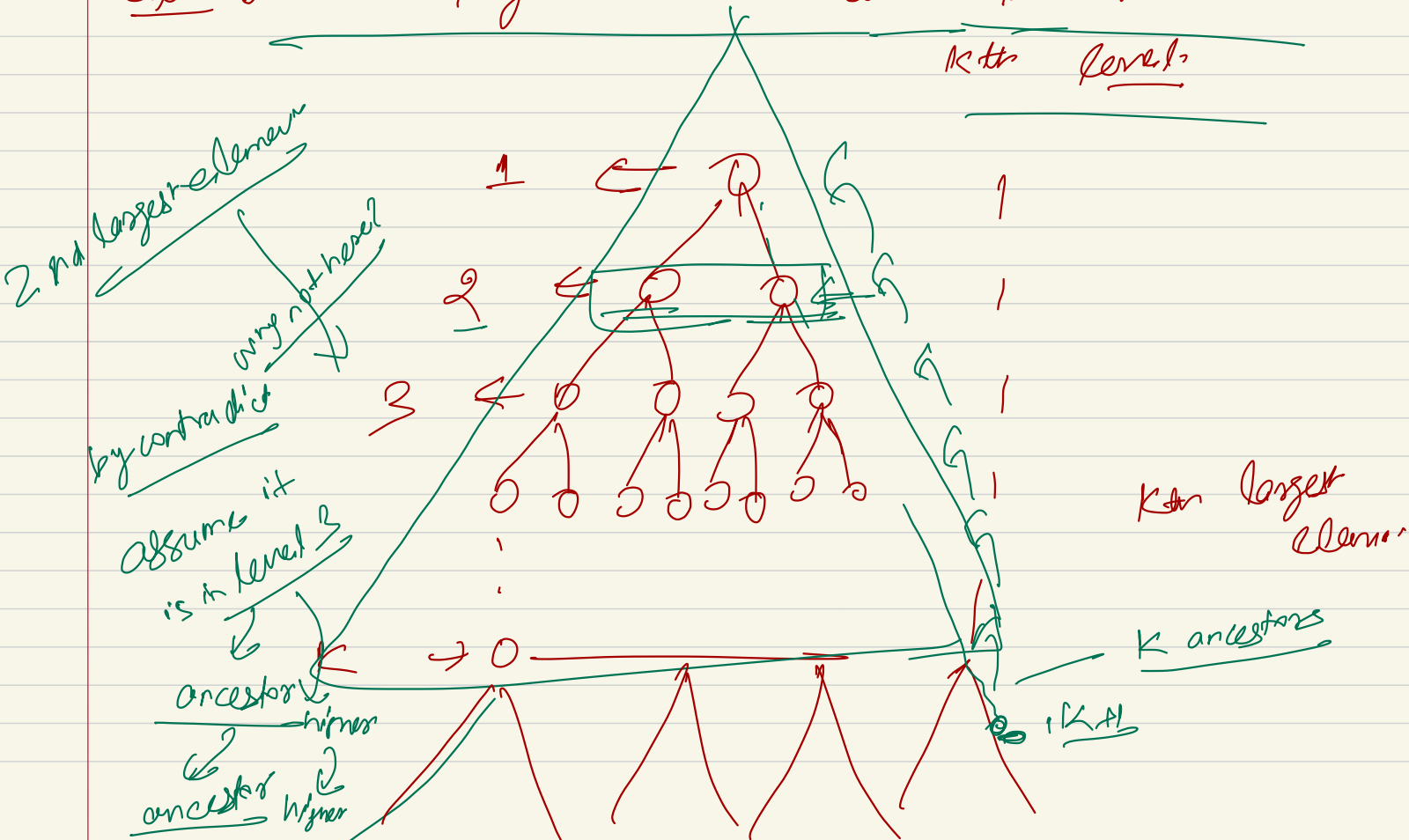
Sol² : n log n

Sol³ : K²

Sol⁴ : (K log n)

K << n !

Claim 6 → Kth largest element cannot lie below the Kth level.



within 2^k nodes

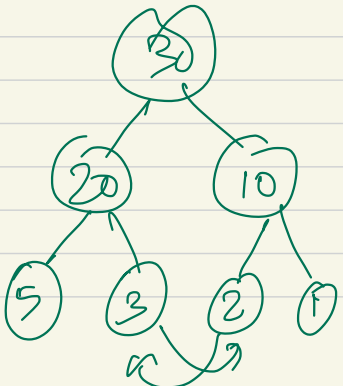
Think as if you've a heap of 2^k nodes

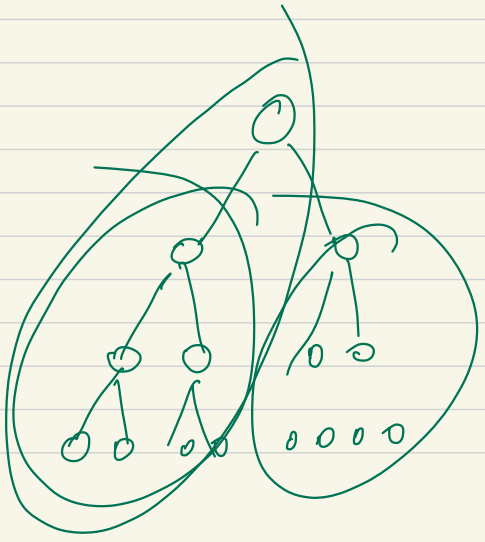
$k \log n = k \log (2^k)$

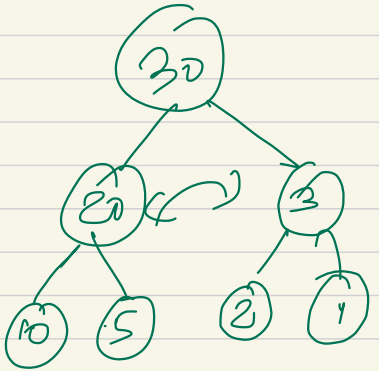
$= k^2$

(independent of n)

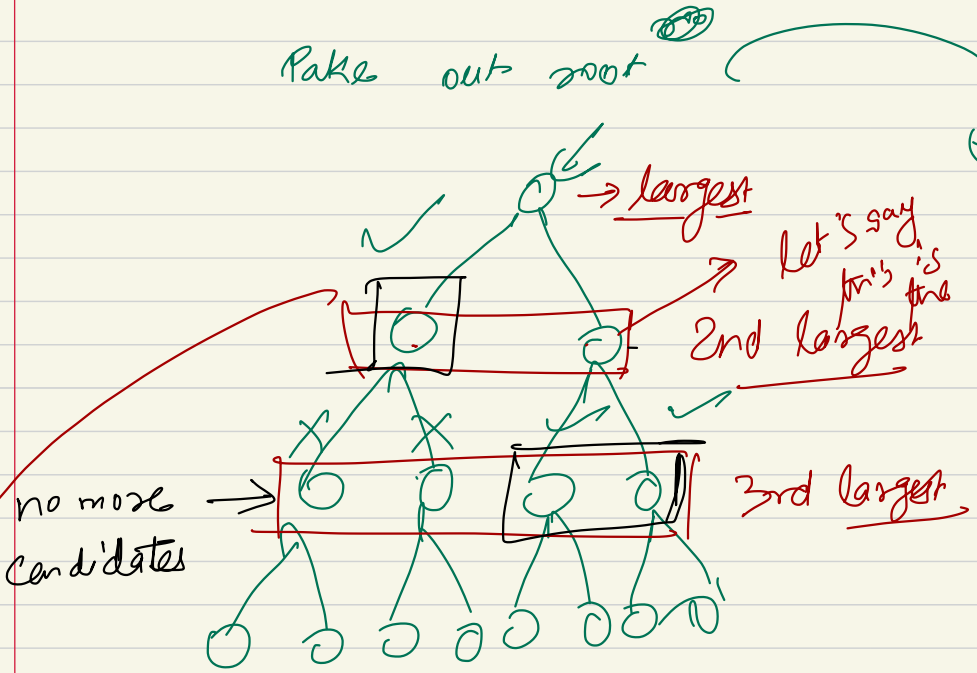
Soln 3. $O(\log k)$







$O(k \log k)$ algo



Create another heap

Take max & insert two children of root & do it recursively

1st largest — 1 candidate
2nd largest — 2 candidate
3rd largest — 3 candidate

4th largest → 4 candidate

find max

findMax

No heap

1 2 3 ... K

$O(n) + O(2) + O(3) \cdots O(K)$

$= O(K^2)$

Use a Heap

- Insert 2 elements in heap
- DeleteMax

loop → K times

size = $O(K)$

$\log K$

$K \log n \rightarrow K^2 \rightarrow = O(K \log K)$ algo