


Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap [†]
make-heap	1	1	1	1
is-empty	1	1	1	1
insert	1	$\log n$	$\log n$	1
delete-min	n	$\log n$	$\log n$	$\log n$
decrease-key	n	$\log n$	$\log n$	1
delete	n	$\log n$	$\log n$	$\log n$
union	1	n	$\log n$	1
find-min	n	1	$\log n$	1

[†] n = number of elements in priority queue

n = number of elements in priority queue

Fibonacci Heaps

- Similar to binomial heaps, but less rigid structure.
- Binomial heap: **eagerly** consolidate trees after each insert.
- Fibonacci heap: **lazily** defer consolidation until next delete-min.



The diagram illustrates the structure of a Fibonacci heap, showing a sequence of trees that demonstrate the lazy consolidation property. The trees are:

- A single node.
- A root node with one child.
- A root node with two children.
- A root node with three children, where the left child has two children of its own.

This sequence shows how nodes are added without immediately consolidating them into larger trees, which is characteristic of the lazy approach in Fibonacci heaps.

- # Fibonacci Heaps: Structure
- Set of **heap-ordered trees**.
 - Maintain pointer to minimum element.
 - Set of marked nodes.
-
- The diagram illustrates the structure of a Fibonacci heap, which is a collection of heap-ordered trees. At the top, four red arrows labeled "roots" point to the root nodes of four separate trees. The first tree has root 17 and child 30. The second tree has root 24 and children 26 and 46; node 26 has child 35. The third tree has root 23. The fourth tree has root 7. To the right, a separate tree with root 3 and children 16, 52, and 41 is shown, with 16 having child 39 and 41 having child 44. This tree is highlighted with a blue background and labeled "heap-ordered tree". The entire structure is labeled "Heap H" at the bottom left.

Diagram illustrating the transformation of a binary tree into a heap-ordered tree. The left part shows a binary tree with root 17 and children 30, 24, 23, and 7. Node 24 has children 26 and 46, and 26 has child 35. The right part shows the same tree after a series of rotations, resulting in a heap-ordered tree with root 3 and children 18, 52, and 41. Node 18 has child 39, and 41 has child 44. Red arrows indicate the sequence of rotations: 17 to 24, 24 to 23, 23 to 7, and 7 to 3.

Fibonacci Heaps: Structure

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.**
- Set of marked nodes.

find-min takes $O(1)$ time

```
graph TD
    17((17)) --- 24((24))
    24 --- 30((30))
    24 --- 26((26))
    26 --- 35((35))
    24 --- 23((23))
    23 --- 7((7))
    7 --- 3((3))
    3 --- 18((18))
    3 --- 52((52))
    3 --- 41((41))
    18 --- 39((39))
    41 --- 44((44))
```

Heap H

-

Fibonacci Heaps: Structure

- Fibonacci heap.
- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

use to keep heaps flat (stay tuned)

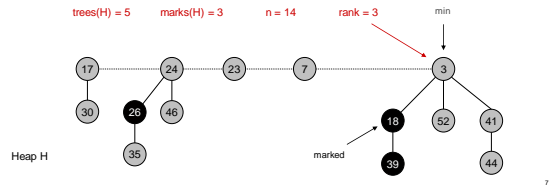
```
graph TD; 3((3)) -- min --> 19((19)); 3 --> 52((52)); 3 --> 41((41)); 19 --> 38((38)); 19 --> 39((39)); 24((24)) --> 17((17)); 24 --> 28((28)); 24 --> 46((46)); 28 --> 35((35)); 23((23)) --> 7((7)); 7 --> 3;
```

Heap H

-
- Heap H

Fibonacci Heaps: Notation

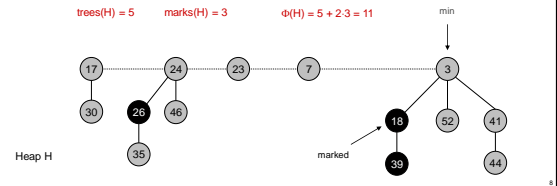
- n = number of nodes in heap.
- $\text{rank}(x)$ = number of children of node x .
- $\text{rank}(H)$ = max rank of any node in heap H .
- $\text{trees}(H)$ = number of trees in heap H .
- $\text{marks}(H)$ = number of marked nodes in heap H .
- The marking step in the Fibonacci heap allows the data structure to count how many children have been lost so far.
- An unmarked node has lost no children, and a marked node has lost one child.
- Once a marked node loses another child, it has lost two children and thus needs to be moved back to the root list for reprocessing.



Fibonacci Heaps: Potential Function

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

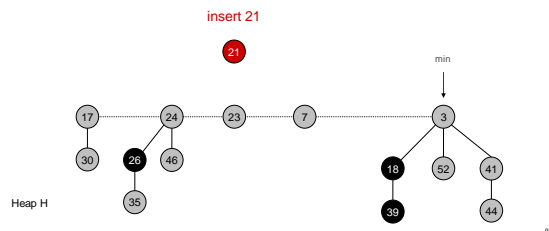
potential of heap H



Fibonacci Heaps: Insert

Insert.

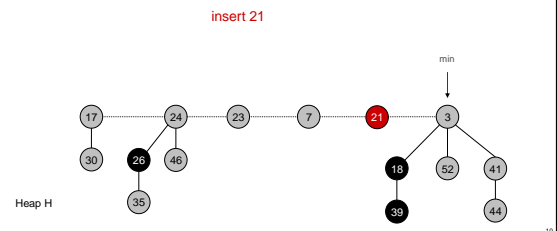
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).



Fibonacci Heaps: Insert

Insert.

- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

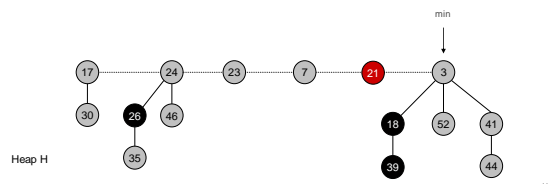


Fibonacci Heaps: Insert Analysis

Actual cost. $O(1)$ Change in potential. $+1$ Amortized cost. $O(1)$

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

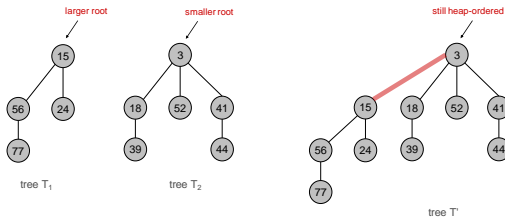
potential of heap H



Delete Min

Linking Operation

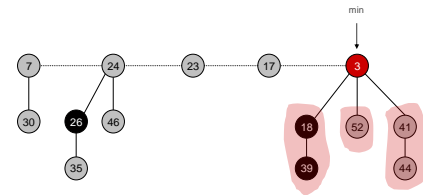
Linking operation. Make larger root be a child of smaller root.



Fibonacci Heaps: Delete Min

Delete min.

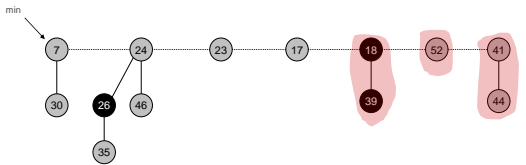
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

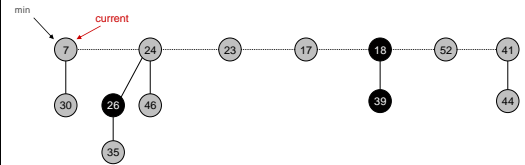


Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

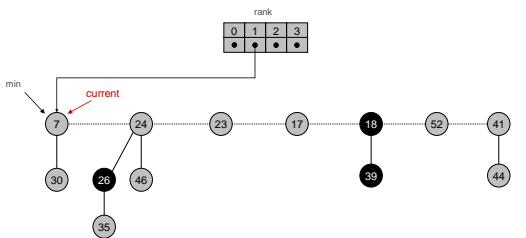
$\text{rank}(x) = \text{number of children of node } x.$



Fibonacci Heaps: Delete Min

Delete min.

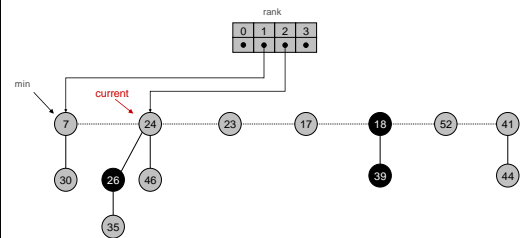
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

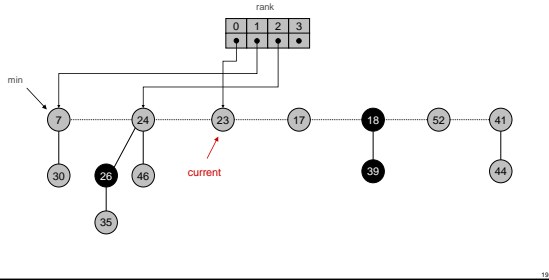
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

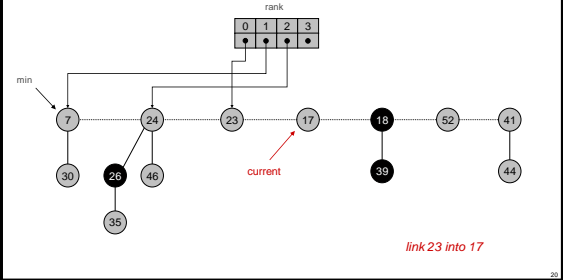
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

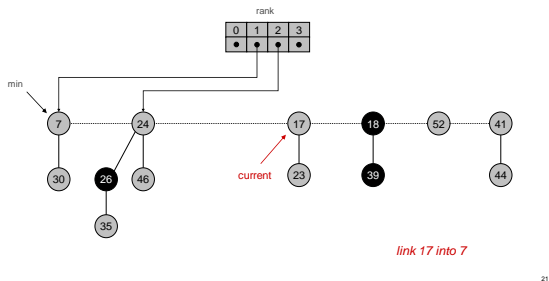
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

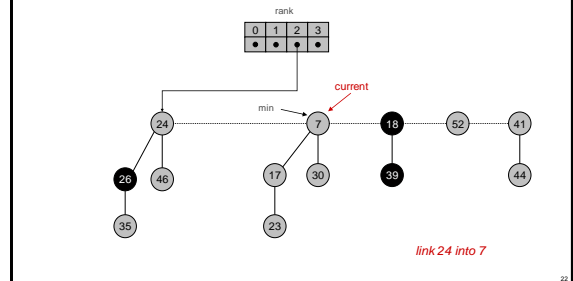
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

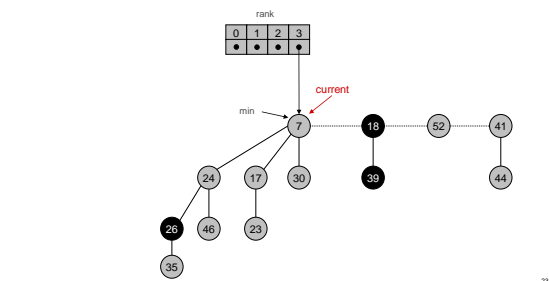
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

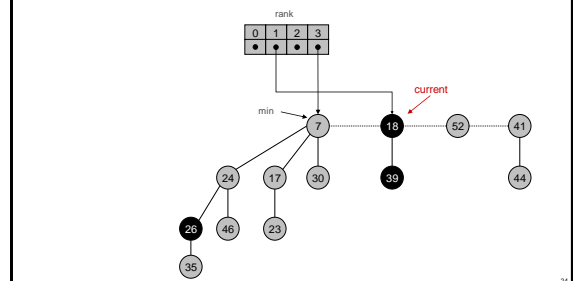
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

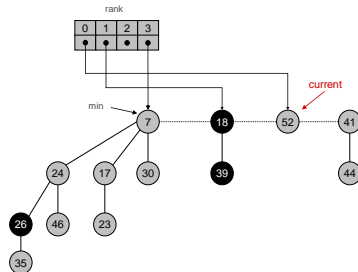
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

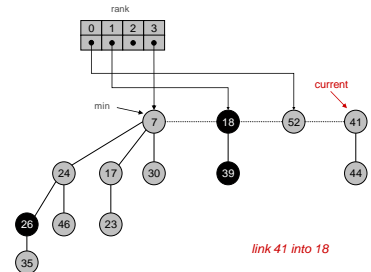
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

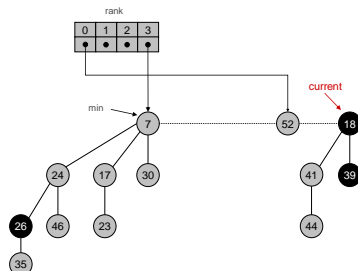
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

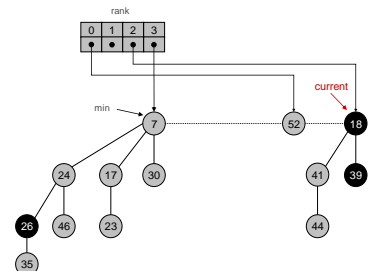
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

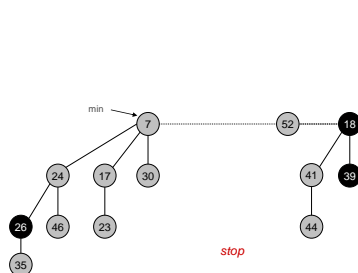
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

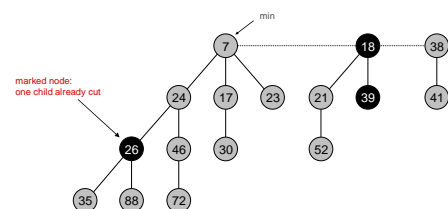
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Decrease Key

Intuition for decreasing the key of node x.

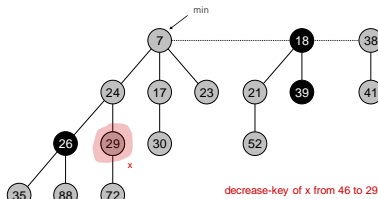
- If heap-order is not violated, just decrease the key of x.
- Otherwise, cut tree rooted at x and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

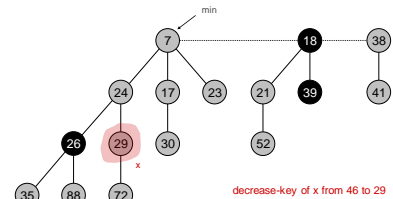
- Decrease key of x .
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

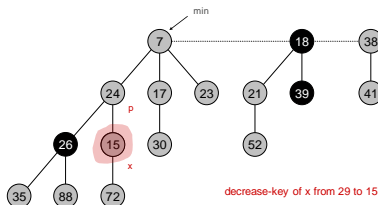
- Decrease key of x .
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

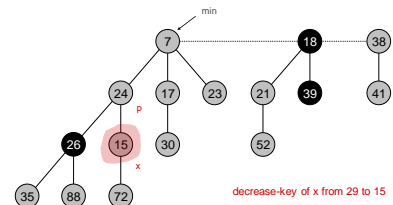
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

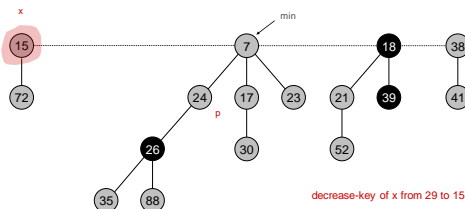
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

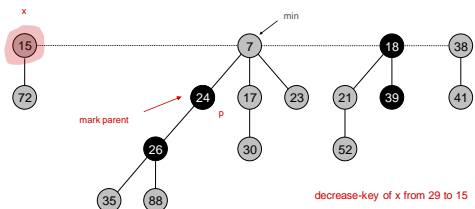
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

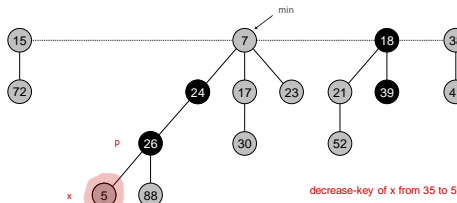
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

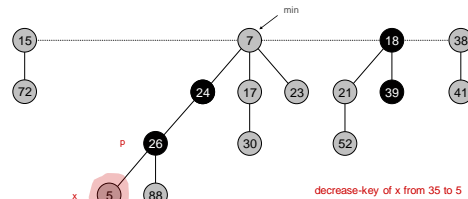
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

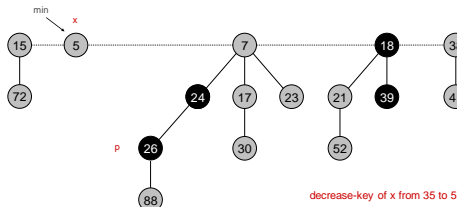
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

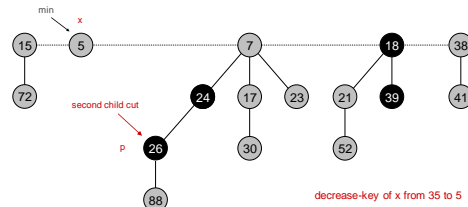
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

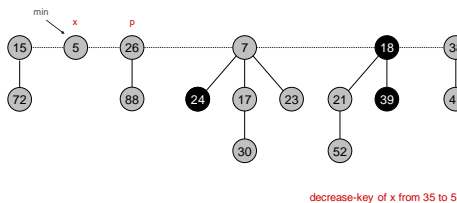
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

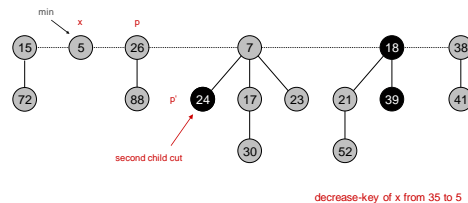
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

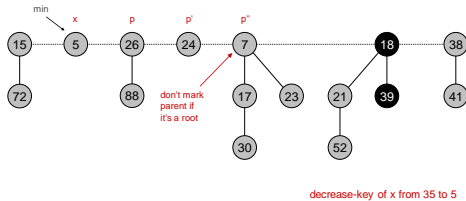
- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it; Otherwise, cut p , meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).



Fibonacci Numbers: Exponential Growth

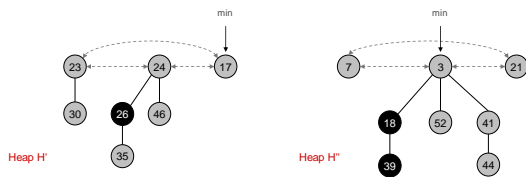
The Fibonacci sequence is: 1, 2, 3, 5, 8, 13, 21, ...

$$F_k = \begin{cases} 1 & \text{if } k = 0 \\ 2 & \text{if } k = 1 \\ F_{k-1} + F_{k-2} & \text{if } k \geq 2 \end{cases} \quad \text{slightly non-standard definition}$$

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

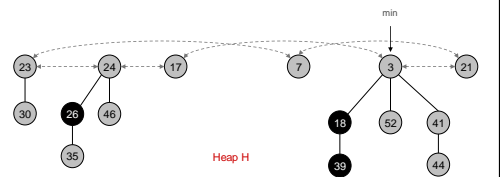
Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

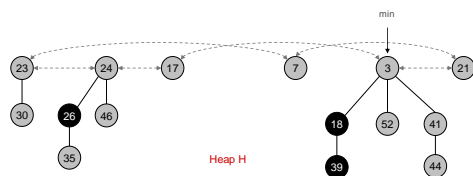
Actual cost. $O(1)$

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

potential function

Change in potential. 0

Amortized cost. $O(1)$



Priority Queues Performance Cost Summary

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap [†]
make-heap	1	1	1	1
is-empty	1	1	1	1
insert	1	$\log n$	$\log n$	1
delete-min	n	$\log n$	$\log n$	$\log n$
decrease-key	n	$\log n$	$\log n$	1
delete	n	$\log n$	$\log n$	$\log n$
union	1	n	$\log n$	1
find-min	n	1	$\log n$	1

n = number of elements in priority queue