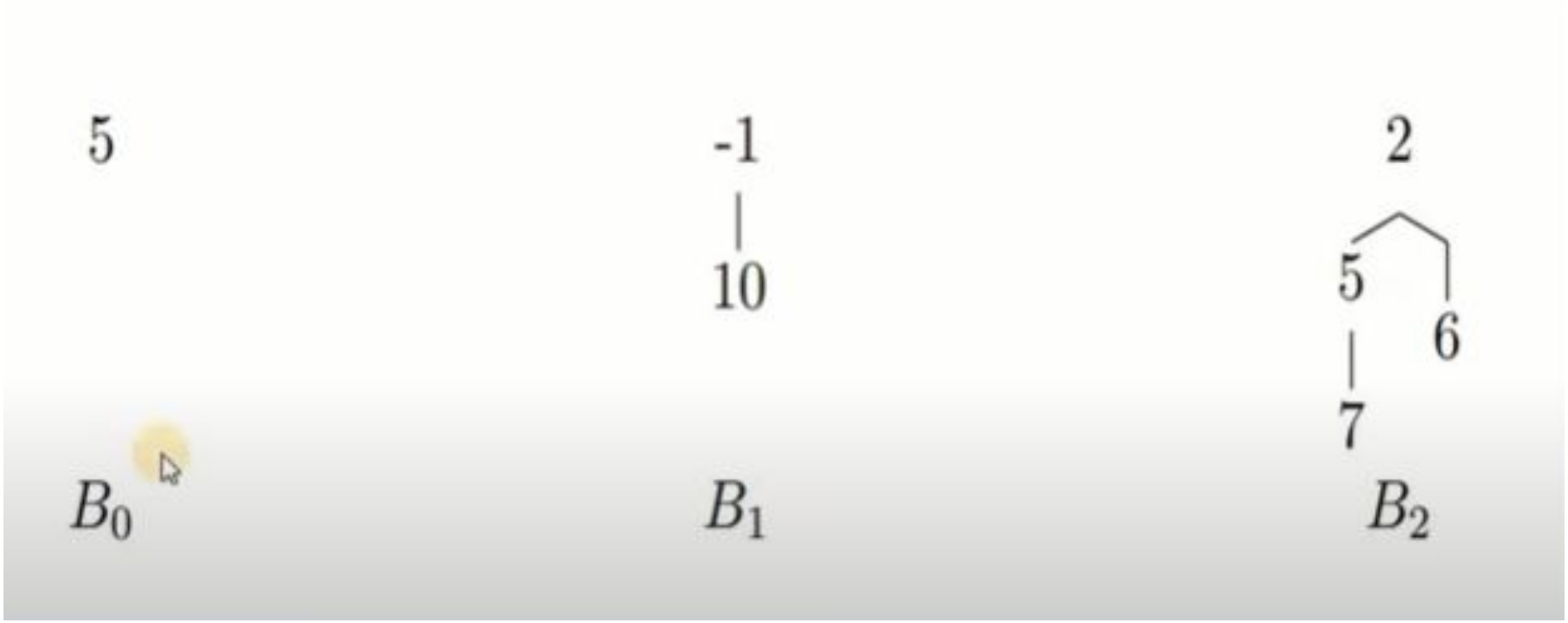
Fibonacci Heap

Heap (Binary Heap)

- Binary tree in which all nodes follow heap property
  - MinHeap: key(parent) <= key(child)
  - MaxHeap: key(parent) >= key(child)
  - All levels are completely filled except the last level, which is left filled
- Insertion - Add child at lowest level and shift up
- Deletion of root - Remove the rightmost leaf at the deepest level and use it for the new root and shift up

Binomial Tree

- A Binomial Tree B_k of order k is defined as follows
  - B_0 is a tree with one node
  - B_k is a pair of B_{k-1} trees, where root of one B_{k-1} becomes the left most child of the other (for all $k \geq 1$)
  - two Bk−1's are combined to get one Bk, the Bk−1 having minimum value at the root will be the root of Bk, the other Bk−1 will become the child node.


5
-1
10
2
5
6
7
B_0
B_1
B_2

Example - [5 -1 3 5 7 8 9]

-1 3 Insert 3 into B_1, we get one B_1 and a B_0.

5

Insert 5, -1 3 5, on merging two B_0's → -1 3 on merging two B_1's → -1

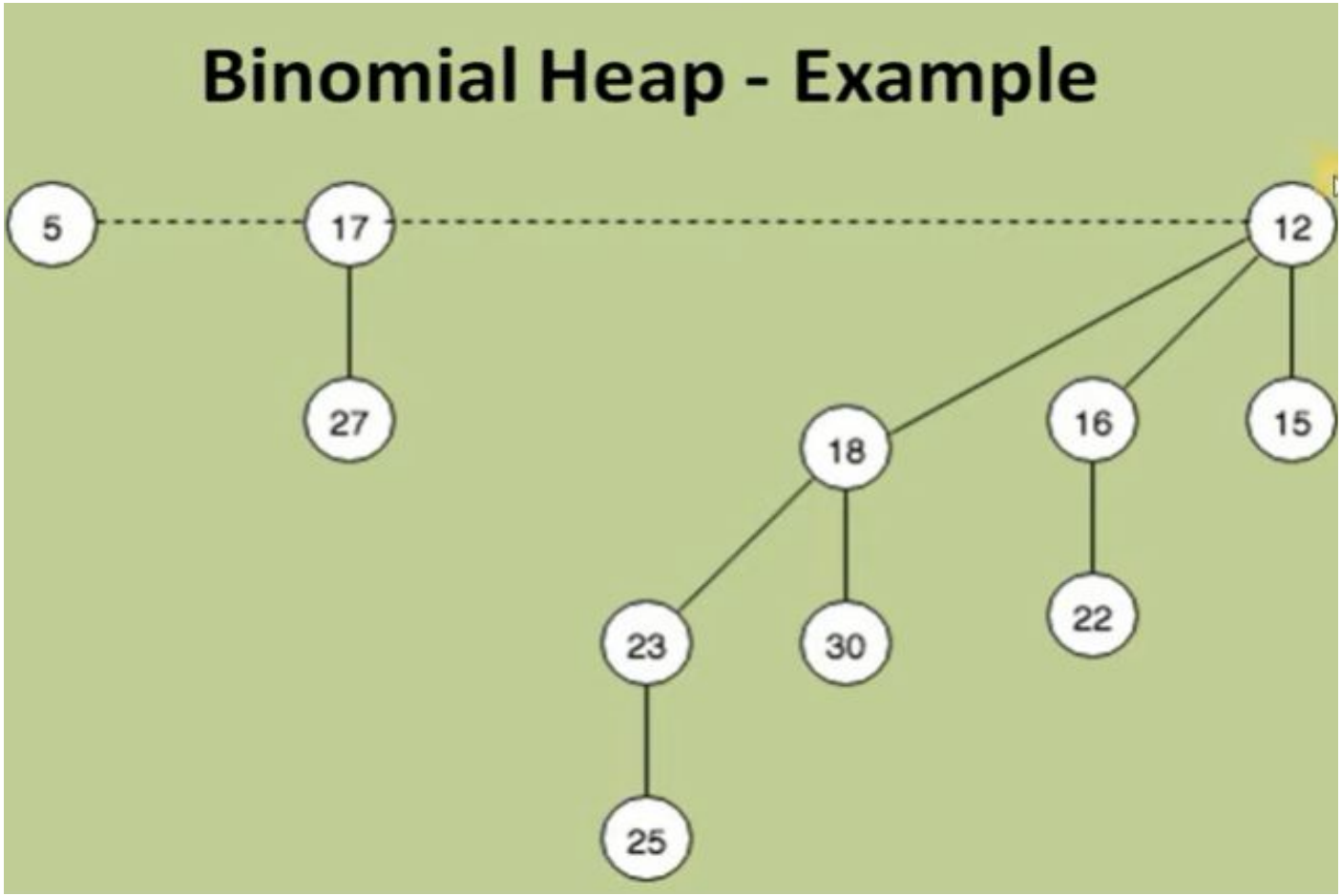
5 5 5 3 5

5

Insert 7, and 8. -1 7 8, on merging two B_0's → -1 7 Insert 9, -1 7 9

3 5 3 5 5 3 5 8 3 5 8

5 5 5

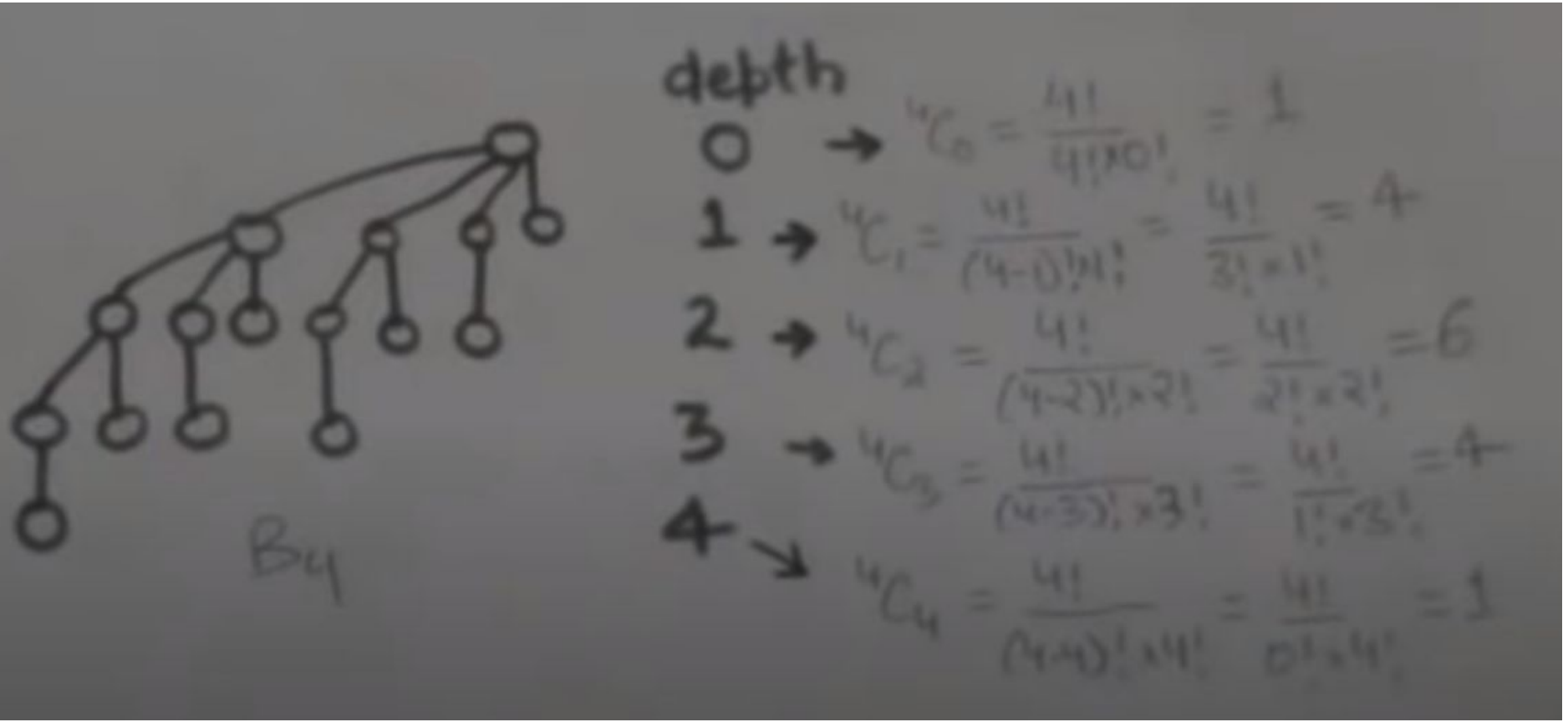

Binomial Heap - Example
5
17
12
27
18
16
15
23
30
22
25

Structural Properties

For the binomial tree Bk.

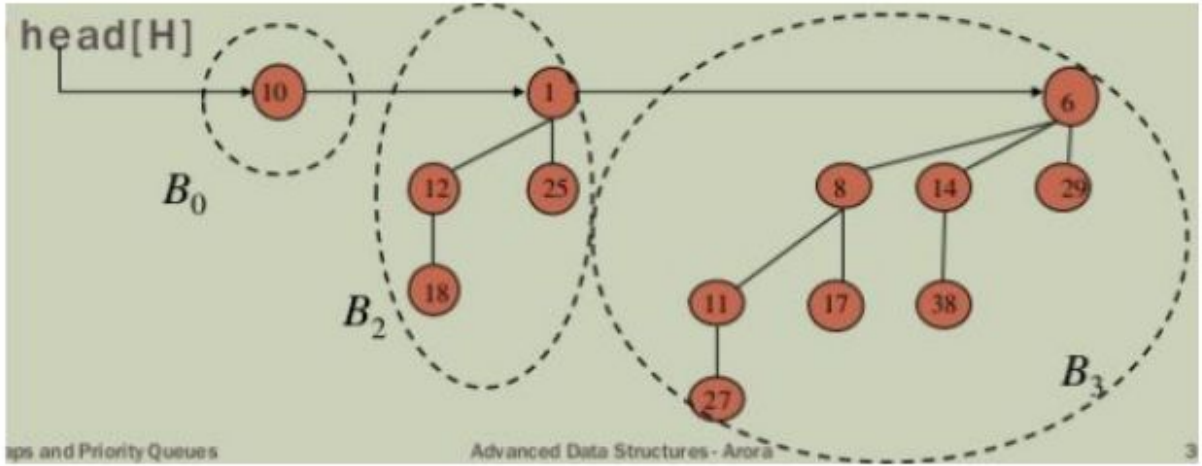
- There are 2k nodes.
- The height of the binomial tree is k.
- There are exactly kCi nodes at depth i = 0, 1, . . . , k.
- The root has degree k, which is greater than that of any other node, moreover if the children of the root are numbered from left to right by k − 1, k − 2, . . . , 0, child i is the root of the Subtree Bi .

Note: Due to Property 3, it gets the name binomial tree (heap).


depth
0 → $^4C_0 = \frac{4!}{4! \times 0!} = 1$
1 → $^4C_1 = \frac{4!}{(4-0)!1!} = \frac{4!}{3! \times 1!} = 4$
2 → $^4C_2 = \frac{4!}{(4-2)! \times 2!} = \frac{4!}{2! \times 2!} = 6$
3 → $^4C_3 = \frac{4!}{(4-3)! \times 3!} = \frac{4!}{1! \times 3!} = 4$
4 → $^4C_4 = \frac{4!}{(4-4)! \times 4!} = \frac{4!}{0! \times 4!} = 1$
B4

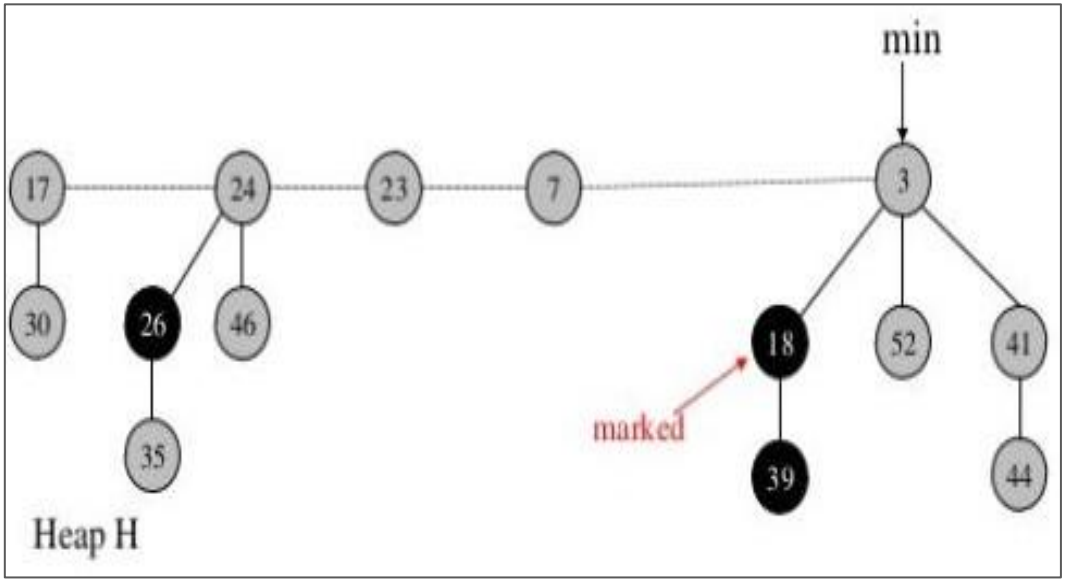
Binomial Heap

- Pointer points to the first node to enter into the heap
- The roots of the trees are connected so that sizes of the connected trees are in order



What is Fibonacci Heap?

- Collection of trees satisfying minimum-heap property i.e parent(value)<child(value)
- Maintains pointer to minimum element
- Contains set of marked nodes (to indicate if node has lost a child)
- Roots of all trees are linked using circular doubly linked list



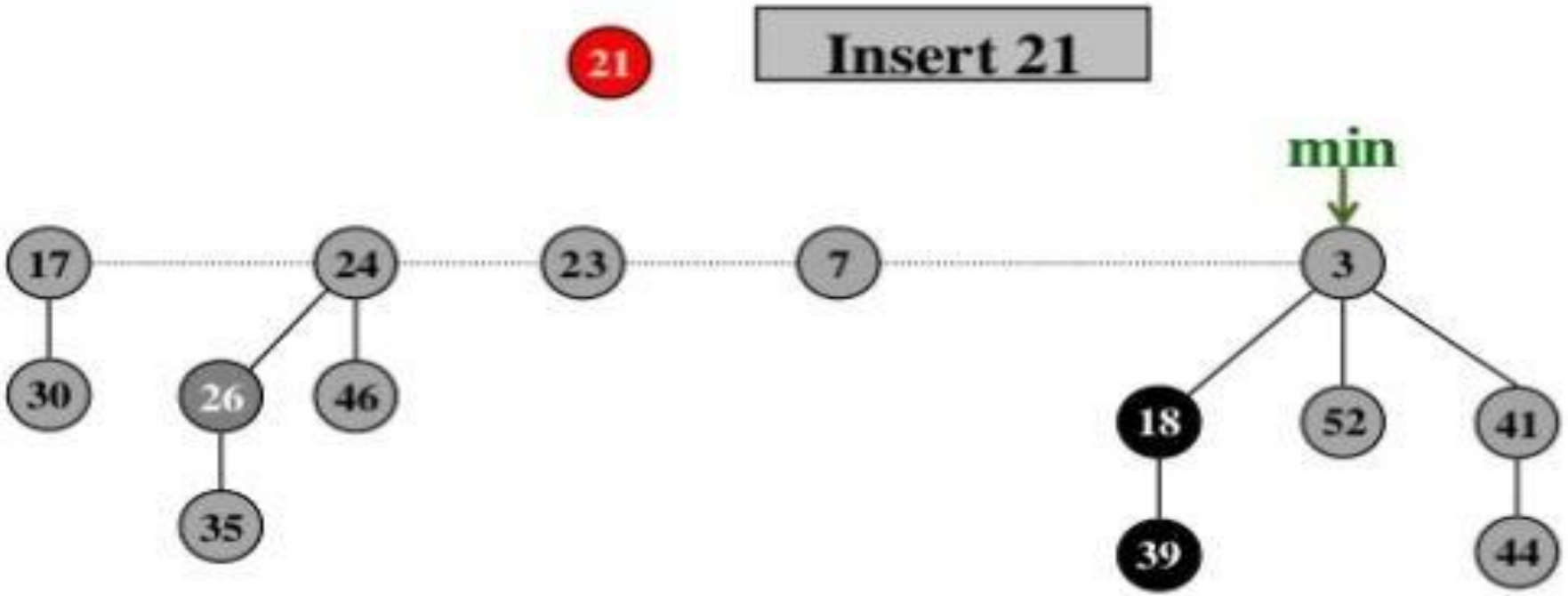
Lazy Consolidation Approach

- Eager Approach (Followed by Binomial Heap)
  - Insert: Create new heap, union and consolidate
  - Union: Combine the lists and consolidate
  - Delete: Delete the minimum, merge the lists and consolidate
- Lazy Approach (Followed by Fibonacci Heap)
  - Insert: Simply add to the list and update minimum if needed
  - Union: Simply combine lists using pointers and update minimum if needed
  - Delete: Delete the minimum, merge the lists and consolidate

Fibonacci Heap Operations

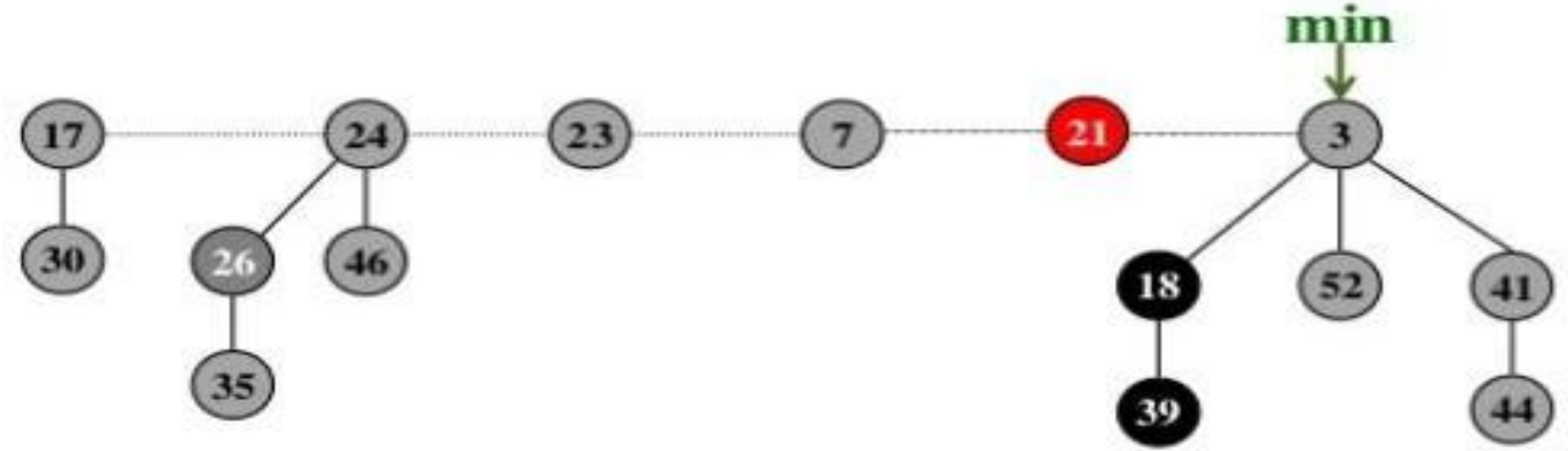
- Insert node
- Extract minimum node / Delete minimum node
- Union
- Decrease key
- Delete node

Insert node



Insert node

- Create new singleton tree
- Add to root list
- Update minimum pointer

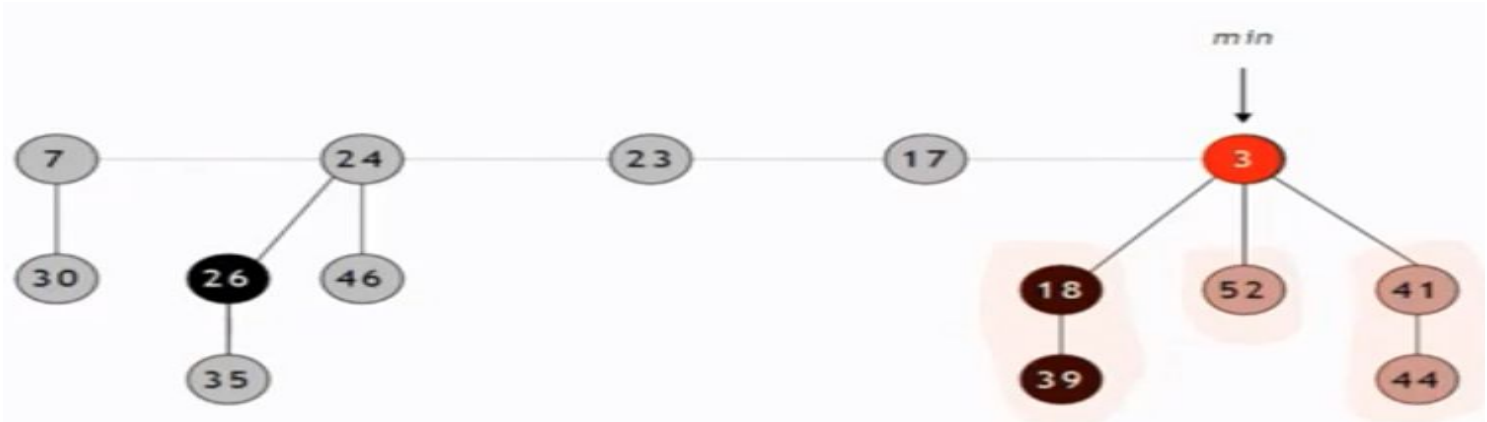


Extract / Delete minimum node

Step 1:Extract min and concatenate children into root list, and update min to the next root

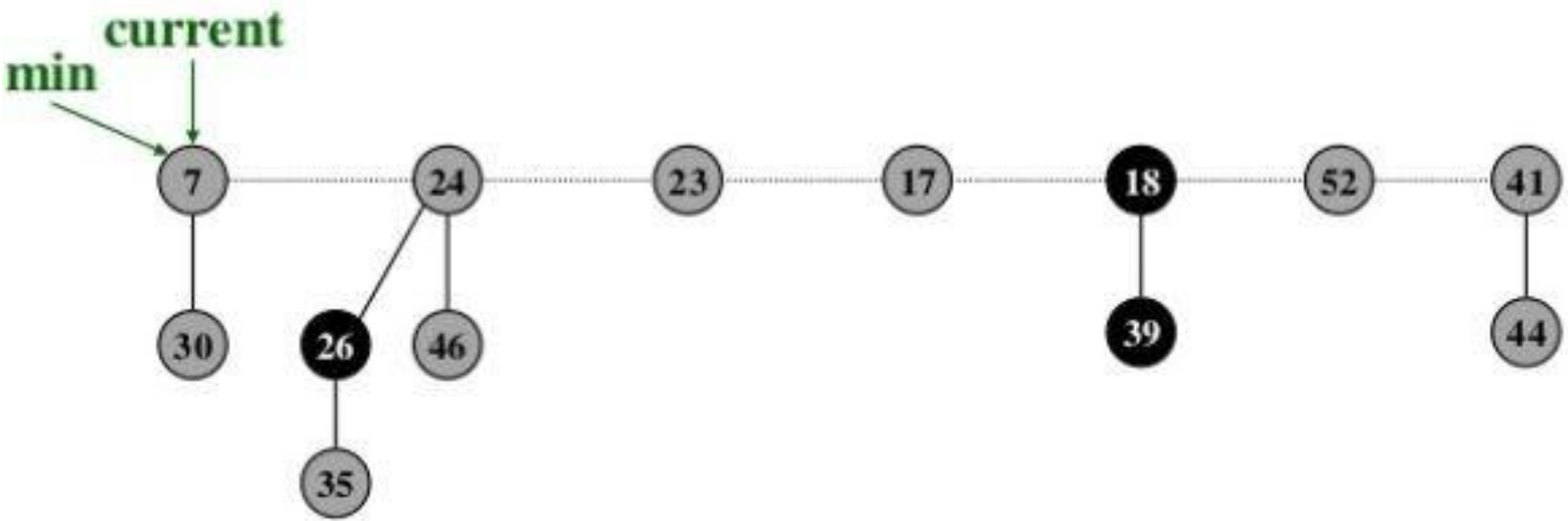
Step 2: Consolidate trees so that no two trees have the same rank/degree

1. Find two roots x and y having the same order and x(value)<y(value)
2. Link y to x i.e remove y from root list and make y the child of x



Extract minimum node (step 1)

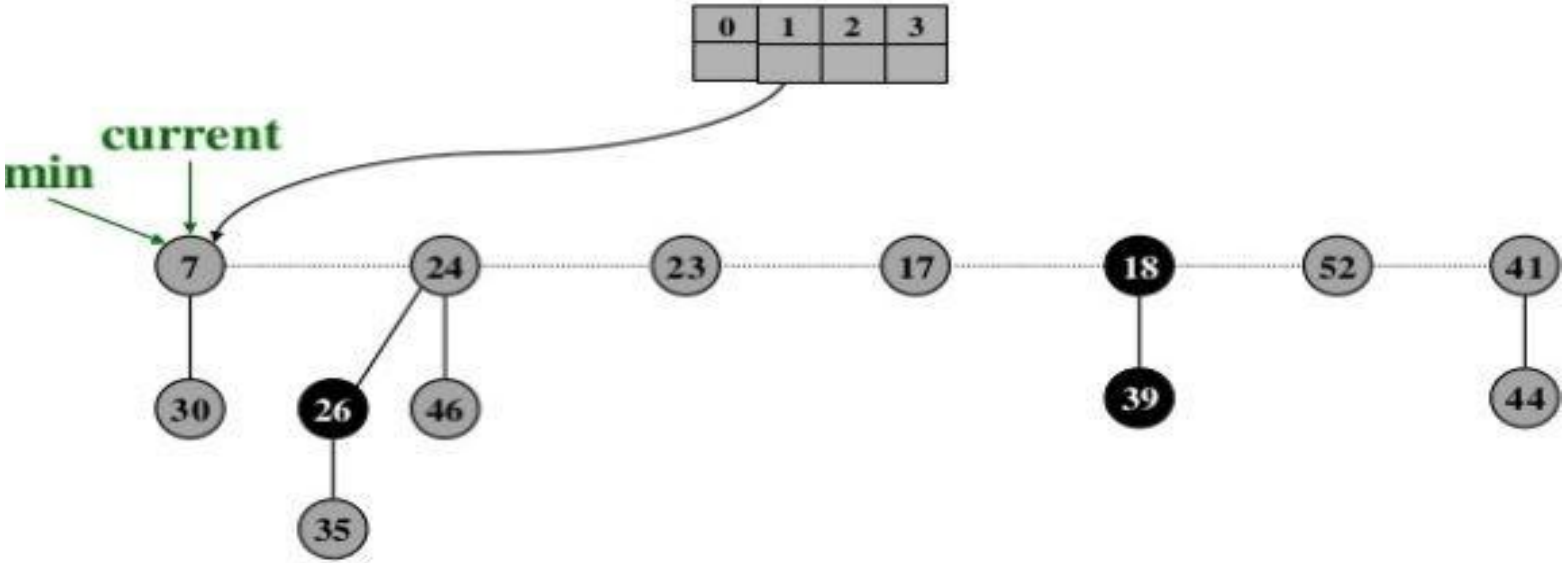
Extract min **and concatenate** children into root list**,** and **update** min to the next root



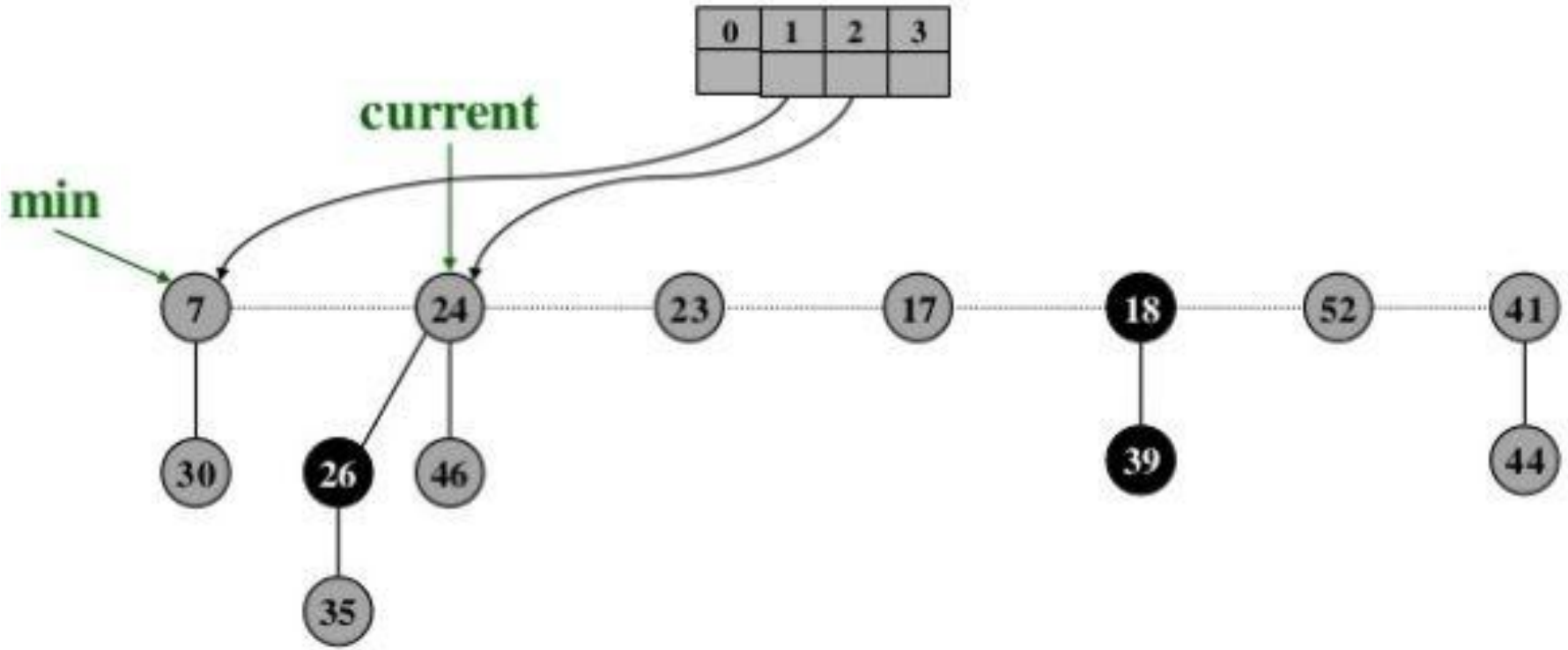
Extract minimum node (step 2)

Consolidate trees so that no two trees have the same rank/degree

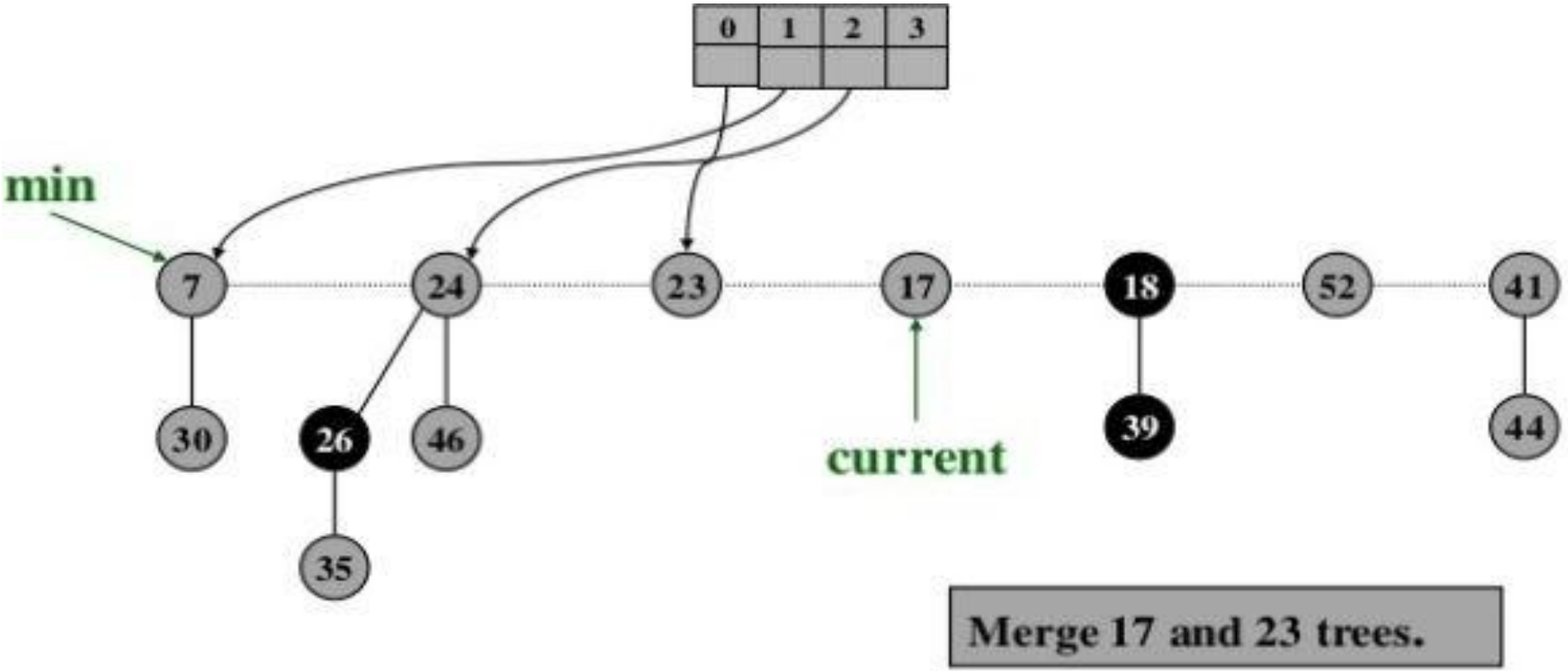
1. Find two roots x and y having the same order and x(value)<y(value)
2. Link y to x i.e remove y from root list and make y the child of x



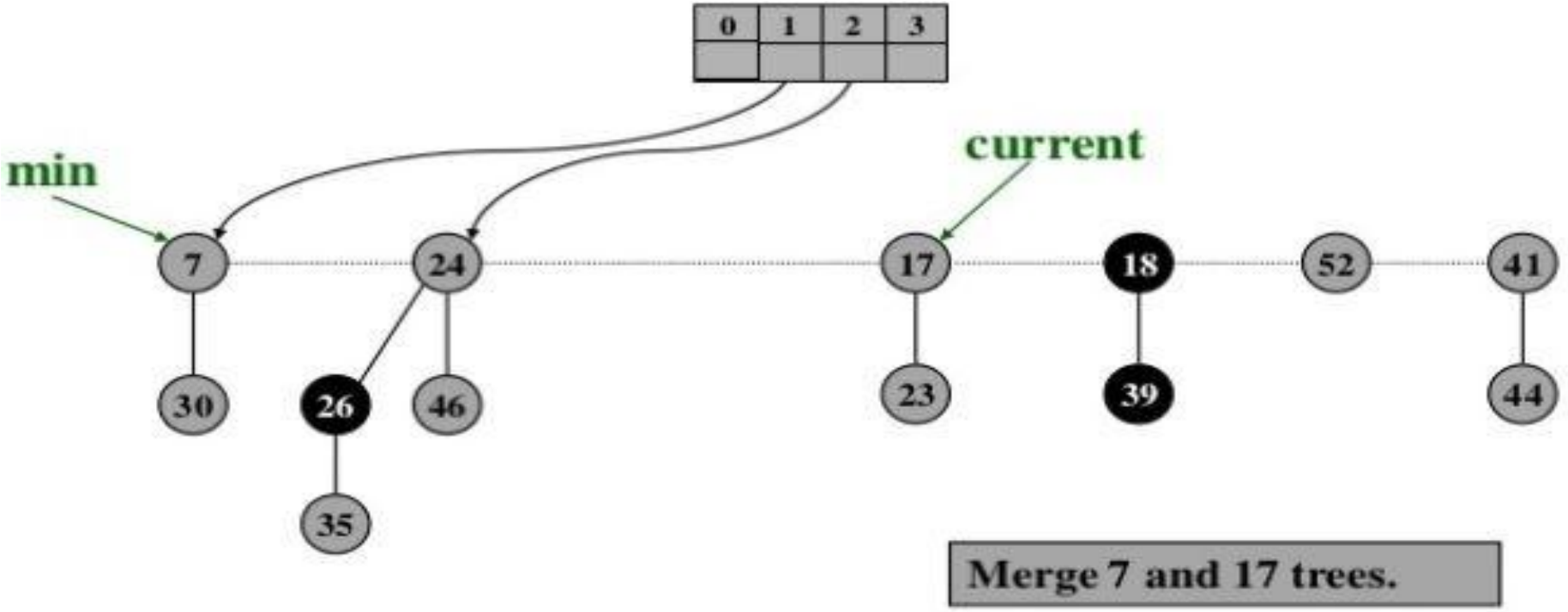
Consolidation (contd)



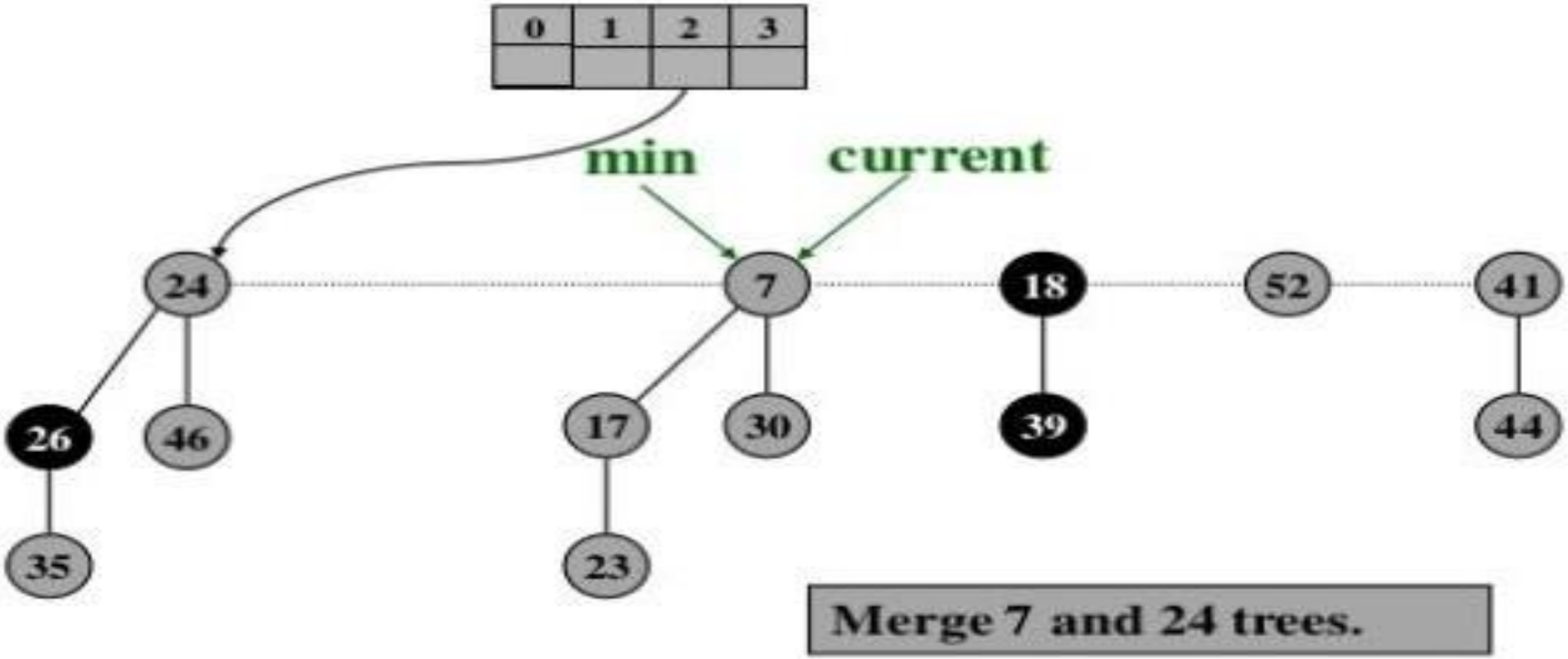
Consolidation (contd)



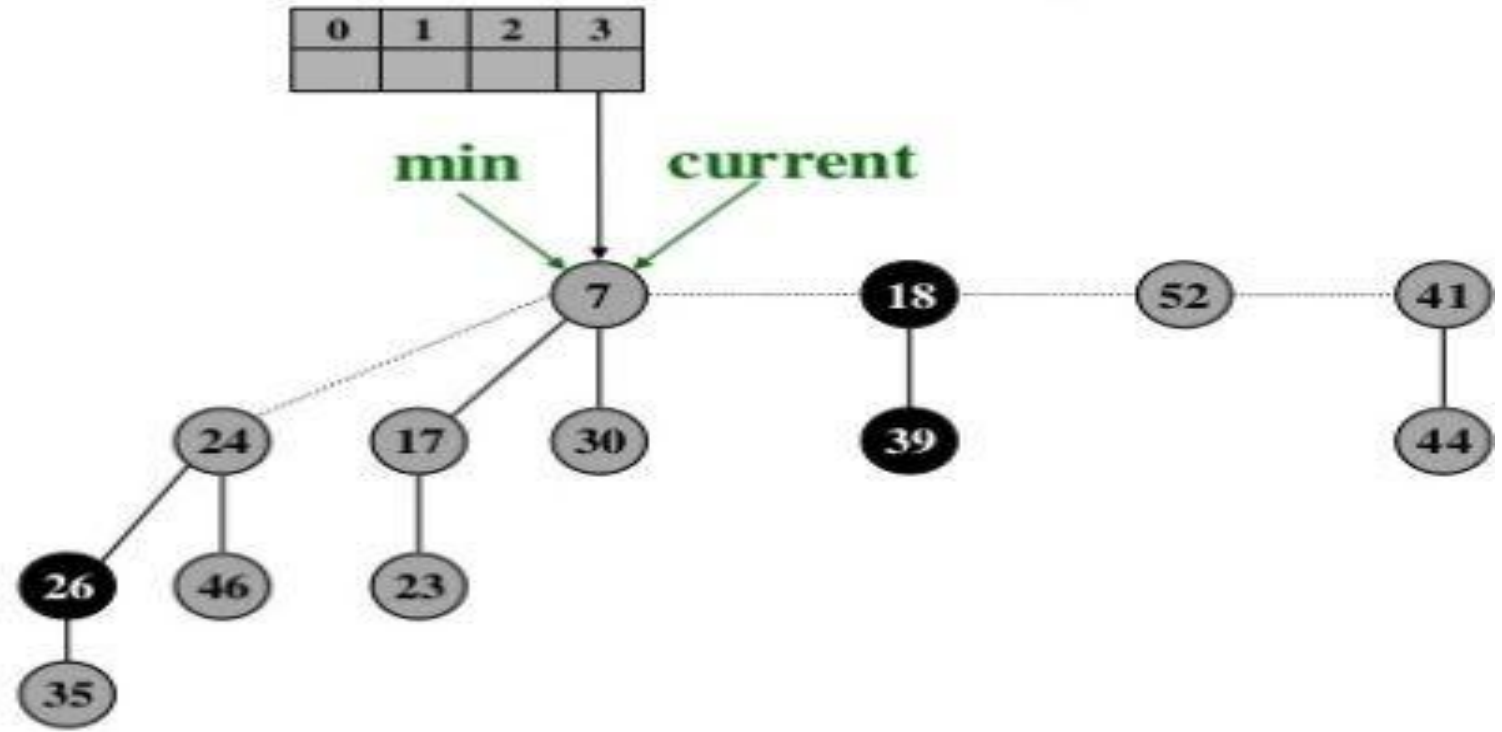
Consolidation (contd)



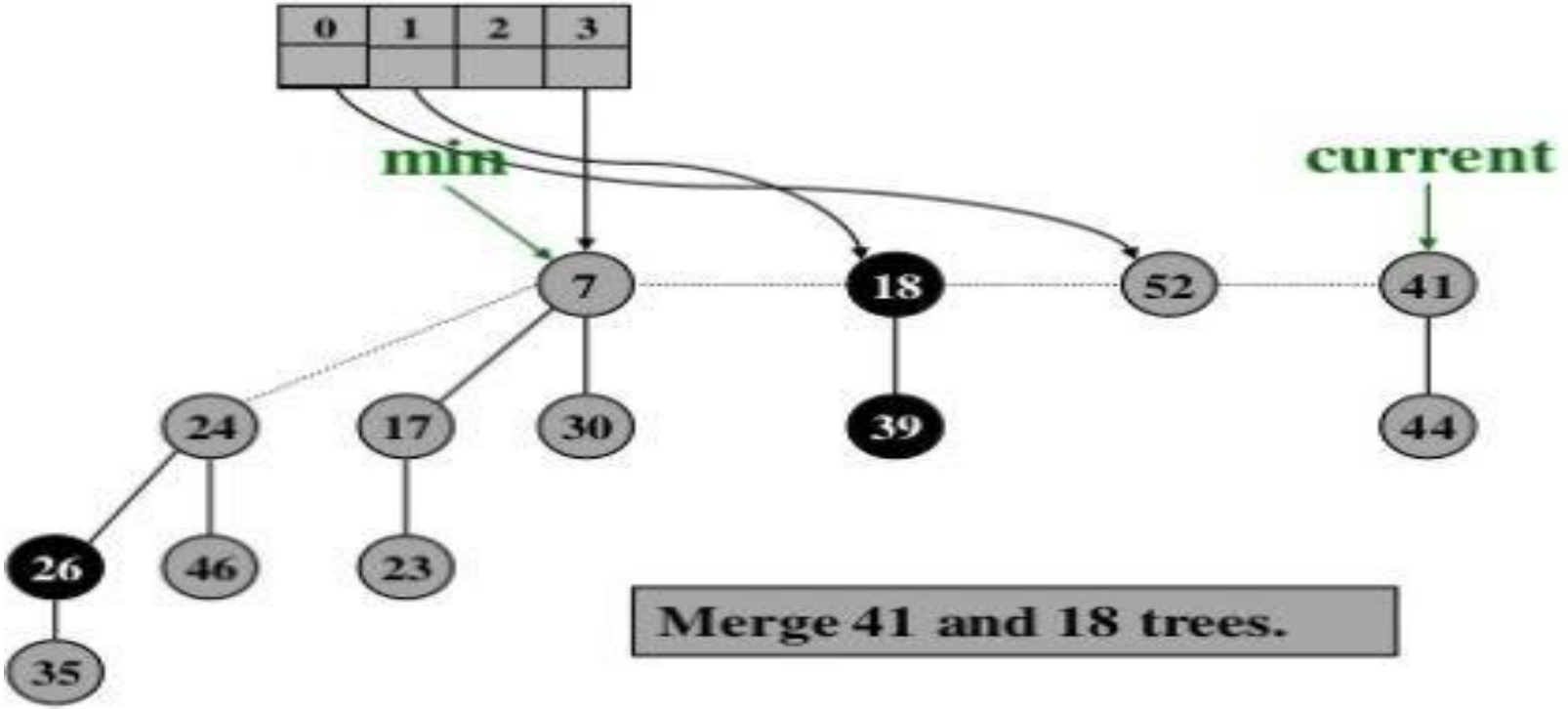
Consolidation (contd)



Consolidation (contd)

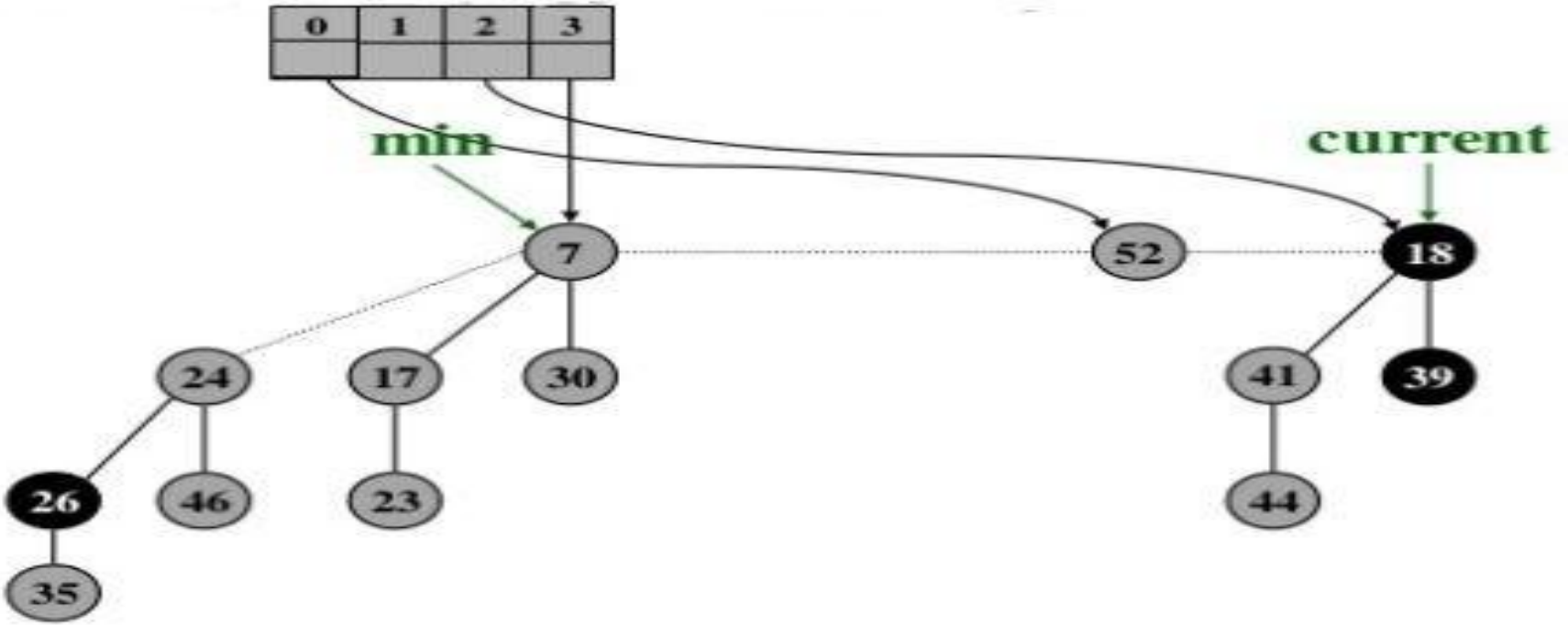


Consolidation (contd)

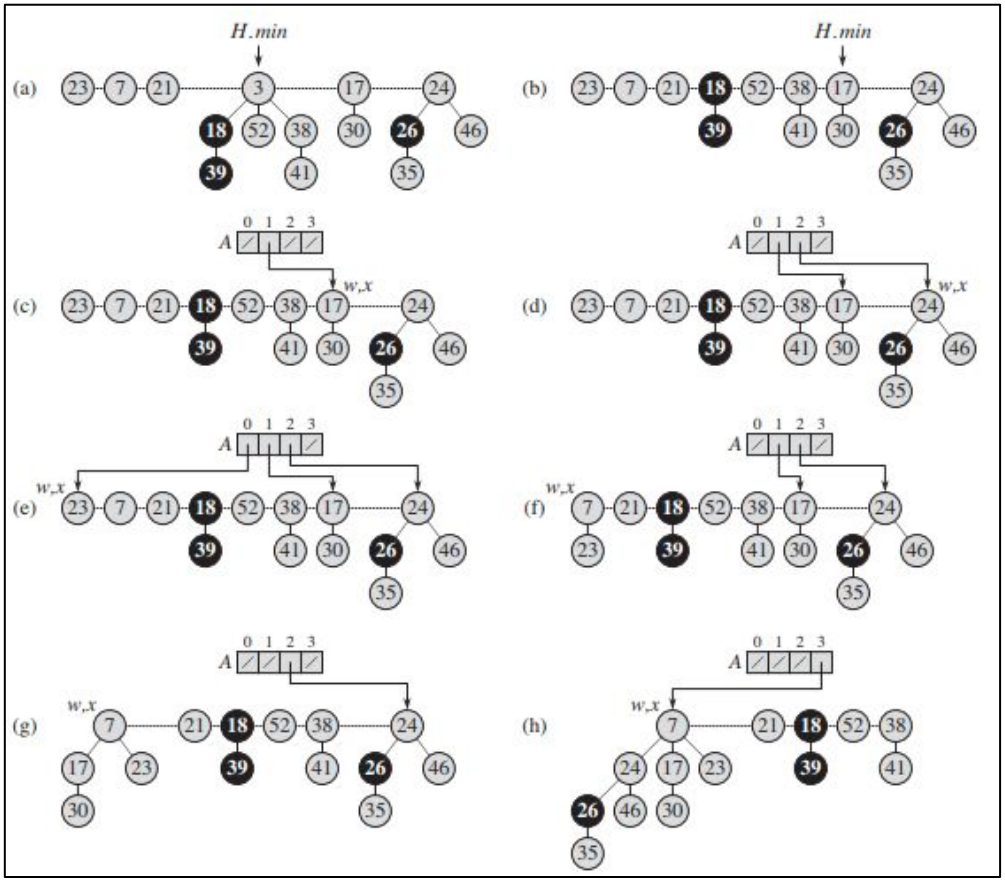


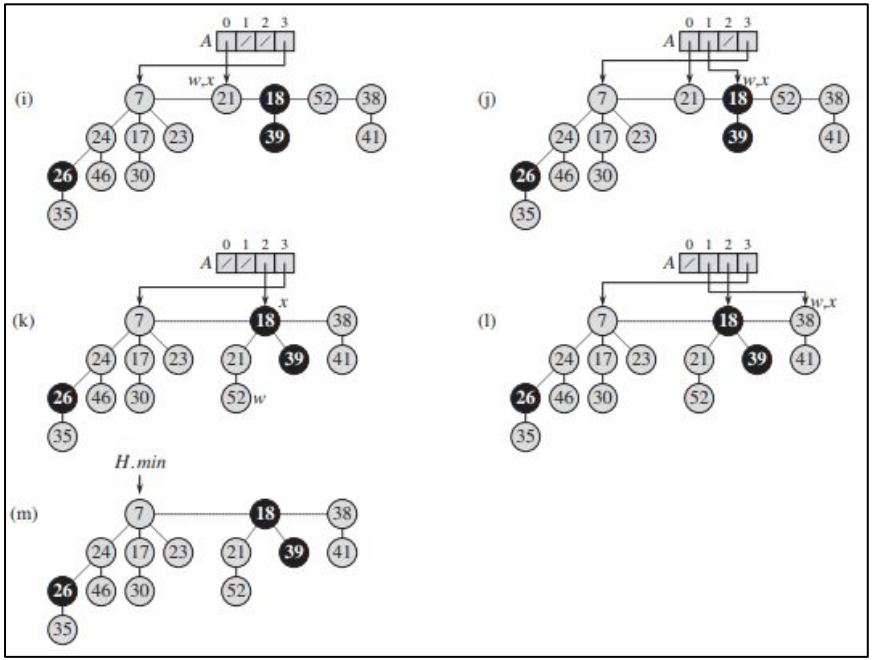
Consolidation (contd)

Stop when all trees have different orders



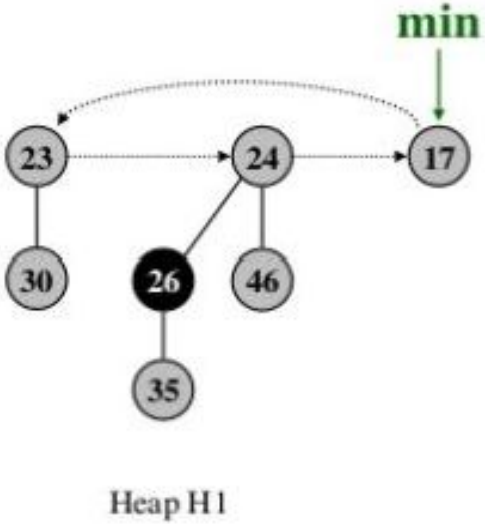
Extract Min Example



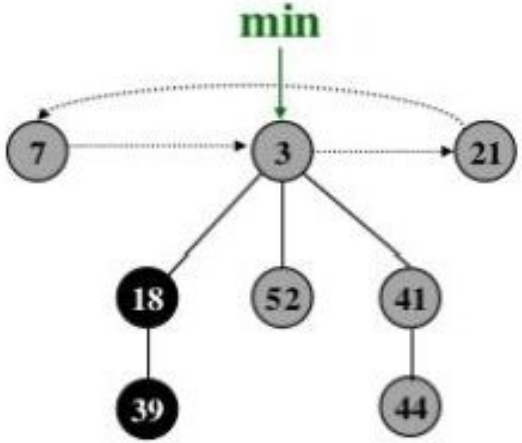


Union

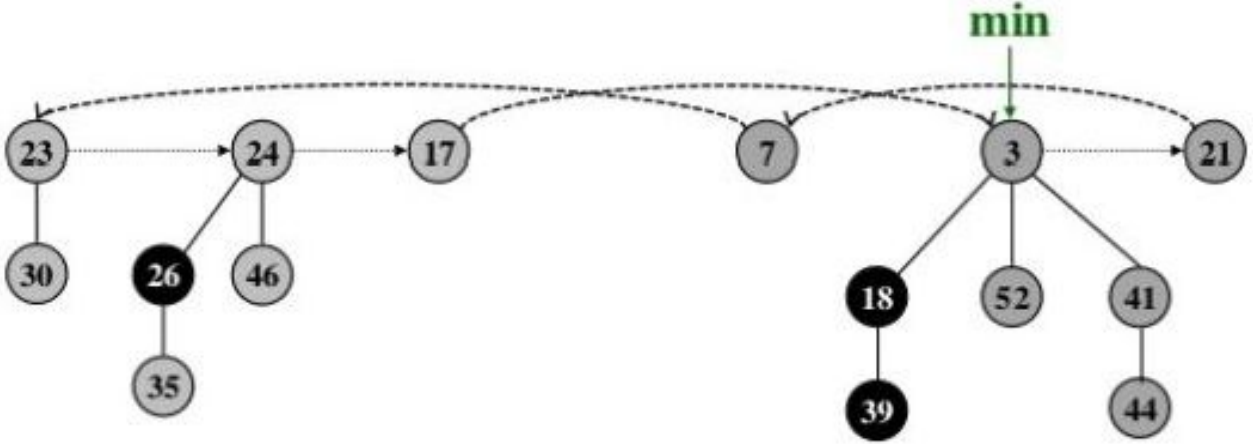
- Concatenate the root list of H1 and H2 into new root list H
- Set the minimum node of H
- Set n[H] to total number of nodes



Heap H1



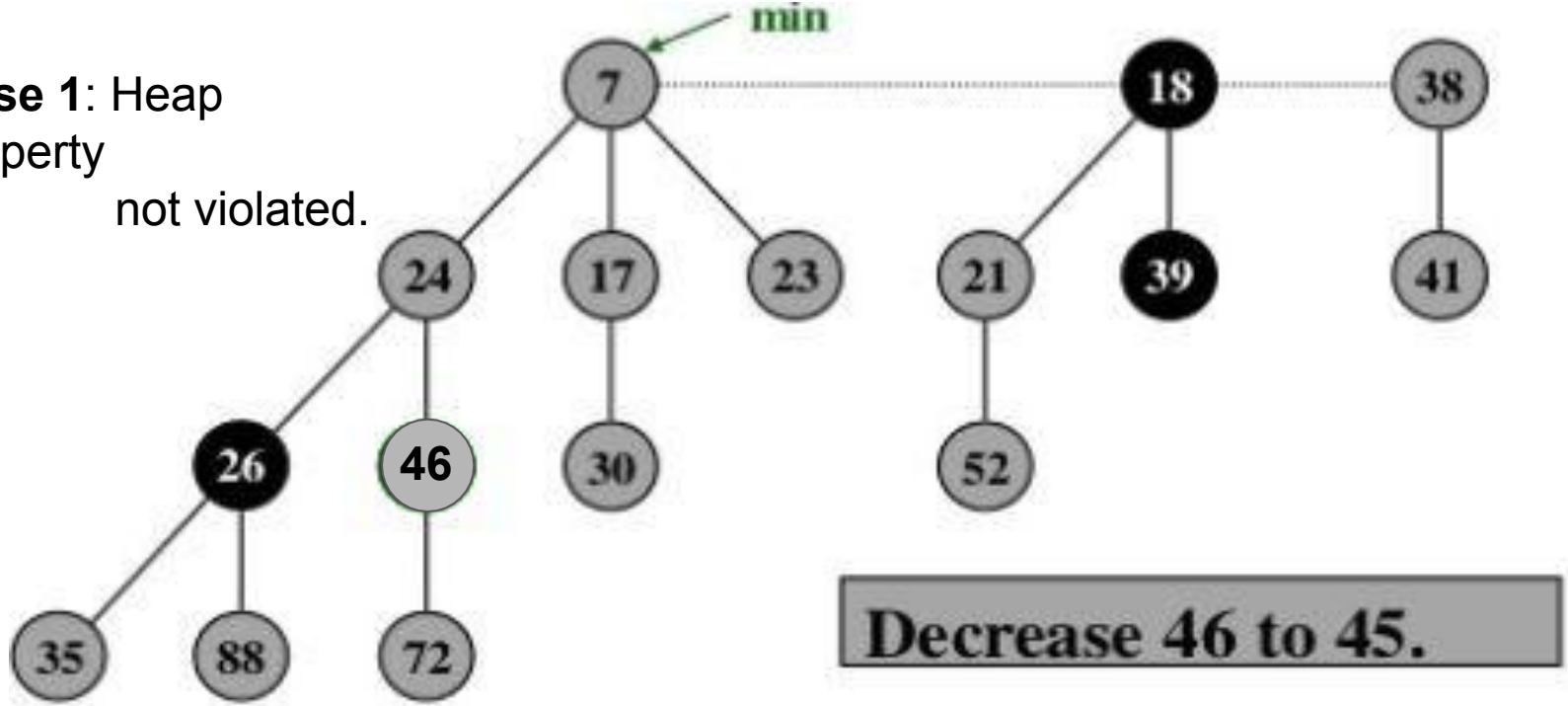
Heap H2



Heap H

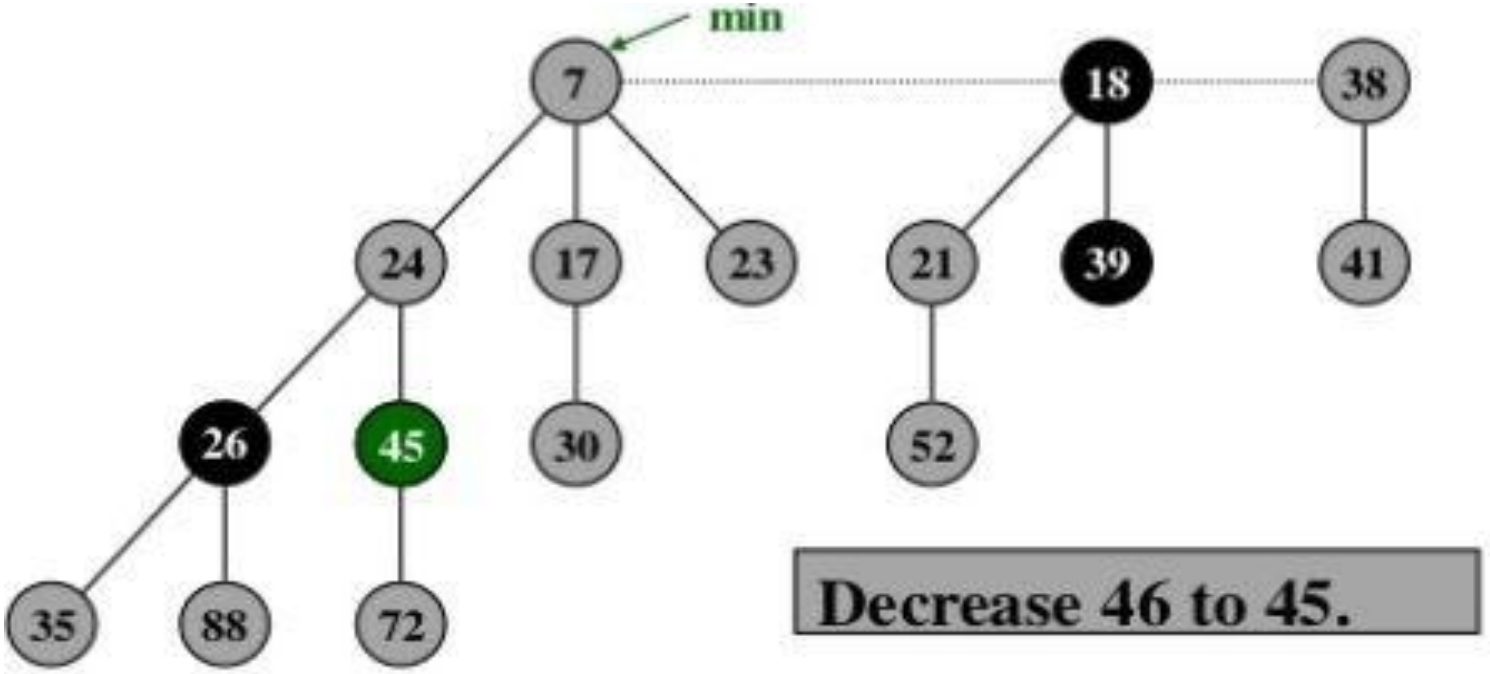
Decrease key (Case 1)

Case 1: Heap property not violated.



Decrease key (Case 1)-Contd

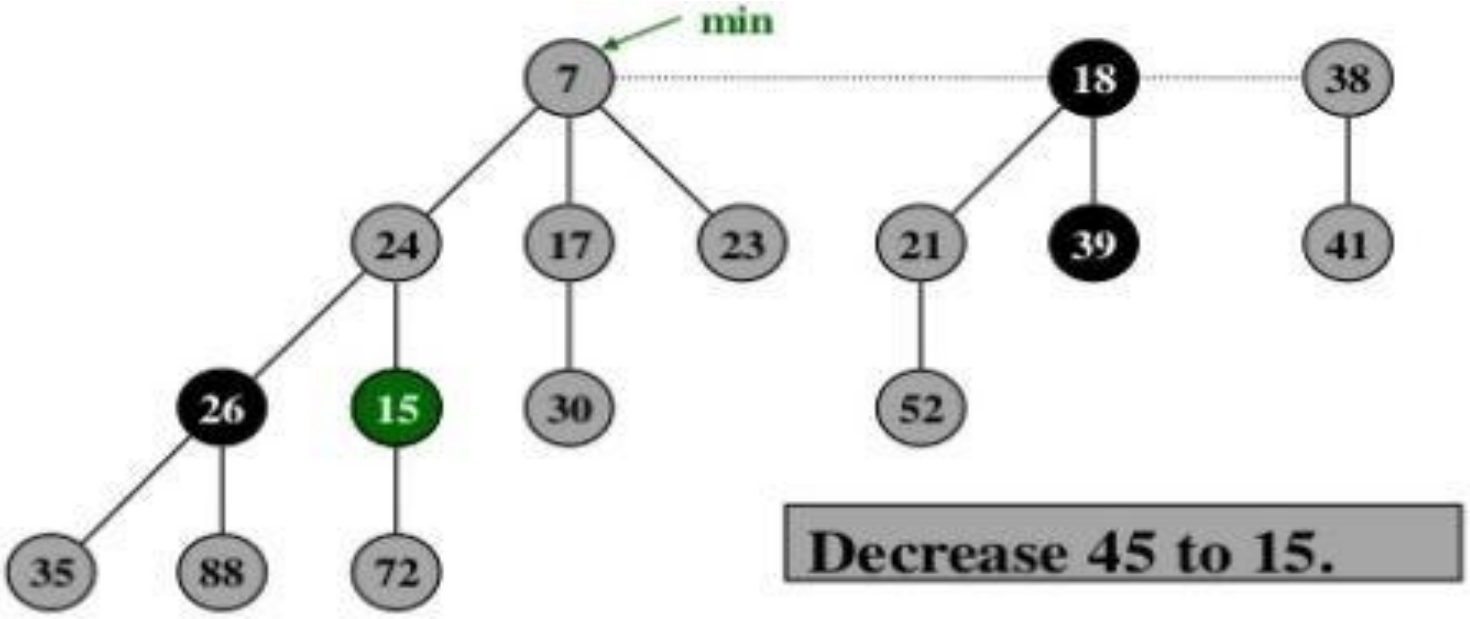
Decrease the value of the node directly, update min if needed



Decrease key (Case 2)

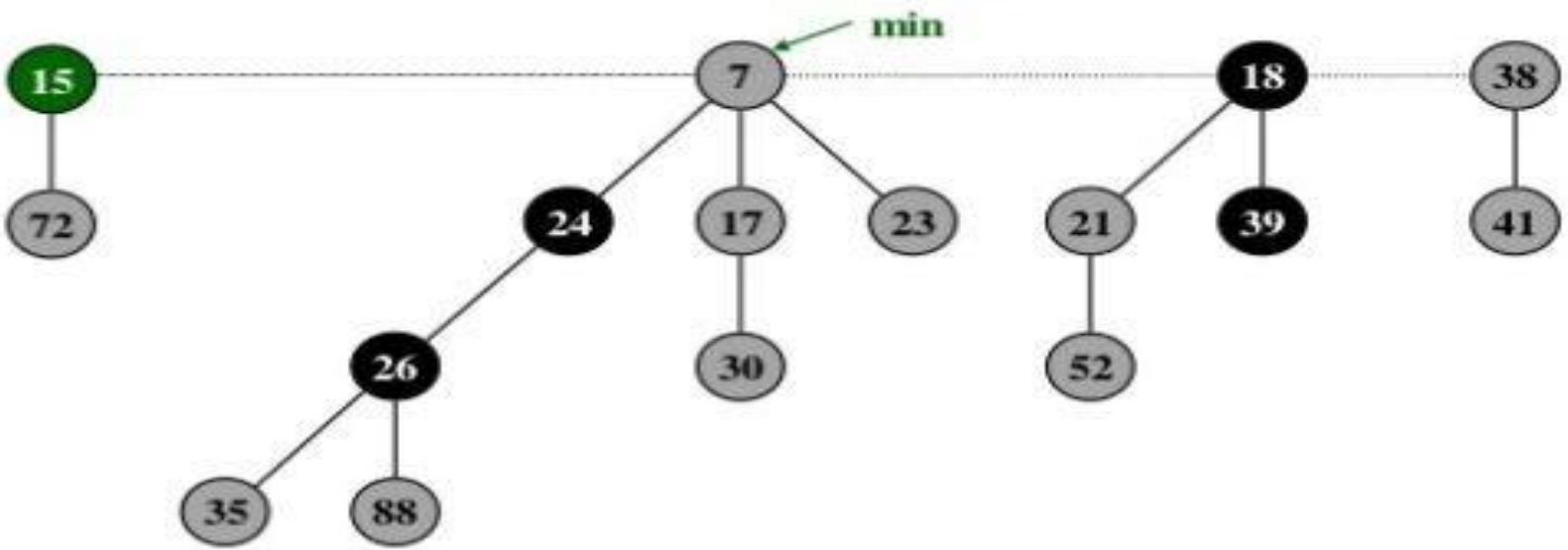
Case 2: Parent of x is unmarked

1. Decrease value of x



Decrease key (Case 2)

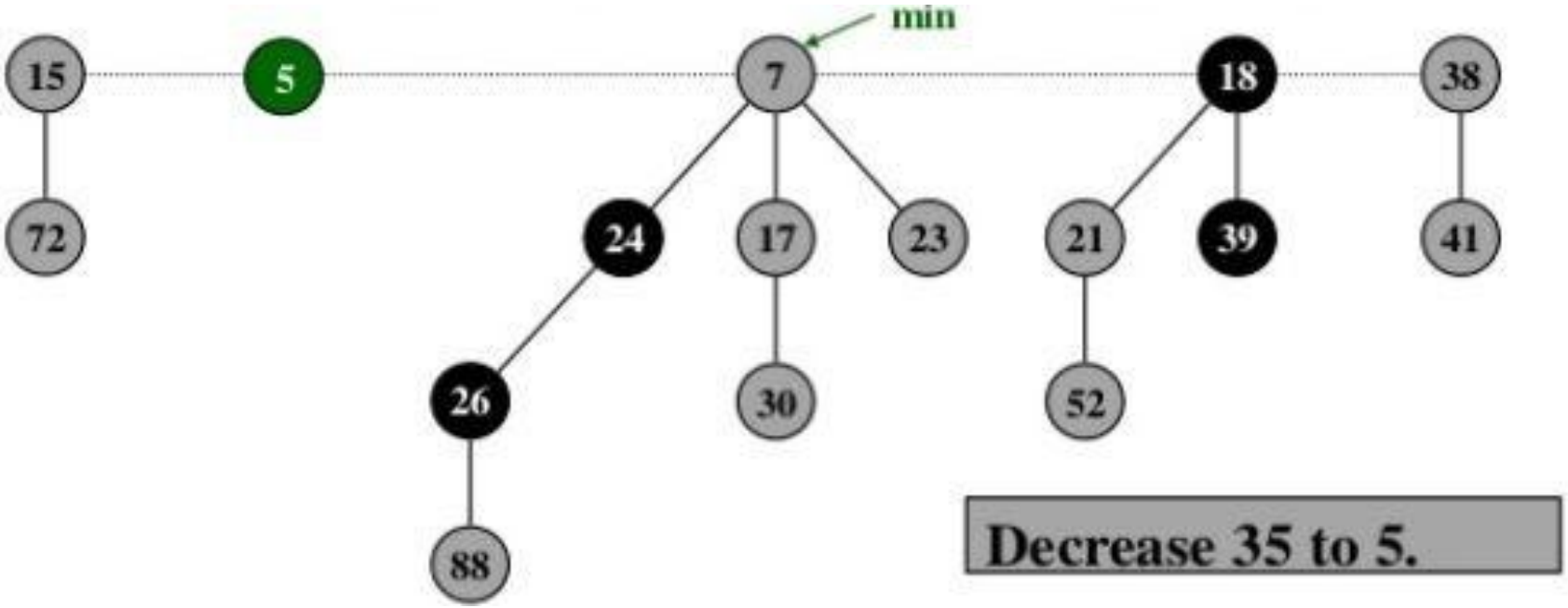
2. Cut off link between x and parent (cascading)
3. Mark parent if it is not the root
4. Add x to root list, update min



Decrease key (Case 3)

Case 3: Parent is marked

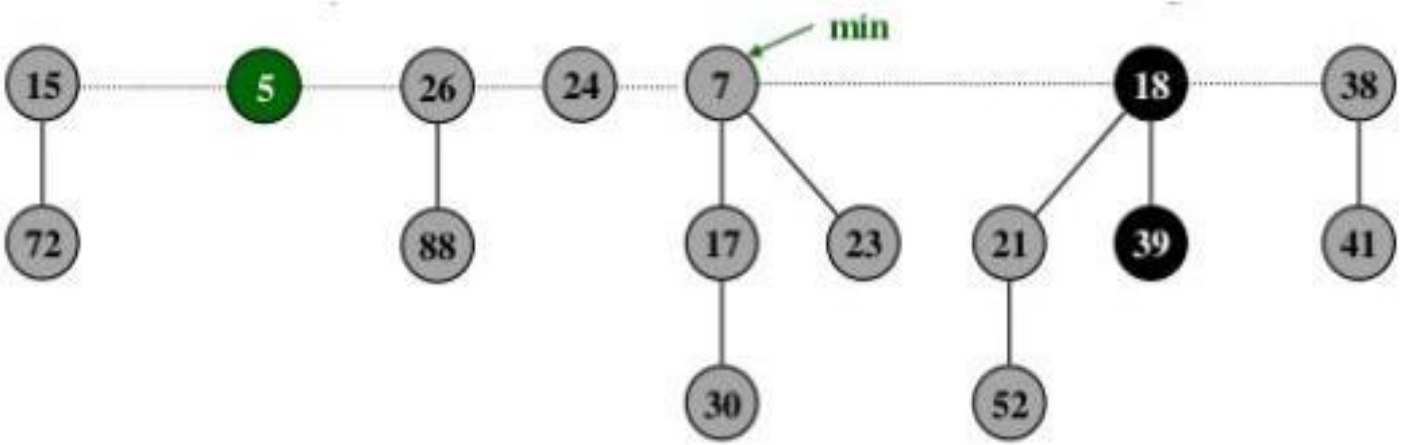
1. Decrease value of x
2. Cut off link between x and parent p[x] and add x to root list



Decrease key (Case 3)

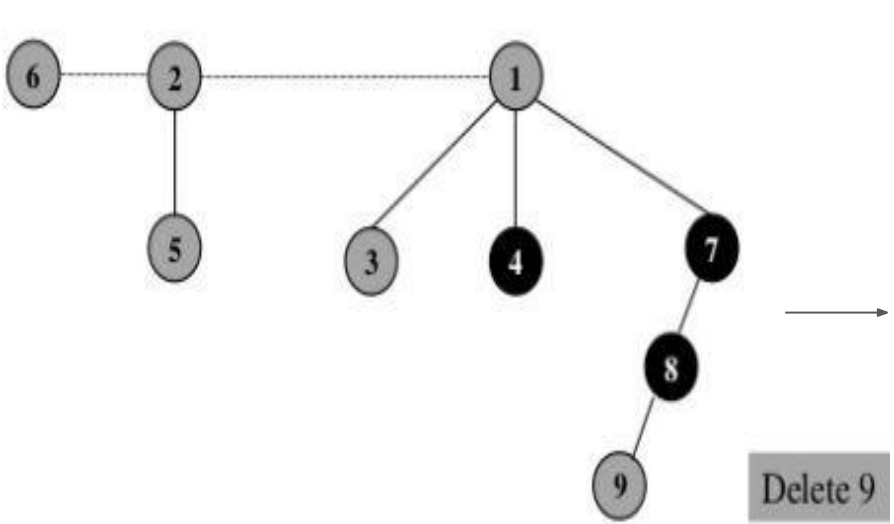
3. Cut off link between p[x] and parent p[p[x]], unmark and add p[x] to root list

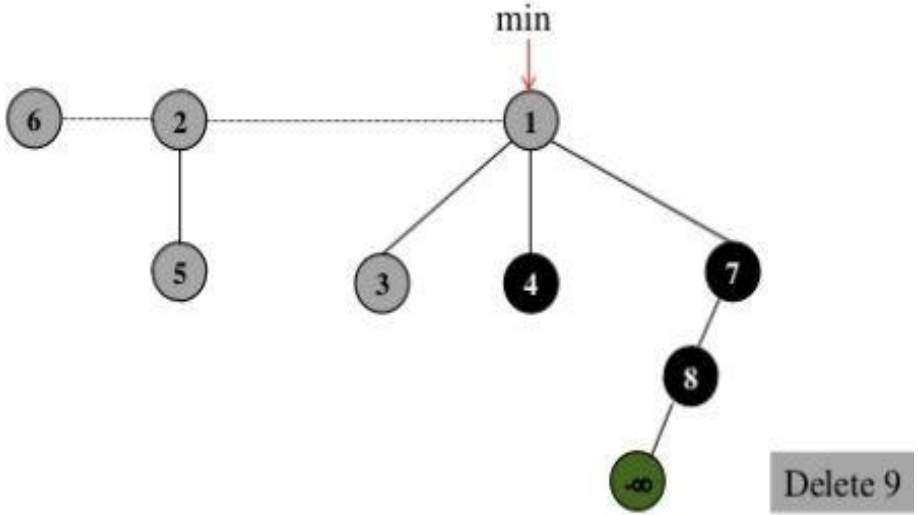
    a. If p[p[x]] is unmarked, mark it if it is not the root
    b. If p[p[x]] is marked, cut off p[p[x]], unmark and repeat



Delete node

- Decrease value of x to -∞
- Extract min or delete min element in heap

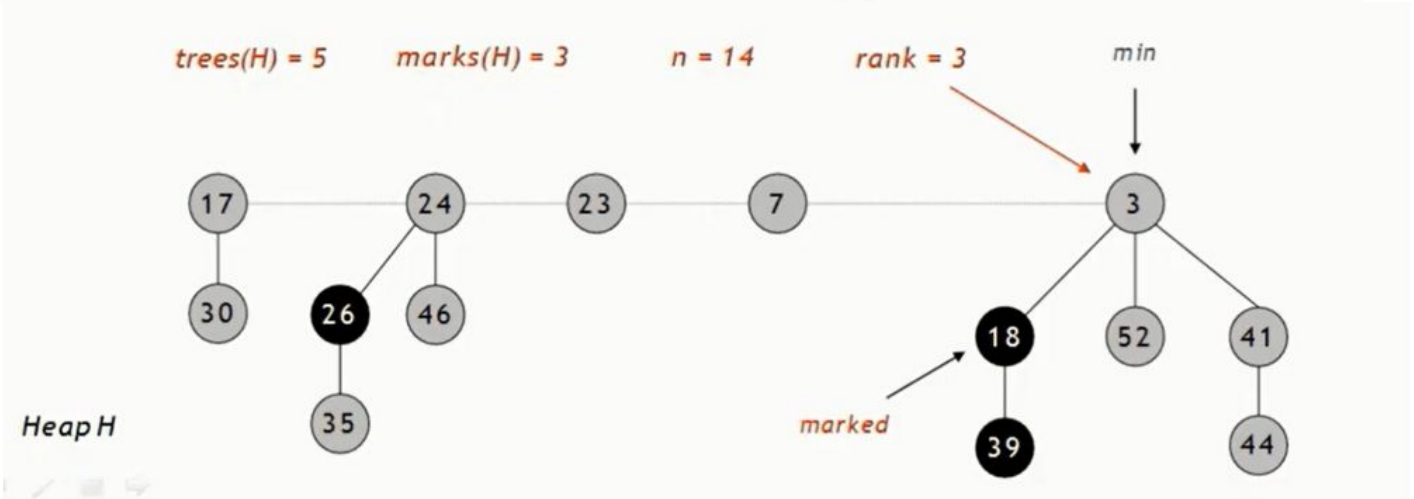




Fibonacci Heap -Notation

Notation.

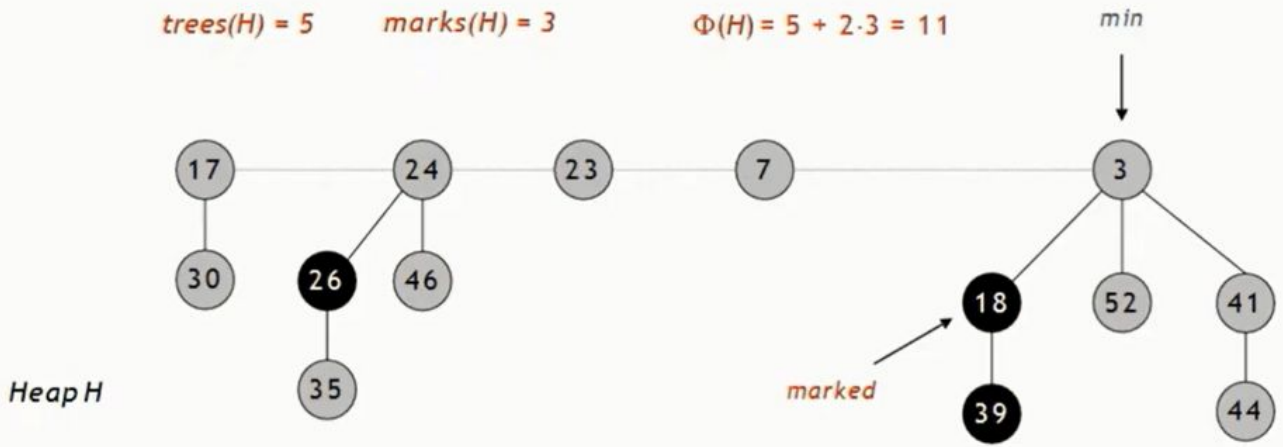
- n = number of nodes in heap.
- $rank(x)$ = number of children of node x.
- $rank(H)$ = max rank of any node in heap H.
- $trees(H)$ = number of trees in heap H.
- $marks(H)$ = number of marked nodes in heap H.



Fibonacci Heap –Potential Function

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

potential of heap H

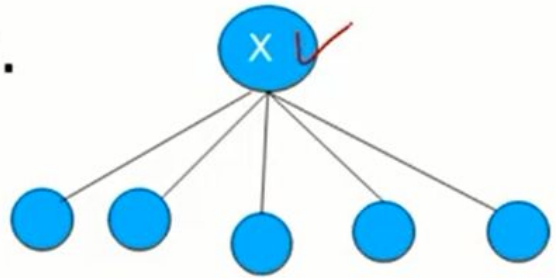


Fibonacci Heap –Analysis

Theorem:

Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

For $i>1$, $\text{Rank}(y_i) \geq i-2$.



Thank You