Ve 280

Programming and Introductory Data Structures

Fibonacci Heap

Midterm Exam

- Time: Oct. 25th, in class.
- Location: see Canvas announcement.
- A written exam.
  - Like our written assignments.
  - Pseudo-code OK (but make sure we can **understand** it!)
- Closed book and closed notes.
- Only basic calculator is allowed.
  - No other electronic devices, including laptops and cell phones.
  - We will show a clock on the screen.
- Abide by the **Honor Code**!

Midterm Topics

- Asymptotic Algorithm Analysis
- Sorting
  - Comparison Sort
  - Non-comparison Sort
- Linear-time Selection
- Hashing (universal hashing not required)
- Bloom Filter
- Tree and Binary Tree Traversal
- Priority Queue and Binary Heap

Fibonacci Heap

- A **mergeable heap,** which supports the following operations
  - **`insert`**: insert a new item into the heap
  - **`getMin`**: get item with **min** key
  - **`extractMin`**: remove and return an item with **min** key
  - **`makeHeap`**: create a new empty heap
  - **union(H_1,H_2)** : create and return a new heap that contains all the elements of heaps H_1 and H_2. Heaps H_1 and H_2 are destroyed by this operation
- Additionally, Fibonacci heap supports
  - **`decreaseKey(Node x, Key k)`** : decrease the key of node x to a smaller value k and restore the heap property

Runtime Complexity Comparison

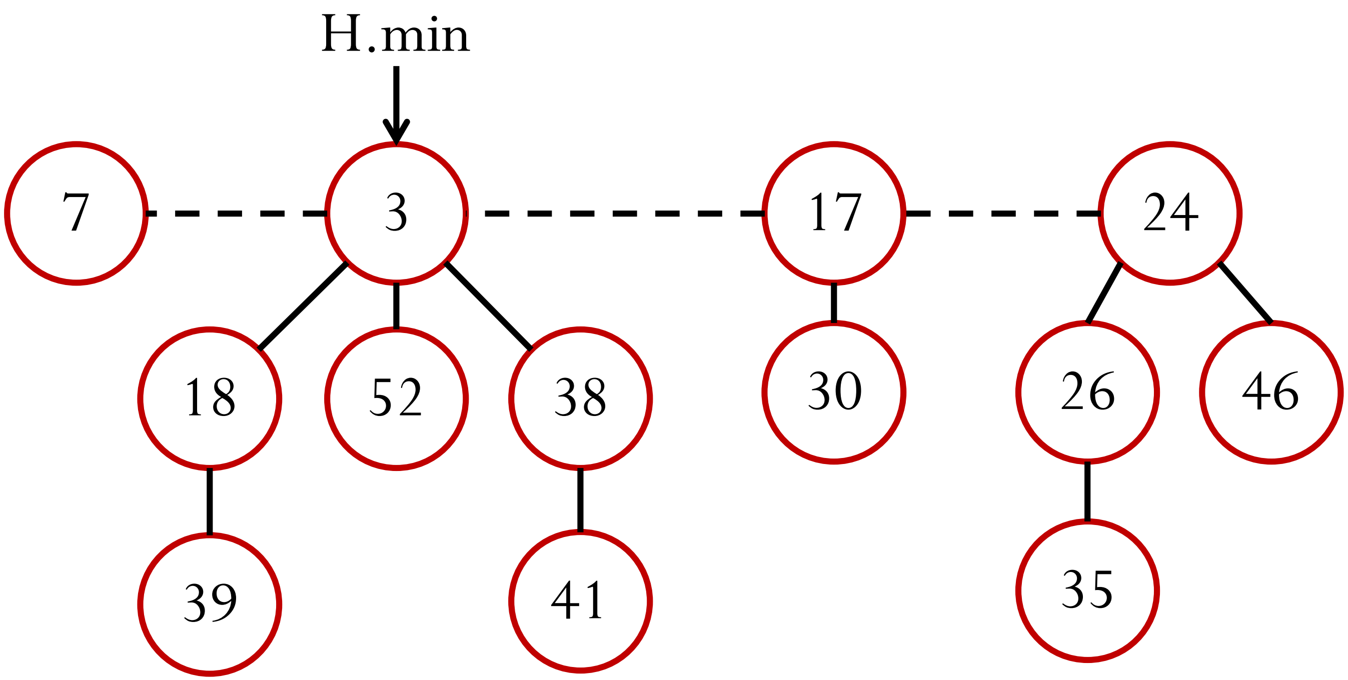
Operation	Binary Heap (worst case)	Fibonacci Heap (amortized analysis)
insert	$\Theta(\log n)$	$\Theta(1)$
extractMin	$\Theta(\log n)$	$O(\log n)$
getMin	$\Theta(1)$	$\Theta(1)$
makeHeap	$\Theta(1)$	$\Theta(1)$
union	$\Theta(n)$	$\Theta(1)$
decreaseKey	$\Theta(\log n)$	$\Theta(1)$

Application

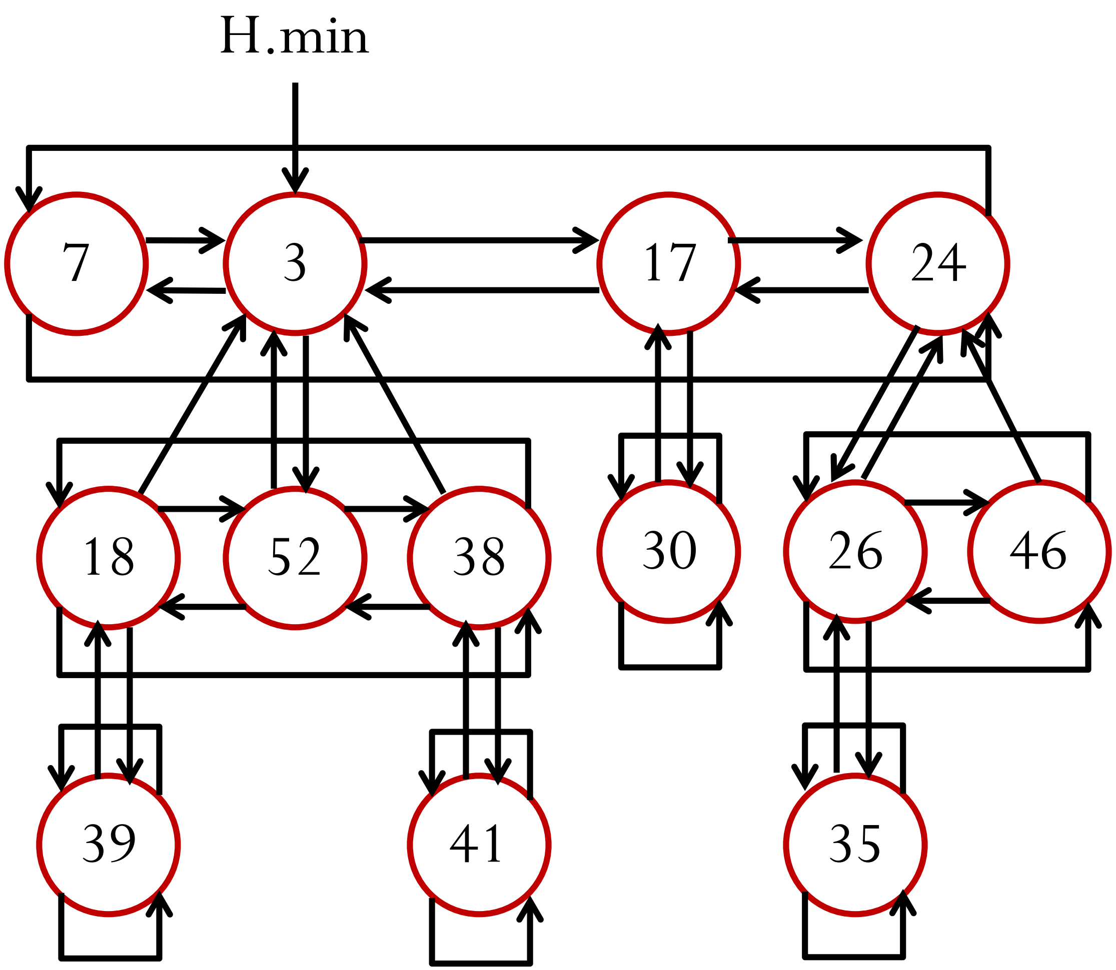
- Fast algorithms for problems such as computing minimum spanning trees and finding single-source shortest paths make essential use of Fibonacci heaps
  - For example, in single-source shortest path problem, we need to extract minimum and decrease key.

Fibonacci Heap: First Look

- A collection of rooted trees, each as a min-heap

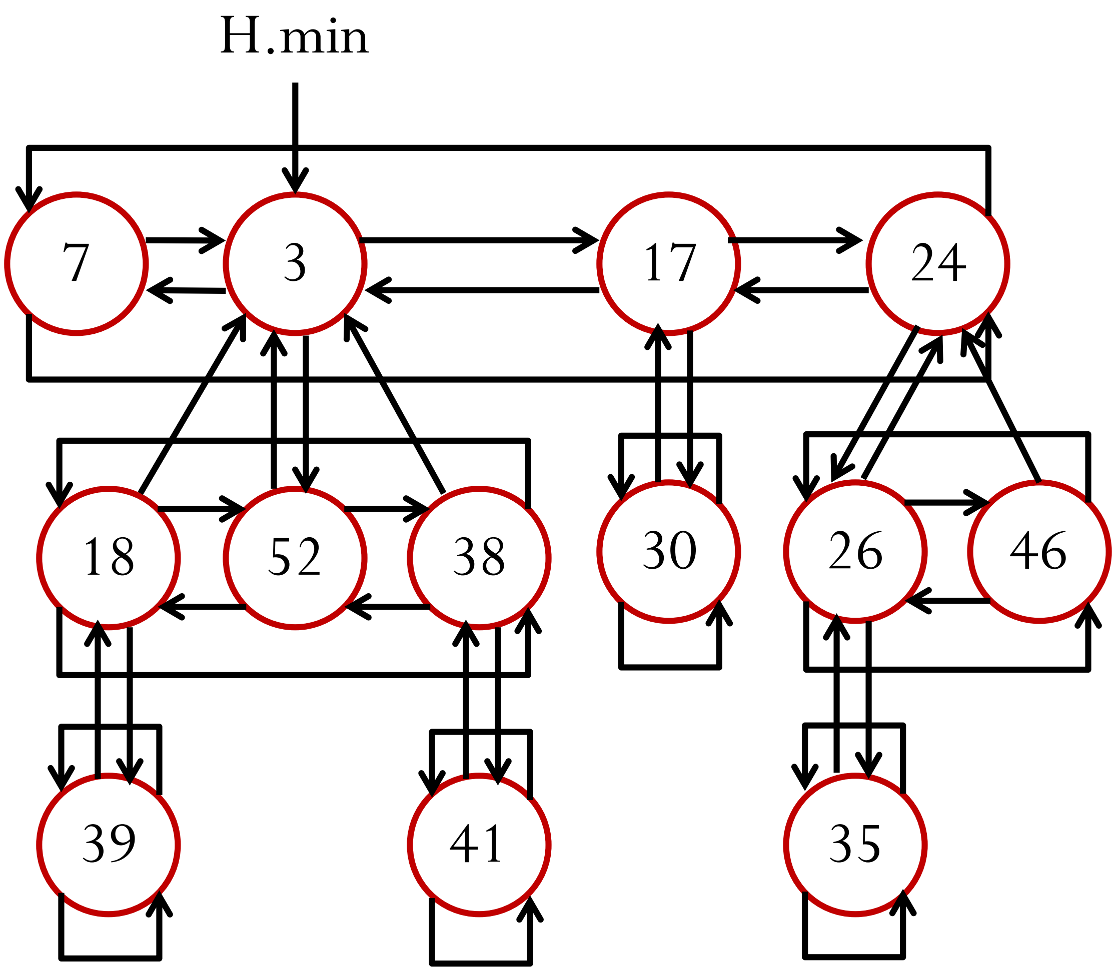


Fibonacci Heap: Implementation Details



- Each node has
  - a pointer to its parent
  - a pointer to one of its children
  - degree (# of children)
- Children are linked by circular, doubly linked list
  - If y is the only child, then `y.prev=y.next=y`
  - Why circular, doubly linked list? O(1) for node insertion, node removal, and list concatenation

Fibonacci Heap: Implementation Details

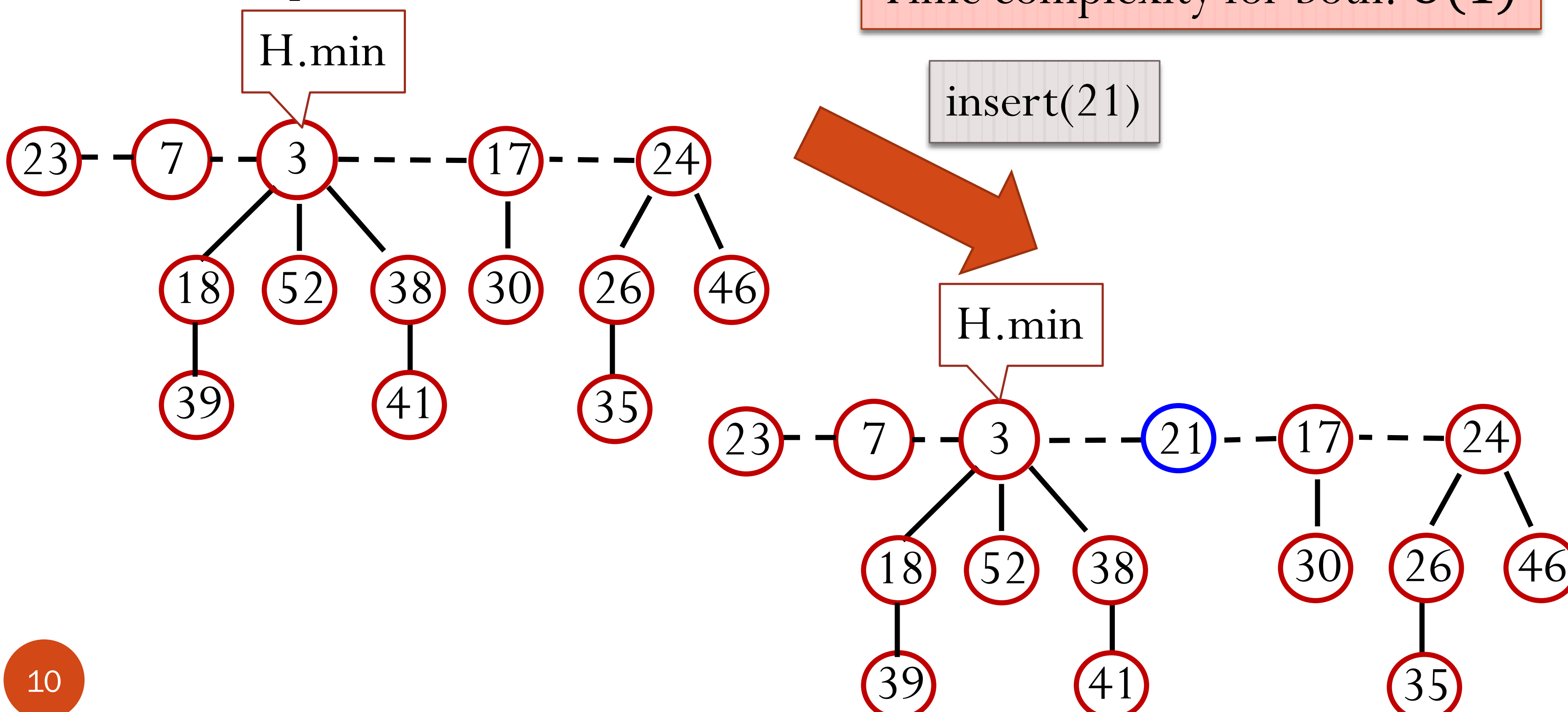


- Roots of the trees are also connected as a circular, doubly linked list
  - called **root list**
- `H.min` points to the minimum root
- `H.n` stores the number of nodes in `H`

Implementation of Some Operations

- **`makeHeap`**: set H.min = NULL and H.n = 0
- **`insert`**: Simply put the node into the **root list**
  - Update the H.min if needed

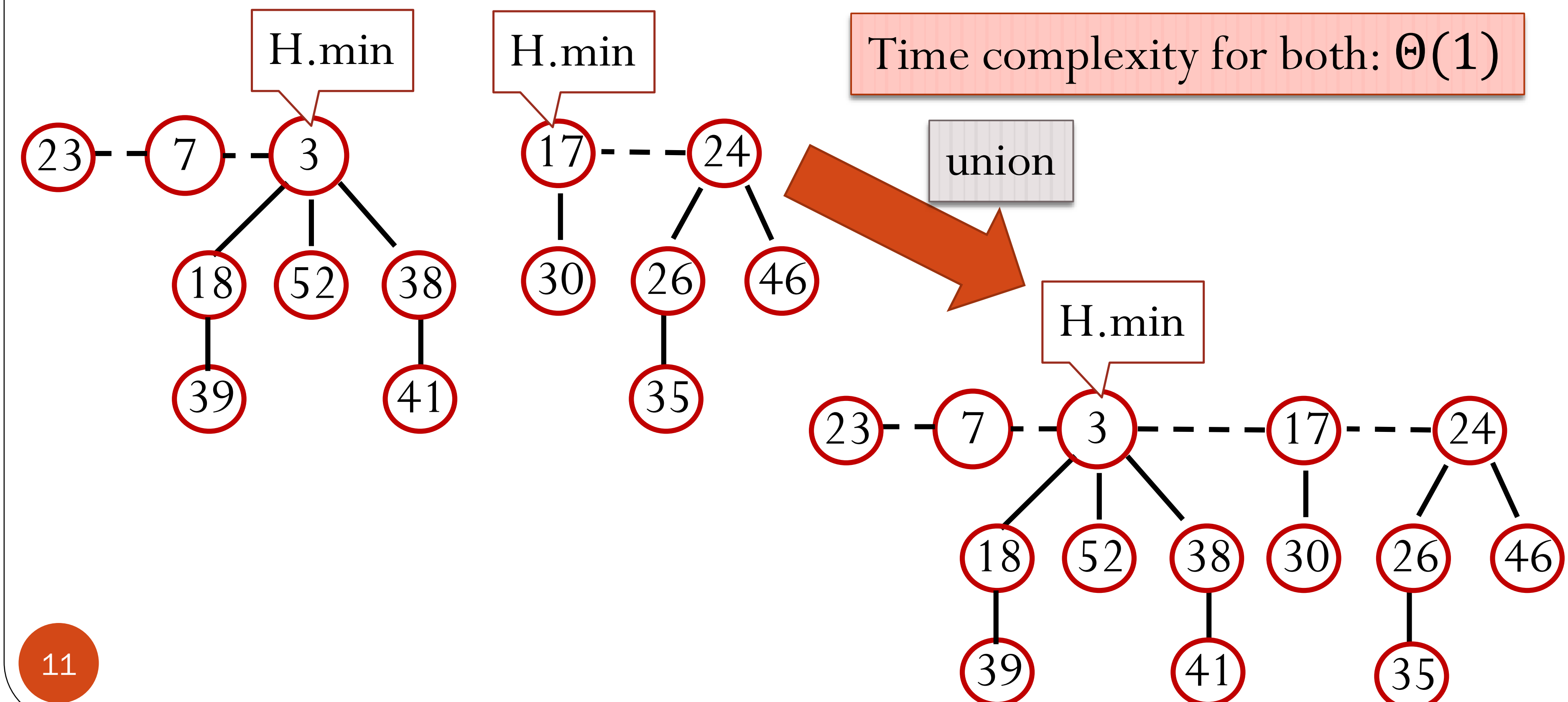
Time complexity for both: $\Theta(1)$



Implementation of Some Operations

- **getMin**: return H.min
- **union**(H_1, H_2): concatenate the root lists of H_1 and H_2 and then determine the new minimum node

Time complexity for both: $\Theta(1)$

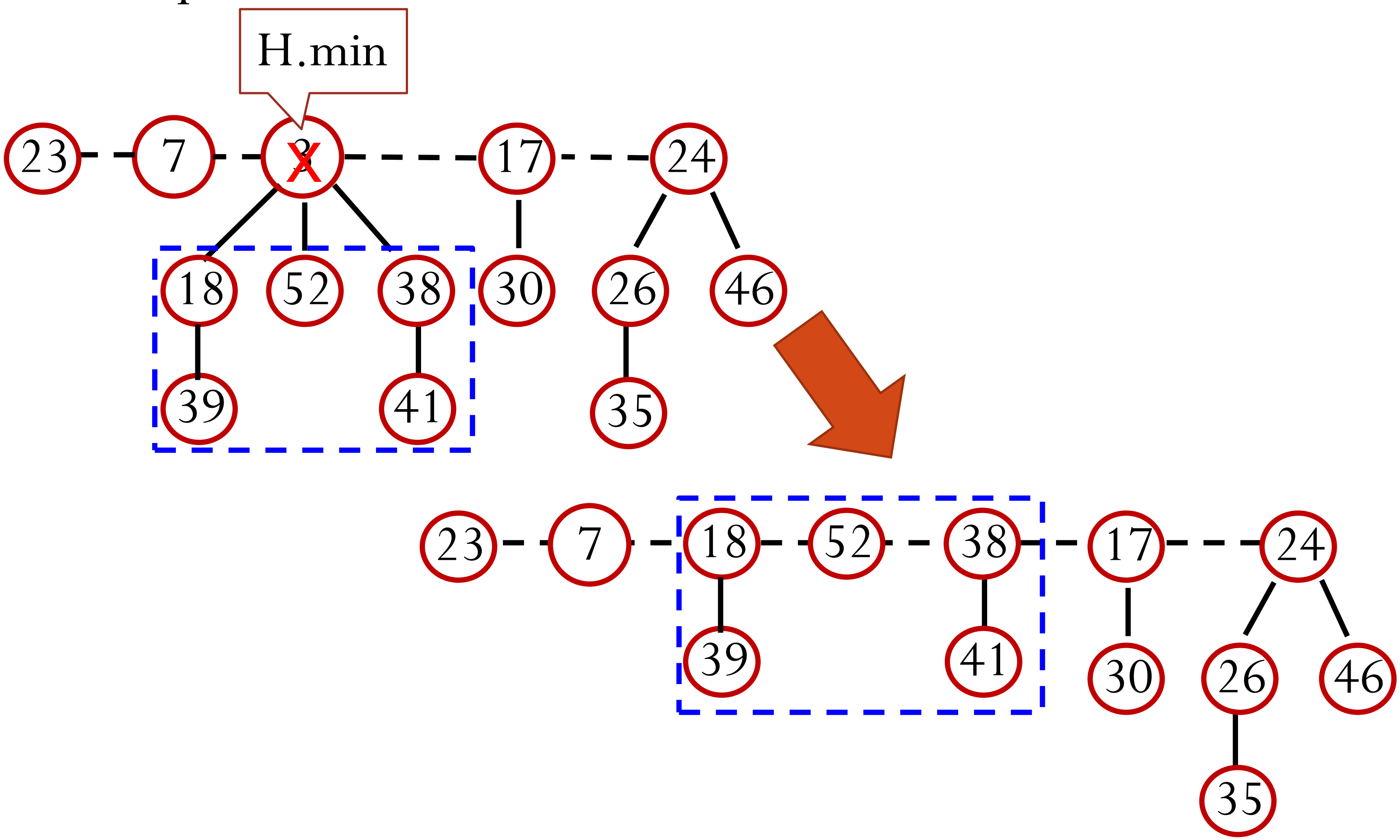


insert and extractMin

- If we start with an empty Fibonacci heap and then insert k items, the Fibonacci heap would consist of just a root list of k nodes
  - If nothing else is done, this degrades to an array
- Fortunately, when we perform a extractMin operation, it will go through the entire root list and consolidate nodes to reduce the size of the root list
- Overall idea: the operations on Fibonacci heaps delay work as long as possible

extractMin

- Step 1: remove min and concatenate its children into root list



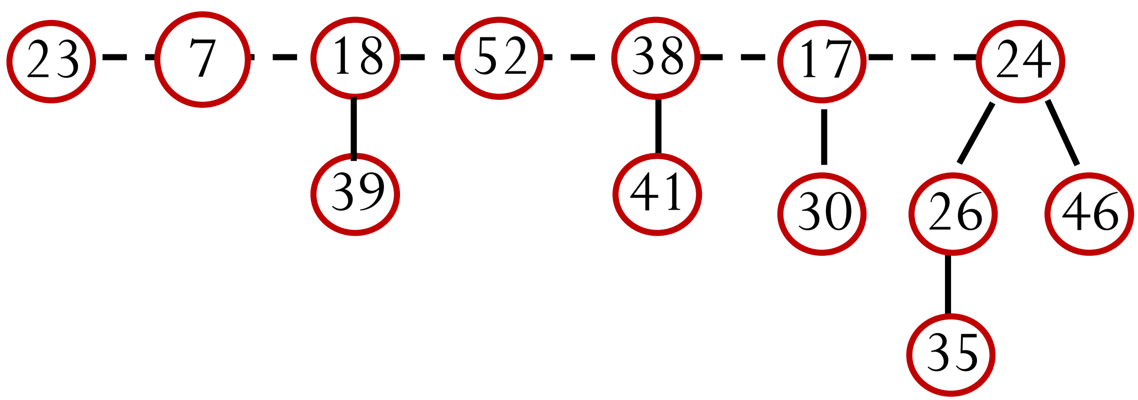
extractMin

- Step 2: **consolidate** the root list
  - Target: merge trees until every root in the root list has a **distinct** degree
- Consolidation iterates over all roots in the root list
  - If find two roots x and y with the same degree and assume $x.key \leq y.key$, remove y from the root list and make it a child of x
- Use an auxiliary array A, where A[i] is either null or storing a root with degree i
  - Size of A is the $D(n) + 1$, where $D(n)$ is the maximum degree of any node in an n-node Fibonacci heap.
    - $D(n) = \lfloor \log_\phi n \rfloor$, where $\phi = (1 + \sqrt{5})/2 \approx 1.618$

Consolidating Illustration

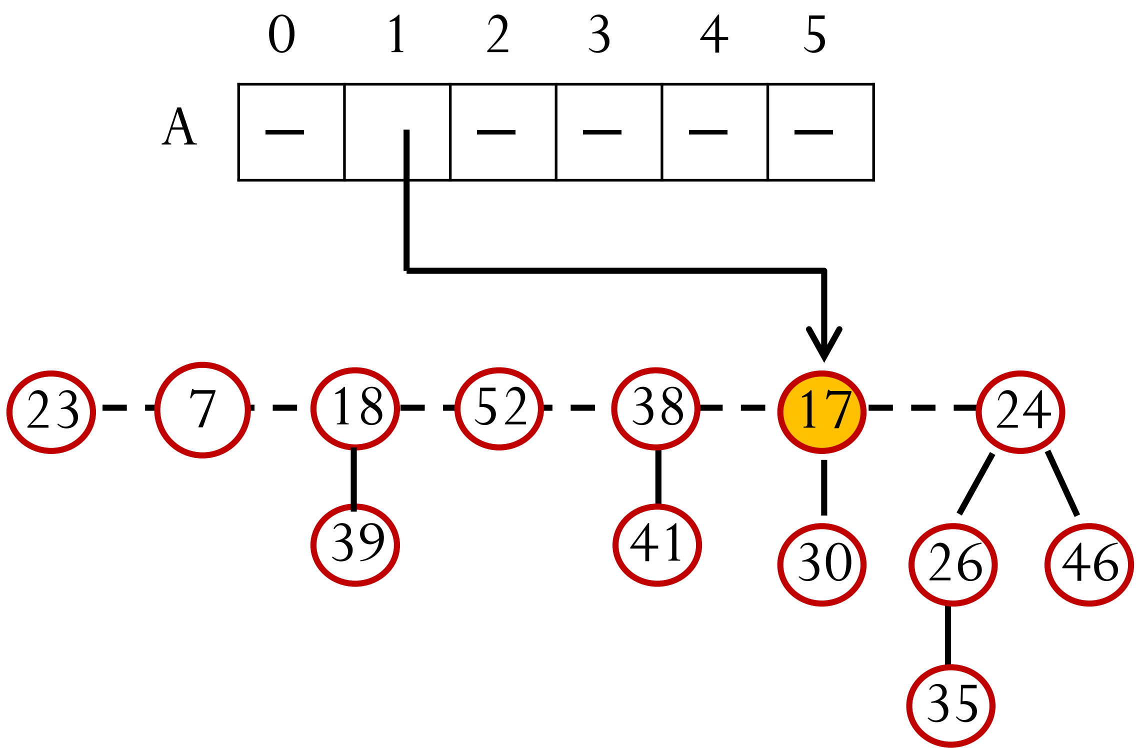
- $D(n) = \lfloor \log_{\phi} 13 \rfloor = 5$

	0	1	2	3	4	5
A	—	—	—	—	—	—



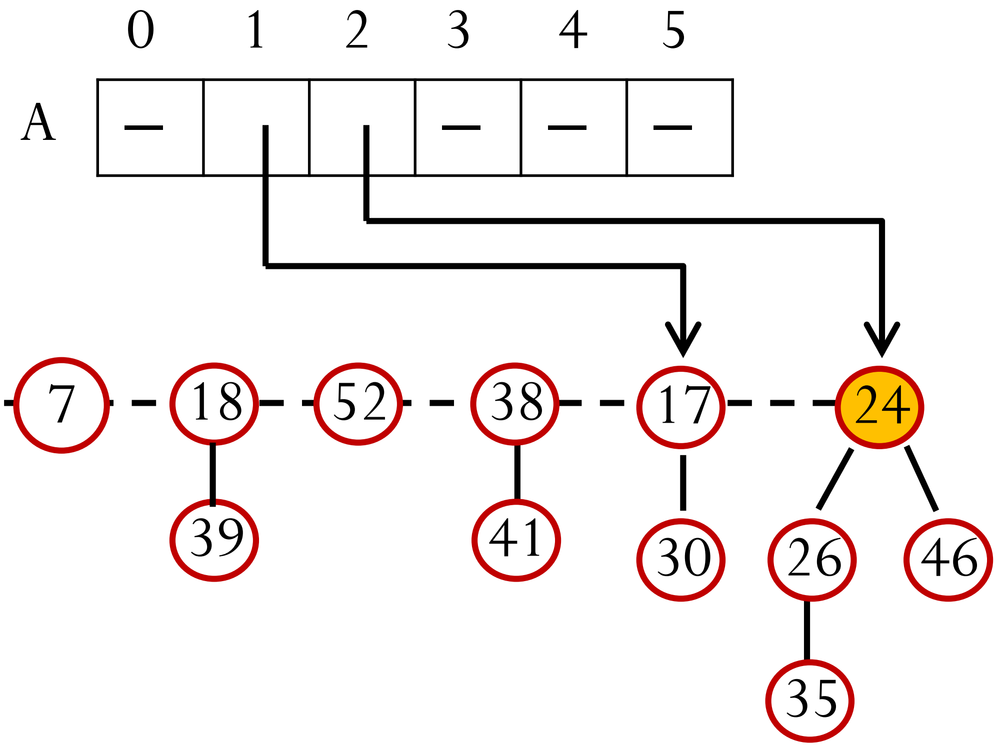
Consolidating Illustration

- Start from the right node of the original H.min, i.e., root 17



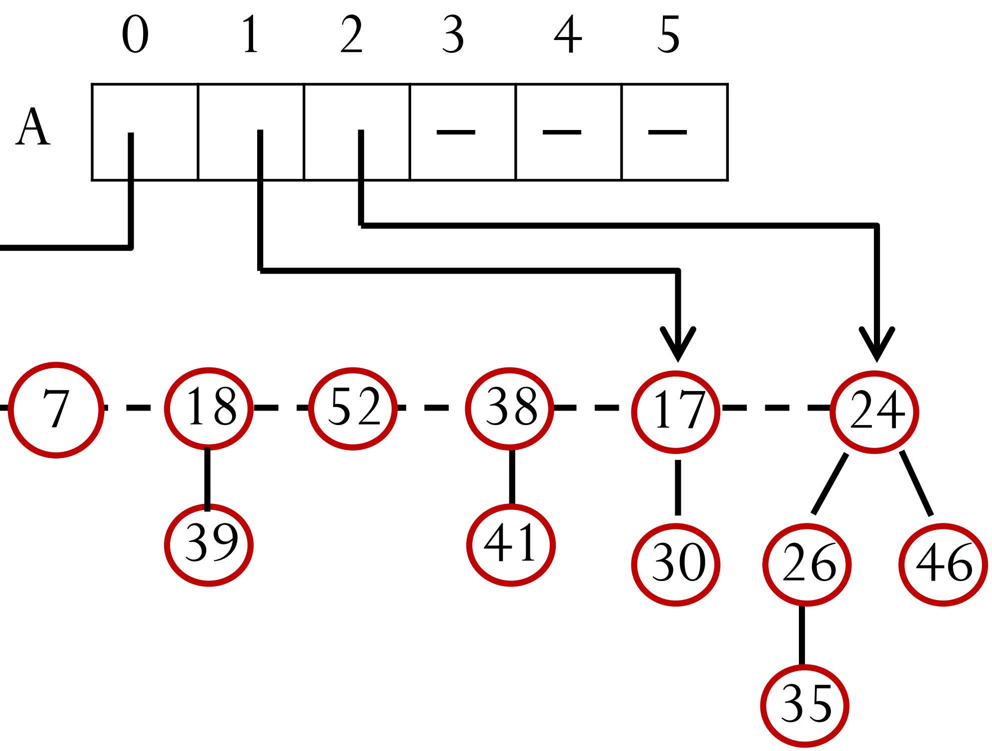
Consolidating Illustration

- Next root to check is 24



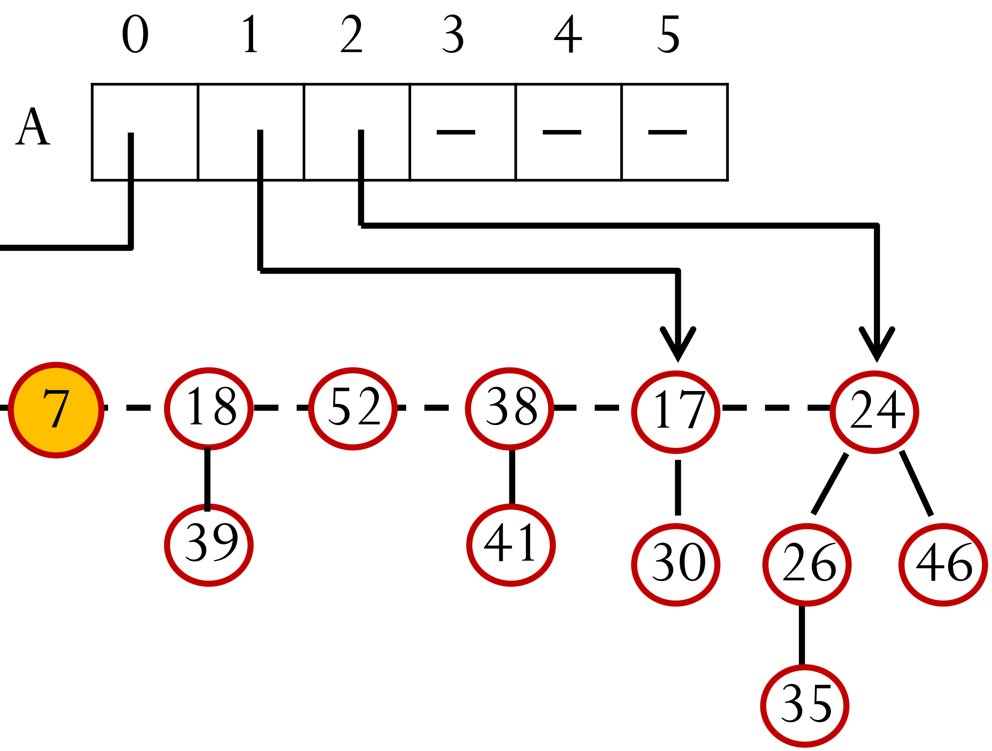
Consolidating Illustration

- Next root to check is 23



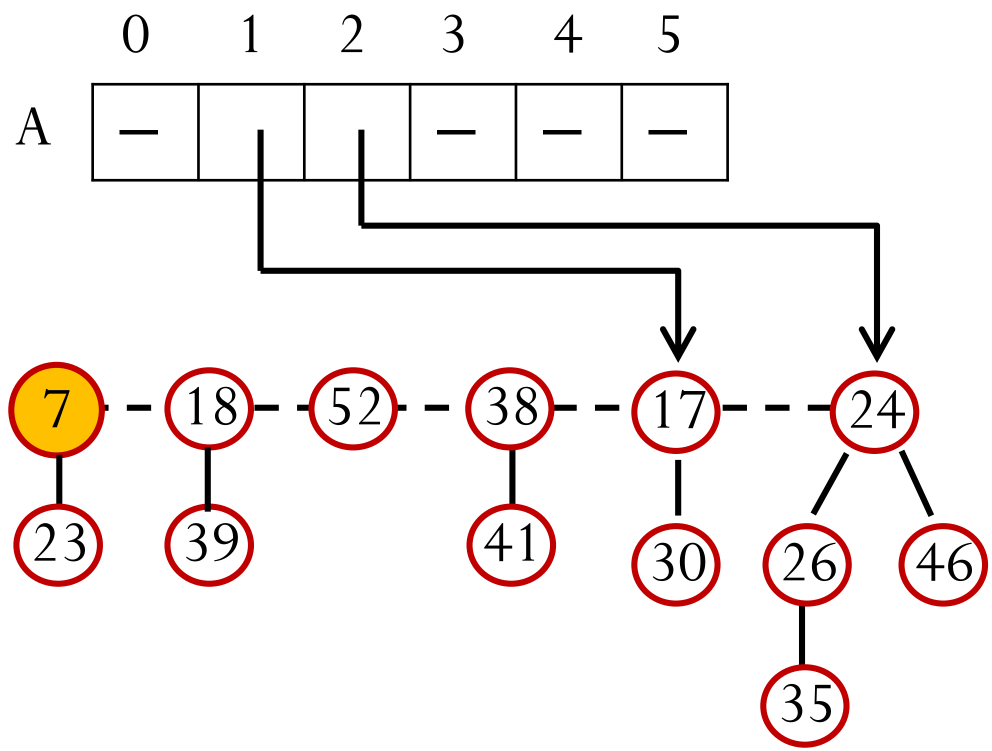
Consolidating Illustration

- Next root to check is 7, with degree 0
  - but we already have a root with degree 0 , i.e., 23. So, merge



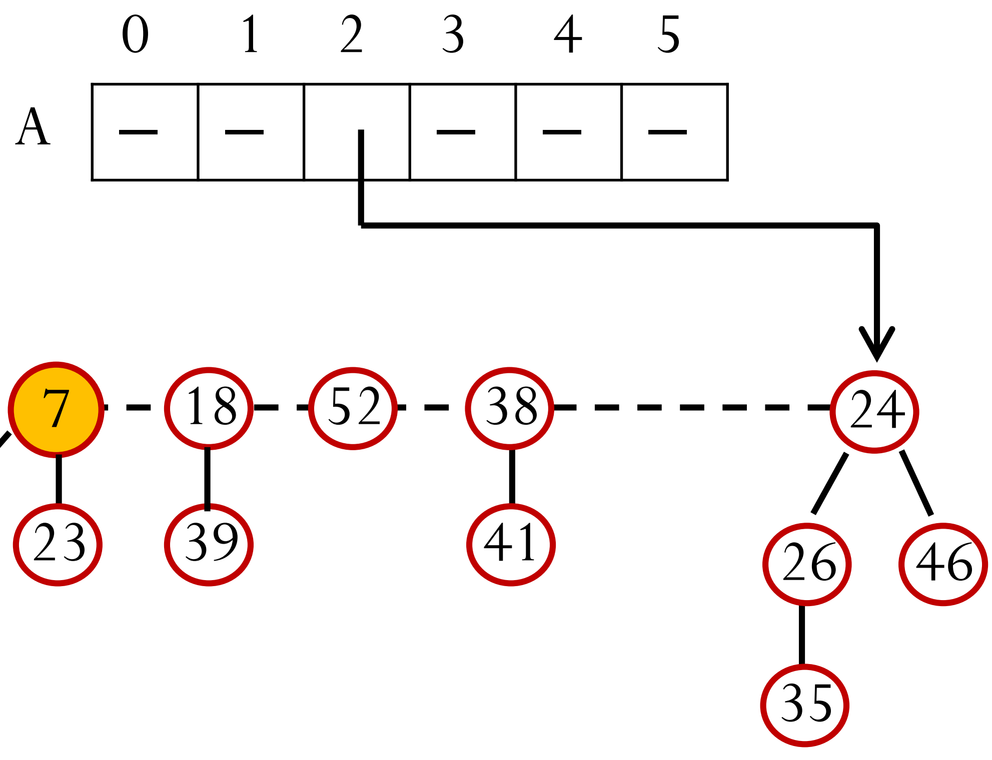
Consolidating Illustration

- Merge tree 23 with tree 7. This create a root of degree 1
  - but we already have a root with degree 1, i.e., 17. So, merge again



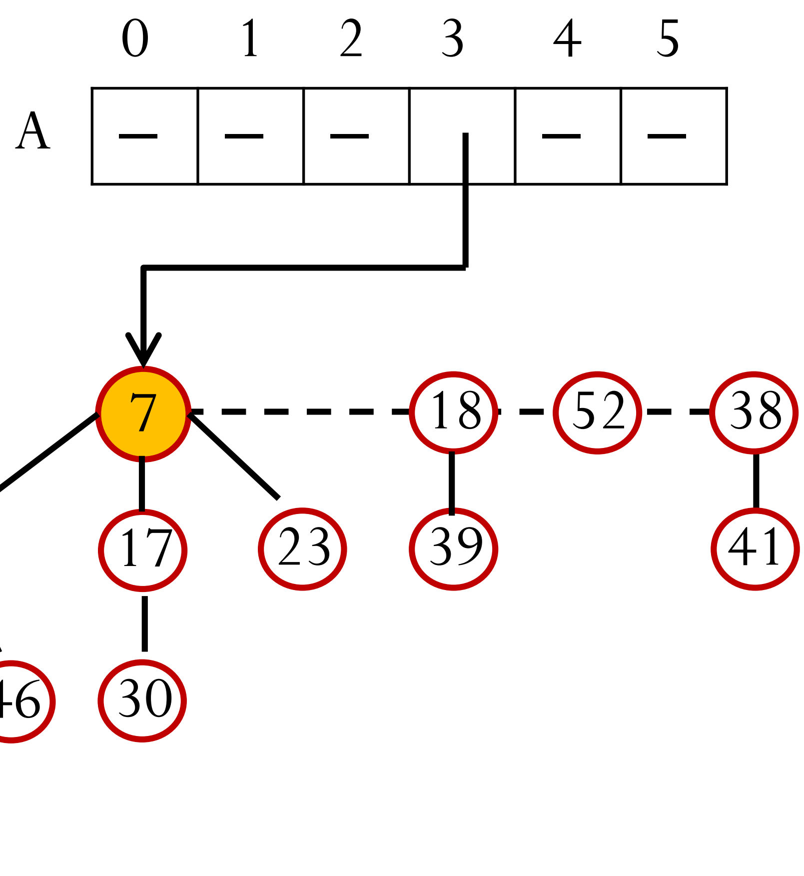
Consolidating Illustration

- Merge tree 7 with tree 17. This create a root of degree 2
  - but we already have a root with degree 2, i.e., 24. So, merge again



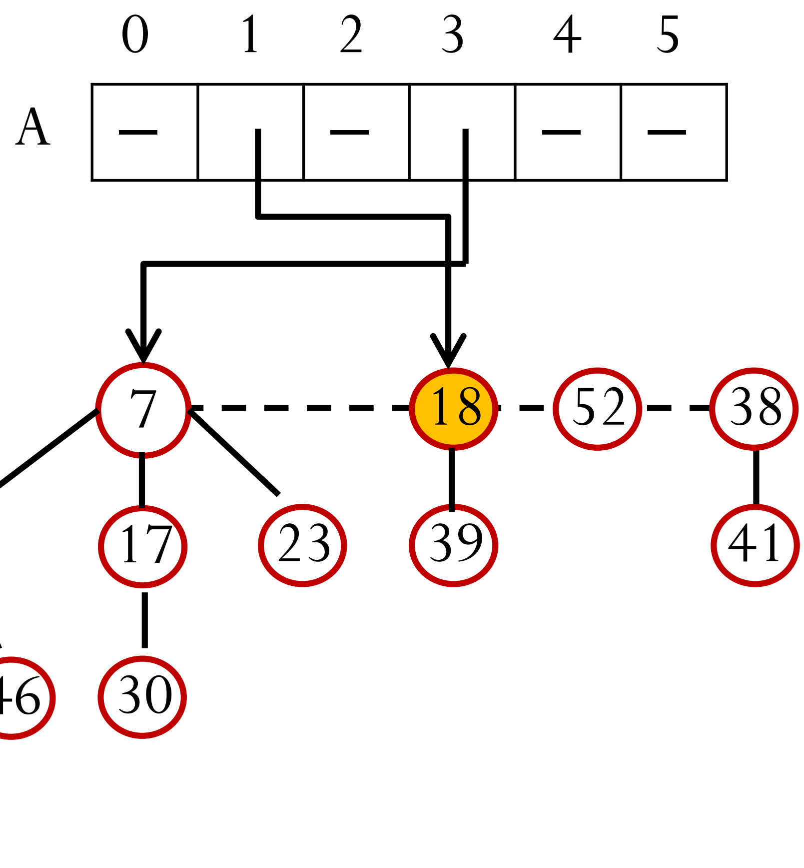
Consolidating Illustration

- Merge tree 7 with tree 24. This create a root of degree 3
  - It is unique. So, we put the new root into A[3]



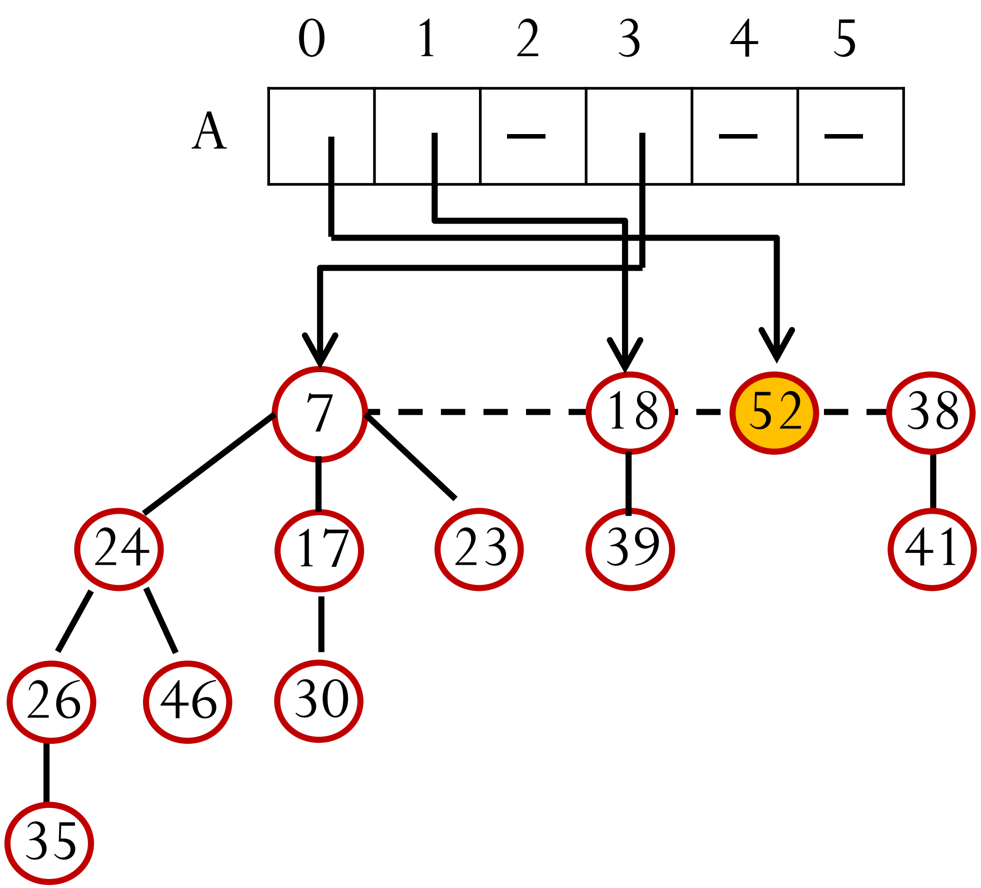
Consolidating Illustration

- Next root to check is 18, with degree 1



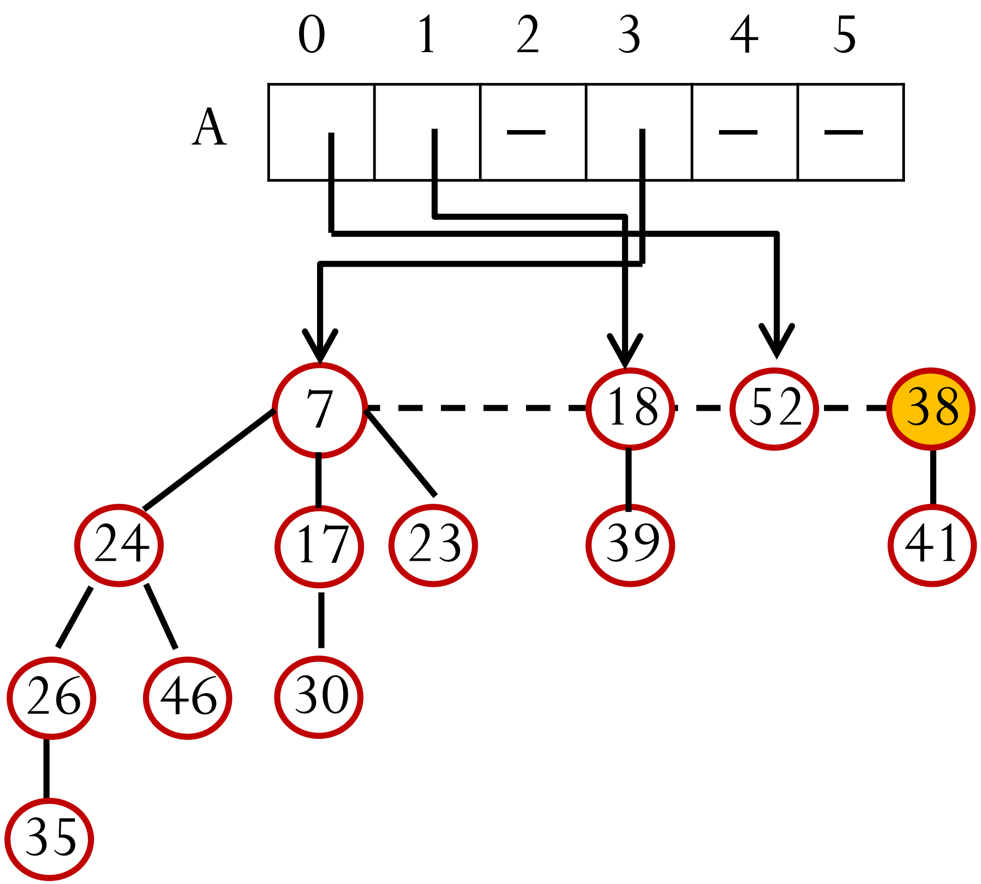
Consolidating Illustration

- Next root to check is 52, with degree 0



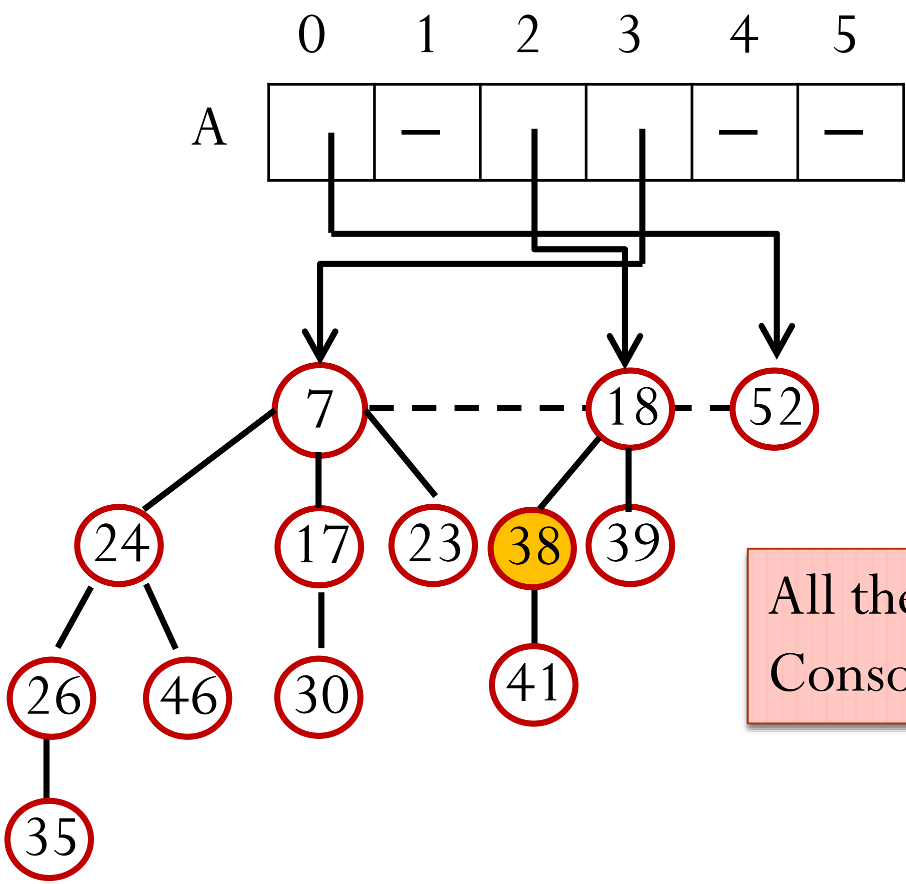
Consolidating Illustration

- Next root to check is 38, with degree 1
  - but we already have a root with degree 1, i.e., 18. So, merge



Consolidating Illustration

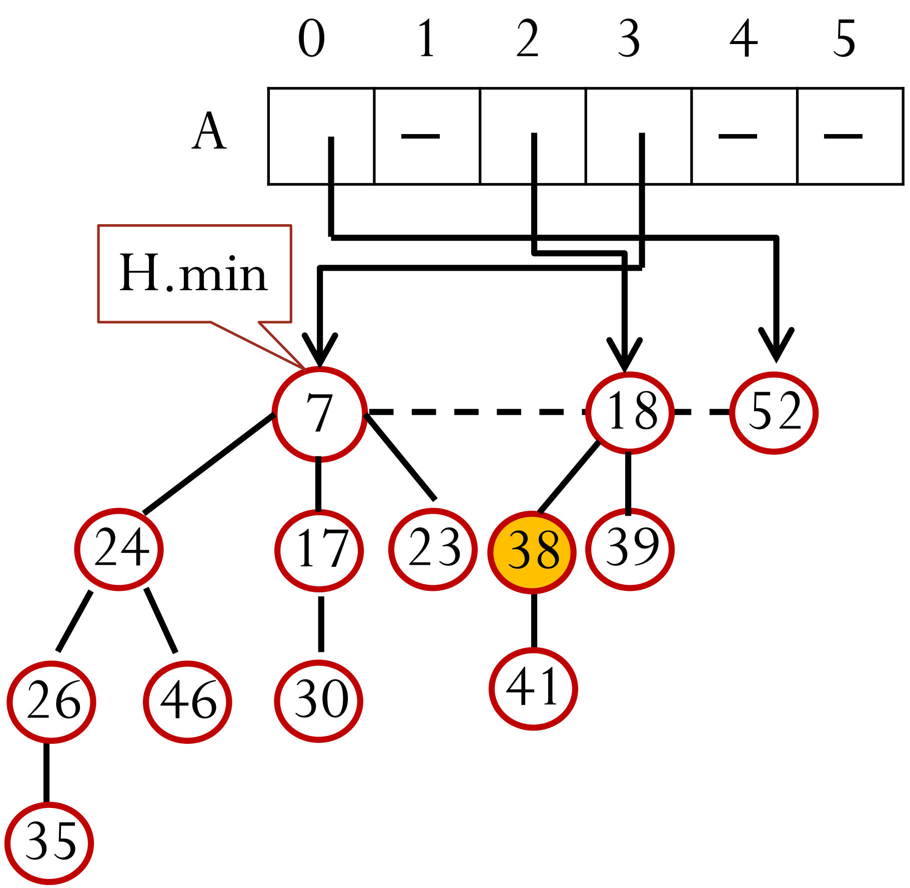
- Merge tree 38 with tree 18. This create a root of degree 2
  - It is unique. So, we put the new root into A[2]



All the roots have been visited. Consolidation completes

extractMin

- Step 3: link all the roots in array A together; update H.min



extractMin: Summary

- Step 1: remove min and concatenate its children into root list
- Step 2: consolidate the root list
  - Target: merge trees until every root in the root list has a distinct degree
- Step 3: link all the roots in array A together; update H.min

- Amortized time complexity: $O(\log n)$