Adv. Data Structures

Binomial Heaps (part 2)

Recap

- HW due Friday
- One more HW after break
  then projects
- Sub on Mon. & Wed.
  after break (?)

Runtimes (Basic heaps)

Get min: $O(1)$

Insert, Delete Min $\Big\}$ $O(\log_2 n) = \lceil \log_2 n \rceil$

(but faster than BSTs)

+ decreaseKey(obj): $\lceil \log_2 n \rceil$

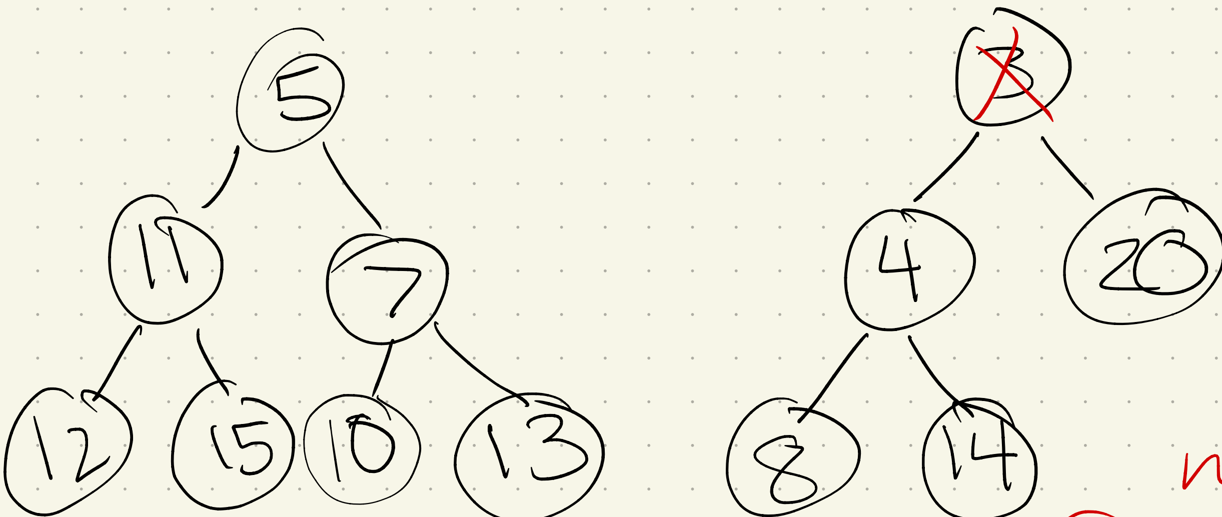
delete: $2\lceil \log_2 n \rceil = O(\log_2 n)$

(next slide)

Another: Merge(H_1, H_2):

create a new heap with all values of H_1 + H_2

How?



new heap

3

compare roots:

Best method:

insert one heap into another

↳ $O(n \log n)$

Runtime: never less than $O(n)$

Binomial Heap

Goal: Improve Merge

$$O(n) \rightarrow O(\log n)$$

at the "cost" of min

$$O(1) \rightarrow O(\log n)$$

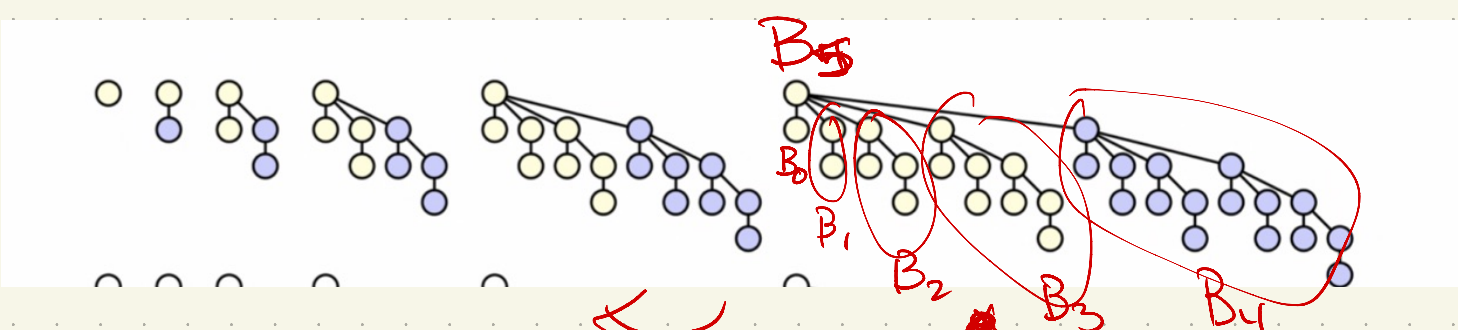
[But really not!! Stay tuned]

amortization...

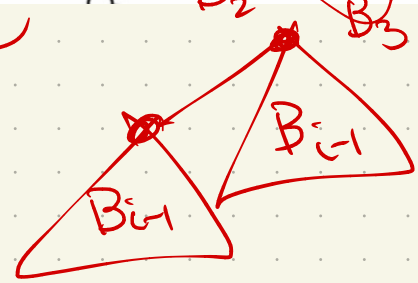
Dfn: A binomial tree, defined recursively:

Base case: B_0

B_i : two copies of B_{i-1}, one root connected as (new) child of the other



$B_i :=$



Two properties:

Size: n nodes
fit in tree of size
$\lceil \log_2 n \rceil$
(since B_k is 2 B_{k-1}'s)

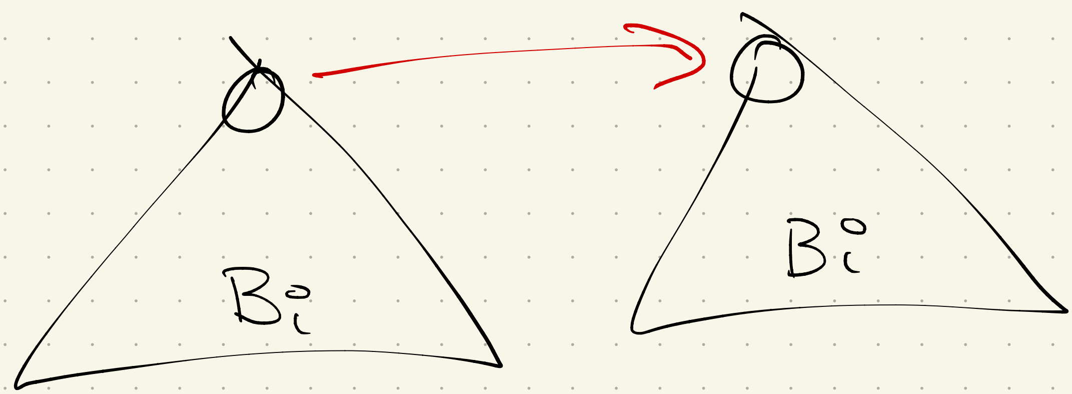
Height: B_k has height k

$\Longrightarrow$ If n values in
bin. tree, height is
$\log_2 n$

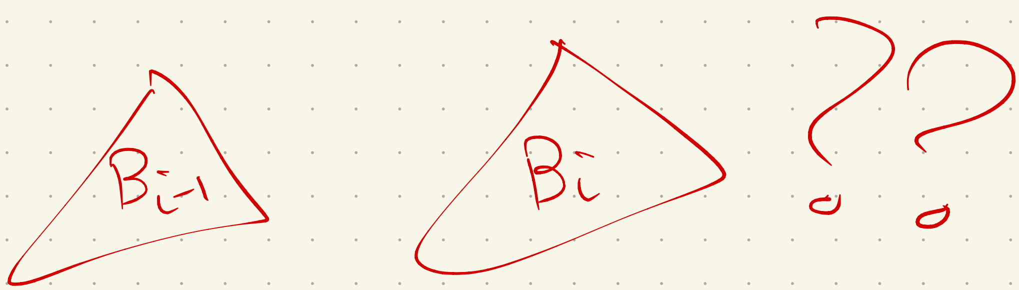
Aside: WHY??

Union can be fast!

Spps two binomial heaps of same size:



union: $O(1)$



But of course, only works if two of the same size.

Binomial Heap

- Like regular heap,
  child > parent
  (for all nodes)
- But: this is a collection of binomial trees, with at most 1 of each size

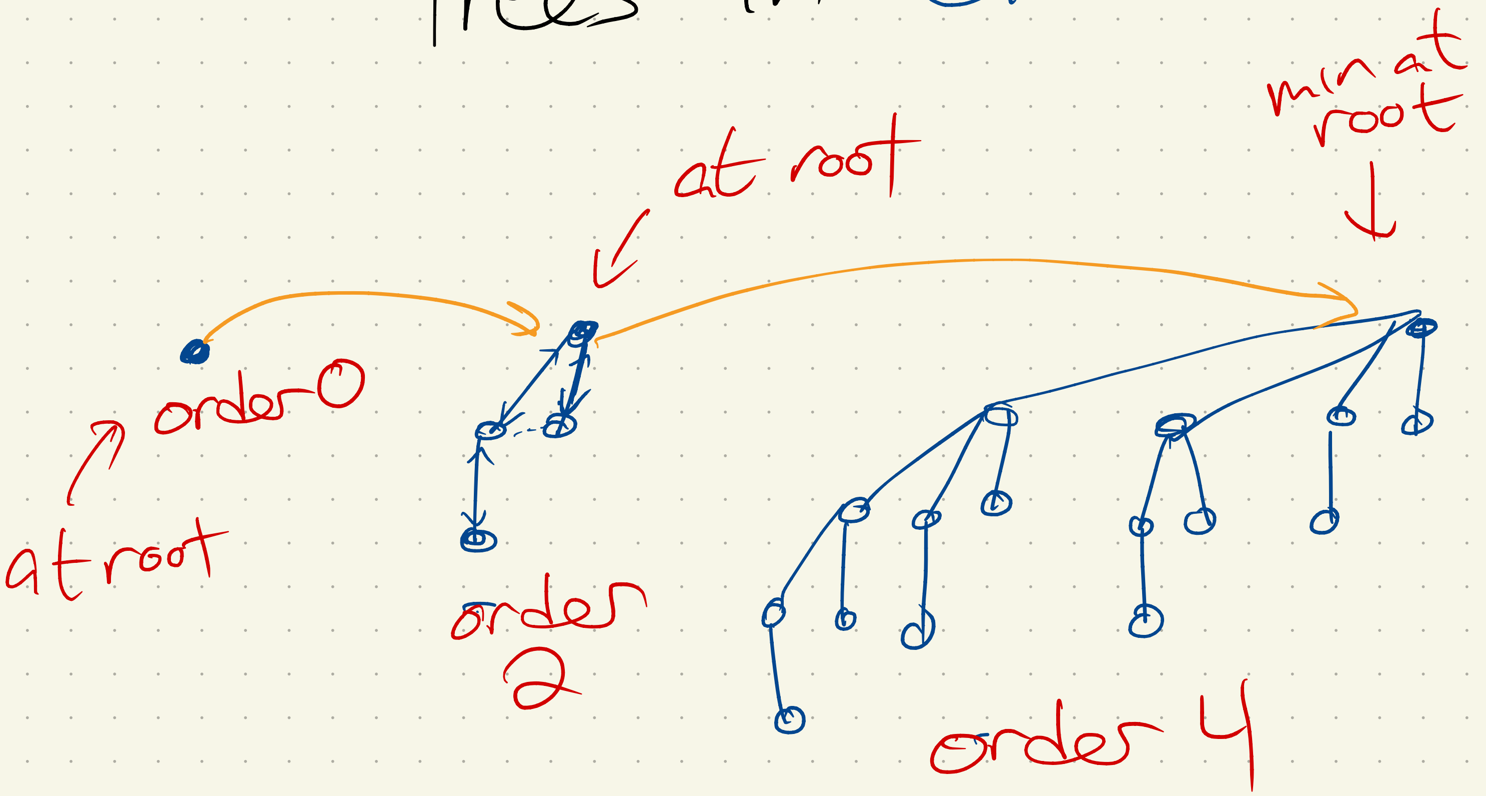
so: one B_0
one B_1
one B_2
⋮
one B_i (some i)

Index via a linked list, sorted by degree $0..i$

Ex

List in orange

trees in blue



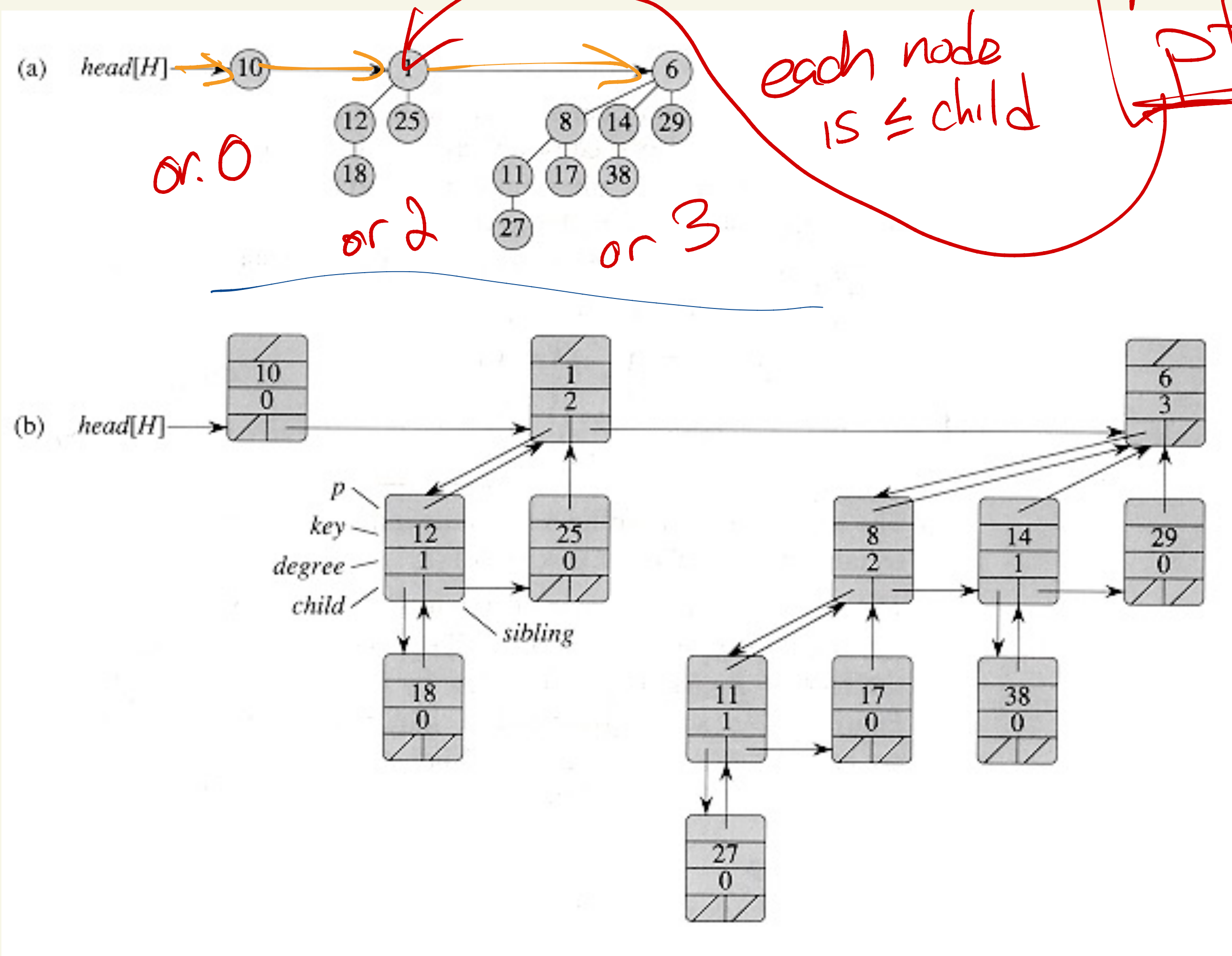
(note: no B_1 or B_3 in this example)

length of list: n nodes total:

$$n \le \sum_{i=1}^{\ell} 2^i = 2^{\ell+1} - 1 \Rightarrow \ell \approx \log_2 n$$

Example (w/ values)

min ptr



each node is ≤ child

or 0

or 2

or 3

What it really stores:

- order of heap
- data
- next list ptr
- heap ptrs: parent, left child, sibling

How to ~~search~~ write min():

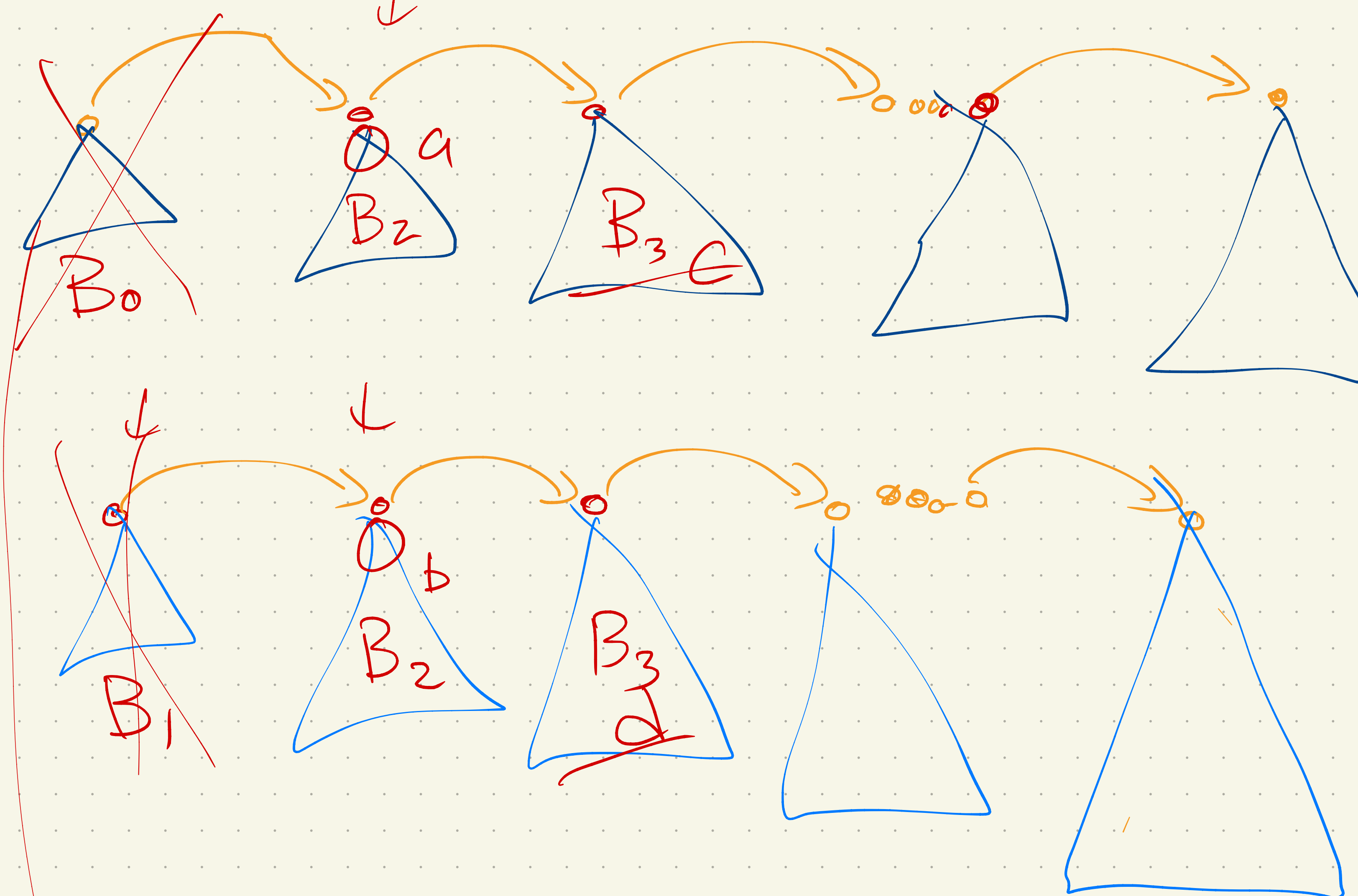
Look at the roots & take min

Runtime: "linear search", $O(\text{length of list}) = O(\log_2 n)$

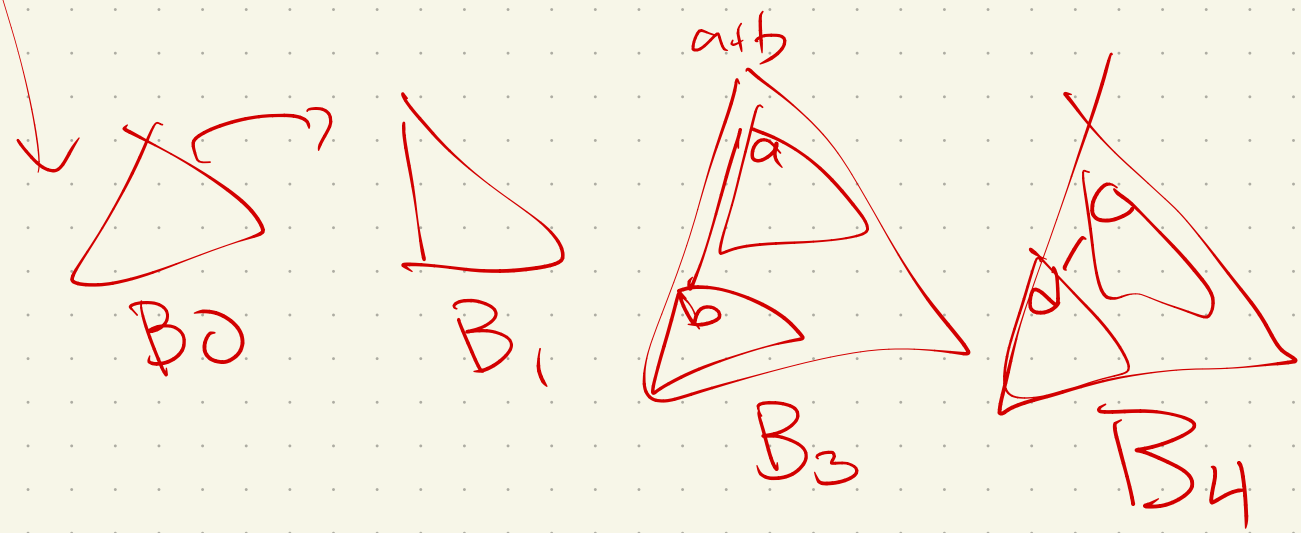
But: can keep global ptr to minimum (& just need to update it as you go)

→ Runtime: $O(1)$

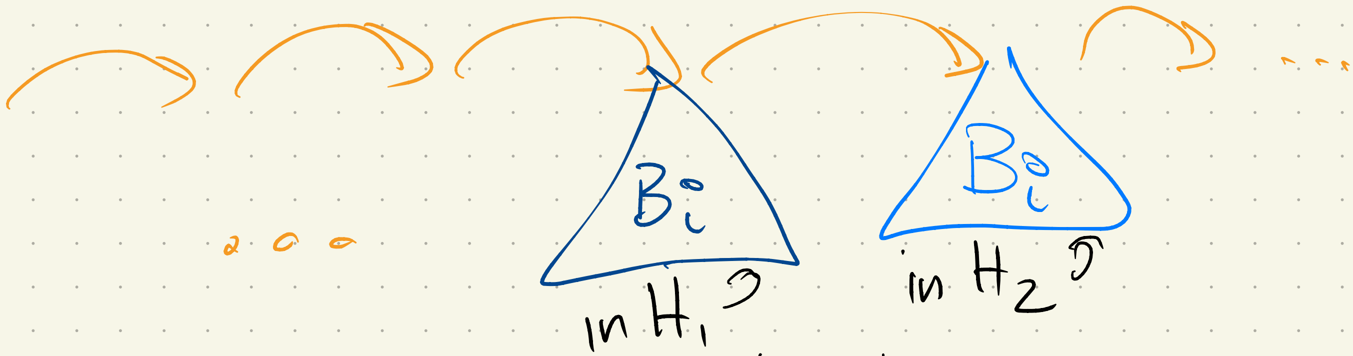
Union (H_1, H_2):



natural idea:

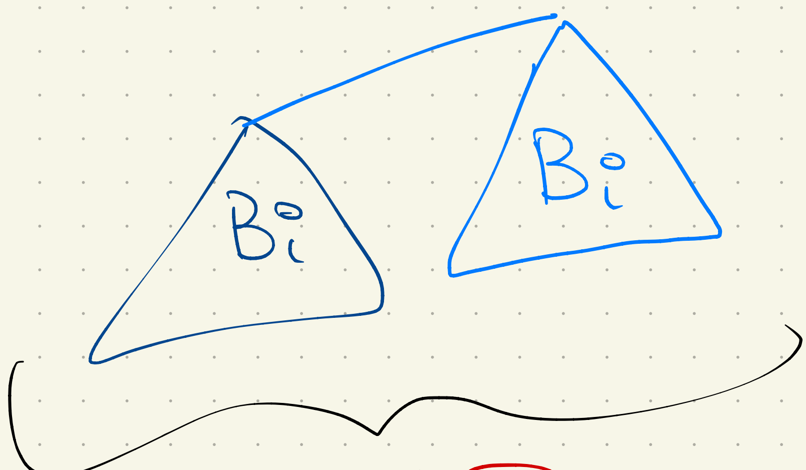


Problem: Merged list:



Could have 2 of same size!

Old trick: combine them!



now: B_{i+1}

More detail:

for i ⟵ 0 to length of merged list:

if no nodes of degree i:

$i = i + 1$

aside: promised to maintain global min

if 1 node of degree i

move on

update min

if 2 nodes of degree i

build a node of degree $i+1$

if 3 nodes of degree i

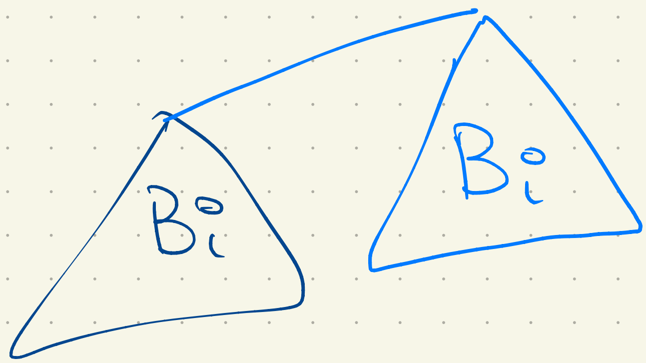
Pick 2 & make a new degree $i+1$

Leave the 3rd

Runtime of merge:

each internal merge:

$O(1)$



Overall: $2 \log_2 n$ lists

$\Rightarrow O(\log_2 n)$

Insert (x, H)

Create a new binomial heap (size 1)

$\Rightarrow \quad \rightarrow (x) \rightarrow \emptyset$

B_0 (size 1 list)

~~Merge~~ Union with H:

(keep pointer to global min)

Runtime: Worst case: $O(\log_2 n)$

adding order 0 tree

↳ order 1 → order 2 → ...

But: amortized insert is O(1) time!

Why insert is faster:

Suppose we do n inserts, & consider a merge inside our loop:

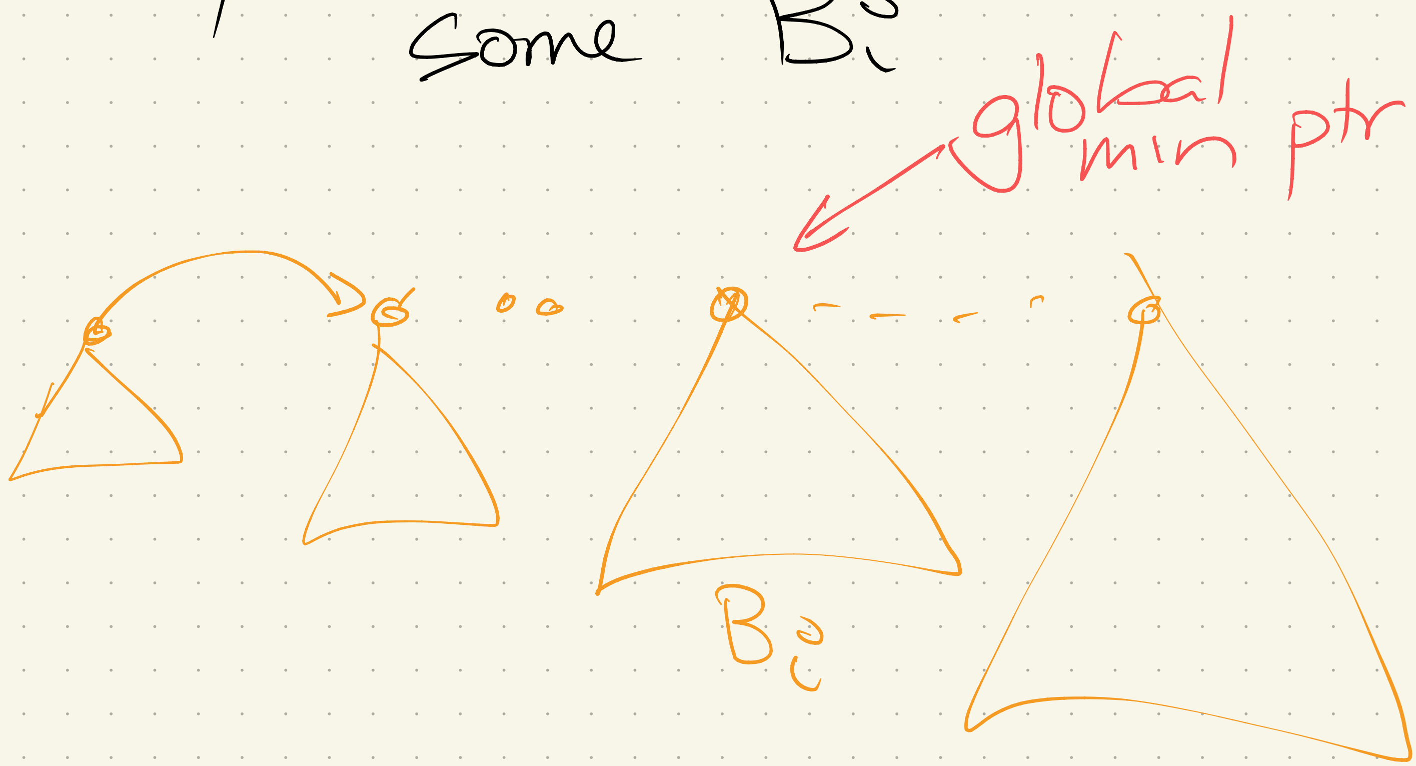
If no B_0:

If $B_0 \ldots B_i$ exist, & B_{i+1} does not:

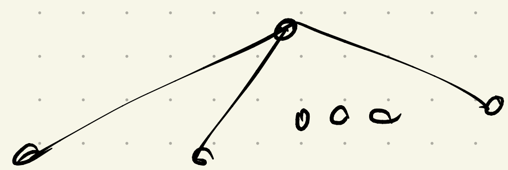
So: use accounting method!

Extract Min():

Say we delete root of some B_i

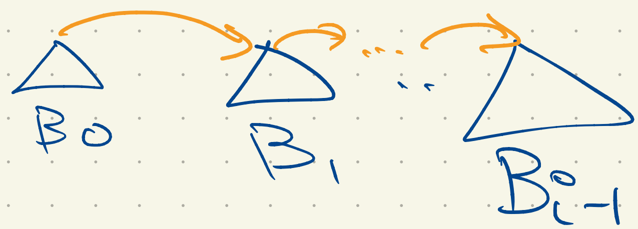


Recall: B_i is what?



& if we delete root:

So: flip children of root in B_i:



Make a heap from these & merge with rest of the heap

Runtime:

Decrease key:

Same as a regular heap:

- "Bubble" up in heap
- Might change global min

$\Rightarrow$

Delete:

- Change to $-\infty$
- DeleteMin()

$\Rightarrow$

Result:

	Heap	Binomial heap
getMin	$O(1)$	$\cancel{O(\log n)} \to O(1)$ (w/pointer)*
insert	$O(\log_2 n)$	$O(\log_2 n)$ & $O(1)$ amortized if n inserts
removeMin	$O(\log_2 n)$	$O(\log_2 n)$
decreaseKey	$O(\log_2 n)$	$O(\log_2 n)$
delete	$O(\log_2 n)$	$O(\log_2 n)$
union	$O(n)$	$O(\log_2 n)$

(* adds overhead to others, but only $O(1)$)

Only downsides: