Adv. Data Structures

Binomial Heaps

Overview

HW due at end of week (anytime Friday)

Recall: Priority Queues

An abstract data type that contains objects, each having a key.

Operations supported:

- insert(obj)
- find_min()
- delete_min()

} get min. key object

In other words, more limited than a search tree!

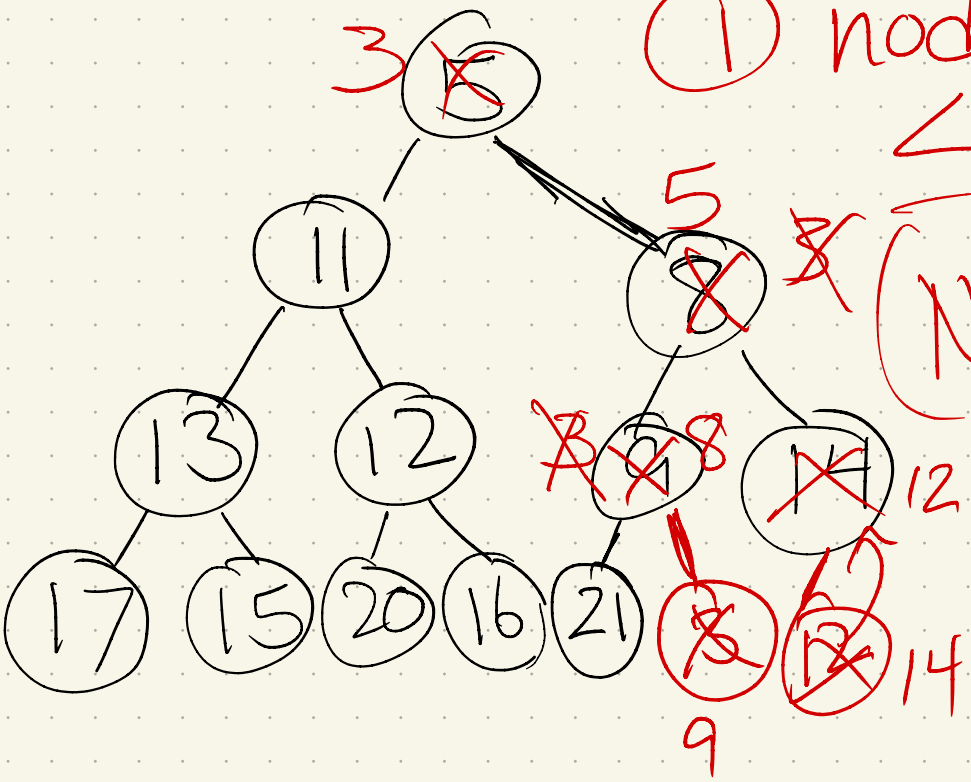
Note: Could use a BST.

Runtimes: $O(\log n)$

or B-trees: $O(\log_B n)$?

or ...

Heaps: one implementation



① node < child

(NOT a BST)

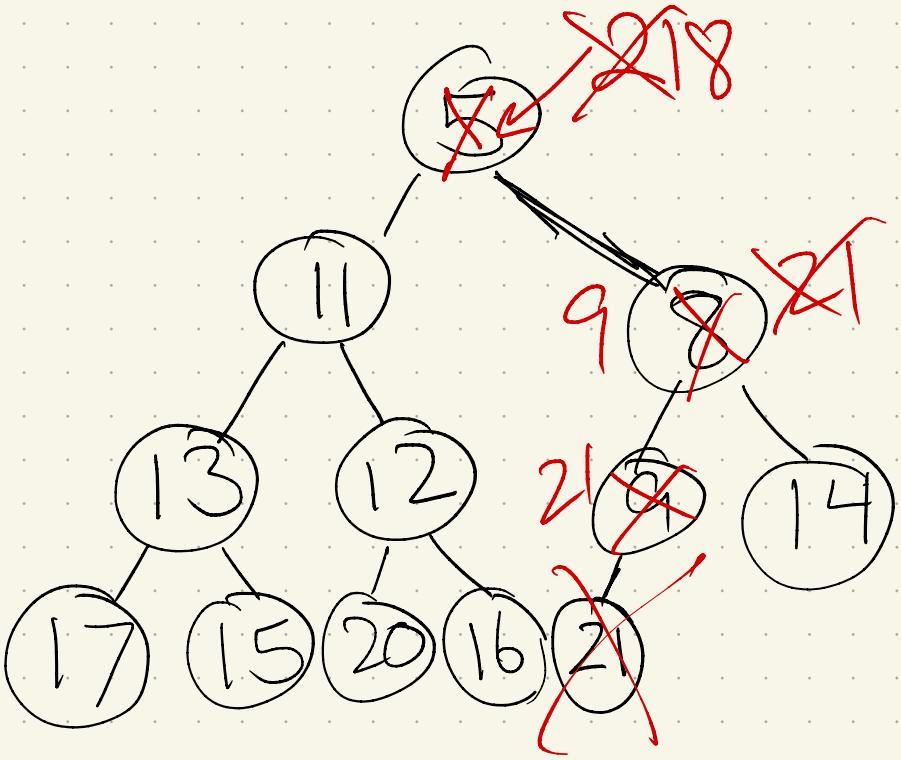
② filled level by level

⇒ height $\lceil \log_2 n \rceil$

Insert(3):

→ insert at bottom & "bubble" up

$\leq \lceil \log_2 n \rceil$ swaps



DeleteMin():

copy lowest right leaf to node, & deallocate

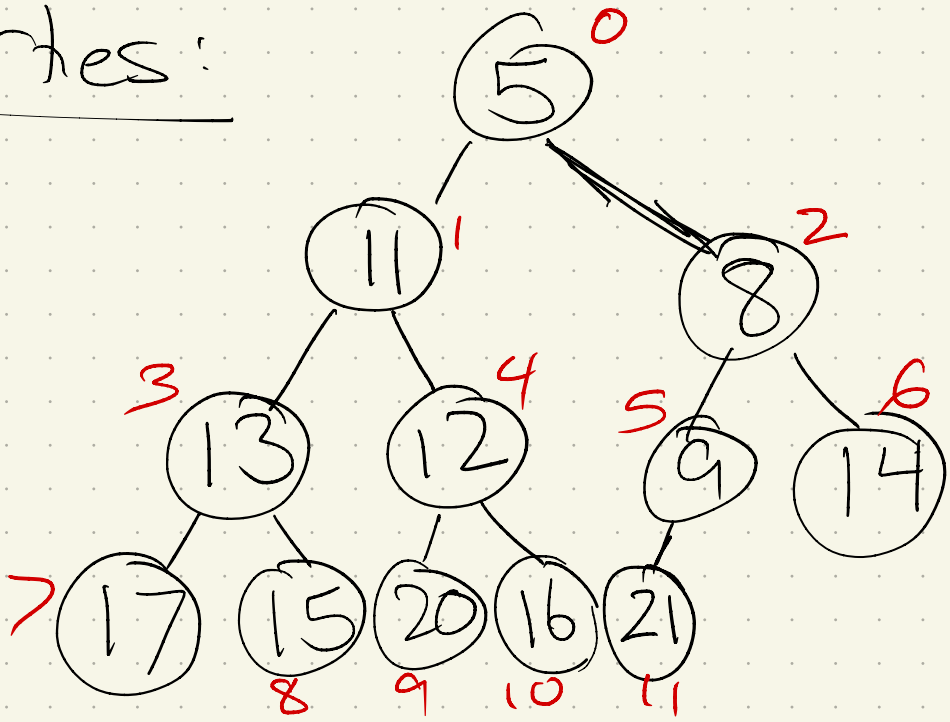
while (child is larger)

swap w/ bigger child

↳ "bubble" down

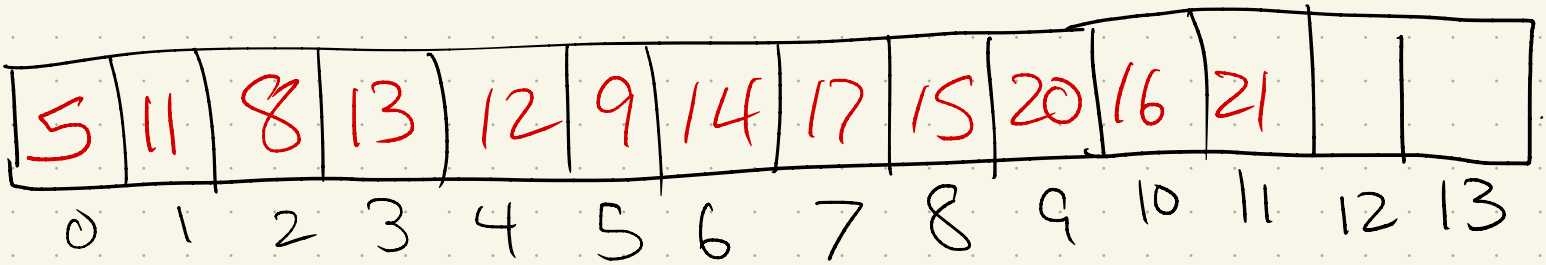
$\lceil \log_2 n \rceil$

Nice Properties:



Height: actually $\log_2 n$ (no constant!)

Layout/Space:



no pointers!

left(v) = 2v+1

right(v) = 2v+2

Runtimes (Basic heaps)

Get min: $O(1)$

Insert
DeleteMin } $O(\log_2 n)$
$= \lceil \log_2 n \rceil$

(but faster than BSTs)

+ decreaseKey(obj):

$\lceil \log_2 n \rceil$

delete: $2\lceil \log_2 n \rceil$

$= O(\log_2 n)$

(next slide)

Adding operations:

Often a heap also supports:

- decreaseKey: bubble up

  Runtime: just like delete $\lceil \log_2 n \rceil$

Note that this also gives us delete:

Given ptr to node, set it to $-\infty$

Bubble up (call decreaseKey)

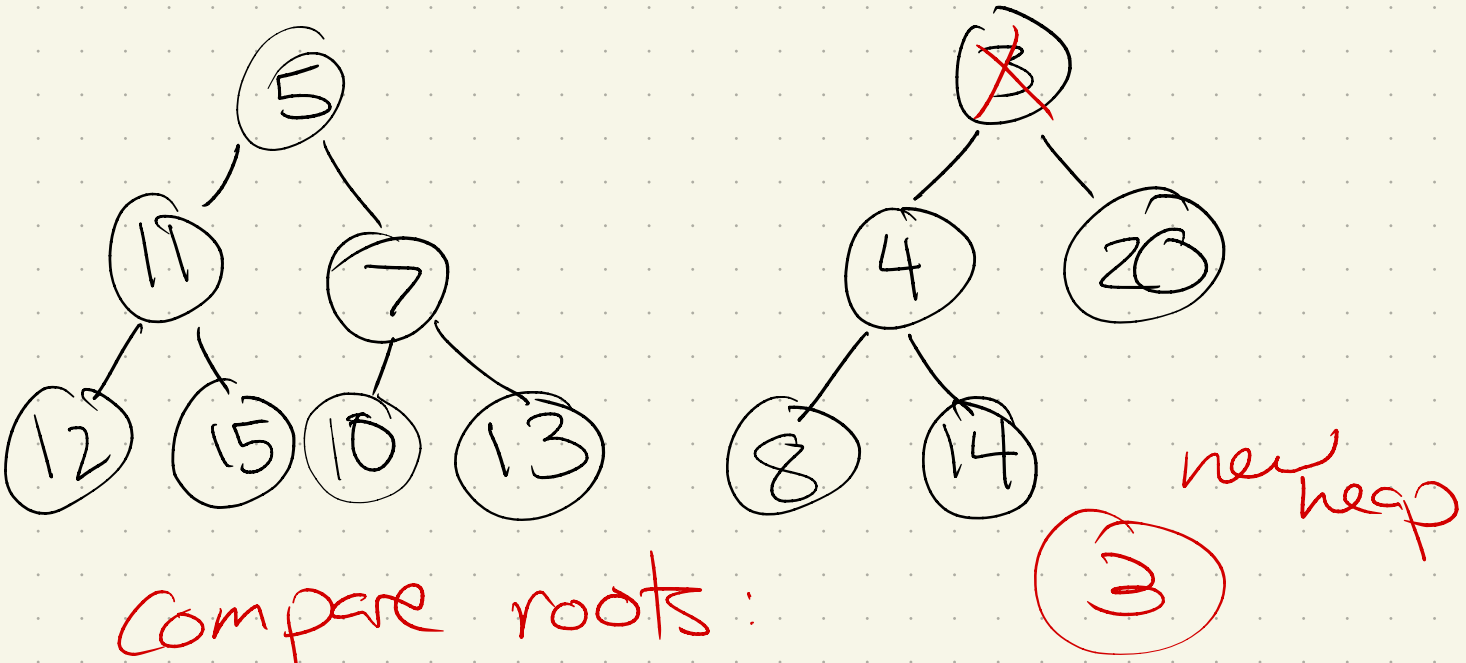
+ deleteMin

Runtime: $2\lceil \log_2 n \rceil$

Another: Merge(H_1, H_2):

Create a new heap with all values of H_1 & H_2

How?



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

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

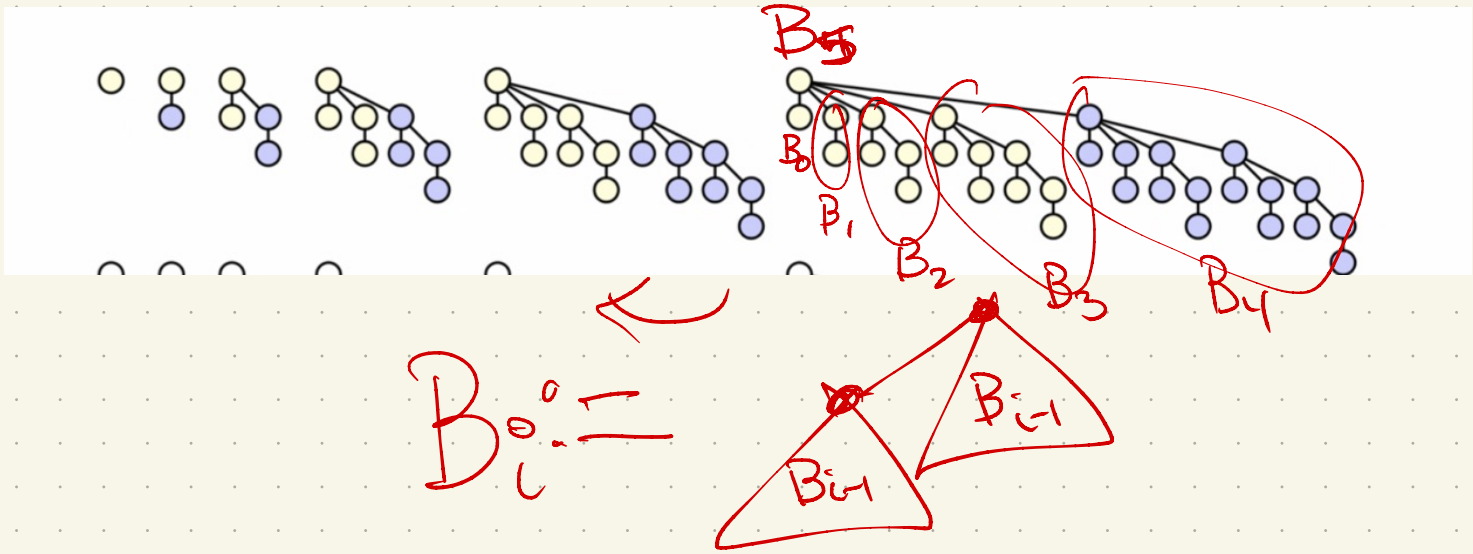
[But really not!! Stay tuned]

amortization...

Dfn: A binomial tree, defined recursively:

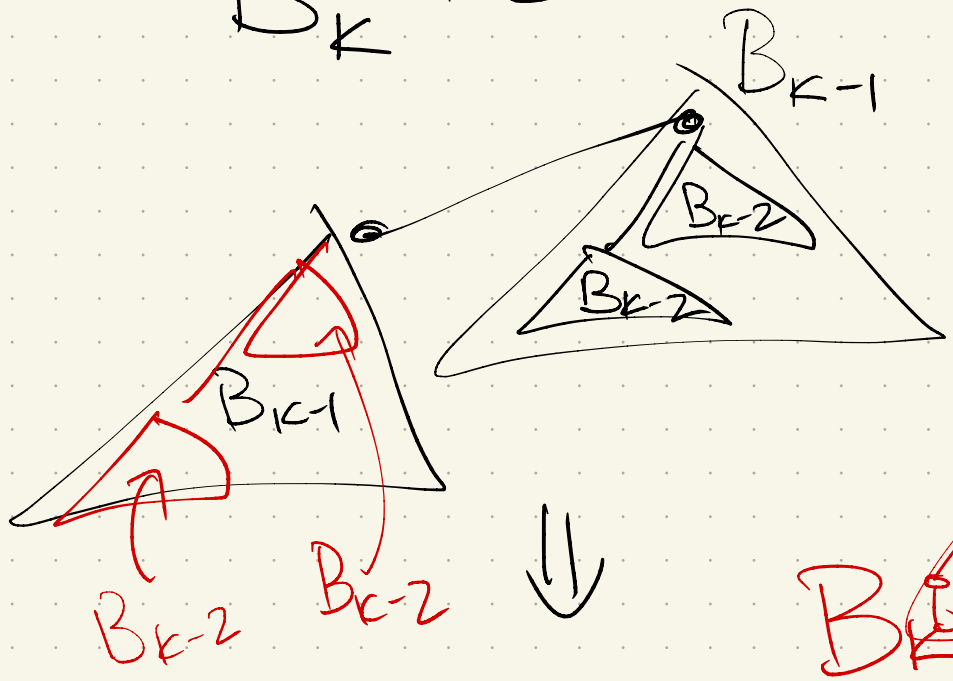
Base case: B_0

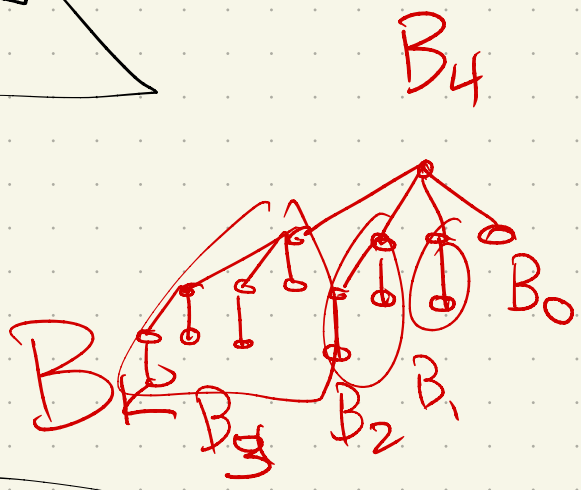
B_i: two copies of B_{i-1}, one root connected as (new) child of the other

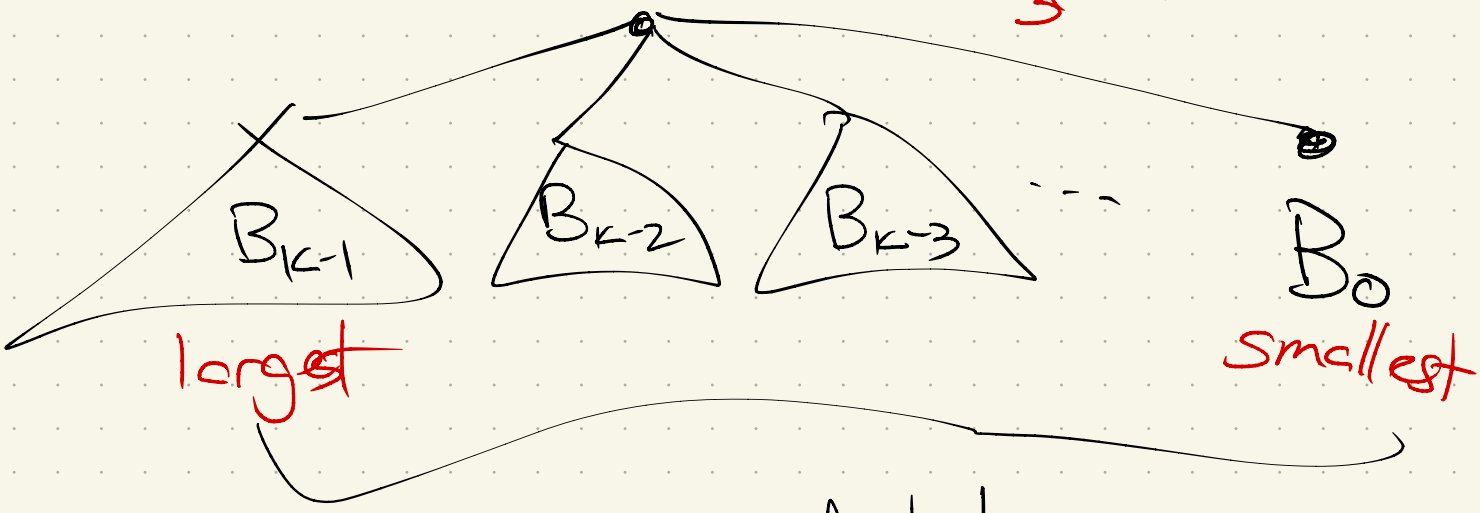


General pattern:

B_k : binomial tree of order k







k children

How many nodes?

(any 5100/3100 students from fall? :))

HW0 question last fall:

4. A binomial tree of order k is defined recursively as follows:

   - A binomial tree of order 0 is a single node.
   - For all $k > 0$, a binomial tree of order k consists of two binomial trees of order $k-1$, with the root of one tree connected as a new child of the root of the other. (See the figure below.)

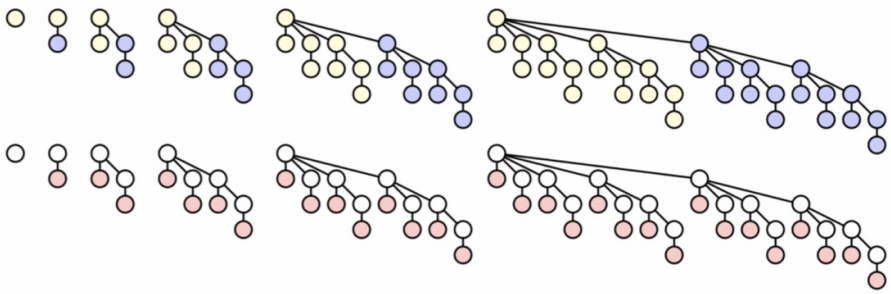
   Prove the following claims:

   (a) For all non-negative integers k, a binomial tree of order k has exactly 2^k nodes.

   (b) For all positive integers k, attaching a leaf to every node in a binomial tree of order $k-1$ results in a binomial tree of order k.

   (c) For all non-negative integers k and d, a binomial tree of order k has exactly $\binom{k}{d}$ nodes with depth d

height is k



Binomial trees of order 0 through 5.
Top row: the recursive definition. Bottom row: the property claimed in part (b).

Size of B(k): by induction!

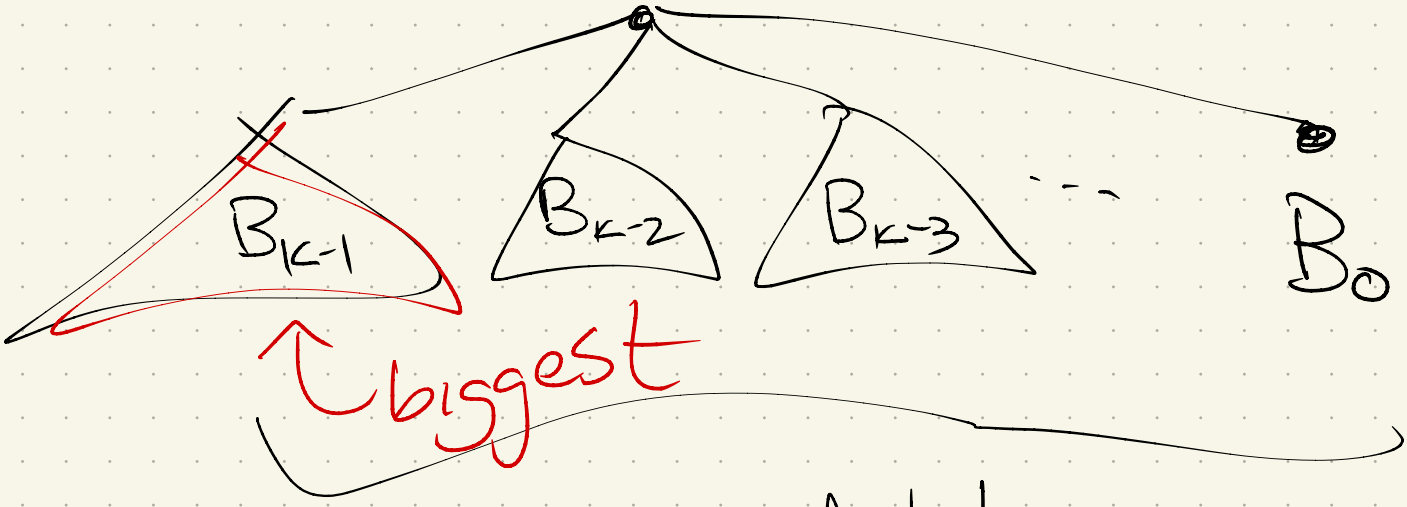
$n = S(k) = 2S(k-1)$

$+ S(0) = 1$ (B_0)

$S(k) = 2 \cdot 2 \cdot 2 \cdots 2 \cdot 1 = 2^k$

k times

Height of $B(k)$:



$$H(k) = 1 + H(k-1)$$

& $H(0) = 0$

$$\Rightarrow H(k) = k$$

& in terms of # of nodes:

$$k = \lceil \log_2 n \rceil$$

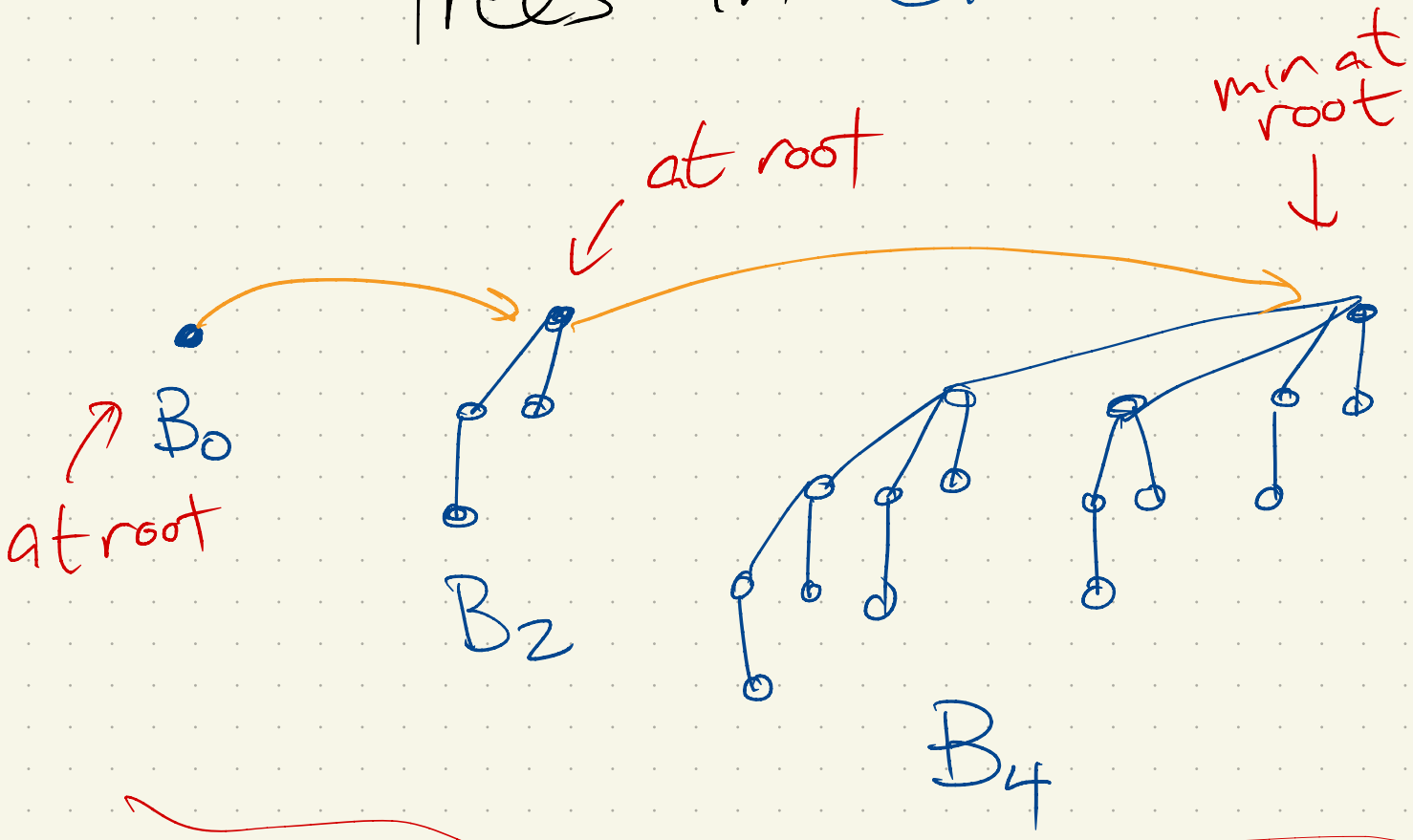
Binomial Heap

- 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:

How to search:

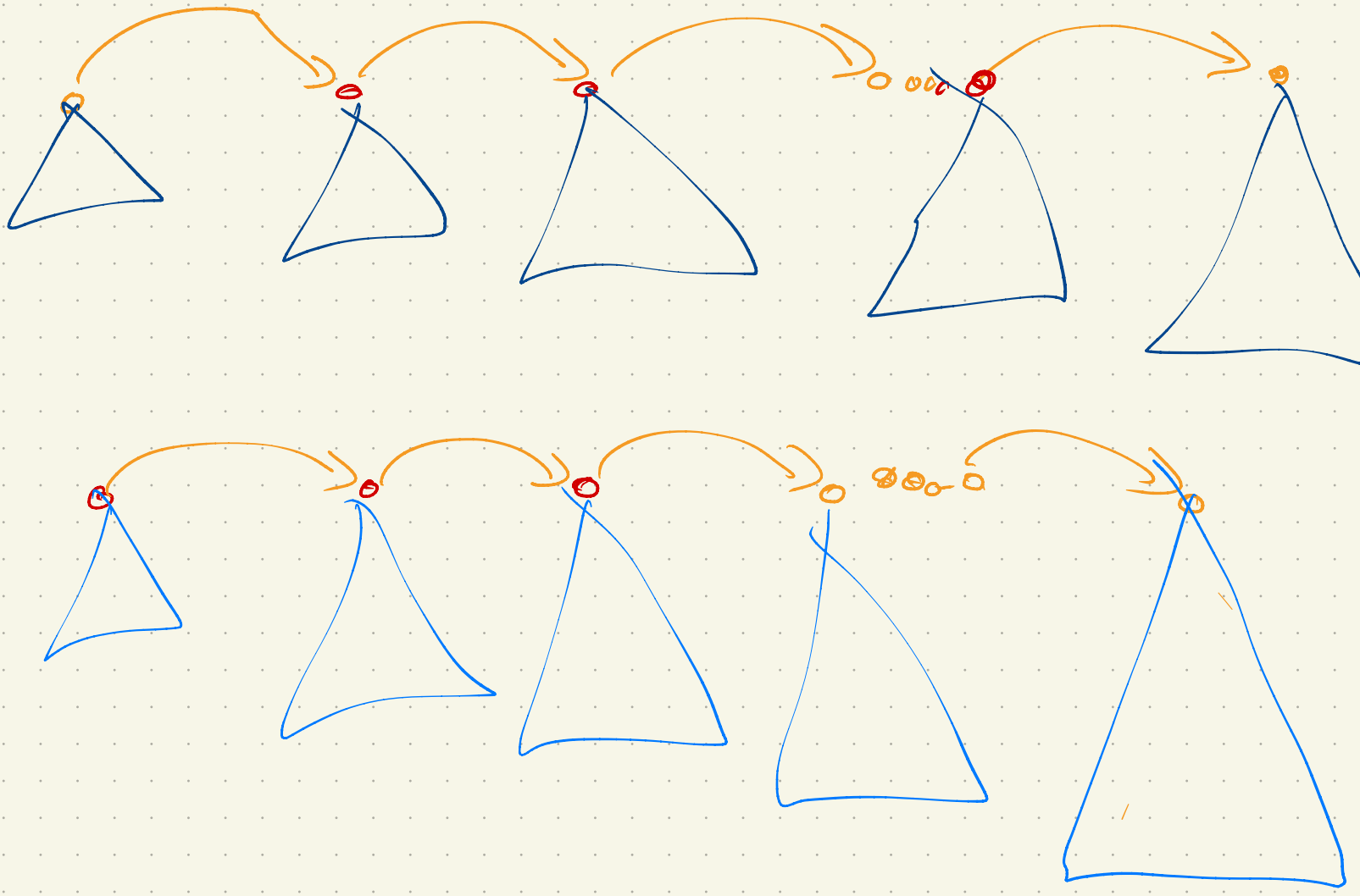
Look at the roots & take min

Runtime:

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

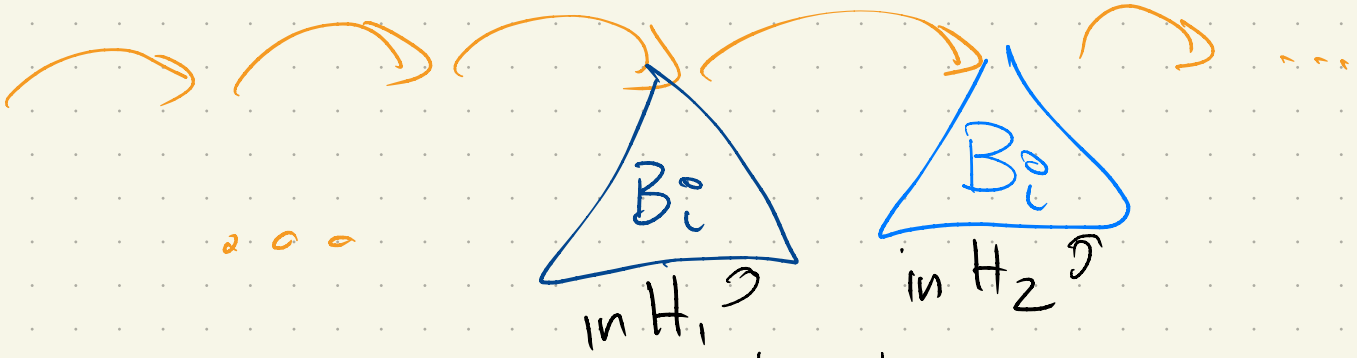
↳ Runtime:

Union (H_1, H_2):



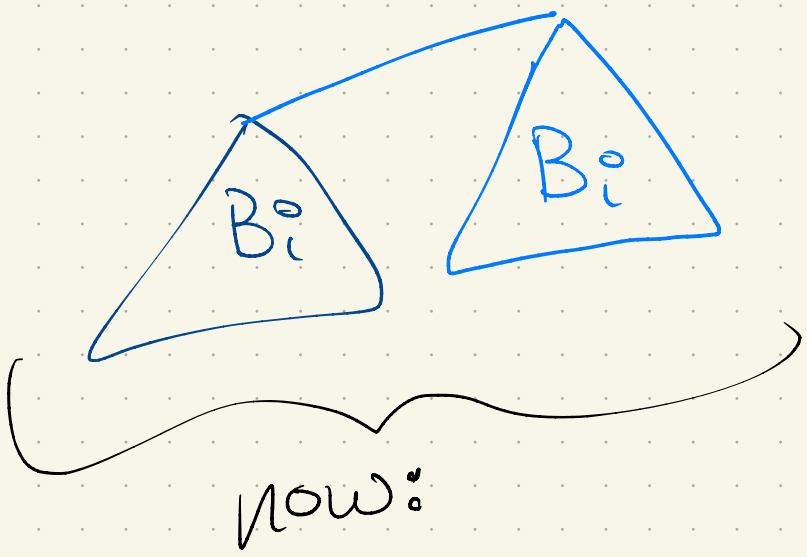
natural idea:

Problem: Merged list:



Could have 2 of same size!

Nice trick: combine them!



More detail:

for i ← 0 to length of merged list:

if no nodes of degree i:

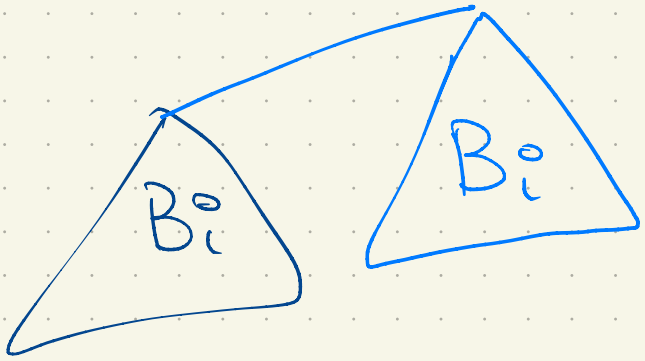
if 1 node of degree i

if 2 nodes of degree i

if 3 nodes of degree i

Runtime of merge:

each internal merge:



Overall:

Insert (x, H)

Create a new binomial heap (size 1)

B_0 (size 1 list)

Merge with H:

(keep pointer to global min)

Runtime: Worst case:

But: amortized insert is $O(1)$ time!

Why insert is faster:

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

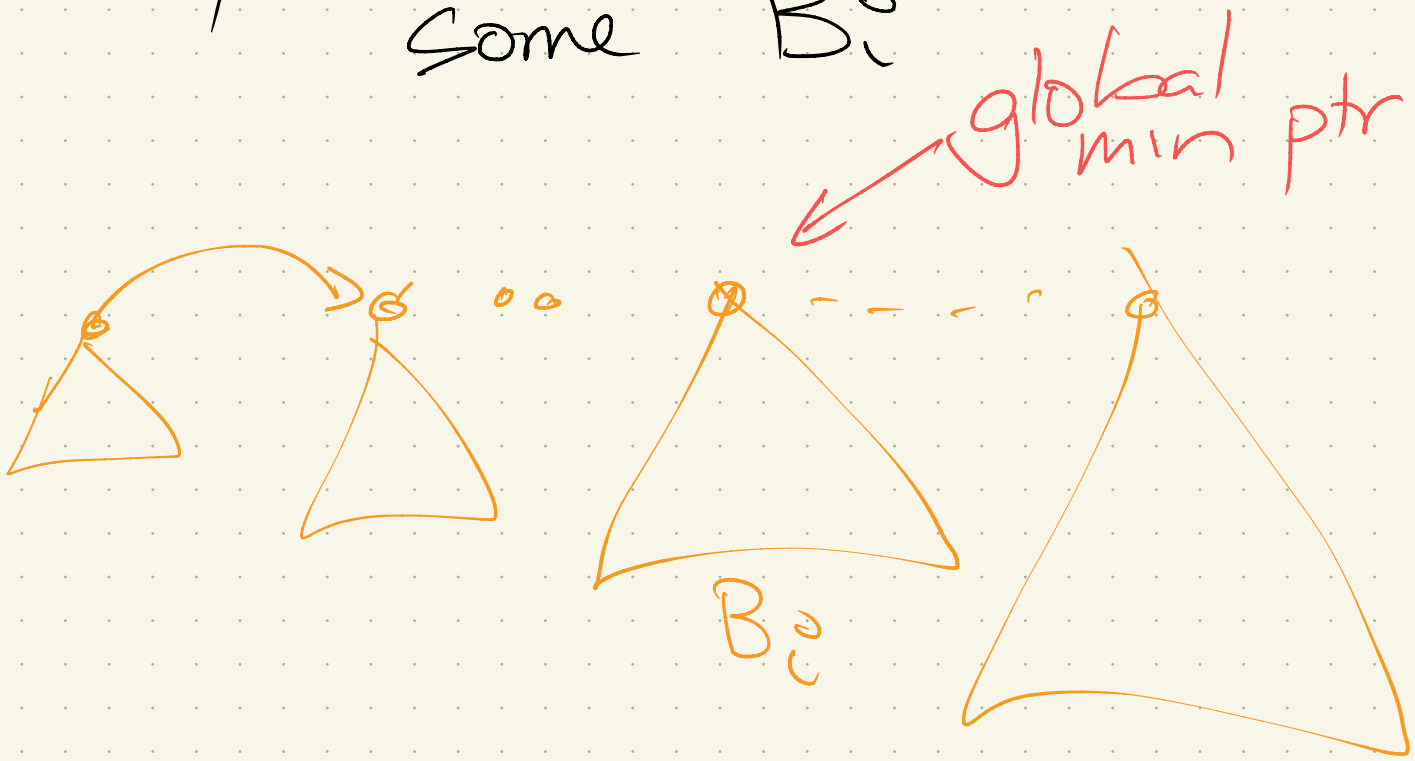
If no B_0:

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

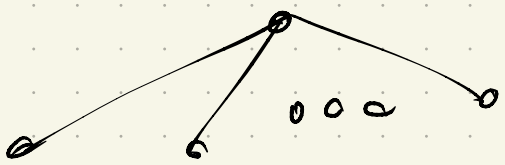
So: use accounting method!

Extract Min():

Say we delete root of some B_i:

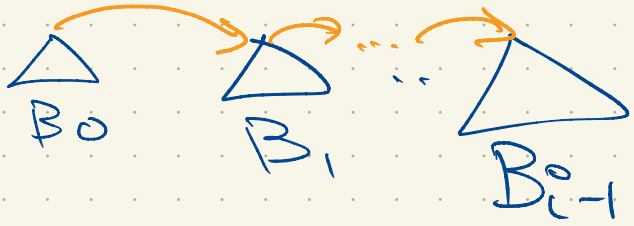


Recall: B_i is what?



& 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()

$\Rightarrow$