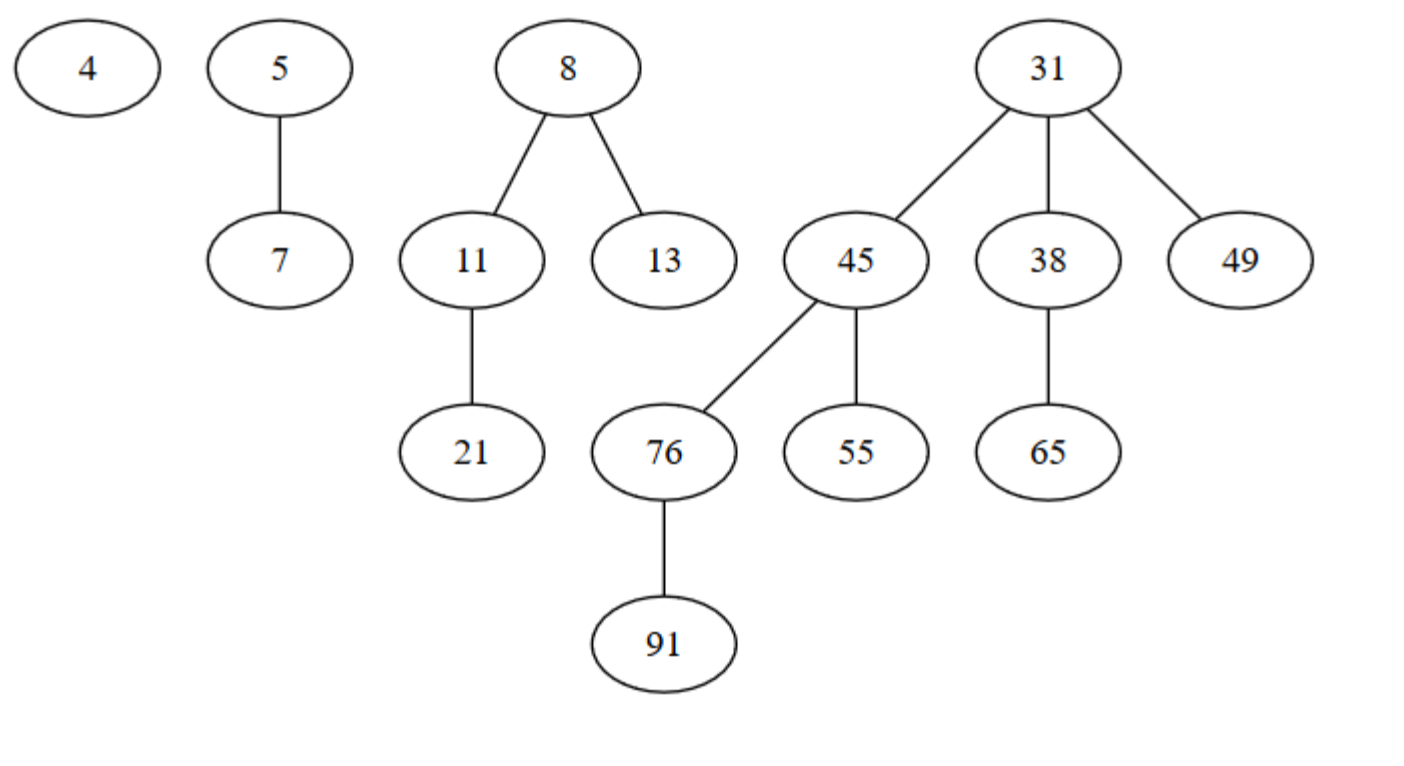
Binomial tree: examples



Binomial trees of order 0, 1, 2 and 3

Binomial tree

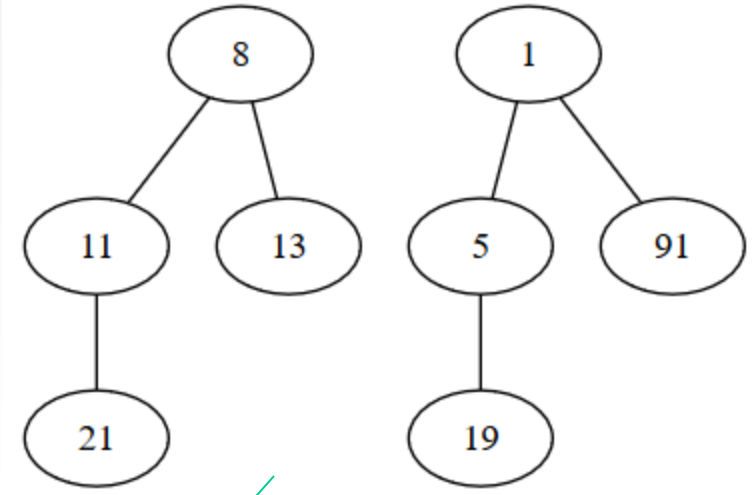
- A binomial tree can be defined in a recursive manner:
  - A binomial tree of order 0 is a single node.
  - A binomial tree of order k is a tree which has a root and k children, each being the root of a binomial tree of order k - 1, k - 2, ..., 2, 1, 0 (in this order).

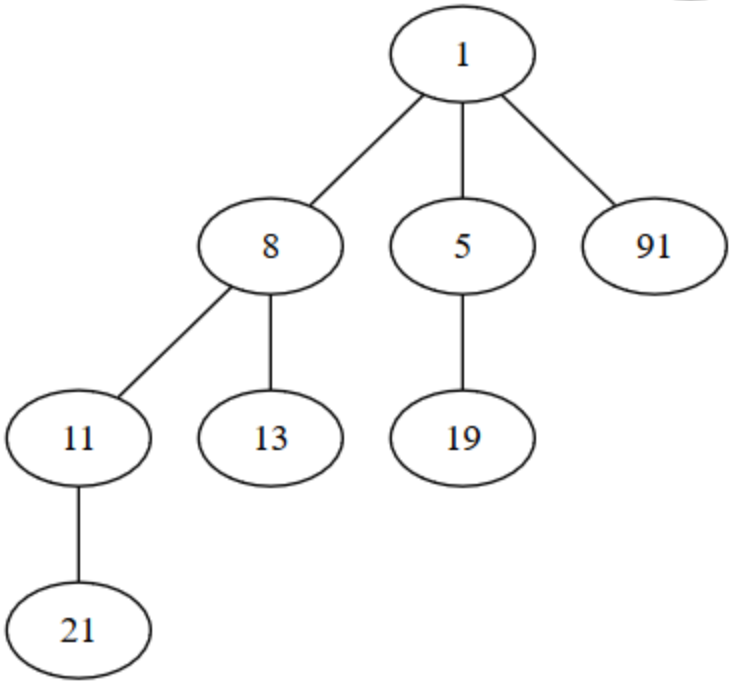
Binomial tree

- A binomial tree of order k has exactly 2^k nodes.
- The height of a binomial tree of order k is k.
- If we delete the root of a binomial tree of order k, we will get k binomial trees, of orders k - 1, k - 2, ..., 2, 1, 0.
- Two binomial trees of the same order k can be merged into a binomial tree of order k + 1 by setting one of them to be the leftmost child of the other.

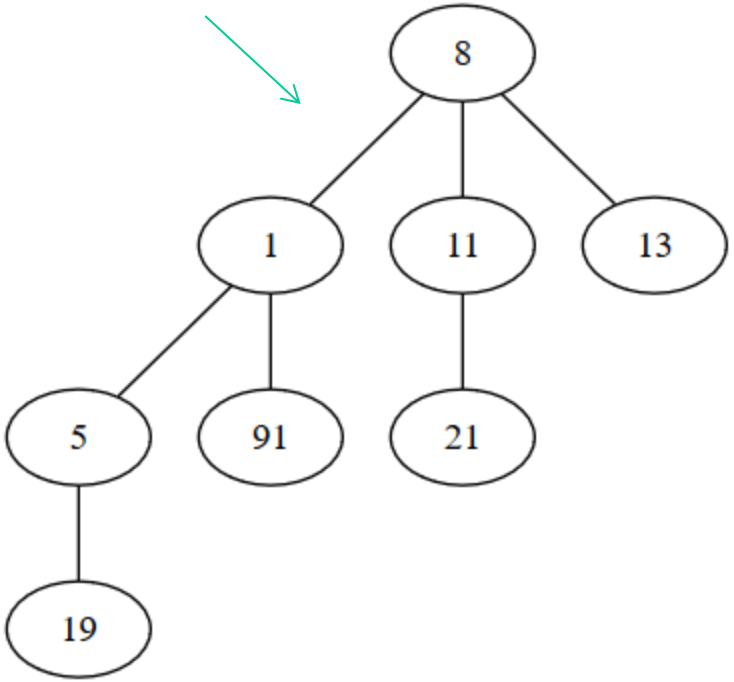
Binomial tree: merge

Before merge:





After merge:



Binomial tree: representation

If we want to implement a binomial tree, we can use the following representation:

- We need a structure for nodes, and for each node we keep the following:
  - The information from the node
  - The address of the parent node
  - The address of the first child node
  - The address of the next sibling node
- For the tree we will keep the address of the root node (and probably the order of the tree)

Binomial heap

A binomial heap is made of a collection/sequence of binomial trees with the following property:

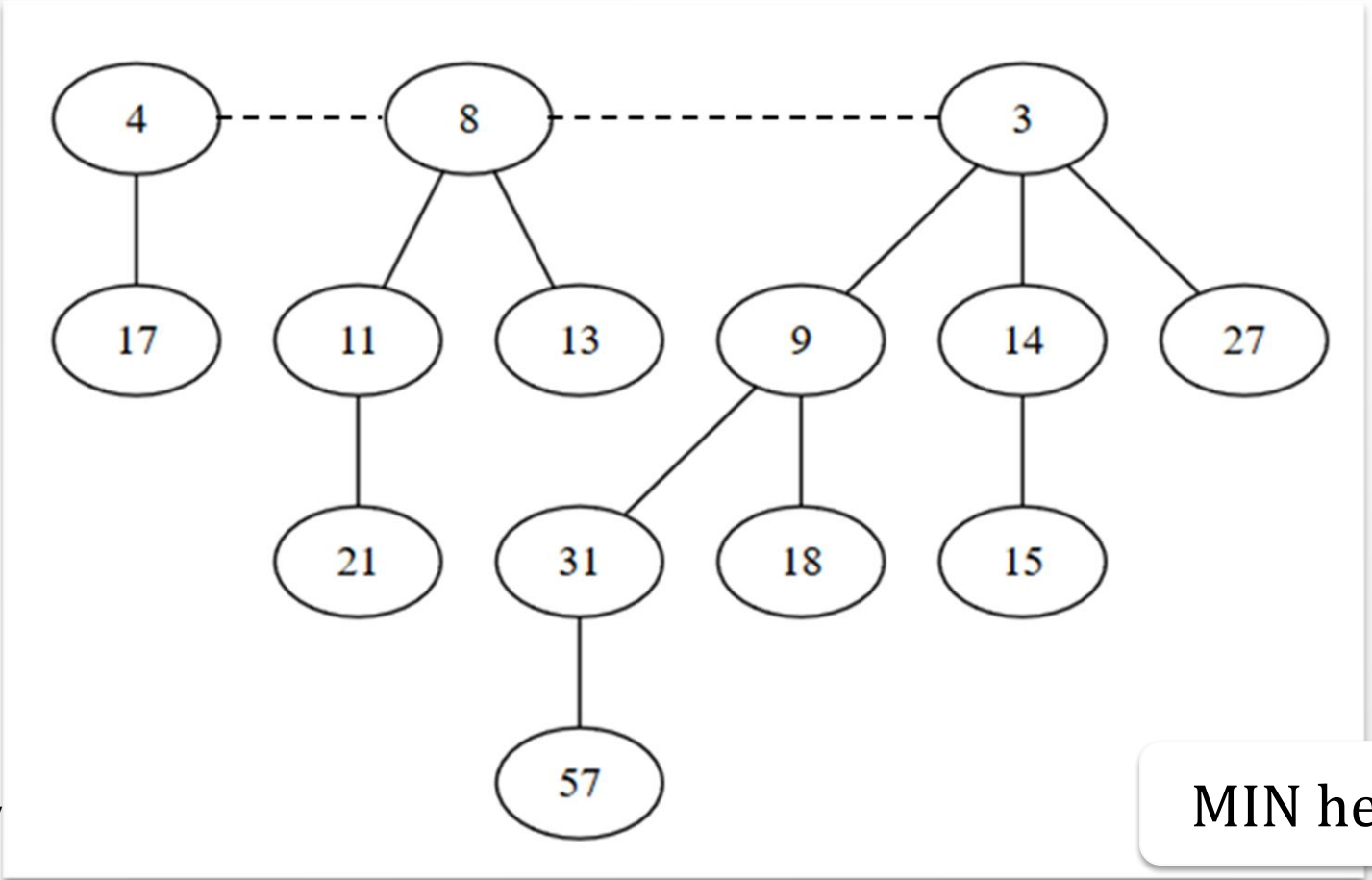
- Each binomial tree respects the heap-property: for every node, the value from the node is less than the value of its children (assume MIN HEAPS).
- There can be at most one binomial tree of a given order k.

As representation, a binomial heap is usually a sorted linked list, where each node contains a binomial tree, and the list is sorted by the order of the trees.

Binomial heap: example

Binomial heap with 14 nodes,

made of 3 binomial trees of orders 1, 2 and 3



MIN heap

Binomial heap: properties

- For a given number of elements, n, the structure of a binomial heap (i.e. the number of binomial trees and their orders) is unique.
- The structure of the binomial heap is determined by the binary representation of the number n.

  For example 14 = 1110 (in binary) = $2^3 + 2^2 + 2^1$

  a binomial heap with 14 nodes contains binomial trees of orders 3, 2, 1 (but they are stored in the reverse order: 1, 2, 3).
- A binomial heap with n elements contains at most $\log_2 n + 1$ binomial trees. $\sim O(\log_2 n)$
- The height of the binomial heap is at most $\log_2 n$.

Binomial heap: merge

- Step 1: merge the two linked lists of binomial trees
- Step 2: transform the trees of the same order

  The result of the merge can contain two binomial trees of the same order, so we have to iterate over the resulting list and transform binomial trees of the same order k into a binomial tree of order k +1.

  When we merge the two binomial trees we must keep the heap property.

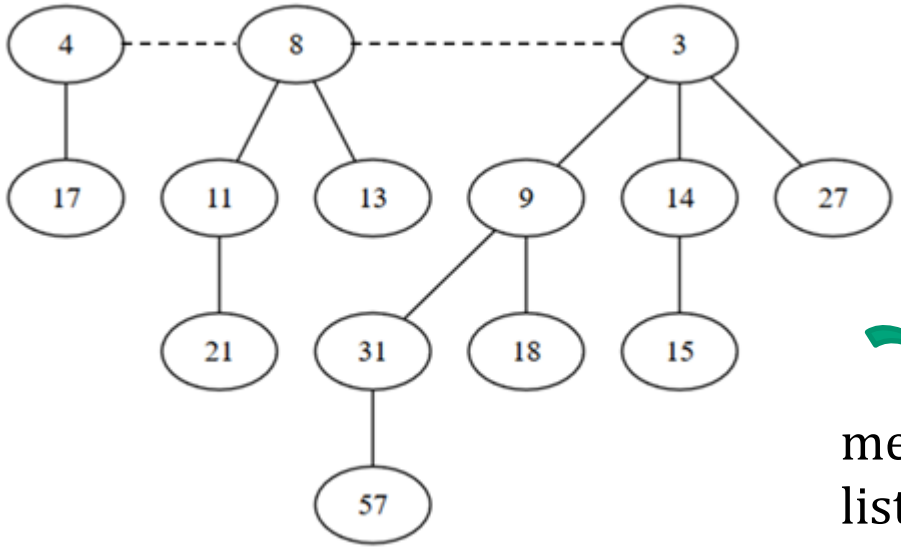
Complexity : $O(\log_2 n)$

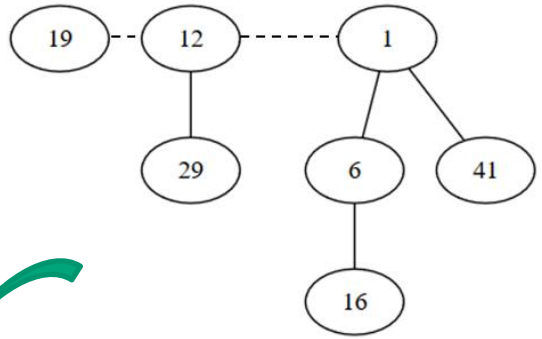
Remarks:

- If H1 contain n1 elements and H2 contain n2 elements,

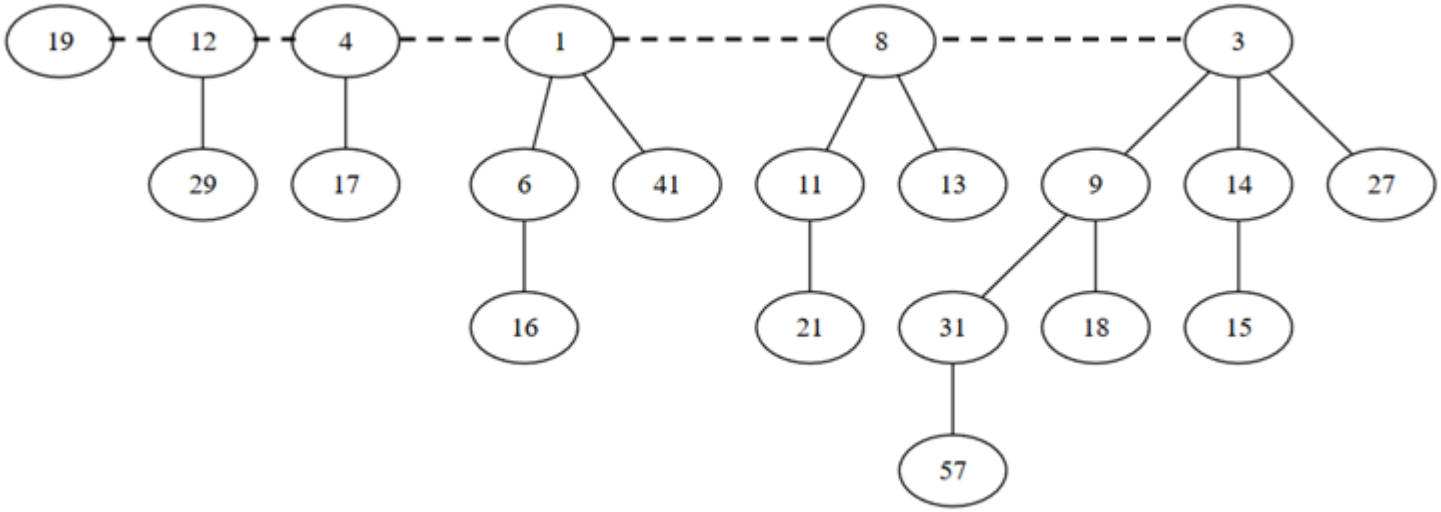
  then H1 contains at most ~ log(n1) binomial trees and H2 … → ~log(n2)
- If n = n1+n2, final binomial heap contains at most ~$\log_2(n)$ binomial trees

Binomial heap: merge, step 1

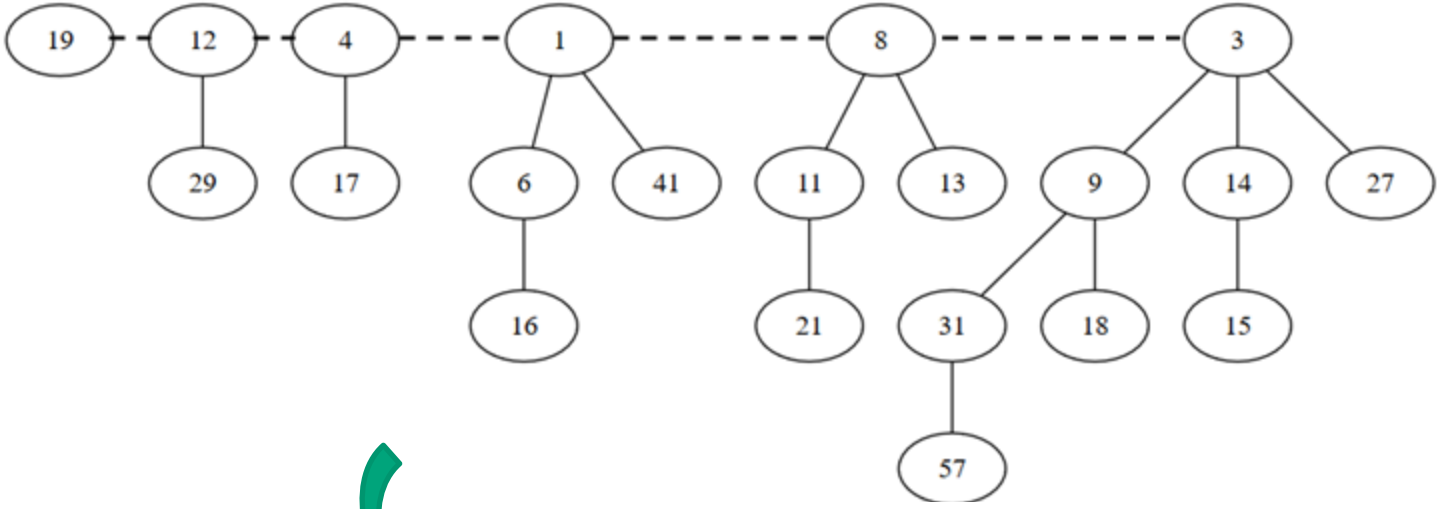




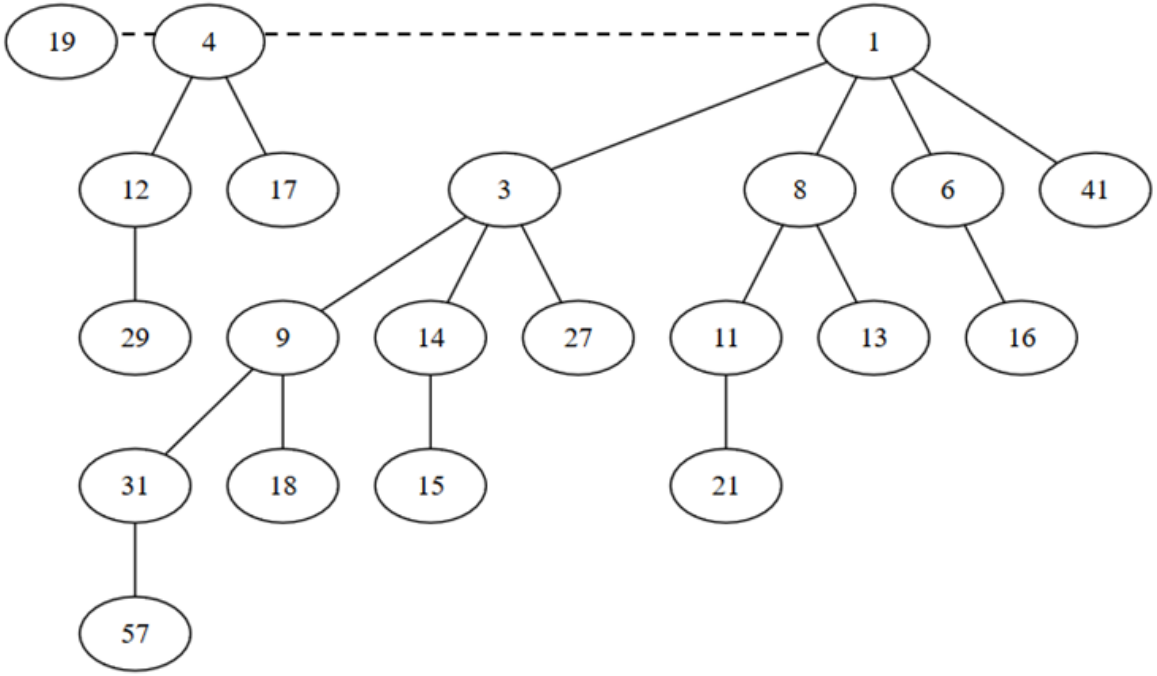
merge the two linked lists of binomial trees



Binomial heap: merge, step 2



transform the trees
of the same order



Binomial heap : operations

- **Push** operation:
  - create a binomial heap with just that element
  - merge it with the existing one.

  Complexity of insert is $O(\log_2 n)$ in worst case (, $\Theta(1)$ amortized).
- **Top** operation:

  The minimum element of a binomial heap (the element with the highest priority) is the root of one of the binomial trees.

  minimum <= checking every root complexity $O(\log_2 n)$
- **Pop** operation:
  - Remove the root of one of the binomial trees (the minimum)

    => get a sequence of binomial trees
  - Transform these trees into a binomial heap (just reverse their order)
  - Merge between this new binomial heap and the one formed by the remaining elements of the original binomial heap.

  The complexity of the remove-minimum operation is $O(\log_2 n)$

Binomial heap : operations

In our examples, lower number means higher priority

"Increase" the priority of an element

Assumption: we have access (pointer?) to the element whose priority has to be changed

- Change the priority and bubble-up the node if its priority is "higher" than the priority of its parent.

Complexity : $O(\log_2 n)$

This will "move" the element to the root of the binomial tree

---

Delete an element (from somewhere in a binomial tree)

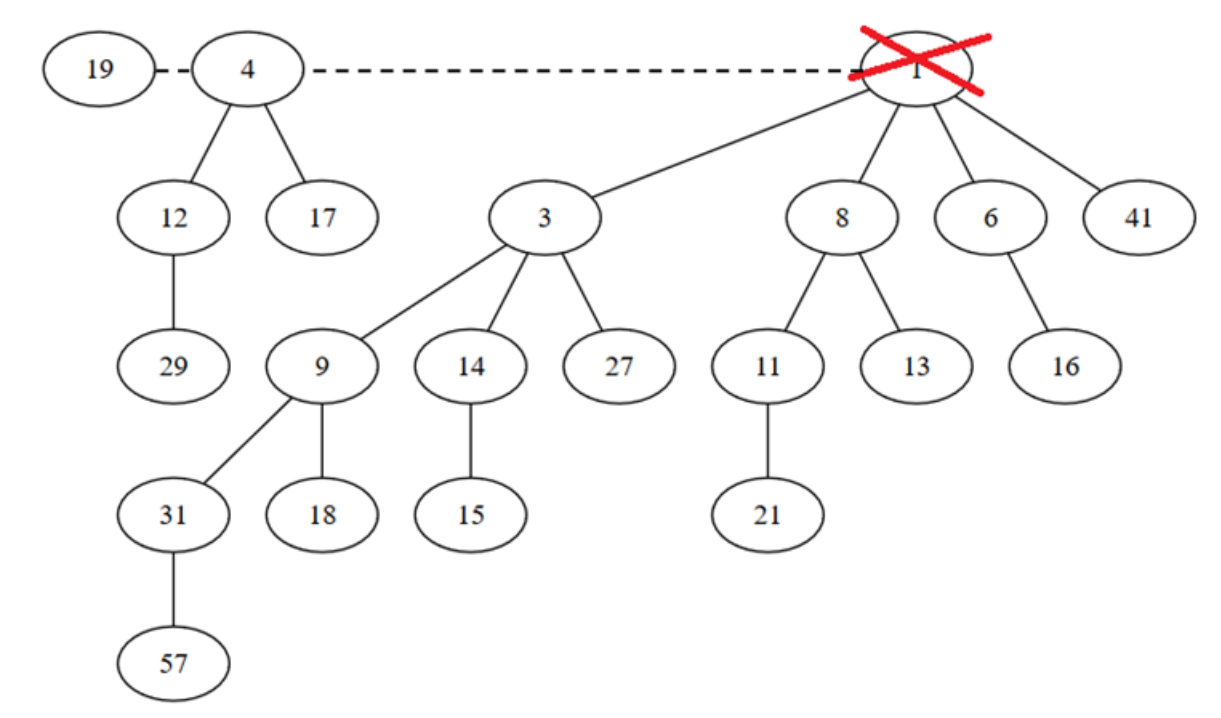
Assumption: we have access (pointer?) to the element

- first set its priority to $-\infty$ (previous operation)
- then remove it.

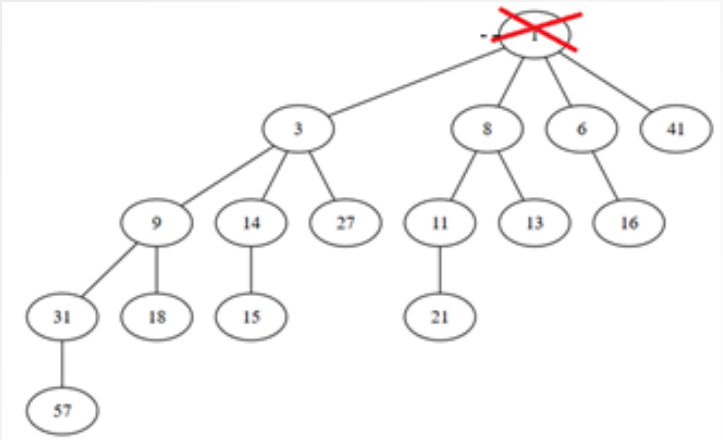
Complexity : $O(\log_2 n)$

Binomial heap. Pop operation. Example

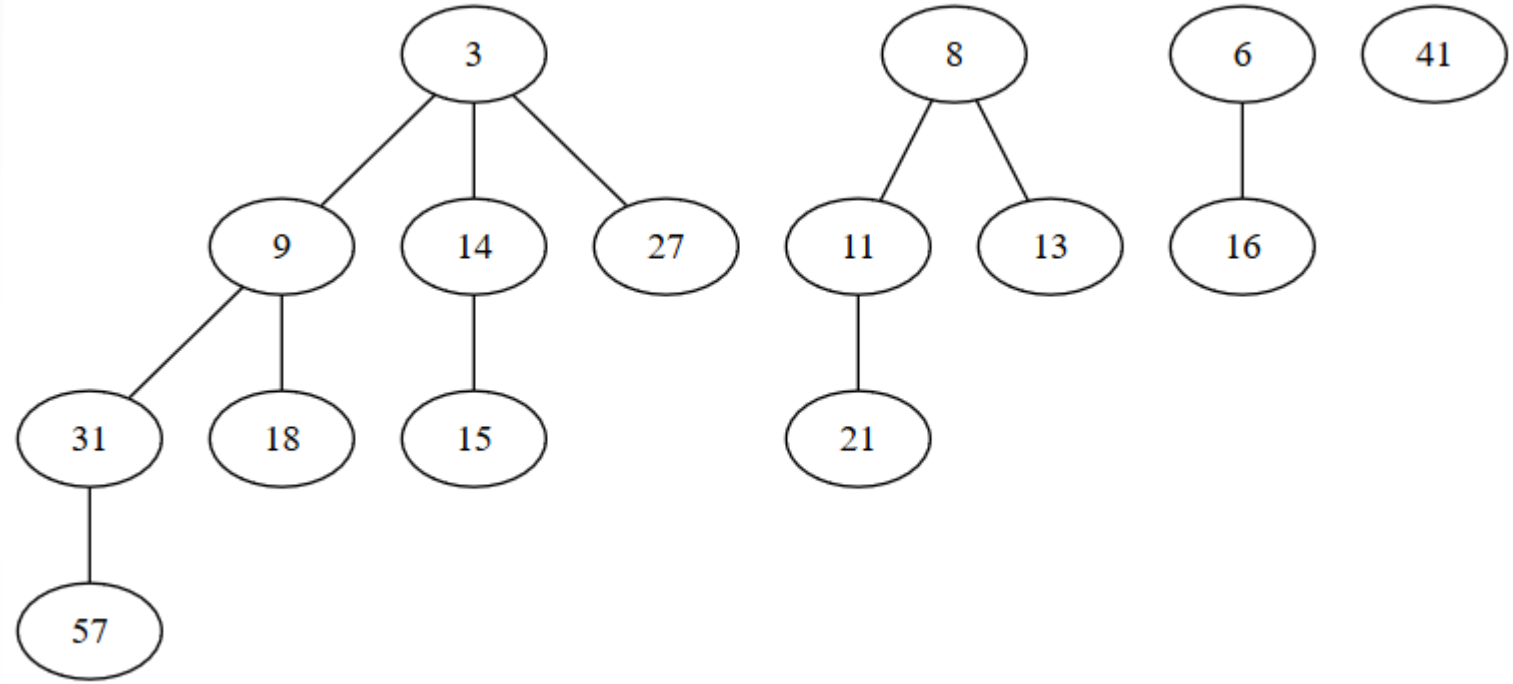
The minimum is one of the roots



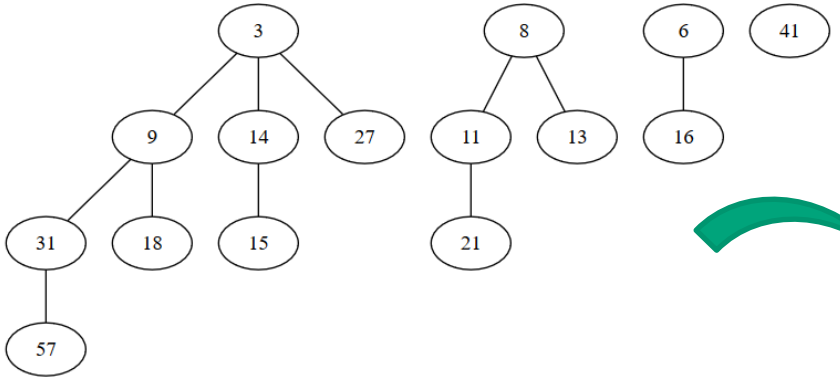
Binomial heap. Pop operation. Example



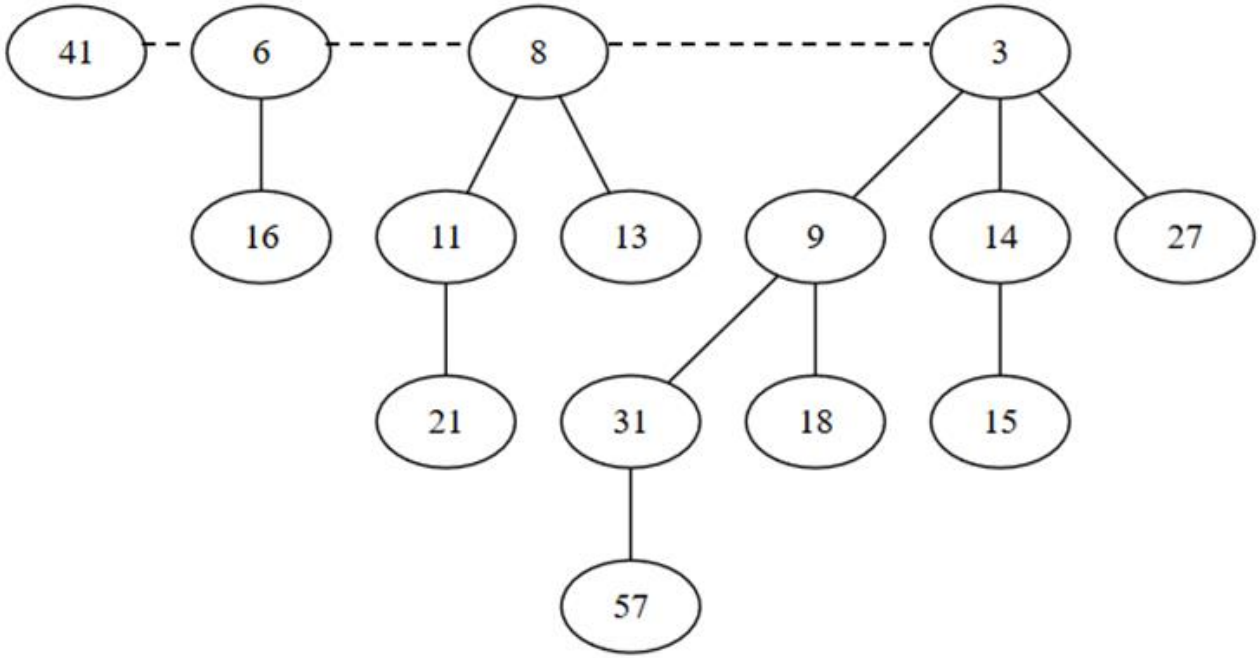
Break the corresponding tree into k binomial trees



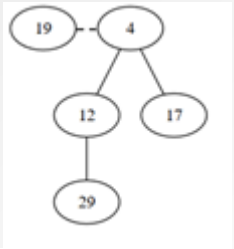
Binomial heap. Pop operation. Example

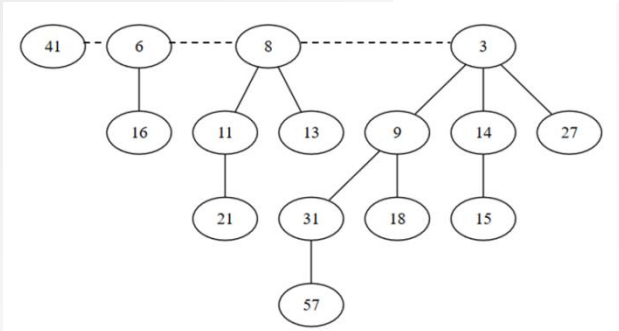


Create a binomial heap of these trees



Binomial heap. Pop operation. Example





Merge it with the existing one (use merge algorithm)