BLG335E Recitation 4
Binomial Heaps

Doğay Kamar
05.12.2019

Course slides from
Kevin Wayne @Princeton
have been used in
preparation of these slides.

Contents

- What is Priority Queue?
- Binary Heaps
- Binomial Trees
- Binomial Heaps

Priority Queues

- A priority queue is a type of queue that assigns priority values to its elements.
- Instead of First-In-First-Out method, values with higher probabilities are dequeued first.
- If smaller values have more priority, which element would be at the top of the queue?

Element with minimum key

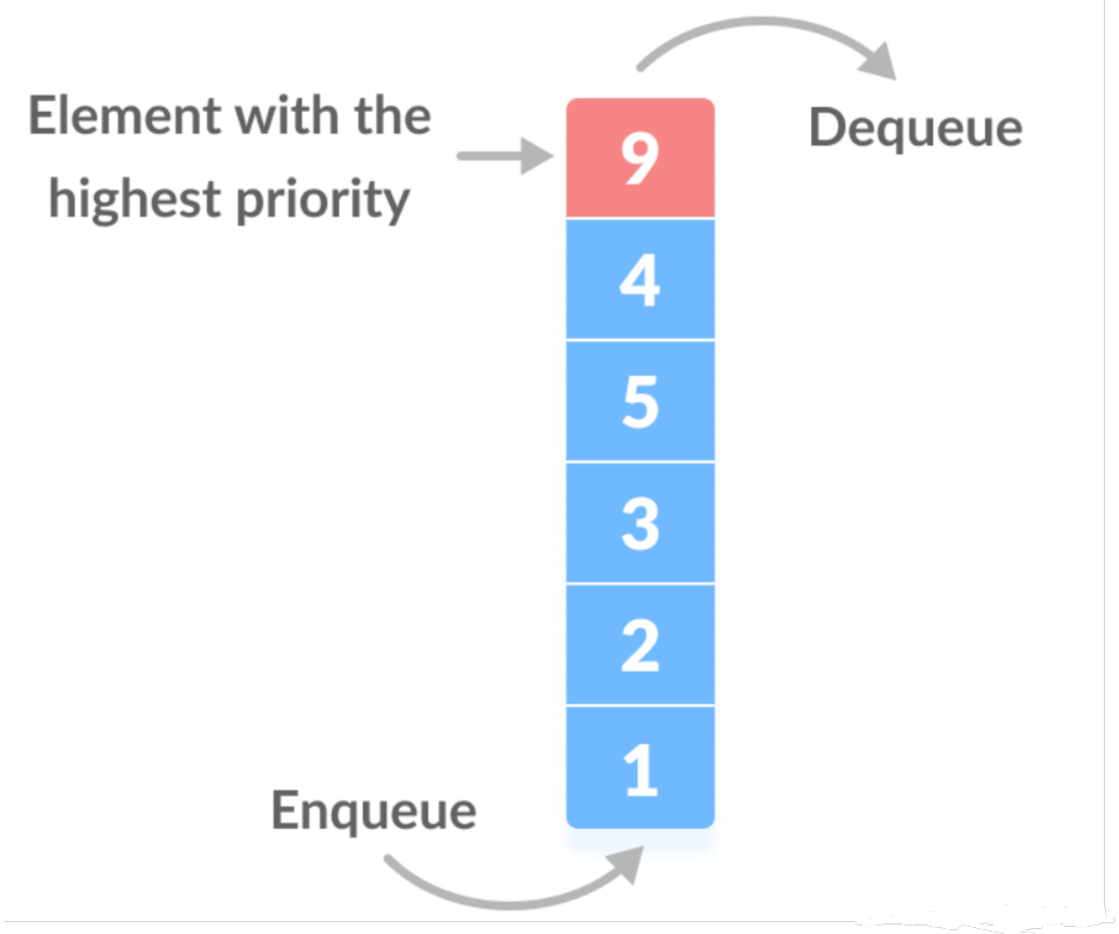


Image retrieved from https://www.programiz.com/dsa/priority-queue

Priority Queues

A priority queue should support following operations:

- Insert an element to the queue
- Pop the minimum element from the queue
- Decrease key of an element

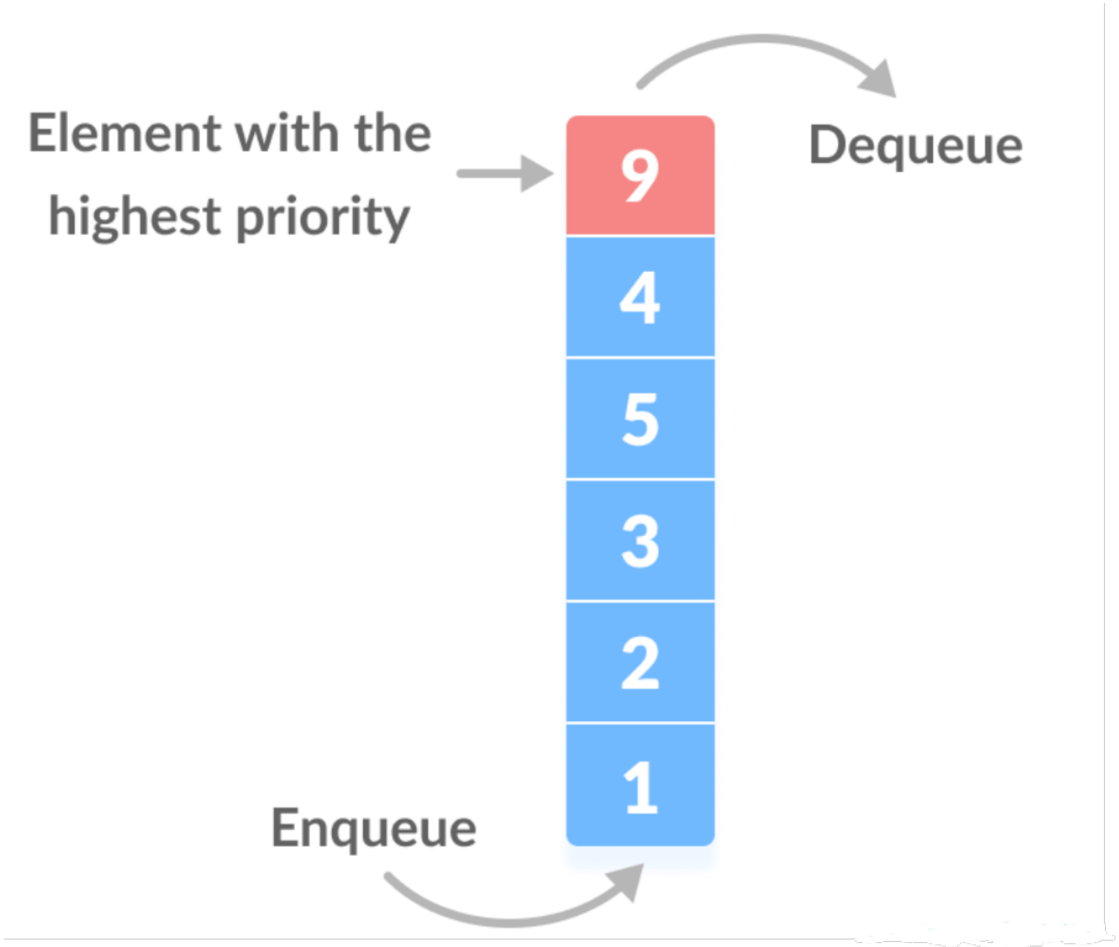
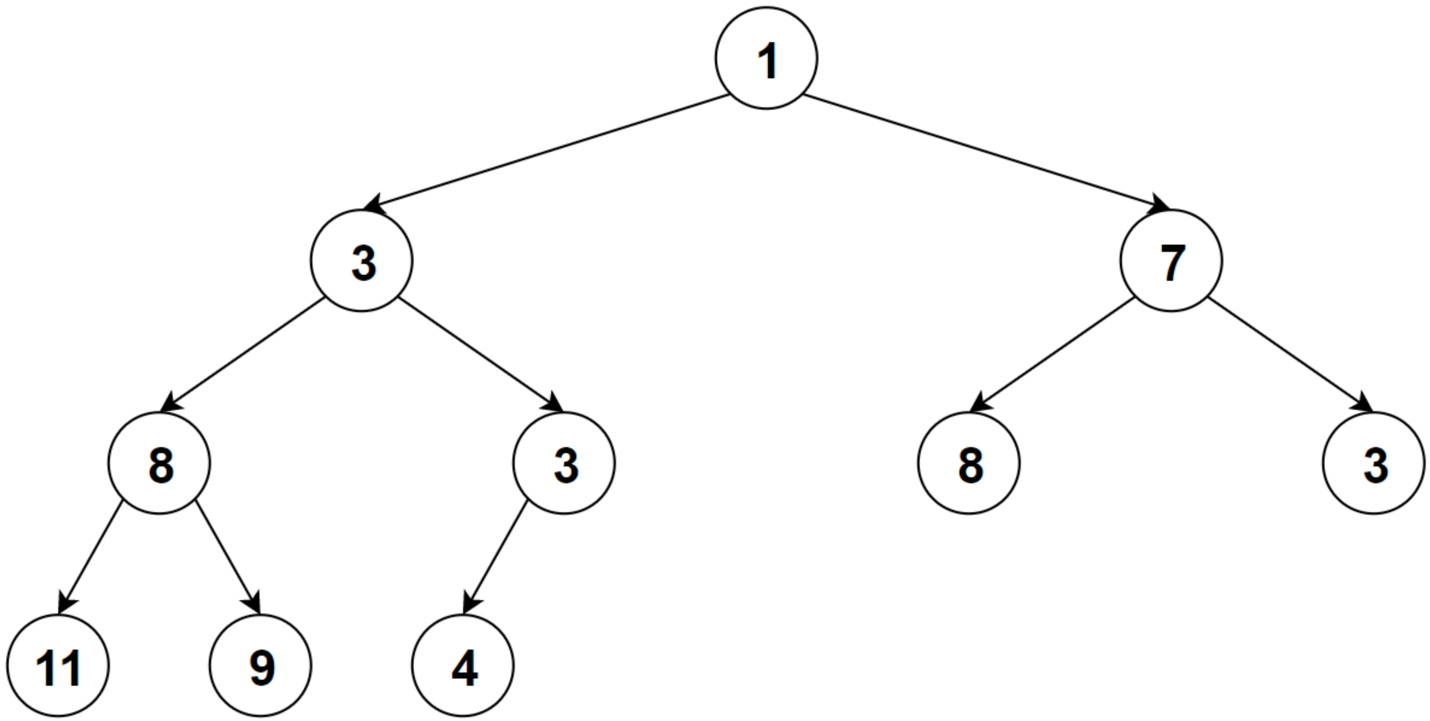


Image retrieved from https://www.programiz.com/dsa/priority-queue

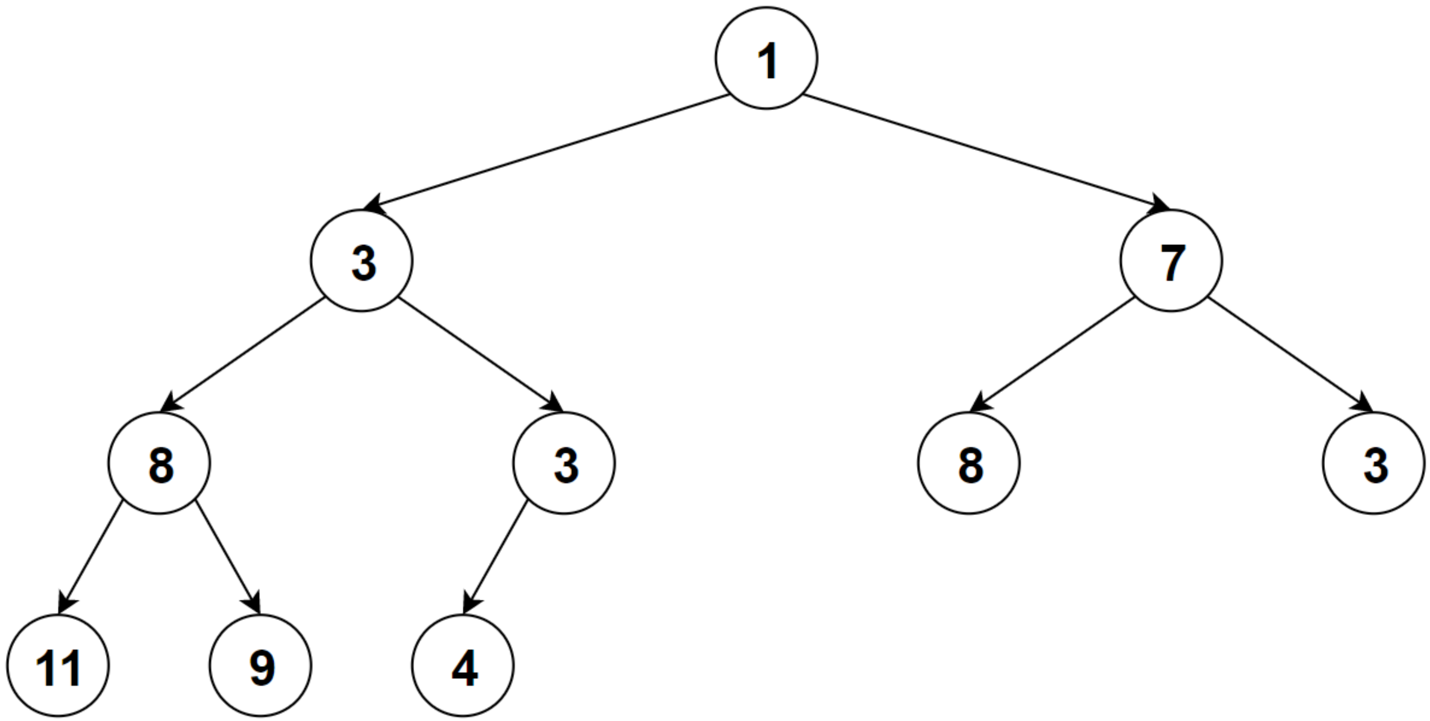
Binary Heap



Binary Heap is a type of priority queue.

- It is a complete binary tree: All levels are filled, except possibly the last level

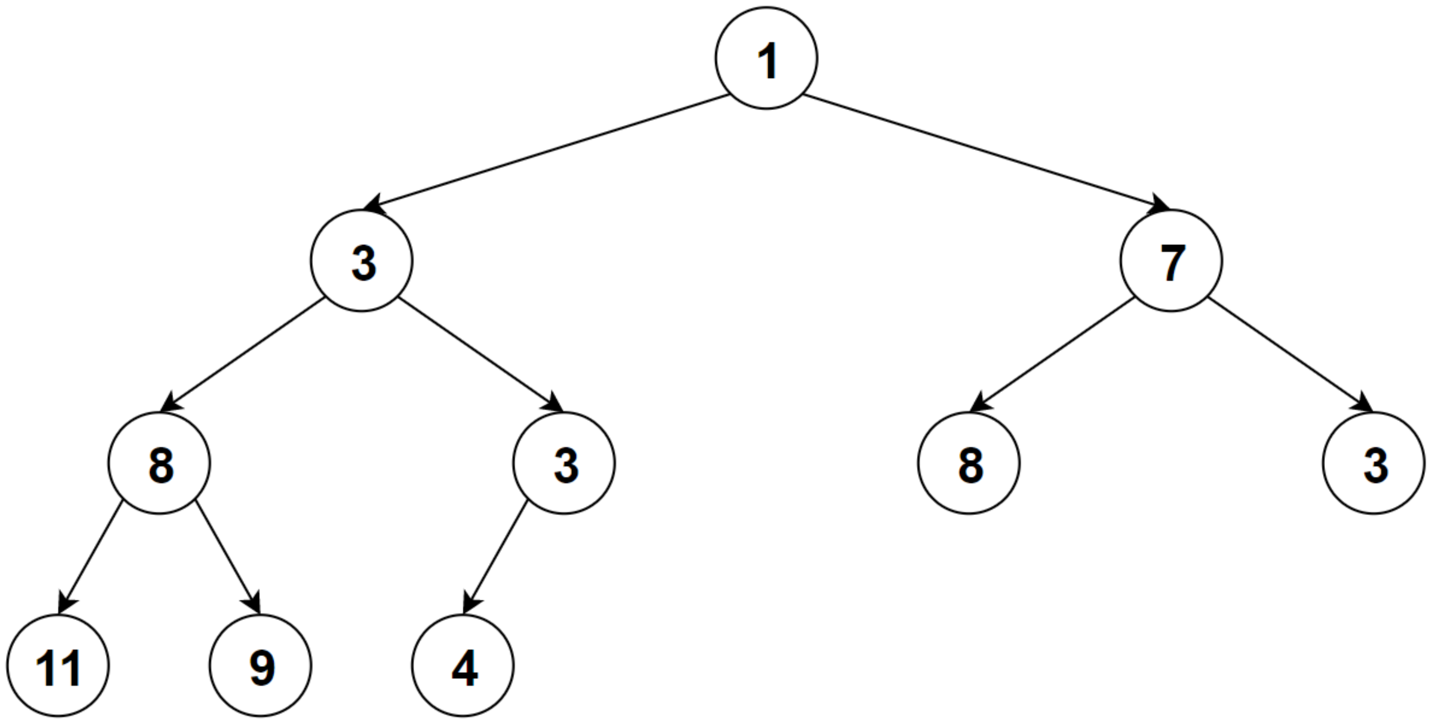
Binary Heap



Satisfies the heap property: Key of each node is smaller than the key of its children nodes(min heap):

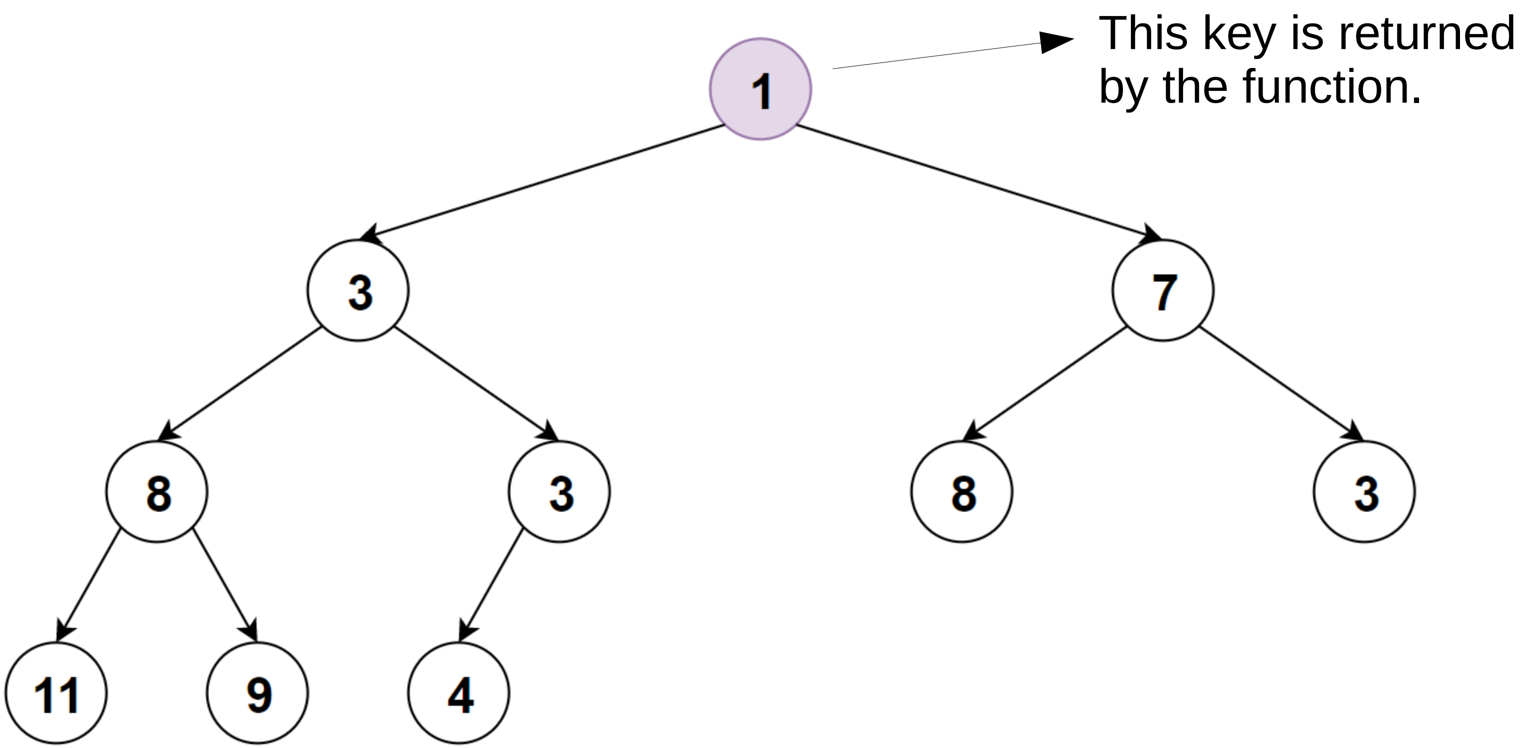
$$key(a) \geq key(parent(a))$$

Binary Heap – extract_min



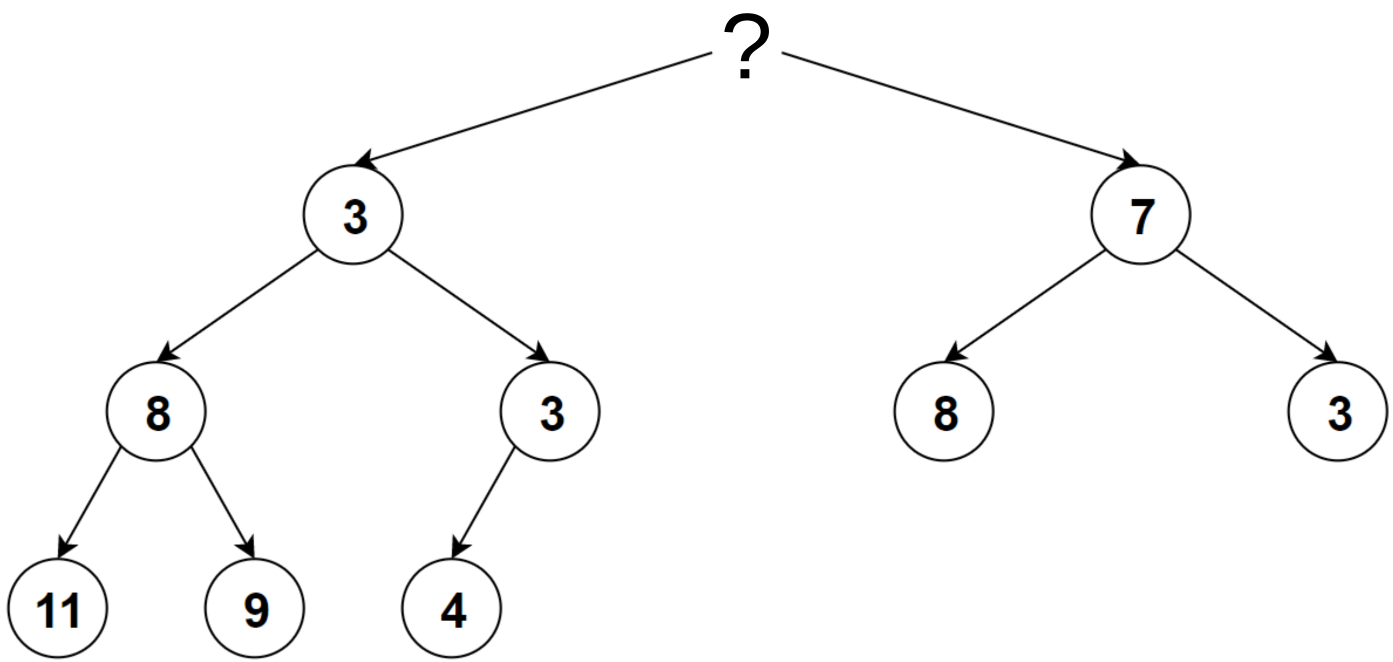
extract_min(): Delete the root item and return it.

Binary Heap – extract_min



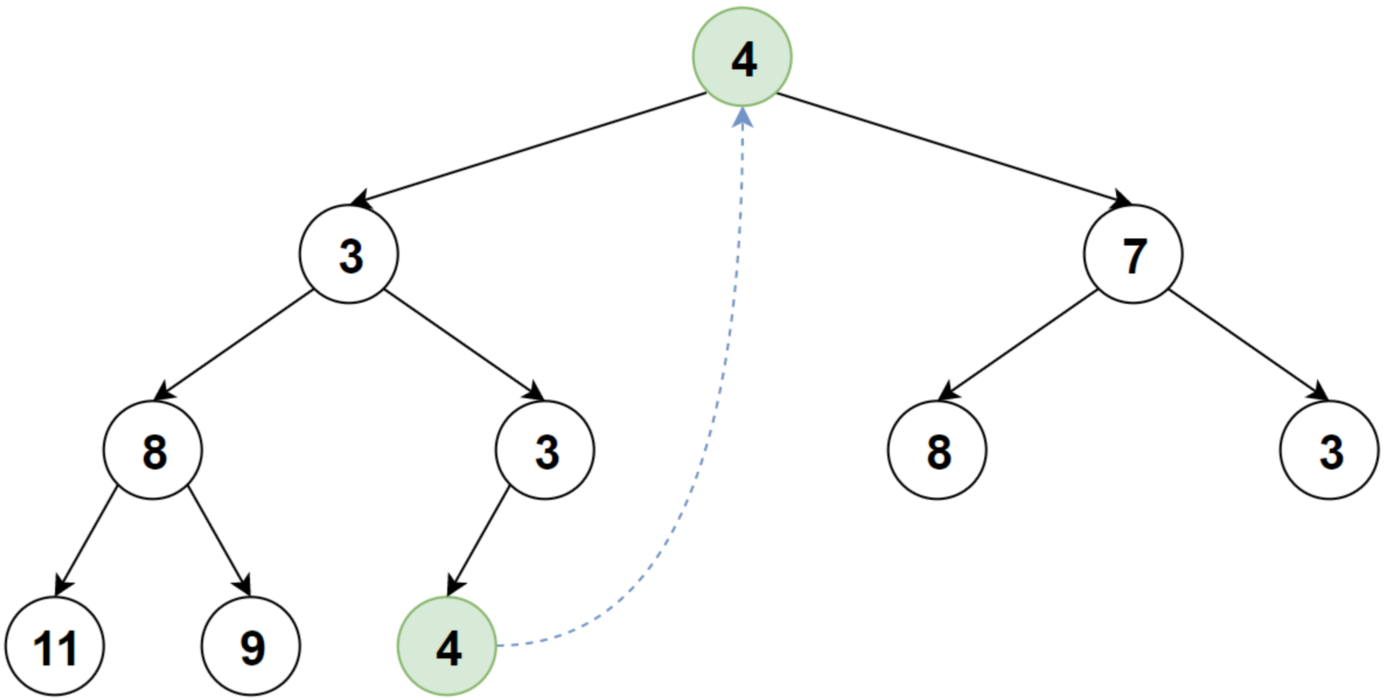
extract_min(): Pop the minimum element from the heap.

Binary Heap – extract_min



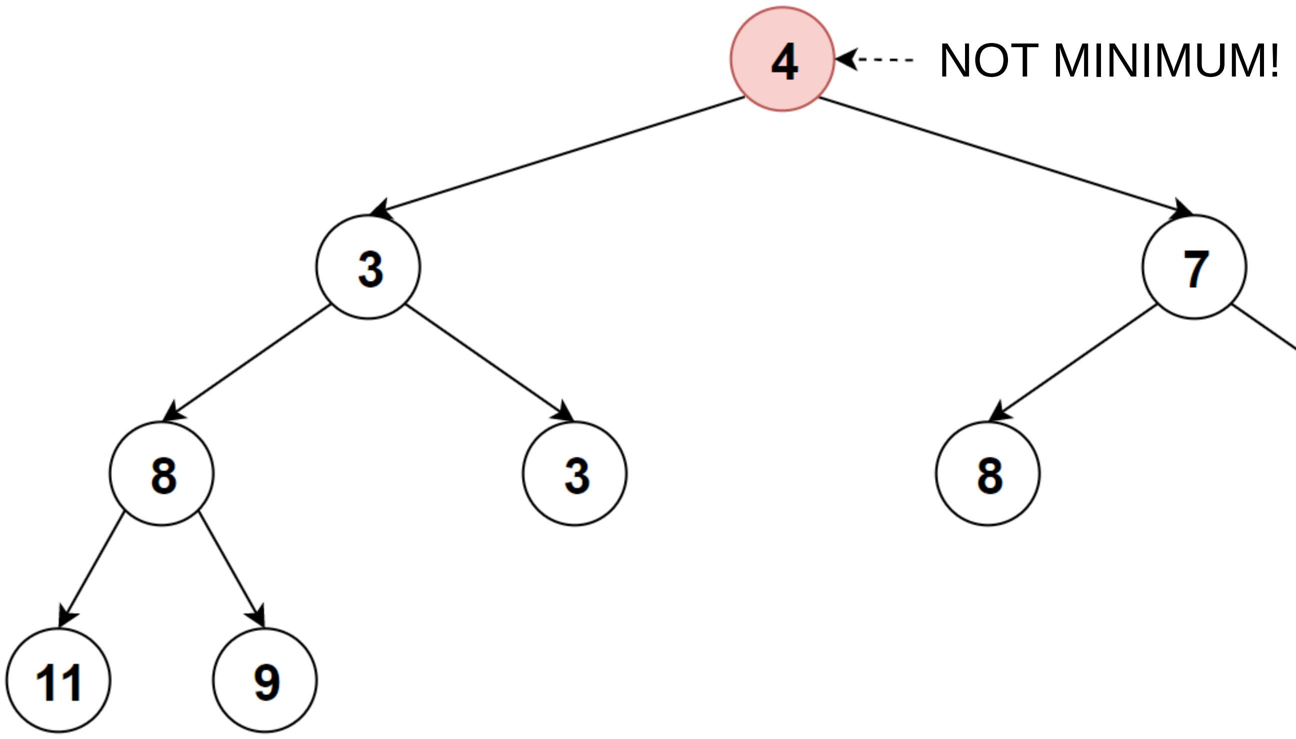
extract_min(): Pop the minimum element from the heap. **Now, we need to replace root!**

Binary Heap – extract_min



Replace root with the **last leaf element!**

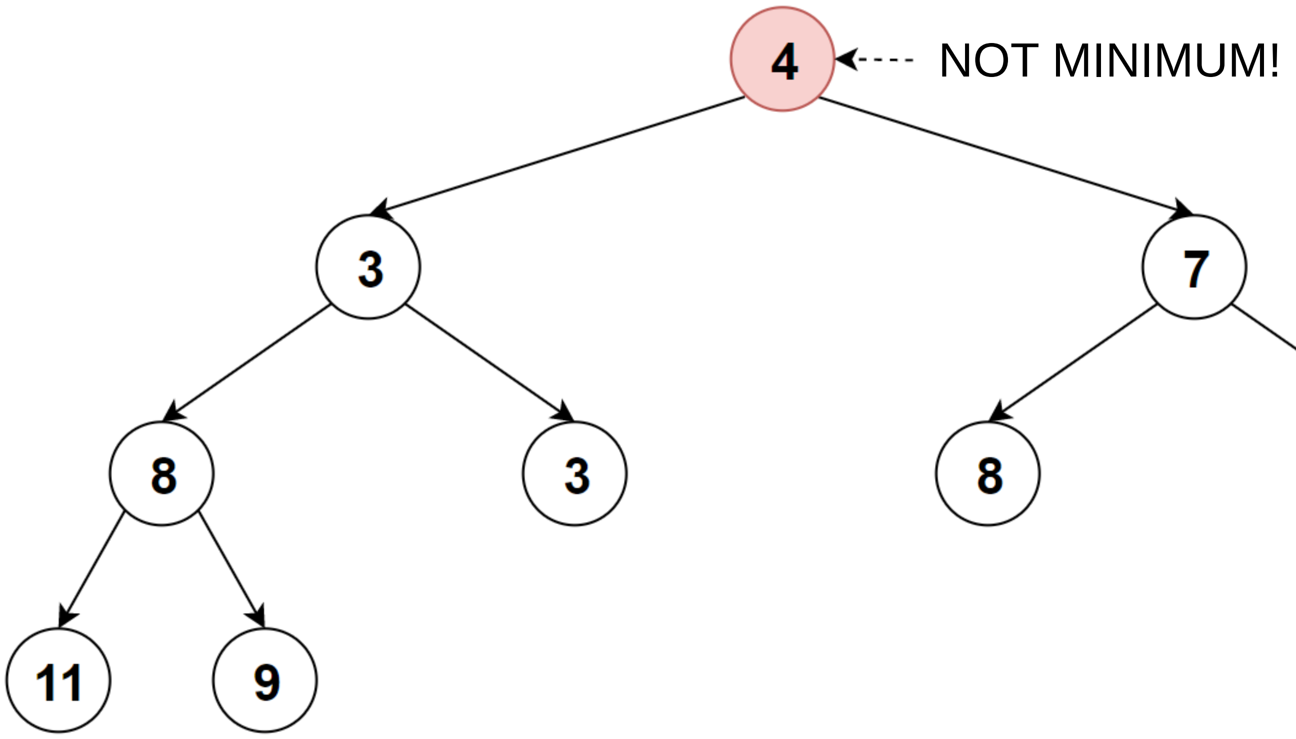
Binary Heap – extract_min



However, the root is not minimum now, therefore heap property does not hold anymore!

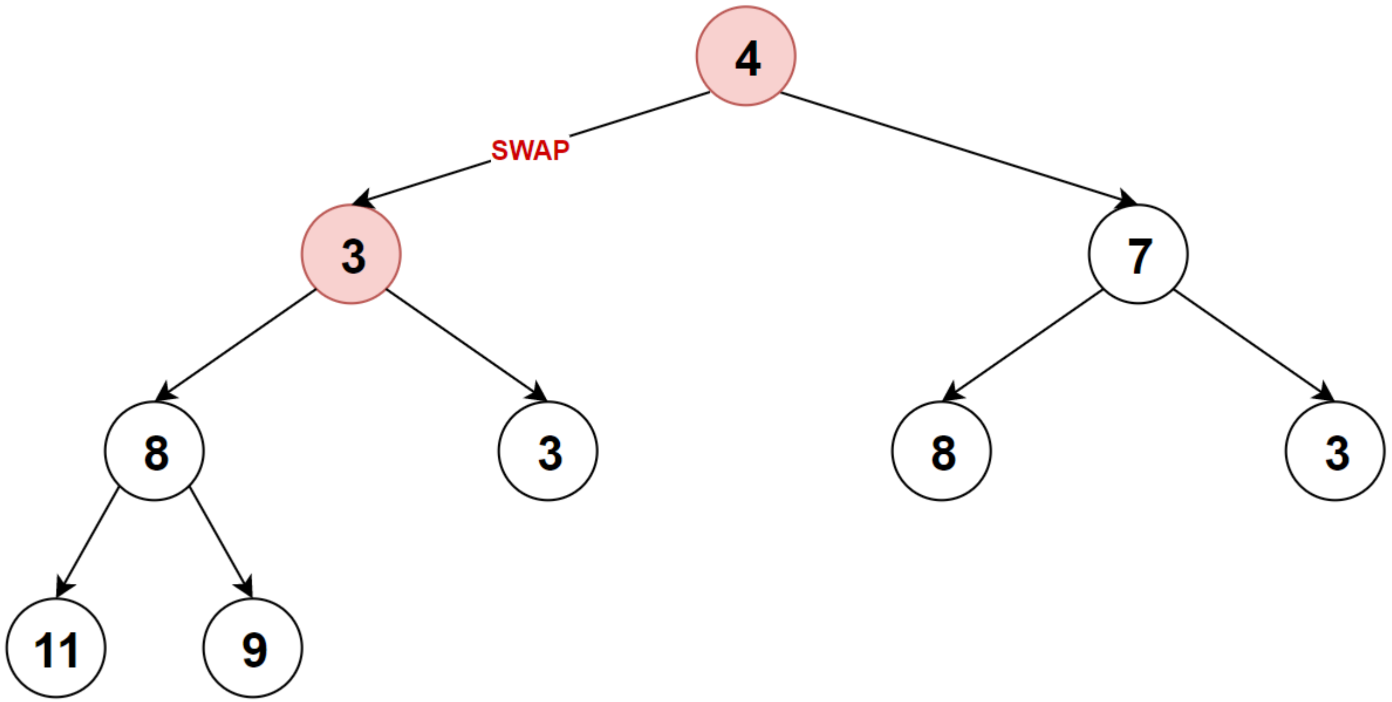
We need to fix it!

Binary Heap – extract_min



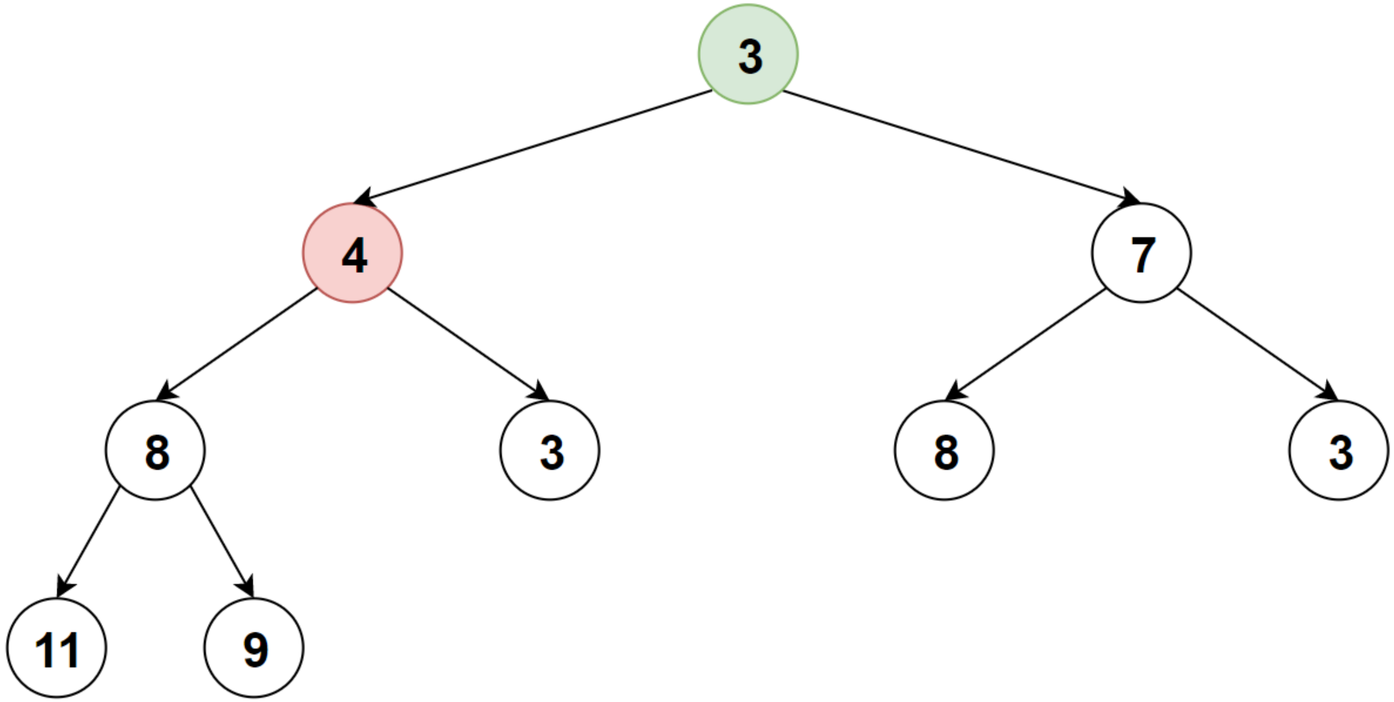
Until the **key 4** is smaller than or equal to its children, we need to swap it with its smallest child key.

Binary Heap – extract_min



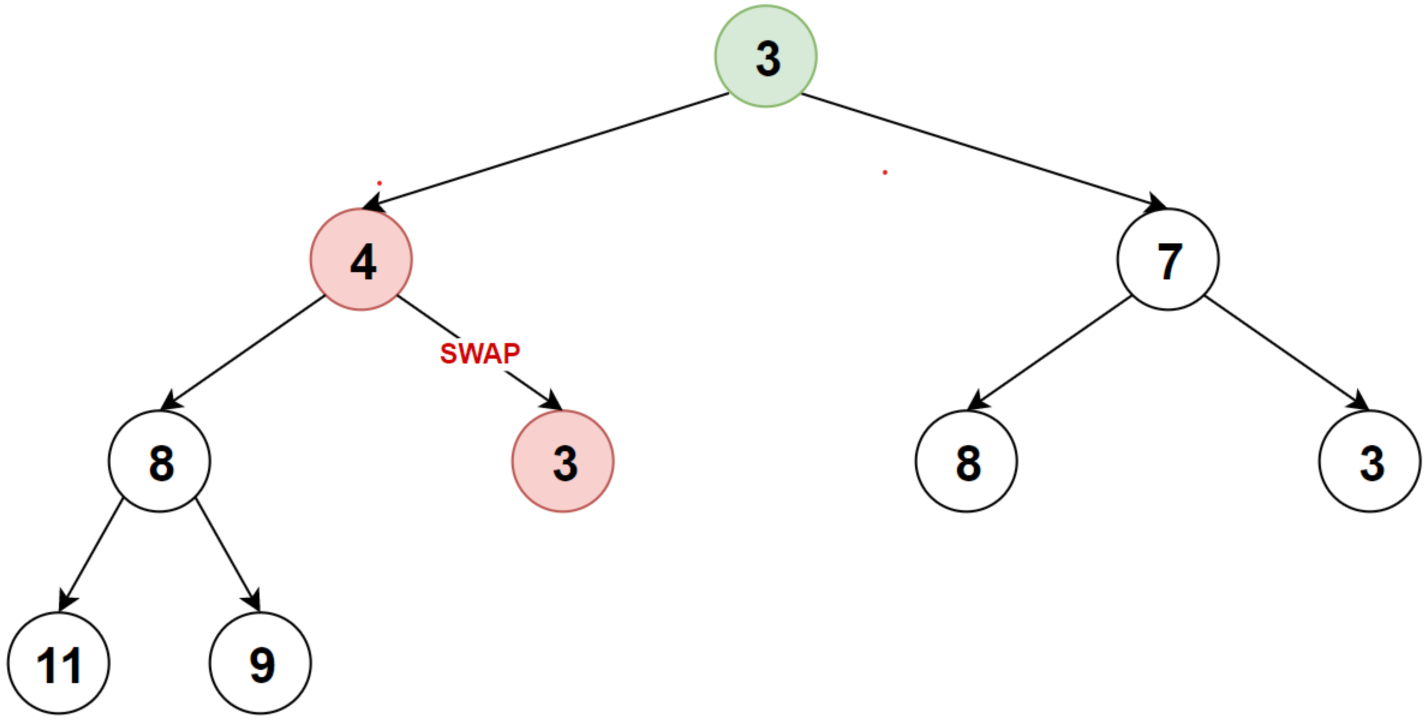
Until the **key 4** is smaller than or equal to its children, we need to swap it with its smallest child key. -> **SWAP 4 <-> 3**

Binary Heap – extract_min



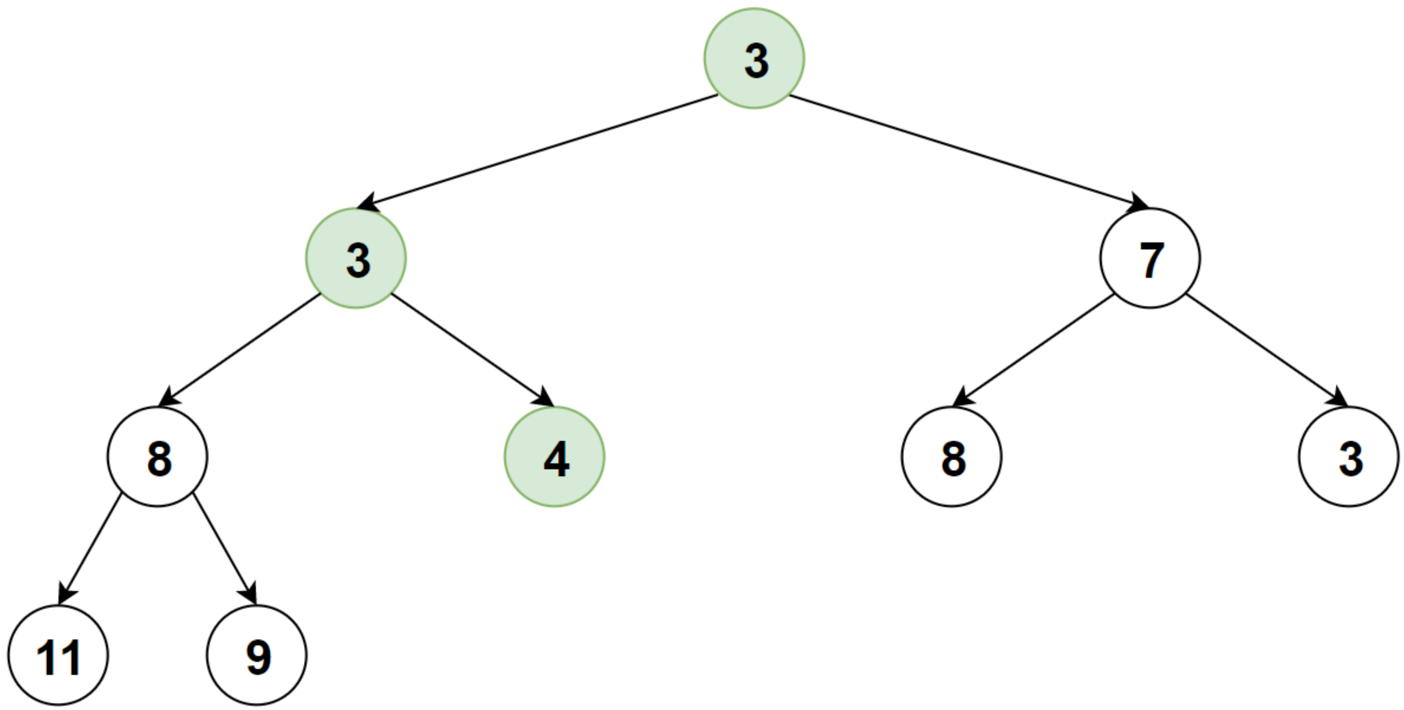
Until the **key 4** is smaller than or equal to its children, we need to swap it with its smallest child key.

Binary Heap – extract_min



Until the **key 4** is smaller than or equal to its children, we need to swap it with its smallest child key. -> **SWAP 4 <-> 3**

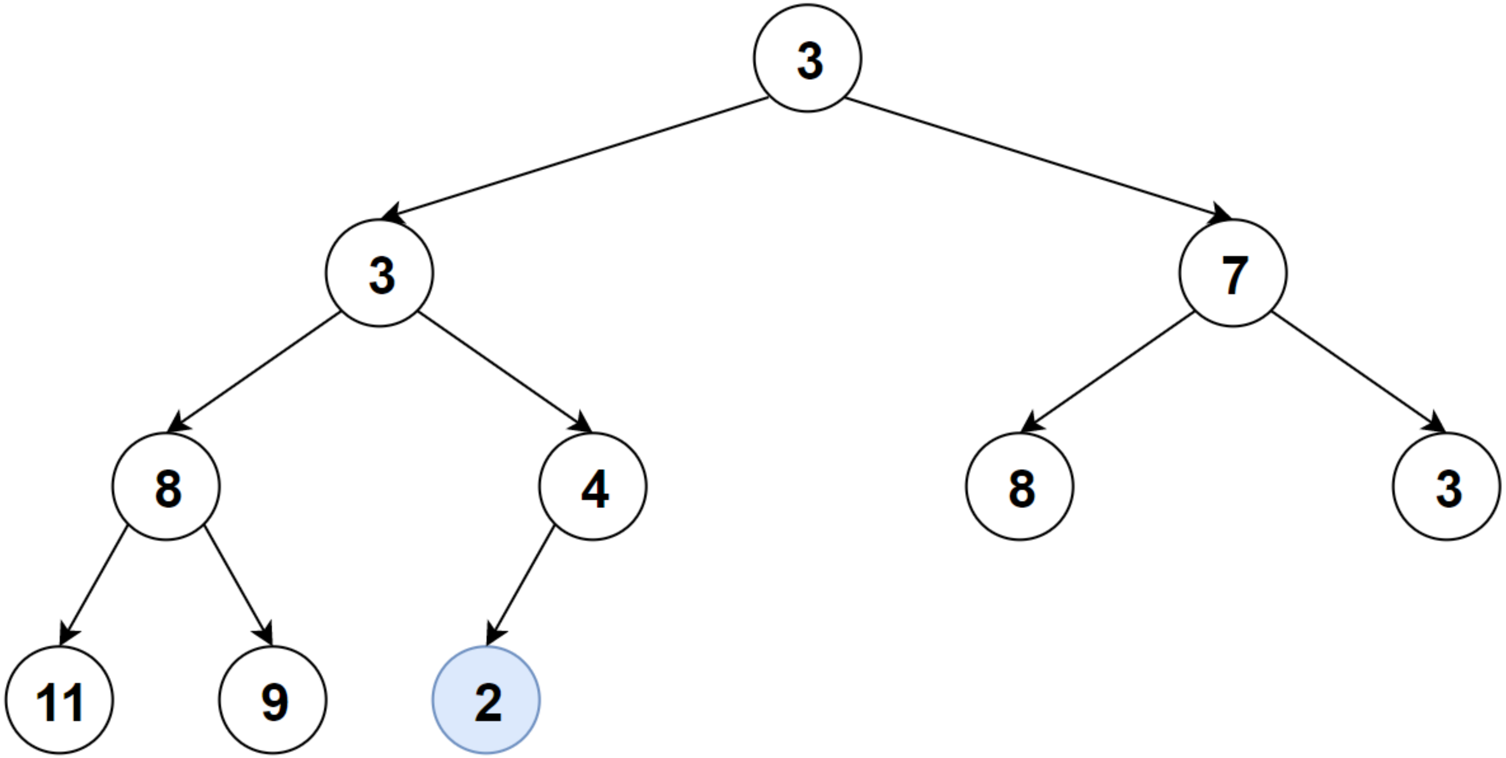
Binary Heap – extract_min



Now the heap property is satisfied! extract_min() function finishes by returning **key 1**(The root in the beginning).

- Time complexity? O(logn)

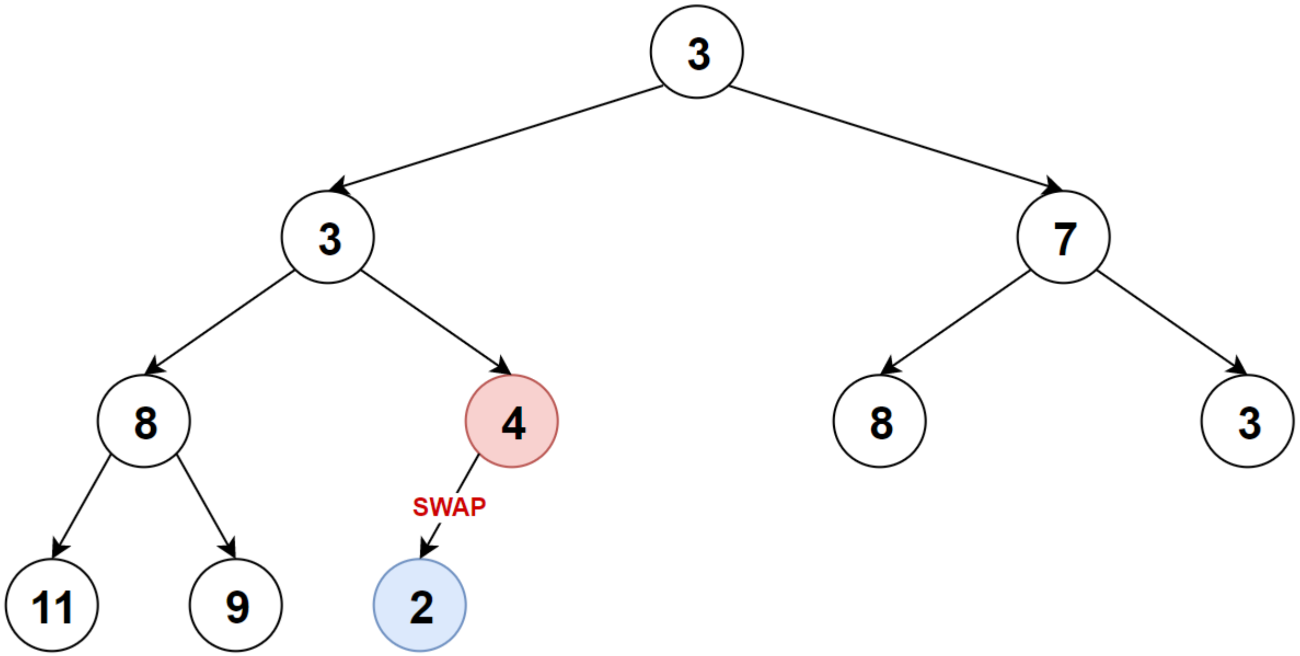
Binary Heap - insert



insert(): inserts a new key as the very last leaf key.

But, heap property is again gone! Now, we need to keep swapping **key 2** with its parent until heap property is satisfied!

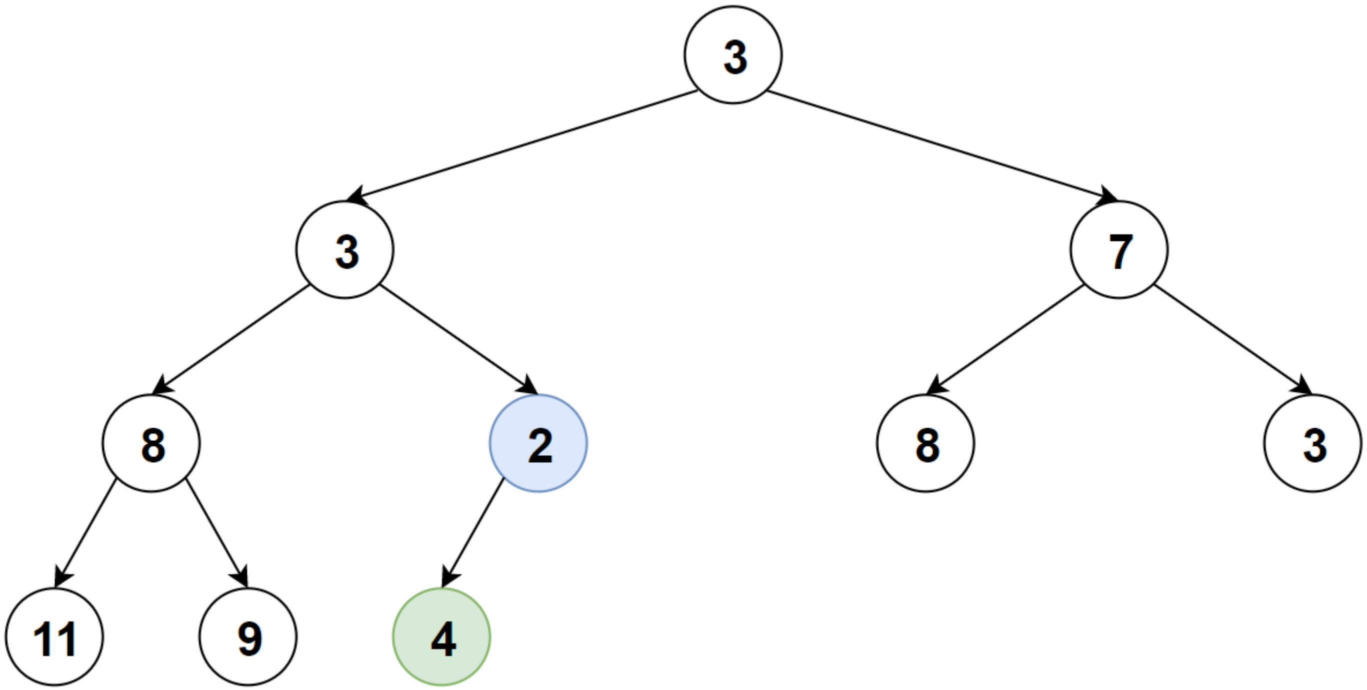
Binary Heap - insert



SWAP 2 <-> 4

Now, we need to keep swapping **key 2** with its parent until heap property is satisfied!

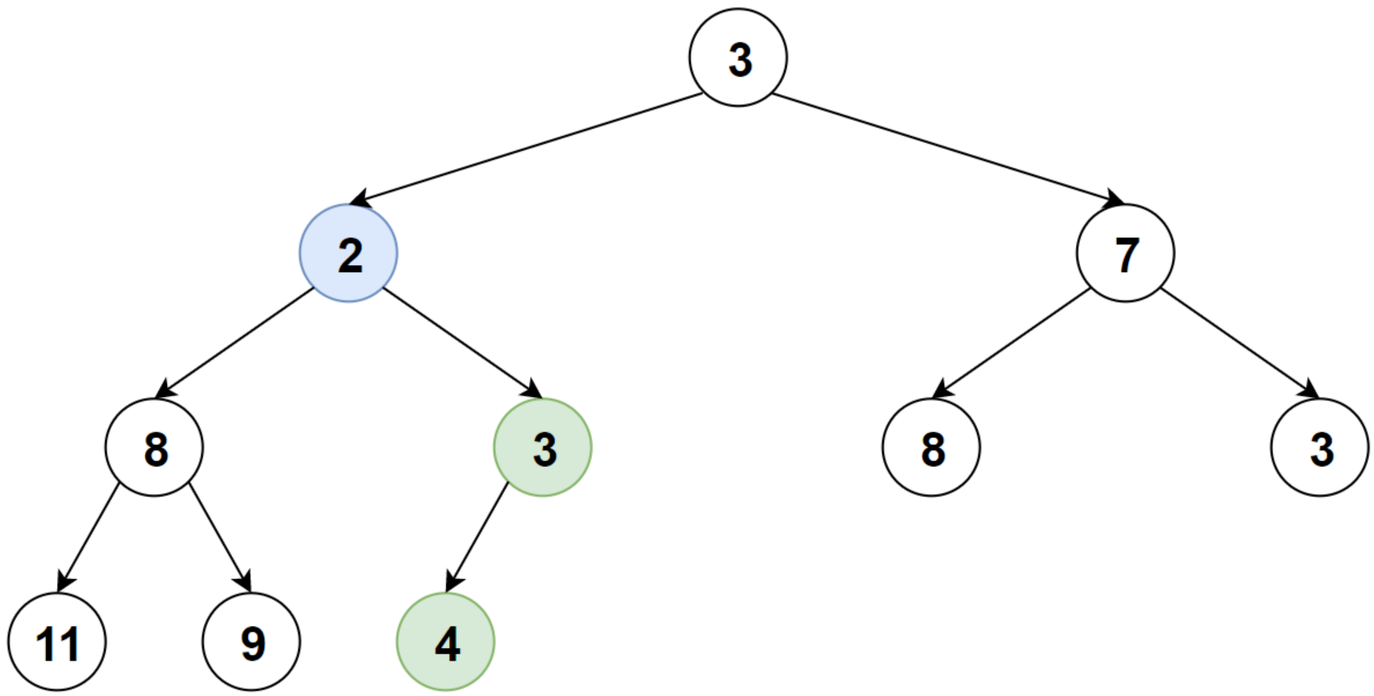
Binary Heap - insert



SWAP 2 <-> 3

Now, we need to keep swapping **key 2** with its parent until heap property is satisfied!

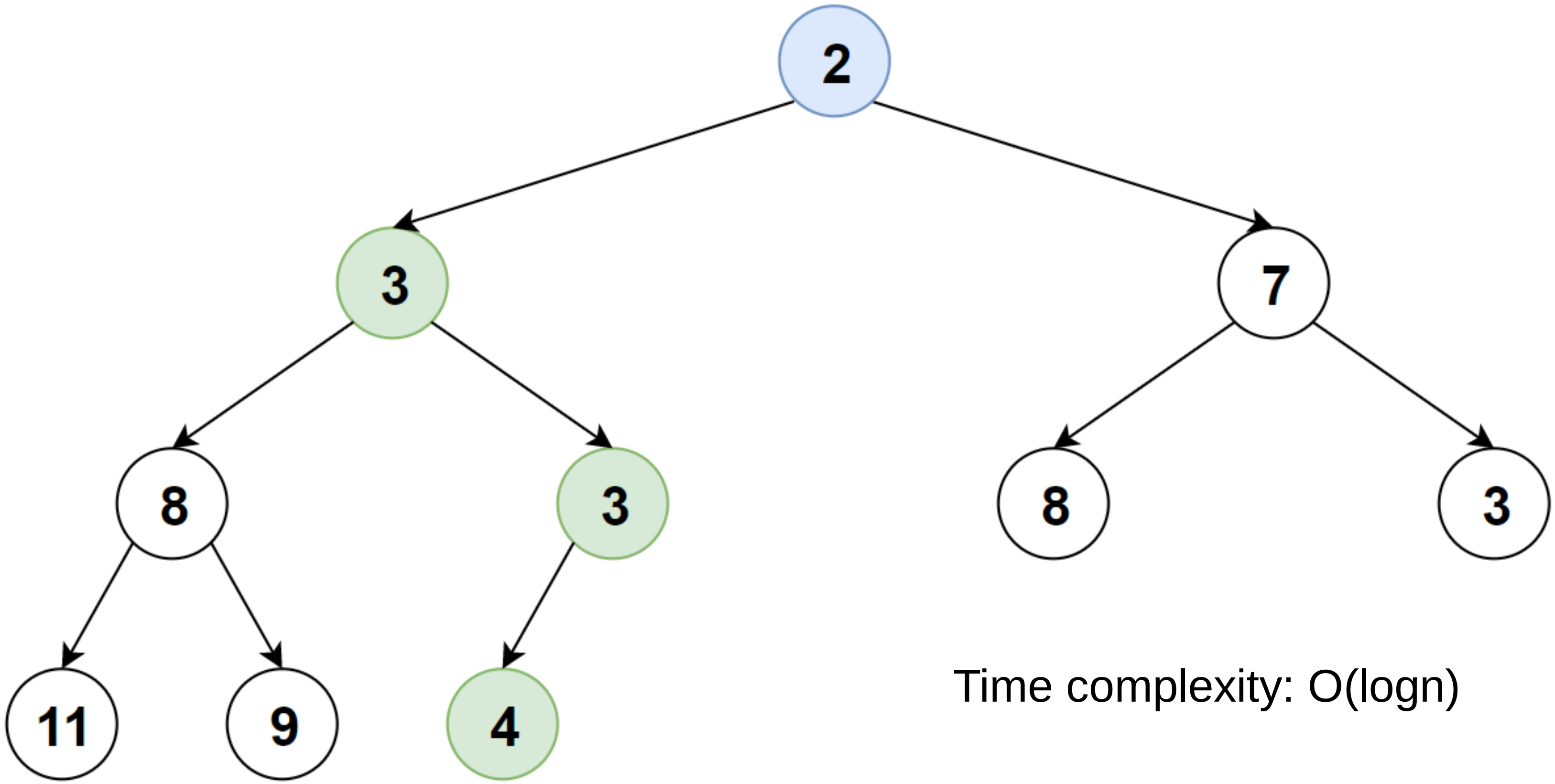
Binary Heap - insert



SWAP 2 <-> 3

Now, we need to keep swapping **key 2** with its parent until heap property is satisfied!

Binary Heap - insert

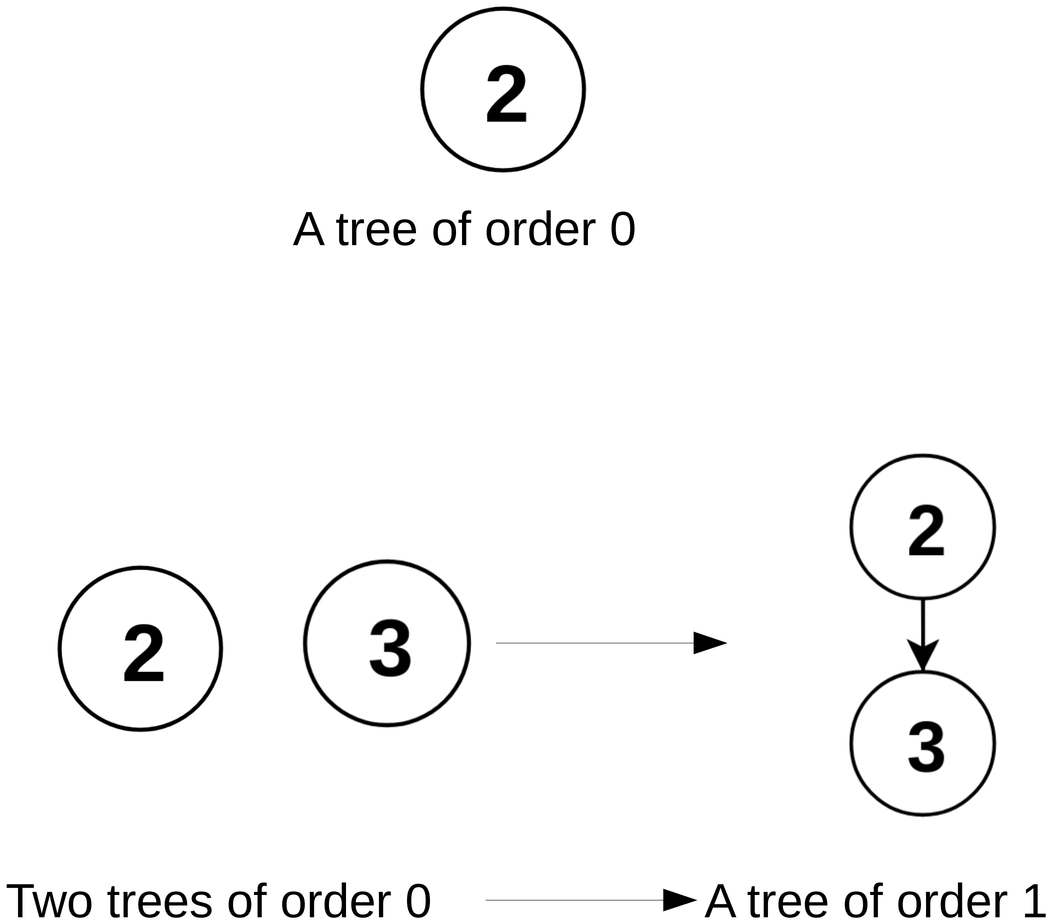


Time complexity: O(logn)

The heap property is satisfied! Insertion of **key 2** is complete!

- Decrease_key operation decreases the key of an element, then performs the same bubble up operations as in “insert” operation.

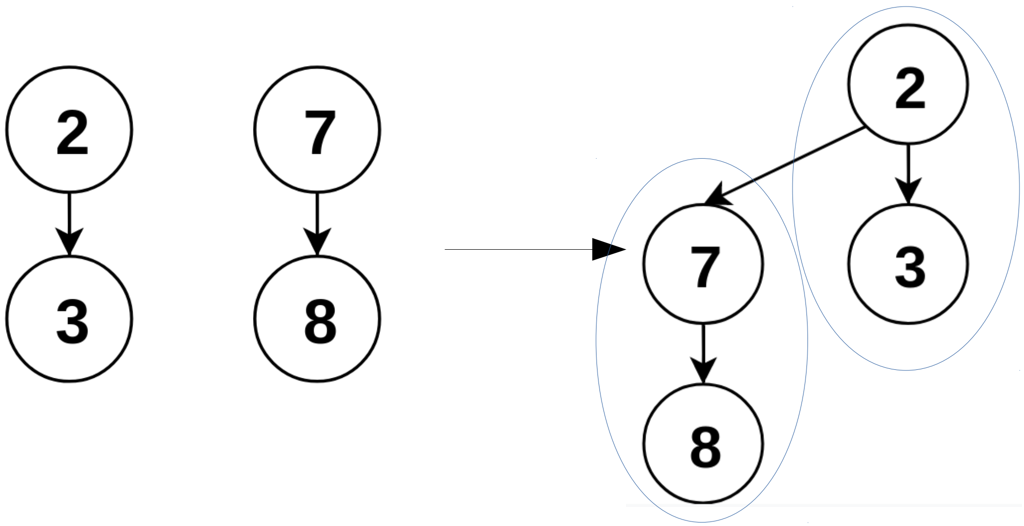
Binomial Tree



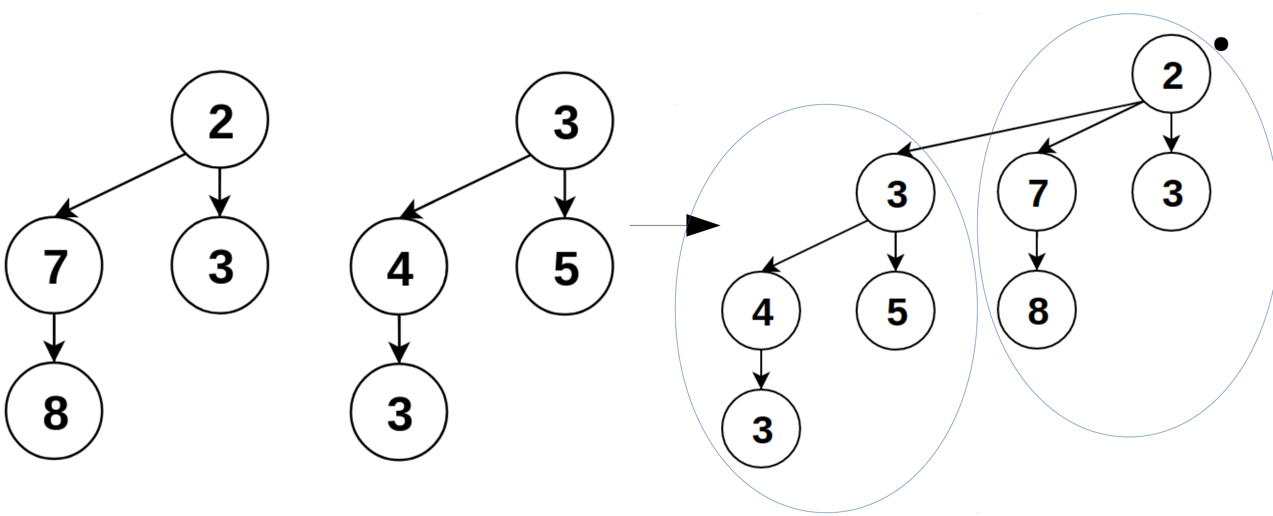
Binomial tree is a set of nodes such that :

- A tree of order 0 is a single node and
- A tree of order k is combination of two trees of order k-1. To combine, one of the trees becomes the leftmost child of the other.

Binomial Tree



Two trees of order 1 → A tree of order 2



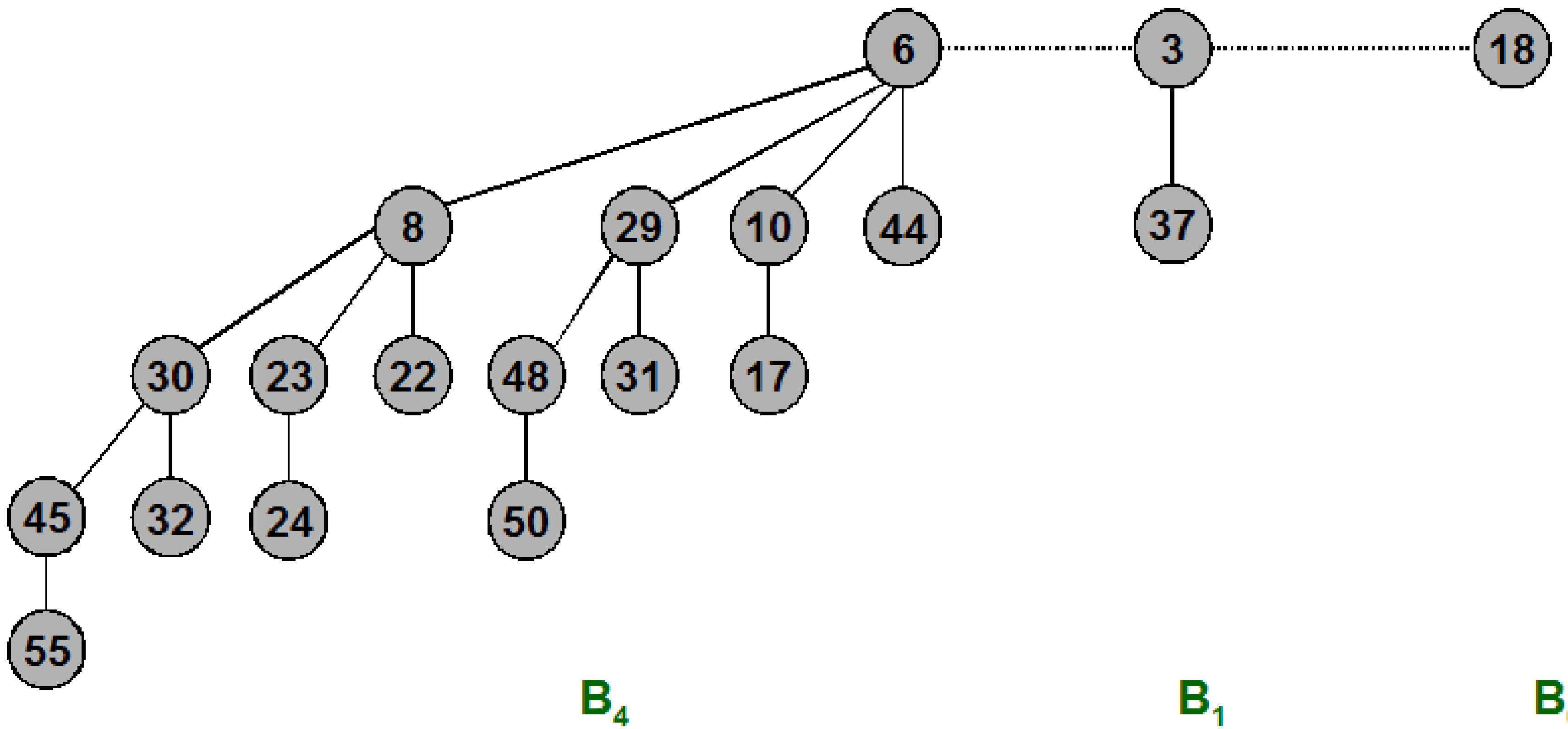
Two trees of order 2 → A tree of order 3

If the order of binomial tree is k, then:

- It has 2^k elements.
- Degree of root is k, a.k.a the root has k children.
- Its depth is k
- Each child of root is a binomial tree with orders 0, 1, 2..., k-1.

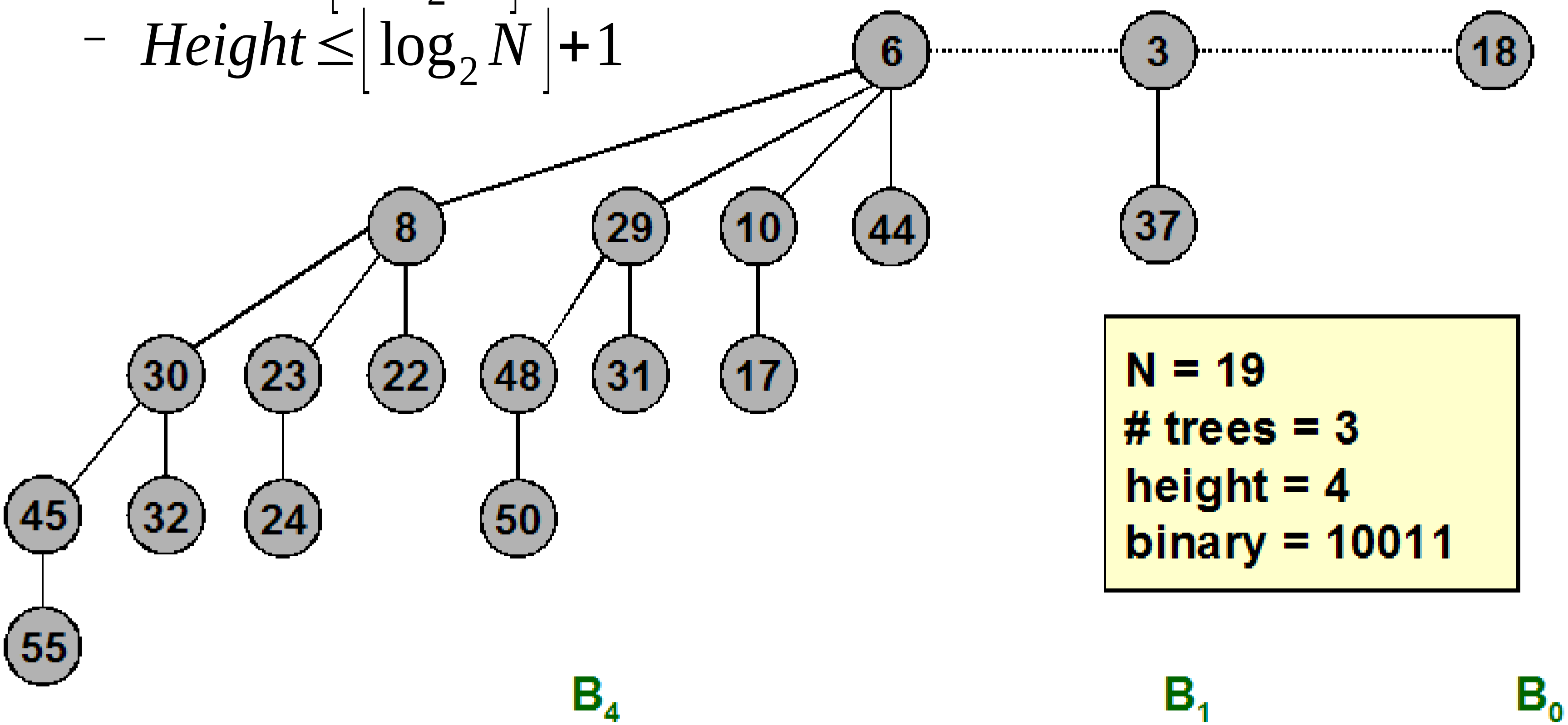
Binomial Heap

- Vuillemin, 1978
- Sequence of binomial trees that satisfy binomial heap property
  - each tree is min-heap ordered
  - 0 or 1 binomial tree of order k



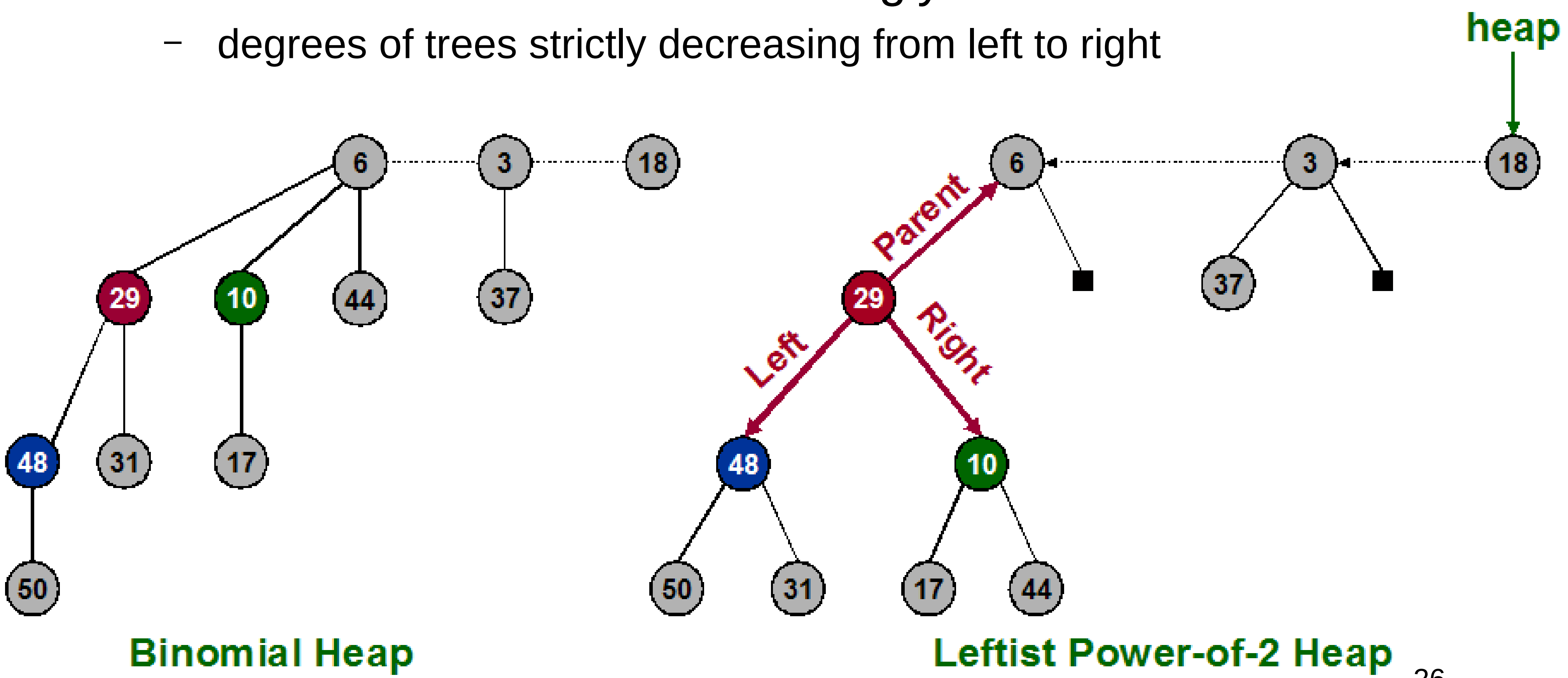
Binomial Heap: Properties

- Properties of N-node binomial heap
  - Min key contained in root of $B_0, B_1, \ldots, B_k$
  - Contains binomial tree B_i iff $b_i = 1$ where $b_n \cdot b_2 b_1 b_0$ is binary representation of N
  - At most $\lfloor \log_2 N \rfloor + 1$ binomial trees
  - $Height \leq \lfloor \log_2 N \rfloor + 1$



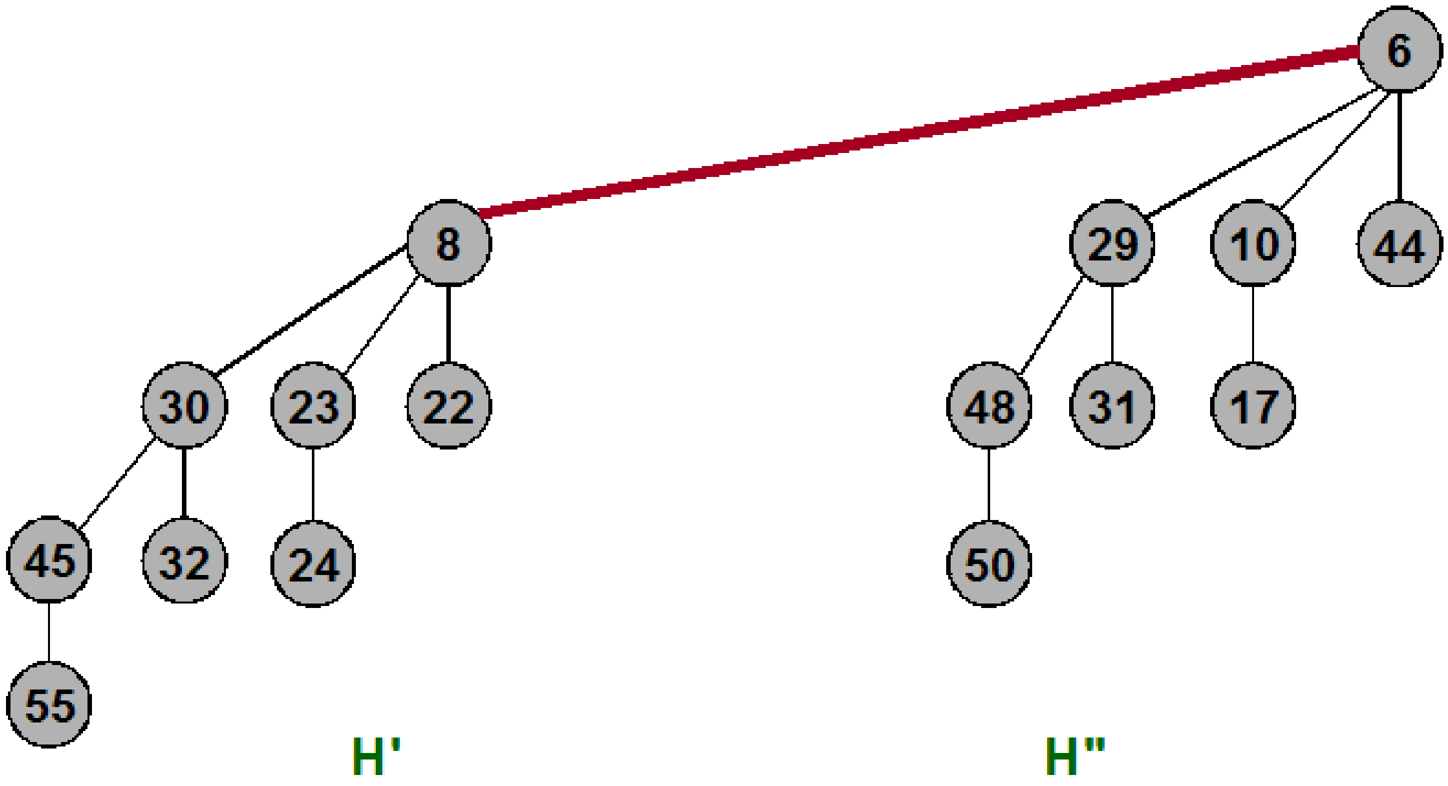
Binomial Heap: Implementation

- Represent trees using left-child, right sibling pointers [Ch. 10.4]
  - three links per node (parent, left, right)
- Roots of trees connected with singly linked list
  - degrees of trees strictly decreasing from left to right

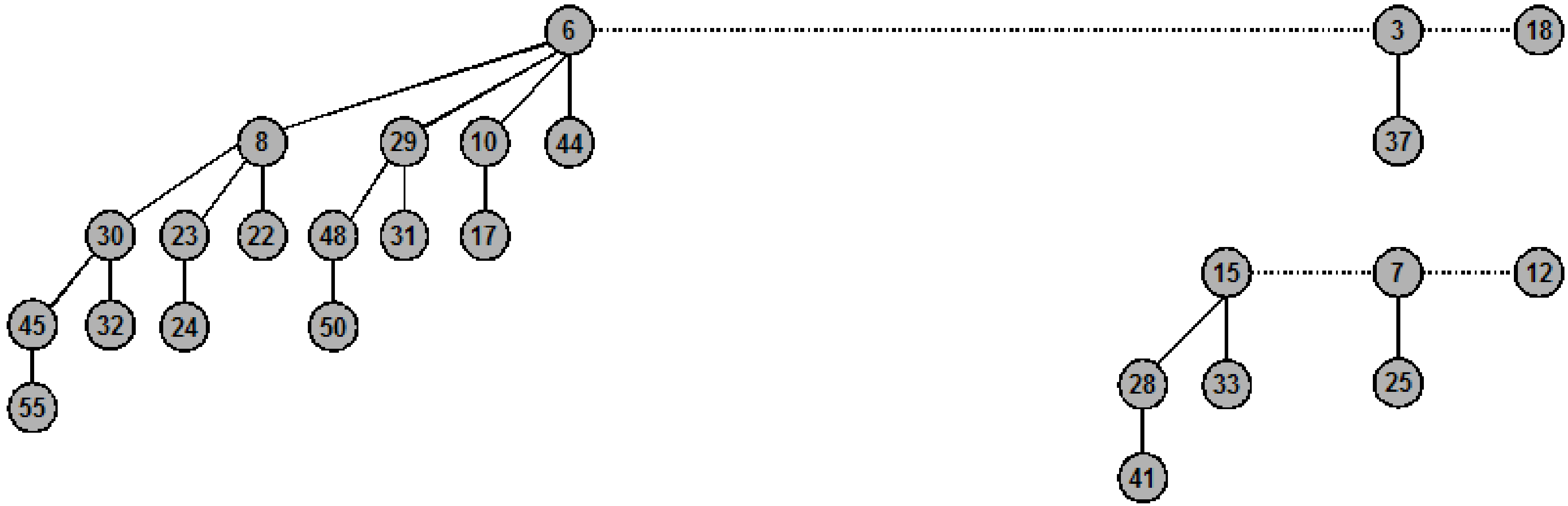


Binomial Heap: Union

- Create heap H that is union of heaps H' and H''
  - "Mergeable heaps"
  - Easy if H' and H'' are each order k binomial trees
    - connect roots of H' and H''
    - choose smaller key to be root of H



Binomial Heap: Union

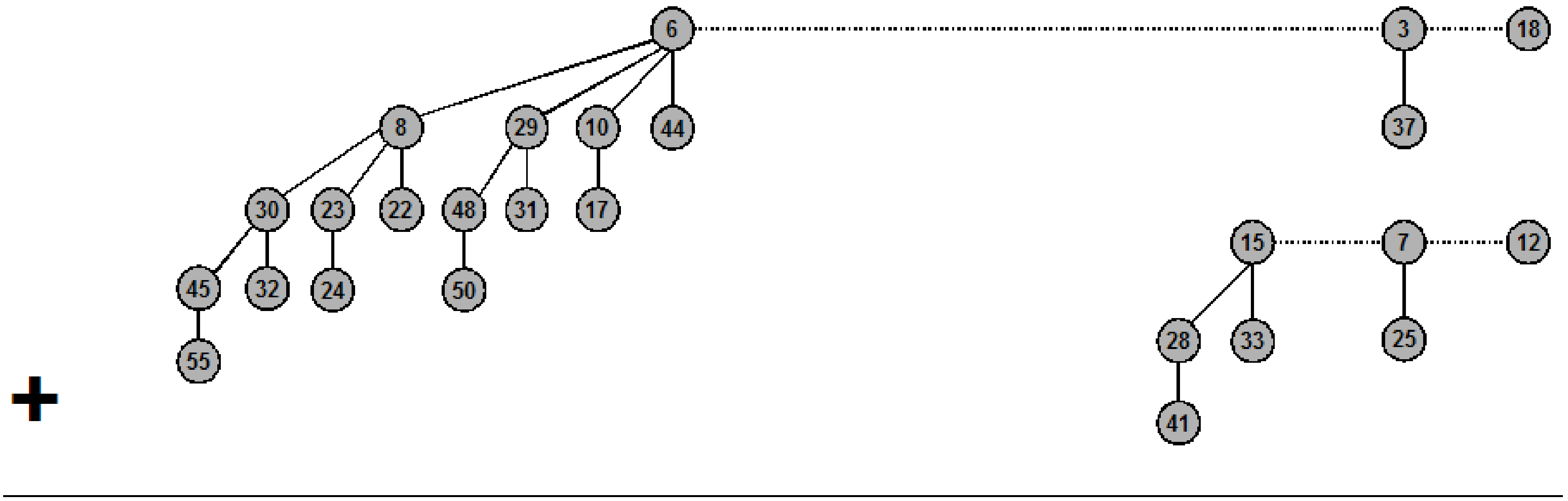


+

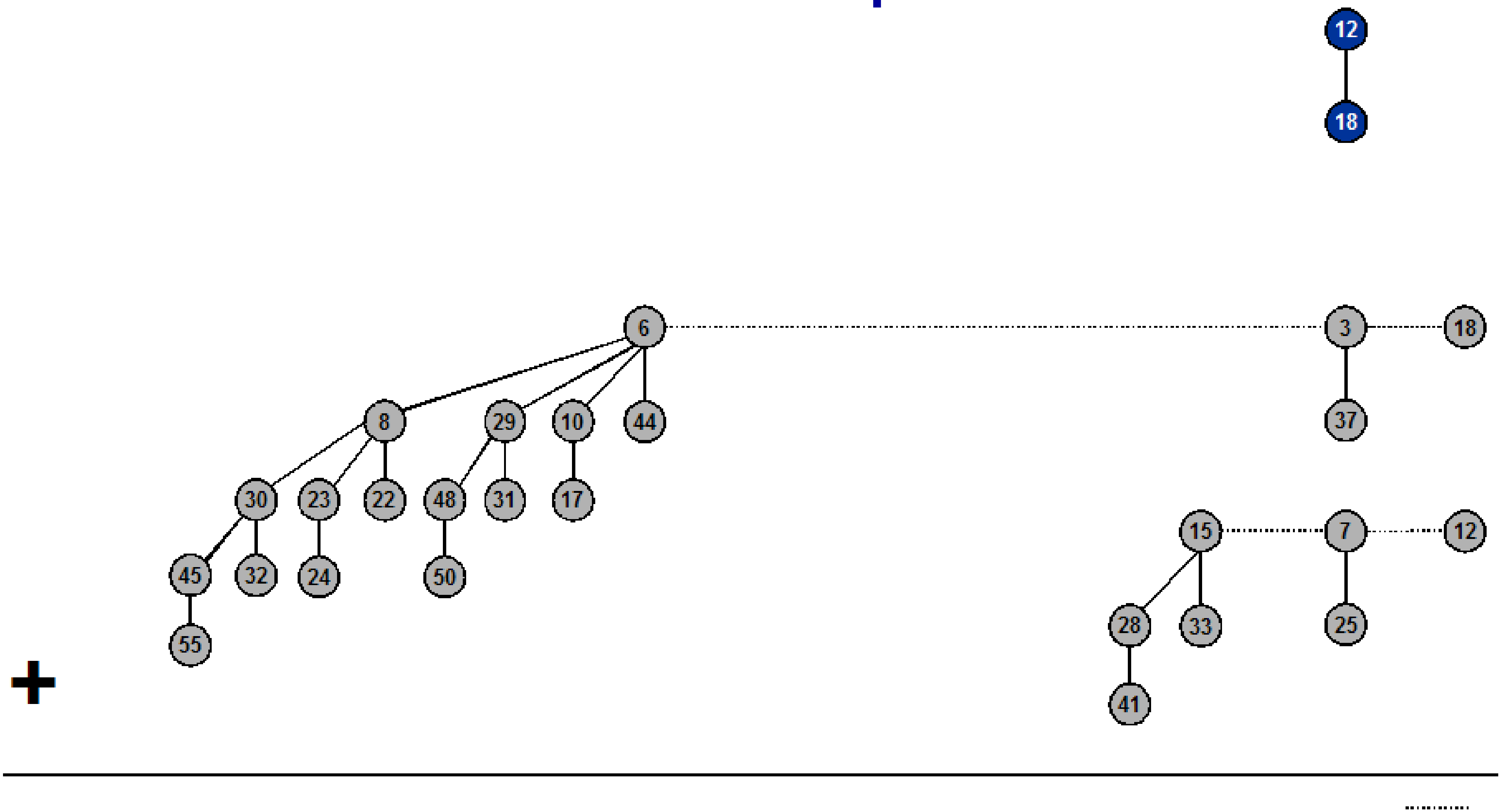
19 + 7 = 26

		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

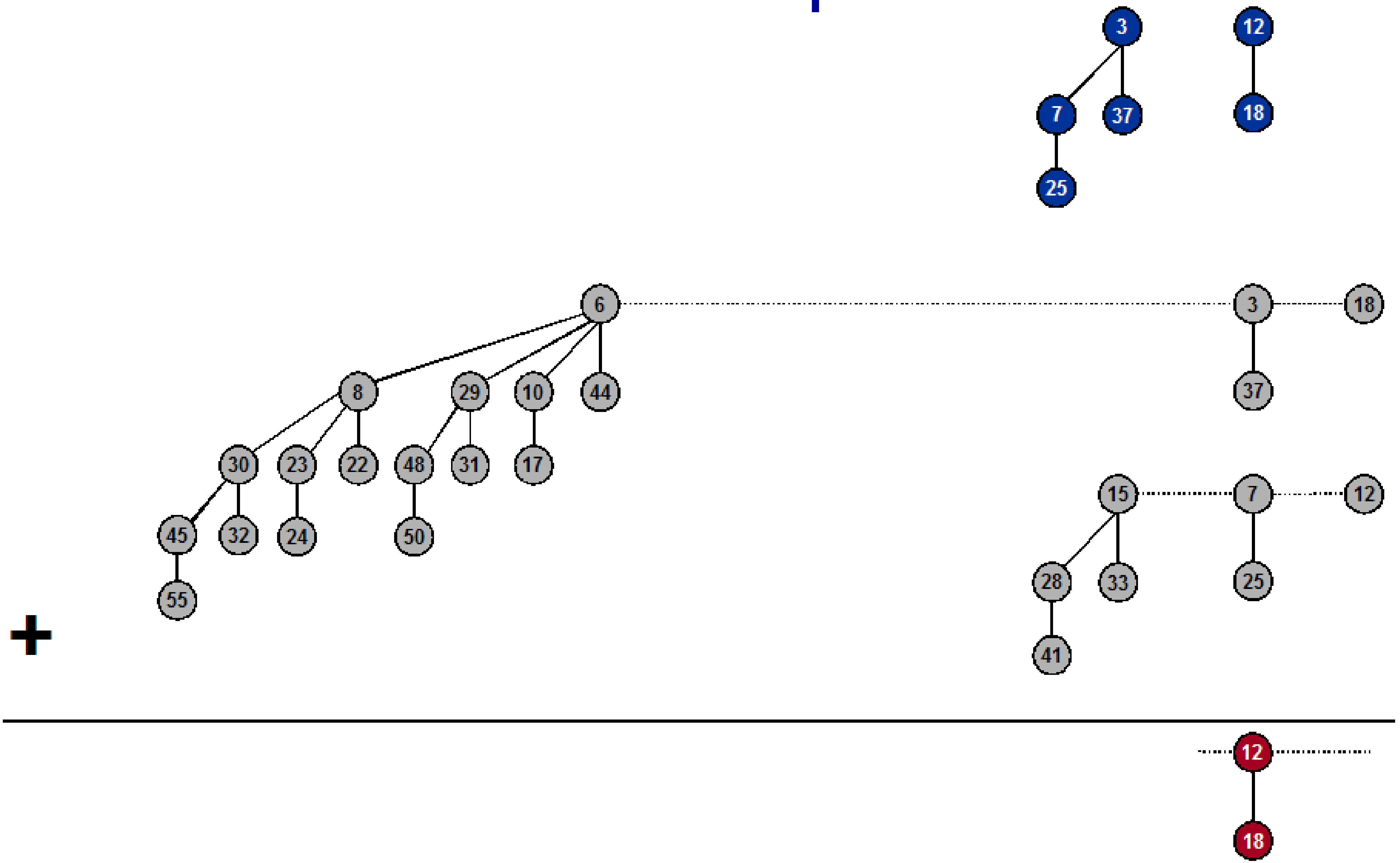
Binomial Heap: Union



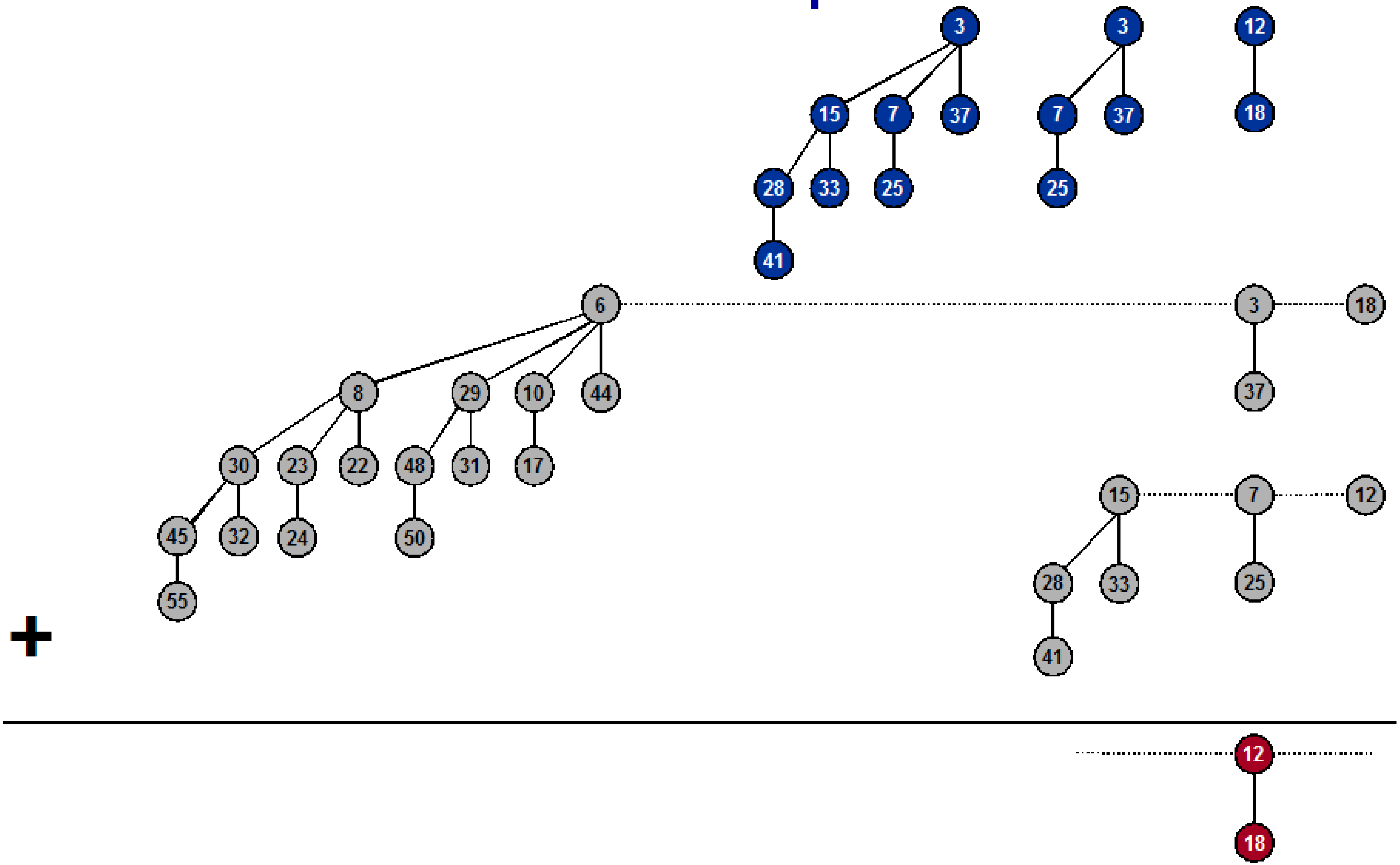
Binomial Heap: Union



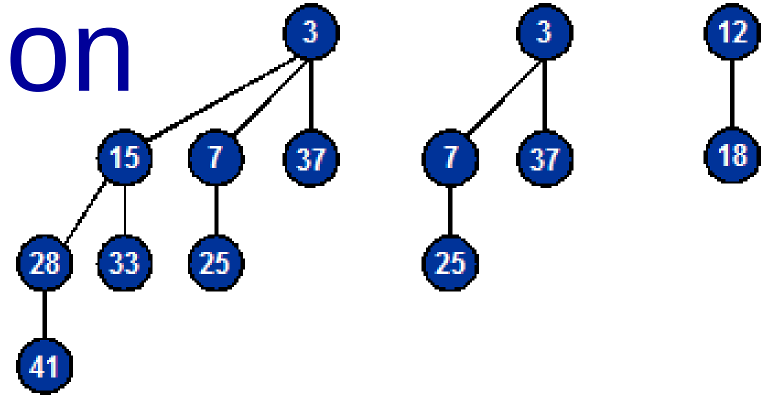
Binomial Heap: Union

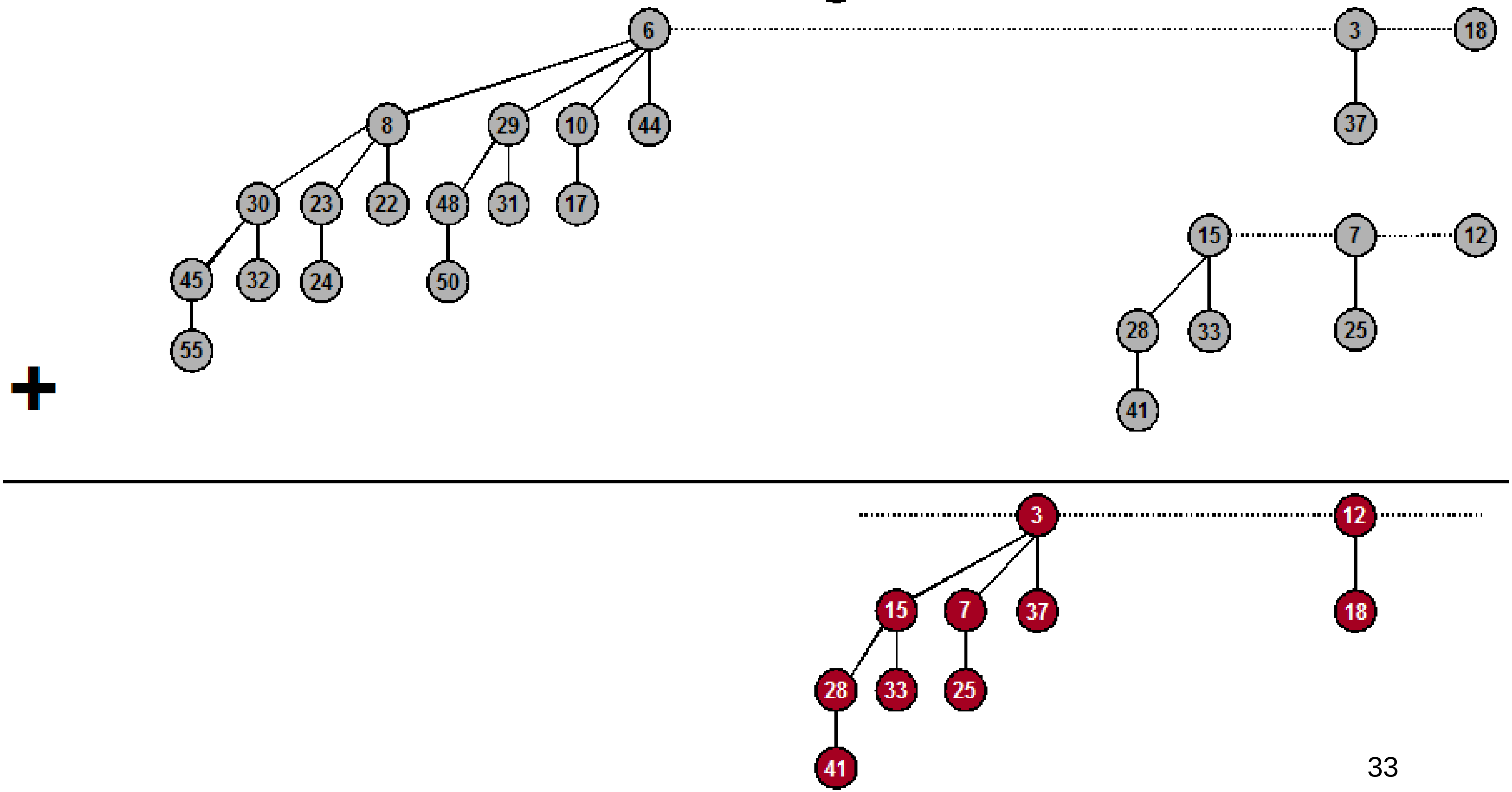


Binomial Heap: Union

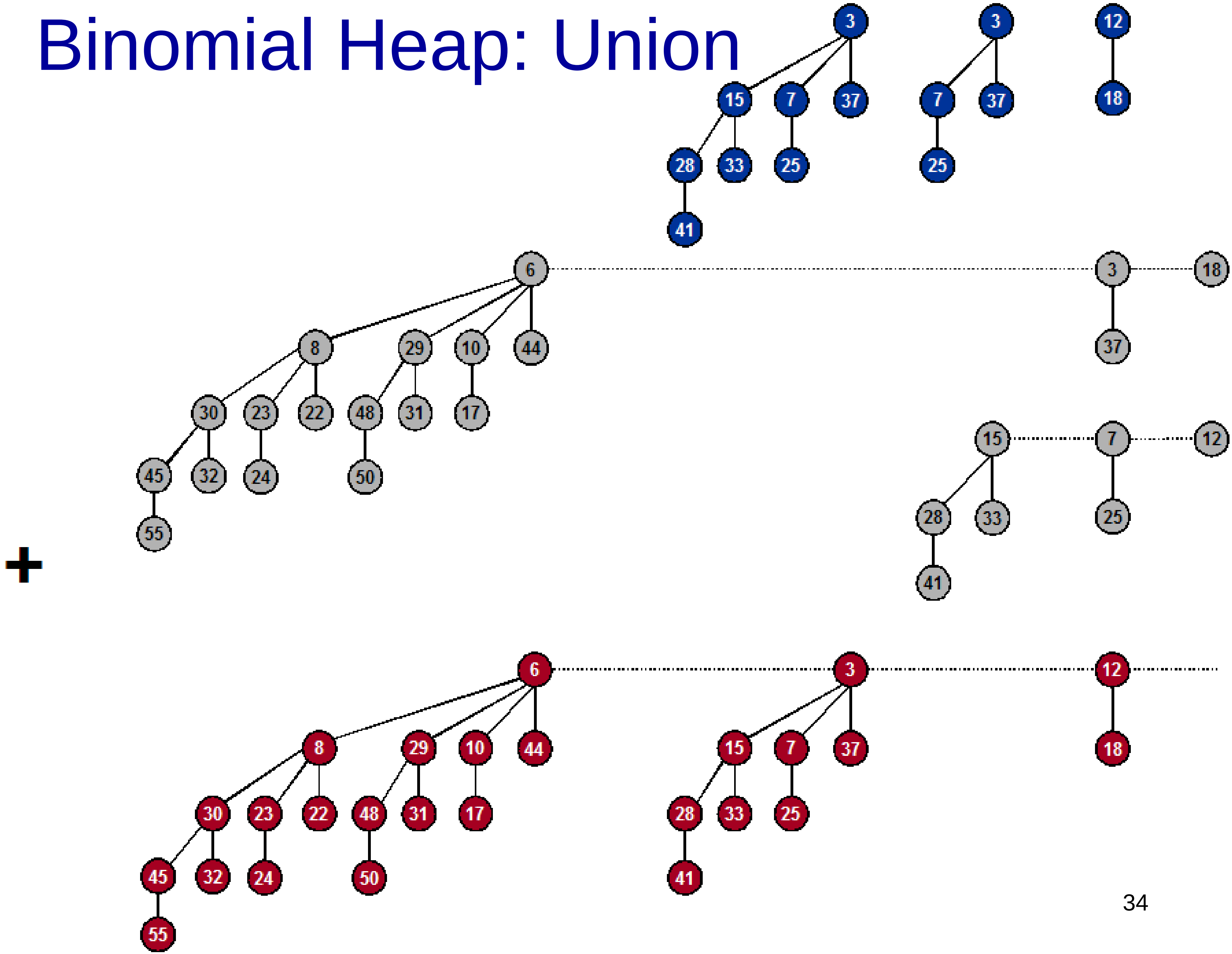


Binomial Heap: Union





Binomial Heap: Union



Binomial Heaps: Union

- Create heap H that is union of heaps H' and H''
  - Analogous to binary addition

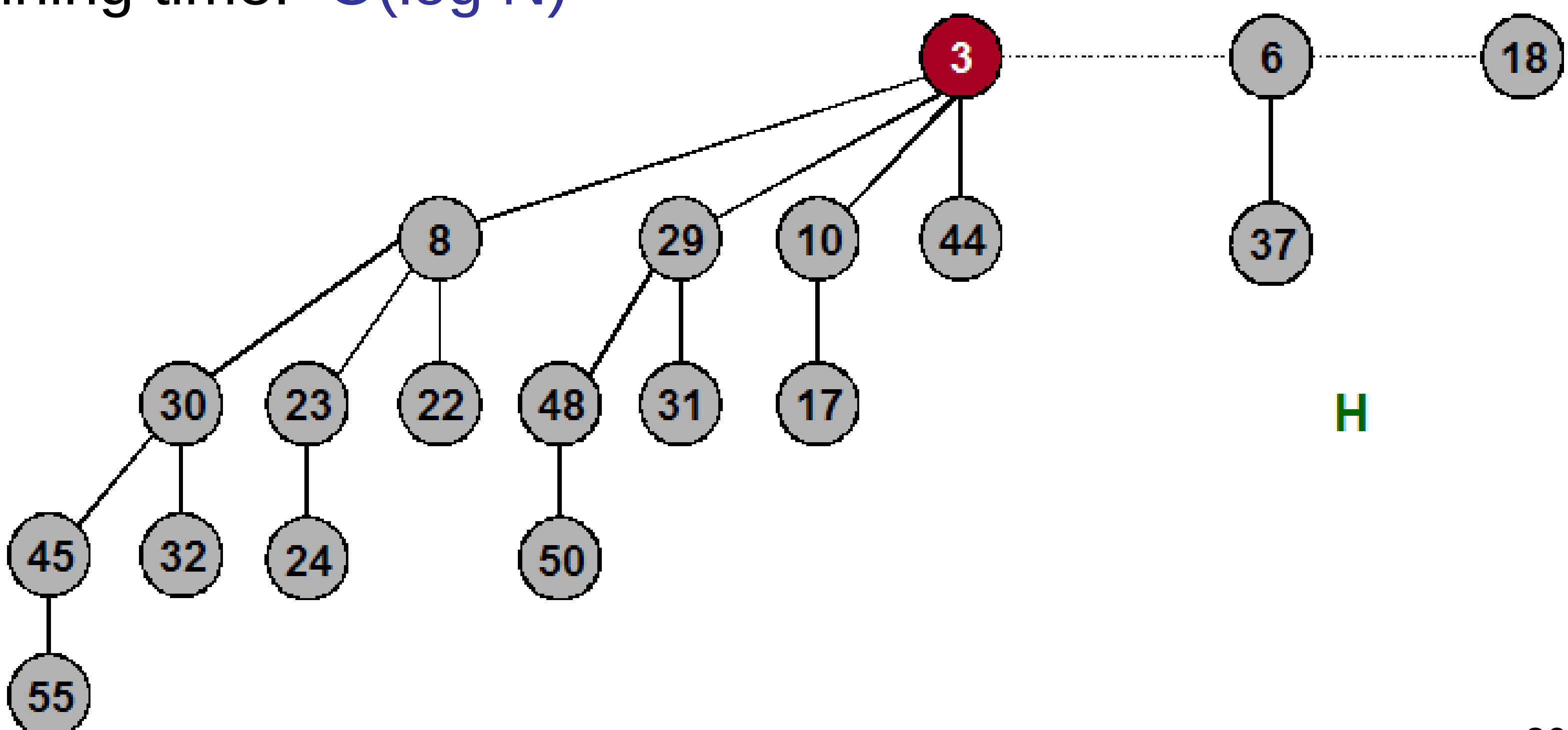
- Running time: O(log N)
  - Proportional to number of trees in root lists $\le 2(\lfloor \log_2 N \rfloor + 1)$

19 + 7 = 26

		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

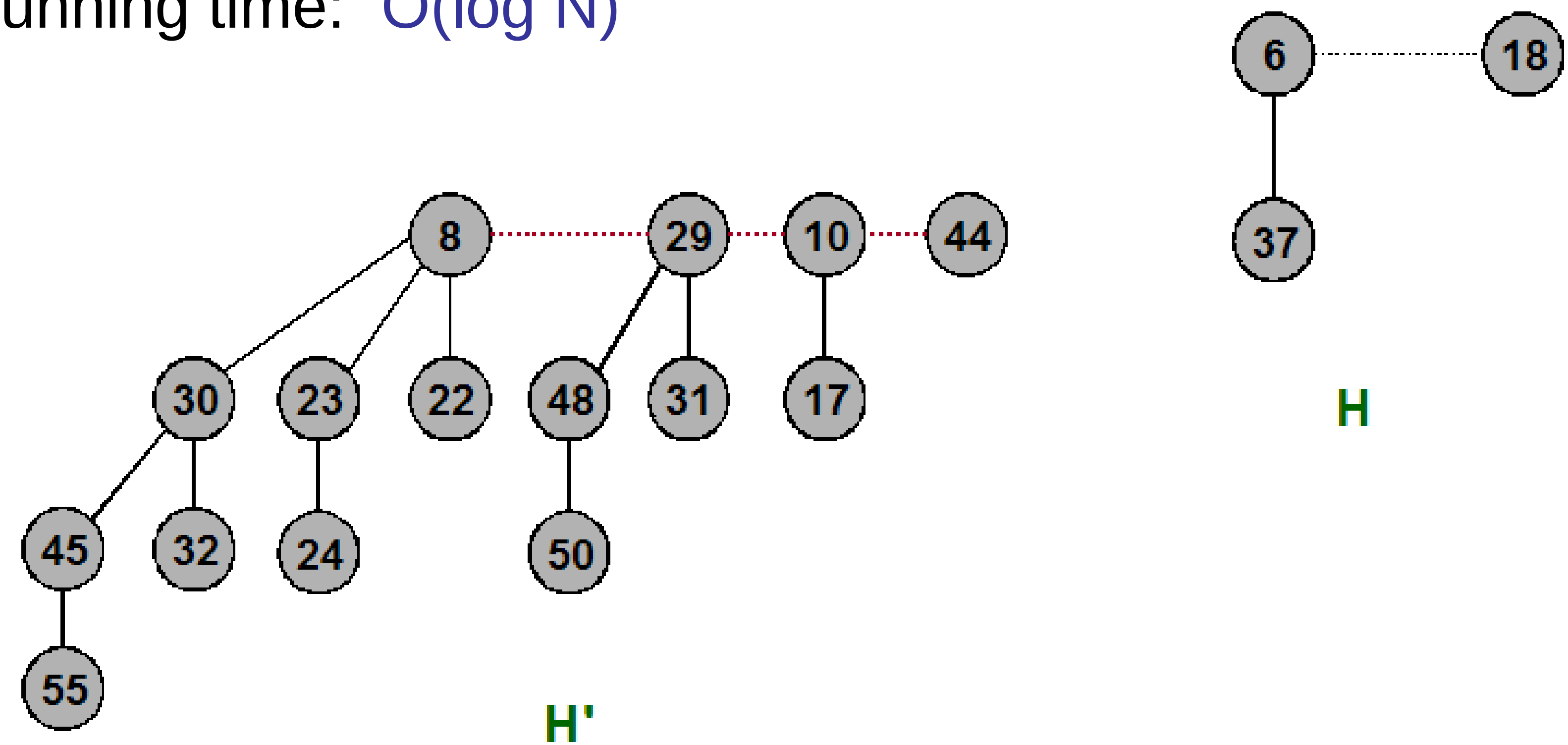
Binomial Heaps: Delete Min

- Delete node with minimum key in binomial heap H
  - Find root x with min key in root list of H, and delete
  - H' ← broken binomial trees
  - H ← Union(H', H)
- Running time: O(log N)



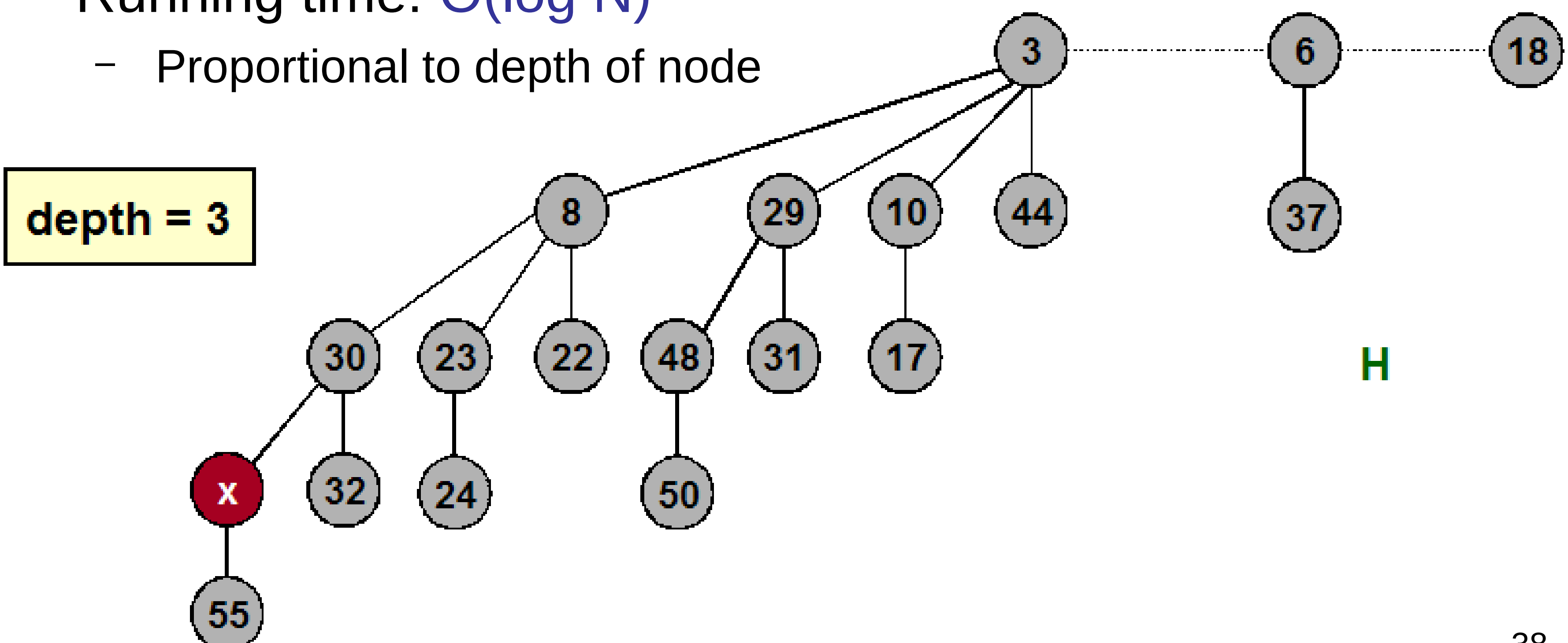
Binomial Heaps: Delete Min

- Delete node with minimum key in binomial heap H
  - Find root x with min key in root list of H, and delete
  - H' ← broken binomial trees
  - H ← Union(H', H)
- Running time: O(log N)



Binomial Heaps: Decrease Key

- Decrease key of node x in binomial heap H
  - Suppose x is in binomial tree B_k
  - Bubble node x up the tree if x is too small
- Running time: O(log N)
  - Proportional to depth of node



Binomial Heaps: Delete

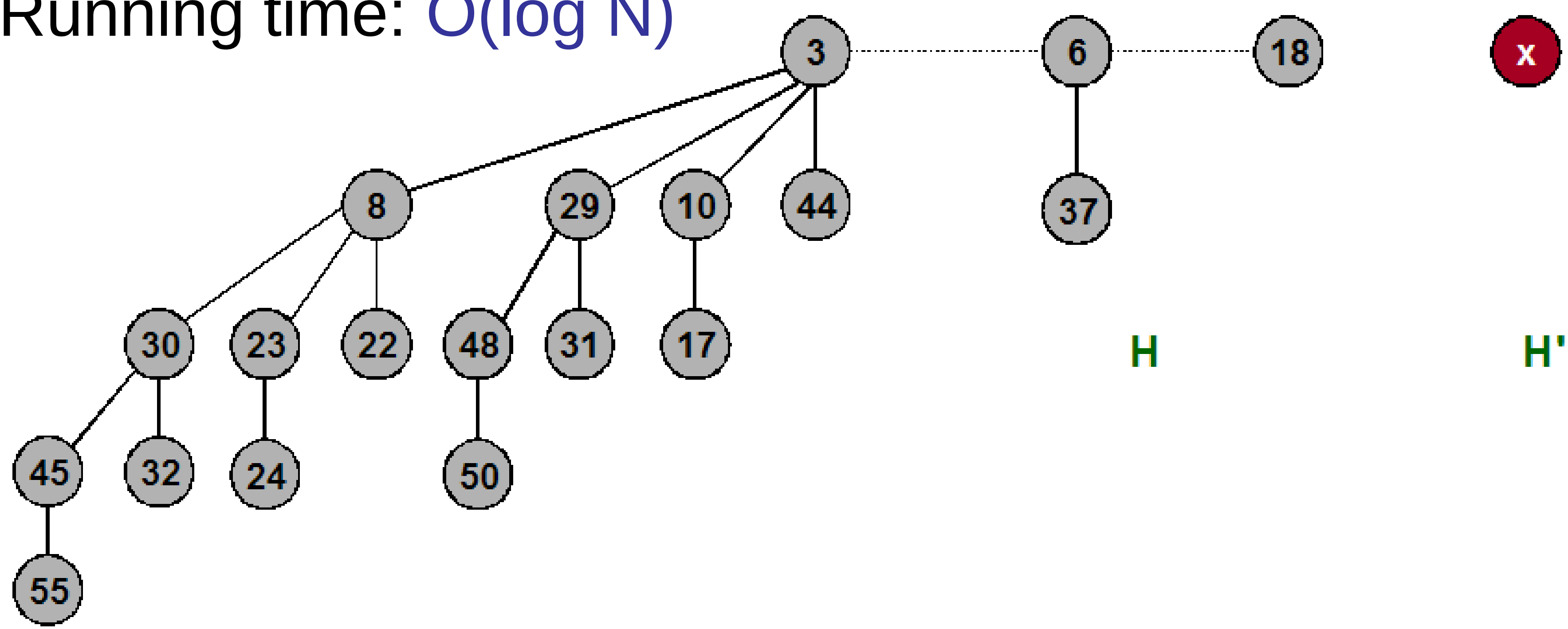
- Delete node x in binomial heap H.
  - Decrease key of x to -∞
  - Delete min

- Running time: O(log N)

Binomial Heaps: Insert

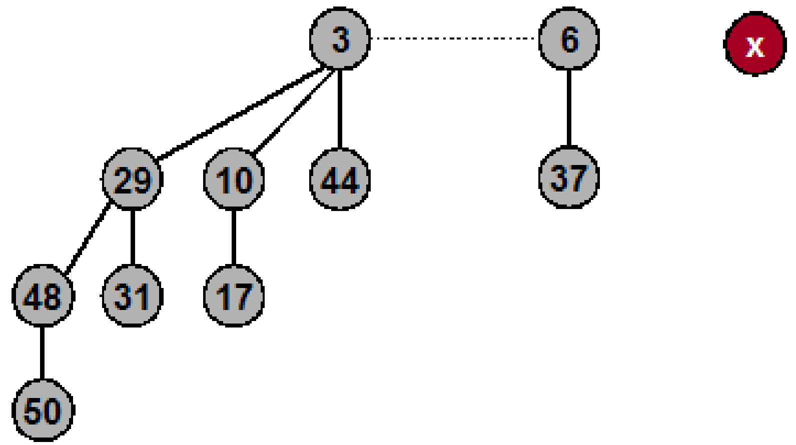
- Insert a new node x into binomial heap H
  - H' ← MakeHeap(x)
  - H ← Union(H', H)

  - Running time: O(log N)



Binomial Heaps: Sequence of Inserts

- Insert a new node x into binomial heap H
  - If N = **`.......0`**, then only 1 steps
  - If N = **`......01`**, then only 2 steps
  - If N = **`.....011`**, then only 3 steps
  - If N = **`....0111`**, then only 4 steps



- Inserting 1 item can take $\Omega(\log_2 \mathrm{N})$ time
  - If N = **`11...111`**, then $\log_2 N$ steps

- But, inserting sequence of N items takes O(N) time!
  - (N/2)(1) + (N/4)(2) + (N/8)(3) + . . . ≤ 2N
  - Amortized analysis
  - Basis for getting most operations down to constant time

$$\sum_{i=1}^{N} \frac{i}{2^i} = 2 - \frac{N}{2^N} - \frac{1}{2^{N-1}}$$
$$\leq 2$$
$$\leq N \sum_{i=0}^{\infty} \frac{1}{2^i} = 2N$$

Binomial Heap Operations

- **MAKE-HEAP()**
- **INSERT(H,x)**
- **MINIMUM(H)**
- **EXTRACT-MIN(H)**
- **UNION(H1,H2)**
- **DECREASE-KEY(H,x,k)**
- **DELETE(H,x)**

- **MAKE-HEAP() [Trivial, just create an object H, head[H]=nil]**
- **MINIMUM(H) [Trivial: Return the minimum among binomial tree roots in the binomial heap]**

Priority Queues

		Heaps			
Operation	Linked List	Binary	Binomial	Fibonacci *	Relaxed
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

just did this

Summary

- Binomial Trees and Binomial Heaps
- Operations on Binomial Heaps
  - Creating New Binomial Heap
  - Finding Minimum Key
  - Uniting Two Binomial Heaps
  - Inserting Node
  - Extracting Node with Minimum Key
  - Deleting Key
  - Decreasing Key