Algorithms and Data Structures (CSci 115)

California State University Fresno

College of Science and Mathematics

Department of Computer Science

H. Cecotti

Learning outcomes

- **Fibonacci heaps**
  - Definitions
  - Methods
    - Creation
    - ExtractMin
    - DecreaseKey

Definitions

- Fibonacci heap (Fredman & Tarjan, 1987)
- Data structure
  - For **priority queue operations**
  - With a dual purpose
    - Supports a set of operations that constitutes a “mergeable heap”
    - Several Fibonacci-heap operations run in **constant** amortized time
      - → It is well suited for applications that invoke these operations frequently.
  - Desirable when the number of Extract-min and Delete operations is **small** relative to the number of other operations performed.
- A collection of trees satisfying the minimum-heap property
    - The key of a node is greater than or equal to the key of its parent

Main functions

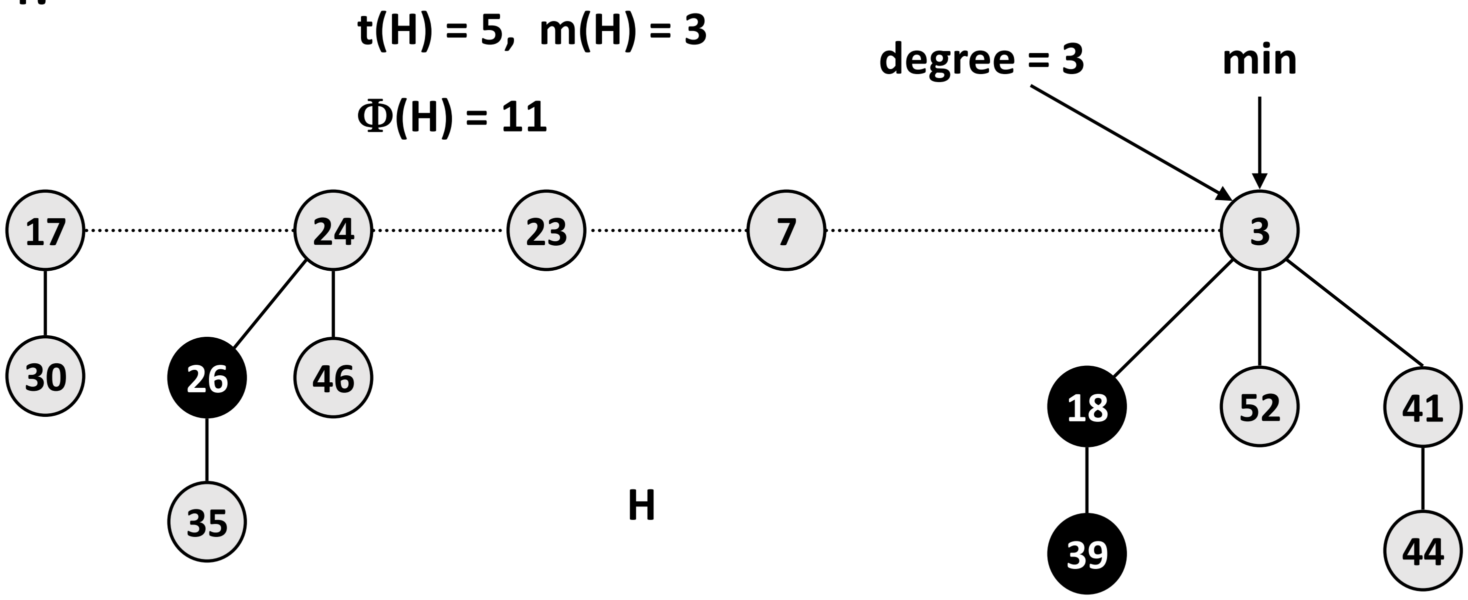
- Another efficient data structure
  - Not trivial to implement
    - Like B-Trees, Red Black Trees …
- 2 key operations
  - ExtractMin
  - DecreaseKey
-

Definitions

- Collection of rooted trees that are min-heap ordered
- Node
  - Pointer to parent (p)
  - Pointer to any of the children (child)
    - Children are linked together: circular **double-chained list**: child list
    - Each child has 2 pointers: **Left** and **Right**
    - Special case: y is an only child → y.left == y.right == y
  - **Degree**: Number of children in the child list
  - **Mark**
    - Tells if a node x lost a child since the last time x was made the child of another node
    - New created nodes: unmarked
    - Unmarked when it is made the child of another node
- Double chained list:
  - Insert in the list: O(1)
  - Concatenate 2 lists: O(1)
- Roots of all the trees in a Fibonacci heap = double chained list

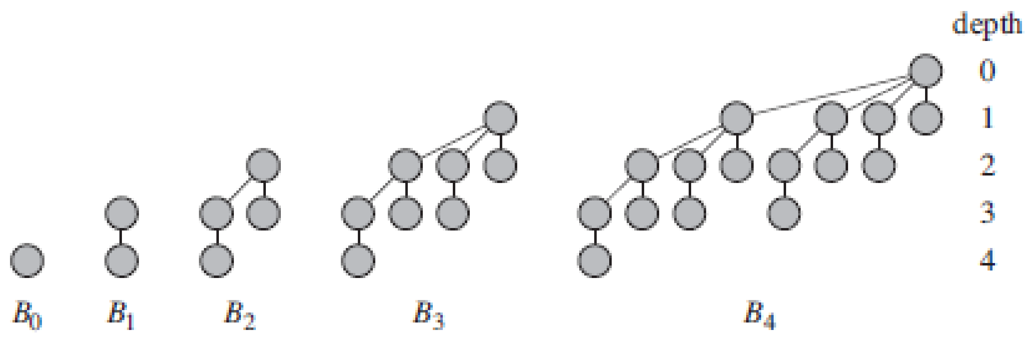
Definitions

- **Maximum degree**
  - ➢D(n) on the maximum degree of any node in an n-node Fibonacci heap
- t(H) = # trees in the root list of H.
- m(H) = # **marked** nodes in H.
- **Potential** function
  - ➢$\Phi(H) = t(H) + 2m(H)$



Definitions

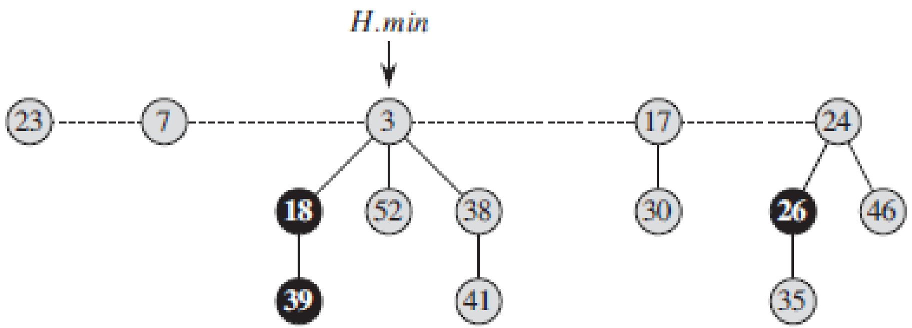
- **Binomial heap**
  - It is a binary heap with quick merging of 2 heaps
  - With these properties
    1. Each node has a key
    2. Each binomial tree obeys the min-heap property
    3. For any non-negative int k, there is **at most** 1 binomial tree whose root is of degree k



Example

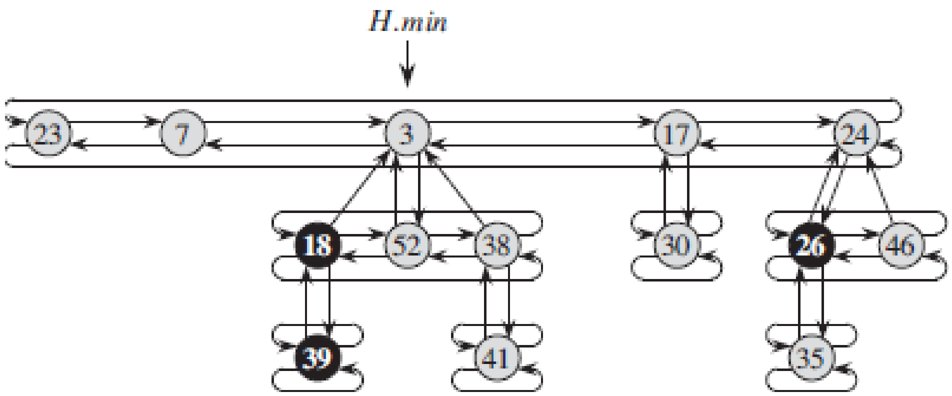
- .

- 5 min-heap-ordered trees and 14 nodes
- dashed line = root list
- minimum node of the heap = node containing the key 3.
- Black nodes are **marked**.
- **Potential (Φ)** of this particular Fibonacci heap is 5+2*3=11.



Representation showing pointers:

- **Parent** (up arrows)
- **Child** (down arrows)
- **Left** and **Right** children (sideways arrows).



Rationale

- Insert a node
  - Adding it to the root list: O(1) ☺
  - Start with an empty Fibonacci heap (H), insert k nodes →a root list of k nodes ☺
- The trade-off:
  - **Extract-Min** operation on H after removing the node that H.*min* points to
  - → Look through **each** of the remaining k-1 nodes in the root list to find the new minimum node
- As long as we have to go through the **entire** root list during the Extract-Min operation
  - → consolidate nodes into **min-heap-ordered trees**
  - Why? to reduce the size of the root list.
- After **Extract-Min** operation
  - each node in the root list has a degree that is **unique** within the root list
    - → a root list of size **at most** D(n)+ 1. (max degree + 1)

Methods

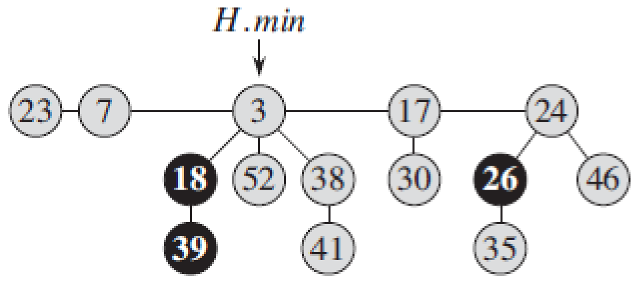
- Create a **new heap**
  - → O(1)
  - Returns the Fibonacci heap object H where
    - H.n=0
    - H.min=NIL
      - no trees in H
    - Φ(H)=0
      - Because t(H)=0 and m(H)=0

Methods

- Insert a node x in H (O(1))
  - Create a new singleton tree.
  - Add to left of min pointer.
  - Update min pointer.
- Pseudo-code:

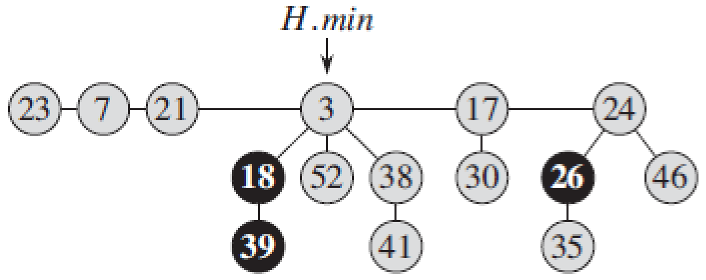
```
FIB-HEAP-INSERT(H, x)
x.degree = 0
x.p = NIL
x.child = NIL
x.mark = FALSE
if H.min == NIL
    create a root list for H containing just x
    H.min = x
else insert x into H's root list
    if x.key < H.min.key
        H.min = x
H.n = H.n + 1
```

Before



Insert 21

After



Find the minimum

- Find the minimum node
  - Given **directly** by H.min
    - Direct in O(1)
  - Potential of H does not change →
    - Amortized cost of find minimum node = O(1)

Union

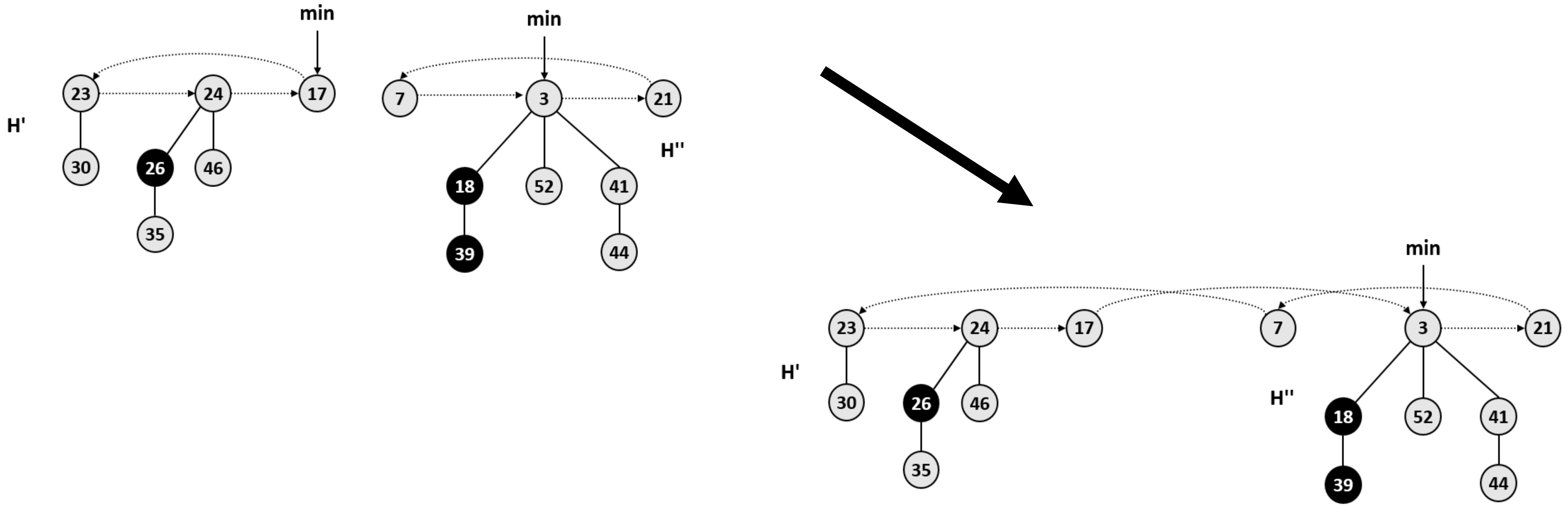
- Union
  - Concatenate 2 Fibonacci heaps (O(1))
  - Change in potential
    - $\Phi(H)=0$
  - Pseudo-code:

FIB-HEAP-UNION(H_1, H_2)

$H = $ MAKE-FIB-HEAP()
$H.min = H_1.min$
concatenate the root list of H_2 with the root list of H
if $(H_1.min == \text{NIL})$ or $(H_2.min \neq \text{NIL}$ and $H_2.min.key < H_1.min.key)$
    $H.min = H_2.min$
$H.n = H_1.n + H_2.n$
return H

Union - example

- .



Extract the minimum node

- **Main idea**
  - Making a root out of each of the minimum node's children + removing the minimum node from the root list.
  - Consolidate the root list by **linking roots of equal degree**
    - Until **at most** 1 root remains of each degree.
- Pseudo-code:

```
FIB-HEAP-EXTRACT-MIN(H)
z = H.min
if z ≠ NIL
    for each child x of z
        add x to the root list of H
        x.p = NIL
    remove z from the root list of H
    if z == z.right
        H.min = NIL
    else H.min = z.right
        CONSOLIDATE(H)
    H.n = H.n − 1
return z
```

Consolidation

- Consolidating the root list
  - Repeat {
    - Find 2 roots x and y in the root list with the **same** degree
      - let x.*key* y.*key*
    - Link y to x:
      - Remove y from the root list
      - Make y a child of x by calling the FIB-HEAP-LINK procedure.
        - Increments the attribute x.*degree*
        - **Clears the mark on y**

    }
  - Until (every root in the root list has a **distinct** degree value)

Consolidation

- Pseudo-code
  - Allocate and initialize the array A by making each entry NIL
  - Main **for** loop
    - Processes each root w in the root list.
    - As we link roots together:
      - w may be linked to some other node and no longer be a root!
    - Yet, w is always in a tree rooted at some node x
      - that may or may not be w itself!
      - as we want **at most** 1 root with each degree
      - we look in A
      - **If** (it contains a root y with the same degree as x)
      - **then** we link the roots x and y but guaranteeing that x remains a root after linking.
      - → we link y to x after first exchanging the pointers to the 2 roots if y.key < x.key.
    - After we link y to x, x.degree+=1
    - … so we continue this process linking x and another root whose degree equals x’s new degree
  - until no other root that was processed has the same degree as x

Consolidation

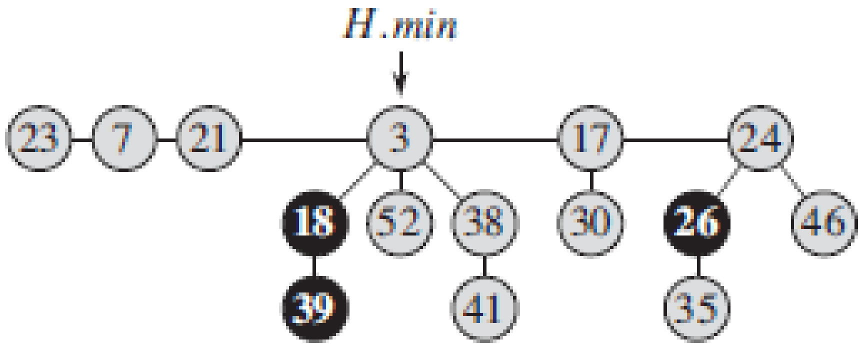
- Pseudo code
  - While loop invariant
    - d=x.degree
  - d=x.degree=y.degree
    - Link x and y
    - Increment x.degree
    - No increment for y.degree
  - While
    - There is a root with the same degree as x

```
CONSOLIDATE(H)
let A[0 . . D(H.n)] be a new array
for i = 0 to D(H.n)
    A[i] = NIL
for each node w in the root list of H
    x = w
    d = x.degree
    while A[d] ≠ NIL
        y = A[d]            // another node with the same degree as x
        if x.key > y.key
            exchange x with y
        FIB-HEAP-LINK(H, y, x)
        A[d] = NIL
        d = d + 1
    A[d] = x
H.min = NIL
for i = 0 to D(H.n)
    if A[i] ≠ NIL
        if H.min == NIL
            create a root list for H containing just A[i]
            H.min = A[i]
        else insert A[i] into H's root list
            if A[i].key < H.min.key
                H.min = A[i]
```

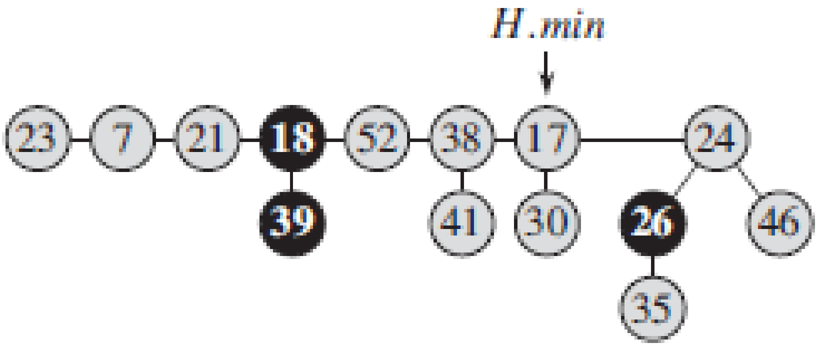
FIB-HEAP-EXTRACT-MIN

- 1

A Fibonacci heap H

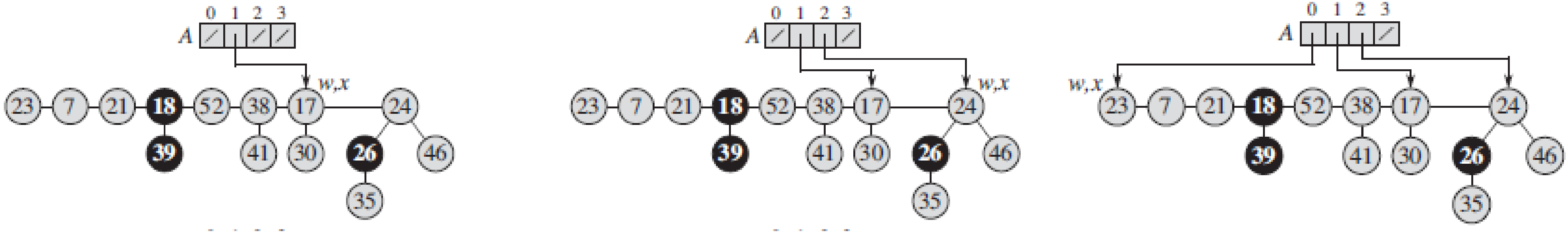


After removing the minimum node z from the root list and adding its children to the root list



FIB-HEAP-EXTRACT-MIN

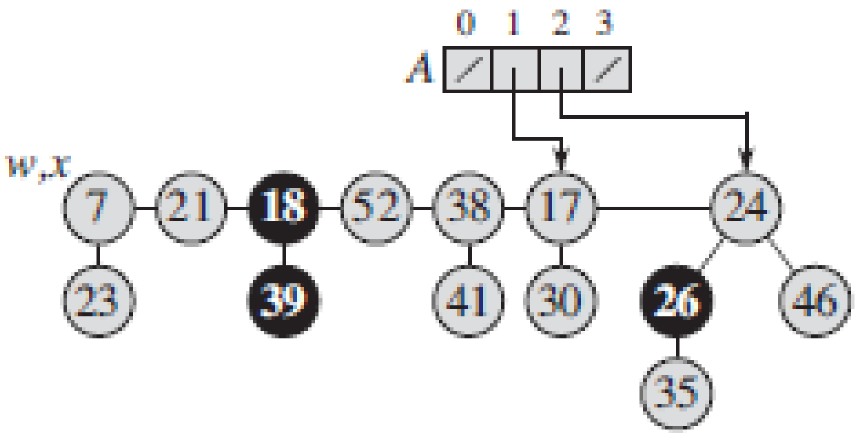
- The array A and the trees after each of the first 3 iterations of the main **for** loop of Consolidate.
  - The index of the Array == Degree of the node in the root list
  - Processes the root list by starting at the node pointed to by H.min and following ***right* pointers**.
  - Each part shows the values of w and x at the end of an iteration.



FIB-HEAP-EXTRACT-MIN

- FIB-HEAP-EXTRACT-MIN
  - ➢Next iteration of the main **for** loop
    - o Values of w and x shown at the end of each iteration of the **while** loop

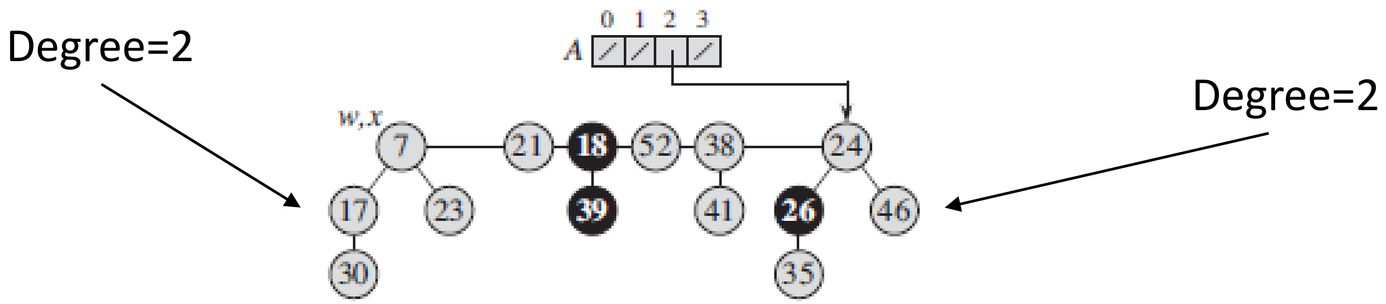
1. Situation after the first time through the **while** loop
    - Node with key 23 has been linked to the node with key 7



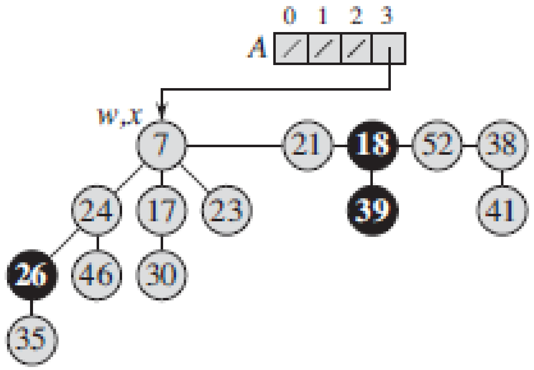
We keep the min heap →
7 on top of 23
Degree 1 for 7
There is already a degree 1 in A (17,30)
→ We don't replace, we attach 17, 30
as a child!

FIB-HEAP-EXTRACT-MIN

2. The node with key 17: linked to the node with key 7, which x still points to.

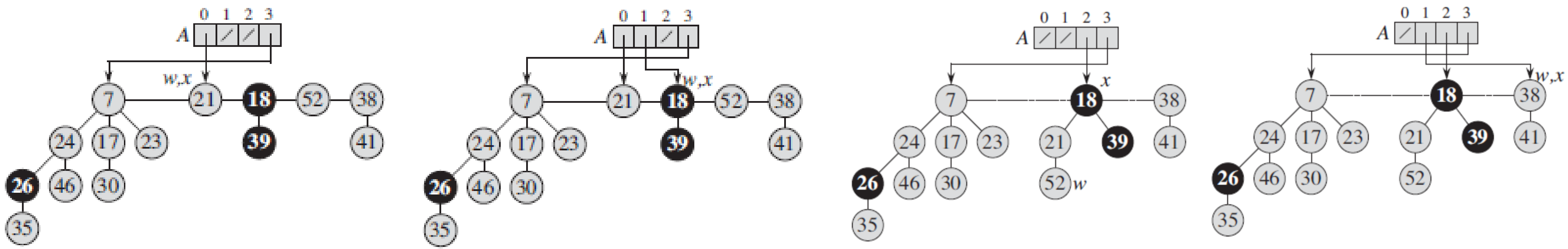


3. The node with key 24: linked to the node with key 7.
   - As no node was previously pointed to by A[3], at the end of the **for** loop iteration,
   - A[3] set to point to the **root** of the resulting tree.



FIB-HEAP-EXTRACT-MIN

➢States after each of the next 4 iterations of the **for** loop



➢H after reconstructing the root list from the array A and determining the new

➢H.min pointer.

Decrease a key and Delete a node

- Mark
  - Guarantees that:
    - **only** delete 1 child for every node
    - "we do not diverse from the binomial tree structure too much"
  - Without deletion every tree in a Fibonacci heap == a binomial tree

Decrease a key and Delete a node

- Decreasing a key
  - **Assumption**: removing a node from a linked list does **not** change any of the attributes in the removed node

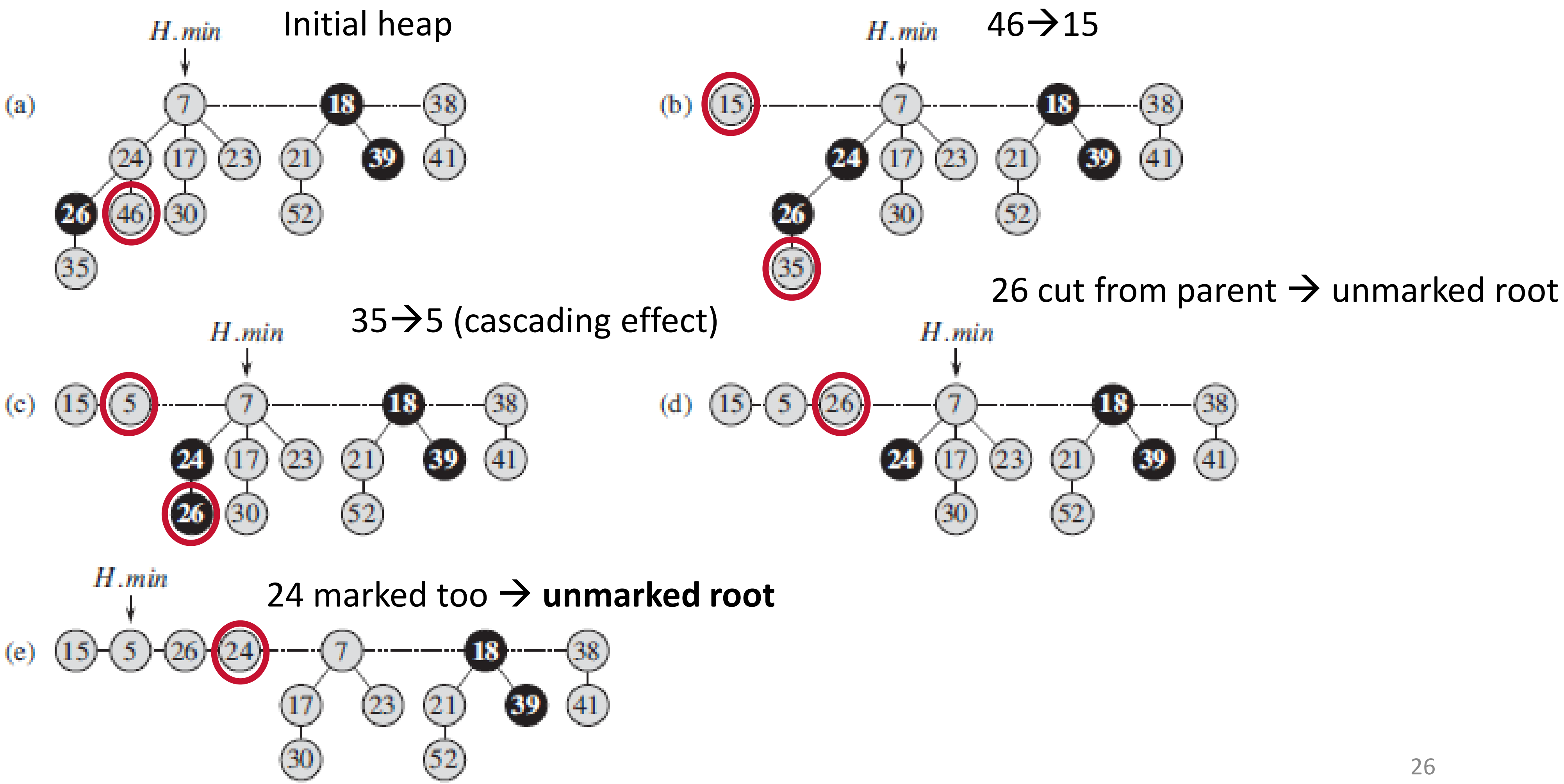
```
FIB-HEAP-DECREASE-KEY(H, x, k)
if k > x.key
    error "new key is greater than current key"
x.key = k
y = x.p
if y ≠ NIL and x.key < y.key
    CUT(H, x, y)
    CASCADING-CUT(H, y)
if x.key < H.min.key
    H.min = x
```

```
CUT(H, x, y)
remove x from the child list of y, decrementing y.degree
add x to the root list of H
x.p = NIL
x.mark = FALSE
```

```
CASCADING-CUT(H, y)
z = y.p
if z ≠ NIL
    if y.mark == FALSE
        y.mark = TRUE
    else CUT(H, y, z)
        CASCADING-CUT(H, z)
```

Fibonacci heap

- .



Delete a node

- Pseudo-code:
  - O(D(n)): amortized time

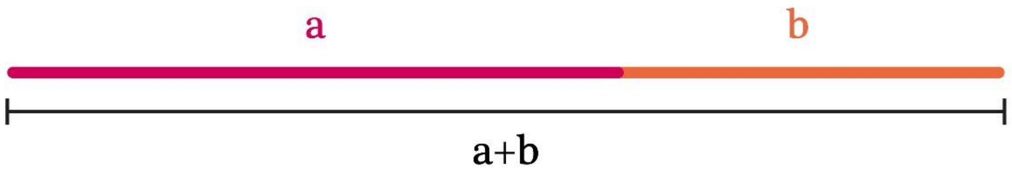
```
FIB-HEAP-DELETE(H, x)
1  FIB-HEAP-DECREASE-KEY(H, x, −∞)
2  FIB-HEAP-EXTRACT-MIN(H)
```

Why Fibonacci

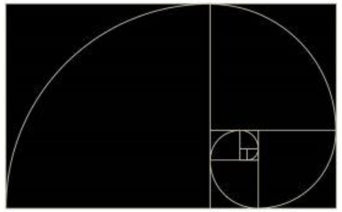
- $\Phi = (1+\sqrt{5})/2 = 1.61803\ldots$ = Golden ratio
- FIB-HEAP-EXTRACT-MIN and FIB-HEAPDELETE: O(log n)
  - → show that the upper bound D(n) on the degree of any node of an n-node Fibonacci heap is O(log n)
  - → show that $D(n) \le \log_{\Phi} n$
- Definitions
  - For each node x within a Fibonacci heap,
    - **size(x)** = number of nodes (**including x**) in the subtree rooted at x.
  - Show that size(x) is **exponential** in x.degree.
    - x.degree is always maintained as an **accurate count** of the degree of x.

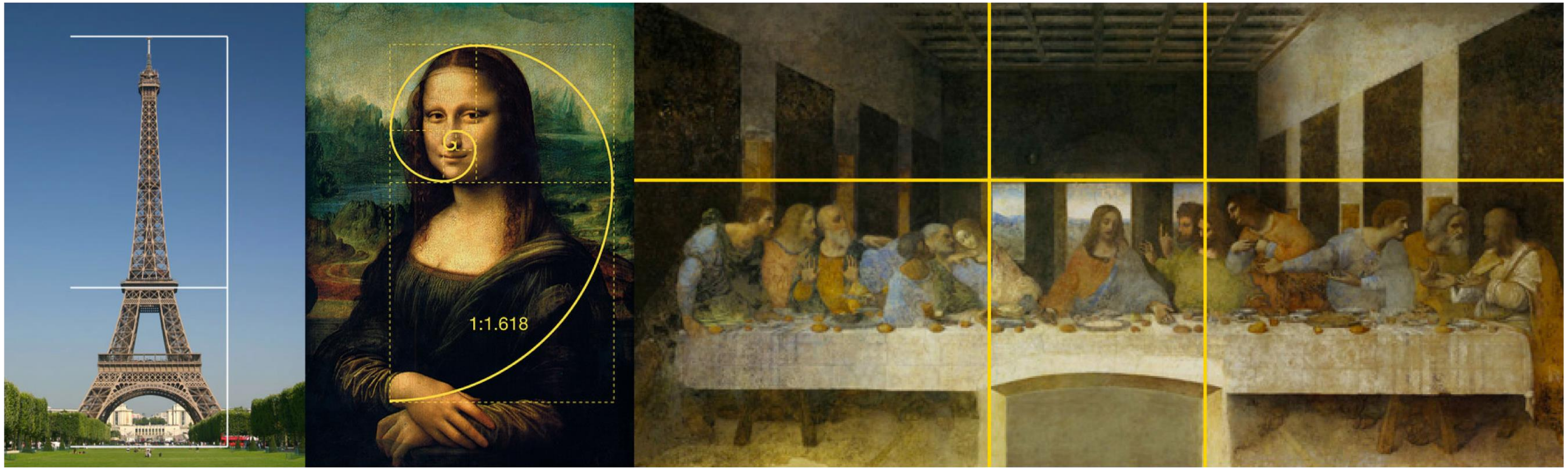
Golden ratio

- Example



$$\frac{a+b}{a} = \frac{a}{b}$$





Lemma

- **Lemma 1**
  - Let x be any node in a Fibonacci heap
    - Suppose that x.*degree*=k.
  - Let $y_1,y_2,\ldots,y_k$: the children of x **in the order** in which they were linked to x
    - from the earliest to the latest **then** y_1.*degree*≥0 and y_i.*degree* ≥ i-2 ∀ i ∊ {2..k}
- **Proof**
  - y1.*degree* ≥ 0 (definition)
  - For i ≥ 2
    - Observe that when y_i was linked to x, all of $y_1,y_2,\ldots,y_{i-1}$ were children of x
    - → we must have had x.*degree* ≥ i-1.
    - Because node y_i is linked to x (through CONSOLIDATE) only if x.*degree*==y_i.*degree*
    - → must have had y_i.*degree*≥i-1 at that time.
    - Since it happened, node y_i has lost at most 1 child
      - Because it would have been cut from x (through CASCADING-CUT) if it had lost 2 children
    - → y_i .*degree* ≥ i-2

Lemma

- **Why is it name Fibonacci?**
  - because …
  - $F_k=0$ if $k=0$, $F_k=1$ if $k=1$, $F_k=F_{k-1}+F_{k-2}$ if $k\geq 2$
- **Lemma 2**:
  - : For all integers $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i \; .$$

  - Proof:
    - By induction
- **Lemma 3**:
  - $\forall$ $k\geq 0$, the $(k+2)^{nd}$ Fibonacci number satisfies $F_{k+2} \geq \Phi^k$
  - Proof
    - By induction

Lemma

- **Lemma 4**
  - Let x be any node in a Fibonacci heap, and let k=x.*degree*
    - Then size(x) $\geq F_{k+2} \geq \Phi^k$ where $\Phi=(1+\sqrt{5})/2$
- **Proof (part 1)**
  - Let s_k denote the minimum possible size of any node of degree k in any Fibonacci heap
  - Trivially, s_0=1 and s_1=2.
    - S_k is at most size(x)
      - because adding children to a node cannot decrease the node's size!
    - the value of s_k increases monotonically with k
  - Consider some node z, in any Fibonacci heap | z.*degree*=k && size(s)=s_k
    - Because $s_k \leq$size(x)
    - Get lower bound on size(x) by computing a lower bound on s_k

Lemma

- **Proof (part 2)**
  - Let $y_1, y_2, \ldots, y_k$: the children of z in the order in which they were linked to z.
  - To bound s_k, we count one for z itself and one for the first child y_1
    - size(y1)≥1
  - :

$$
\begin{aligned}
\text{size}(x) &\geq s_k \\
&\geq 2 + \sum_{i=2}^{k} s_{y_i.degree} \\
&\geq 2 + \sum_{i=2}^{k} s_{i-2}\,,
\end{aligned}
$$

(from Lemma 2 + monotocity of s_k)

Lemma

- **Proof (part 3)**
  - By induction $s_k \geq F_{k+2} \quad \forall\, k \in \mathbb{N}$

$$
\begin{aligned}
s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\
&\geq 2 + \sum_{i=2}^{k} F_i \\
&= 1 + \sum_{i=0}^{k} F_i \\
&= F_{k+2} && \text{Lemma 2} \\
&\geq \phi^k && \text{Lemma 3}
\end{aligned}
$$

  - → **size(x) ≥ F_{k+2}**

Corollary

- The maximum degree D(n) of any node in an n-node Fibonacci heap
  - O(log n)
- Proof
  - Let x be any node in an n-node Fibonacci heap
  - Let k=x.*degree*
  - By Lemma 4 : $n \geq size(x) \geq \Phi^k$
    - using base- Φ logarithms, we get $k \leq \log_\Phi n$
  - → maximum degree D(n) of any node is O(log n)

Conclusion

- Binomial heap:
  - Direct consolidate trees **after each insert**
- Fibonacci heap:
  - Lazily defer consolidation **until next delete-min**
- Complexity

Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

Questions ?

- Reading & Acknowledgement
  - Chapter 19, Fibonacci heaps, Introduction to Algorithms, 3rd Edition
  - Fredman and Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.