More Heaps

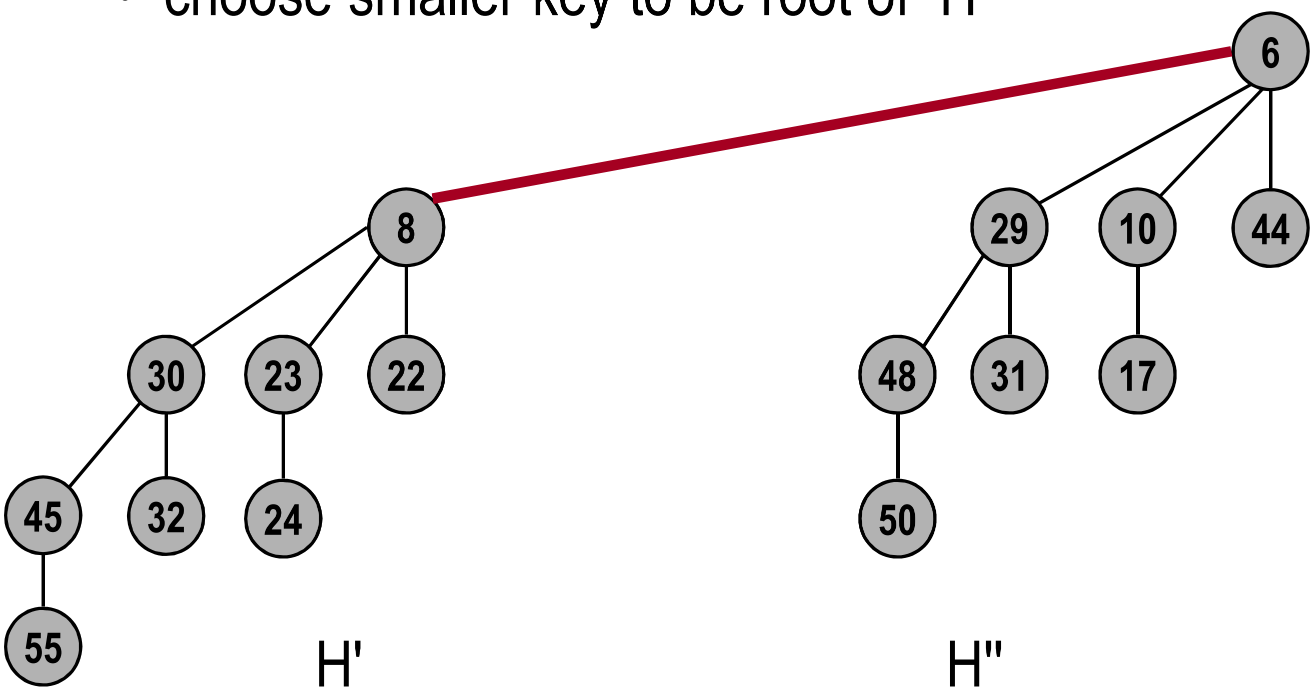
Data Structures and Algorithms

Andrei Bulatov

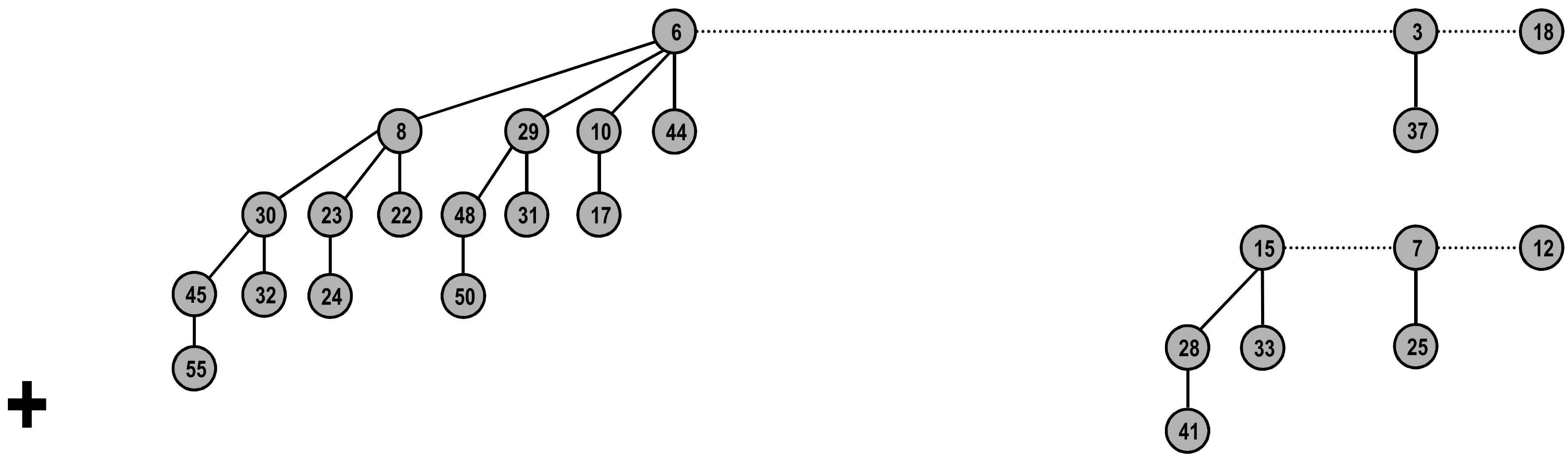
Binomial Heap: Union

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



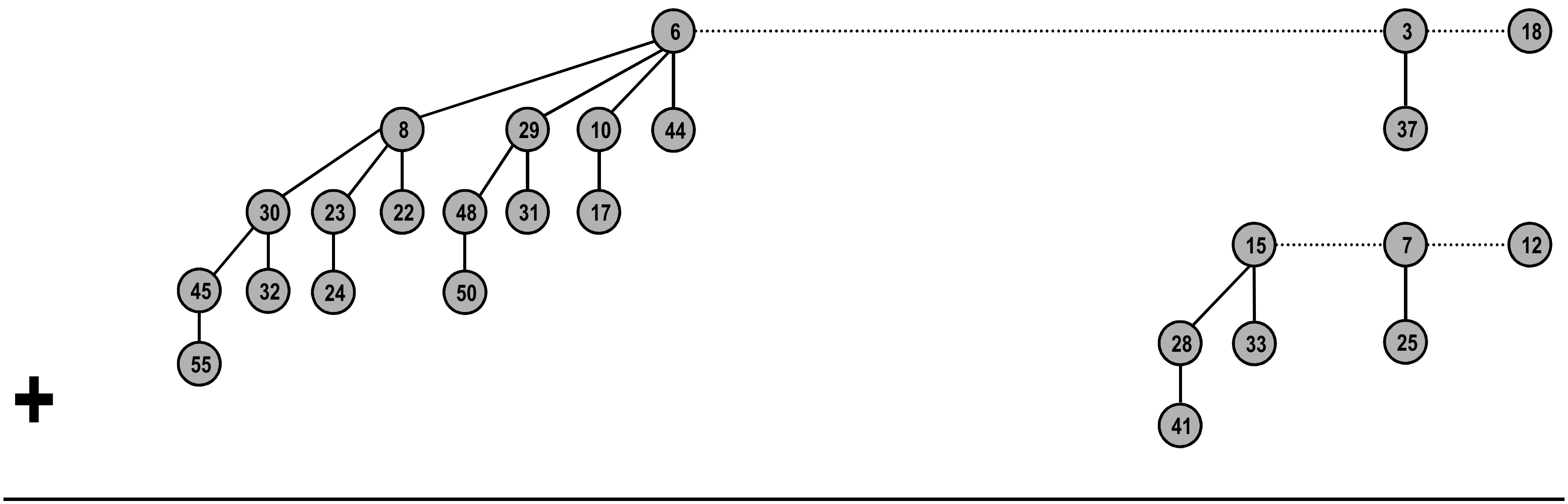
Binomial Heap: Union



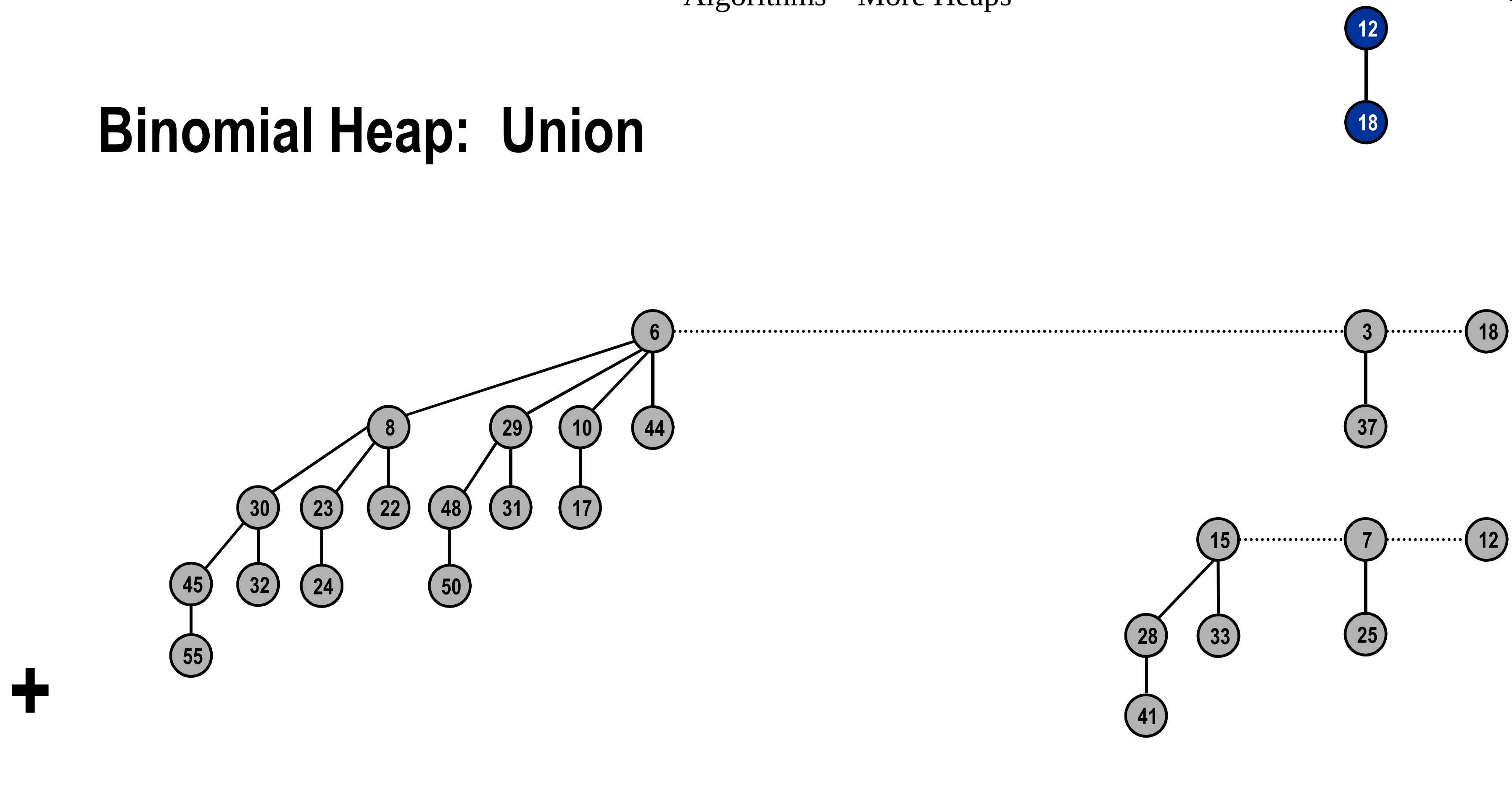
19 + 7 = 26

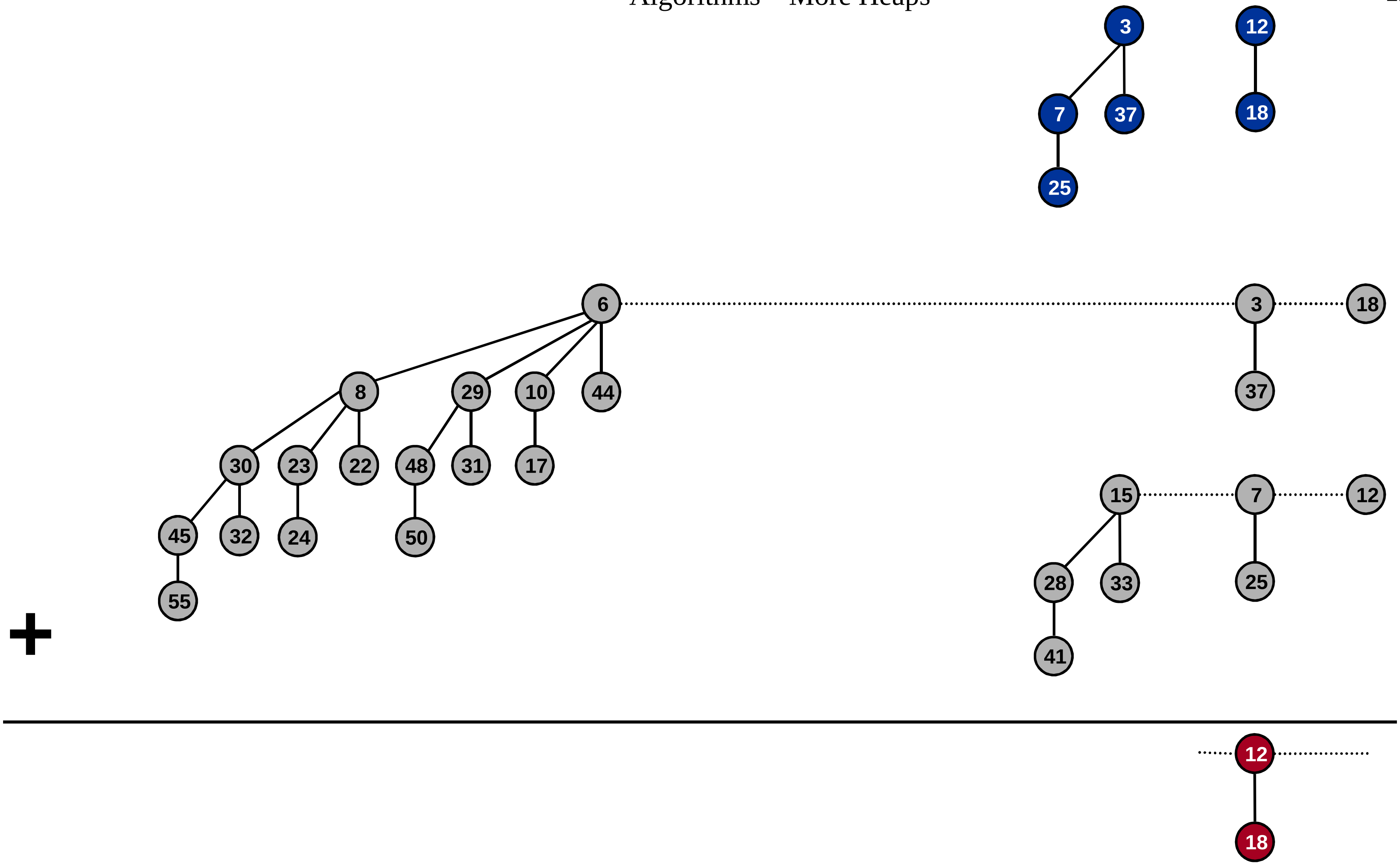
		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

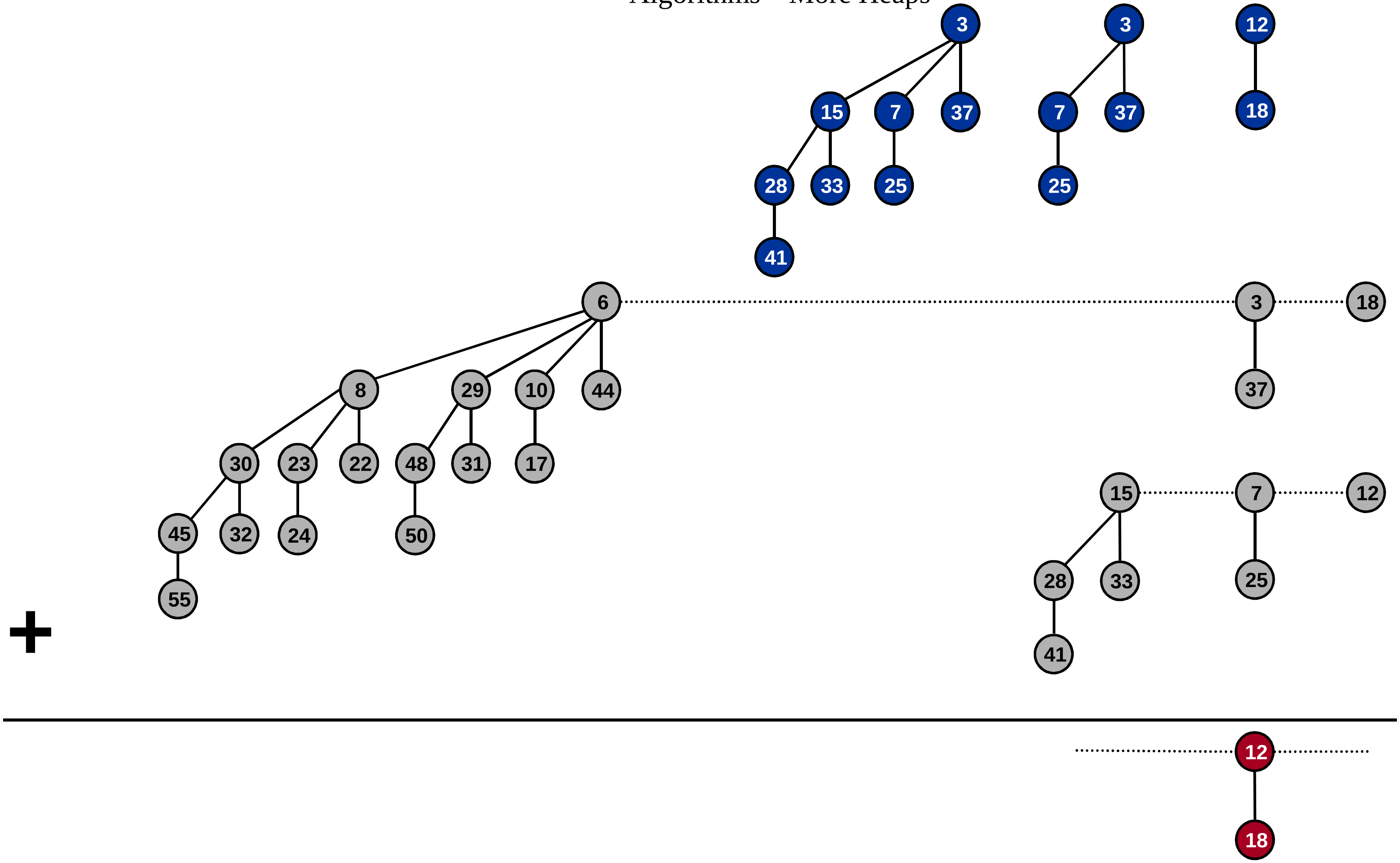
Binomial Heap: Union

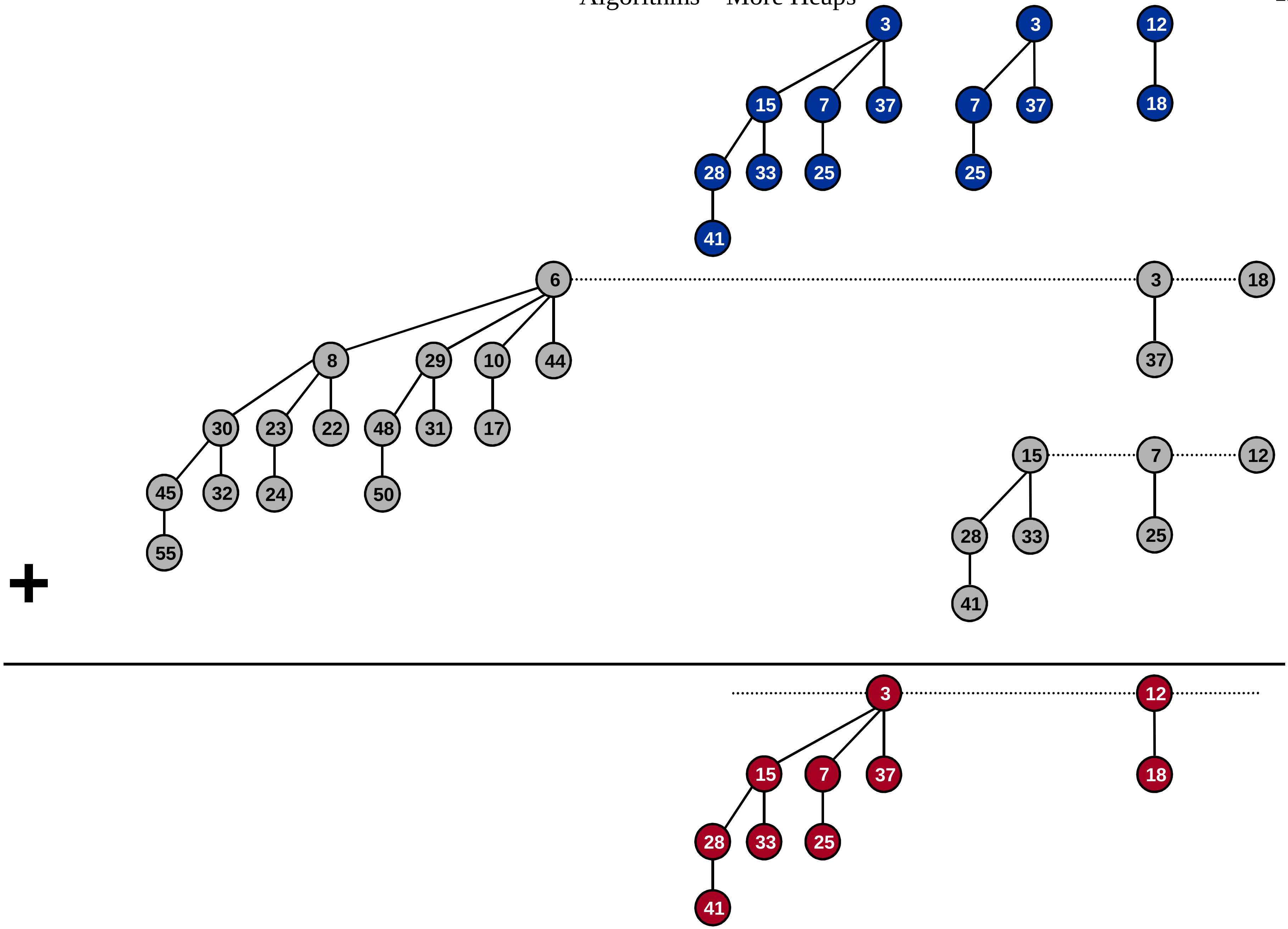


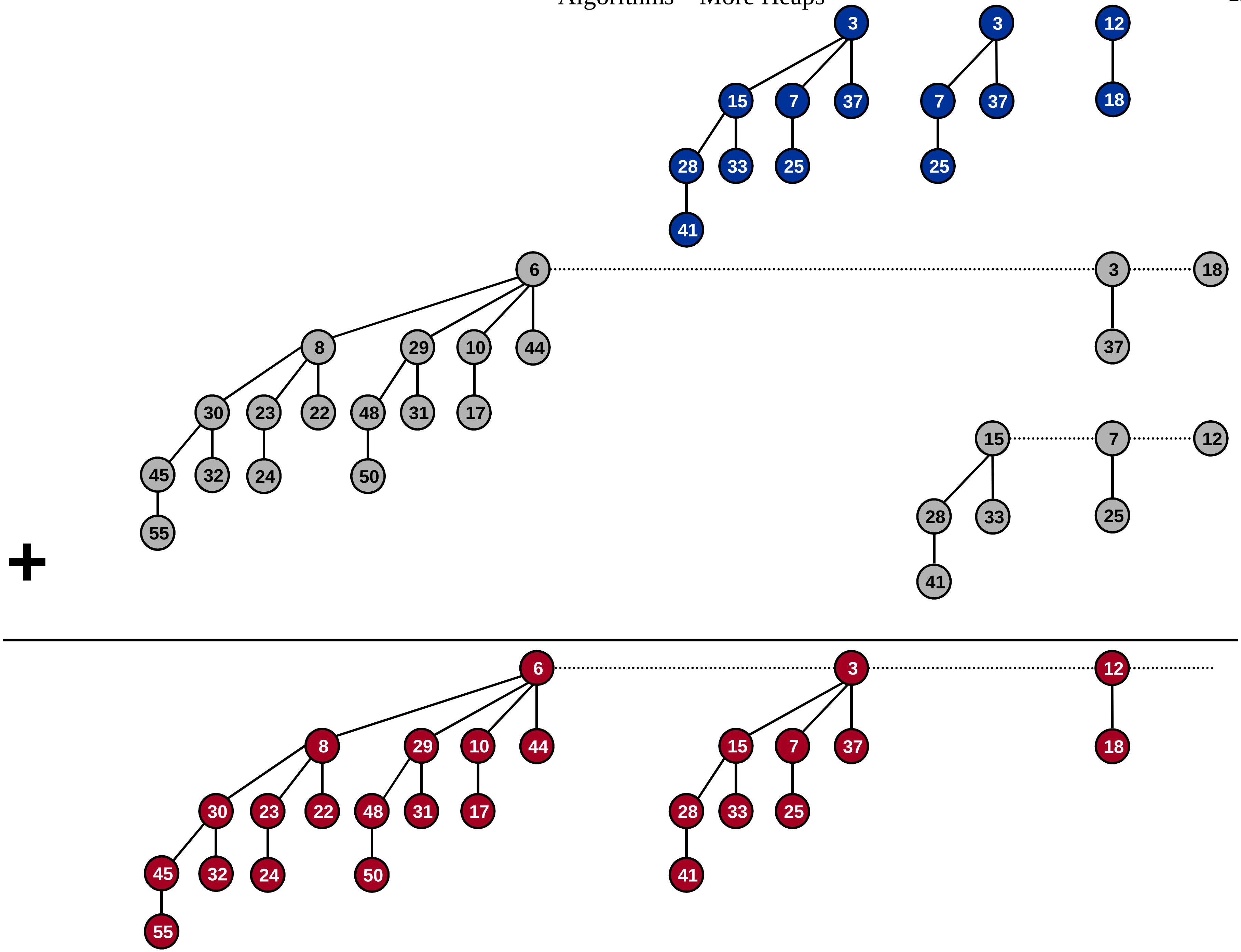
Binomial Heap: Union




3
12
7
37
18
25
6
3
18
8
29
10
44
37
30
23
22
48
31
17
15
7
12
45
32
24
50
28
33
25
55
+
41
12
18


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
12
18


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
3
12
15
7
37
18
28
33
25
41


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
37
18
15
7
12
28
33
25
41
+
6
3
12
8
29
10
44
15
7
37
18
30
23
22
48
31
17
28
33
25
45
32
24
50
41
55

Union: Running Time

Theorem

Union can be executed in $O(\log n)$ time

Proof

The running time is proportional to the number of trees in root lists, which is at most $2(\lfloor \log N \rfloor + 1)$.

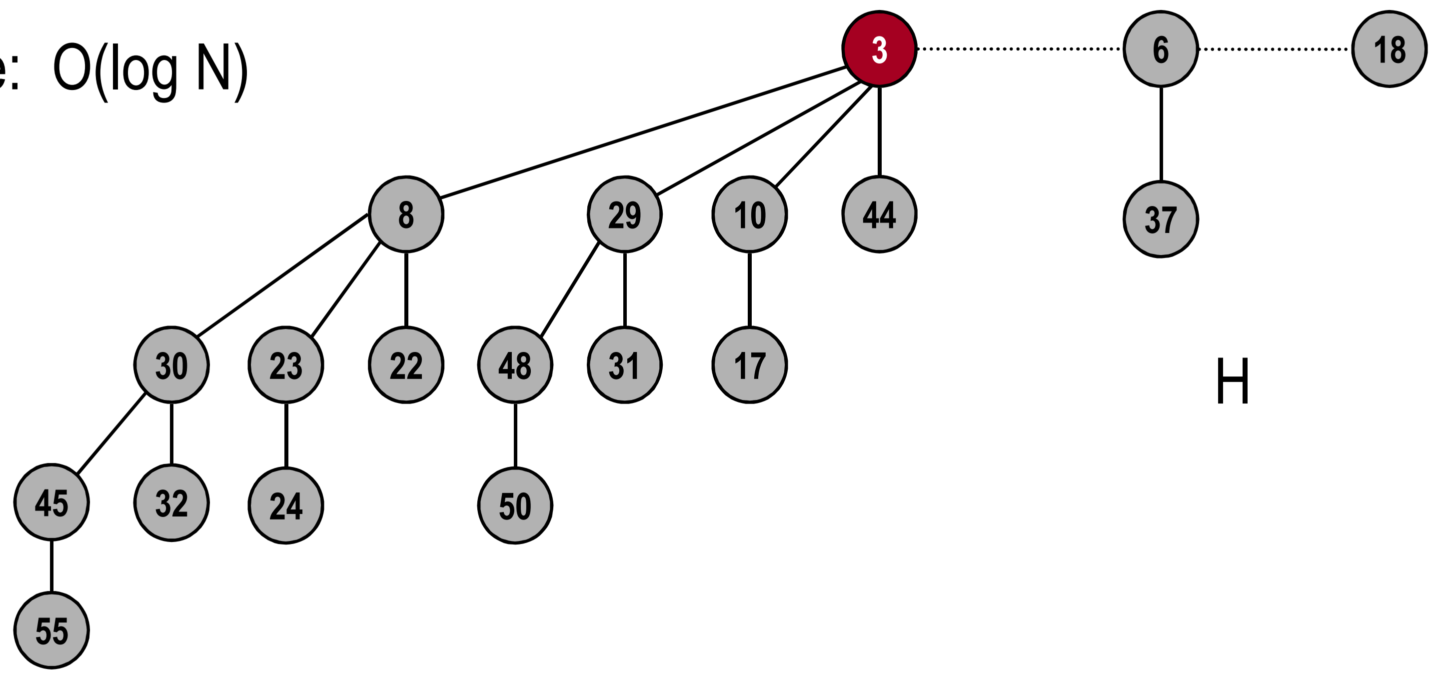
QED

Delete Minimal

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)

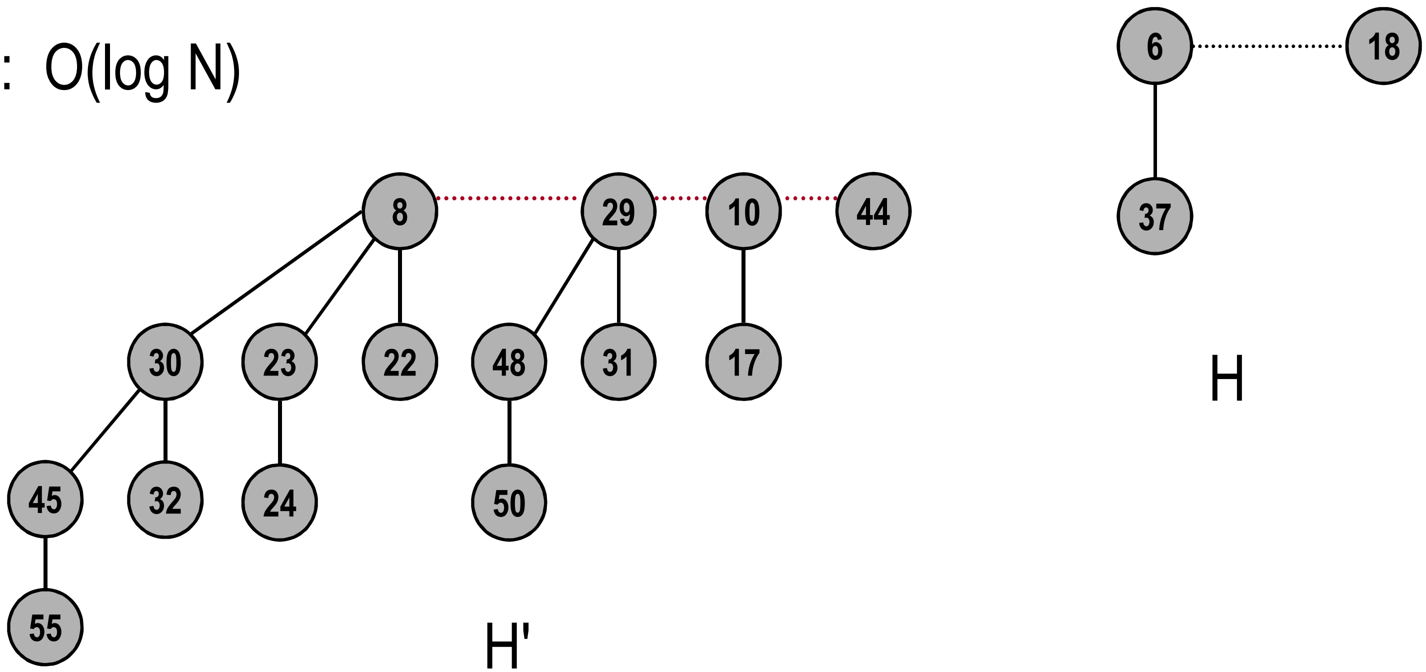


Delete Minimal

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)



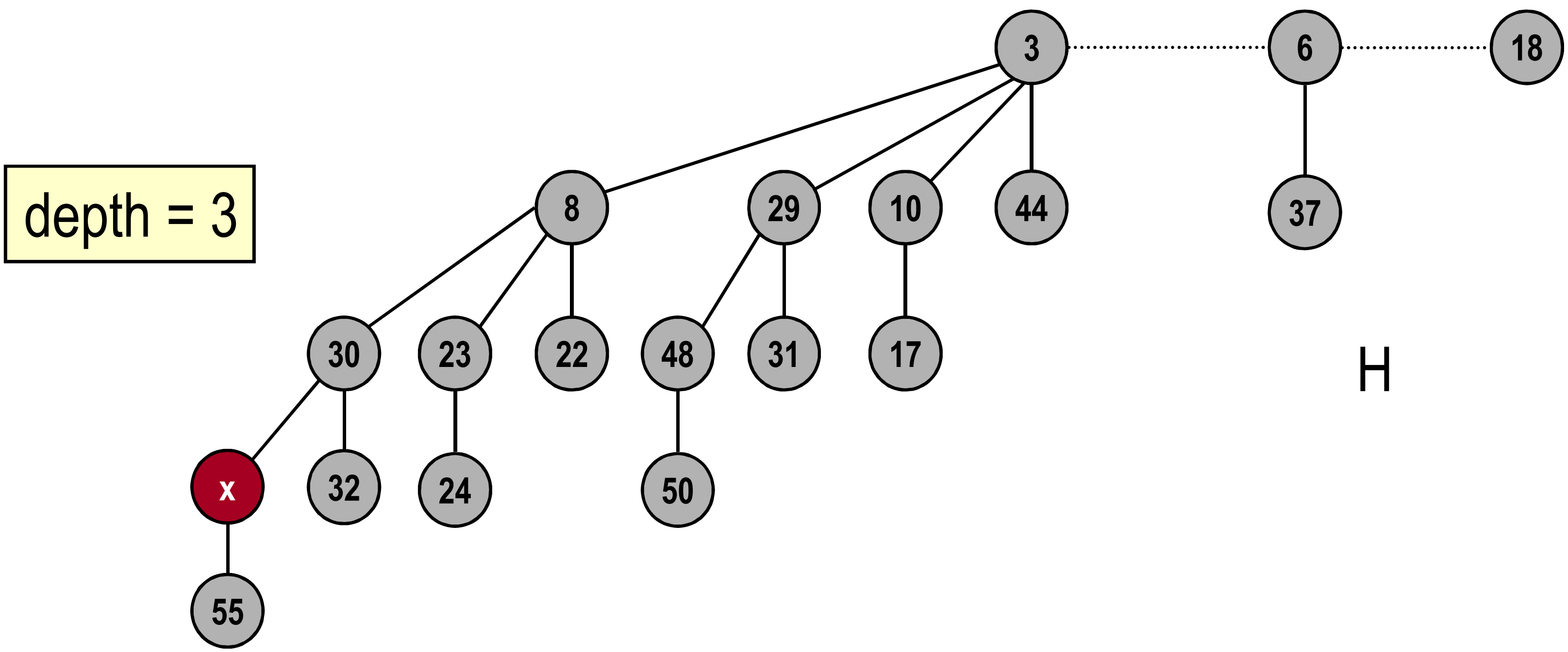
Decrease Key

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 $x \leq \lfloor \log_2 N \rfloor$.



Delete

Delete node x in binomial heap H.

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

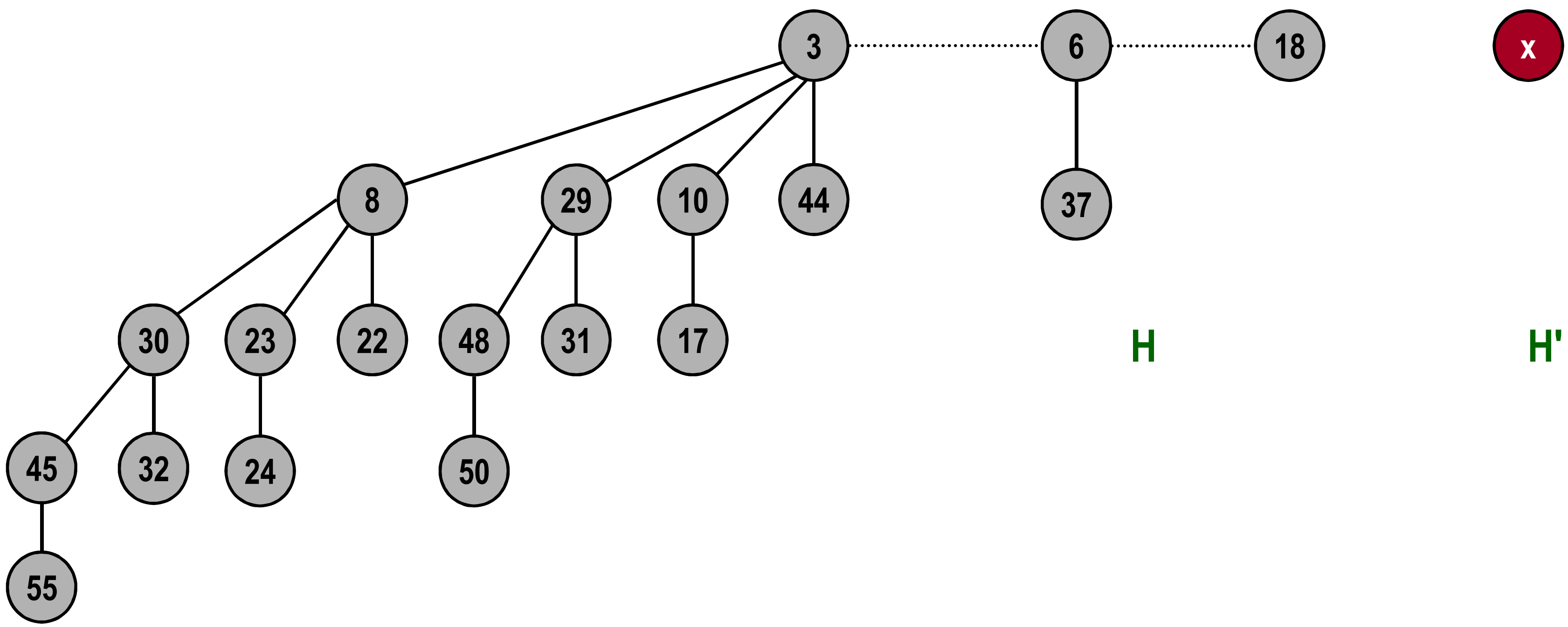
Running time: O(log N)

Insert

Insert a new node x into binomial heap H.

- H' $\leftarrow$ MakeHeap(x)
- H $\leftarrow$ Union(H', H)

Running time: O(log N)



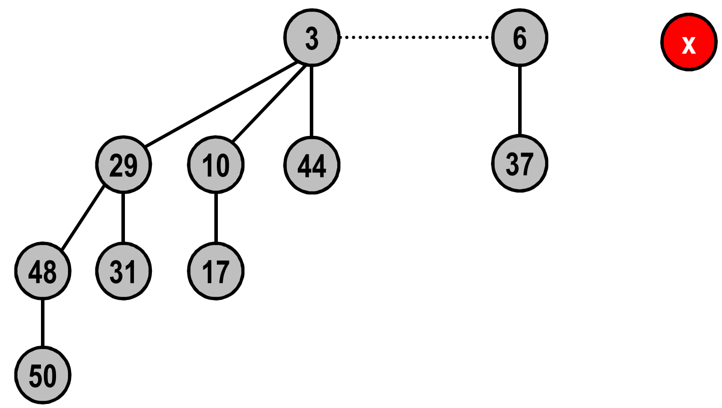
Amortized Analysis

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 $\Omega(\log N)$ time.

- If N = 11...111, then $\log_2 N$ steps.



Theorem

n Insert operations can be executed in O(n) time

Amortized Analysis (cntd)

Proof

By what we saw on the previous slide, time needed for n Inserts is

$$\frac{n}{2}\cdot 1+\frac{n}{4}\cdot 2+\frac{n}{8}\cdot 3+\ldots \le \sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i}\cdot i$$

Or, in other words,

$$\sum_{i=1}^{\lceil \log n \rceil} \frac{n}{2^i}\cdot i = \frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\ldots$$

$$+ \quad \frac{n}{4}+\frac{n}{8}+\ldots$$

$$+ \quad \frac{n}{8}+\ldots$$

$$= n+\frac{n}{2}+\frac{n}{4}+\ldots \le 2n$$

Fibonacci Heaps

Data Structures and Algorithms

Andrei Bulatov

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

Linked List: $O(|V|^2)$

Binary / Binomial: $O(|E| \log |V|)$

Fibonacci / Relaxed: $O(|E| + |V| \log |V|)$

Dijkstra/Prim
1 make-heap
|V| insert
|V| delete-min
|E| decrease-key

Fibonacci Heaps

Intuition

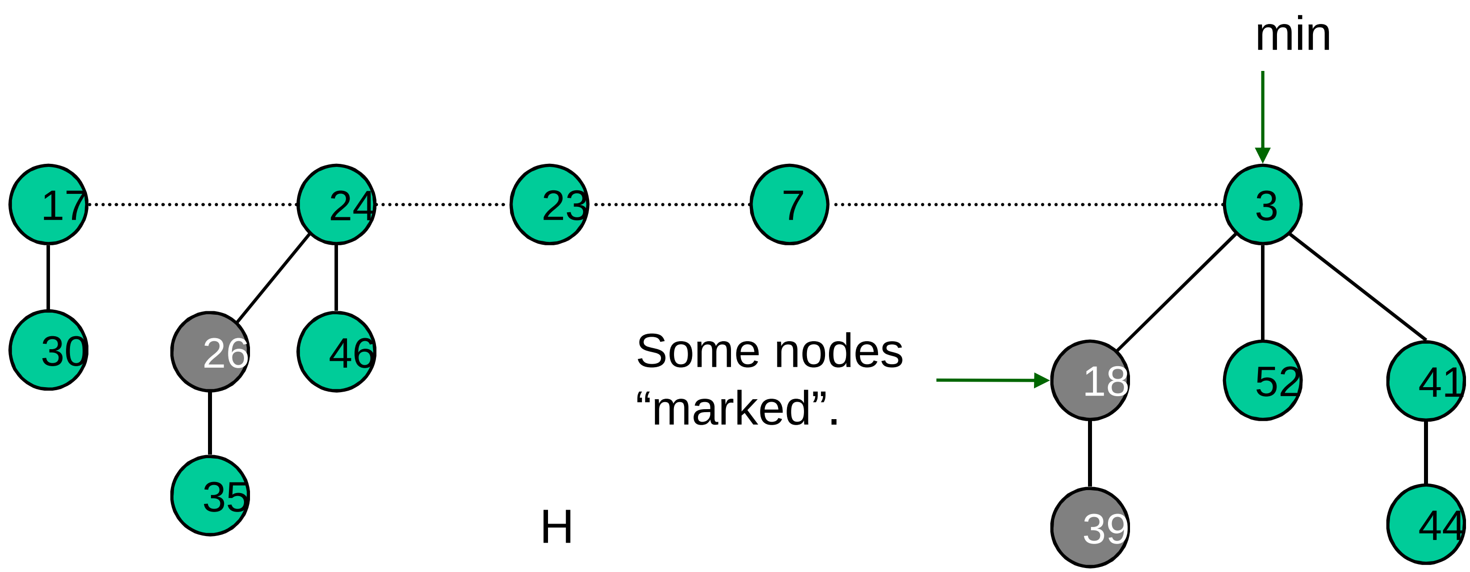
- Similar to binomial heaps, but less structured.
- "Lazy" unions.

Original motivation

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

Structure

Set of min-heap ordered trees



Implementation

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

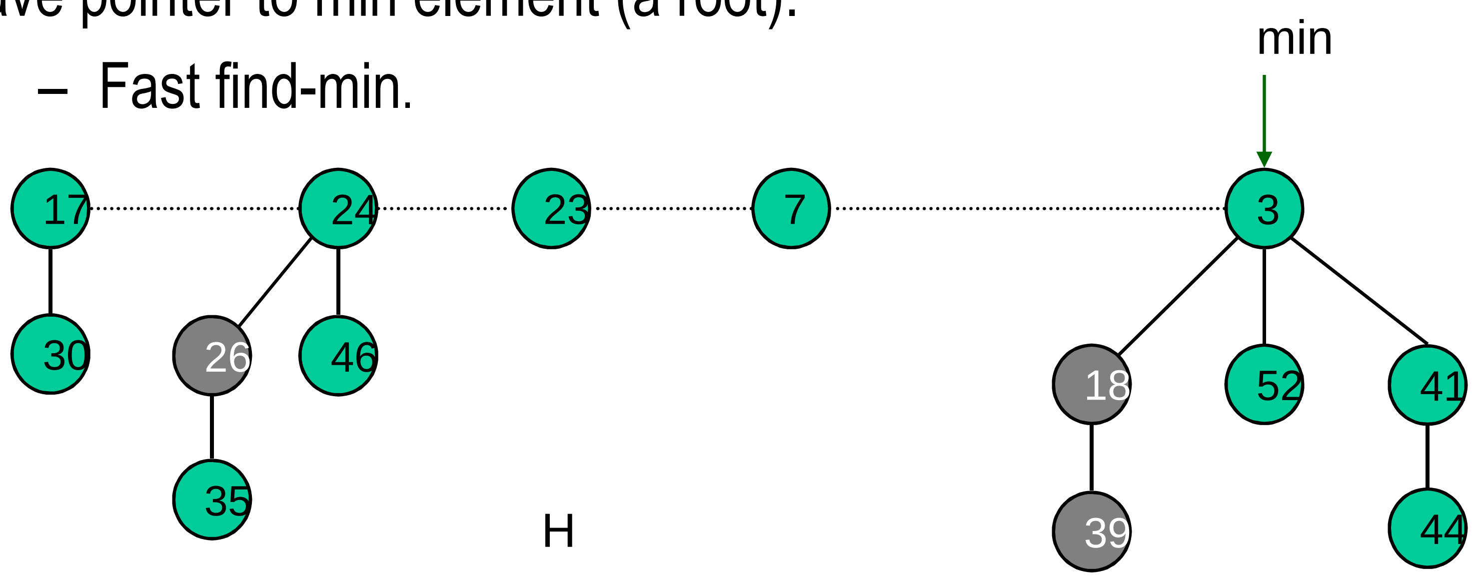
- Sibling pointers form circular, doubly-linked list.
- Can quickly splice off subtrees.

Roots in circular, doubly-linked list.

- Fast union.

Have pointer to min element (a root).

- Fast find-min.



Implementation

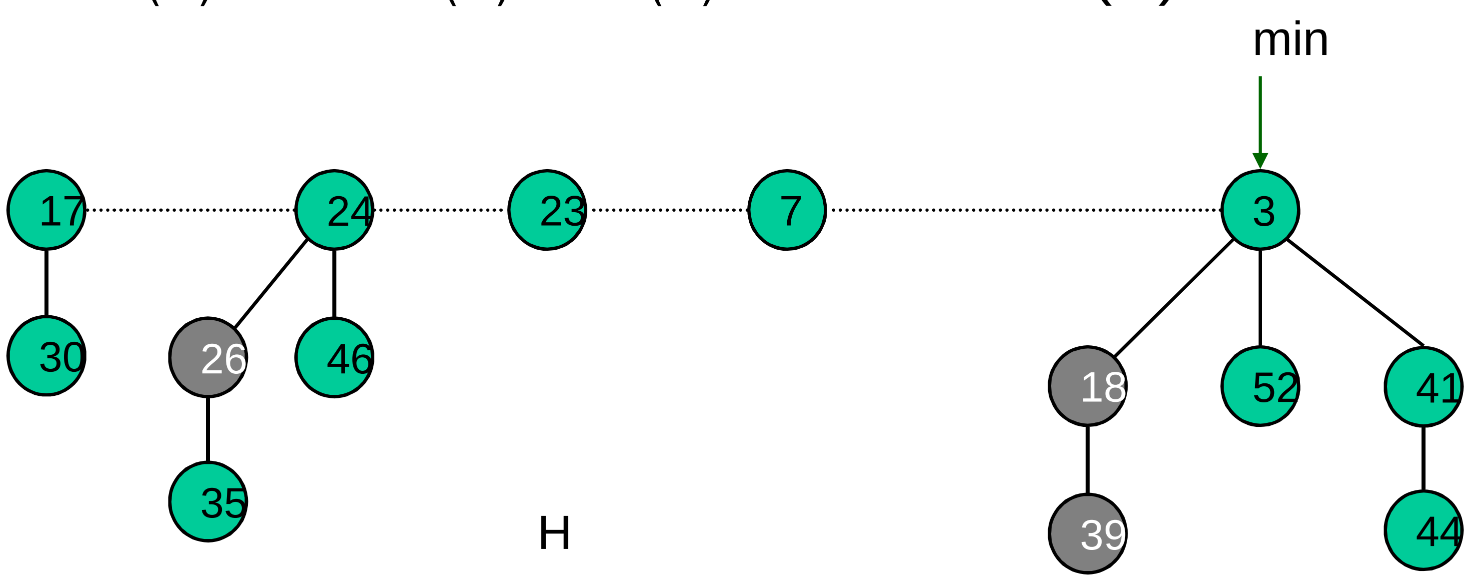
Key quantities:

- degree(x) = degree of node x.
- mark(x) = is node x marked?
- t(H) = # trees.
- m(H) = # marked nodes.
- Φ(H) = t(H) + 2m(H)

t(H) = ? 5

m(H) = ? 3

Φ(H) = ? 11



Insert

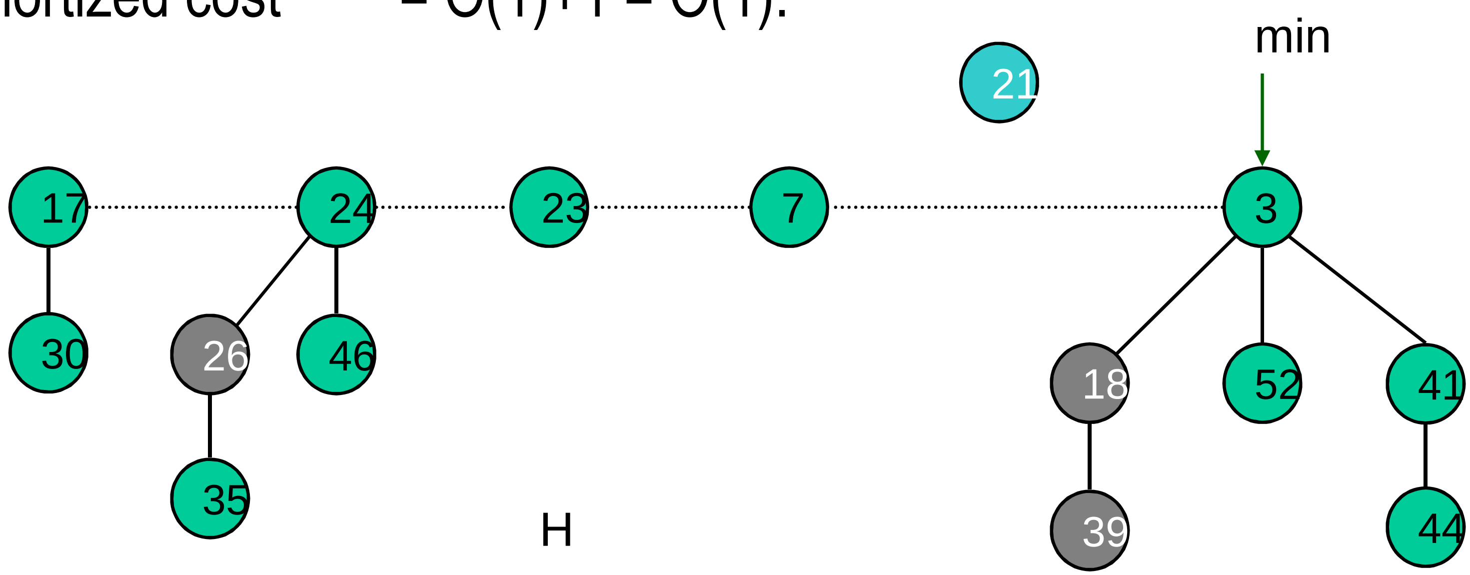
1. Create a new singleton tree.
2. Add to left of min pointer.
3. Update min pointer.

Actual cost = O(1)

Change in potential = ? 1

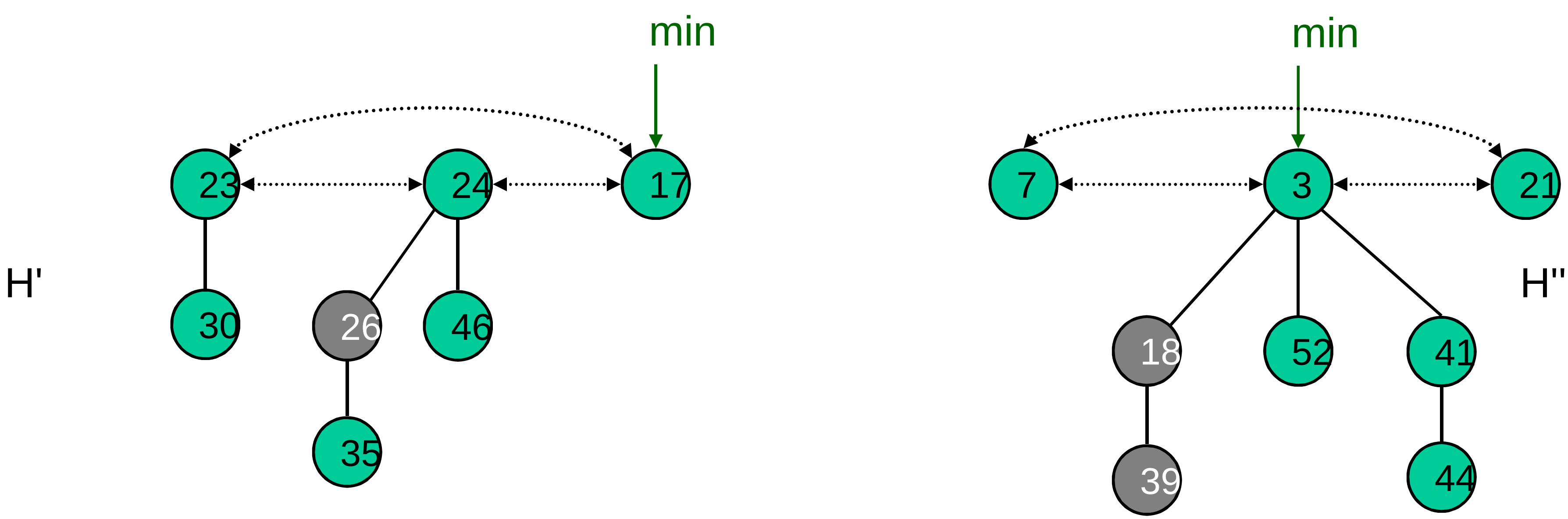
Amortized cost = O(1)+1 = O(1).

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



Union

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



Union

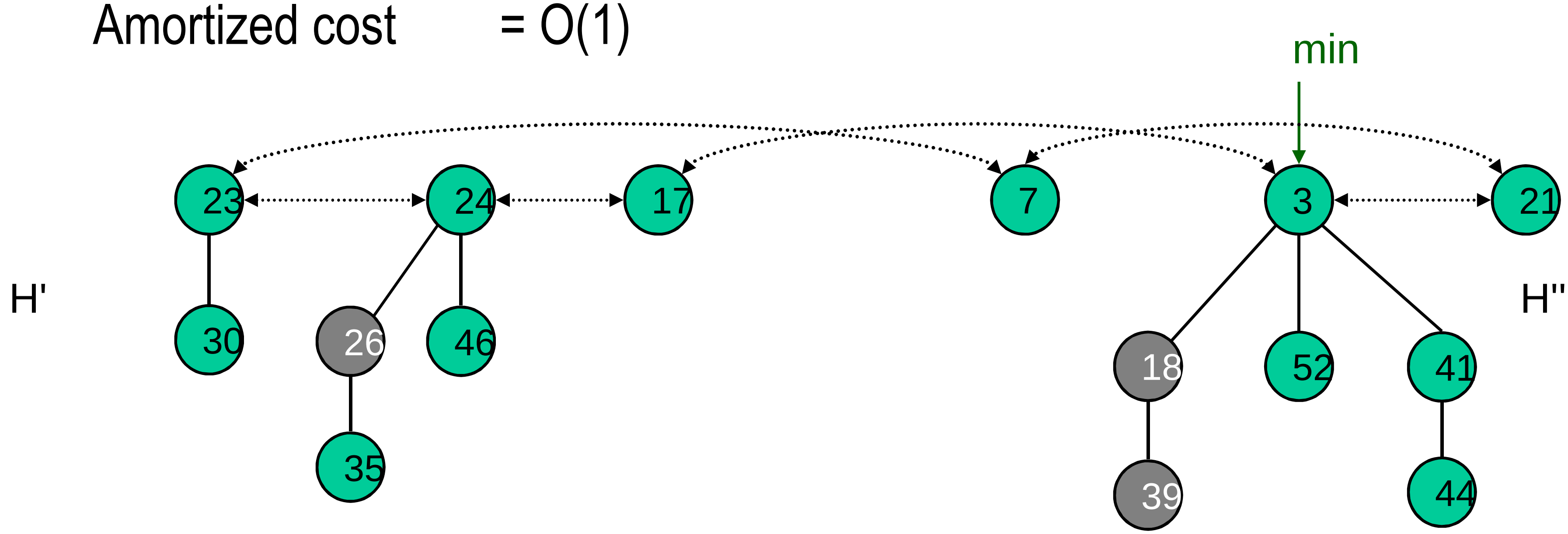
1. Concatenate heaps.
2. Keep pointer to the minimum of the two minimums.

Actual cost = O(1)

Change in potential = 0

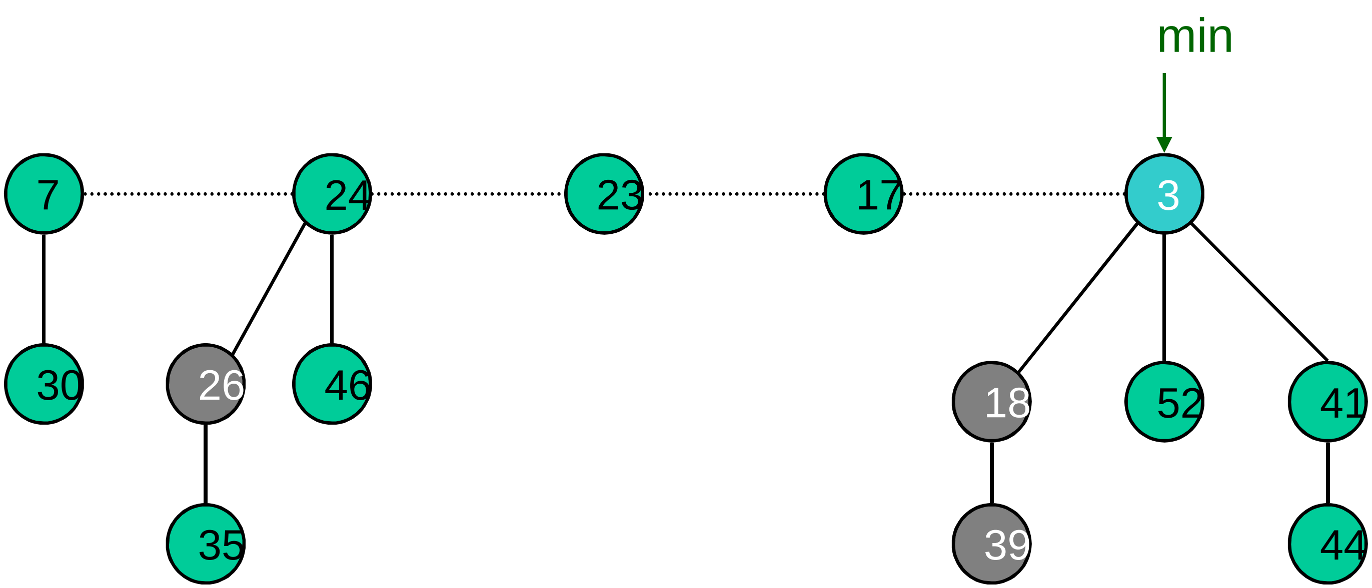
Amortized cost = O(1)

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



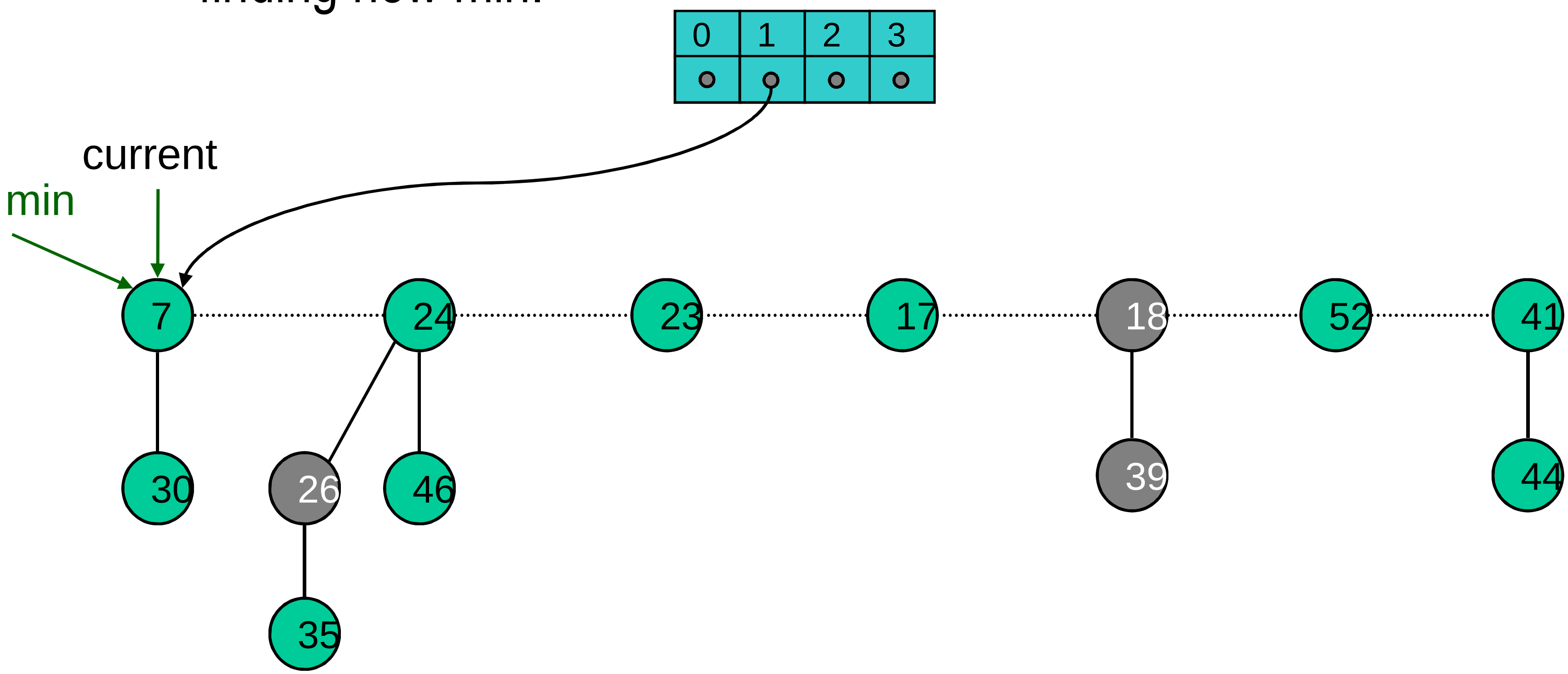
Delete Min

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



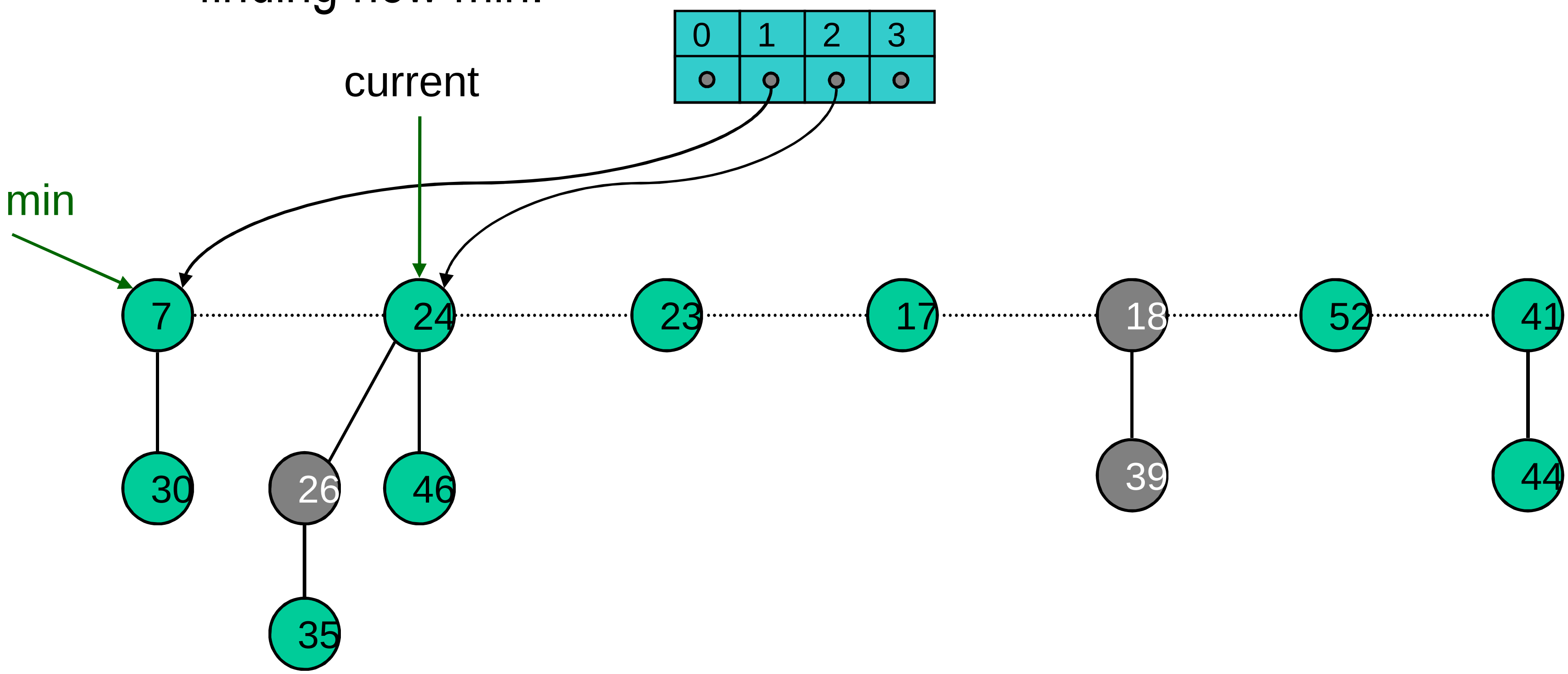
Delete Min

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



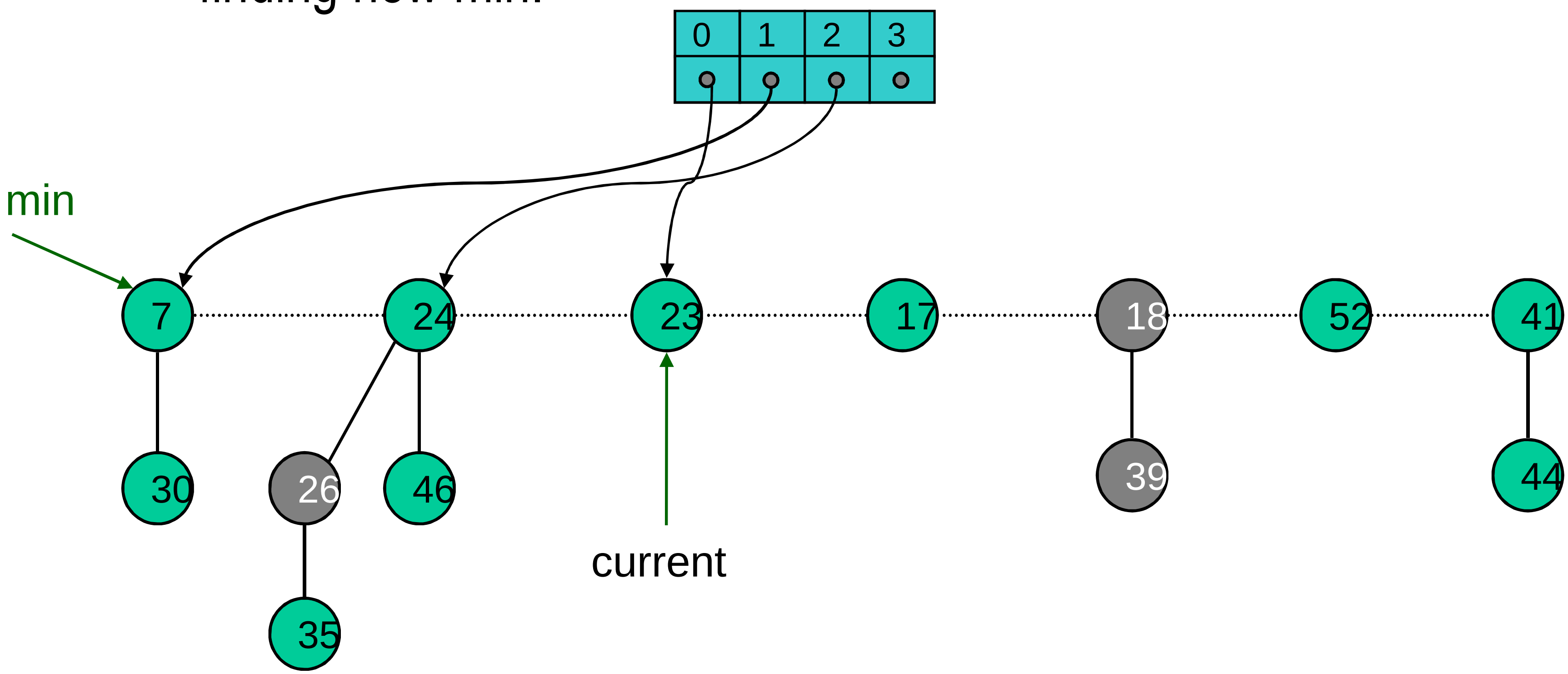
Delete Min

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



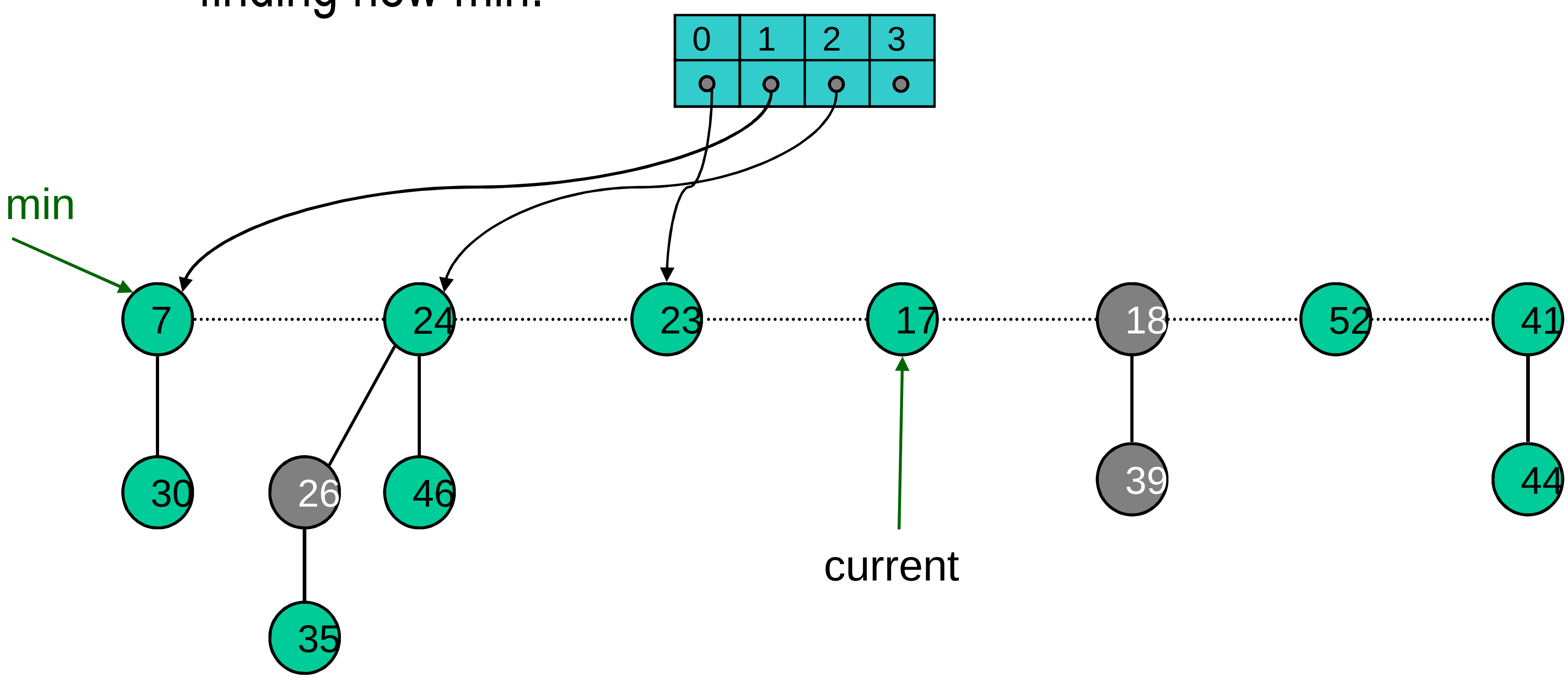
Delete Min

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



Delete Min

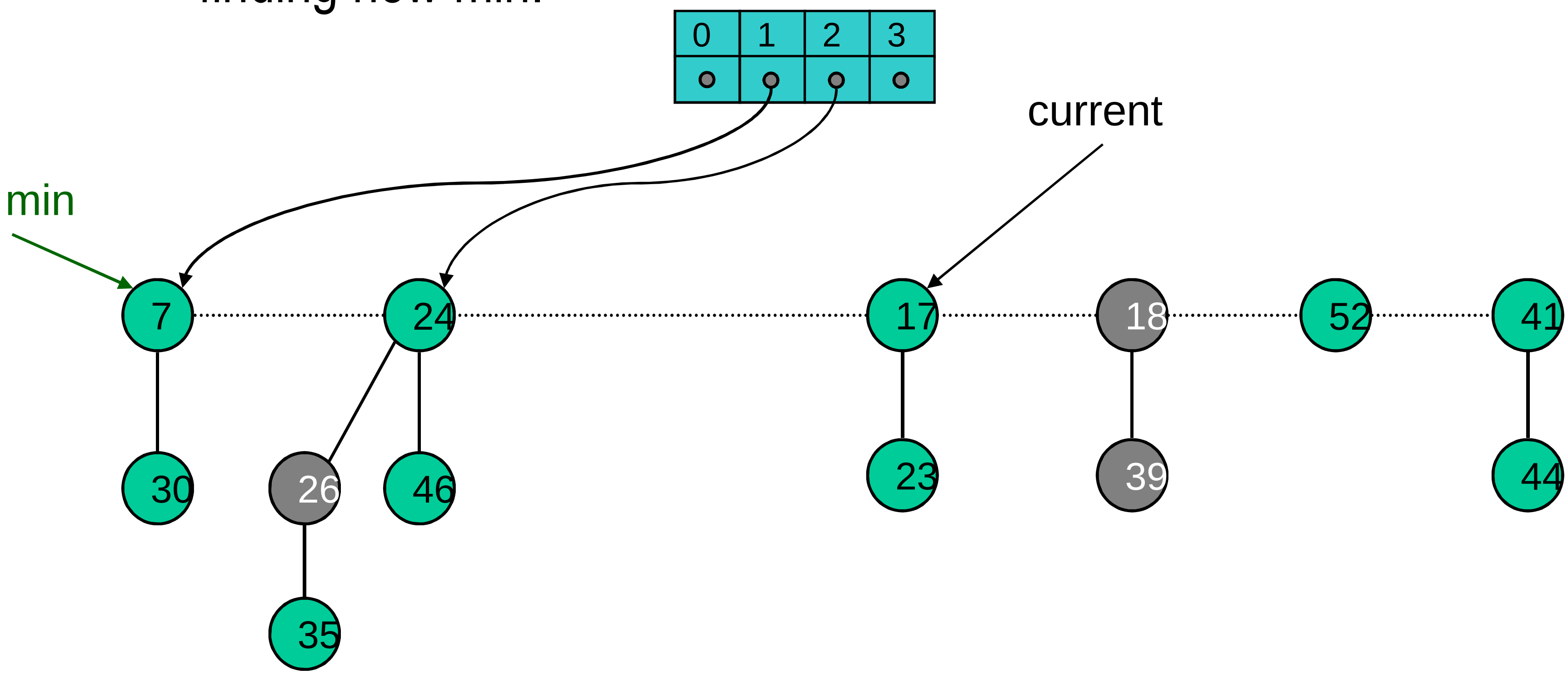
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 17 & 23 trees.

Delete Min

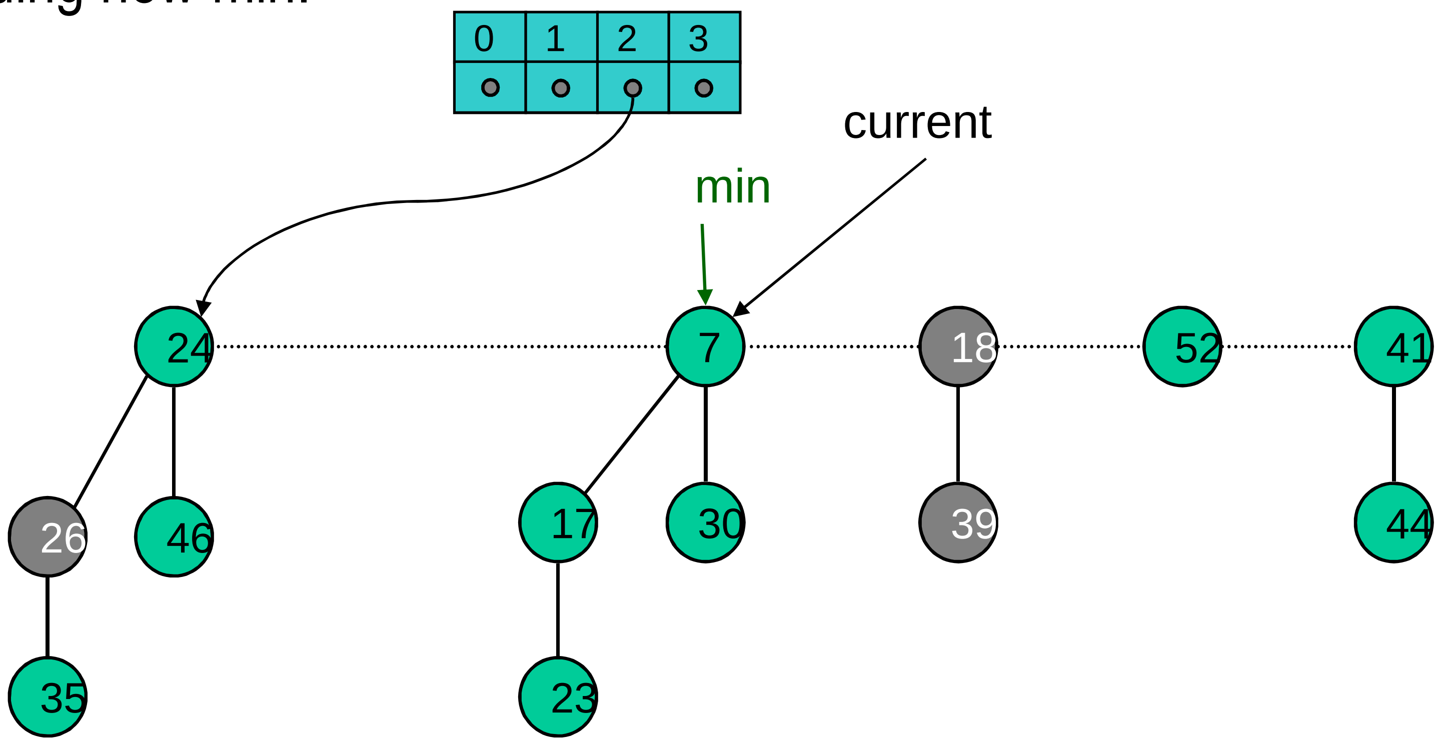
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 17 & 7 trees.

Delete Min

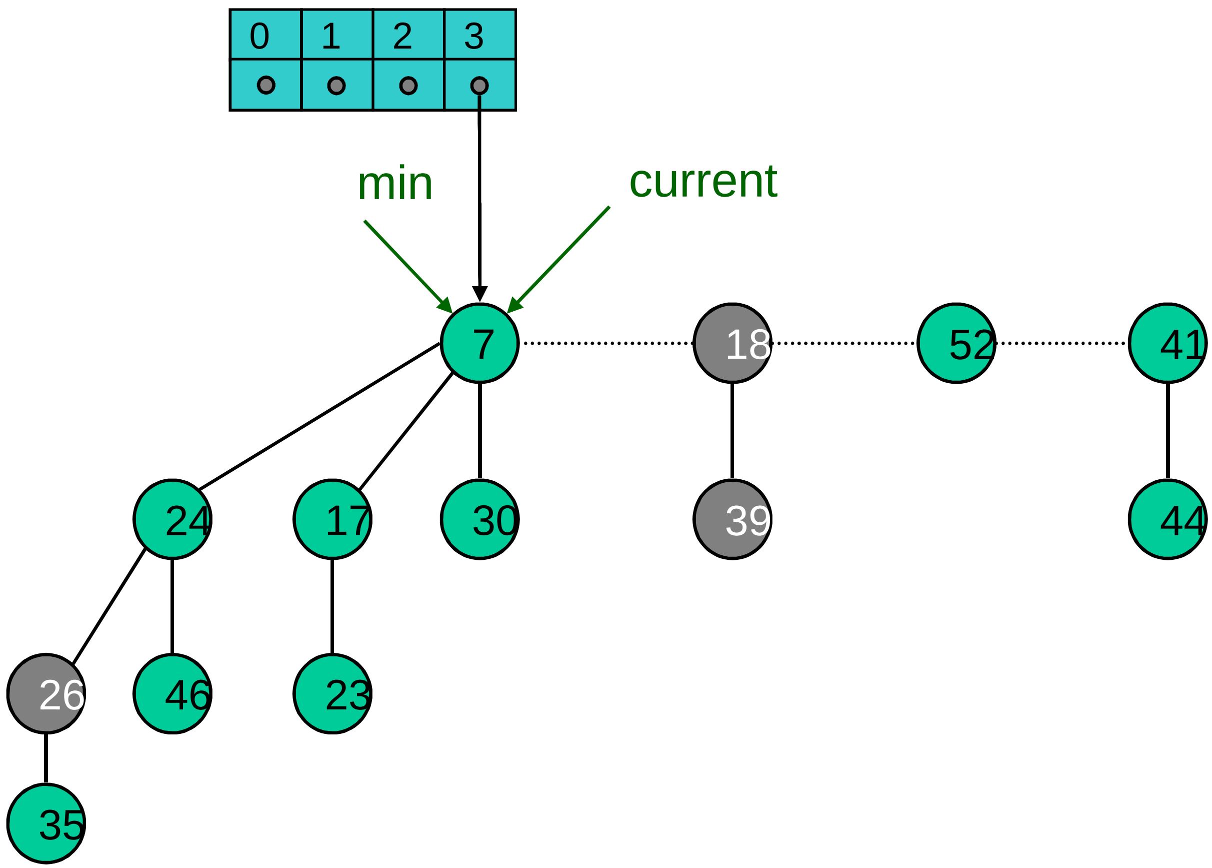
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 7 & 24 trees.

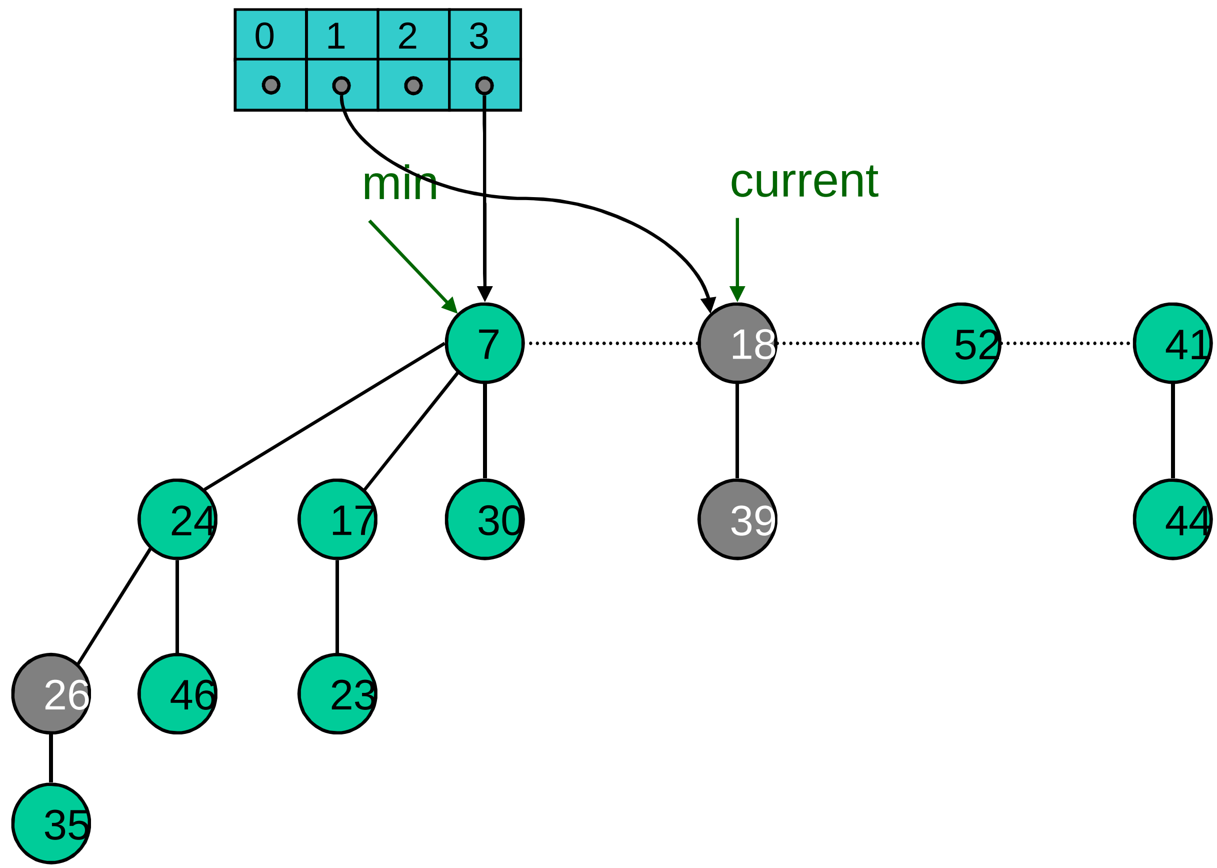
Delete Min

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



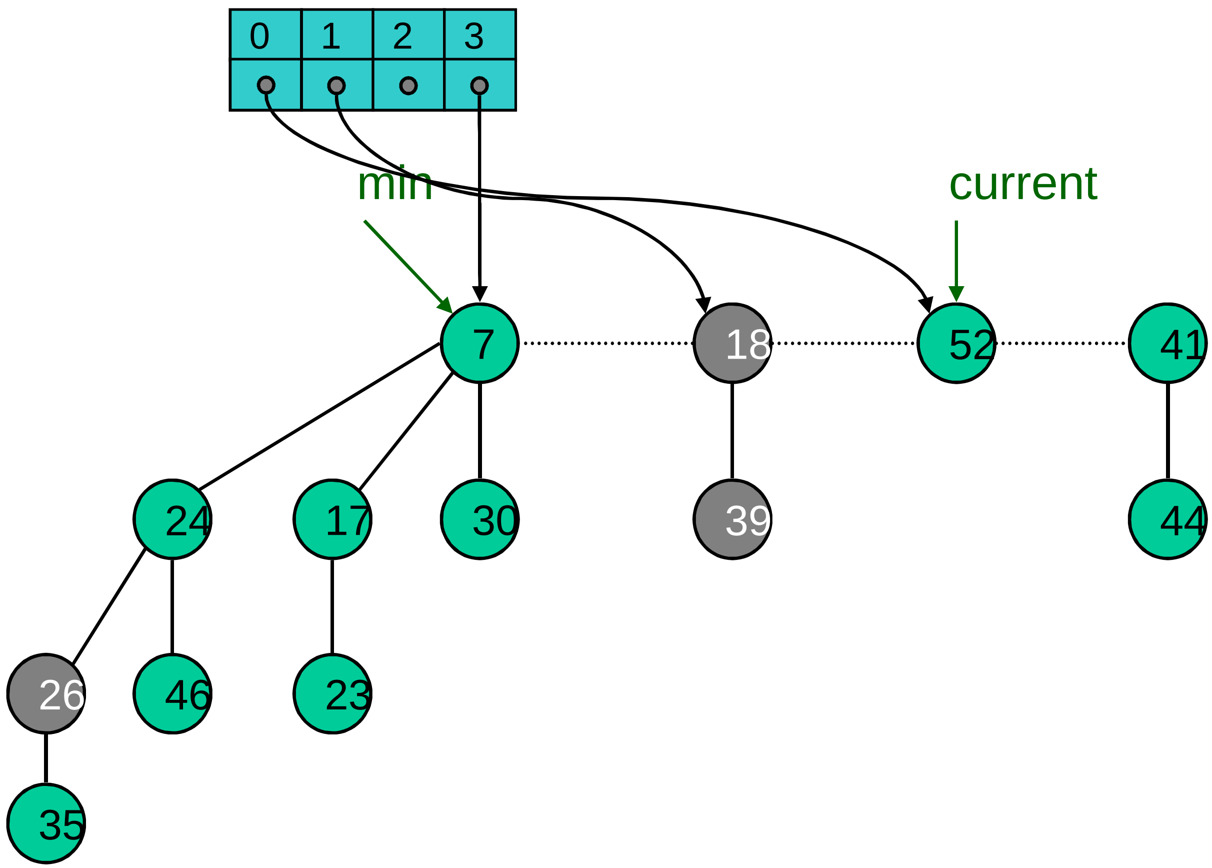
Delete Min

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



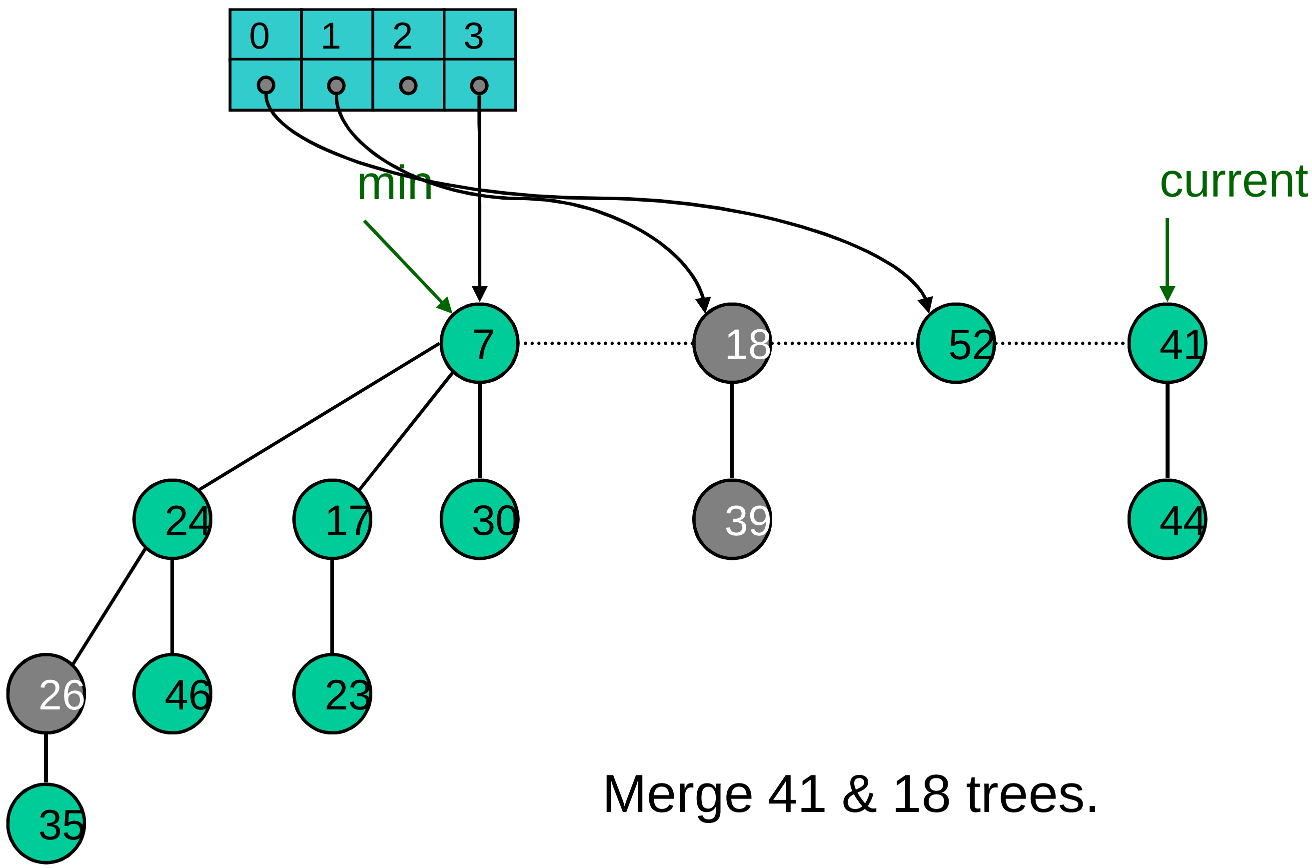
Delete Min

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



Delete Min

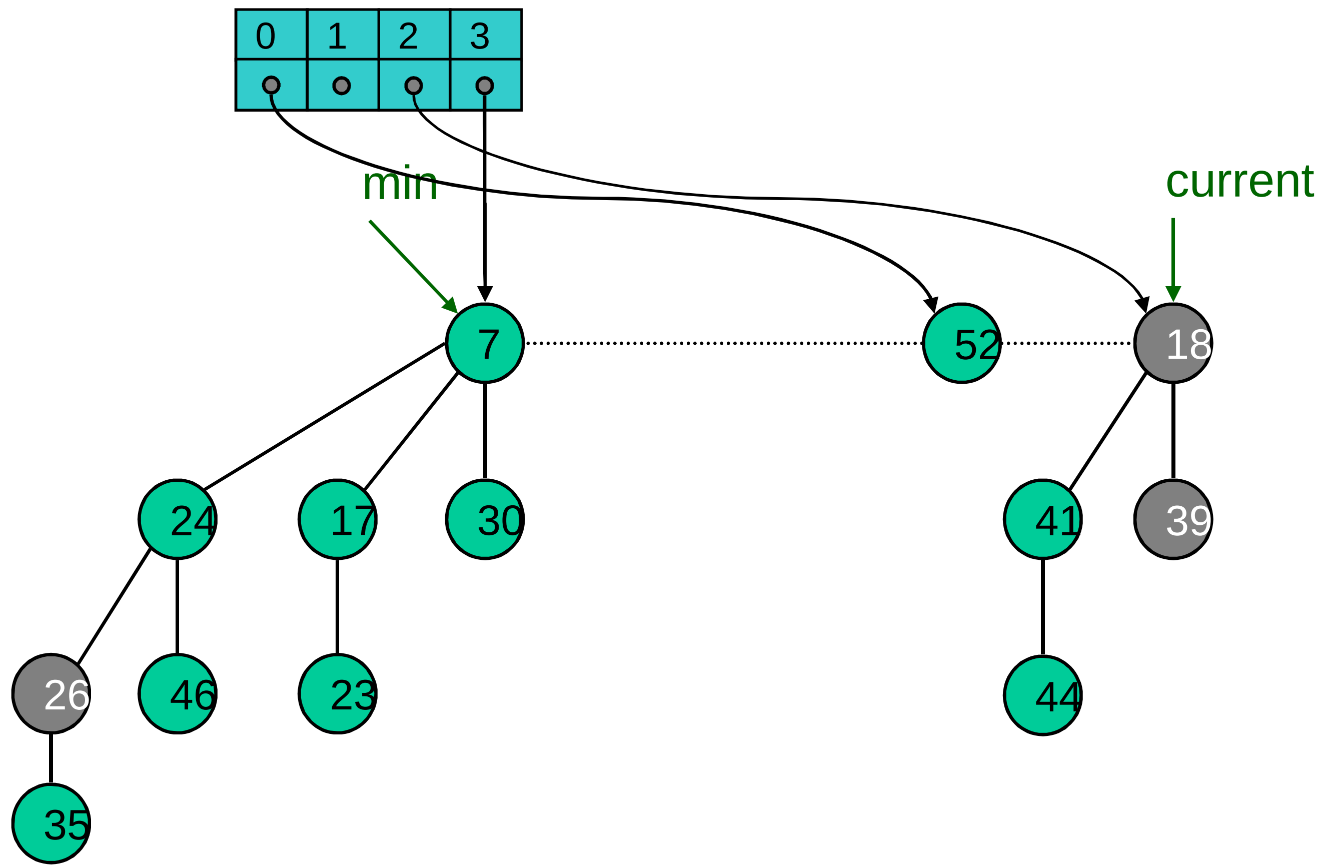
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 41 & 18 trees.

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.



Delete-Min Analysis

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

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

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

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.
- $m(H') \leq m(H)$
- $\Delta\Phi(H) \leq D(n) + 1 - t(H)$

Delete-Min Analysis

Theorem

The Delete-Min operation can be implemented to run in $O(D(n) + t(n))$ actual time, and $O(D(n))$ amortized time

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.
- $D(n) \leq \lfloor \log_2 N \rfloor$

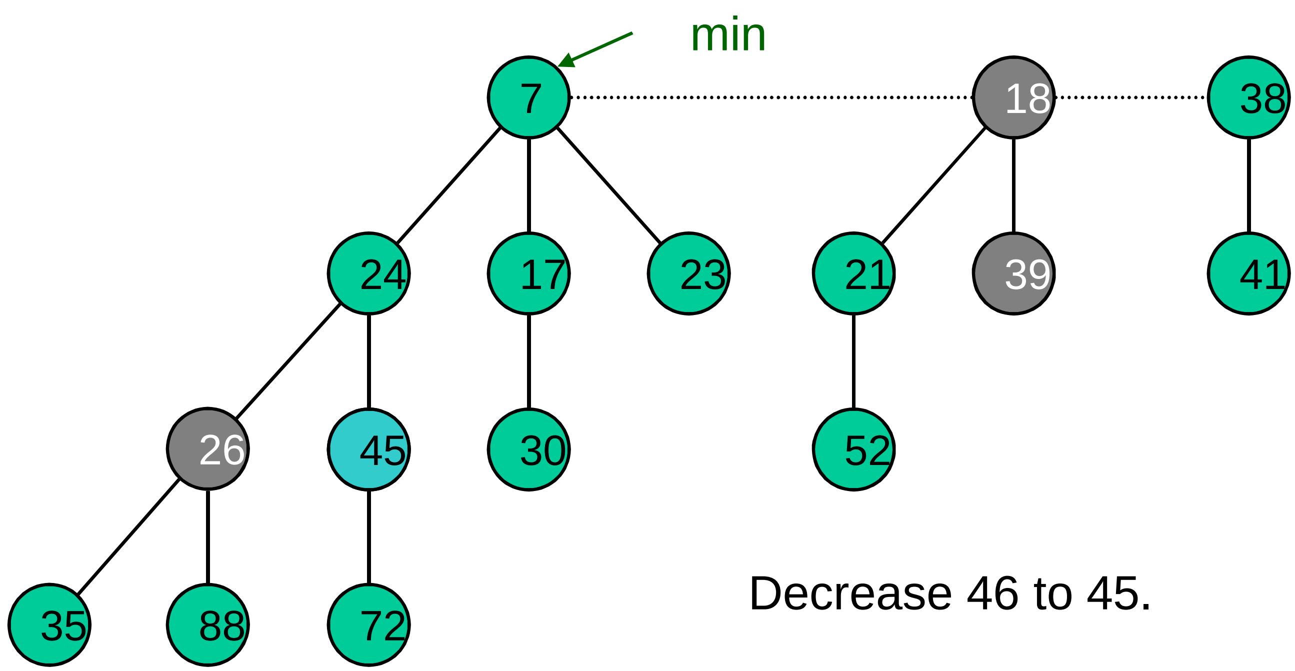
Yes, if we support Decrease-key cleverly.

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

Decrease Key

Case 0: min-heap property not violated.

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

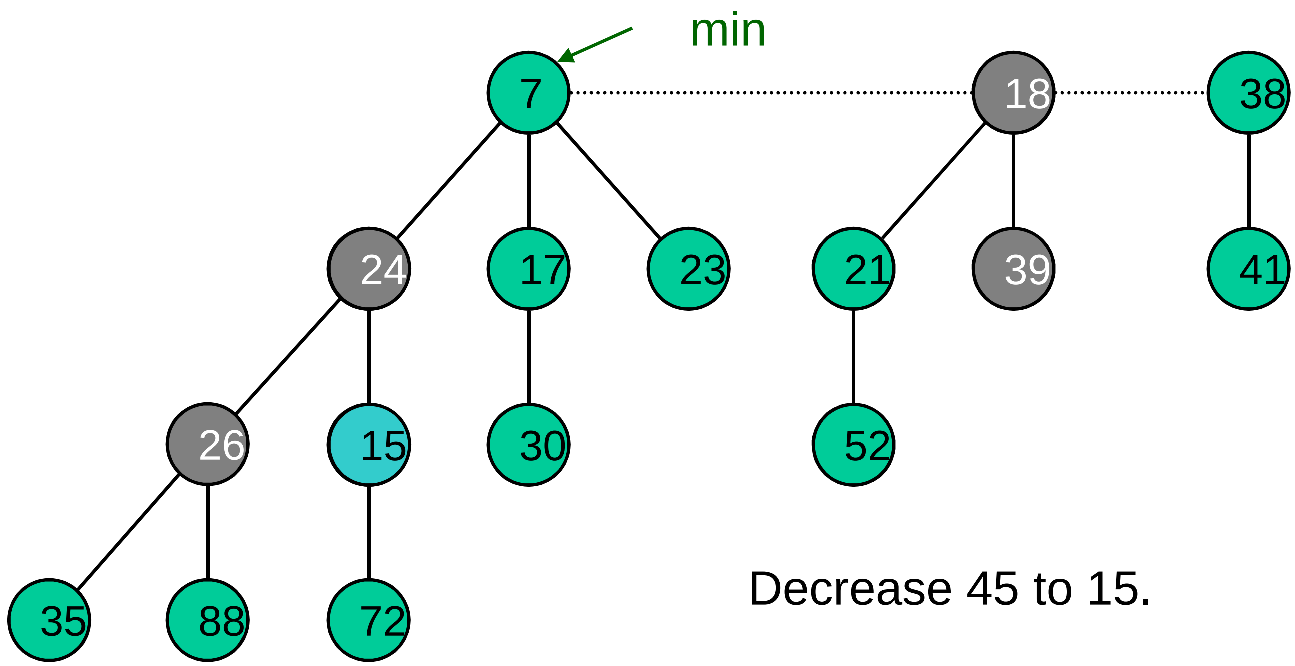


Decrease 46 to 45.

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.

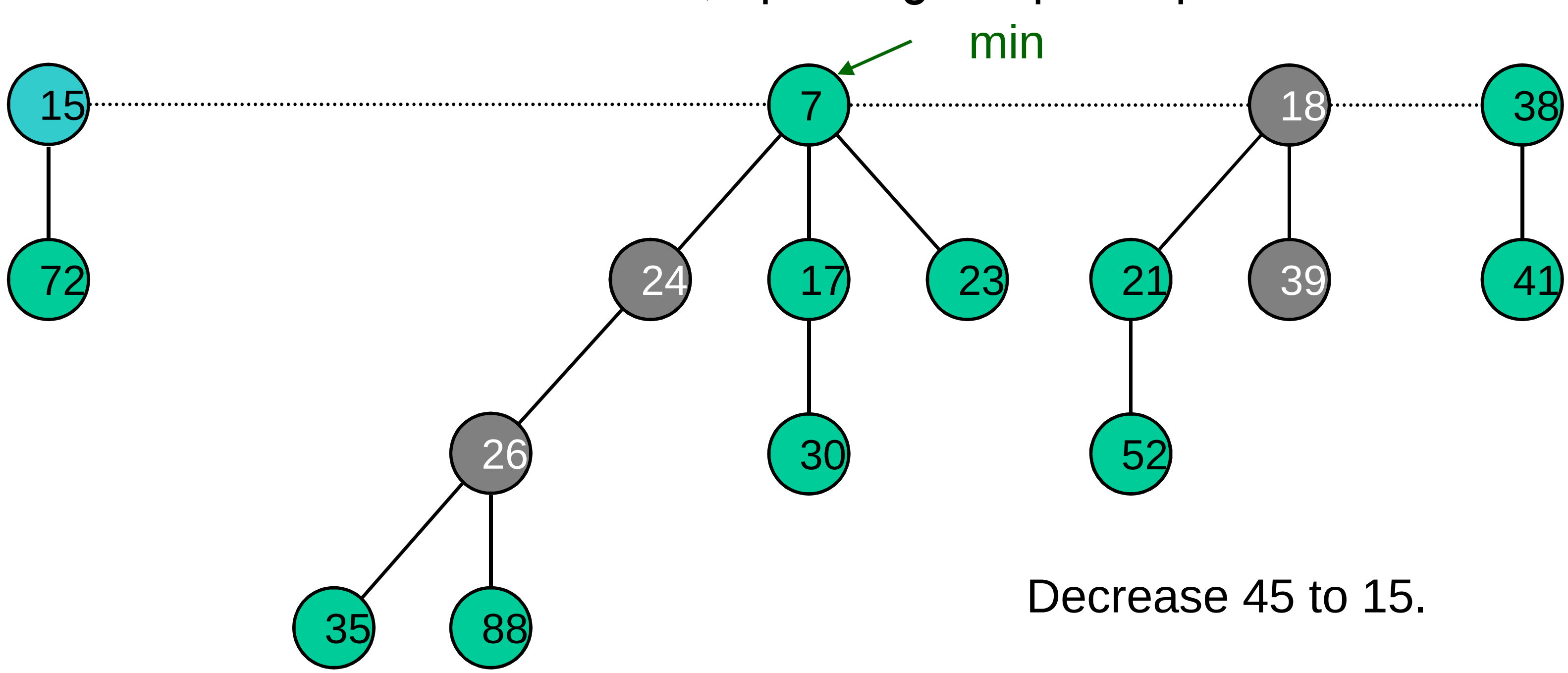


Decrease 45 to 15.

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.

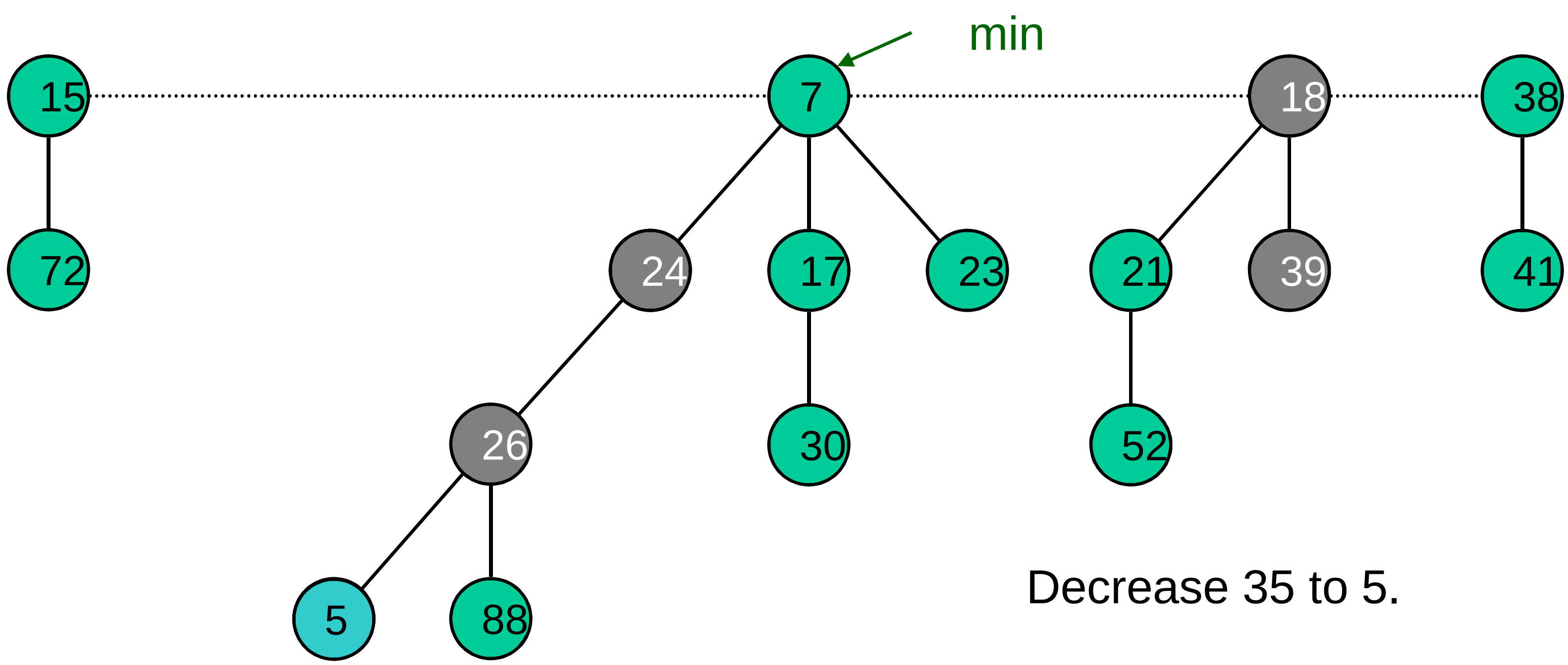


Decrease 45 to 15.

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.

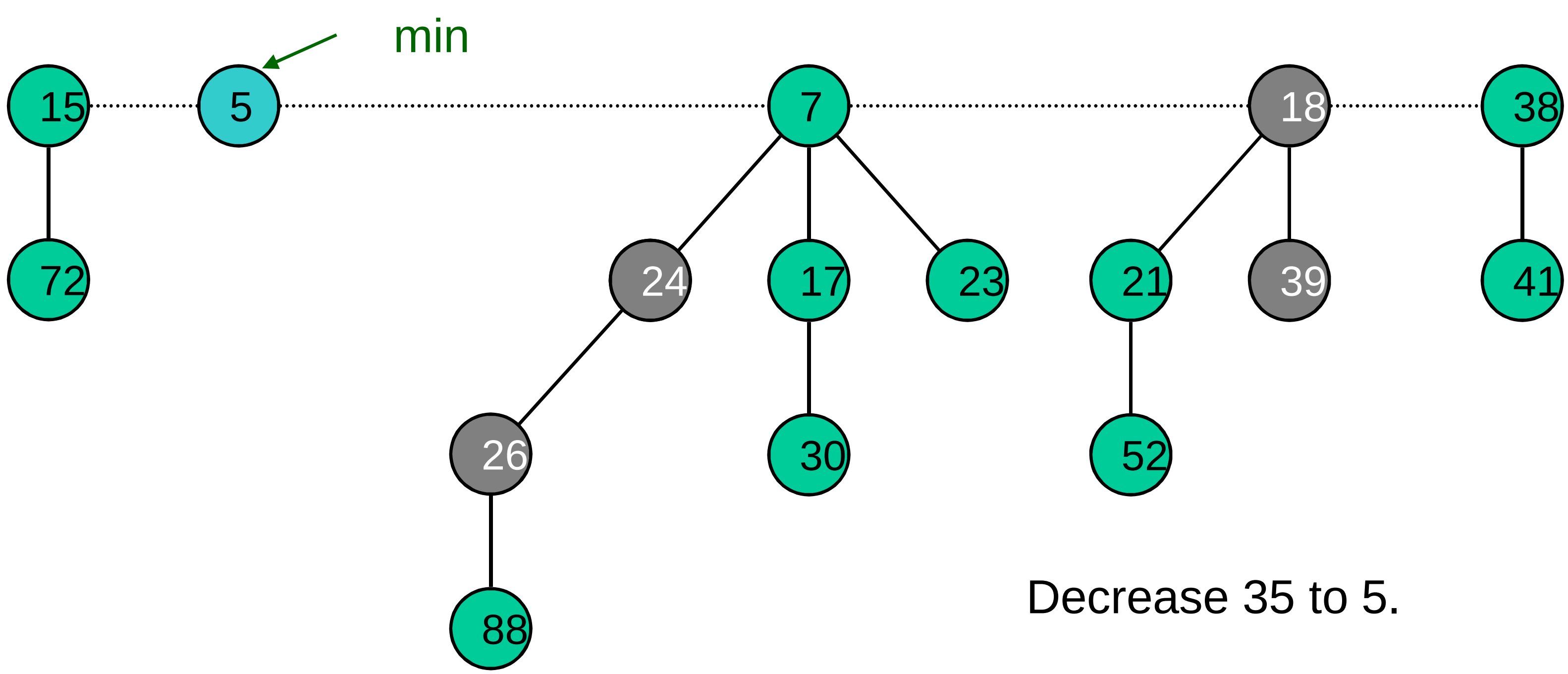


Decrease 35 to 5.

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.

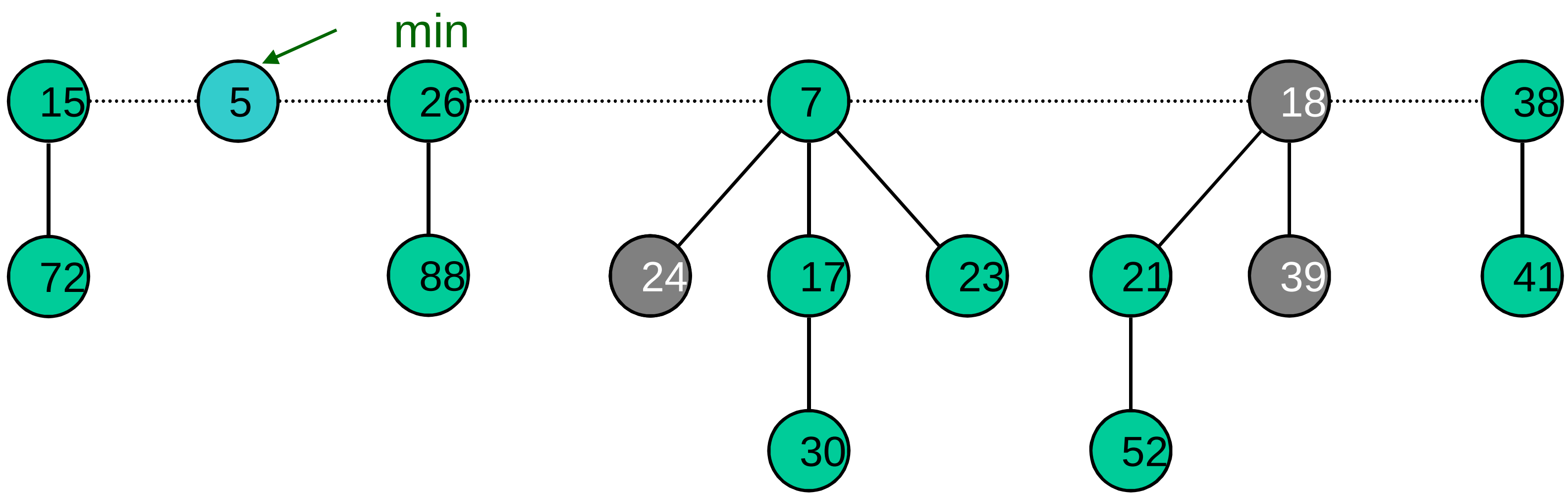


Decrease 35 to 5.

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.

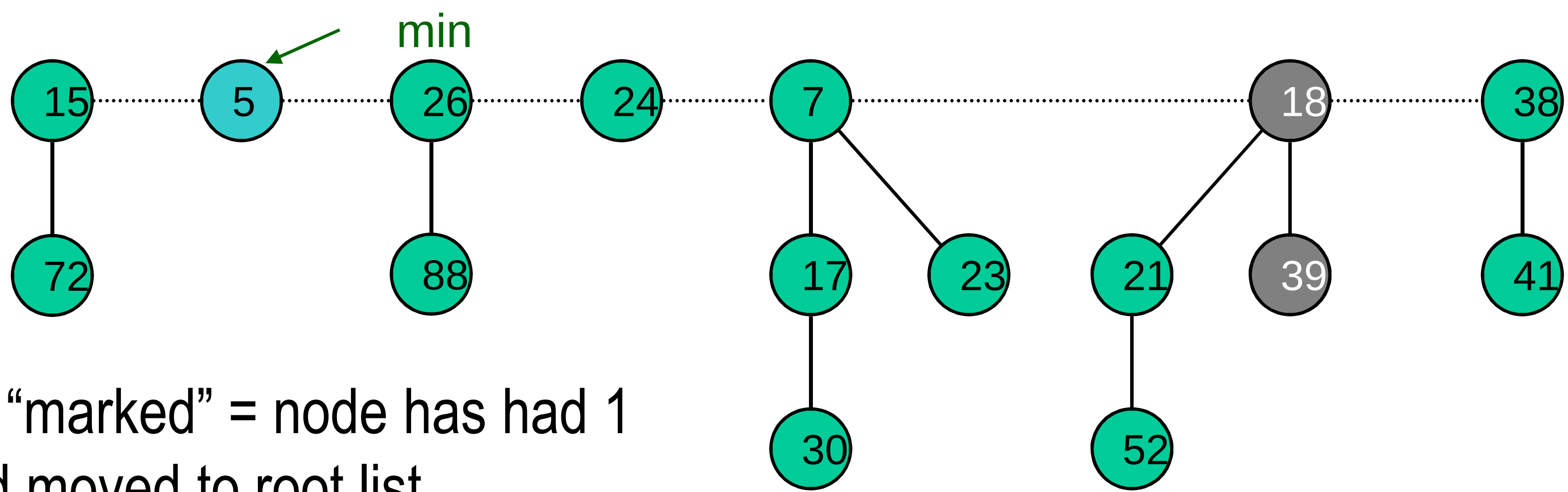


Decrease 35 to 5.

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 2^{nd} child of node, we also move the node.

Decrease 35 to 5.

Decrease-Key Analysis

- $t(H)$ = # 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.
- $\Delta\Phi(H) \leq c + 2(-c + 2) = 4 - c$.

Decrease-Key Analysis

Theorem

The Decrease-Key operation can be implemented to run in $O(c)$ actual time, where c is the number of nodes "cut", and in $O(1)$ amortized time

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.

Bounding D(n)

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

Theorem

$D(n) \leq \log_\phi n$, 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 size(x) = #nodes in the subtree rooted at x.

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