



COMP 482 / ELEC 420

Heaps

Reading

- Skim [CLRS] 6 – Review binary heaps.
- Read [CLRS] 19,20.

Summary of Results

	Binary heaps (worst case)	Binomial Heaps (worst case)	Fibonacci Heaps (amortized)
Insert	$\log n$	$\log n$	1
Minimum	1	$\log n$	1
Delete-Min	$\log n$	$\log n$	$\log n$
Union	n	$\log n$	1
Delete	$\log n$	$\log n$	$\log n$
Decrease- Key	$\log n$	$\log n$	1

Alternatively, Maximum.

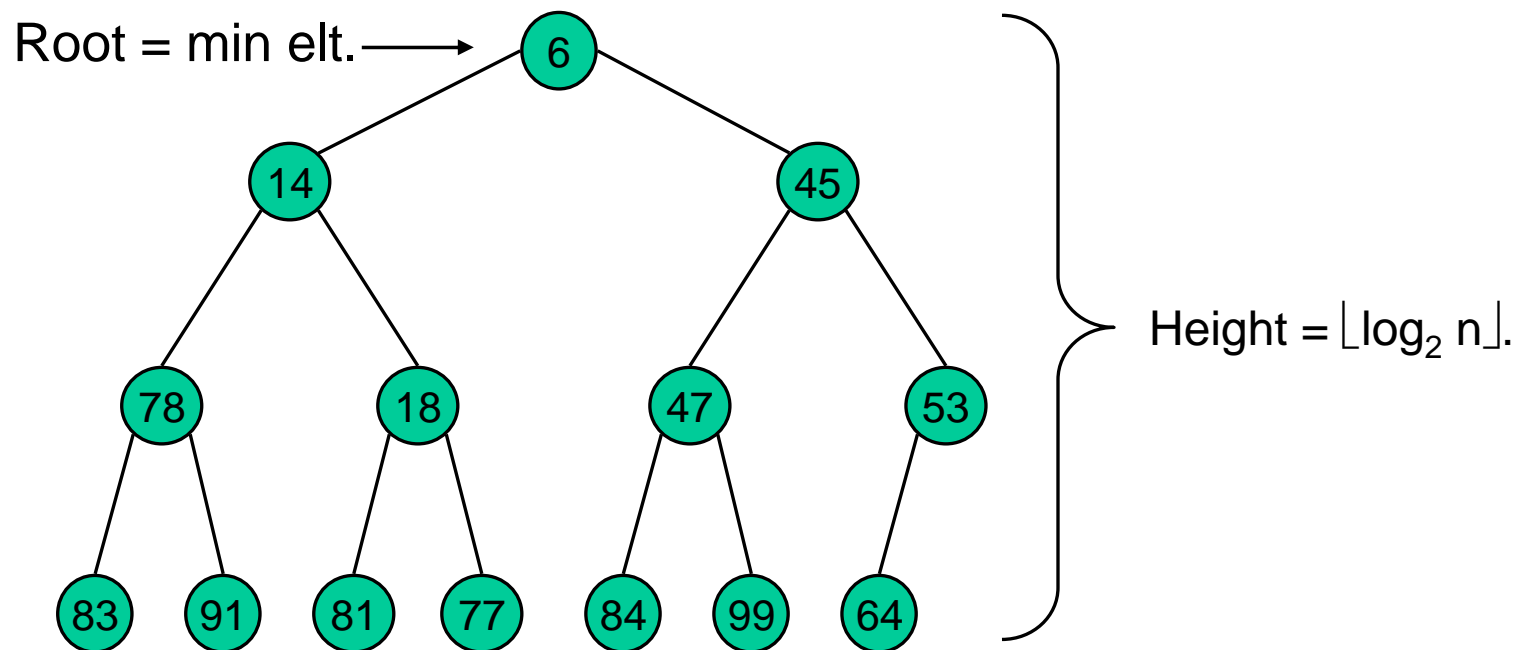
Binary Heap: Review

Almost complete binary tree

- All levels filled except last, where filled left to right.

Min-heap ordered

- Children greater than (or equal to) parents.

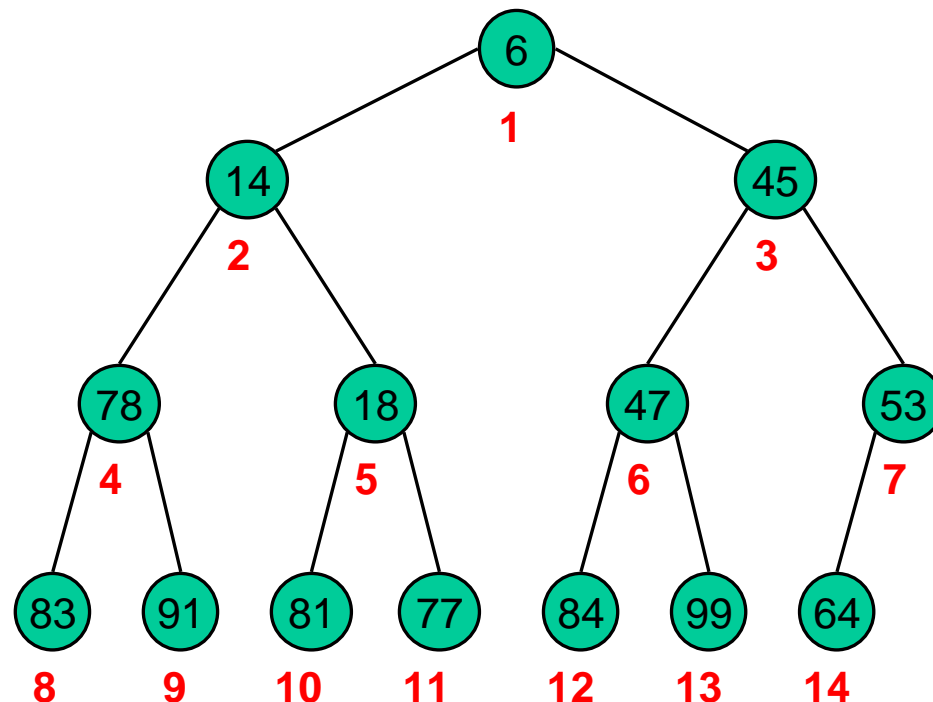


Binary Heaps: Implementation

Use array: no need for explicit pointers.

$$\text{Parent}(i) = \lfloor i/2 \rfloor \quad \text{Left}(i) = 2i \quad \text{Right}(i) = 2i + 1$$

Irrelevant for current discussion – concentrate on tree structure.

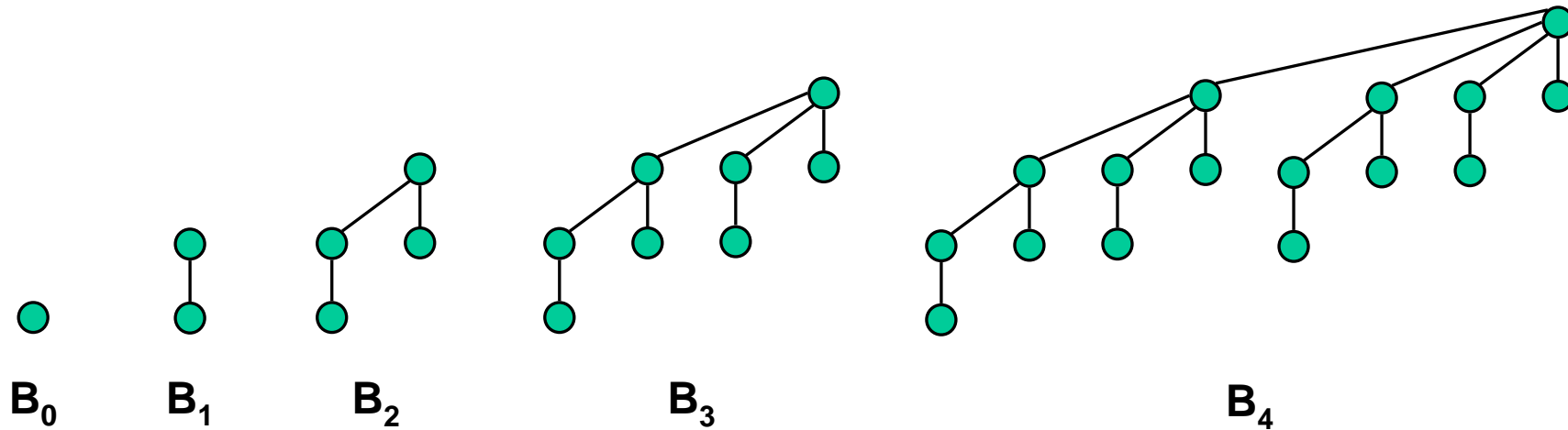
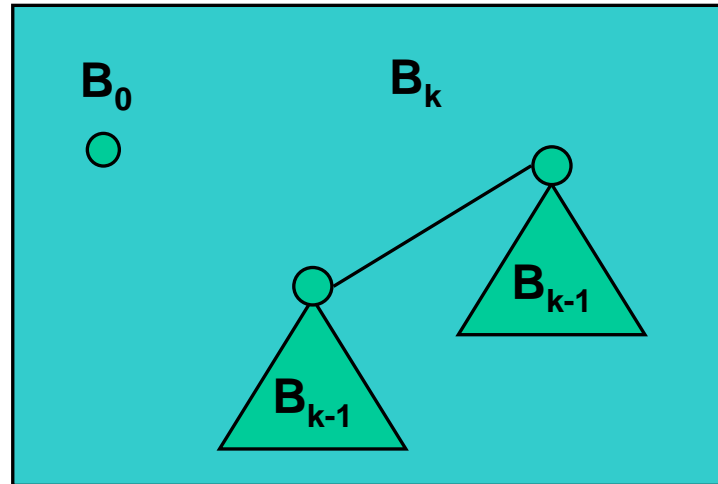


Binomial Heaps: Overview

A list of *binomial trees*.

Each has some of the data.
Each has the heap property.

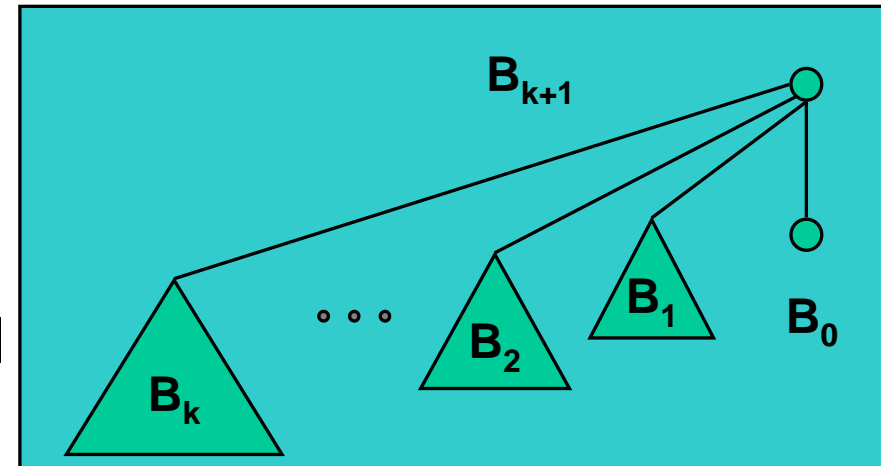
Binomial Tree



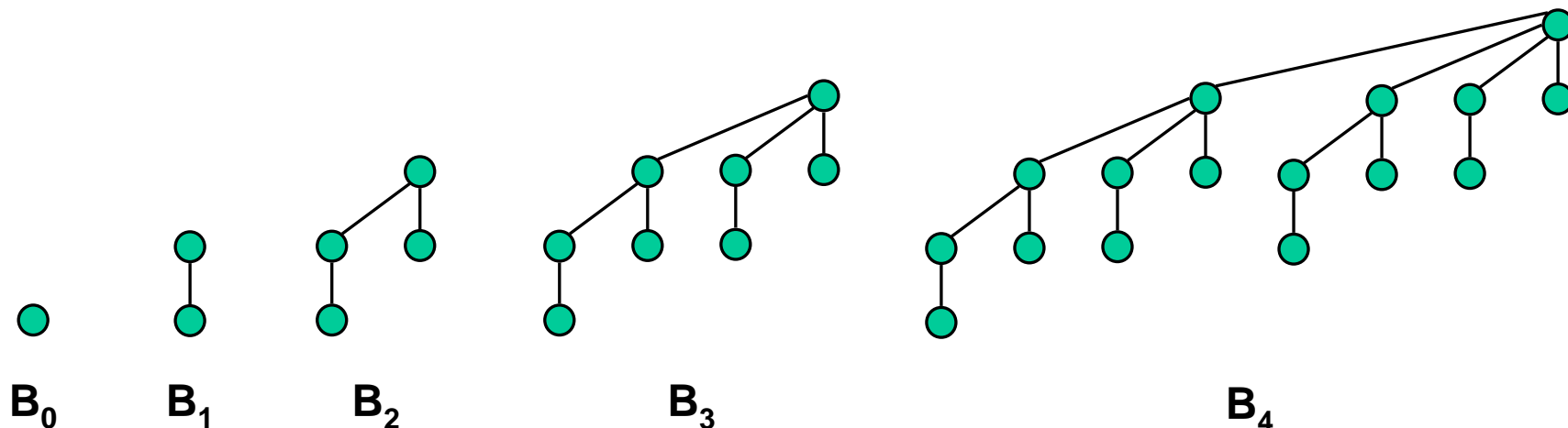
Binomial Tree: Properties 1

Binomial tree B_k :

- #nodes = 2^k .
- Height = k .
- Degree of root = k .
- Deleting root yields binomial trees B_{k-1}, \dots, B_0 .



Can prove by induction on k .



Binomial Tree: Properties 2

B_k has $\binom{k}{i}$ nodes at depth i .

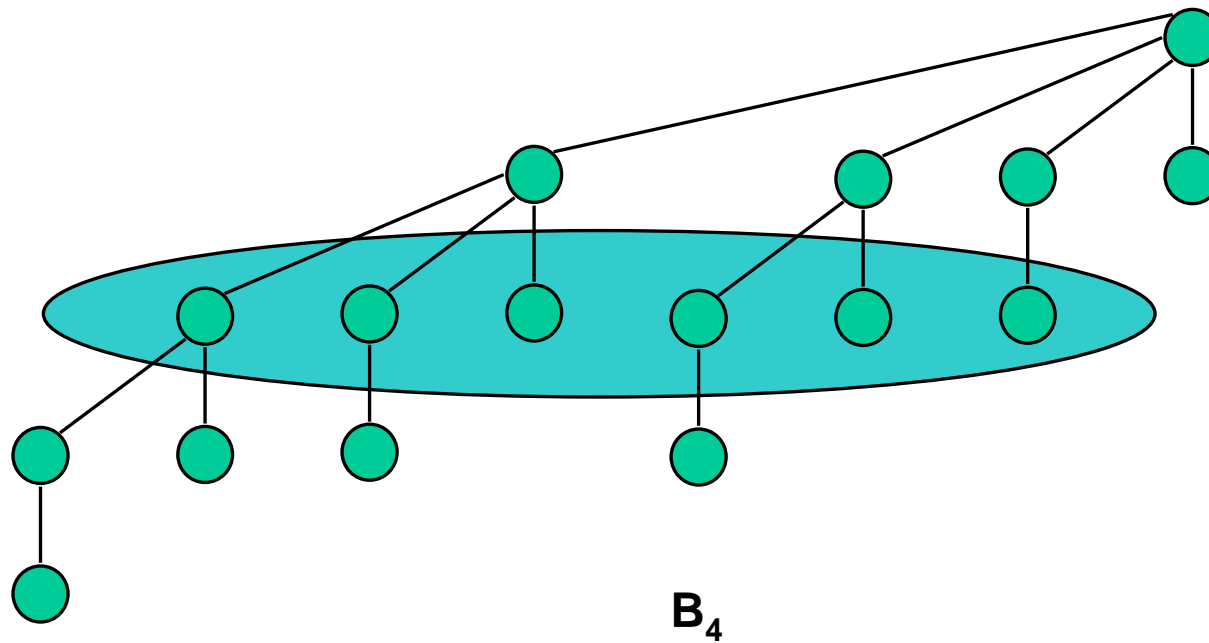
depth 0

depth 1

depth 2

depth 3

depth 4



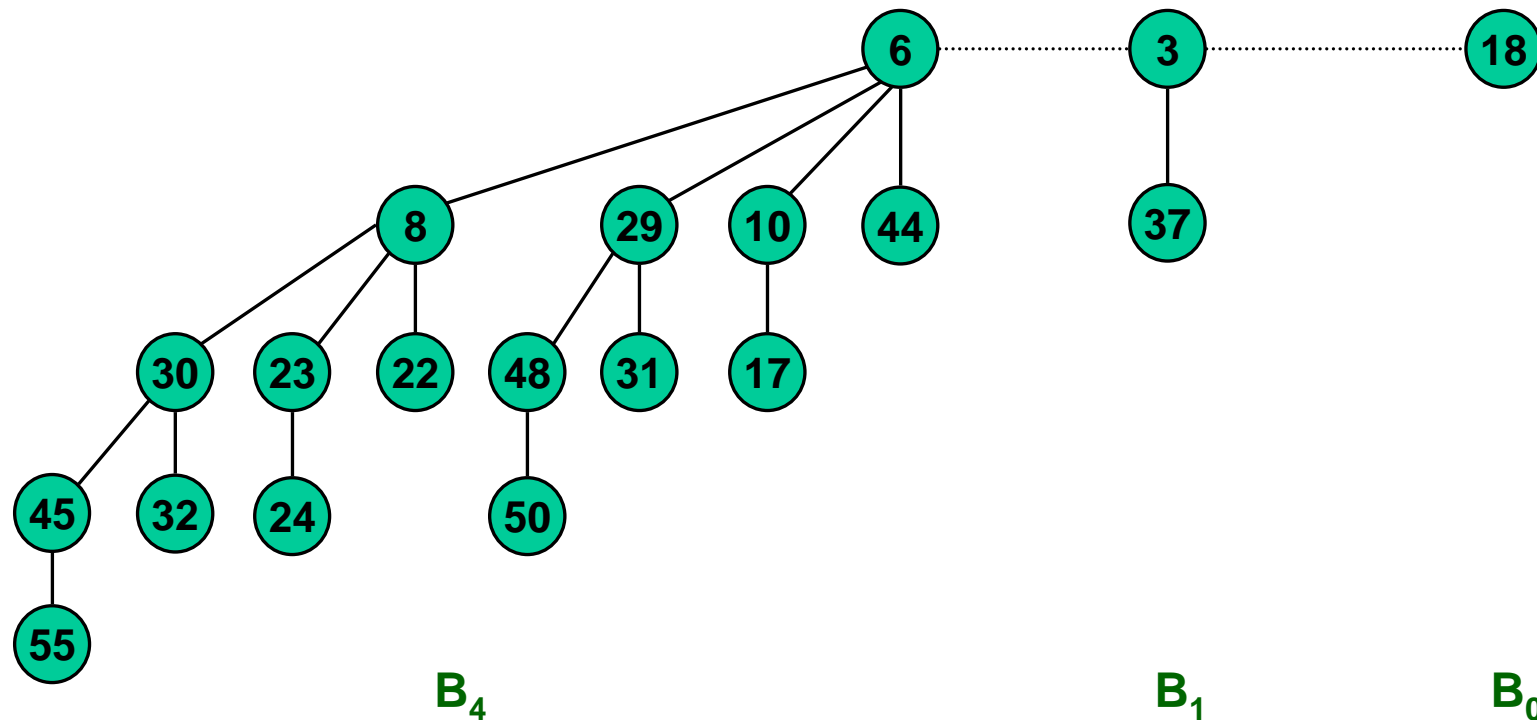
$$\binom{4}{2} = 6$$

Binomial Heap

Sequence of binomial trees satisfying binomial heap property:

- Each tree is min-heap ordered.
- 0 or 1 binomial tree of order k

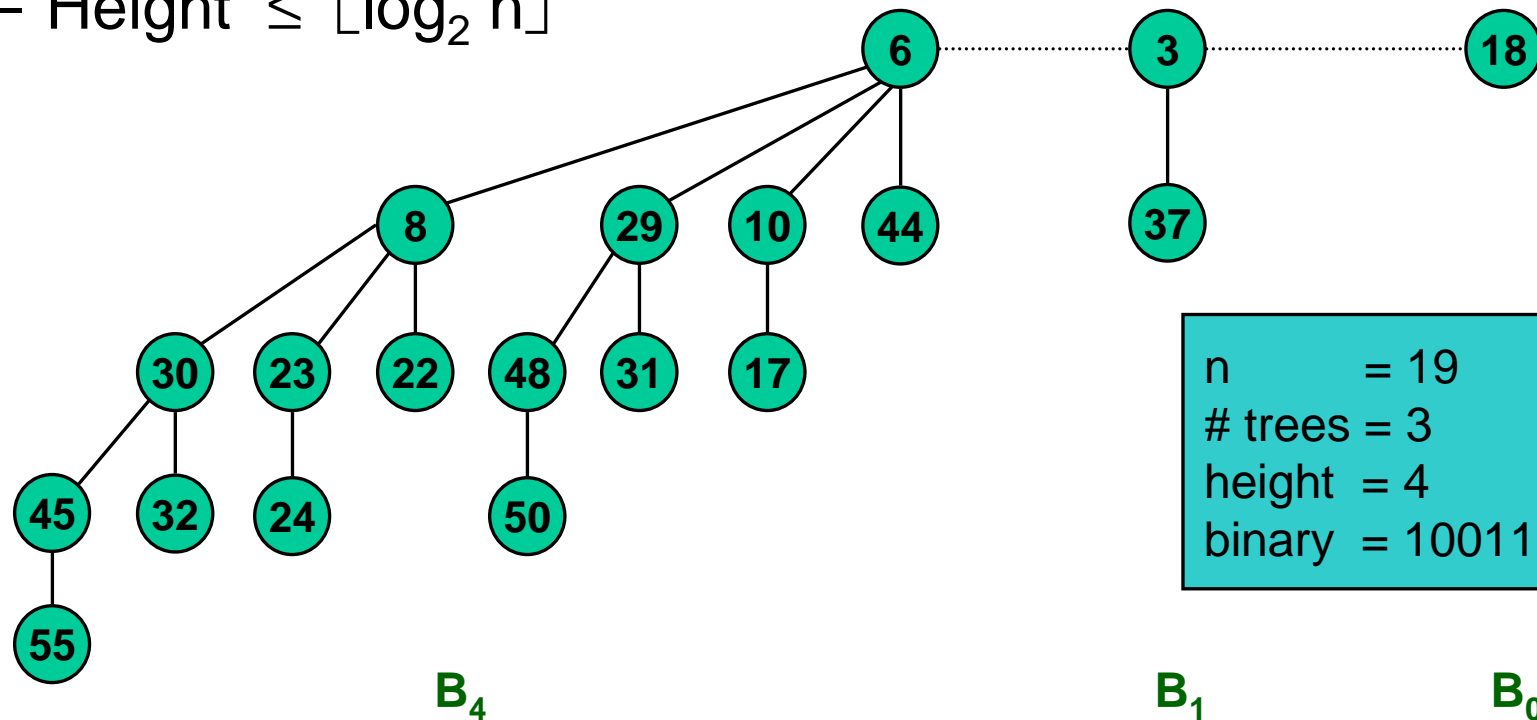
Vuillemin (1978)



Binomial Heap: Properties

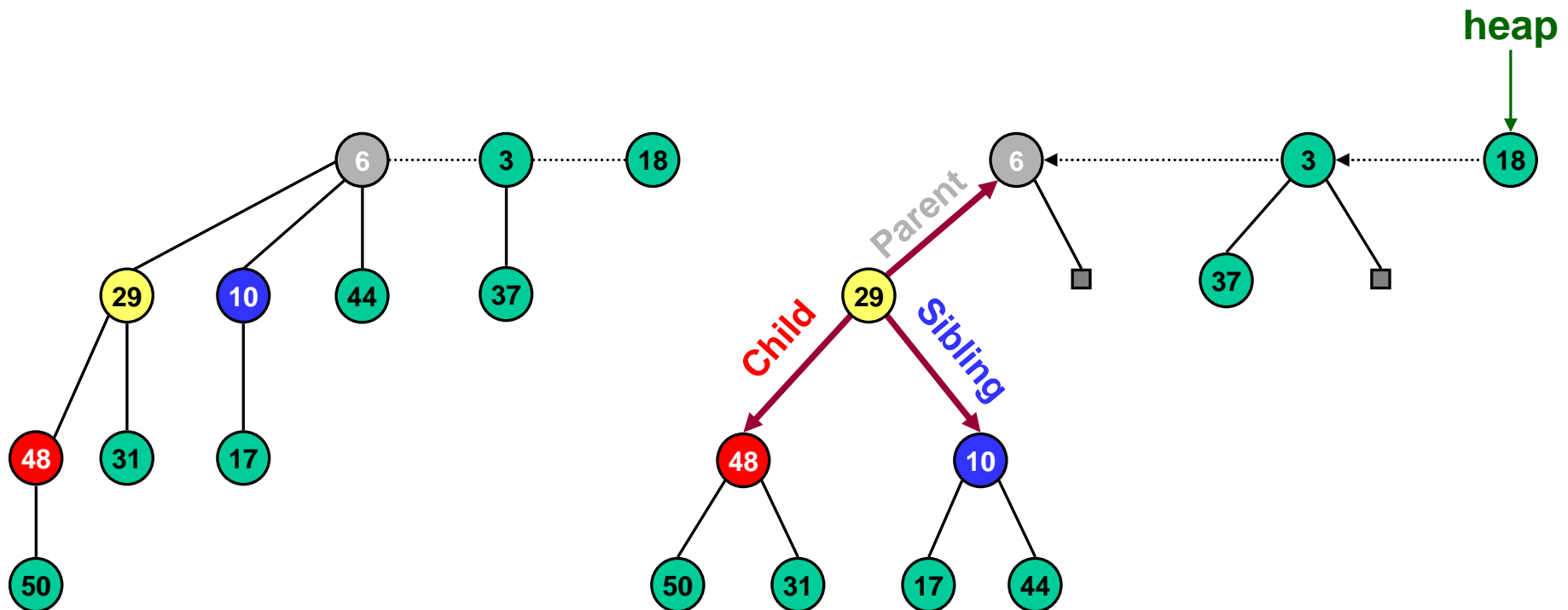
Binomial heap of n nodes:

- Min key = root of B_0, B_1, \dots , or B_k .
- Contains binomial tree B_i iff $b_i = 1$.
 - Where $b_n \dots b_2 b_1 b_0$ is binary representation of n .
- #trees $\leq \lfloor \log_2 n \rfloor + 1$
- Height $\leq \lfloor \log_2 n \rfloor$



Binomial Heap: Implementation

Each node has 3 pointers – parent, 1st child, next sibling.
Roots in singly-linked list – from smallest tree to largest.



Binomial Heap

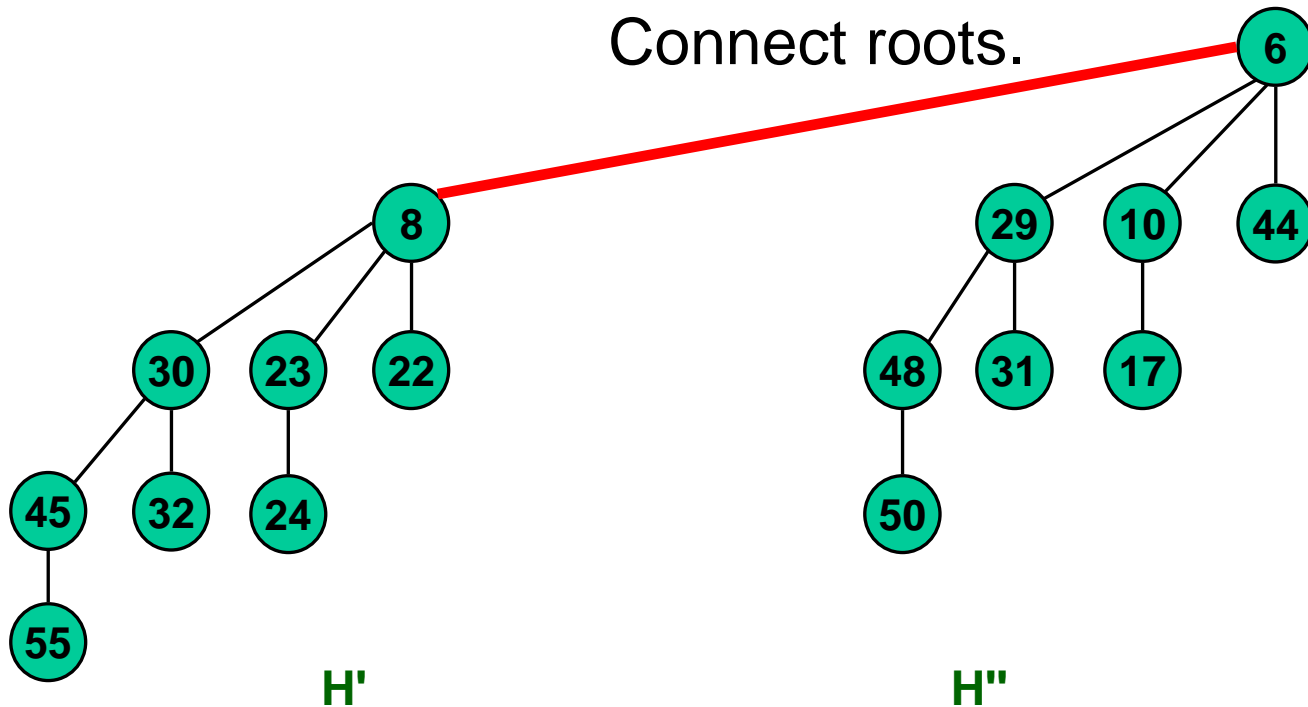
Implementation: Leftist Power-of-2 Heap

Binomial Heap: Union – Special Case

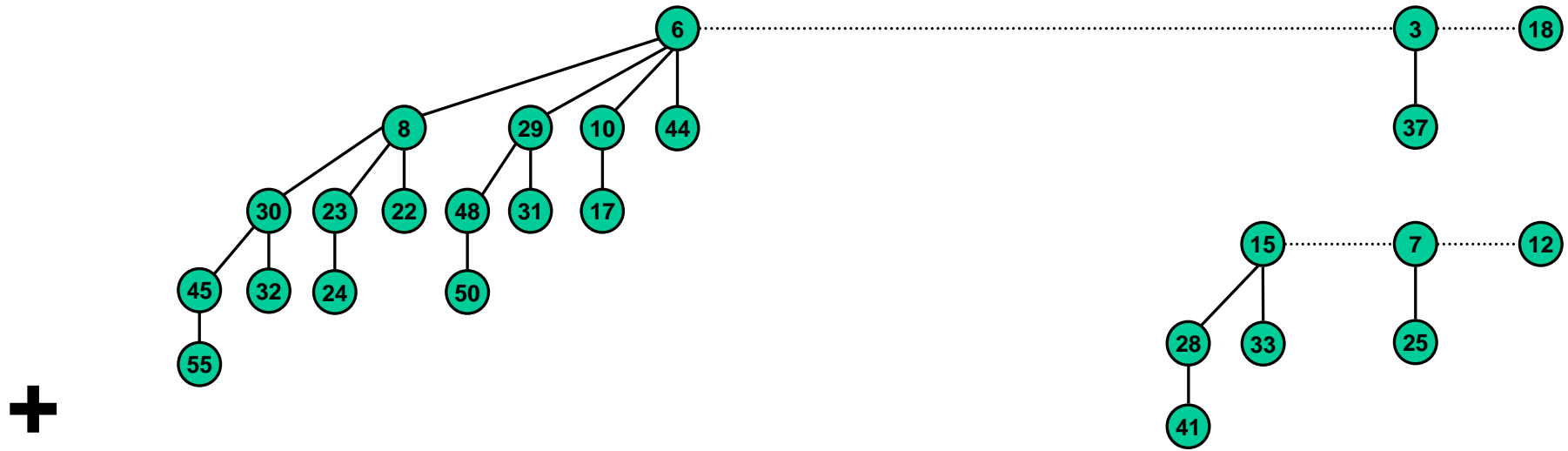
Easy if each are order k binomial trees.

Choose smaller
key as new root.

Connect roots.



Binomial Heap: Union – General Case



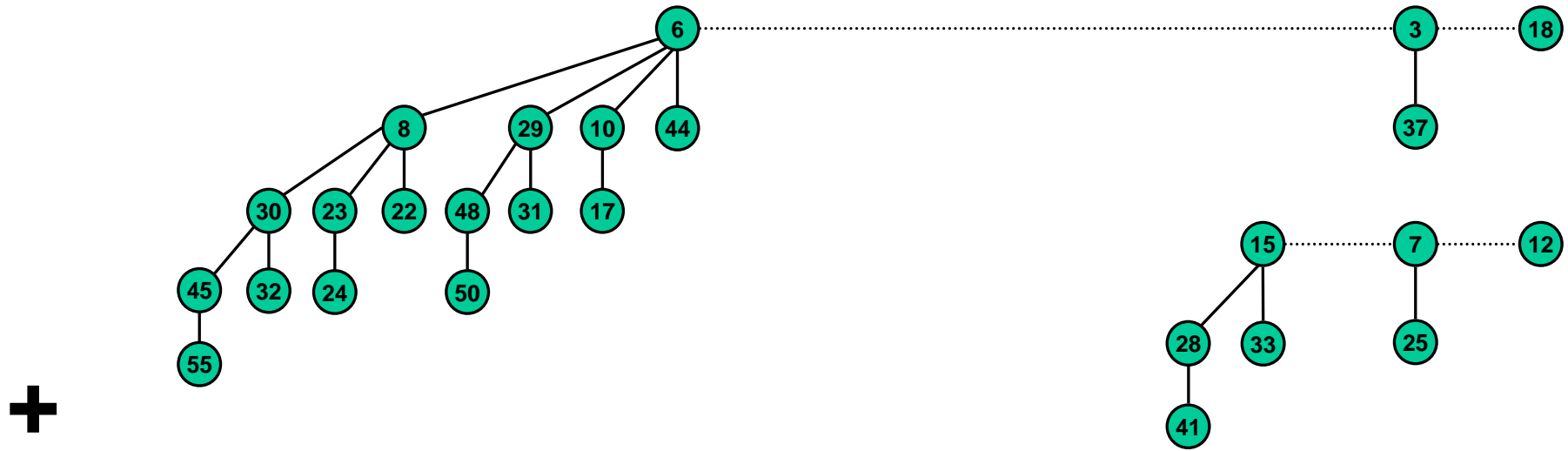
+

Analogy:

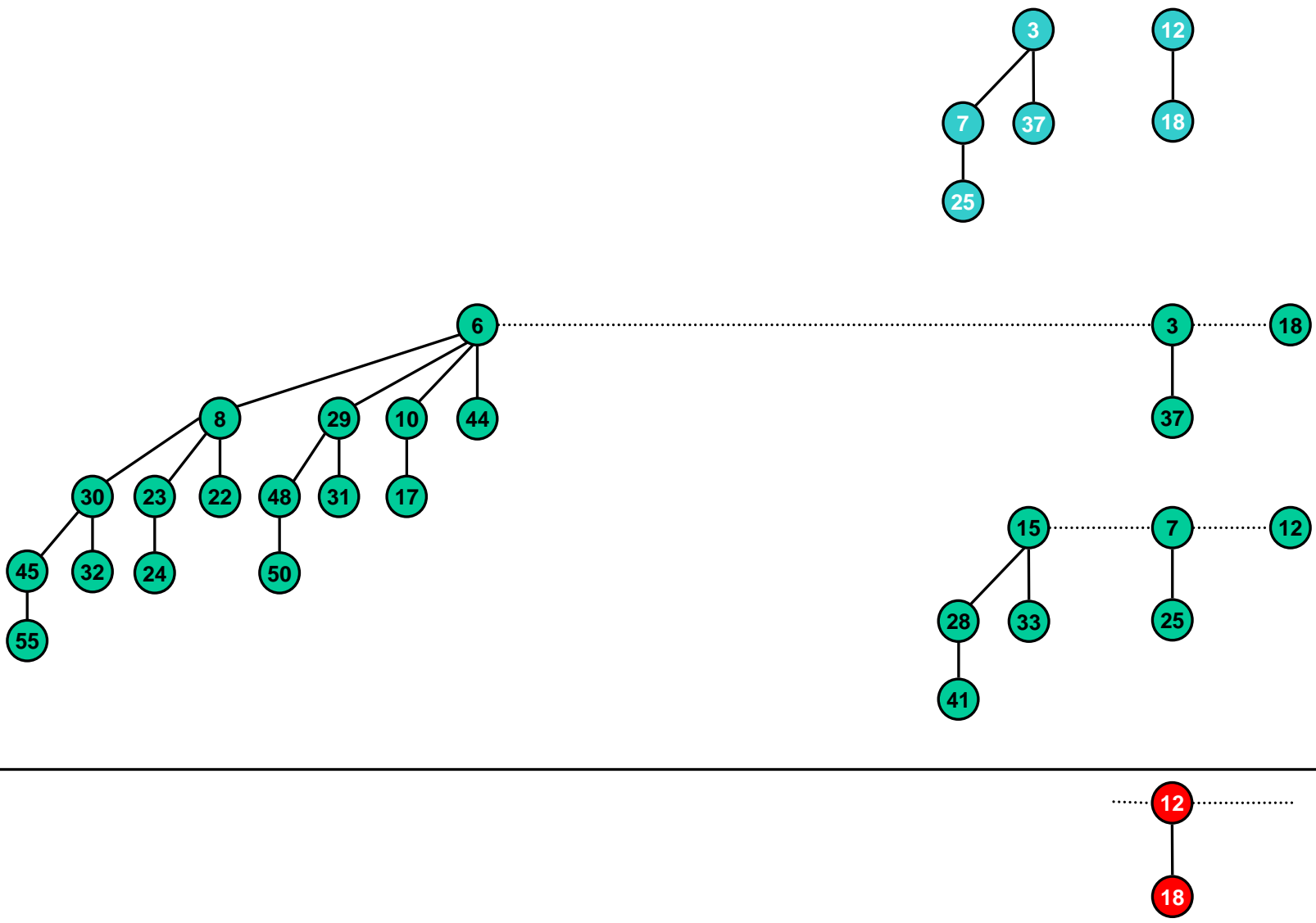
19 + 7 = 26

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

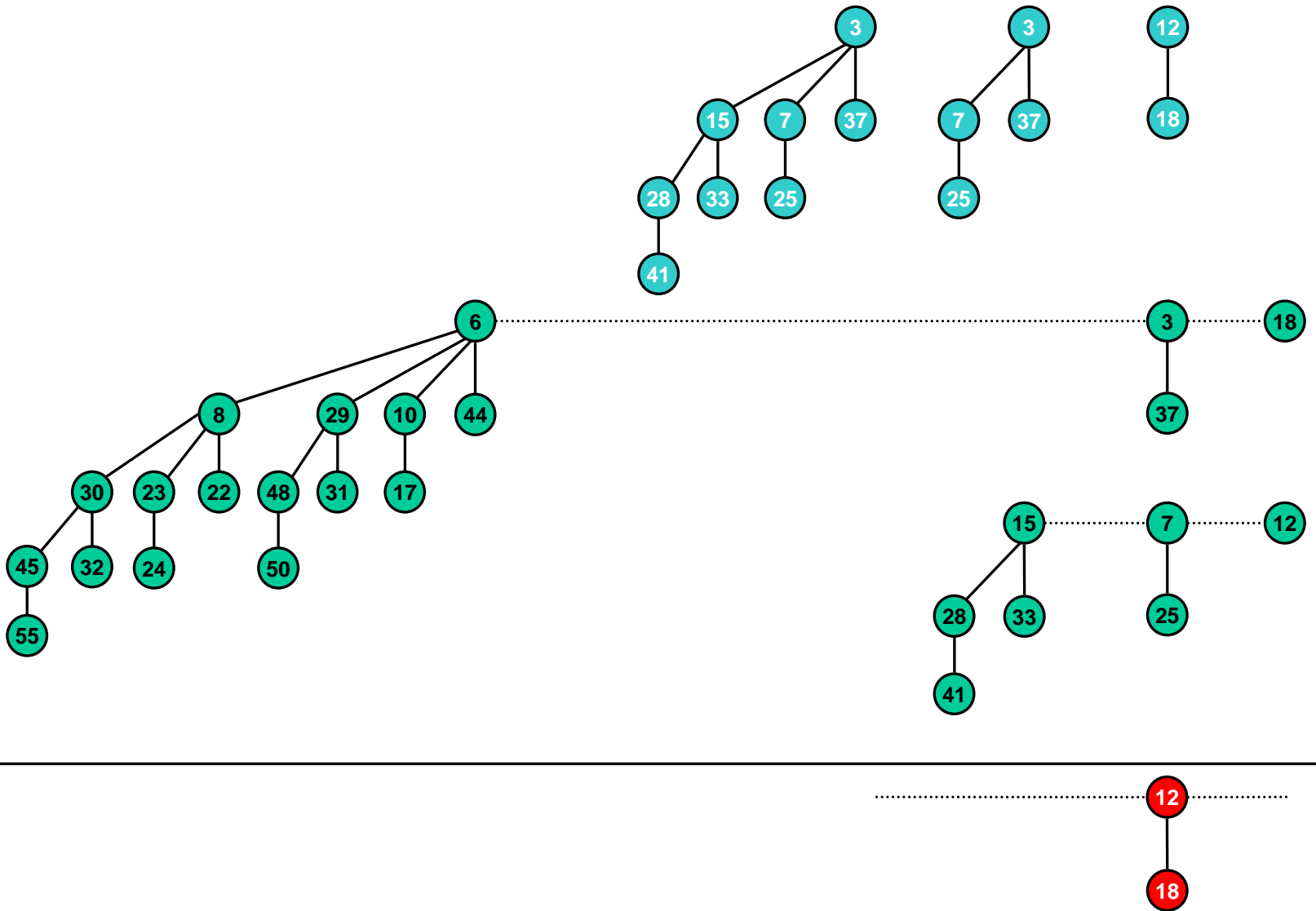
Binomial Heap: Union – Example



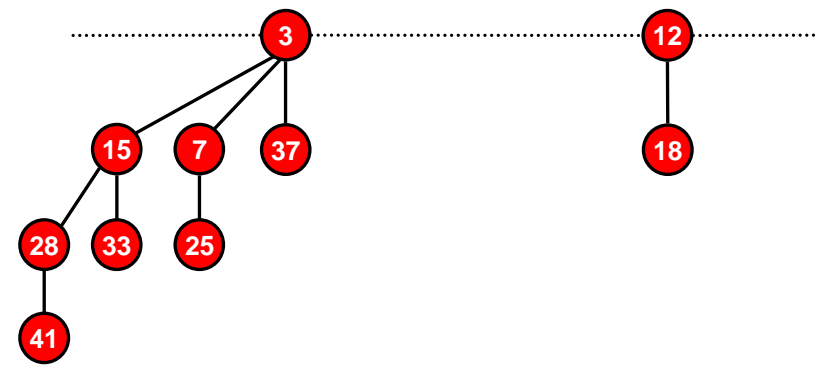
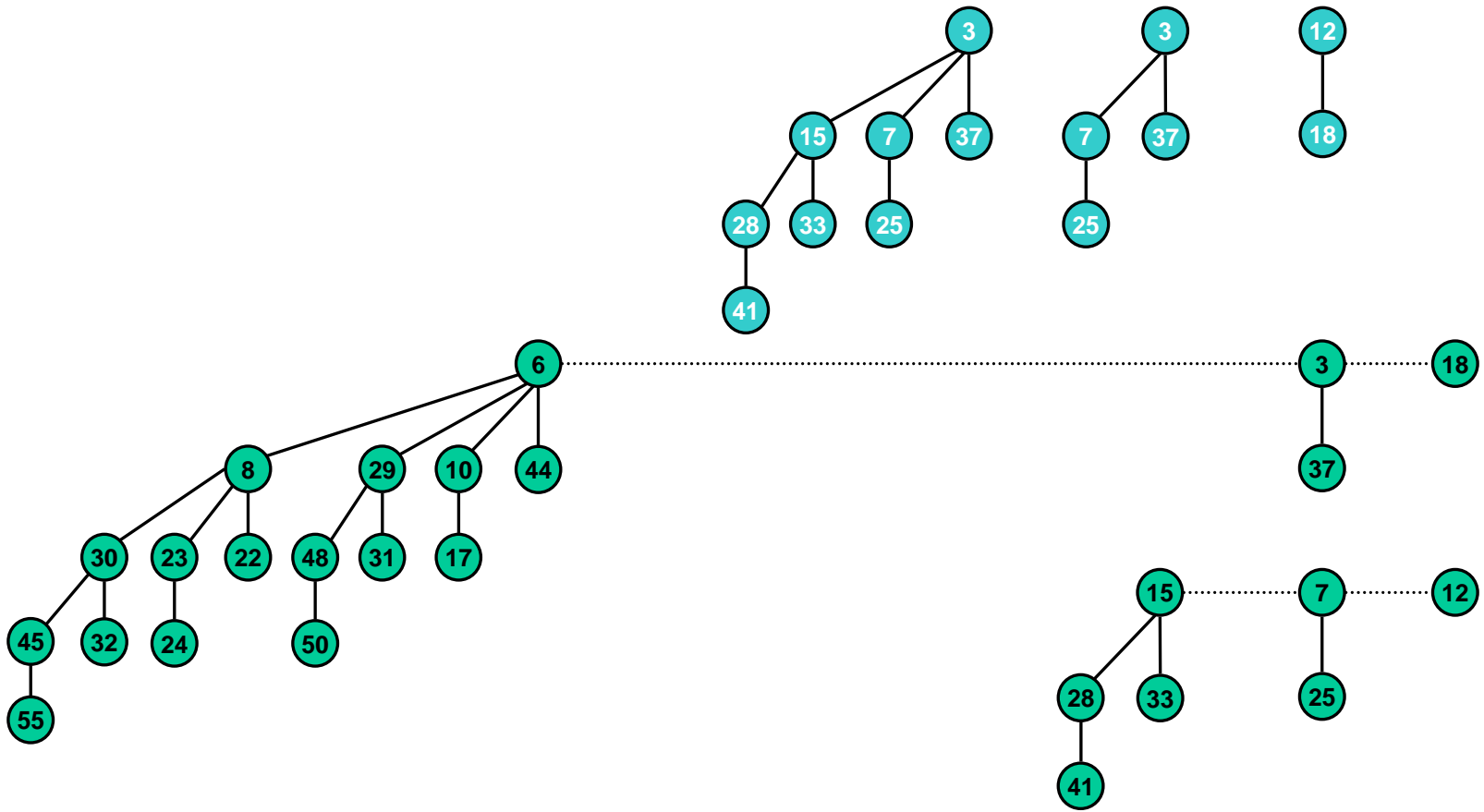
+



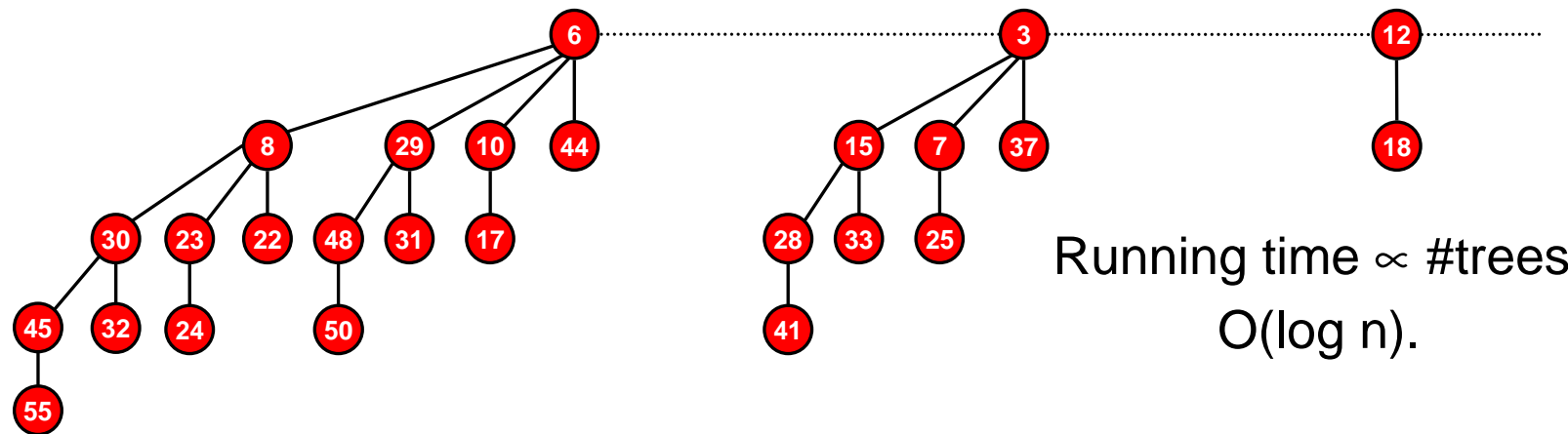
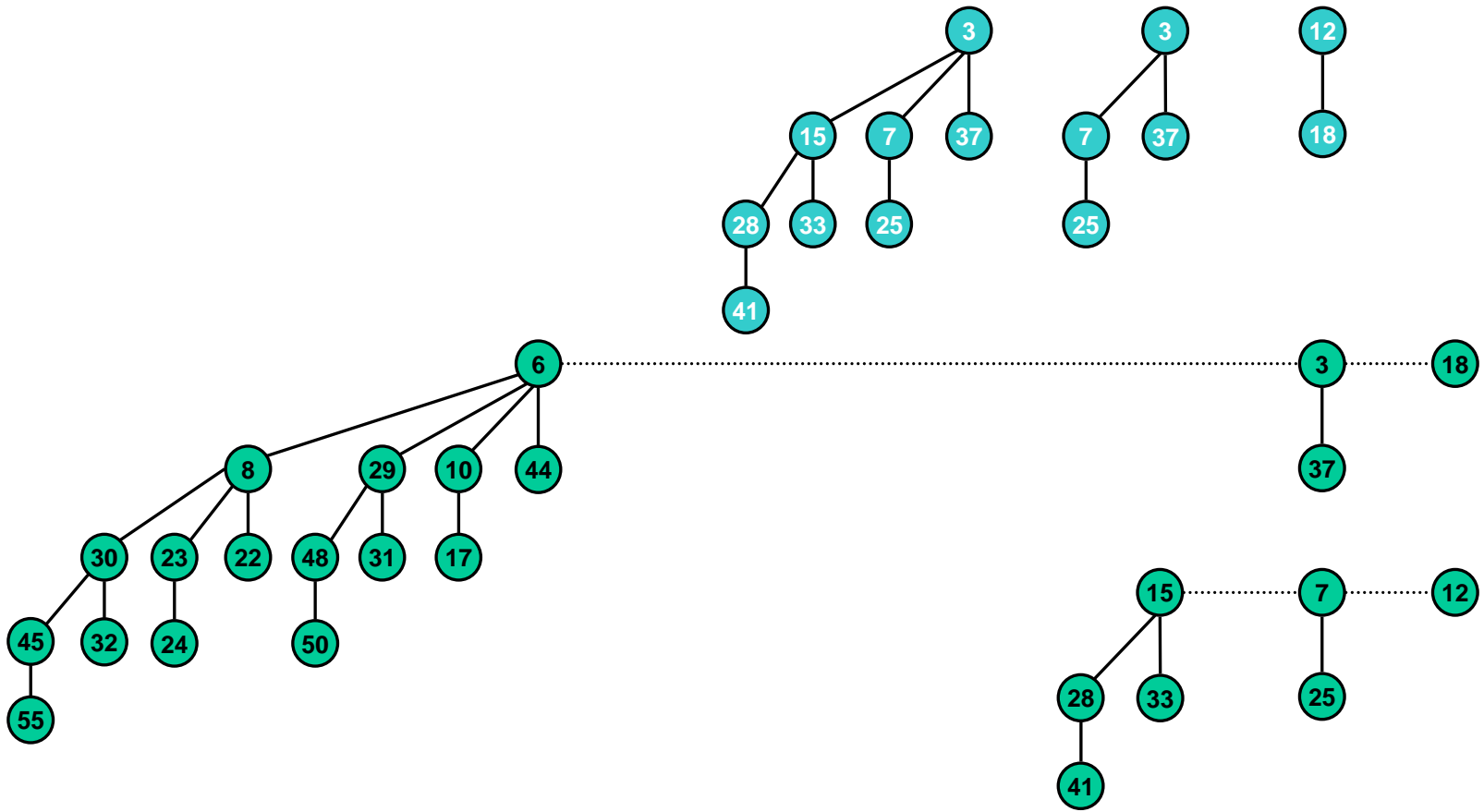
+



+



+

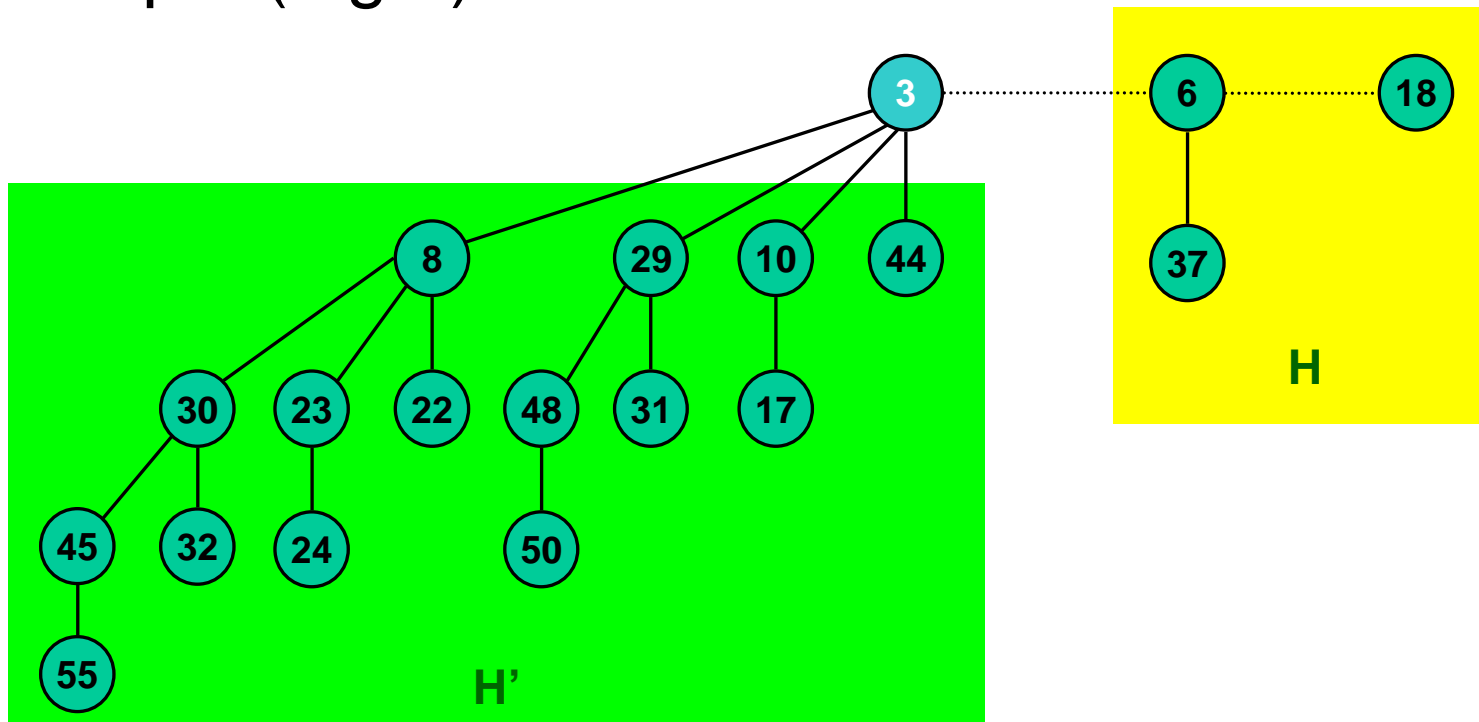


Running time \propto #trees:
 $O(\log n)$.

Binomial Heap: Delete Min

1. Find root min key of H , and delete.
2. $H' \leftarrow$ Children of deleted key.
3. $H \leftarrow \text{Union}(H', H)$

Each step $O(\log n)$.

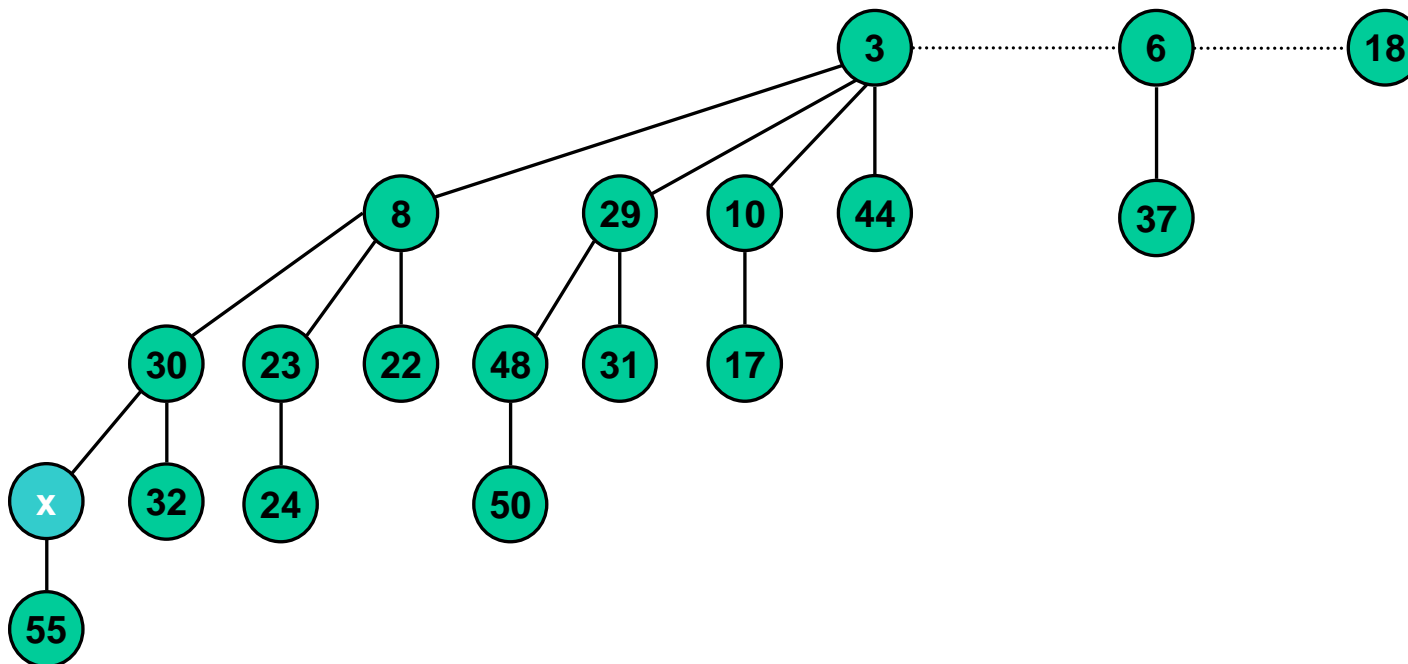


Binomial Heap: Decrease Key

Assume given pointer to node – don't have to find it.

1. Change key.
2. While $x < \text{parent}$, swap – bubbles node up.

$O(\log N)$ – ∞ depth of node $x \leq \lfloor \log_2 n \rfloor$.



Binomial Heap: Delete

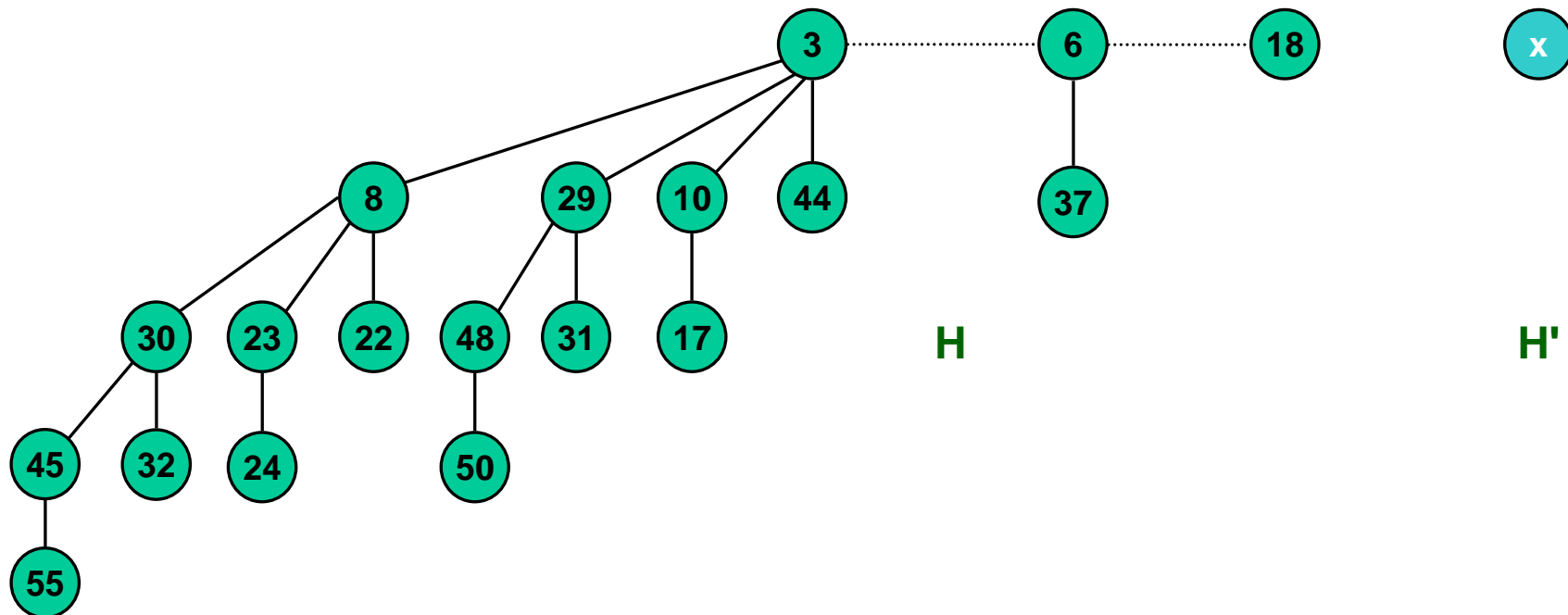
1. Decrease key of x to $-\infty$.
2. Delete min.

Each step $O(\log N)$.

Binomial Heap: Insert

1. $H' \leftarrow \text{MakeHeap}(x)$
2. $H \leftarrow \text{Union}(H', H)$

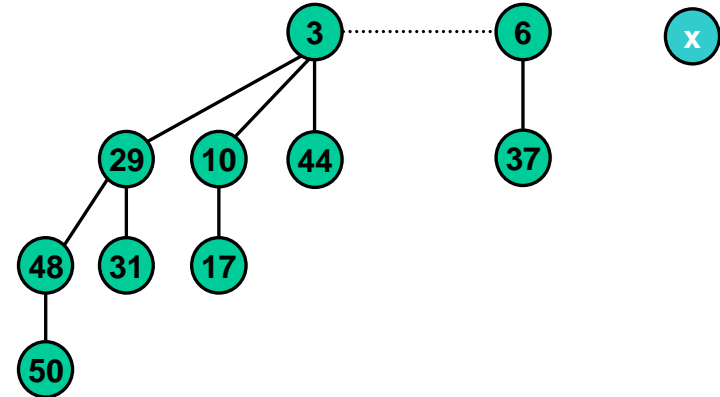
$O(\log N)$



Binomial Heap: Sequence of Inserts

Inserting 1 item can take $\Omega(\log n)$ time.

- $n = \dots\dots\dots 0 \rightarrow 1$ step.
- $n = \dots\dots\dots 01 \rightarrow 2$ steps.
- $n = \dots\dots\dots 011 \rightarrow 3$ steps.
- $n = \dots\dots\dots 0111 \rightarrow 4$ steps.
- $n = 11\dots\dots 111 \rightarrow \log_2 n$ steps.



Inserting sequence of n items takes $O(n)$ time!

$$- (n/2)(1) + (n/4)(2) + (n/8)(3) + \dots \leq 2n$$

$$\sum_{i=1}^n \frac{i}{2^i} = 2 - \frac{n}{2^n} - \frac{1}{2^{n-1}} \leq 2$$

Summary of Results, Again

	Binary heaps (worst case)	Binomial Heaps (worst case)	Fibonacci Heaps (amortized)
Insert	$\log n$	$\log n$	1
Minimum	1	$\log n$	1
Delete-Min	$\log n$	$\log n$	$\log n$
Union	n	$\log n$	1
Delete	$\log n$	$\log n$	$\log n$
Decrease- Key	$\log n$	$\log n$	1

Fibonacci Heaps

Intuition

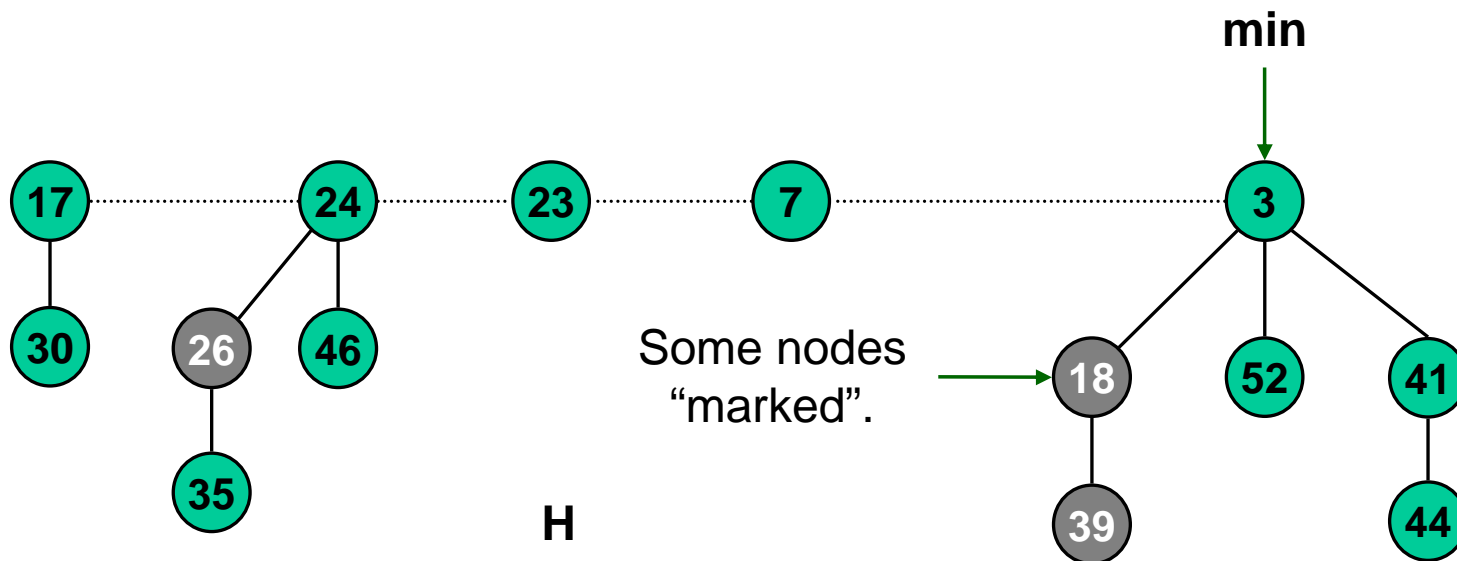
- Similar to binomial heaps, but less structured.
- “Lazy” unions.

Original motivation

- $O(m + n \log n)$ shortest path algorithm.
- Also led to faster algorithms for MST, weighted bipartite matching.
- Fredman & Tarjan (1986)

Fibonacci Heaps: Structure

Set of min-heap ordered trees.



Fibonacci Heaps: Implementation

Each node has 4 pointers: parent, 1st child, next & previous siblings.

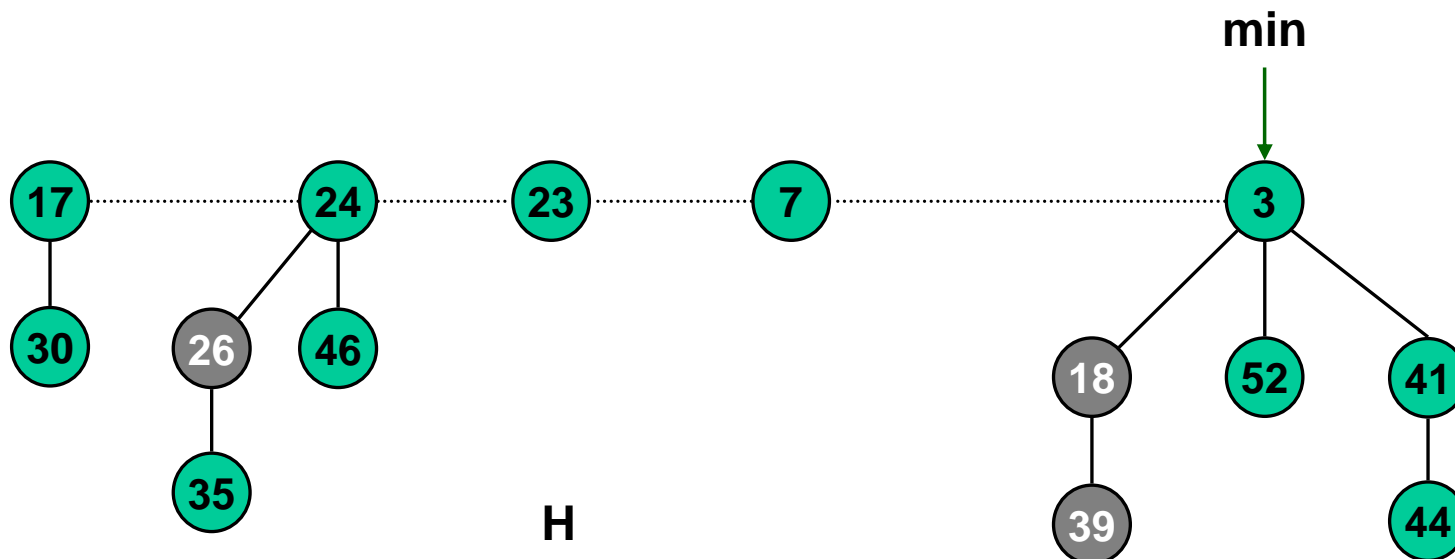
- Sibling pointers form circular, doubly-linked list.
- Can quickly splice off subtrees.

Roots in circular, doubly-linked list.

- Fast union.

Have pointer to min element (a root).

- Fast find-min.

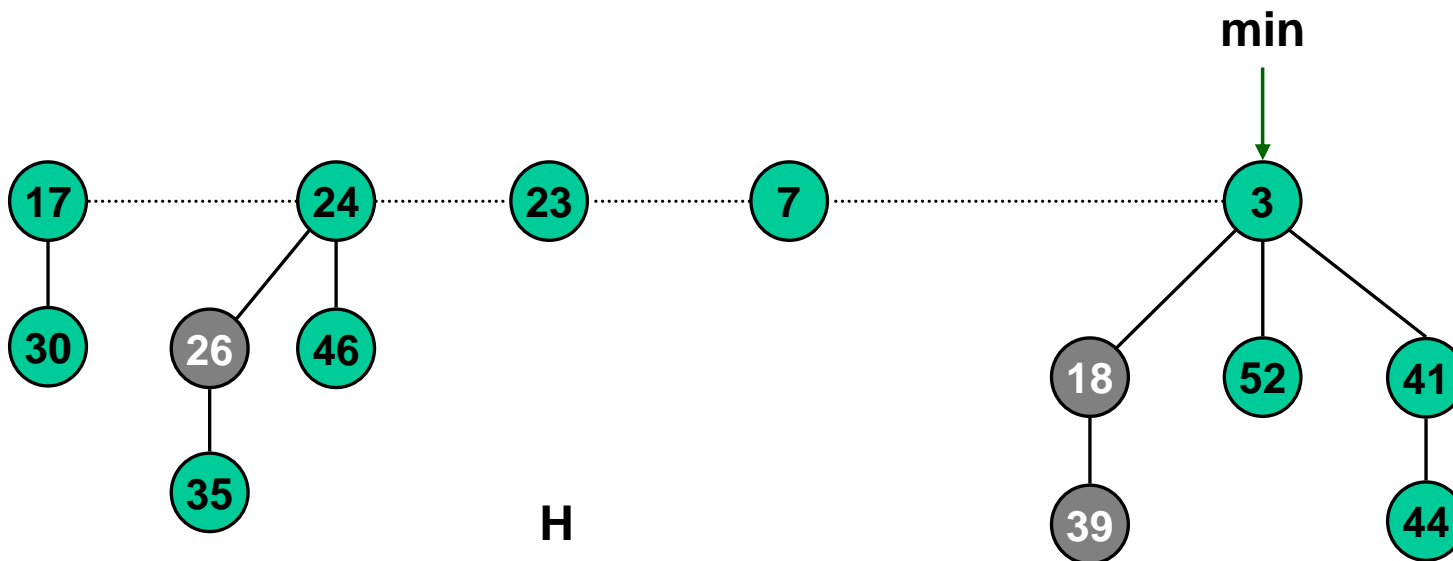


Fibonacci Heaps: Potential Function

Key quantities:

- $\text{degree}(x)$ = degree of node x .
- $\text{mark}(x)$ = is node x marked?
- $t(H)$ = # trees.
- $m(H)$ = # marked nodes.
- $\Phi(H)$ = $t(H) + 2m(H)$

$$\begin{aligned} t(H) &= ? & 5 \\ m(H) &= ? & 3 \\ \Phi(H) &= ? & 11 \end{aligned}$$



Fibonacci Heaps: Insert

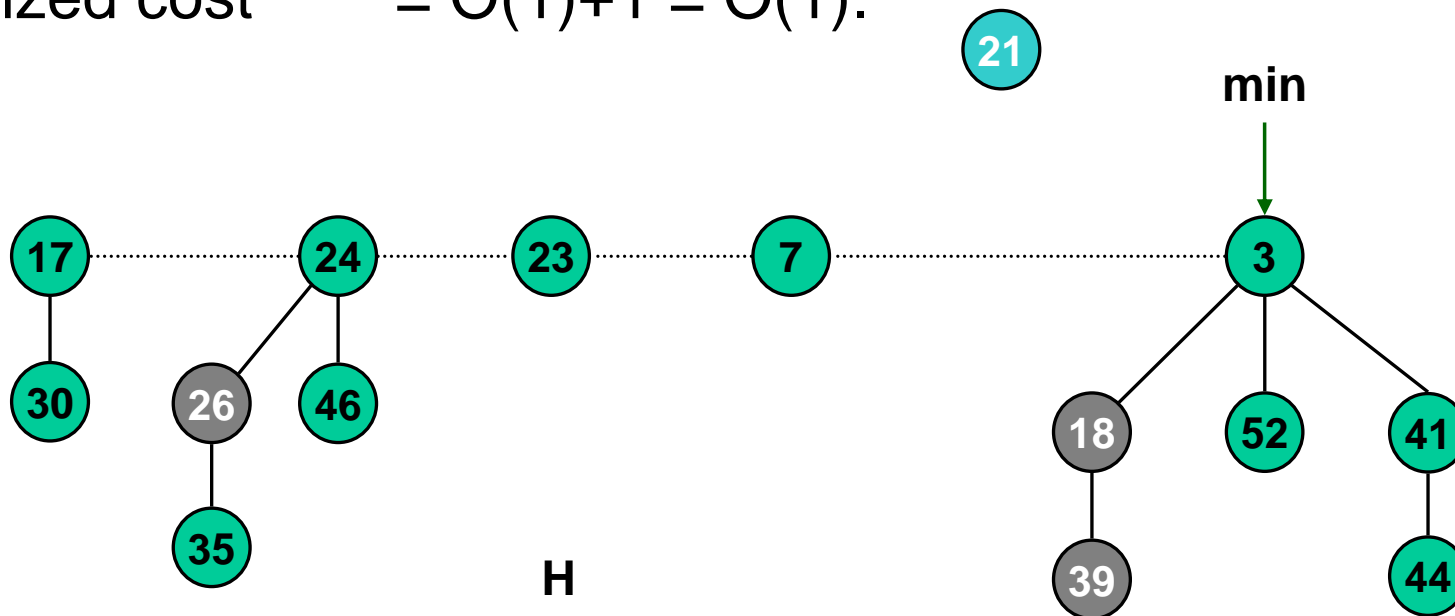
1. Create a new singleton tree.
2. Add to left of min pointer.
3. Update min pointer.

Actual cost = $O(1)$.

Change in potential = ? 1

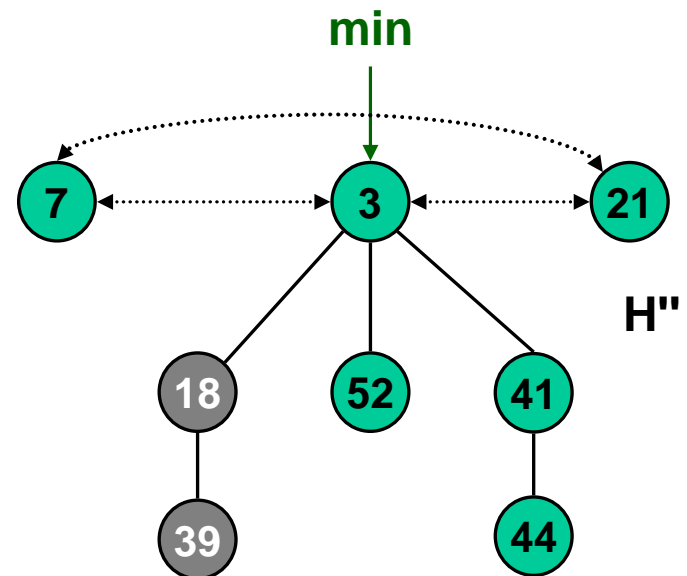
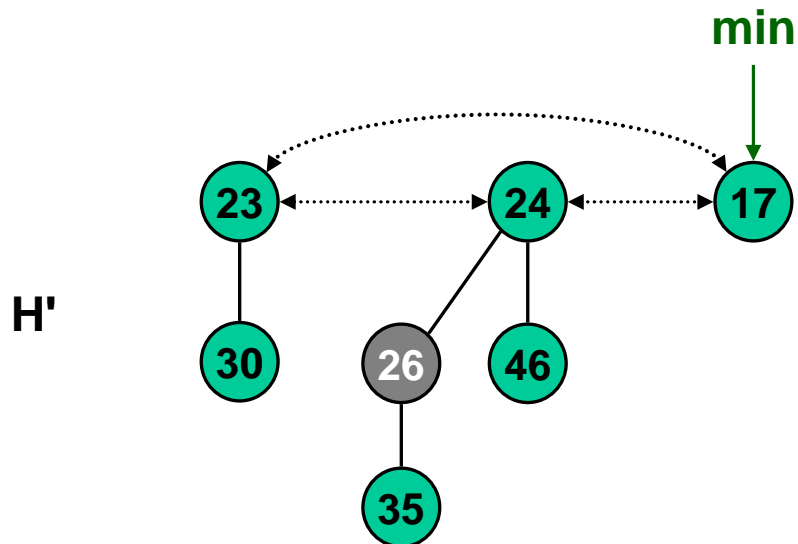
Amortized cost = $O(1)+1 = O(1)$.

$$\Phi(H) = t(H) + 2m(H)$$



Fibonacci Heaps: Union

1. Concatenate heaps.
2. Keep pointer to the minimum of the two minimums.



Fibonacci Heaps: Union

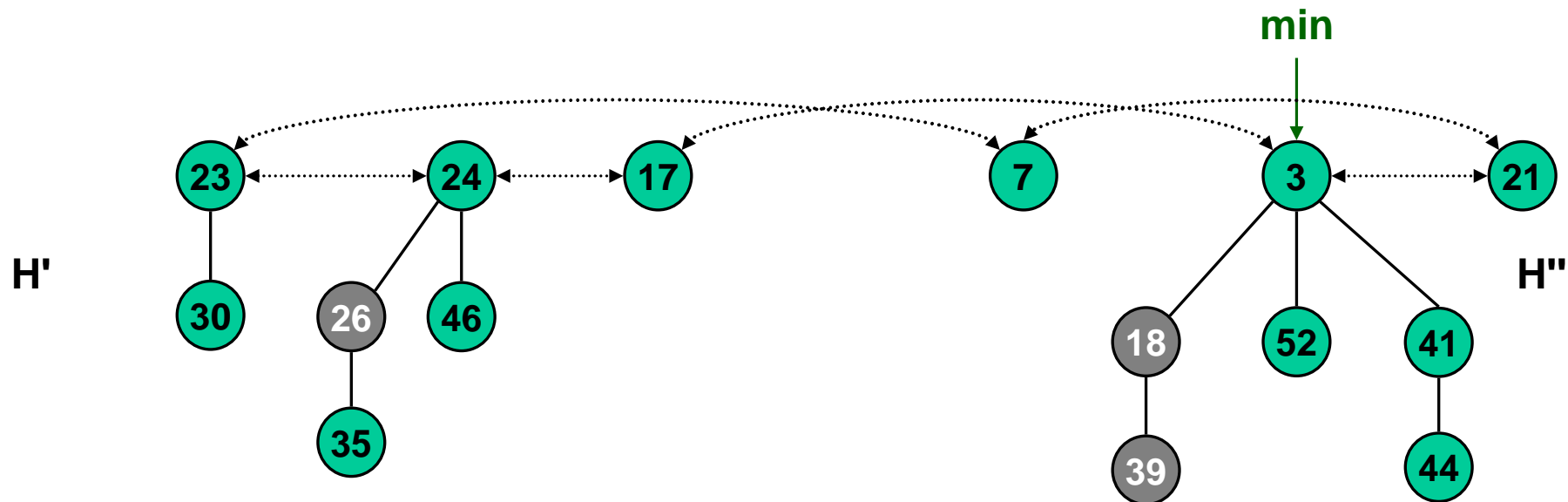
1. Concatenate heaps.
2. Keep pointer to the minimum of the two minimums.

Actual cost = $O(1)$

Change in potential = 0

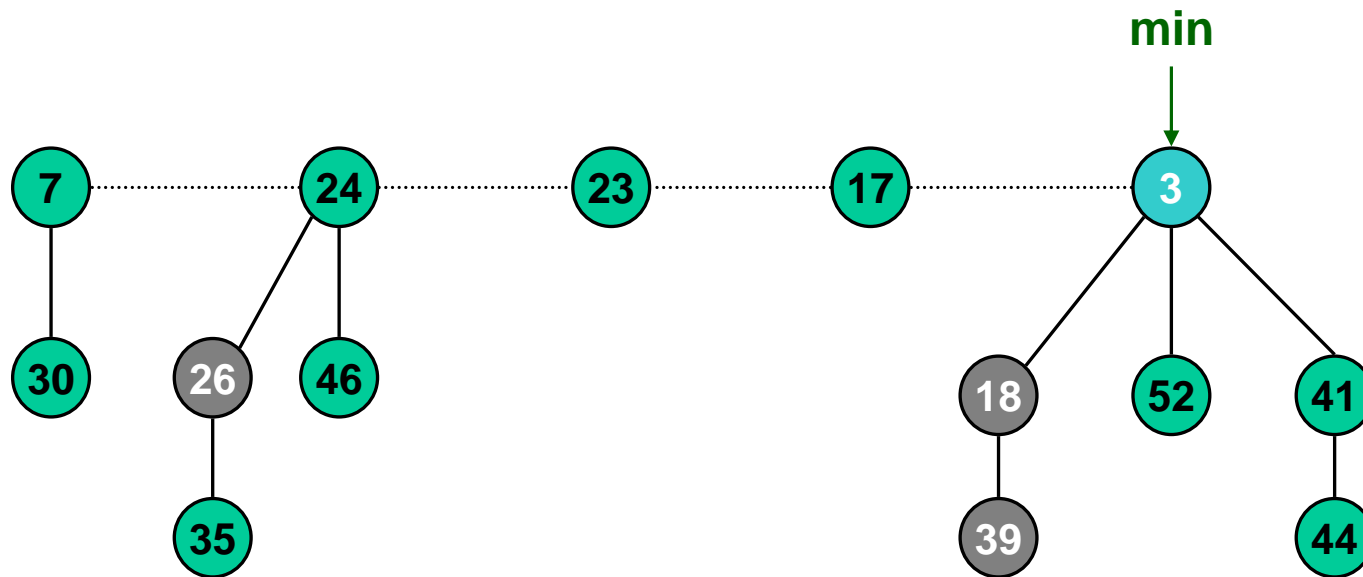
Amortized cost = $O(1)$

$$\Phi(H) = t(H) + 2m(H)$$



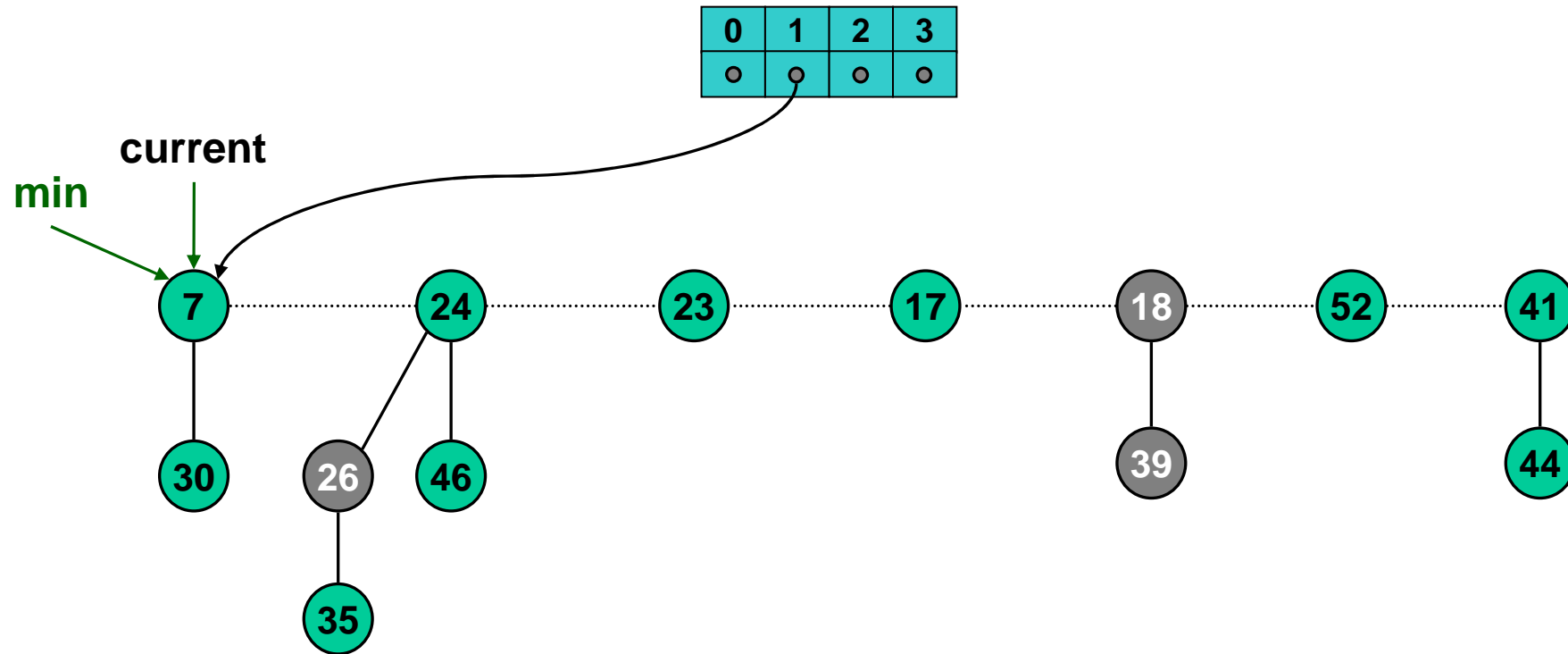
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



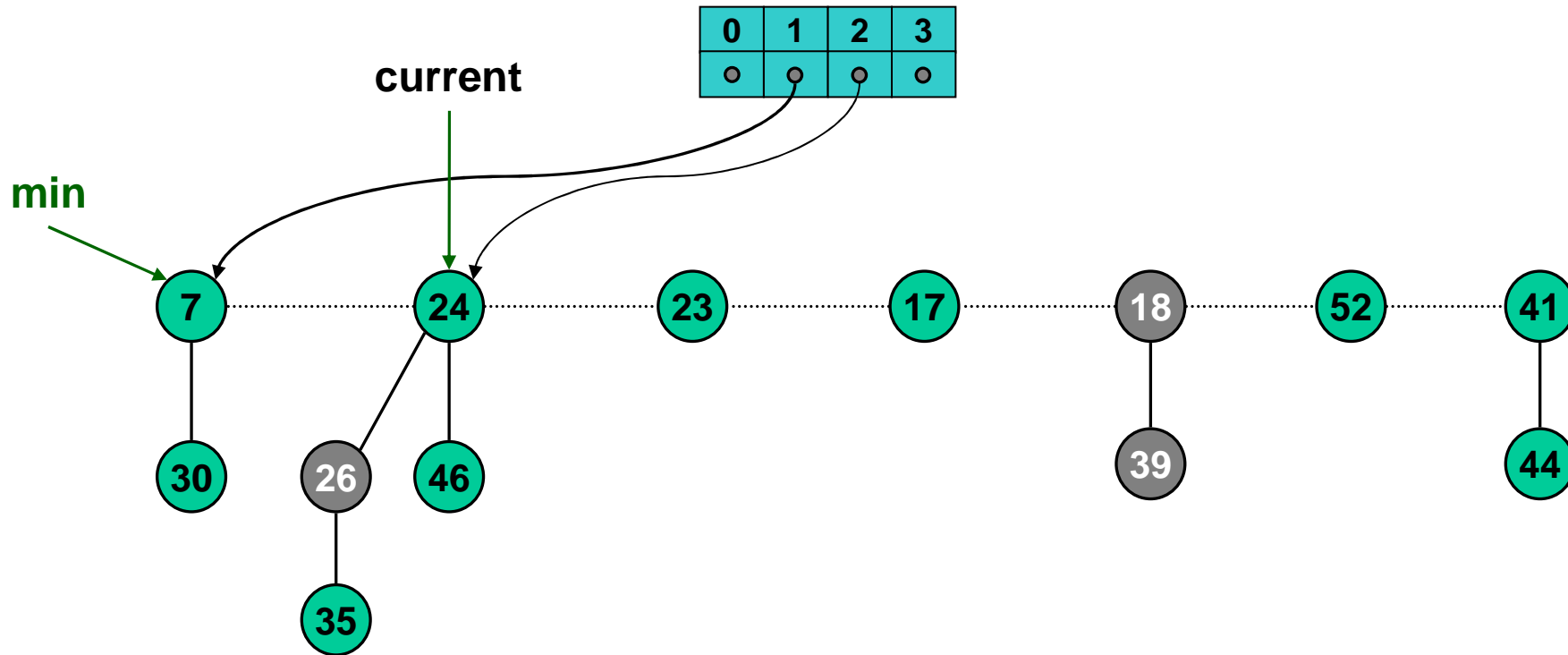
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



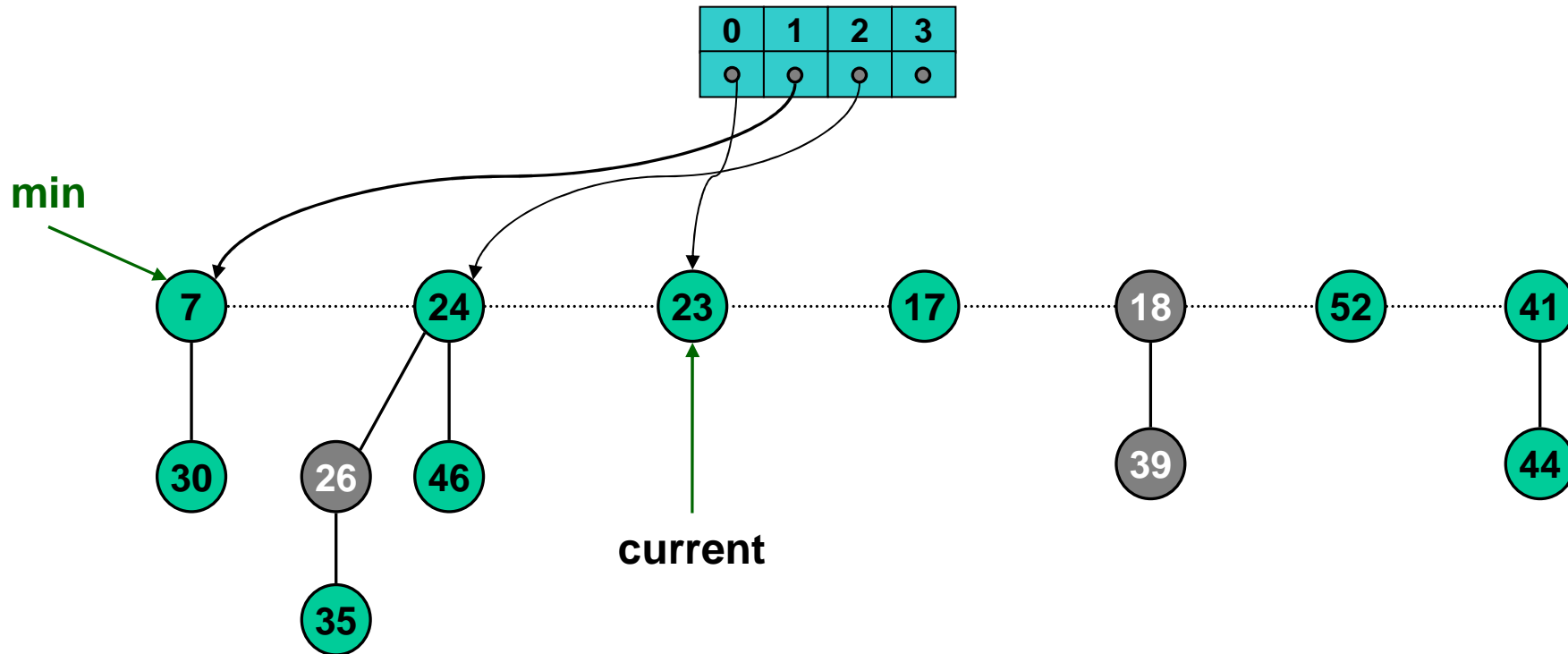
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



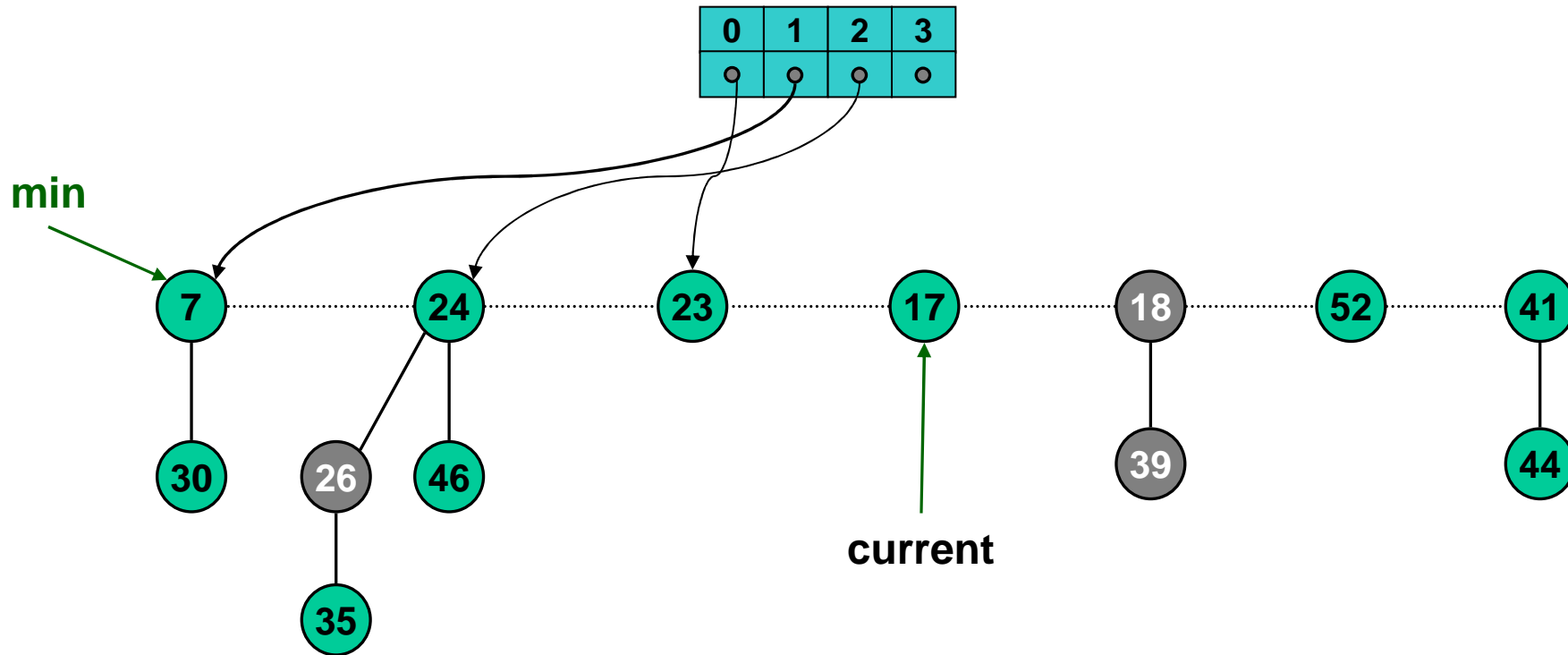
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Fibonacci Heaps: Delete-Min

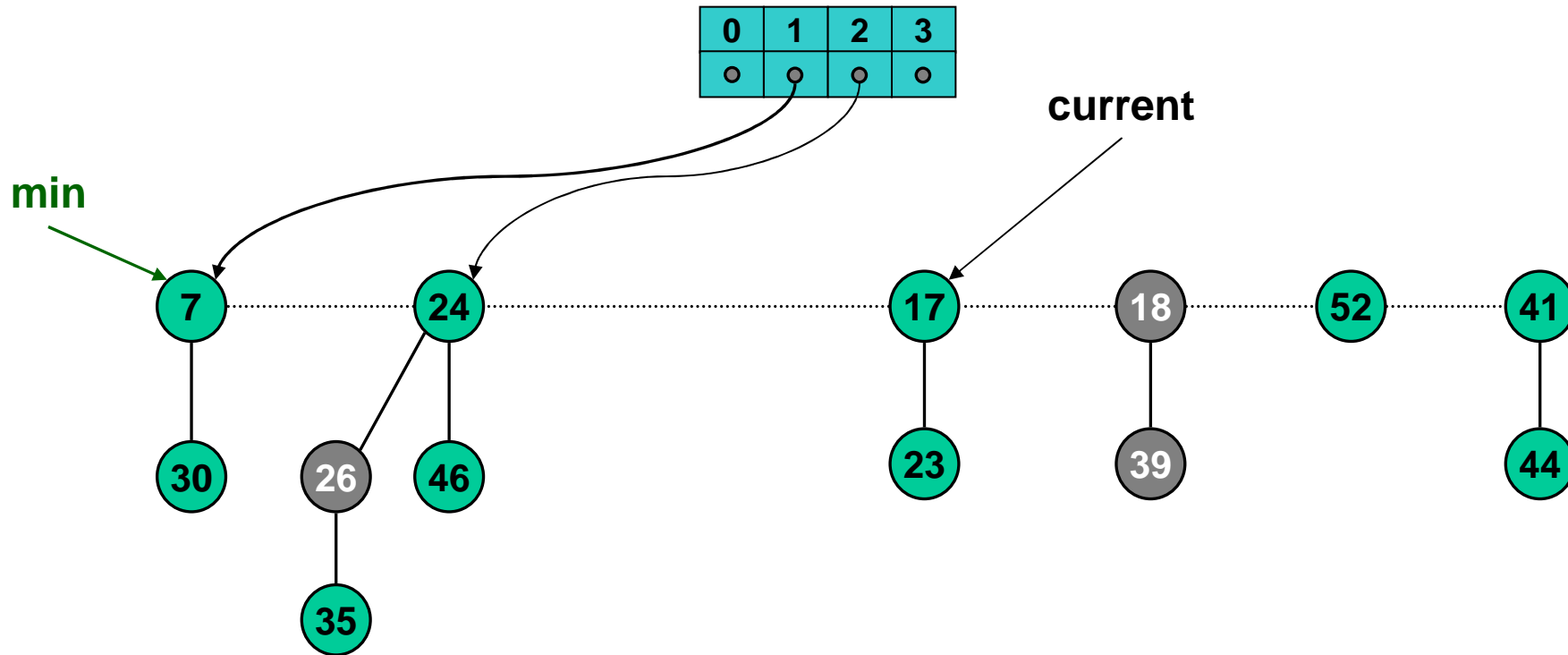
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 17 & 23 trees.

Fibonacci Heaps: Delete-Min

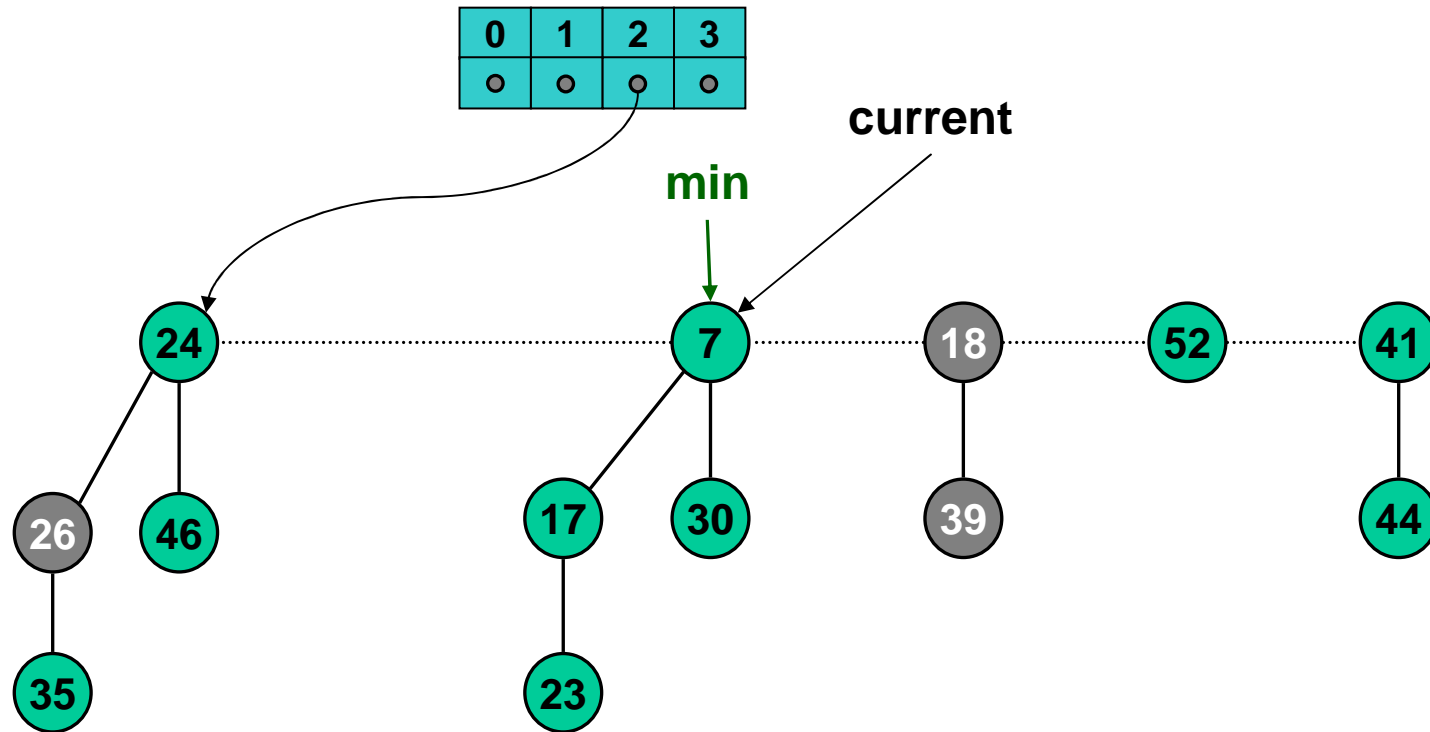
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 17 & 7 trees.

Fibonacci Heaps: Delete-Min

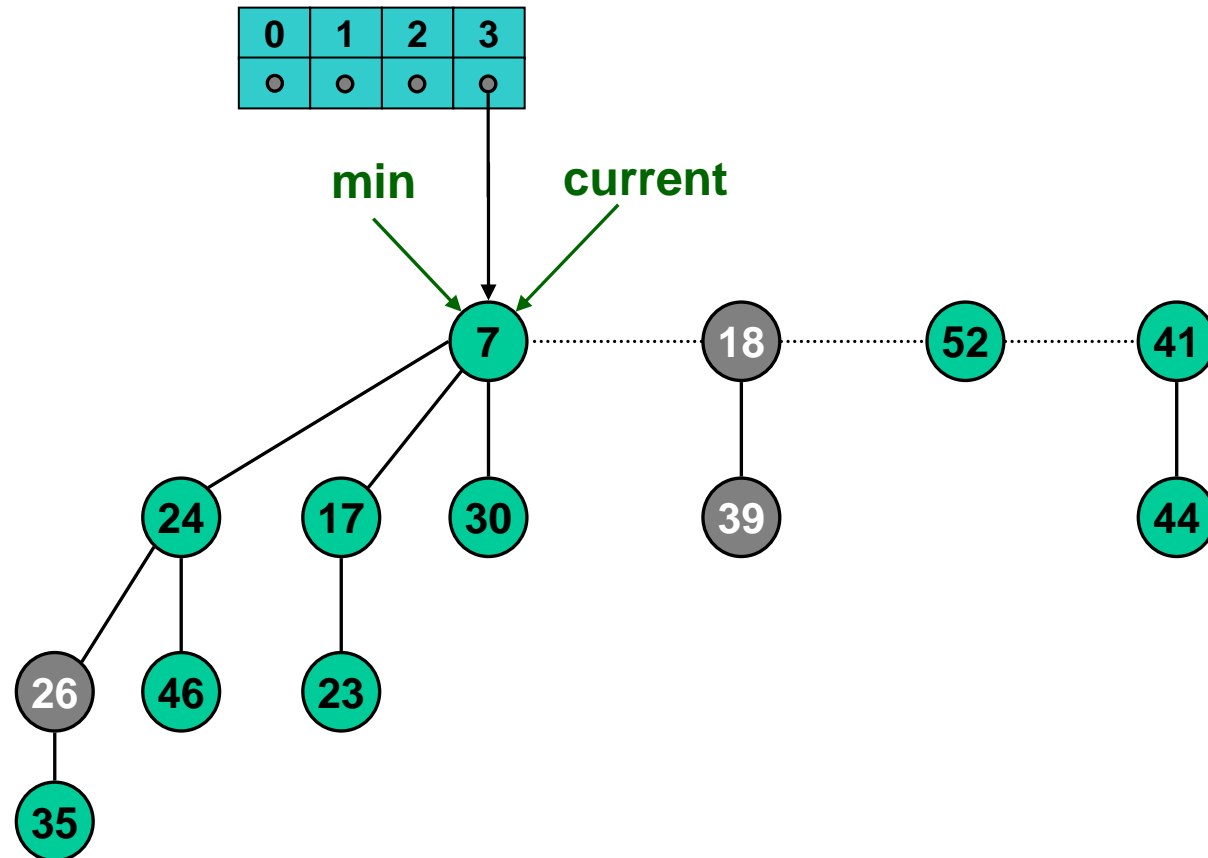
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 7 & 24 trees.

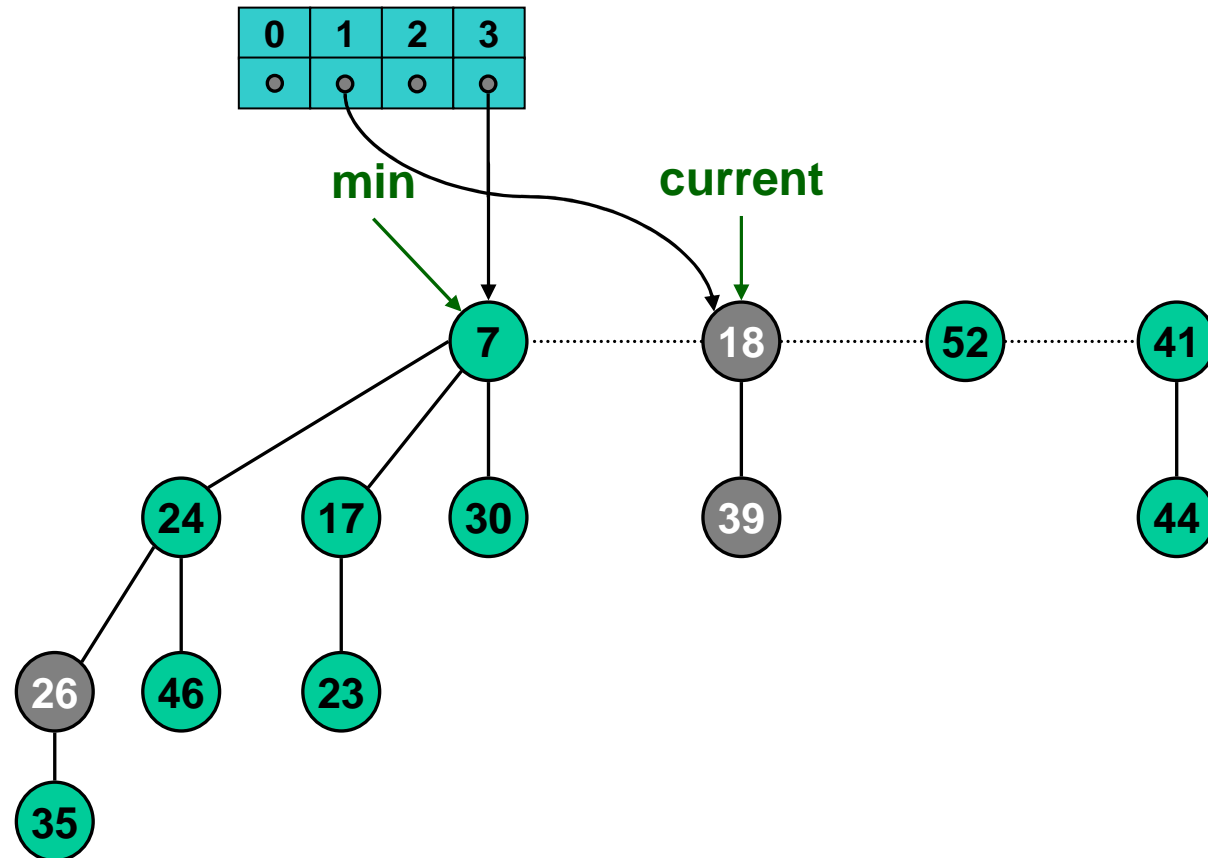
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



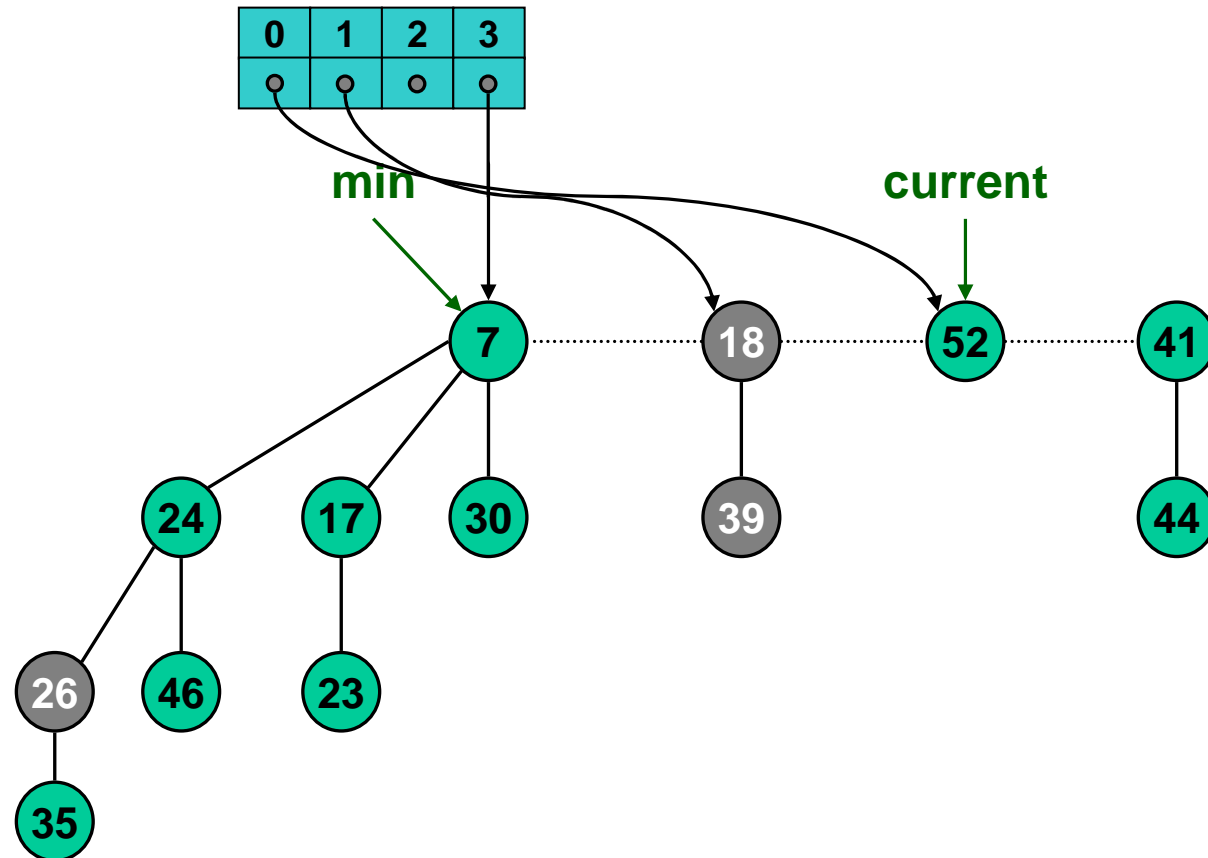
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



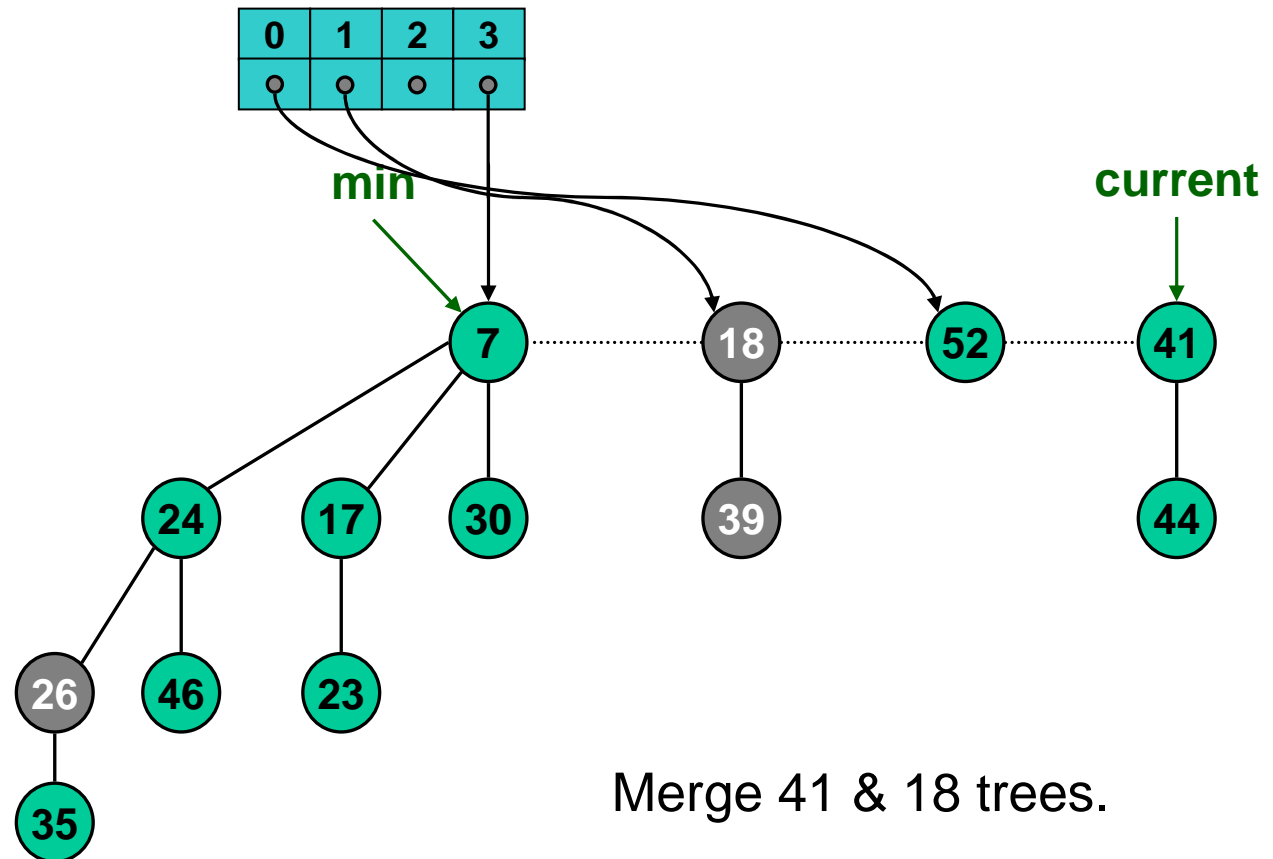
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



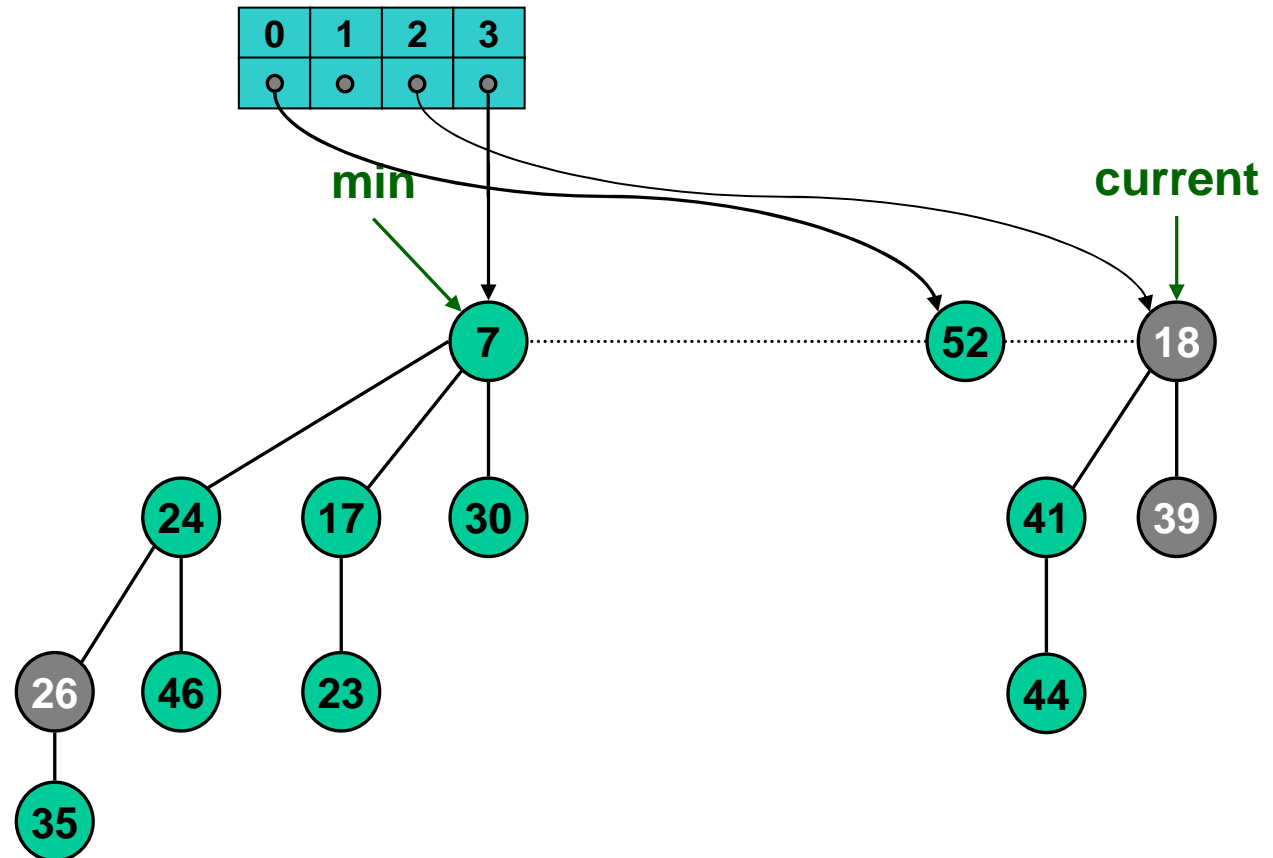
Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Fibonacci Heaps: Delete-Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Fibonacci Heaps: Delete-Min Analysis

$$\Phi(H) = t(H) + 2m(H)$$

$$D(n) = \text{max degree of any node in Fibonacci heap with } n \text{ nodes}$$

$$\text{Actual cost} = O(D(n) + t(H))$$

- $O(1)$ work adding min's children into root list & updating min.
- $O(D(n) + t(H))$ work consolidating trees.
 - At most $D(n)$ children of min node.
 - $\leq D(n) + t(H) - 1$ trees at beginning of consolidation.
 - #trees decreases by one after each merging

$$\text{Amortized cost} = O(D(n) + t(H)) + \Delta\Phi(H) = O(D(n))$$

- $t(H') \leq D(n) + 1$, since no two trees have same degree.
- $m(H') \leq m(H)$
- $\Delta\Phi(H) \leq D(n) + 1 - t(H)$.

Fibonacci Heaps: Delete-Min Analysis

Is amortized cost of $O(D(n))$ good?

Yes, if only Insert, Union, & Delete-min supported.

- In this case, Fibonacci heap contains only binomial trees, since we only merge trees of equal root degree.
- $D(n) \leq \lfloor \log_2 N \rfloor$

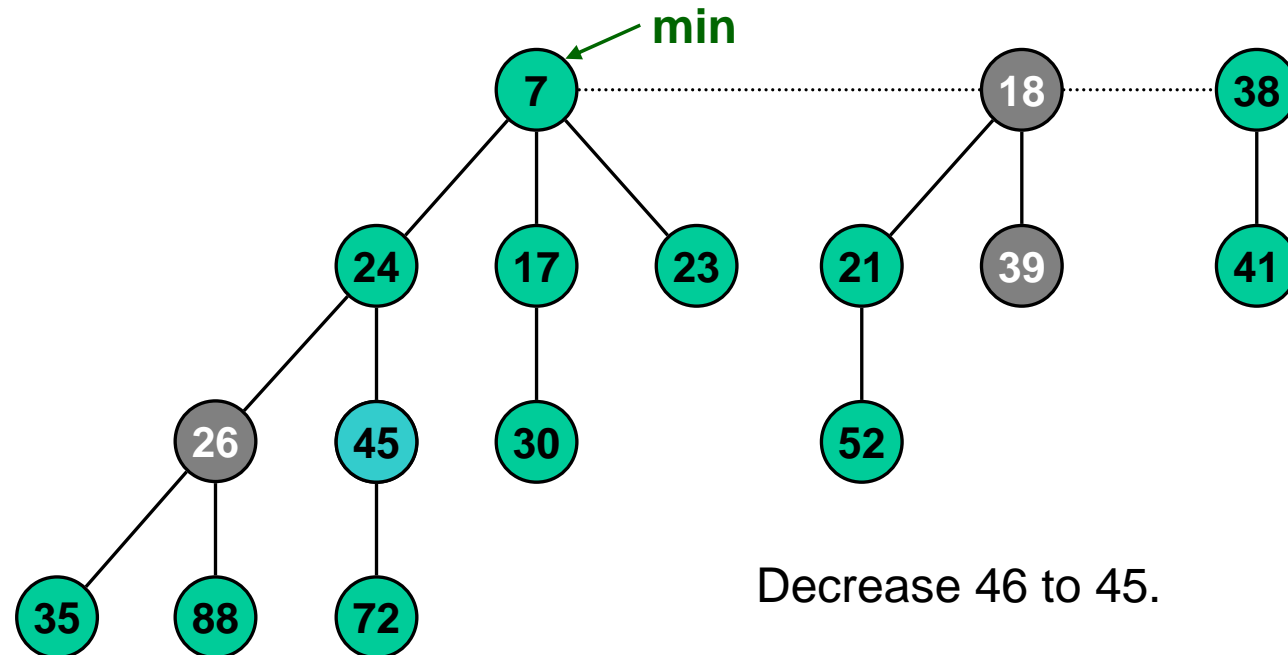
Yes, if we support Decrease-key cleverly.

- $D(n) \leq \lfloor \log_\phi N \rfloor$, where ϕ is golden ratio = 1.618...

Fibonacci Heaps: Decrease-Key

Case 0: min-heap property not violated.

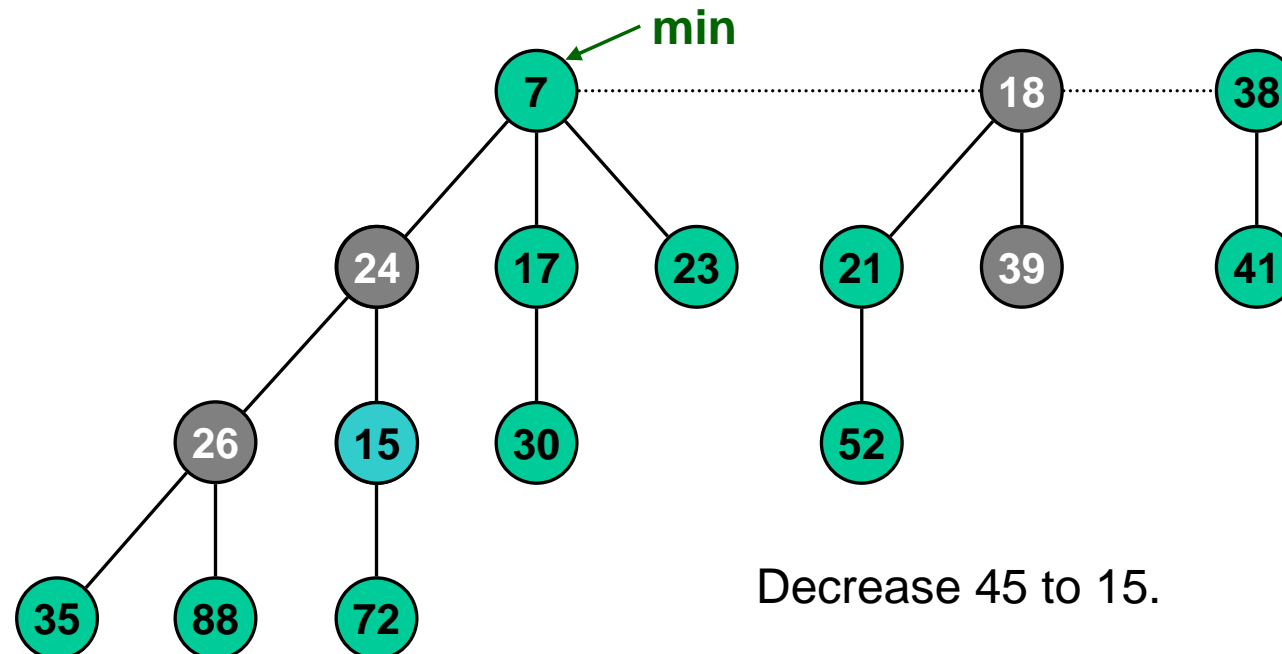
1. Decrease key.
2. Change min pointer if necessary.



Fibonacci Heaps: Decrease-Key

Case 1: parent of x is unmarked.

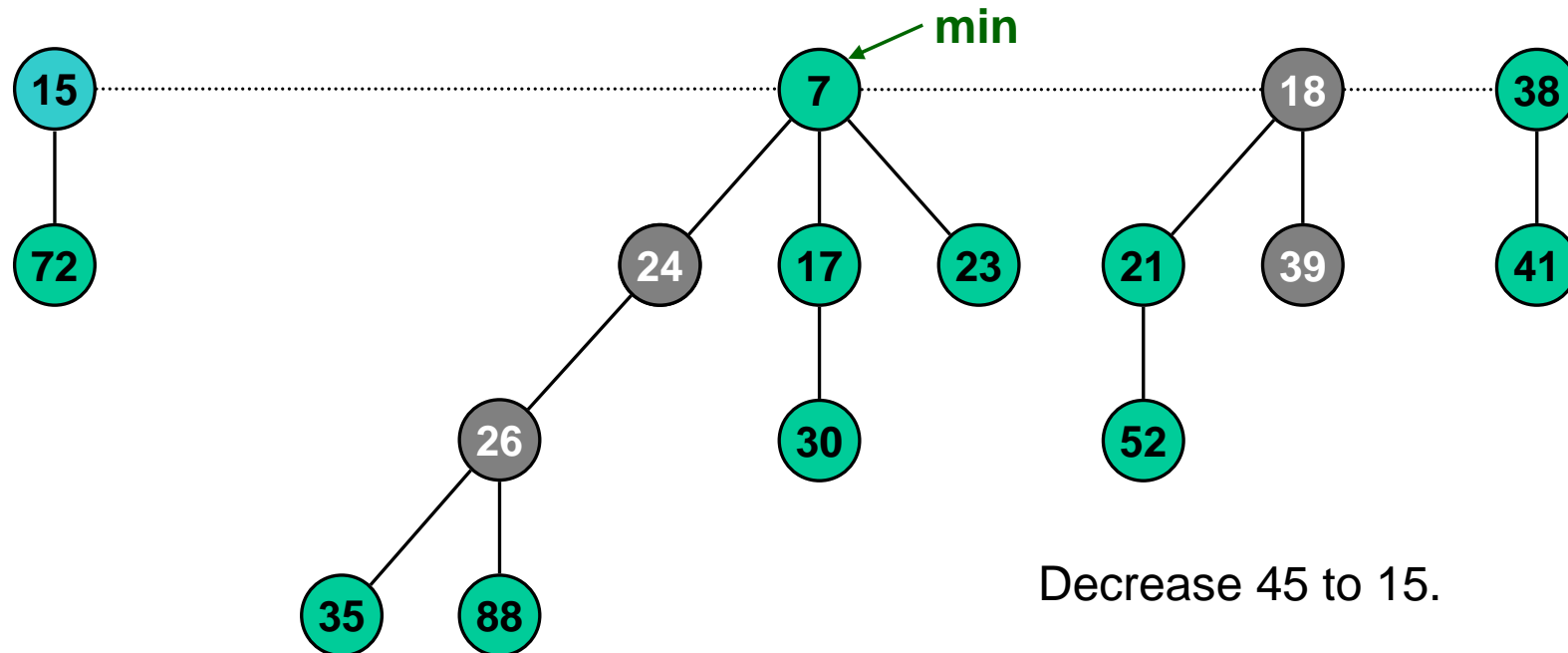
1. Decrease key.
2. Remove link to parent.
3. Mark parent.
4. Add x's tree to root list, updating heap min pointer.



Fibonacci Heaps: Decrease-Key

Case 1: parent of x is unmarked.

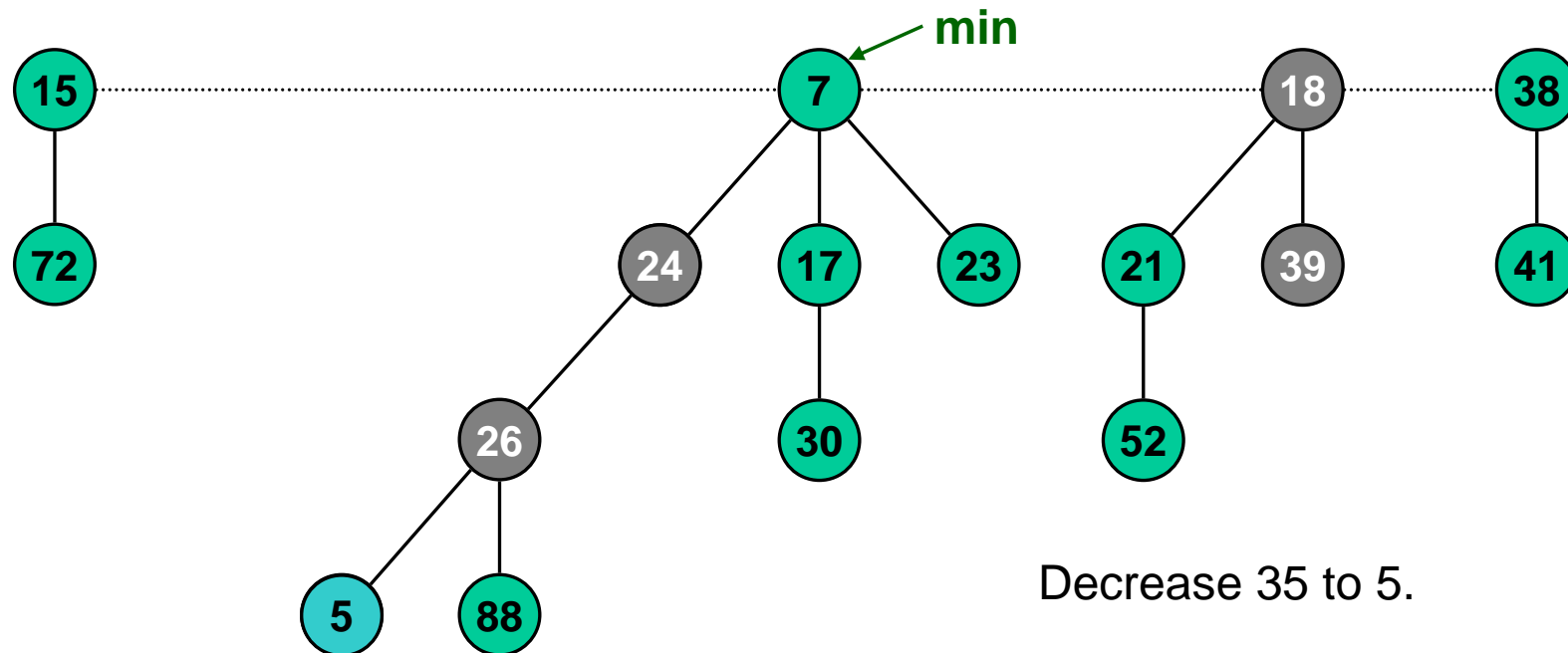
1. Decrease key.
2. Remove link to parent.
3. Mark parent.
4. Add x's tree to root list, updating heap min pointer.



Fibonacci Heaps: Decrease-Key

Case 2: parent of x is marked.

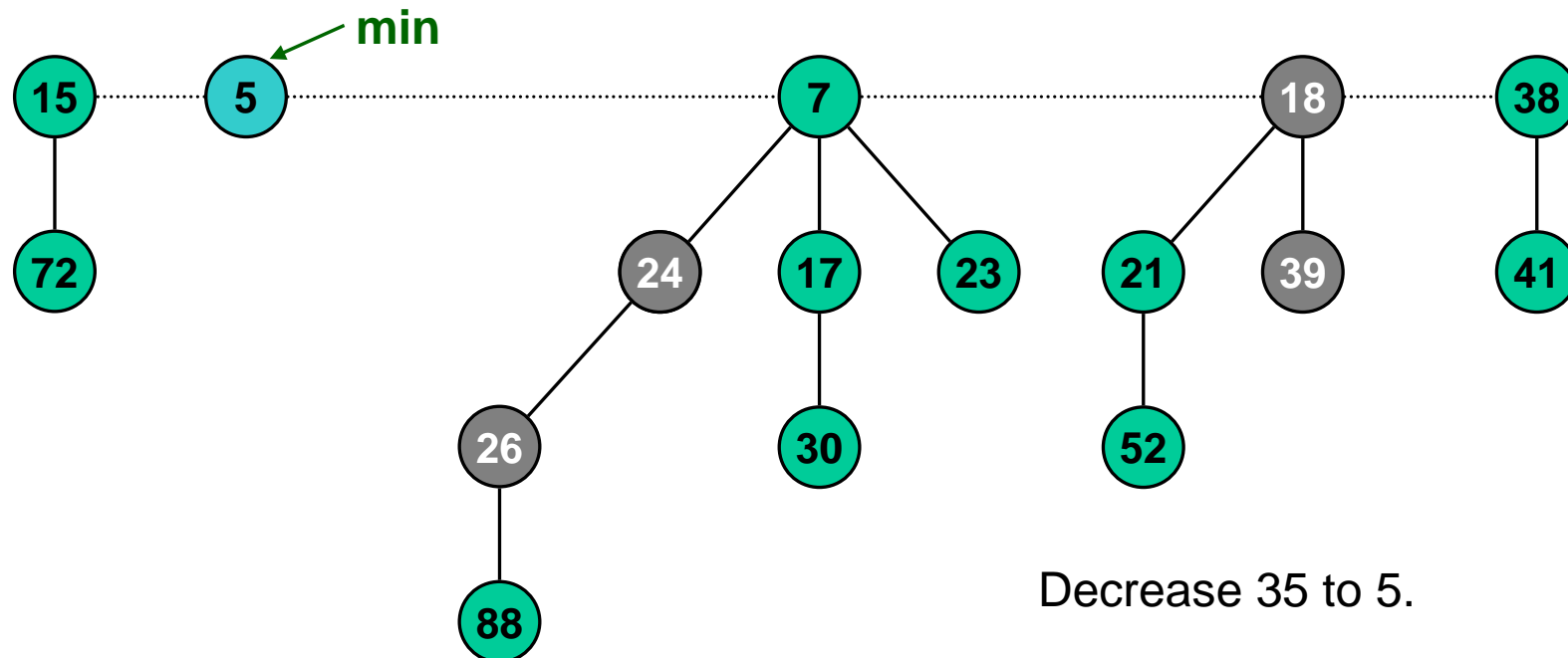
1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.



Fibonacci Heaps: Decrease-Key

Case 2: parent of x is marked.

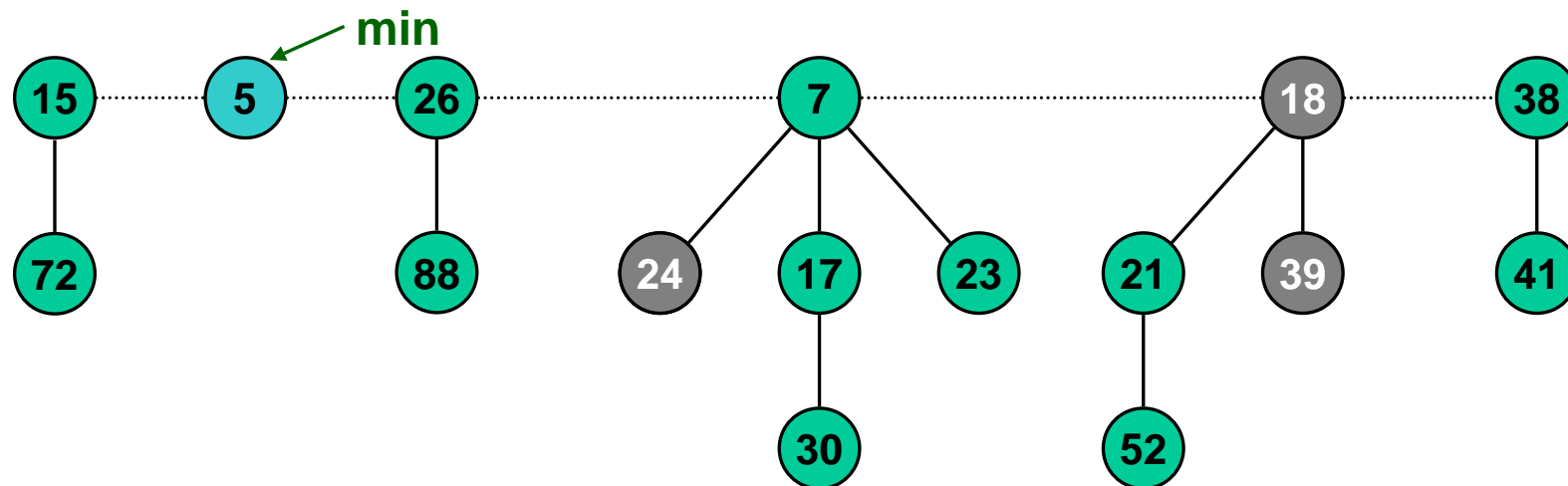
1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.



Fibonacci Heaps: Decrease-Key

Case 2: parent of x is marked.

1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.

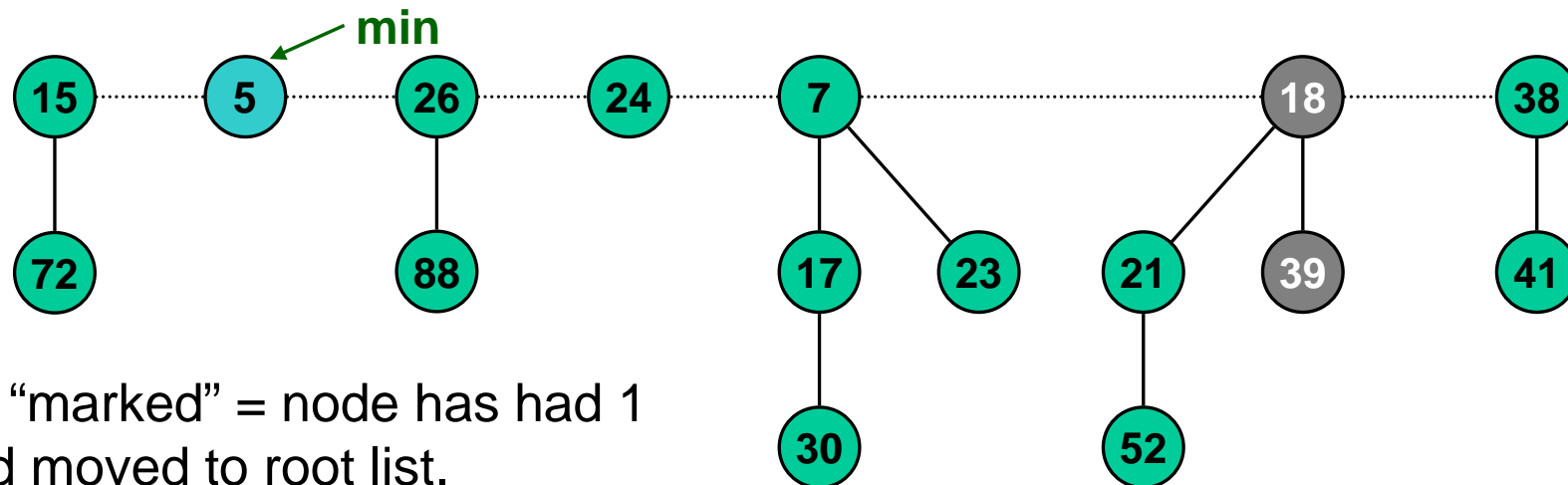


Decrease 35 to 5.

Fibonacci Heaps: Decrease-Key

Case 2: parent of x is marked.

1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.



I.e., “marked” = node has had 1 child moved to root list.
Once we move a 2nd child of node, we also move the node.

Decrease 35 to 5.

Fibonacci Heaps: Decrease-Key Analysis

Notation

- $t(H)$ = # trees in heap H .
- $m(H)$ = # marked nodes in heap H .
- $\Phi(H)$ = $t(H) + 2m(H)$.

Actual cost = $O(c)$, where c = # of nodes “cut”

- $O(1)$ time for decrease key.
- $O(1)$ time **for each** cut.

Amortized cost = $O(c) + \Delta\Phi(H) = O(1)$

- $t(H') = t(H) + c$
- $m(H') \leq m(H) - c + 2$
 - Each cut unmarks a node.
 - Last cut could potentially mark a node.
- $\Delta\Phi(H) \leq c + 2(-c + 2) = 4 - c$.

Fibonacci Heaps: Delete

1. Decrease key of x to $-\infty$.
2. Delete min element in heap.

Amortized cost = $O(D(n))$

- $O(1)$ for decrease-key.
- $O(D(n))$ for delete-min.
- $D(n)$ = max degree of any node in Fibonacci heap.

Fibonacci Heaps: Bounding $D(n)$

$D(n)$ = max degree in Fibonacci heap with n nodes.

Can show:

$$D(n) \leq \log_{\phi} n, \text{ where } \phi = (1 + \sqrt{5}) / 2.$$

Thus, Delete & Delete-min take $O(\log n)$ amortized time.

Proof is somewhat tedious & explained well in [CLRS].

Key Lemma:

Let $\text{size}(x)$ = #nodes in subtree rooted at x .

$$\text{Then, } \phi^{\text{degree}(x)} \leq \text{size}(x).$$