Priority Queues

Priority Queue ADT

Examples

- Casualty
- Discrete Event Simulation
- Shortest Paths
- Coding Algorithms
- Compression Algorithms
- Optimisation Algorithms
- Games!

Priority Queue ADT

- first() - get the smallest key-value (but leave it there)
- insert() - add a new key-value
- extractMin() - remove the smallest key-value
- decreaseKey() - reduce the key of a node
- merge() - merge two queues together

Priority Queue ADT

Key: insert, extractMin.

Best performance expected?

<u>Imagine doing sorting using a P.Q.</u>

$\Rightarrow$ n inserts, n extractMin()

$\Rightarrow$ O(n) lots of P.Q. operations

Best sorting $O(n \lg n)$

$\therefore$ insert, extractMin must be at least $\underline{\underline{O(\lg n)}}$

Sorted Array Implementation

- Put everything into an array
- Keep the array sorted by sorting after every operation

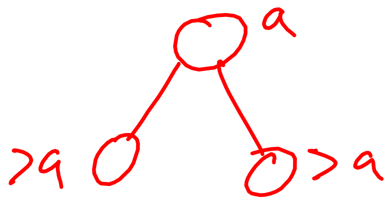
- first() — $O(1)$ lookup
- insert() — $O(n)$
- extractMin() — $O(n)$
- decreaseKey() — $O(n)$
- merge() — $O(n)$

RB Tree Implementation

- first() — $O(\lg n)$
- insert() — $O(\lg n)$
- extractMin() — $O(\lg n)$
- decreaseKey() — $O(\lg n)$
- merge() — insert all of one tree into other $O(n \lg n)$

Binary Heap Implementation

- Could use a *min-heap* (like the max-heap we saw for heapsort)



- insert()

  · Add to bottom
  · "Bubble" up
  worst case ⇒ $O(\text{levels})$
  $= O(\lg n)$

- first()

  $O(1)$ → Lookup

Binary Heap Implementation

- extractMin()   Analogous to heapsort.

  $\Rightarrow$ Extract

  $\Rightarrow$ Fix

  $\underline{\underline{O(\lg n)}}$

- decreaseKey()

  · Make change

  · Bubble

  $O(\lg n)$

- merge() $O(n \lg n)$

Limitations of the Binary Heap

- It's common to want to merge two priority queues together
- With a binary heap this is costly...

Binomial Heap Implementation

- First define a binomial **tree** Recursive definition
  - Order 0 is a single node
  - Order k is made by merging two binomial trees of order (k-1) such that ~~the smaller root~~ one becomes the new root

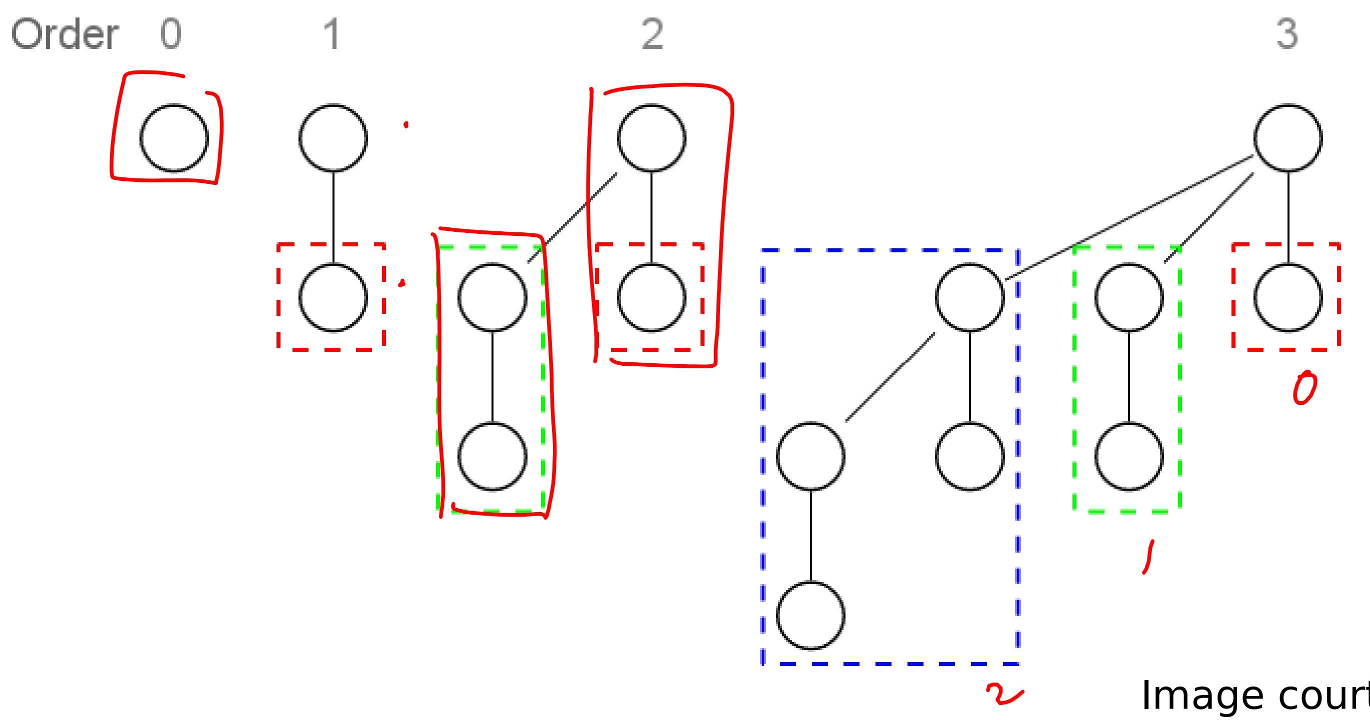
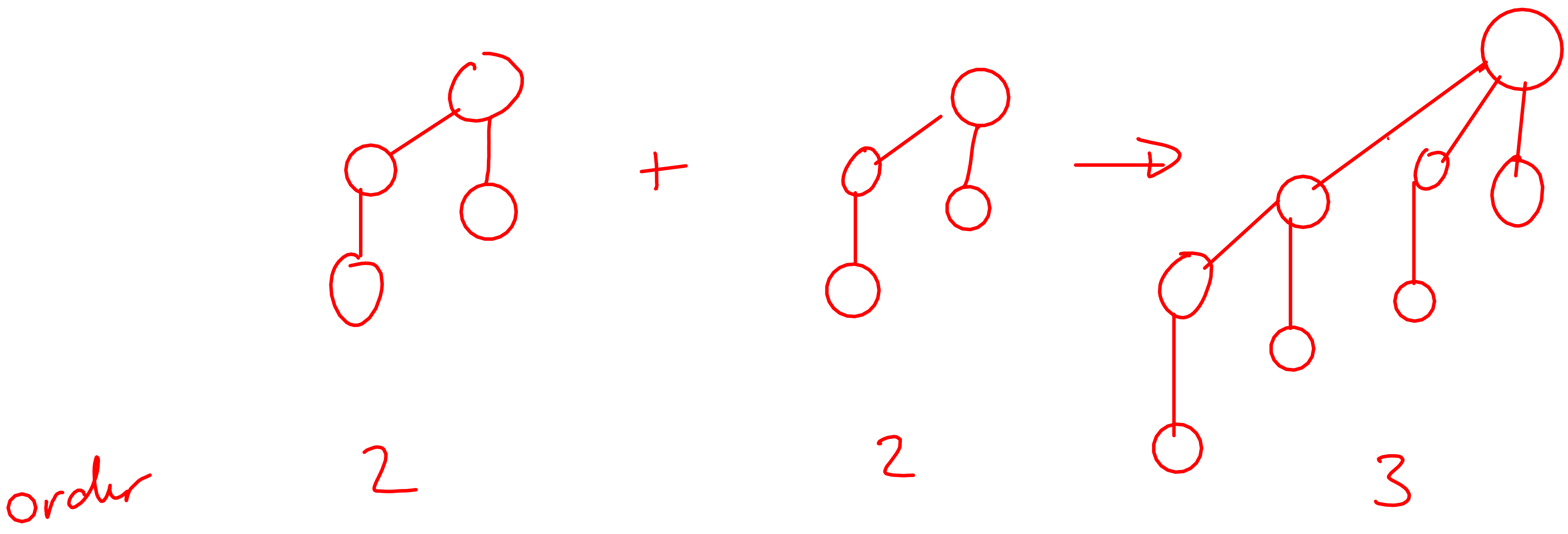


Image courtesy of wikipedia

Merging Trees

- Note that the definition means that two trees of order X are trivially made into one tree of order X+1



$\boxed{O(1) \text{ process}}$

How Many Nodes in a Binomial Tree?

- Because we combine two trees of the same size to make the next order tree, we double the nodes when we increase the order
- Hence:

order	# nodes
0	1
1	2
2	4
3	8

$$n = 2^{k}$$

order

Binomial Heap Implementation

- Binomial **heap**
- A set of binomial trees where every node is smaller than its children
- And there is at most one tree of each order attached to the root

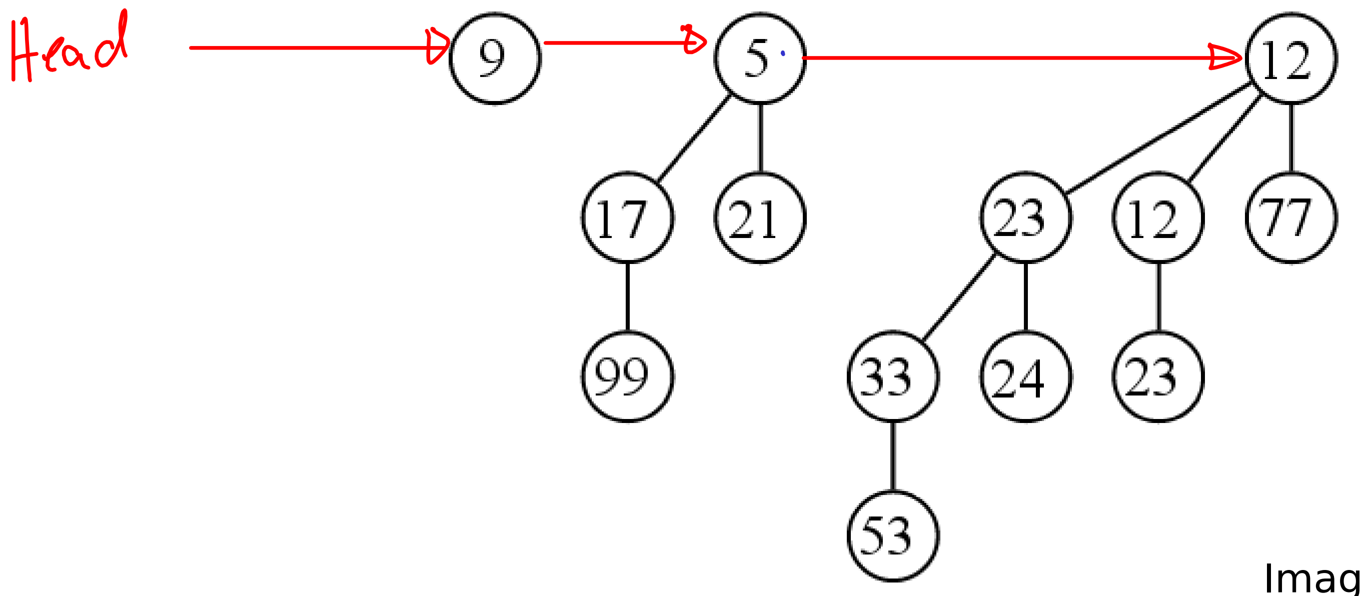
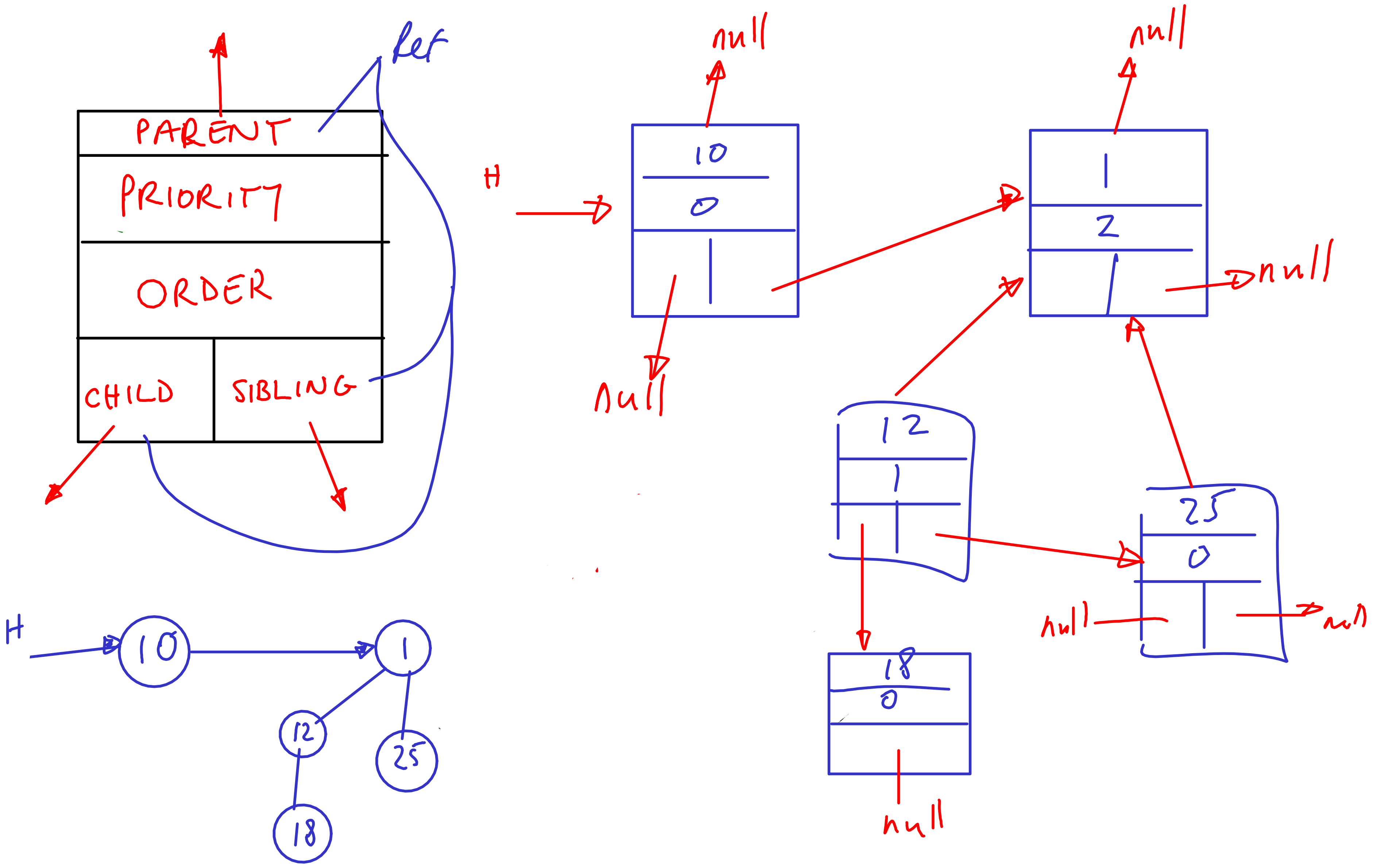


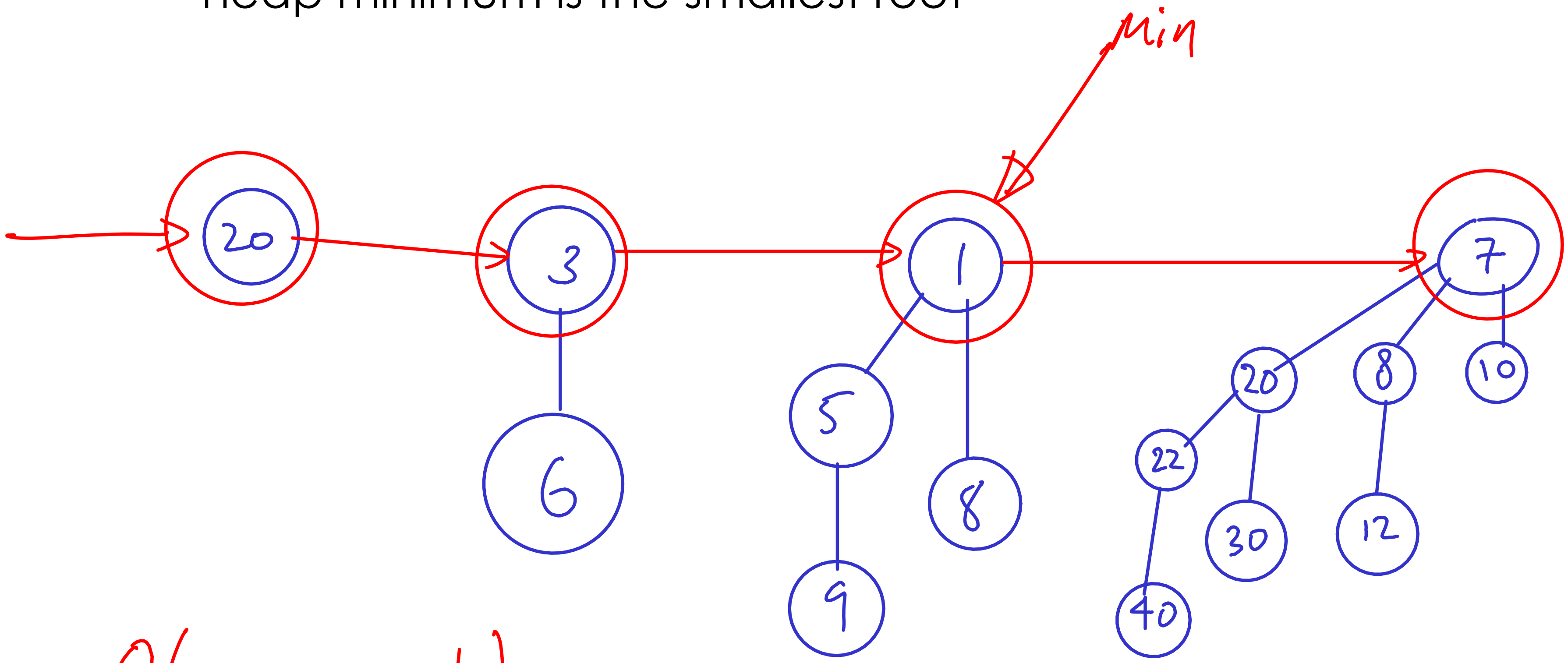
Image courtesy of wikipedia

Binomial Heap Implementation



Binomial Heaps as Priority Queues

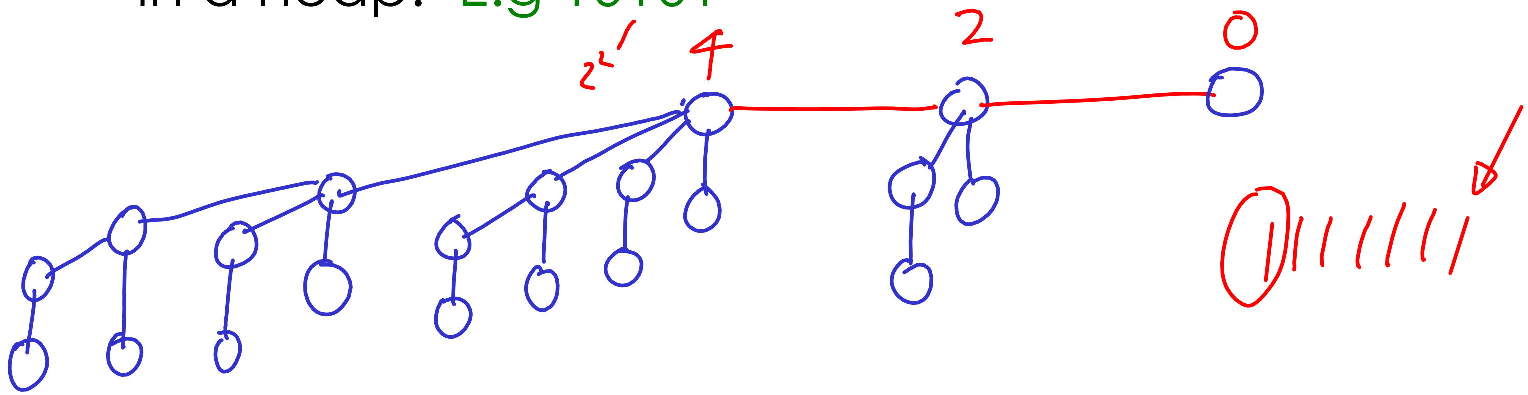
- first()
  - The minimum node in each *tree* is the tree root so the heap minimum is the smallest root



O(no of roots)

How many roots in a binomial heap?

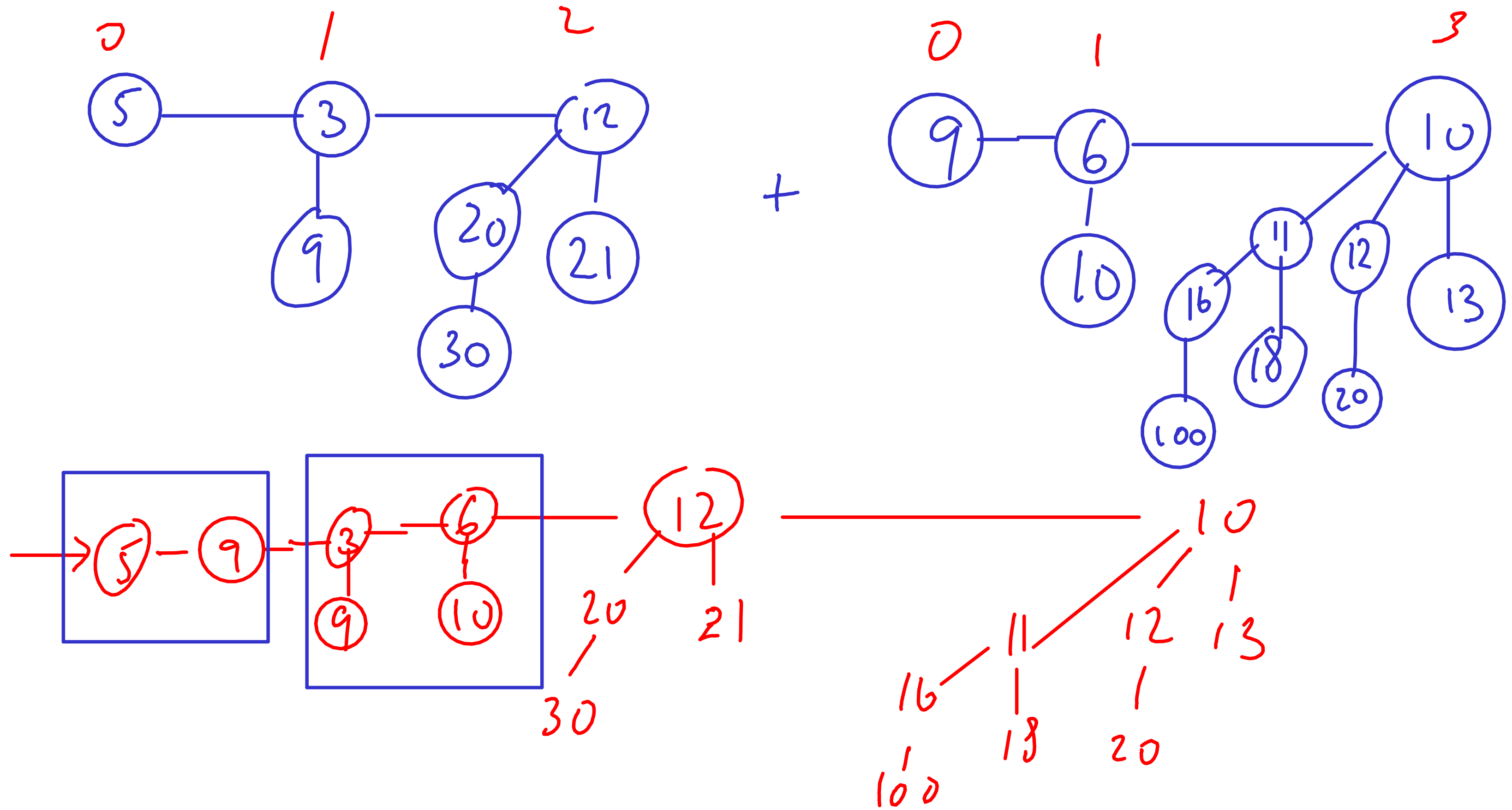
- For a heap with n nodes, how many root (or trees) do we expect?
- Because there are 2^k nodes in a tree of order k, the binary representation of n tells us which trees are present in a heap. E.g 10101 2^0 2^2



- The biggest tree present will be of order log n, which corresponds to the ($\lfloor \log n \rfloor$+1)-th bit
  - So there can be no more than ($\lfloor \log n \rfloor$+1) roots
- first() is O(no. of roots) = O(lg n)

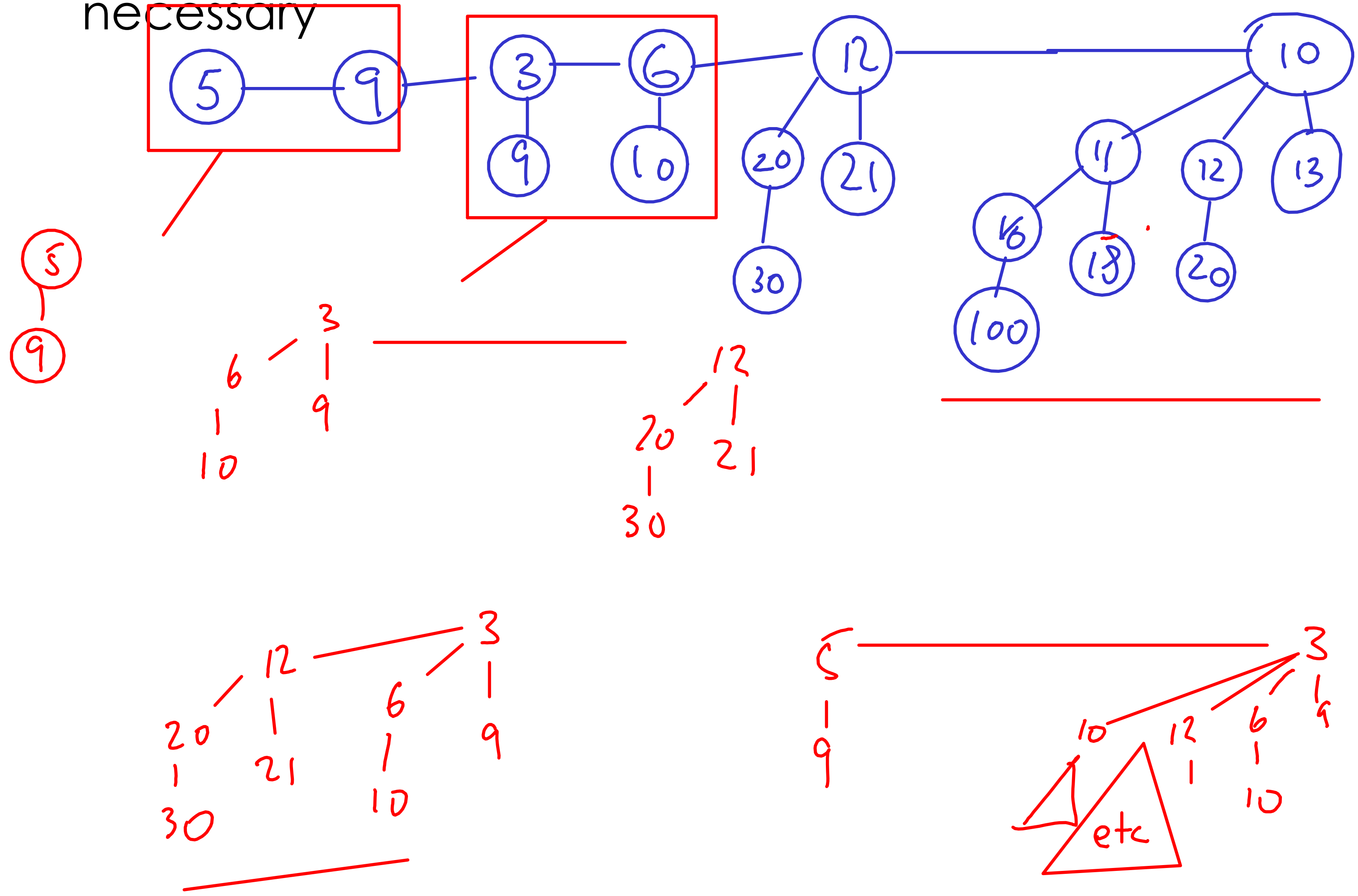
Merging Heaps

- Merging two heaps is useful for the other priority queue operations
- First, link together the tree heads in increasing tree order



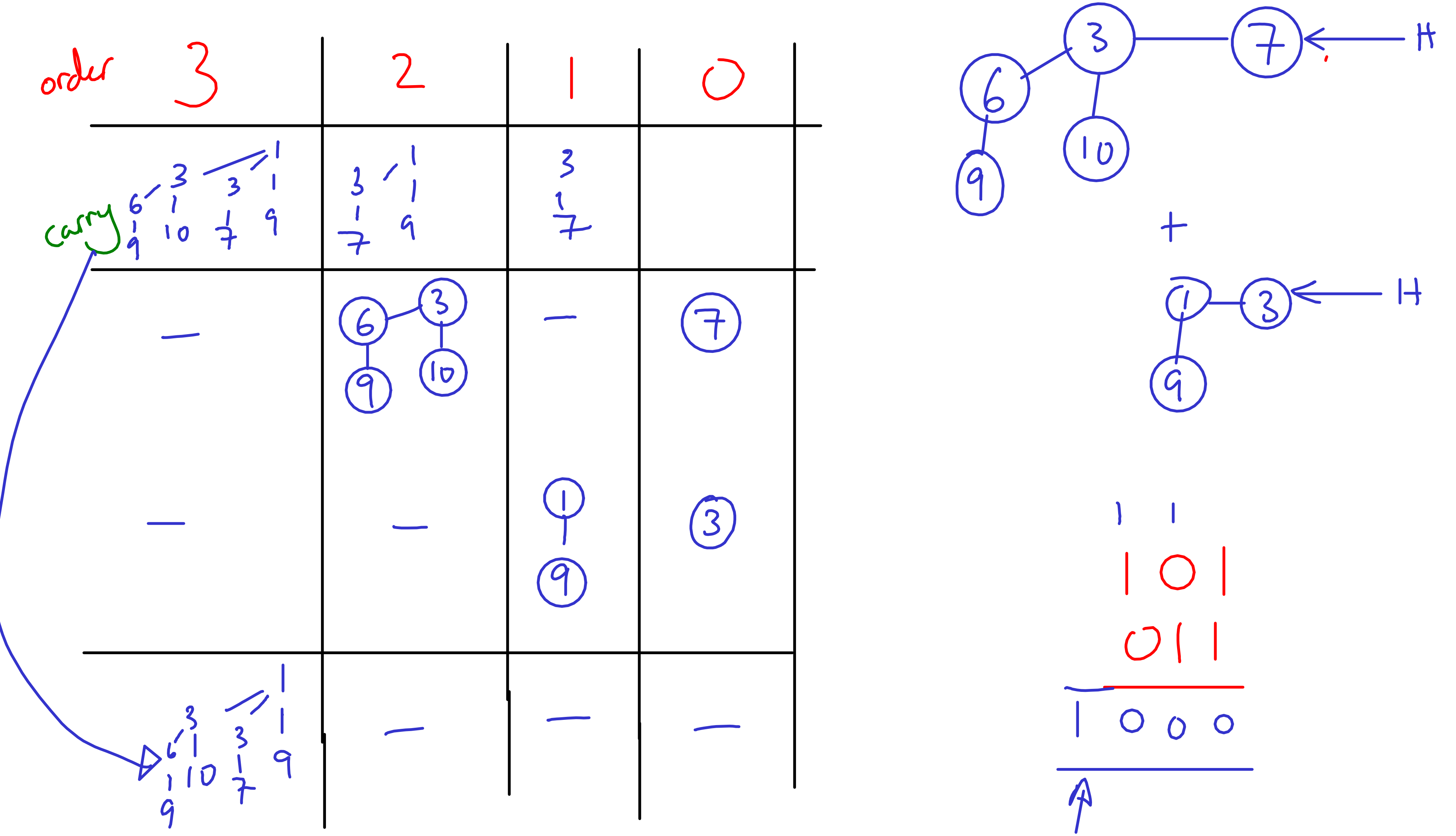
Merging Heaps

- Now check for duplicated tree orders and merge if necessary



Merging Heaps: Analogy

- This process is actually analogous to binary addition!



Merging Heaps: Costs

- Let H1 be a heap with n nodes and H2 a heap with m nodes

$\{H_1 + H_2\} \longrightarrow (n+m)$ nodes

$$H_1 + H_2 \leq \underline{\log_2 n + \log_2 m + 2}$$

No of roots to merge

merge $\Rightarrow o(1)$

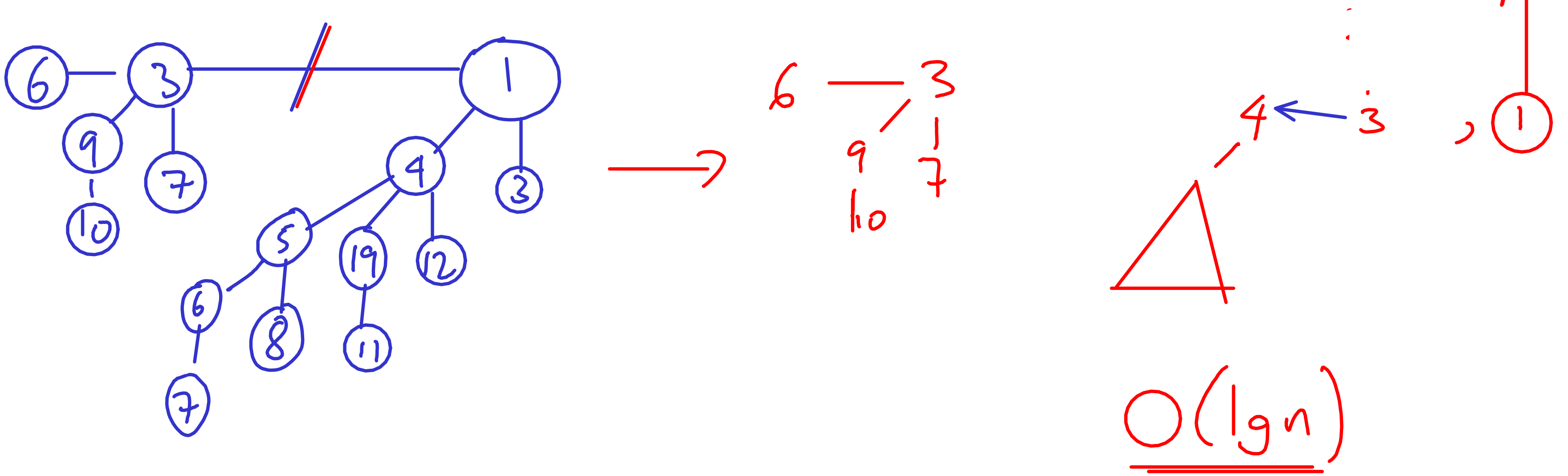
$$\underline{\underline{O(\lg n)}}$$

Priority Queue Operations

- insert()
  - Just create a zero-order tree and merge! $O(\lg n)$

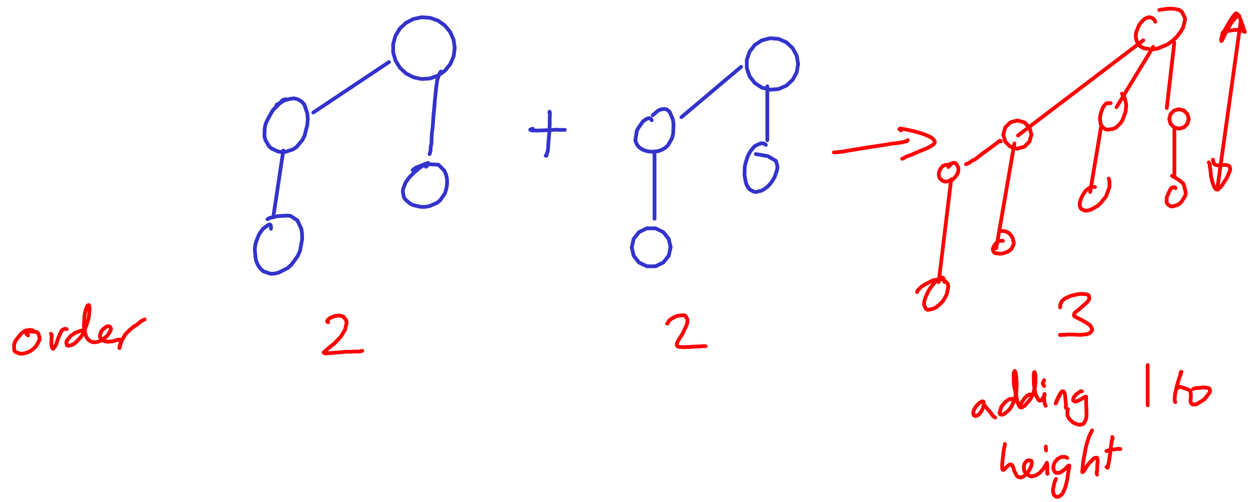
- extractMin()
  - Splice out the tree with the minimum
  - Form a new heap from the 2[nd] level of that tree
  - merge the resulting heap with the original

extract



Priority Queue Operations

- decreaseKey()
  - Change the key value
  - Let it 'bubble' up to its new place
  - O(height of tree)



$\underline{\underline{O(\lg n)}}$

Priority Queue Operations

- deleteKey()
  - Decrease node value to be the minimum — $O(\lg n)$
  - Call extractMin() (!) — $O(\lg n)$

$\underline{\underline{O(\lg n)}}$

Recap...

- Sorting
  - Bubble, (Binary) Insertion, Selection, Merge, Quick, Heap.
- Algorithm Design
  - Greedy, Brute force, Backtracking, Divide and Conquer, Dynamic Programming
- Data Structures
  - Stack, Queue, Deque
  - Tables: R-B Trees, AVL Trees, B-Trees, Hash Tables
  - Priority Queue: Binary heap, Binomial Heap