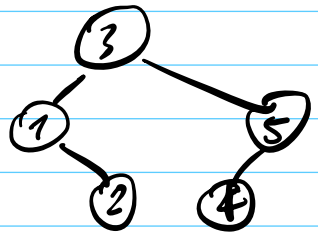
0. Warm-up

a. **Search Tree property**

b. B.S.T n nodes. which cannot be its height?

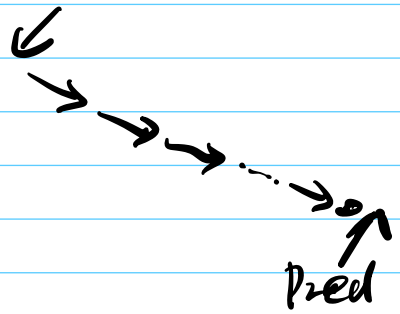
A. 1 B. $\log n$. B. $n/2$. D. $n-1$

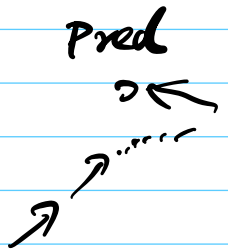
c. Pred.



Pred(3) → 2
Pred(4) → 3

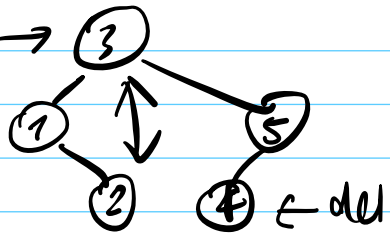
How to find pred(3). & pred(4)?





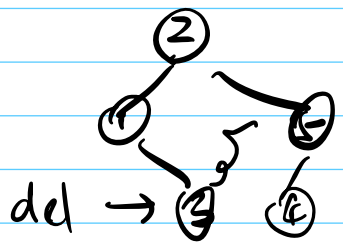
d. delete

del → 3



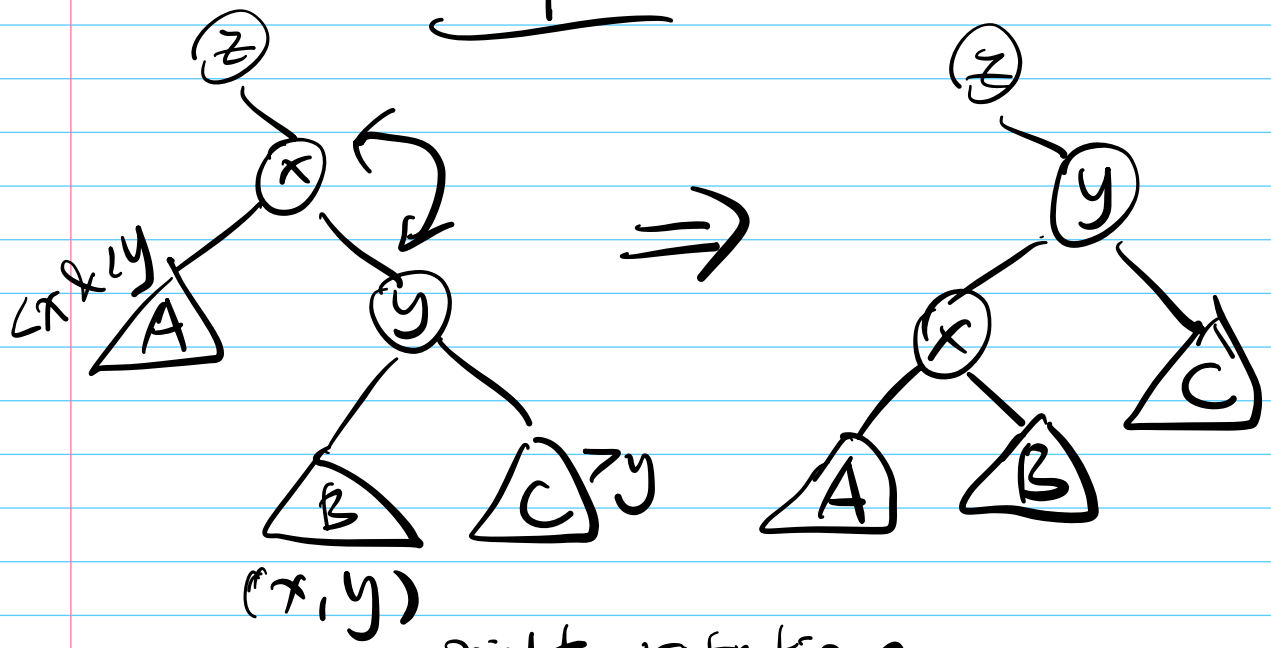
- 4 : 0/1 child easy
- 3 : find pred. 0/1 child

WHY ok to SWAP? ✓

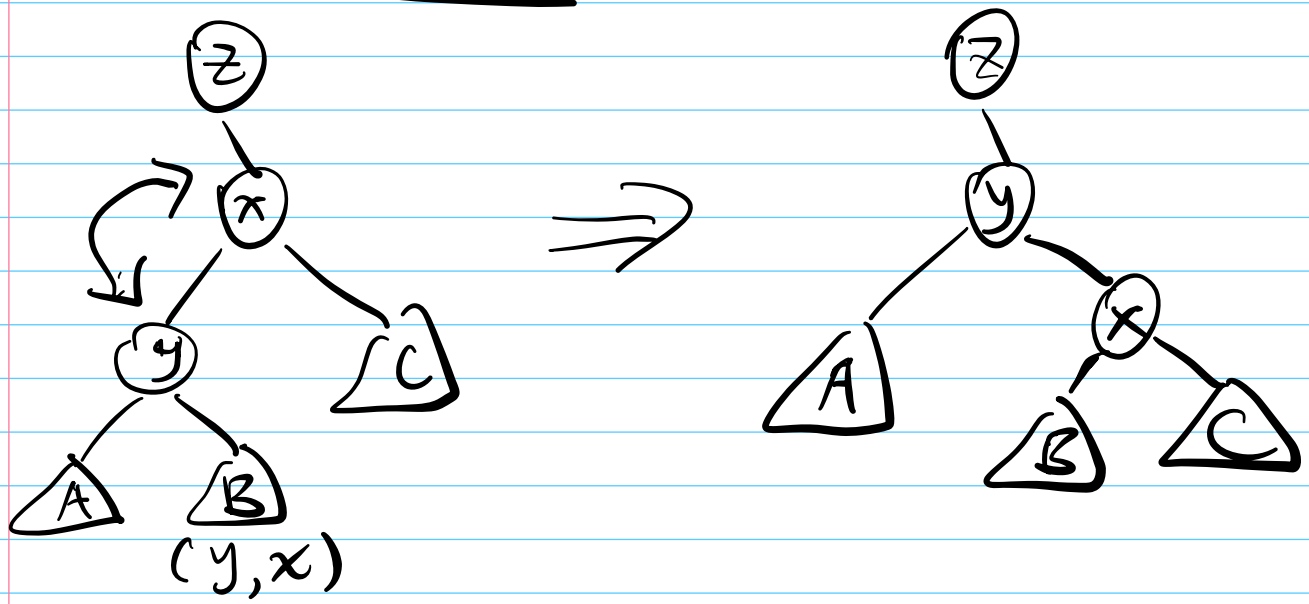


e. Rotation to maintain balanced search tree.

Left rotation



right rotation



☺ search tree property maintains.

takes O(1) time.

1. B.S.T. summary

*. Search tree property

* Search, MIN/MAX, Pred/Succ, select, rank, ins, del, all $O(\log n)$

* We use a Balanced B.S.T. if your app. requires a totally ordered rep of an evolving set of objs.

↙ if static. sorted array just fine

2. Heap (堆)

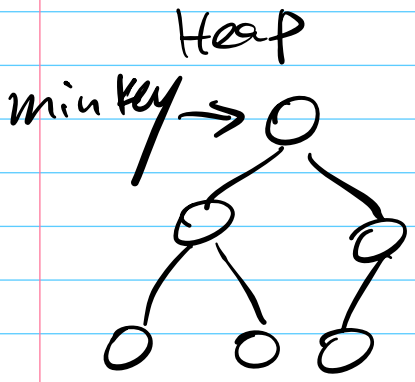
a. Basics.

what: A container for objects that have keys.

Supported op's:	Time
Ins. (add a new obj) to a heap	$O(\log n)$
Extract-MIN (remove obj. w/ min key value)	$O(\log n)$
HEAPIFY (n batched Insert)	$O(n)$
Delete.	$O(\log n)$

How to think about it?

A tree: { min key at root of every subtree
every level as filled as possible. (to keep height small)



> **Heap property:**
> every obj. x, key of x $\leq$ keys of its children

✓(min-Heap) (max-heap)

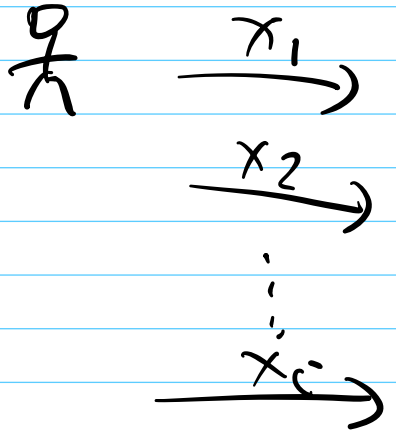
b. Applications

- Sorting: repeated Extract-Min

> **Heapsort:** 1) insert n objs. in heap
> 2) Extract-min to output in sorted order

Time: $O(n \cdot \log n)$

- Median maintenence. (Streaming)



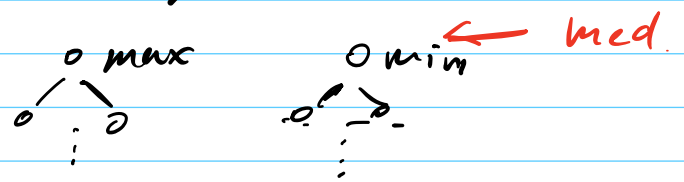
i med $\{x_1, \ldots, x_i\}$

Basic: sort then find $\Theta(i \cdot \log i)$

Contraint: at step i, $O(\log i)$

Soln: - maintain 2 heaps

- H_{max} . H_{min}

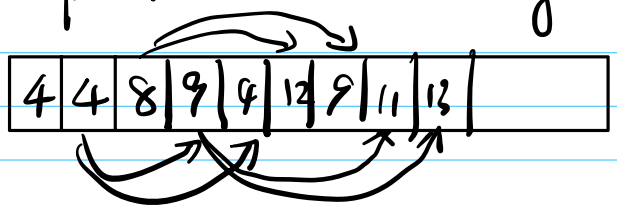


Invariant: $\frac{i}{2}$ smallest elems. $\frac{i}{2}$ largest elem's

Ex: ① maintain invariant $O(\log i)$ cost.

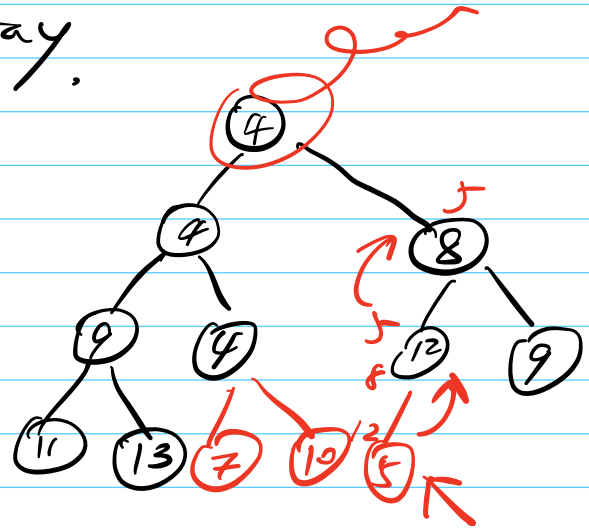
② INV: compute Med $O(\log i)$ cost.

c. Implementation by array.



children(i) = $2i$ & $2i+1$

parent(i) = $\begin{cases} \frac{i}{2} & \text{even} \\ \lfloor \frac{i}{2} \rfloor & \text{odd} \end{cases}$



Ex. 7

- Ins:

  Insert key k:

  i. stick k at the end of last level.

  ii. Bubble up k until Heap property is restored. (i.e. k's parent ≤ k)

  ✓ Bubble up must stop ?!
  & Heap property maintains.

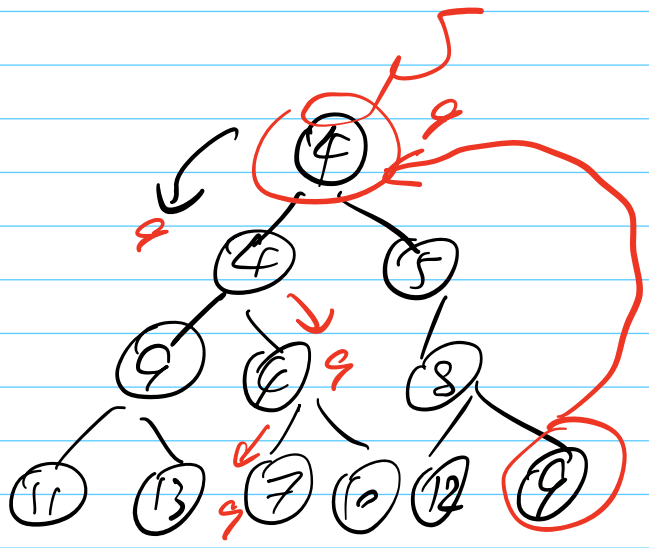
  Time: $O(\log n)$

- Extract-MIN:

  i. delete root.

  ii. move last leaf to new root.

  iii iteratively bubble down till Heap property.



→ swap w/ smaller child.

Time: $O(\log n)$.

2. Hash tables.

a. Basics

Purpose: maintain a (possibly evolving) set of objs. w/ fast lookup

Supported op's:

Insert: add new record

Delete: delete existing record.

Lookup: check for a particular record

★ All op's $O(1)$ time (+)

(+) when properly implemented. data NOT malicious

b. Applications.

- De-duplication

Given: A stream of objs.

Goal: removes duplicates.

Soln.: when new obj. k arrives.

- lookup k in hash table H.
- if NOT found, insert k into H.

Time: O(1) every check.