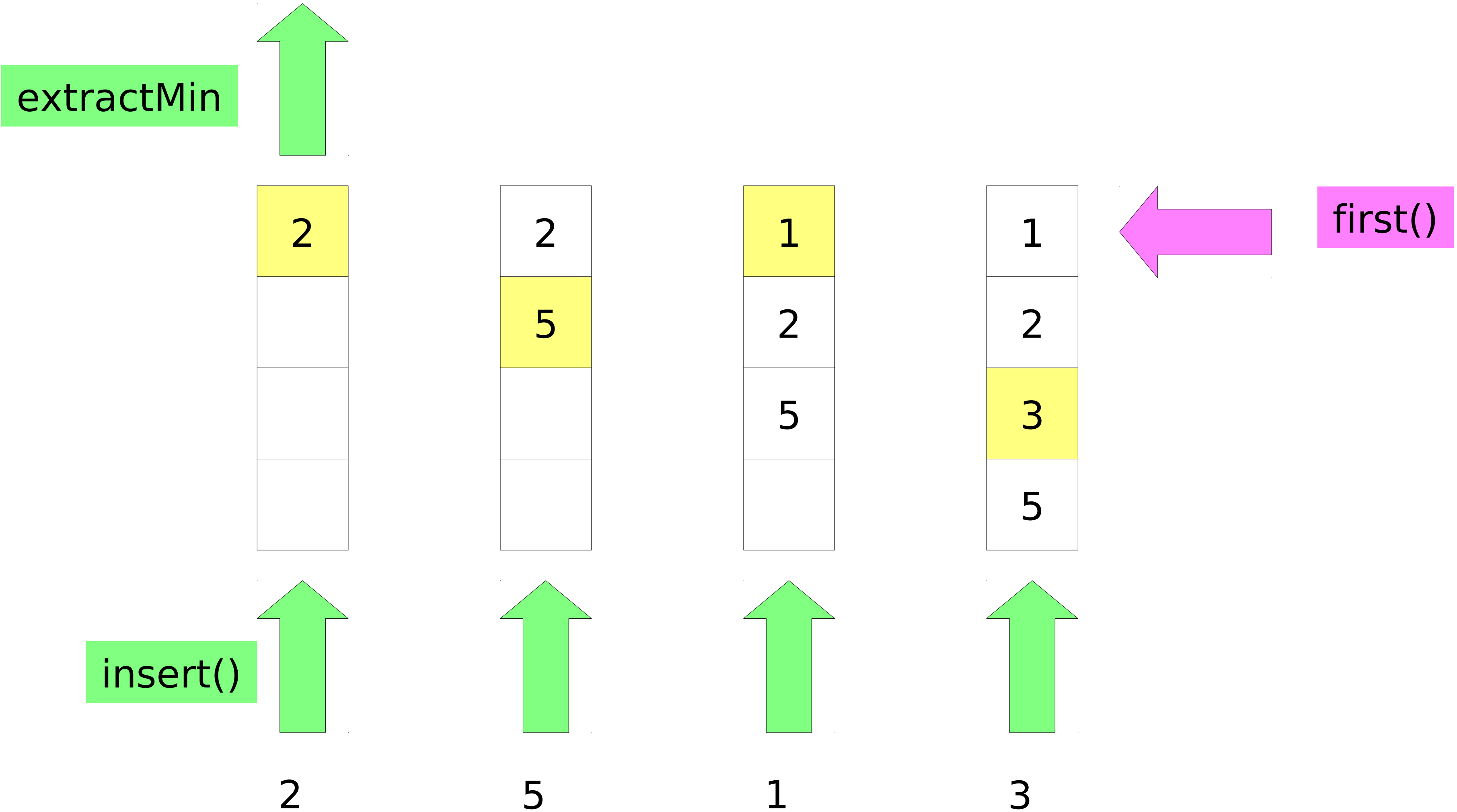
Priority Queues

Priority Queue



Priority Queue Applications

- Event-driven simulations (particle collisions, queuing customers, traffic)
- Data compression
- Statistical analysis
- Operating systems (process queue)
- Graph searching
- Optimisation algorithms

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

Example: order statistics

- Need to find top 100 results for a web search
- Can't use quickselect because not enough memory

```
function top100() {
    PriorityQueue pq;
    while ( elements_remain ) {
        next=get_next_element();
        pq.add(next);
        if (pq.size() > 100) {
          pq.extractMin();
        }
    }
}
```

Array Implementations

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

	first()	insert()	extractMin()	decreaseKey()	merge()
Unsorted ~~List~~ Array	n	1	n	n	n
Sorted ~~List~~ Array	1	n	n	n	n

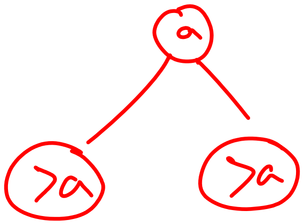
RB Tree Implementation

- Put everything into a Red-Black Tree

	first()	insert()	extractMin()	decreaseKey()	merge()
Unsorted List	n	1	n	n	n
Sorted List	1	n	n	n	n
RB Tree	lg n	lg n	lg n	lg n	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

  $\Rightarrow O(\text{no. of levels}) = O(\lg n)$

- first()   • Top $O(1)$

Binary Heap Implementation

- extractMin()

  - Extract
  - Fix heap

  Like one iteration of heapsort $\Rightarrow O(\lg n)$

- decreaseKey()

  - Find $O(n)$?
  - Change $O(1)$
  - Bubble $O(\lg n)$

- merge()

  - Insert all of 1 in 2 $\Rightarrow O(n \lg n)$

Limitations of the Binary Heap

	first()	insert()	extractMin()	decreaseKey()	merge()
Unsorted List	n	1	n	n	n
Sorted List	1	n	n	n	n
RB Tree	lg n	lg n	lg n	lg n	nlg n
Binary Heap	1	lg n	lg n	lg n	nlg n

- Binary heap is pretty good except for merging.
- Can we do better?

Binomial Heap Implementation

- First define a binomial **tree**
  - Order 0 is a single node
  - Order k is made by merging two binomial trees of order (k-1) such that the root of one remains as the overall root

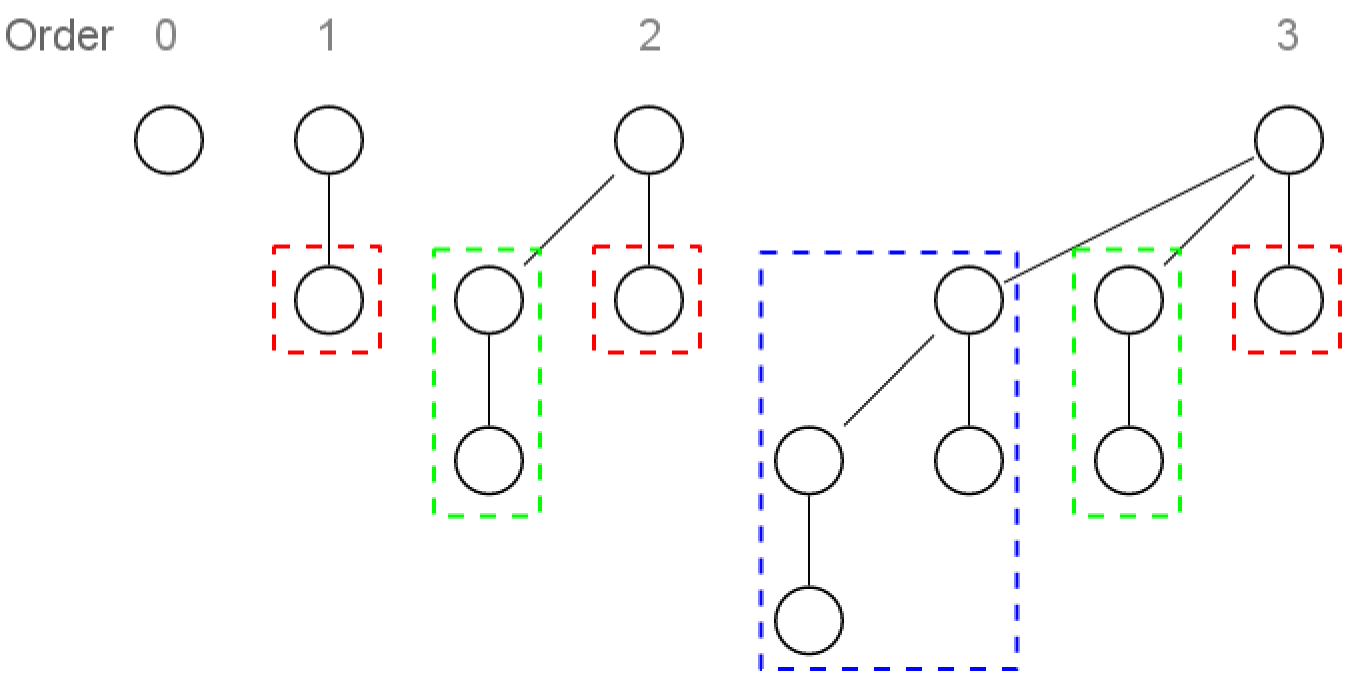
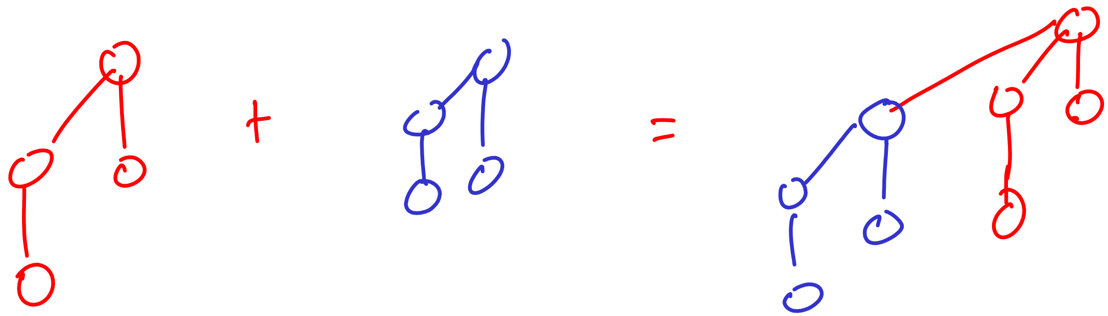


Image courtesy of wikipedia

Merging Trees

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



order 2 2 3

⇒ $O(1)$ to merge two <u>trees</u> of same order

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	n
0	1
1	2
2	4

$$\Rightarrow n_{tree} = 2^{order}$$

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

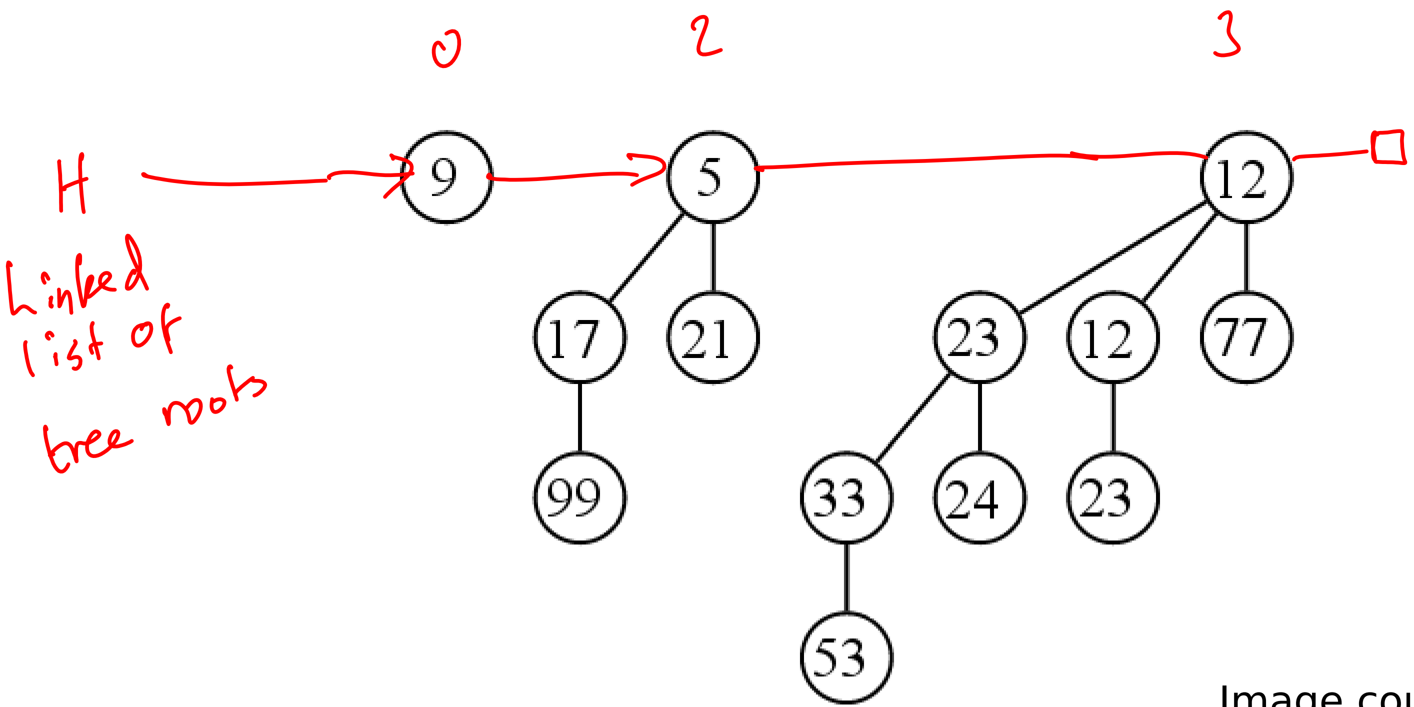
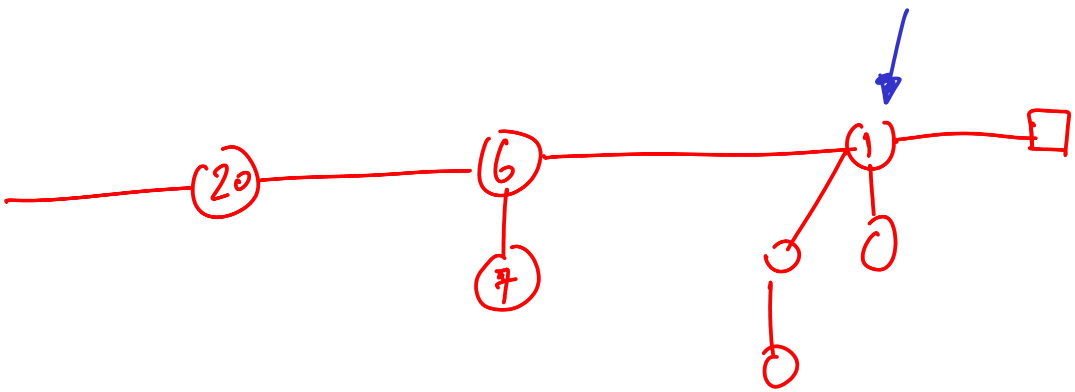


Image courtesy of wikipedia

Binomial Heaps as Priority Queues

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



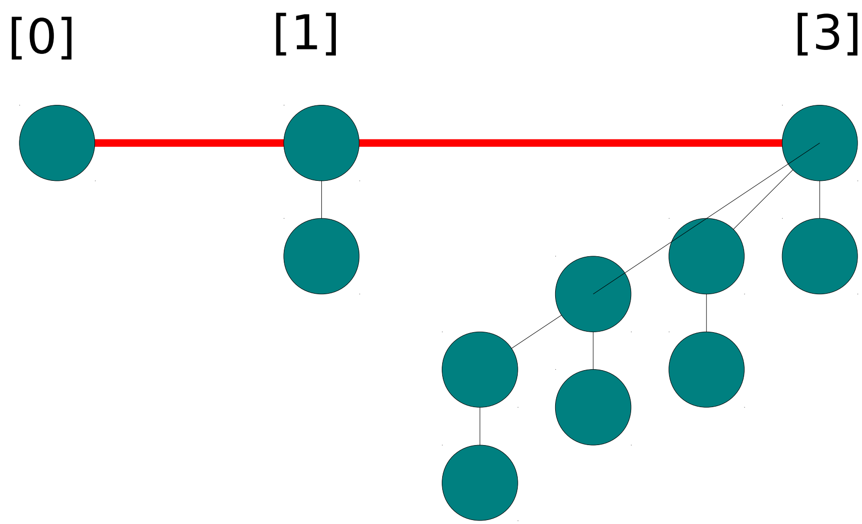
∴ Traverse linked list to find min.

$\Rightarrow O(\text{no. of trees})$

How Many Roots?

- We can only have one or zero of each tree order
- Therefore represent compactly as a string of ones and zeroes:

F[0]=1 F[1]=1 F[2]=0 F[3]=1



$S = \sum_i F[i] \cdot 2^i$

$= 1011$

S=

F[3] F[0]

- Then n = $\sum S[i]*2^i$
- i.e. S is just the binary representation of n...

How Many Roots in a binomial heap?

- The largest bit possible is therefore the (lg n + 1)-th bit
- So there can't be more than (lg n + 1) roots/trees
- first() is O(no. of roots) = O(lg n)

$5 = \overbrace{101}^{3}$ $\lfloor \lg 5 \rfloor + 1 = 3$

Max length of $S = \lfloor \lg n \rfloor + 1$

$\therefore$ Biggest tree has order $\lfloor \lg n \rfloor$

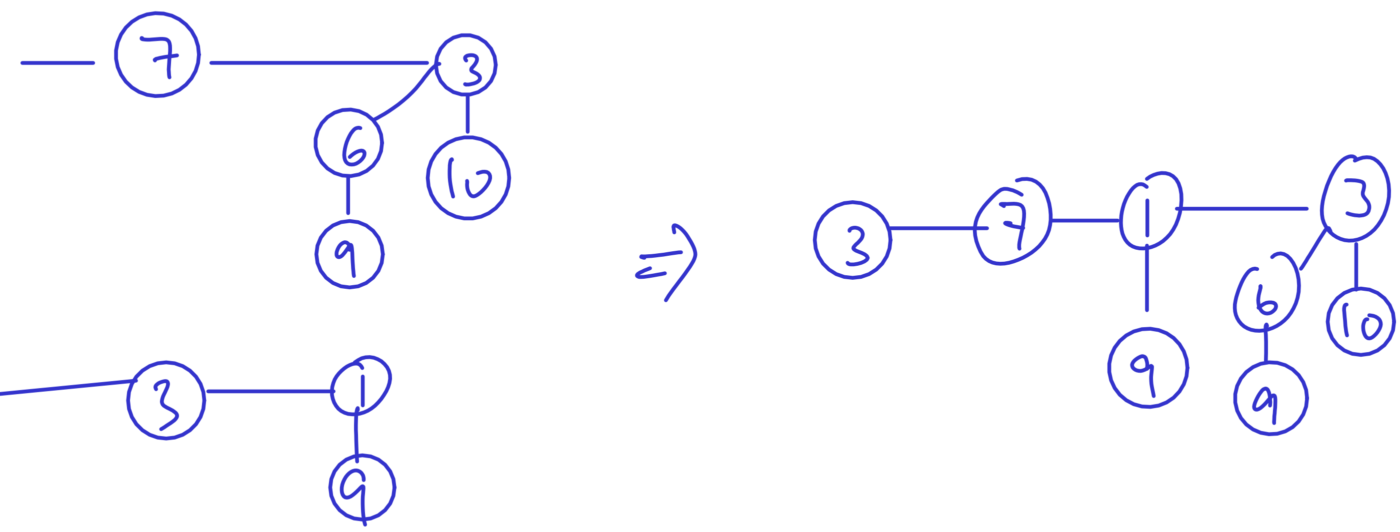
$\therefore$ No. trees = No. roots = $\lfloor \lg n \rfloor$

$= O(\lg n)$

$\therefore$ First is $O(\lg n)$

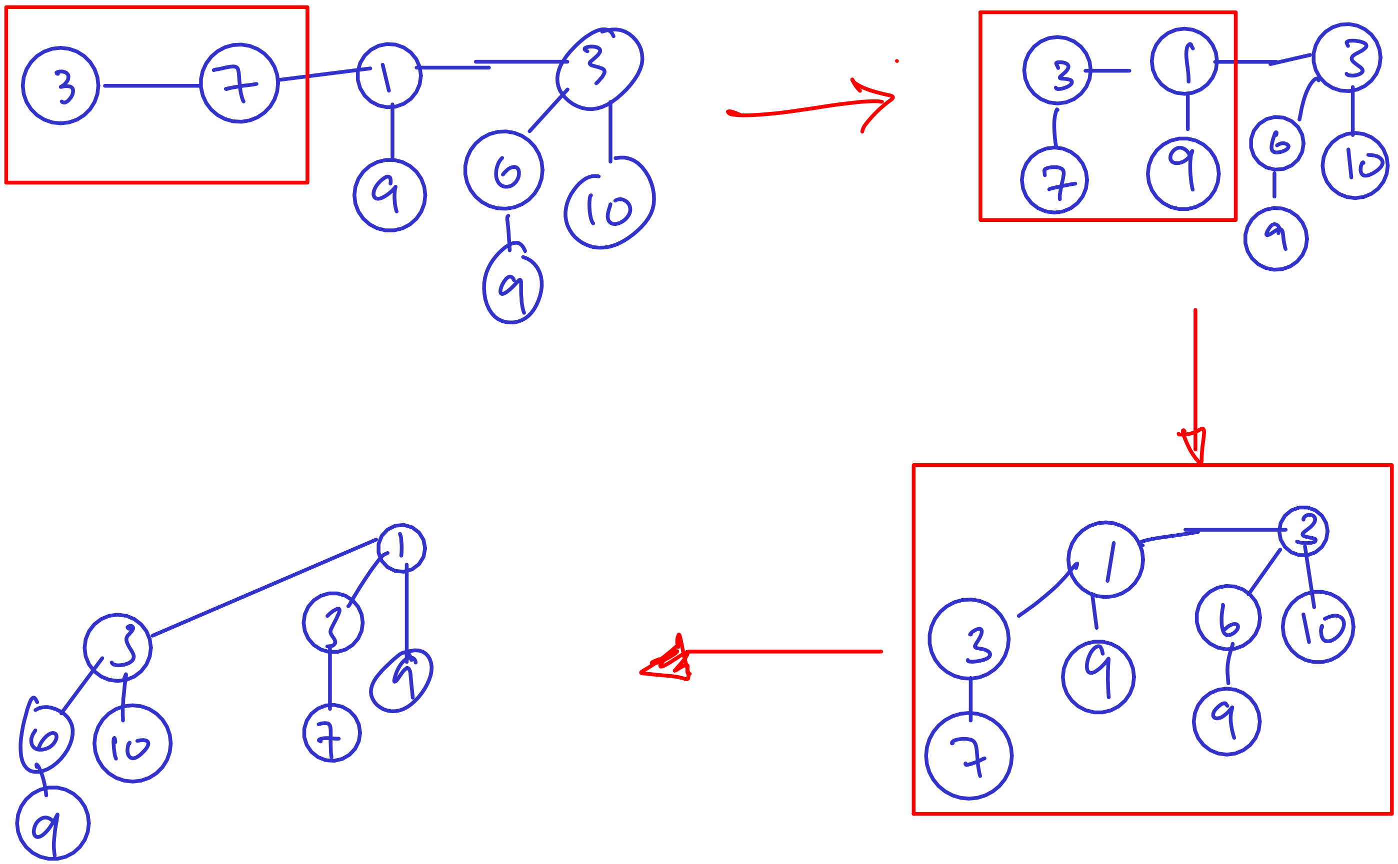
Merging Heaps

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



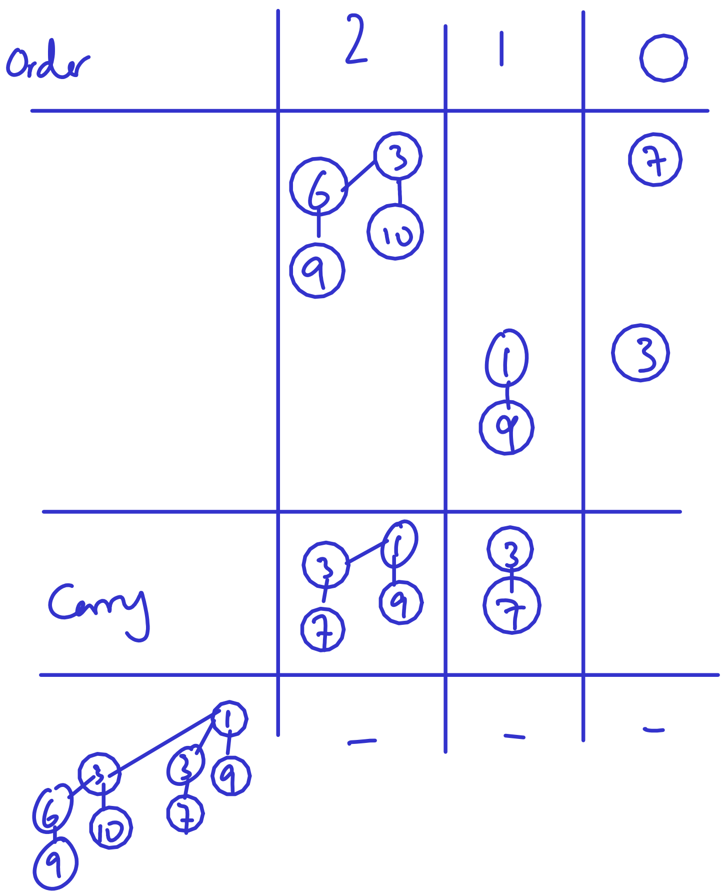
Merging Heaps

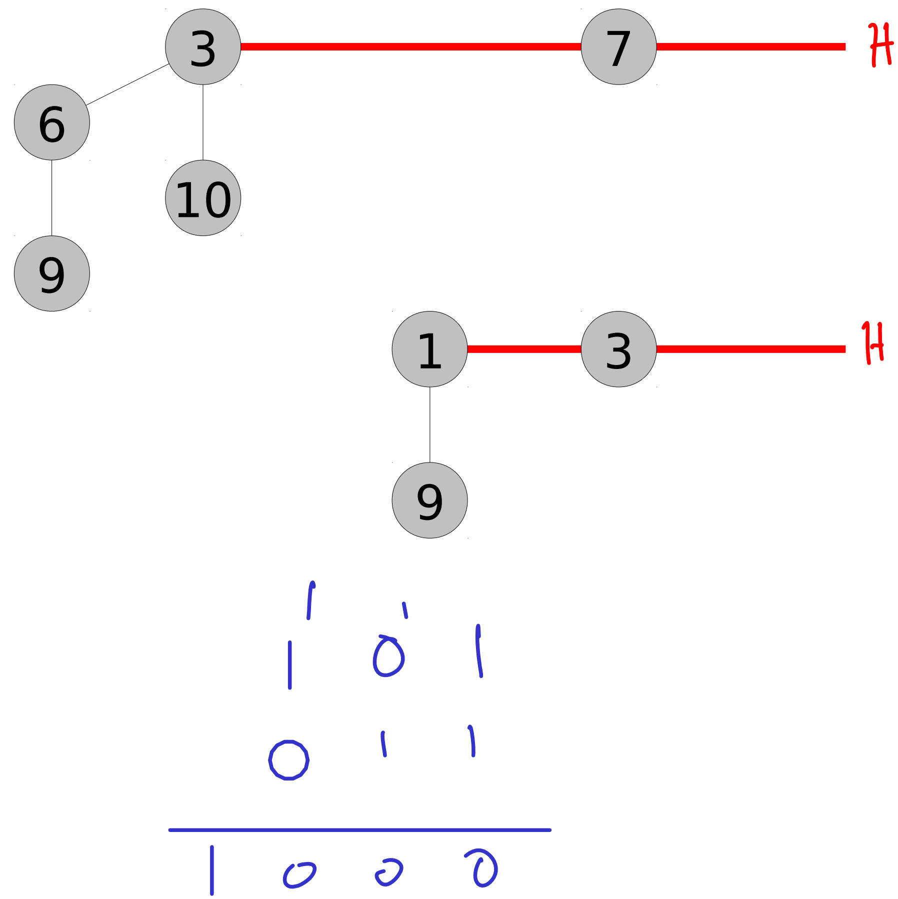
- Now check for duplicated tree orders and merge if necessary



Merging Heaps: Analogy

- Actually this is just binary addition





Merging Heaps: Costs

- The addition analogy makes this easy to analyse
- Worst case: need to merge at every step and end up with an overflow into the next highest bit position

$$\begin{array}{r} 11111 \\ 1111 \\ \hline 101000 \end{array}$$

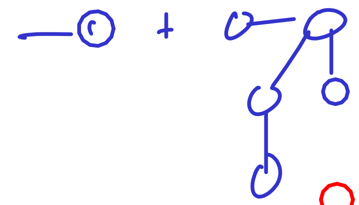
Each merge is $O(1)$

We will need $((\lfloor \log n \rfloor + 1) + 1)$ merges

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

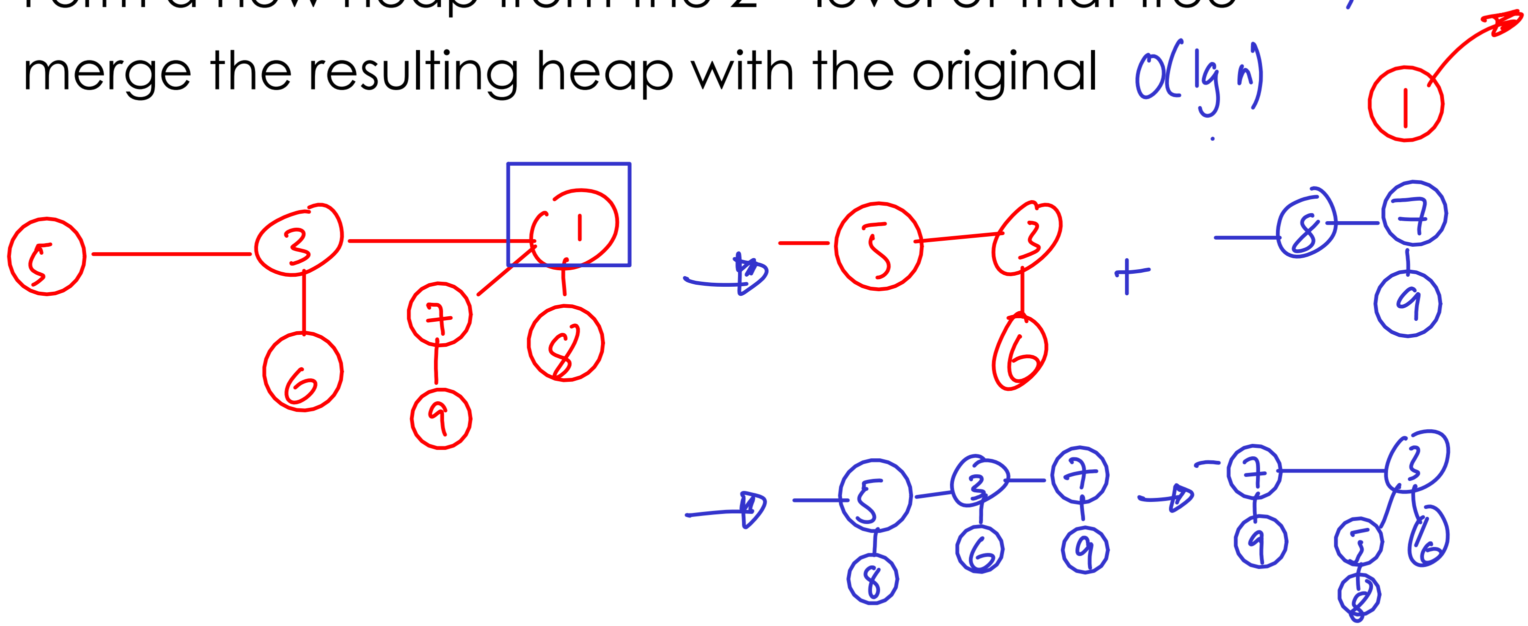
Priority Queue Operations

- insert()
  - Just create a zero-order tree and merge!



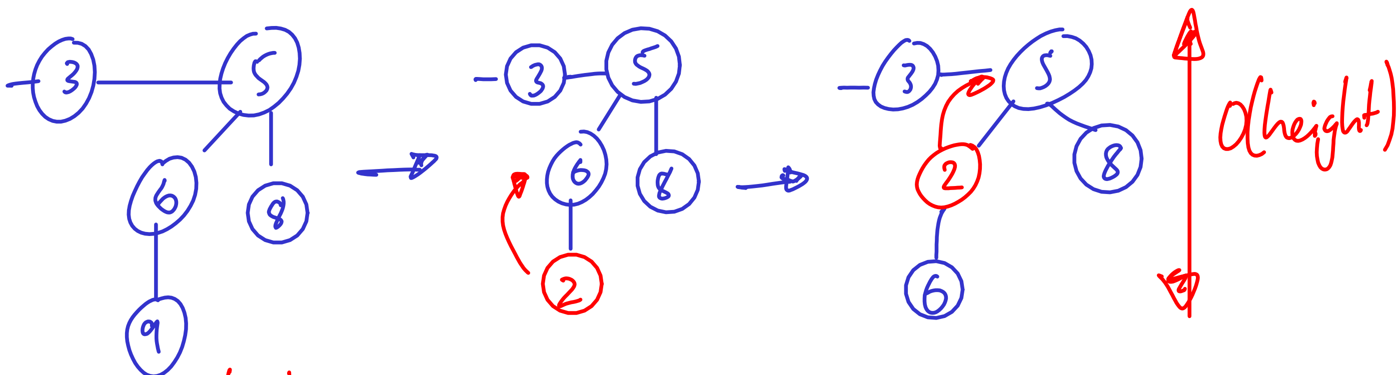
$O(\lg n)$

- extractMin()
  - Splice out the tree with the minimum $O(1)$
  - Form a new heap from the 2nd level of that tree $O(1)$
  - merge the resulting heap with the original $O(\lg n)$



Priority Queue Operations

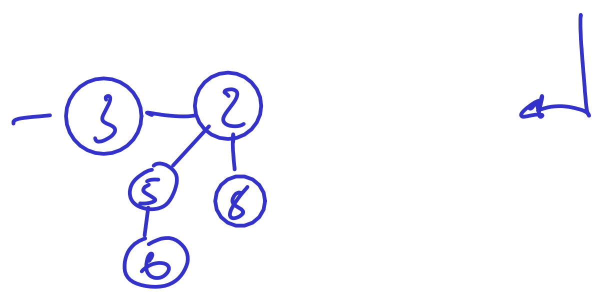
- decreaseKey()
  - Change the key value
  - Let it 'bubble' up to its new place
  - O(height of tree)



height of order k = k+1

Biggest order $\lfloor \lg n \rfloor + 1$

$\Rightarrow O(\lg n)$



So…

	first()	insert()	extractMin()	decreaseKey()	merge()
Unsorted List	n	1	n	n	n
Sorted List	1	n	n	n	n
RB Tree	lg n	lg n	lg n	lg n	nlg n
Binary Heap	1	lg n	lg n	lg n	nlg n
Binomial Heap	lg n	lg n	lg n	lg n	lg n

That's all folks...

- Sorting
  - Bubble, (binary) insertion, selection, mergesort, quicksort, heapsort
- Algorithm Design
  - Brute force, backtracking, greedy, divide and conquer, dynamic
- Data Structures
  - Stack, queue, deque, priority queues
  - BST, RB Tree, B-Tree, hash tables
- String Searching
  - Naïve, Rabin-Karp, KMP

Finally…

- Good luck in your exams..!