Fibonacci Heaps

Ch. 9.4

B-Heaps serve as Special Cases of F-Heaps

Fibonacci heaps

- Time complexity for different operations

If with consolidation
(In textbook, delete "any" has no consolidation.)

	Actual	Amortized
Insert	O(1)	O(1)
Delete min (or max)	O(n)	O(log n)
Meld	O(1)	O(1)
Delete "any"	O(n)	O(log n)
Decrease key (or increase)	O(n)	O(1)

Additional operations

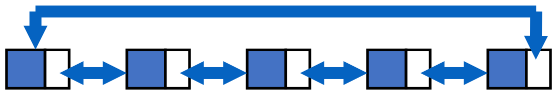
Min Fibonacci heap

- Collection of min trees.
- The min trees need NOT be Binomial trees.
  - Still can be binomial trees.

- No search operation

Node Structure

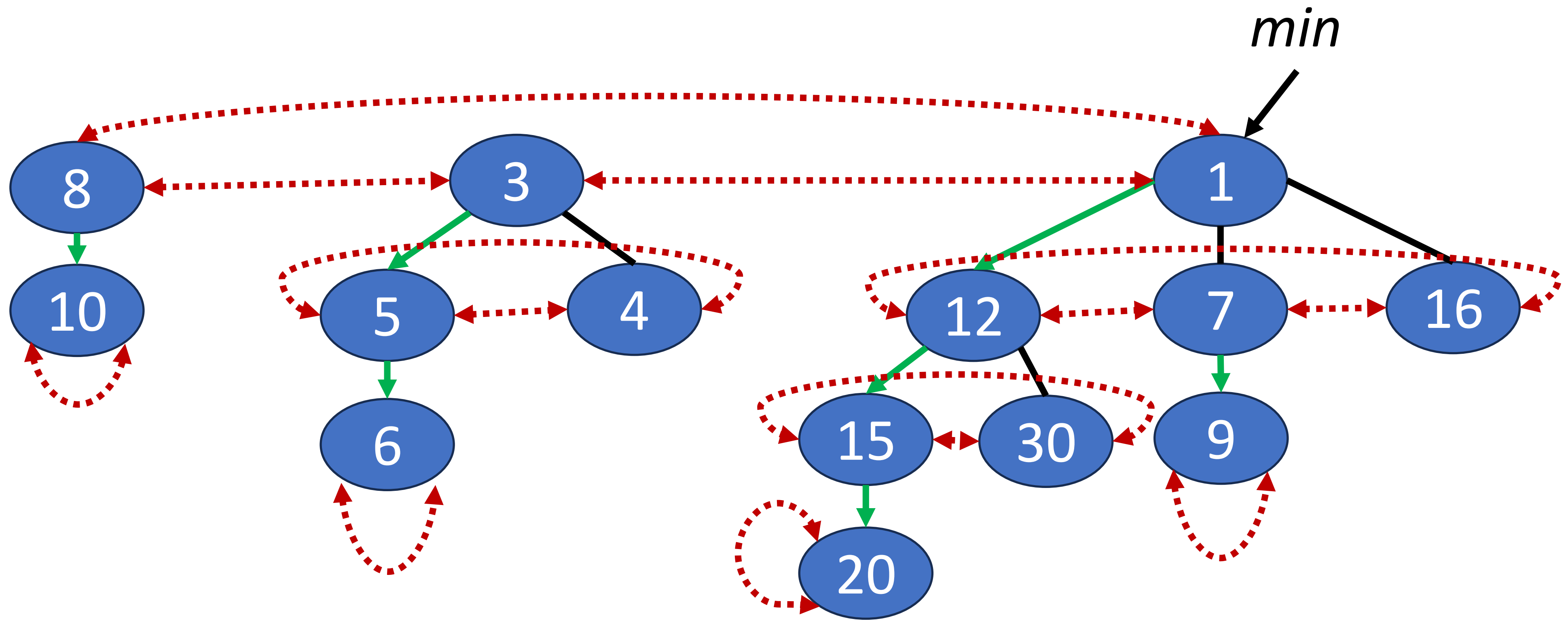
- Data
- Child[degree] pointers
- Left and Right Sibling
  - Used for circular doubly linked list of siblings.



New fields

- Parent
  - Pointer to parent node.
- ChildCut flag
  - True if node has lost a child since it became a child of its current parent.
  - Set to false by remove min, which is the only operation that makes one node a child of another.
  - Undefined for a root node.

Fibonacci heap representation



Note: Parent and ChildCut fields not shown.

Operations

- Insertion: the same as the case of B heaps

  (add the new node to the top-level list, and reset the pointer to min)

- Delete min: the same as the case of B heaps

  (delete the min, perform min-tree joining, and reset the pointer to min)

- Meld: the same as the case of B heaps

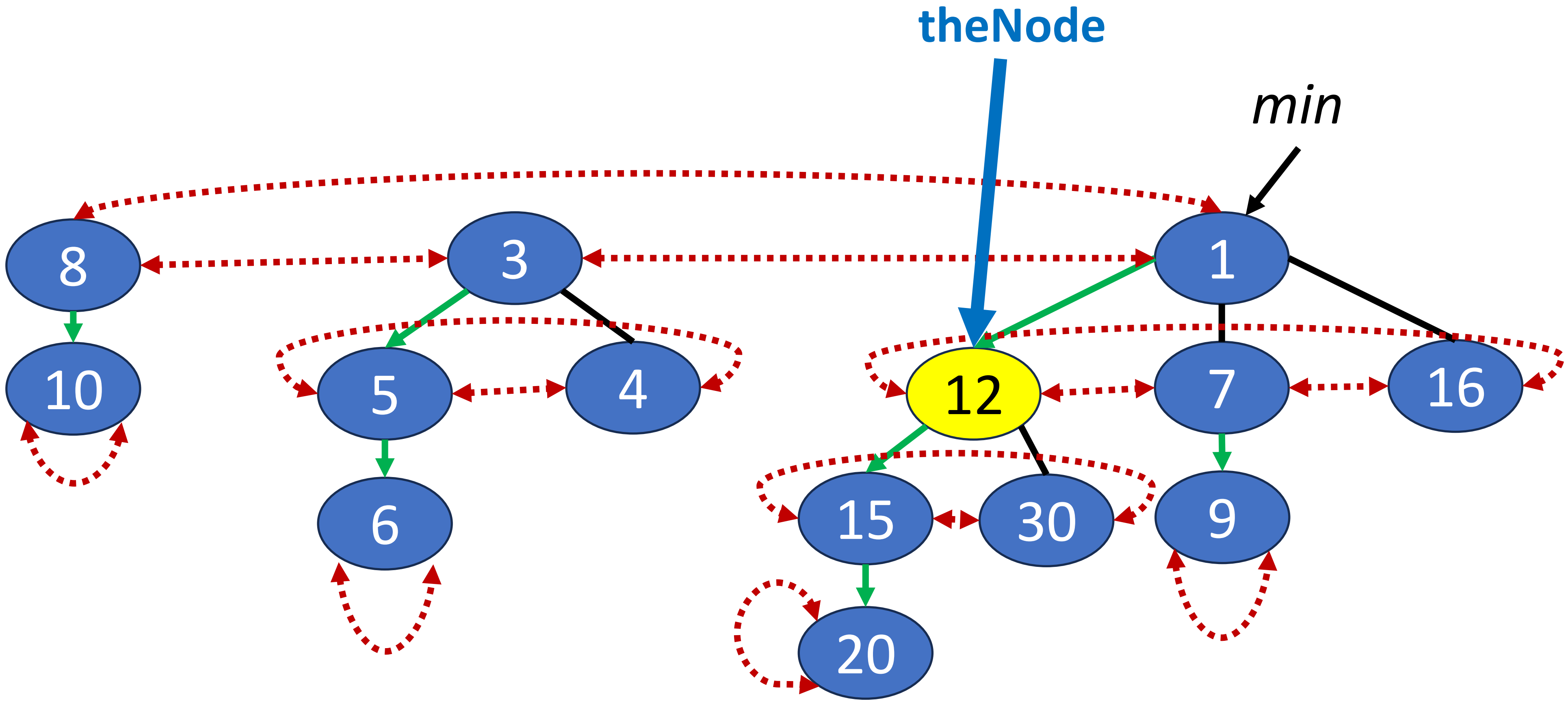
- Delete: delete an arbitrary node

- Decrease key: reduce the key of an arbitrary node

Operation: Delete(theNode)

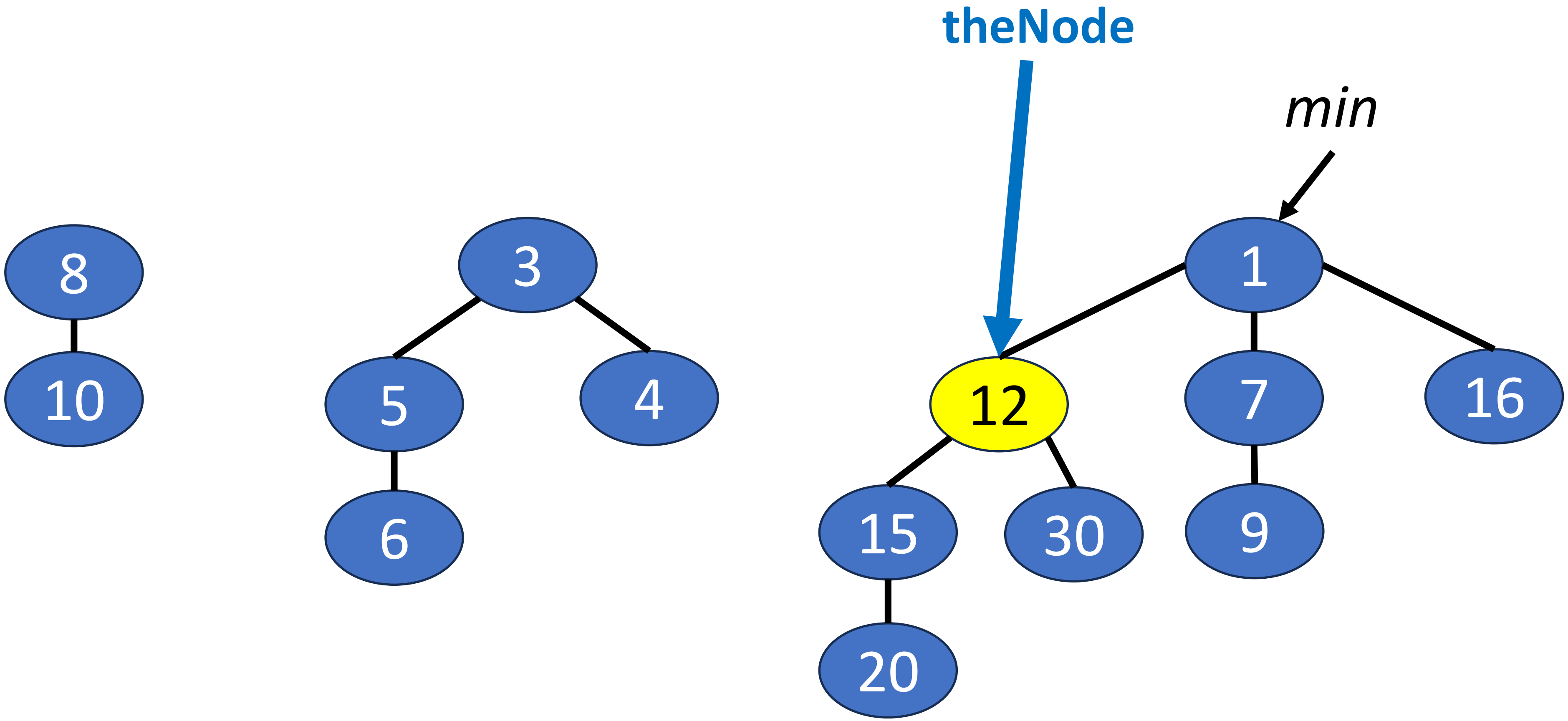
- theNode can be an arbitrary node from the F-heap.
- theNode points to the Fibonacci heap node that contains the element that is to be deleted.
- theNode points to min element → do a delete min.
    - In this case, complexity is the same as that for delete min.

Example: Deletion of 12



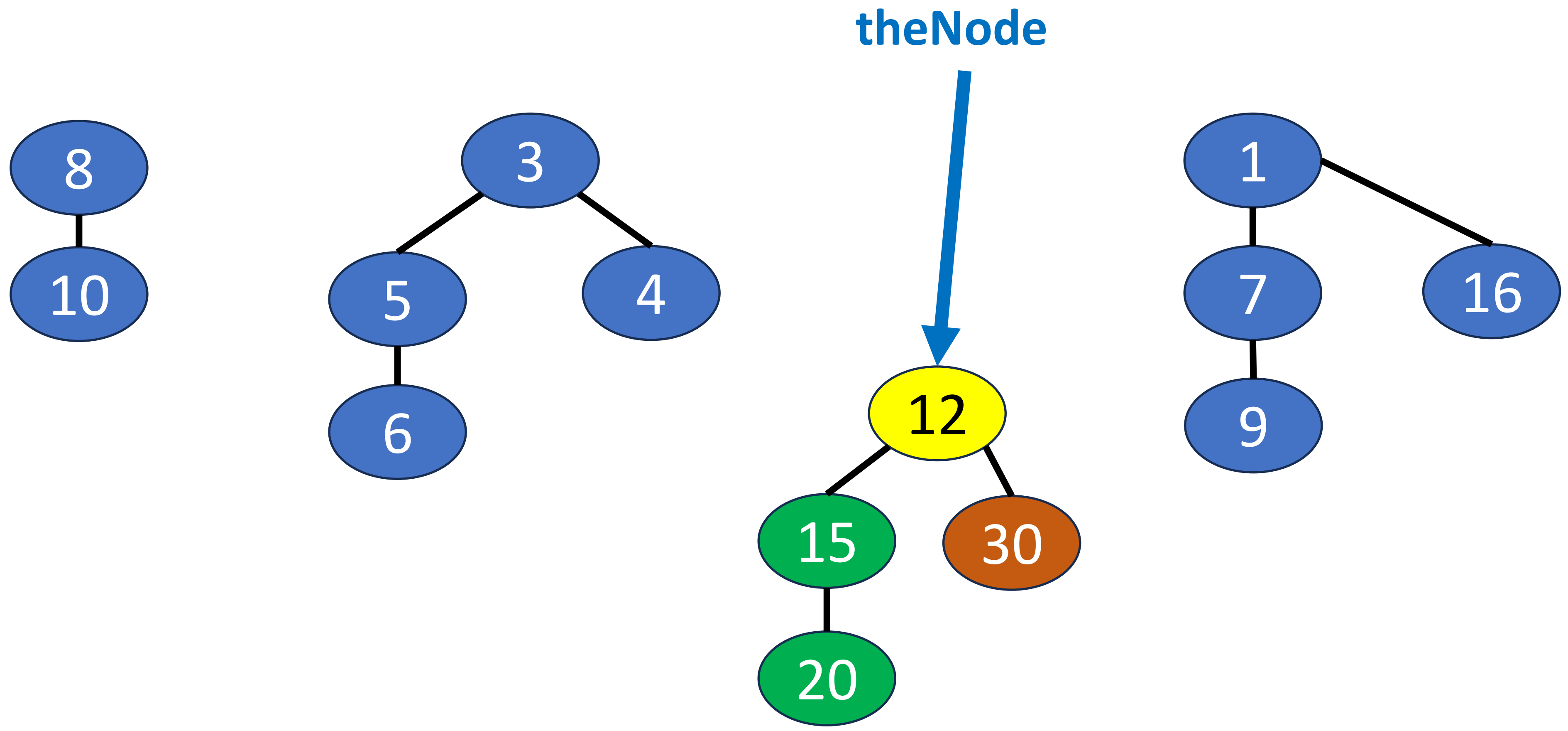
Remove theNode from its doubly linked sibling list.

Example: Deletion of 12



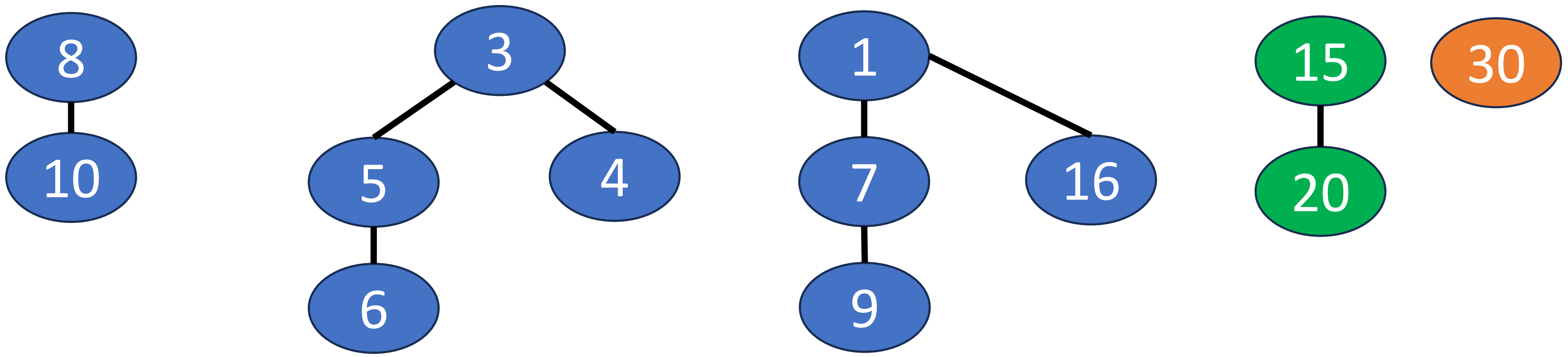
Remove theNode from its doubly linked sibling list.

Example: Deletion of 12



Remove theNode from its doubly linked sibling list.

Example: Deletion of 12



Combine top-level list and children of theNode.

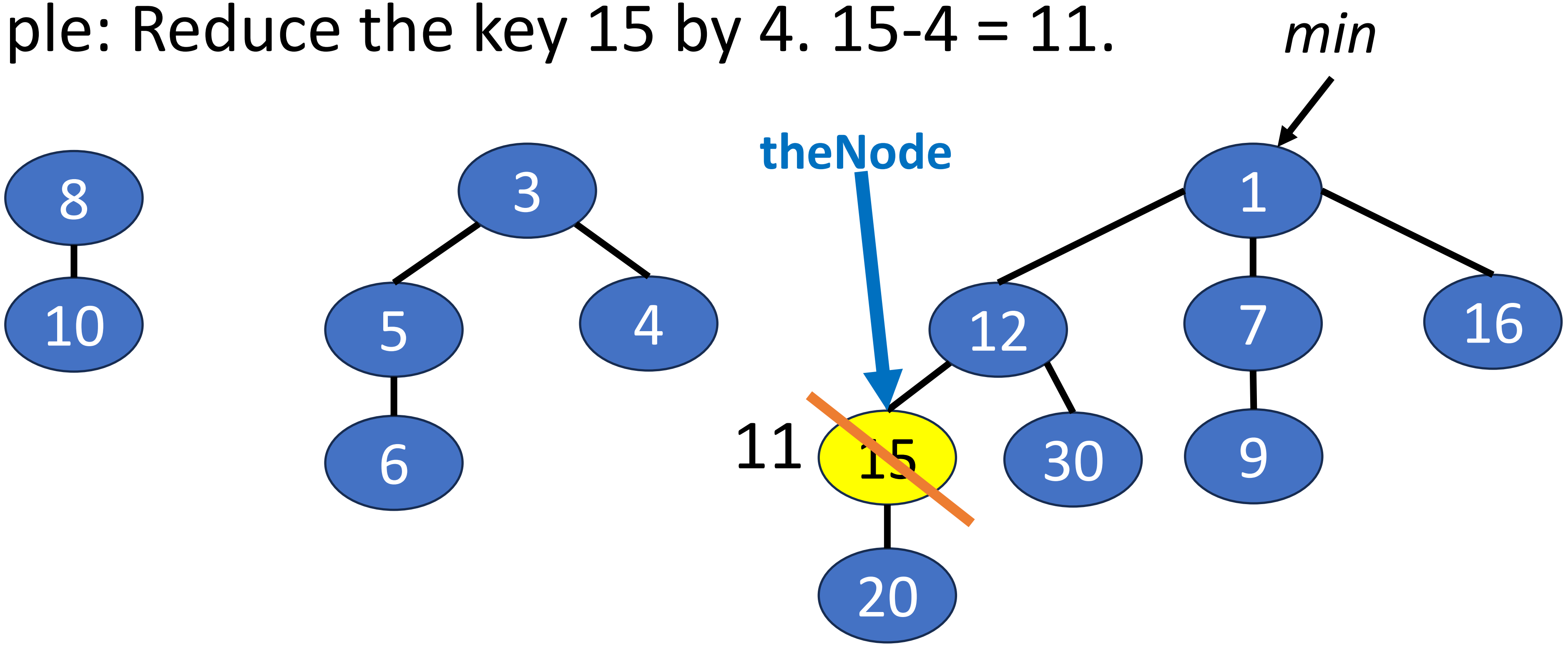
Trees of equal degree are not joined together as in delete-min.

Operation: DecreaseKey(theNode, theAmount)

(1) Decrease key

(2) If theNode is not a root and new key **<** parent key, remove subtree rooted at theNode from its doubly linked sibling list.

Example: Reduce the key 15 by 4. 15-4 = 11.

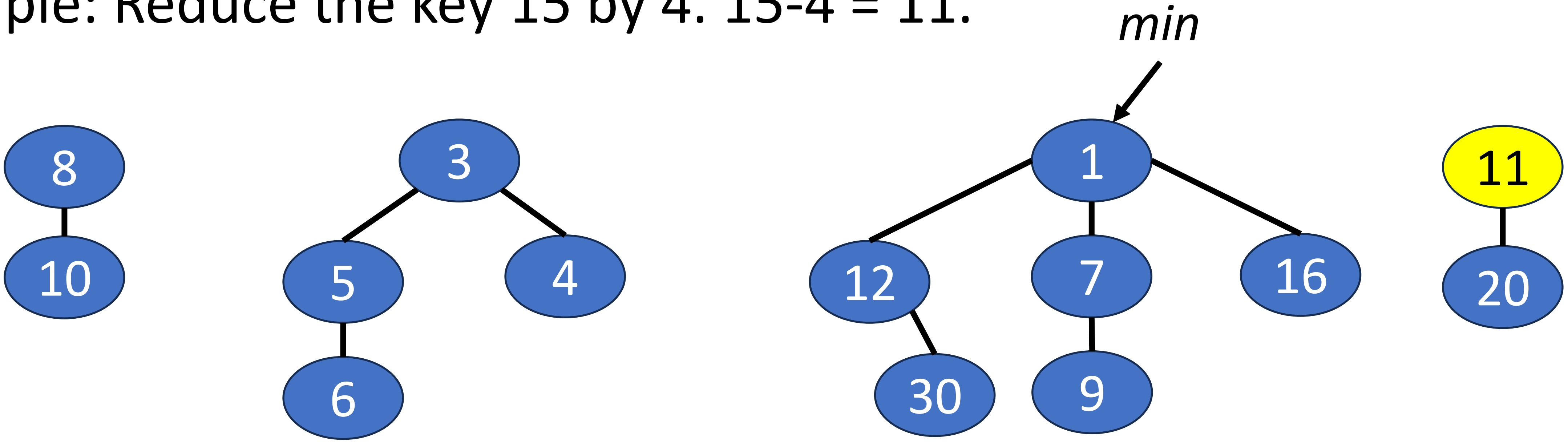


Operation: DecreaseKey(theNode, theAmount)

(3) Insert into top-level list.

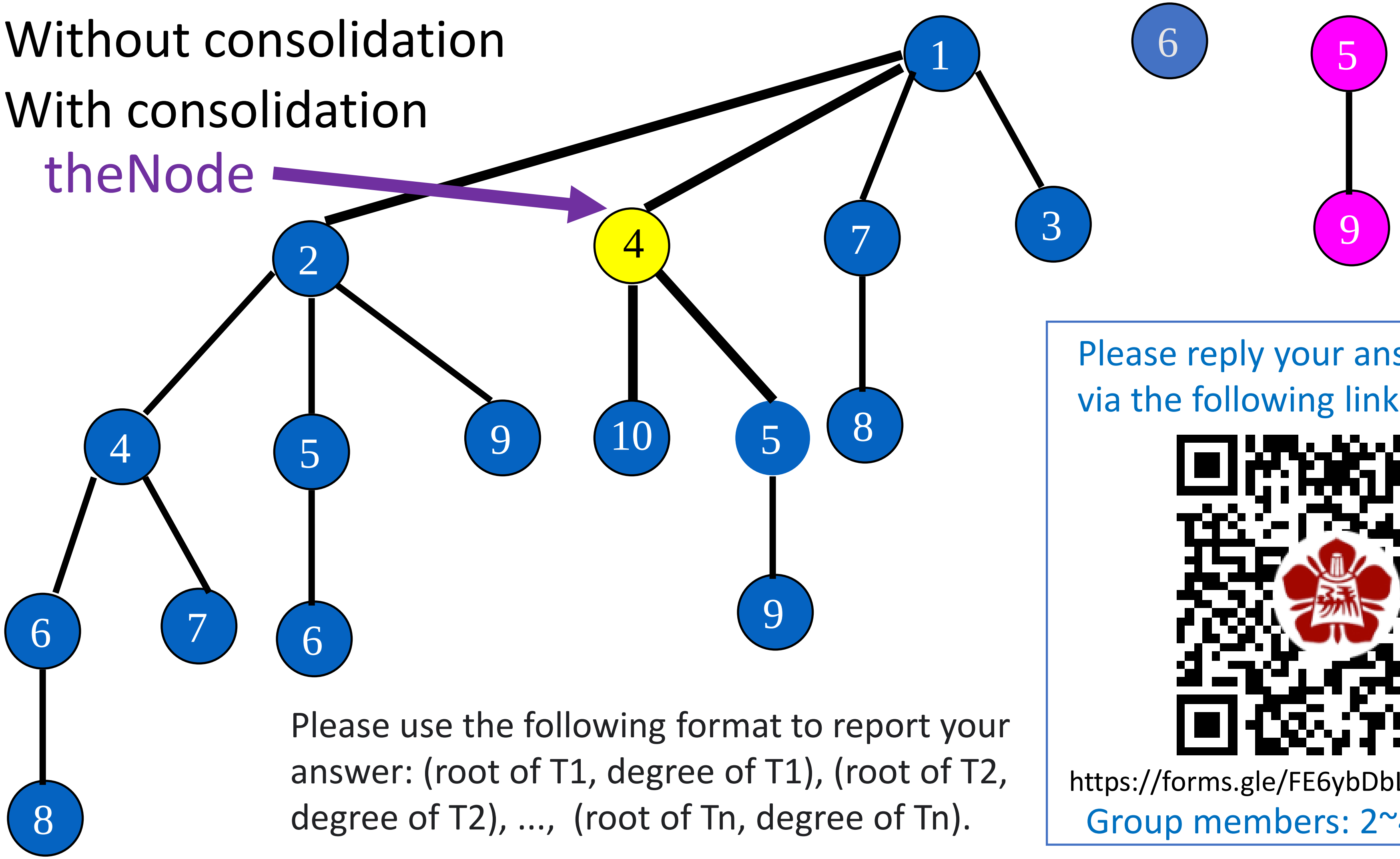
(4) Update *min* pointer

Example: Reduce the key 15 by 4. 15-4 = 11.



Exercise

- Delete 4 from the following Fibonacci heap. Please write out the roots and degrees of the min trees in the resulting F-heap.
  - Q1-1: Without consolidation
  - Q1-2: With consolidation



Please use the following format to report your answer: (root of T1, degree of T1), (root of T2, degree of T2), ..., (root of Tn, degree of Tn).

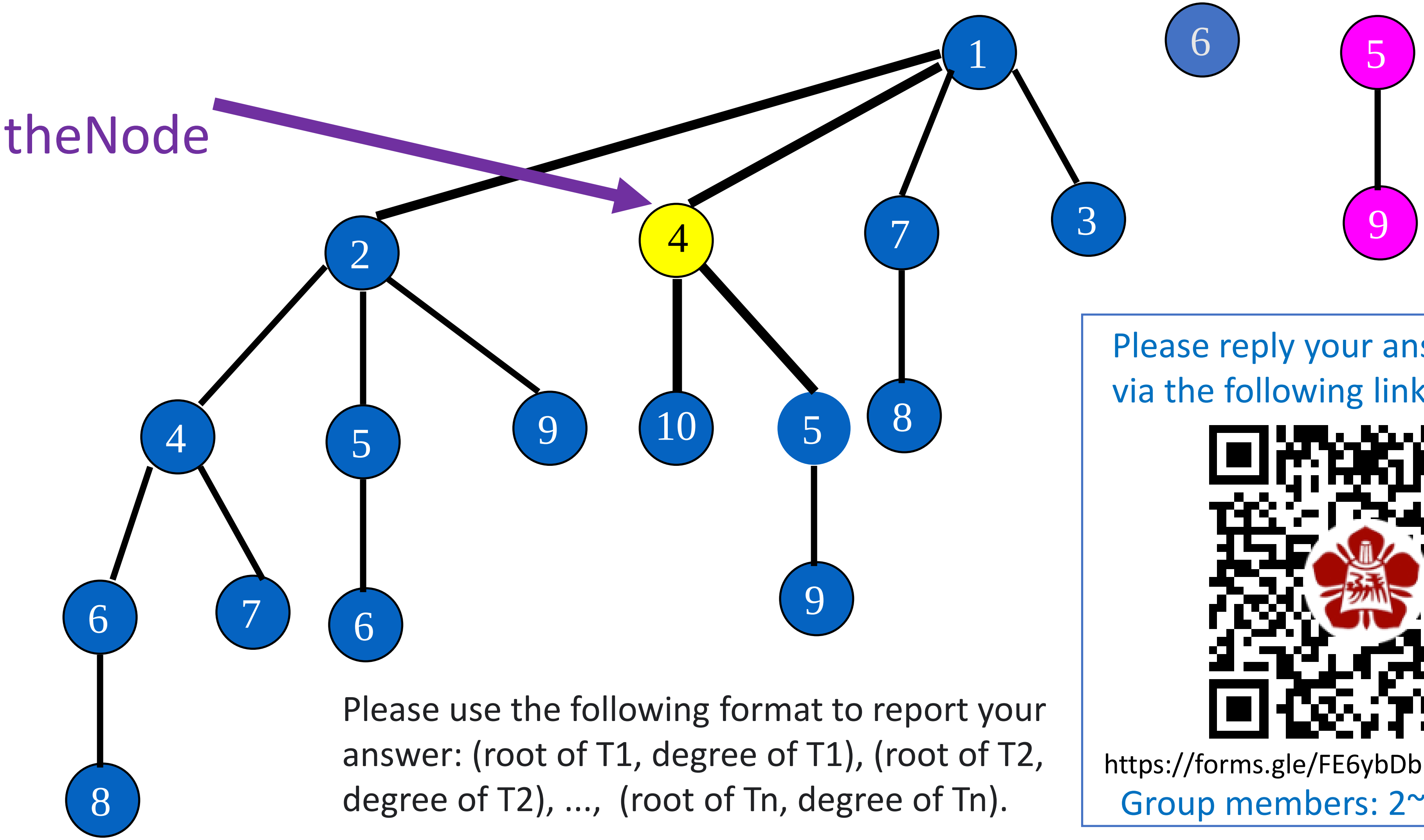
Please reply your answers of Q1 via the following link:

https://forms.gle/FE6ybDbLdaoGbWCQ6

Group members: 2~4 people

Exercise

- Q2: Reduce the key 4 by 4 (that is, 4 becomes 0). Please write out the roots and degrees of the min trees in the resulting F-heap.



Please use the following format to report your answer: (root of T1, degree of T1), (root of T2, degree of T2), ..., (root of Tn, degree of Tn).

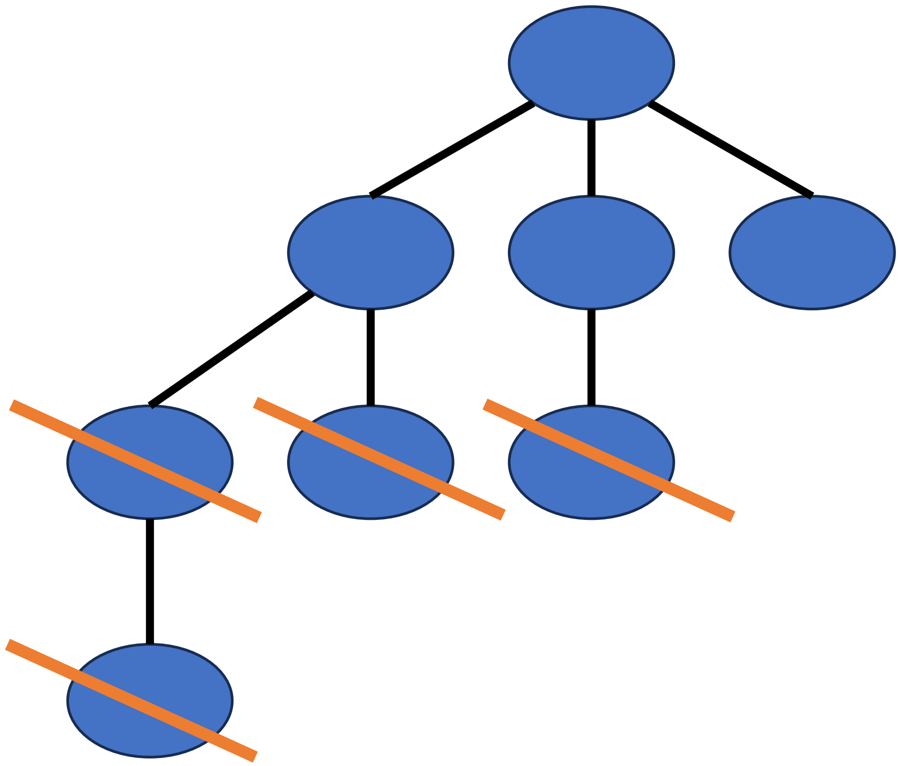
Please reply your answers of Q2 via the following link:

https://forms.gle/FE6ybDbLdaoGbWCQ6

Group members: 2~4 people

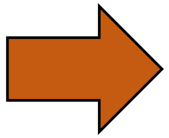
Min-trees after deletion and decrease key

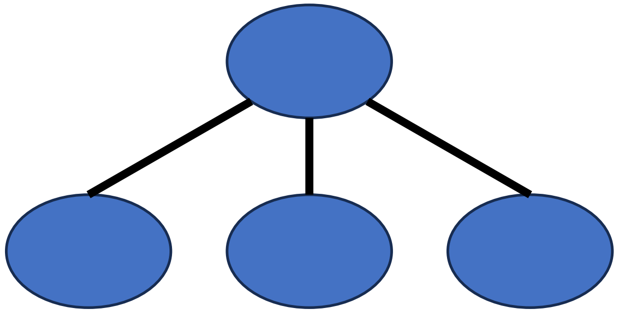
- The min trees in an F-heap need not be binomial trees.
- A min-tree of degree k may have number of nodes $\leq k+1$.



B3

After four times of deletion



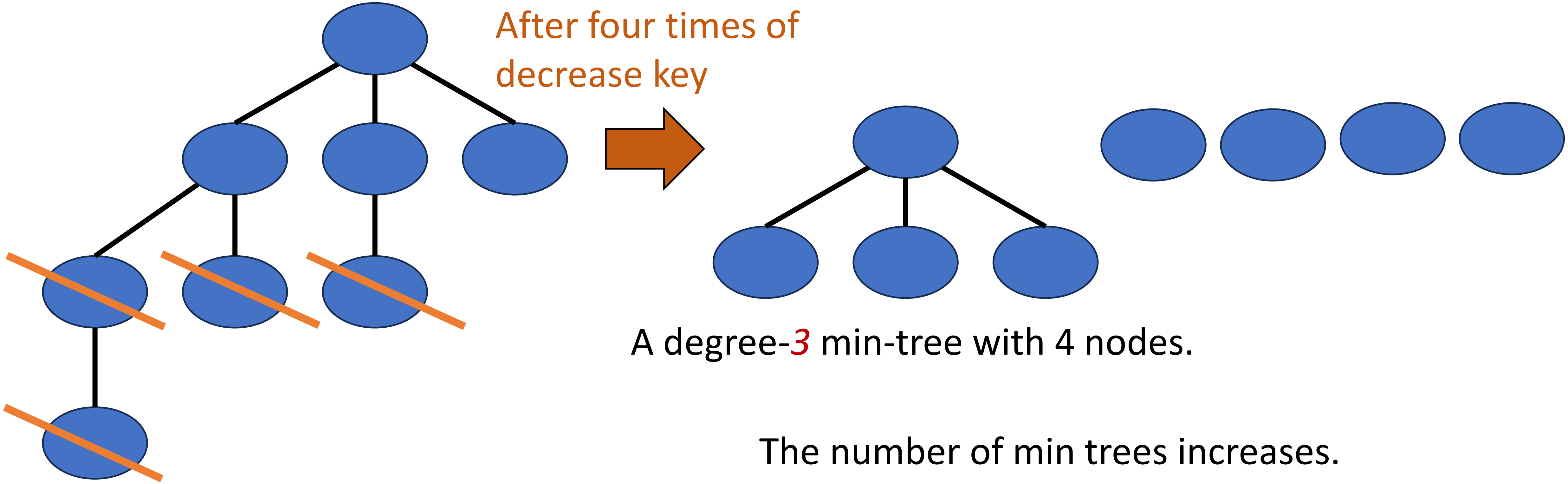


A degree-*3* min-tree with 4 nodes.

- In a B-heap: B_k has exactly 2^k nodes. The number of min trees following a delete-min is $\leq \log_2 n$.

Min-trees after deletion and decrease key

- The min trees in an F-heap need not be binomial trees.
- A min-tree of degree *k* may have number of nodes ≤ *k*+1.

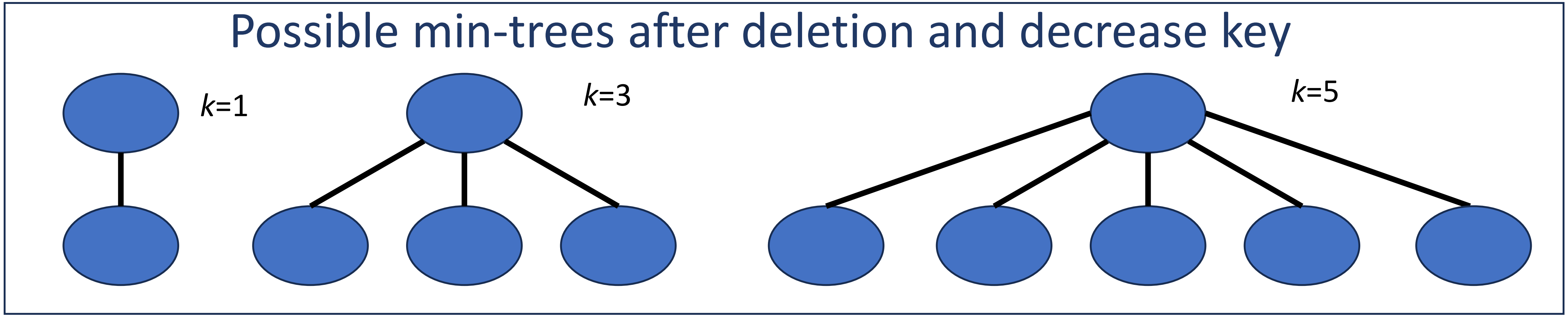


B3

A degree-*3* min-tree with 4 nodes.

The number of min trees increases.
➔ Time complexity of delete-min increases.

of nodes in a tree should be limited



Theorem 9.1 P.440

- To have amortized time complexity of insert and meld to be O(1) and that of delete-min to be O(log n), the number of nodes in a min-tree of degree k should be at least c^k, $c>1$.

How? Limit the number of cuts among the children of any node to 2.

Limit the number of cuts among the children of any node to 2

Next time, if a child of a marked node is lost, it is the second time to lose child and the operation will invoke **cascade cut**.

ChildCut flag

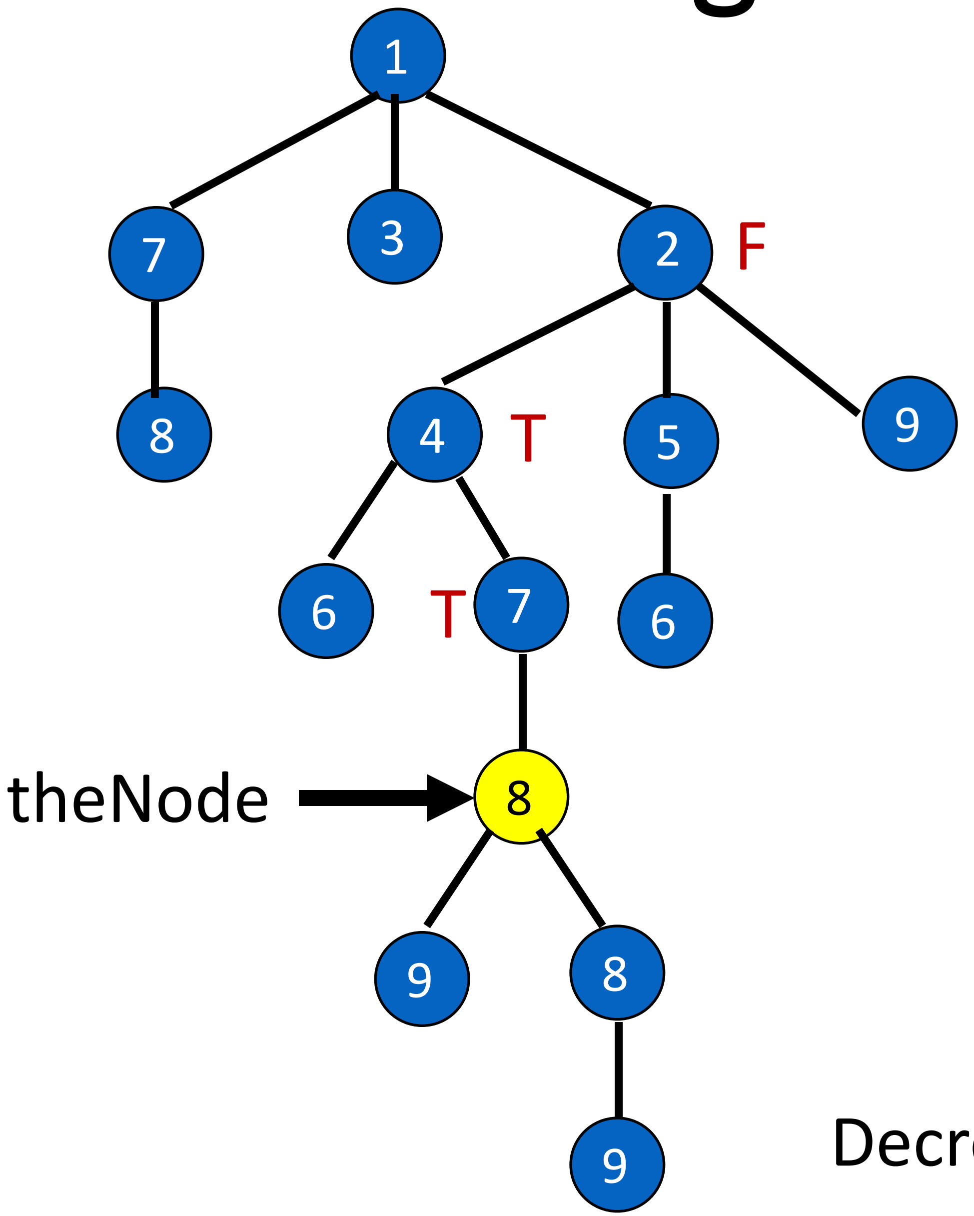
- True if node has lost a child since it became a child of its current parent.
- Set to false by remove min, which is the only operation that makes one node a child of another.
- Undefined for a root node.

Cascading cut

- When theNode is cut out of its sibling list in a delete or decrease key operation, follow path from parent of theNode to the root.
- Encountered nodes (other than root) with ChildCut = true are cut from their sibling lists and inserted into top-level list.
- Stop at first node with ChildCut = false.
  - For this node, set ChildCut = true.

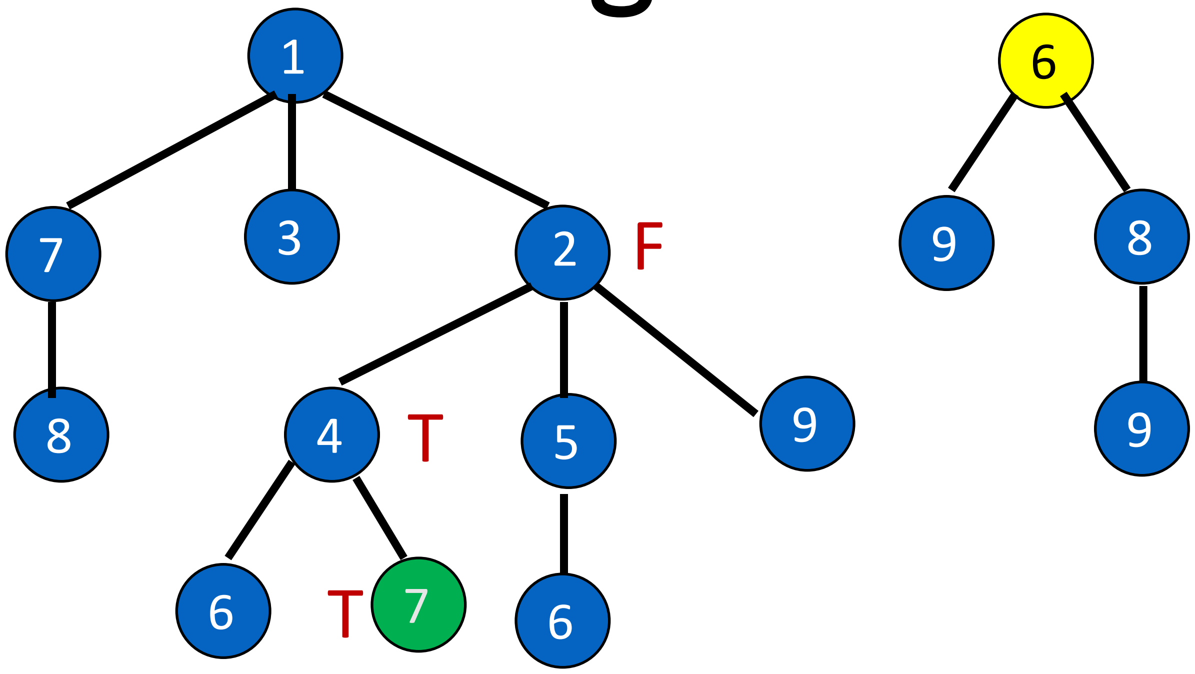
- Note: ChildCut becomes “false” if being merged due to delete min (or delete)

Cascading Cut Example

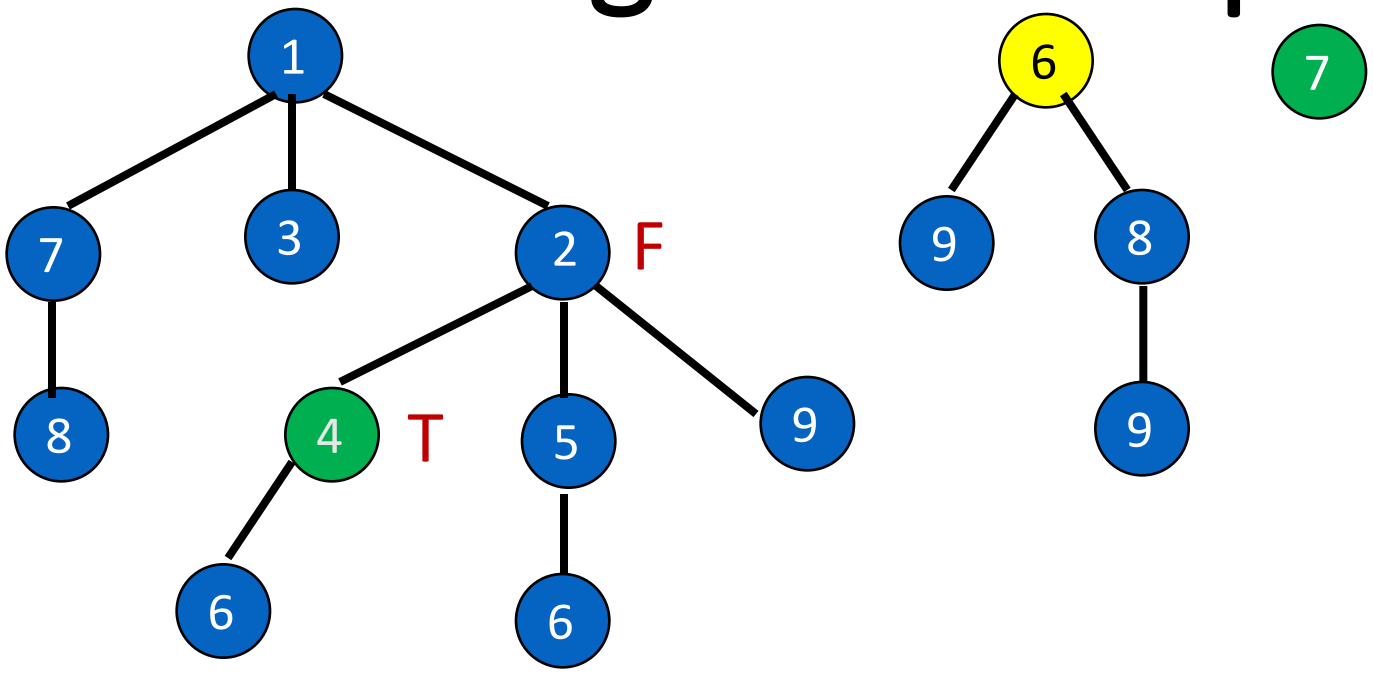


Decrease key by 2.

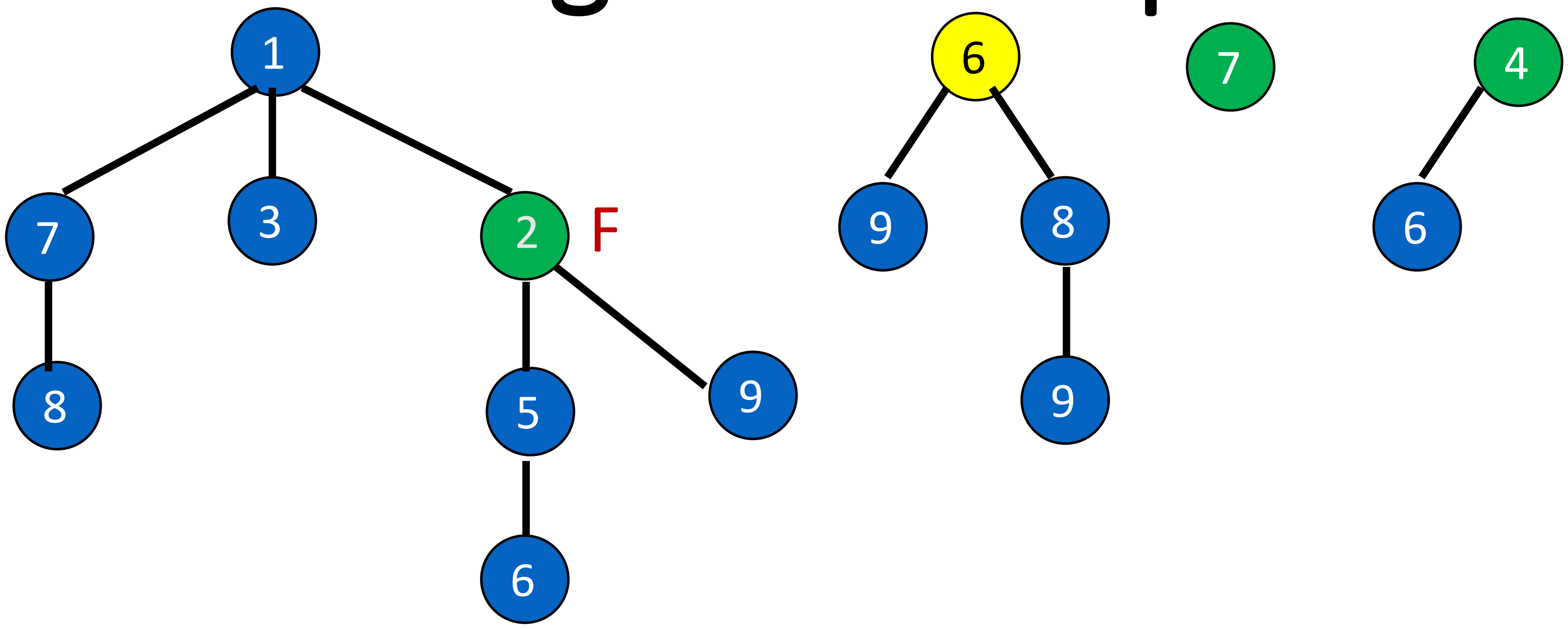
Cascading Cut Example



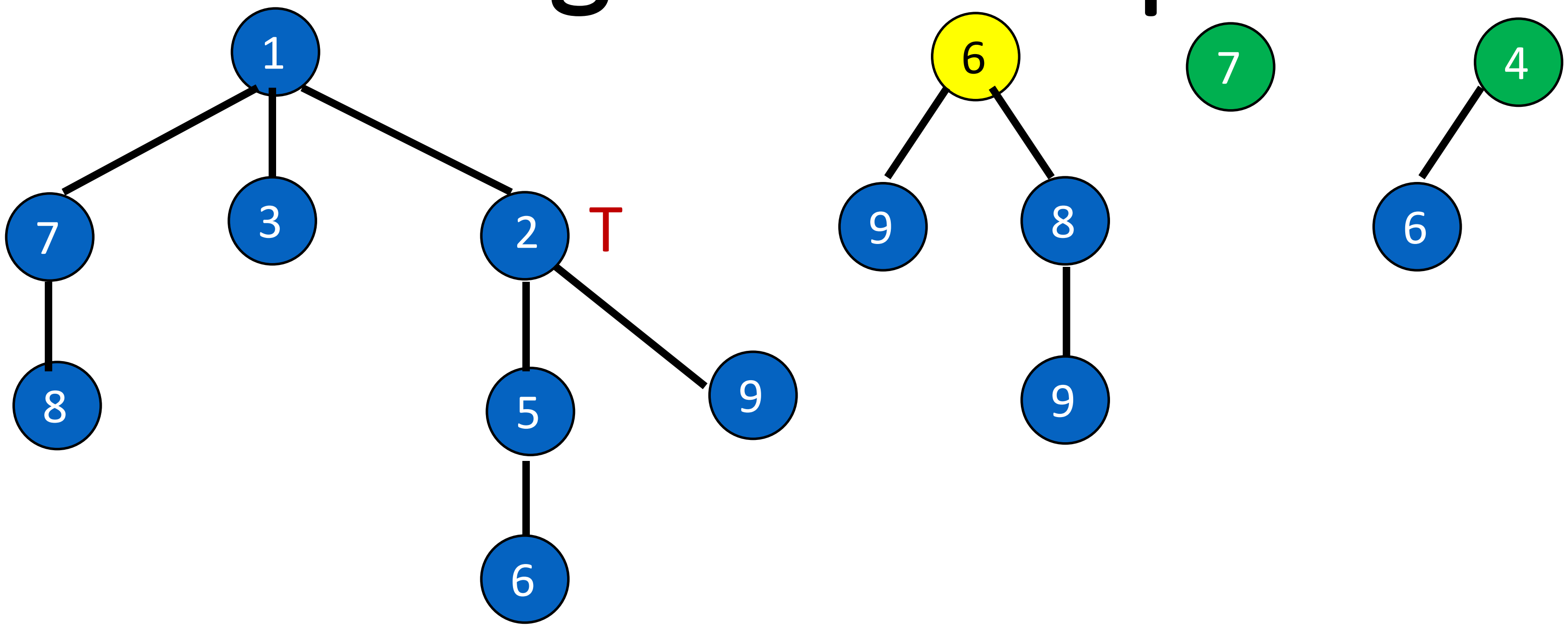
Cascading Cut Example



Cascading Cut Example



Cascading Cut Example

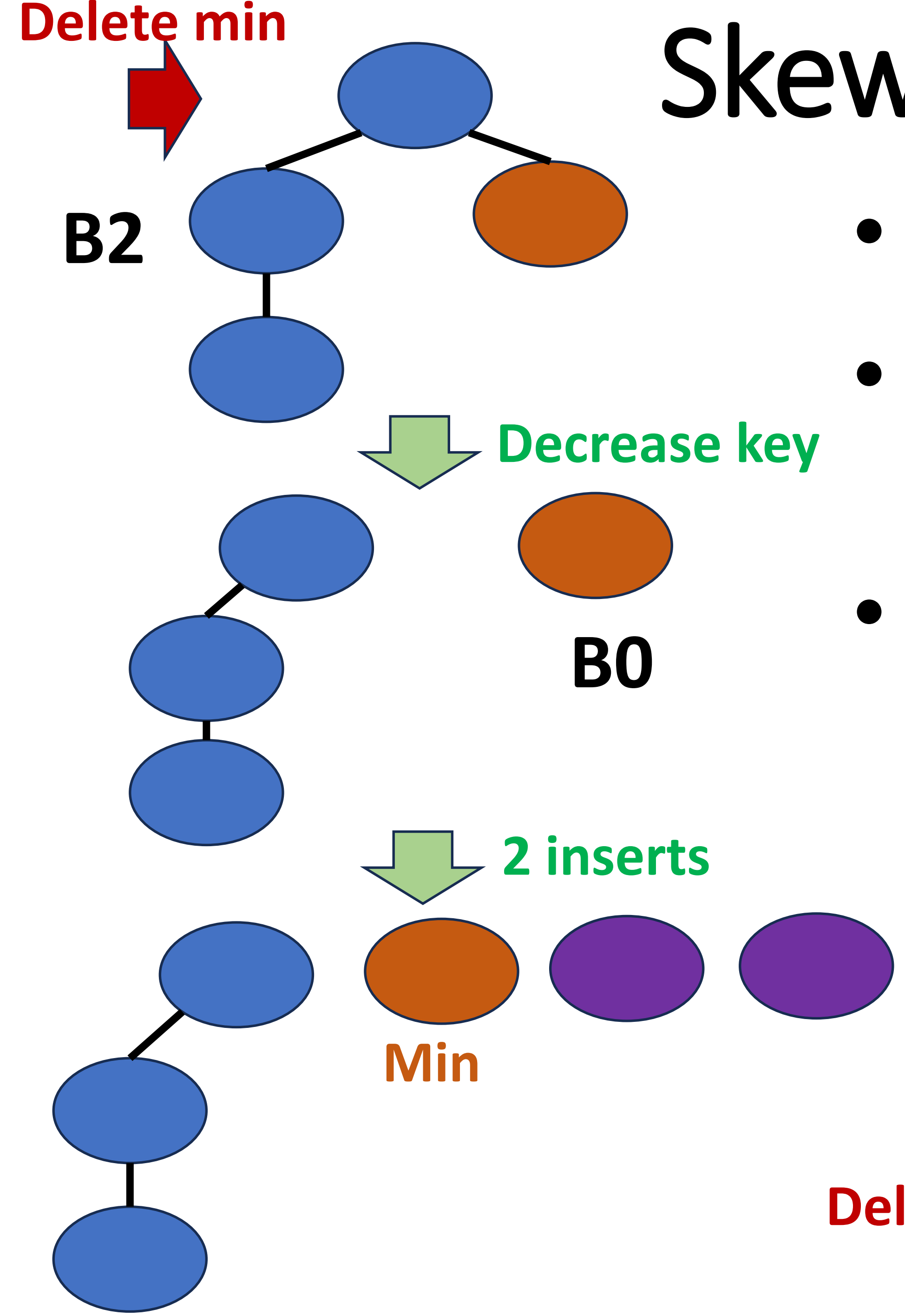


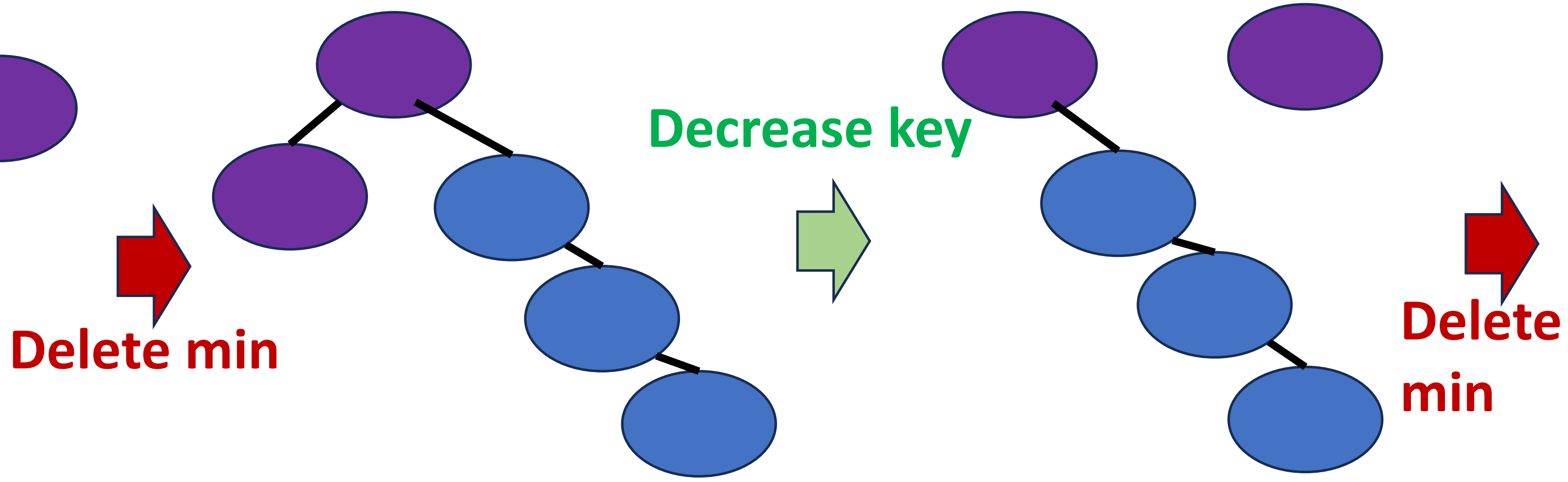
Actual complexity of cascading cut is O(h) = O(n). //Why?

It is correlated to the number of insertion and decrease key operations (Since last delete min). // amortized cost analytic

Skewed trees may be generated

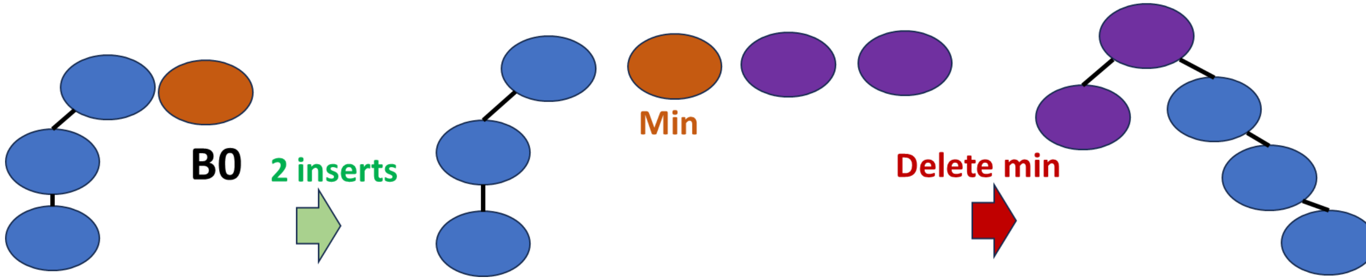
- Do operations to create a tree B2 (height=3).
- Do a decrease key to get a chain of height 3 and a B0.
- Do 2 inserts, a remove min, and a decrease key to get a chain of height 4 and B0.



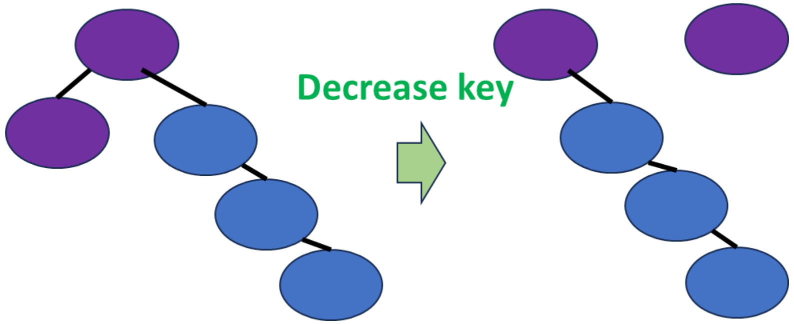


Insertion and decrease affect tree height (now or in the future)

- Two inserts + one delete-min: The height is increased by 1.



- Decrease-key may result in skewed trees or decrease tree height.



An F-heap may have trees with height=O(n), where n is number of operations or number of elements in tree.

Why does cascading cut work?

- It’s relationship to Fibonacci.
- The internal node should be cut if two children are cut. Why? (Why not one child or three children?)
- Lemma 9.4

Fibonacci number

- Definition: $\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} & \text{for } n \geq 2 \end{cases}$

- Example:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
0	1	1	2	3	5	8	13	21	34	55	89	144

- Closed form: $$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{\boxed{\varphi^n} - \psi^n}{\sqrt{5}},$$

where

golden ratio $$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.61803\,39887\ldots$$

When n>3, F_n is close to F_3*1.618^{n-3}.

F_n grows exponentially.

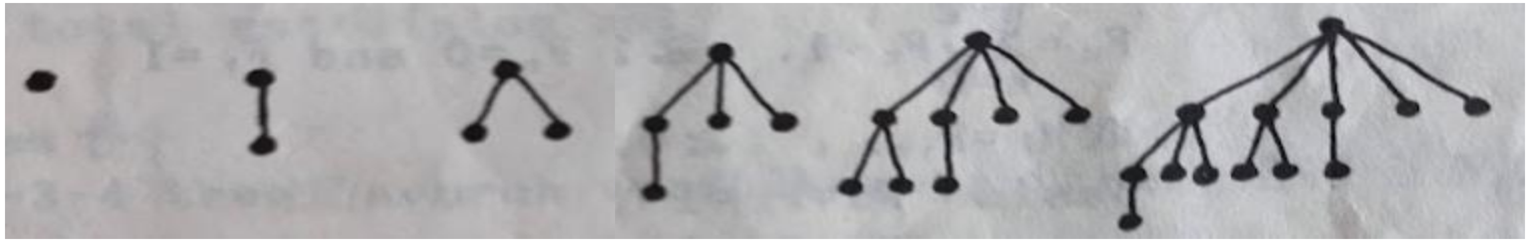
Fibonacci number vs. min trees

- Example:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
0	1	1	2	3	5	8	13	21	34	55	89	144

F_2 F_3 F_4 F_5 F_6 F_7

- Trees:

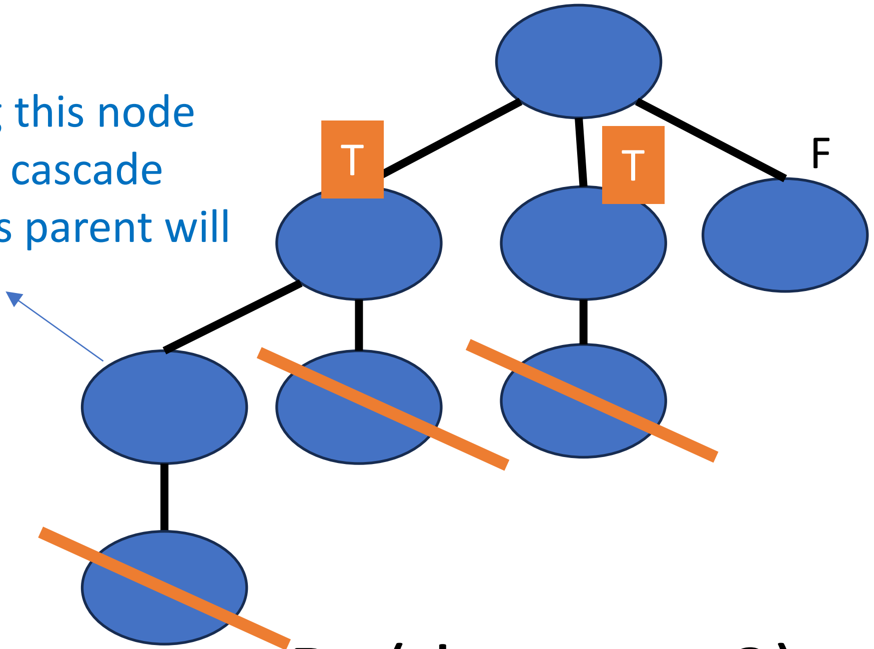


	F_2	F_3	F_4	F_5	F_6	F_7
Number of nodes:	1	2	3	5	8	13
Degree:	0	1	2	3	4	5

“Fibonacci” heap

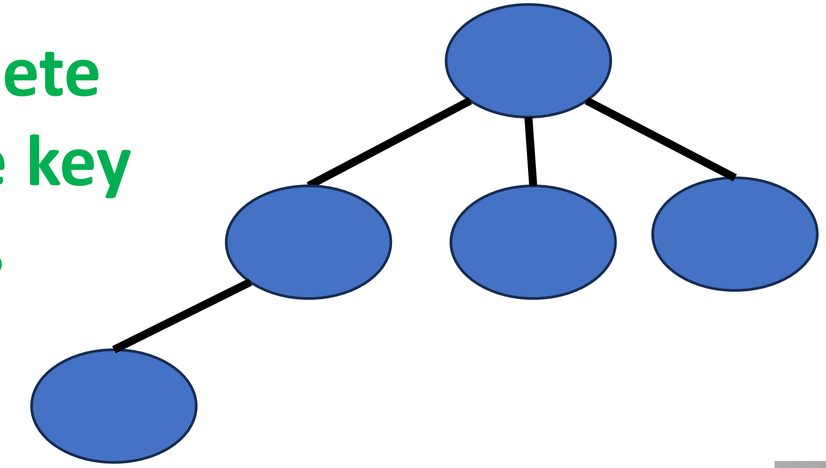
- Given a b-heap, after performing several *delete* and *decrease key* operations, at least how many nodes can remain in the min tree if the degree of the b-heap maintains the same?
- Example: Degree 3 (B_3 -> F_5)

Removing this node will cause cascade cut and its parent will be cut.

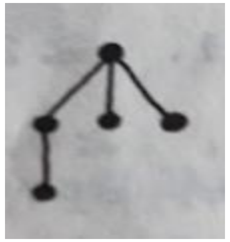


B_3 (degree = 3)

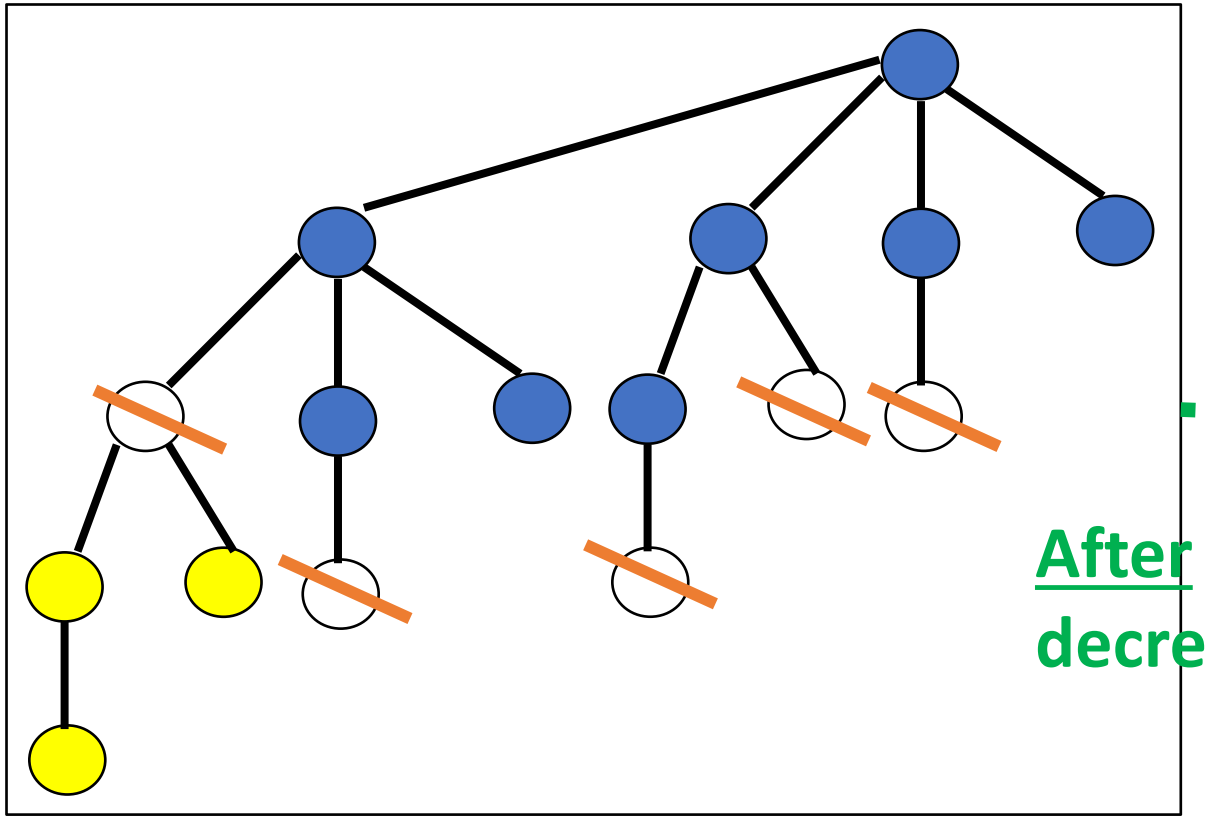
<u>At most</u> delete or decrease key <u>three times</u>



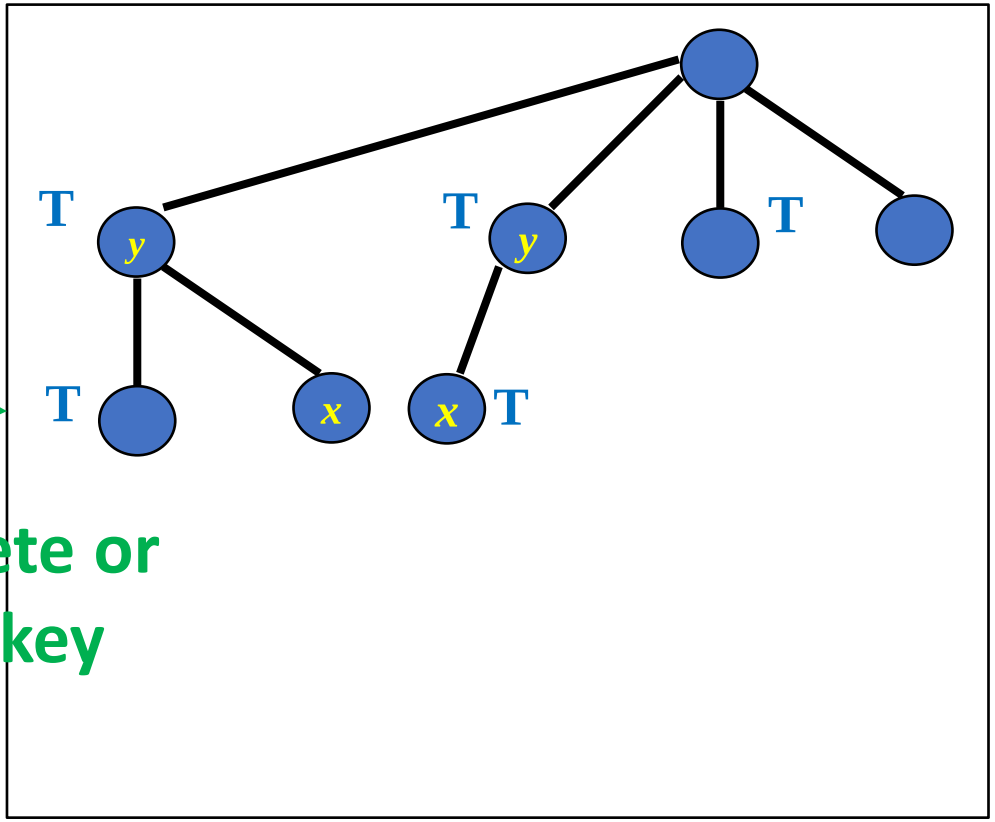
F_5 (degree = 3)



One more example (Degree 4: B_4 -> F_6)

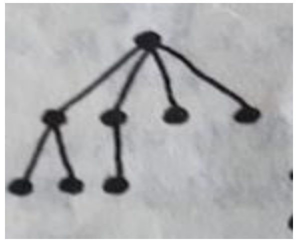


After delete or decrease key



B_4

degree = 4



F_6

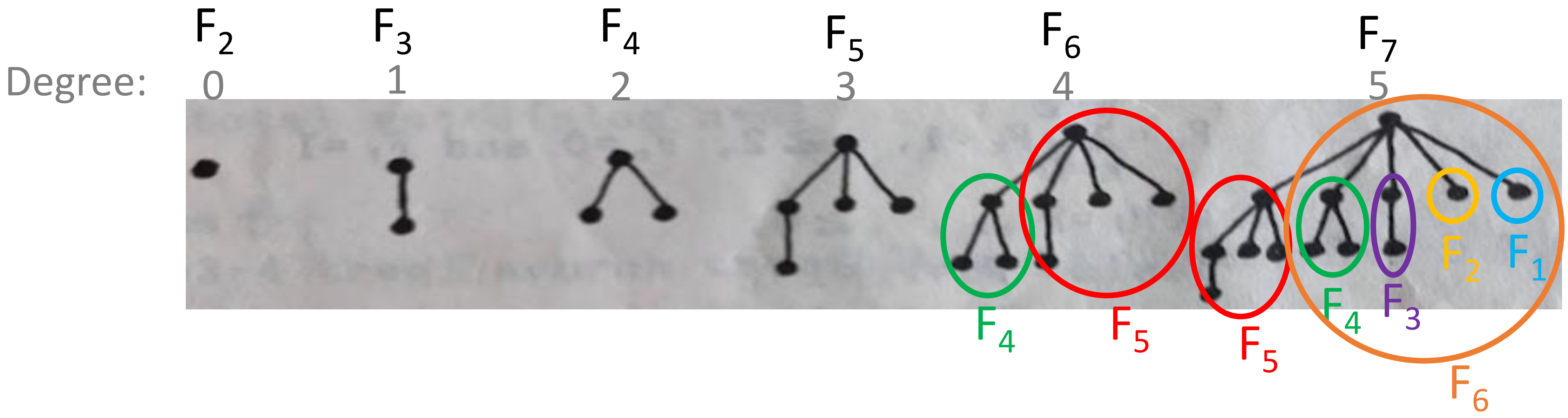
degree = 4

- Given a b-heap, after performing several *delete* and *decrease key* operations, at least how many nodes can remain in the min tree if the degree of the b-heap maintains the same?

Answer: Fibonacci number

$F_6=F_5+F_4$ (by definition)

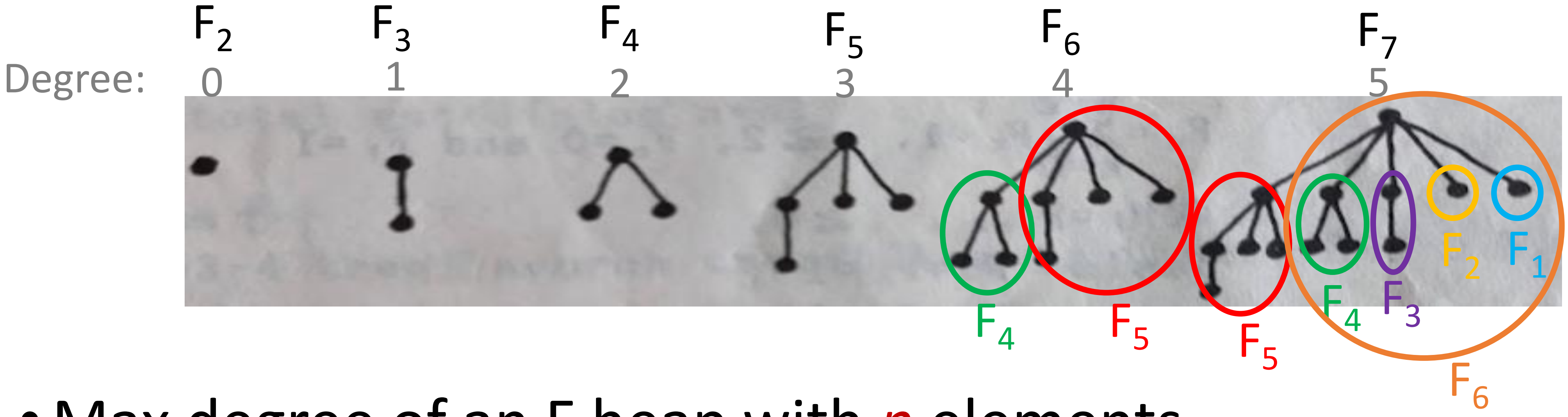
$F_7= F_6+F_5=F_5+F_4+F_3+F_2+F_1+1$



With cascading cut, the number of node in a min tree of degree *i* is at least F_{i+2} .

Max degree

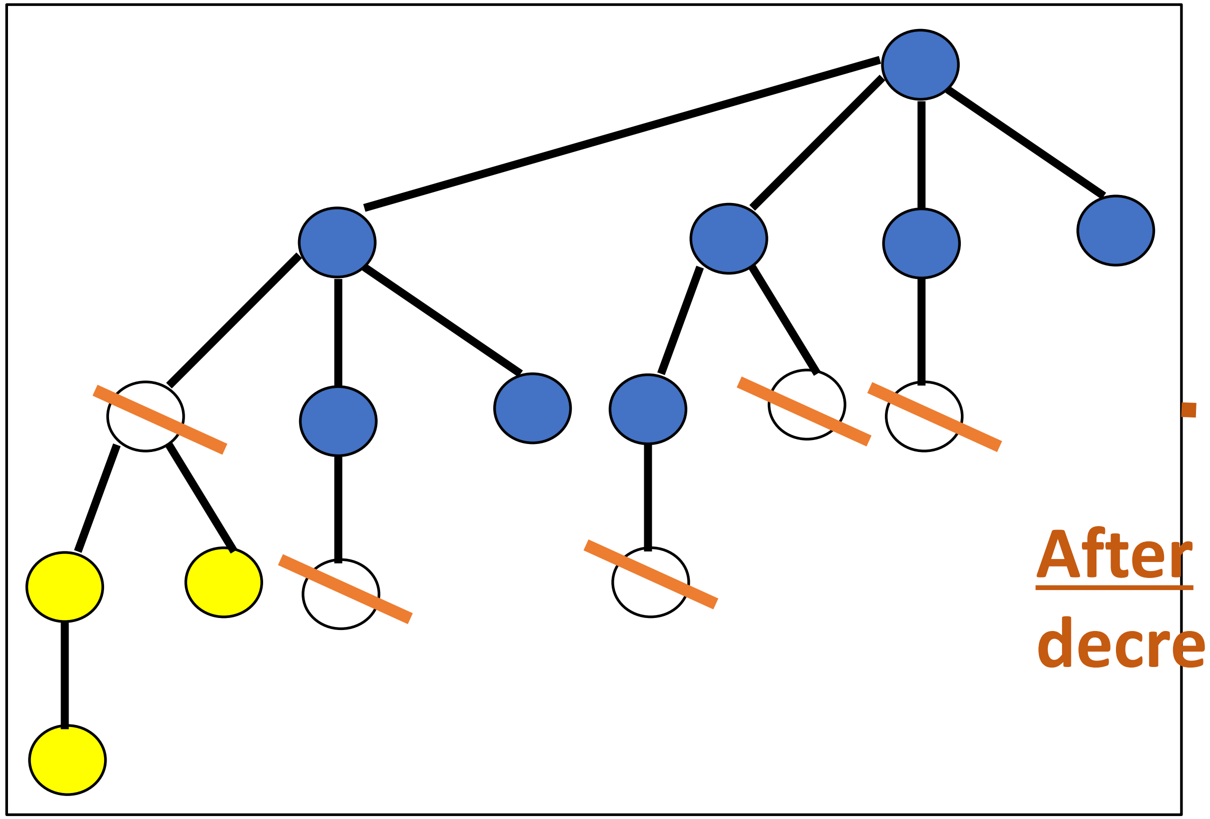
- When $k>3$, F_k is close to $F_3*1.618^{k-3}$.
- The following figure shows that degree of F_k is k-2.



- Max degree of an F heap with n elements
  - k is close to $\log_{1.618}(n)$
  - Degree = $k-2 \approx \log_{1.618}(n) - 2 \rightarrow O(\log n)$

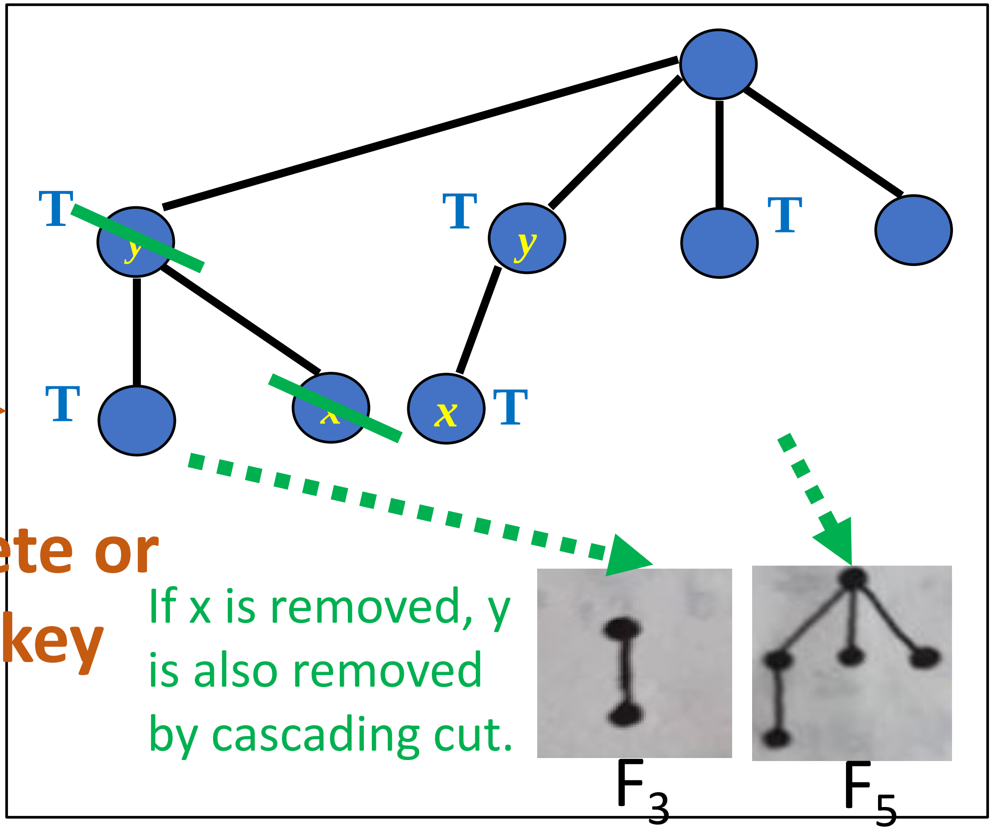
More on cascading cut

- If any nonroot node in F_6 is removed, # of nodes becomes at least F_5.
- Cascading cut can maintain # of nodes to be $F_?$.
  The removed subtree also belongs to $F_?$ family.



$\mathbf{B_4}$

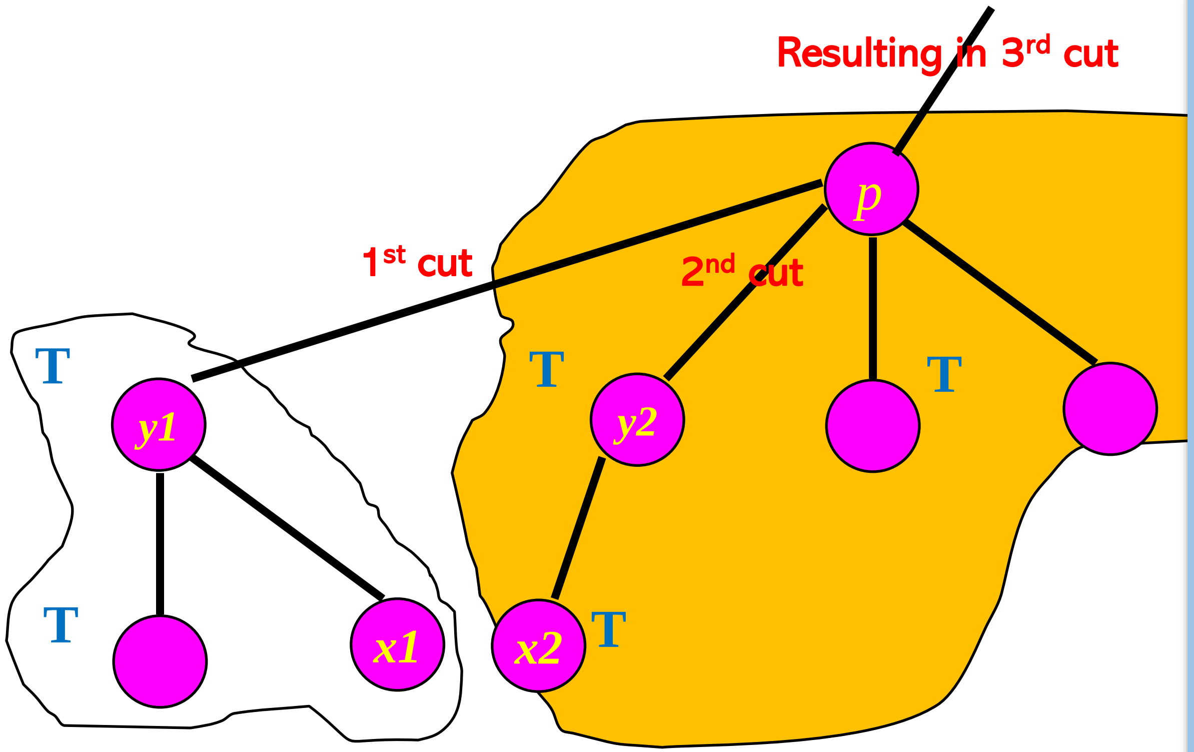
After delete or decrease key



$\mathbf{F_6}$

More on cascading cut

- When a parent p removes one of its child *y1* (a subtree rooted at *y1*), the subtree *y2* of *p* is also cut if any *p*'s descendant subtree (say, *x2*) is further eliminated.



$F_{k-1}=F_{k-2}+F_{k-3}$

y1

$F_k=F_{k-2}+F_{k-3}+\ldots+F_1+1$

y2

Q: After cutting two subtrees (y1 and y2), why do we cut the entire the entire p's subtree (3rd cut)?

A: If two largest subtrees are cut, the remaining nodes in p is less than ½ of the original size.

Recall: $F_i \approx 1.618F_{i-1}$ (for any i)

\# of nodes in p: $F_{k+1}=F_k+F_{k-1}$

\# of nodes in y1: F_{k-1}

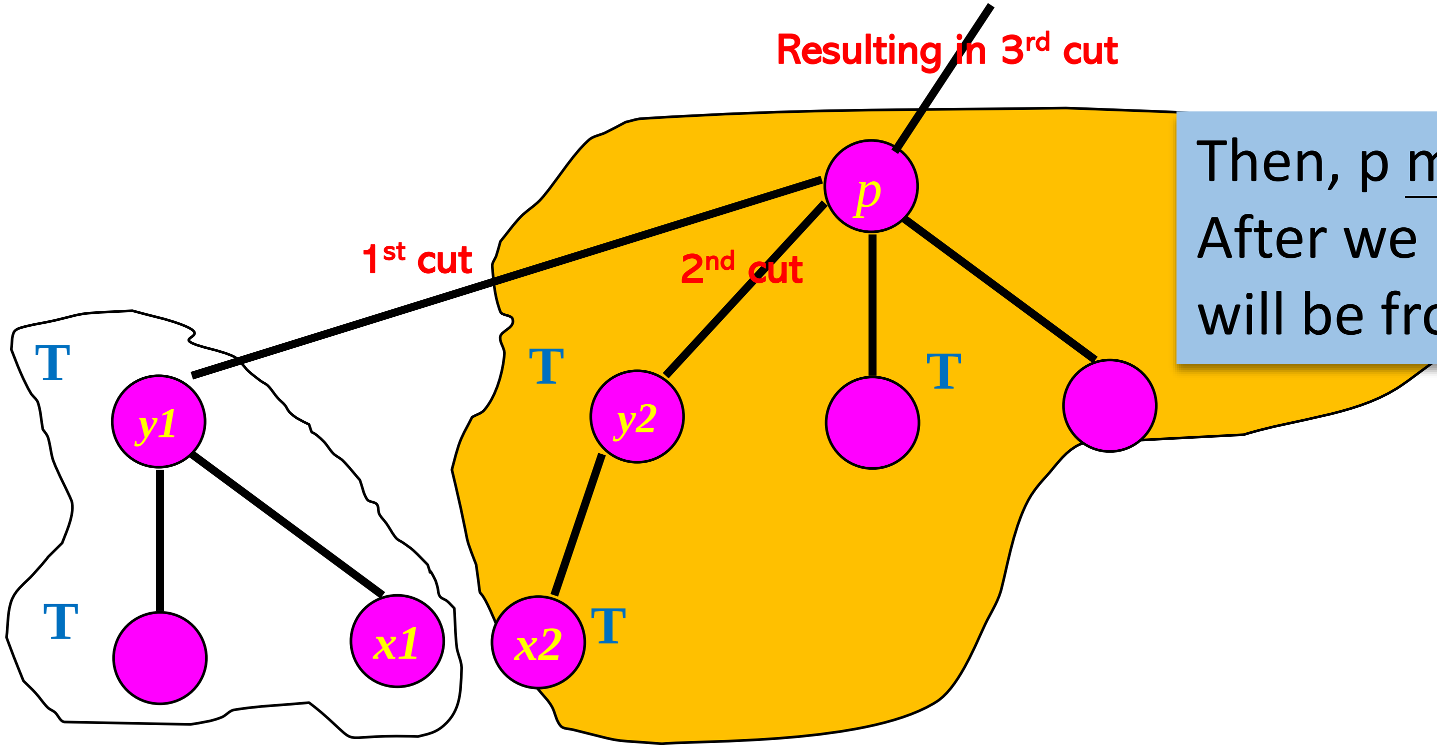
\# of nodes in y2: F_{k-2}

\# of nodes in y1+y2: $F_k = F_{k-1}+F_{k-2}$

Two top level F heaps F_{k-1} and F_{k-2} are created.

More on cascading cut

Then, p may be the biggest subtree in p's parent. After we remove p from p's parent, p's parent will be from F_t to F_{t-1}.



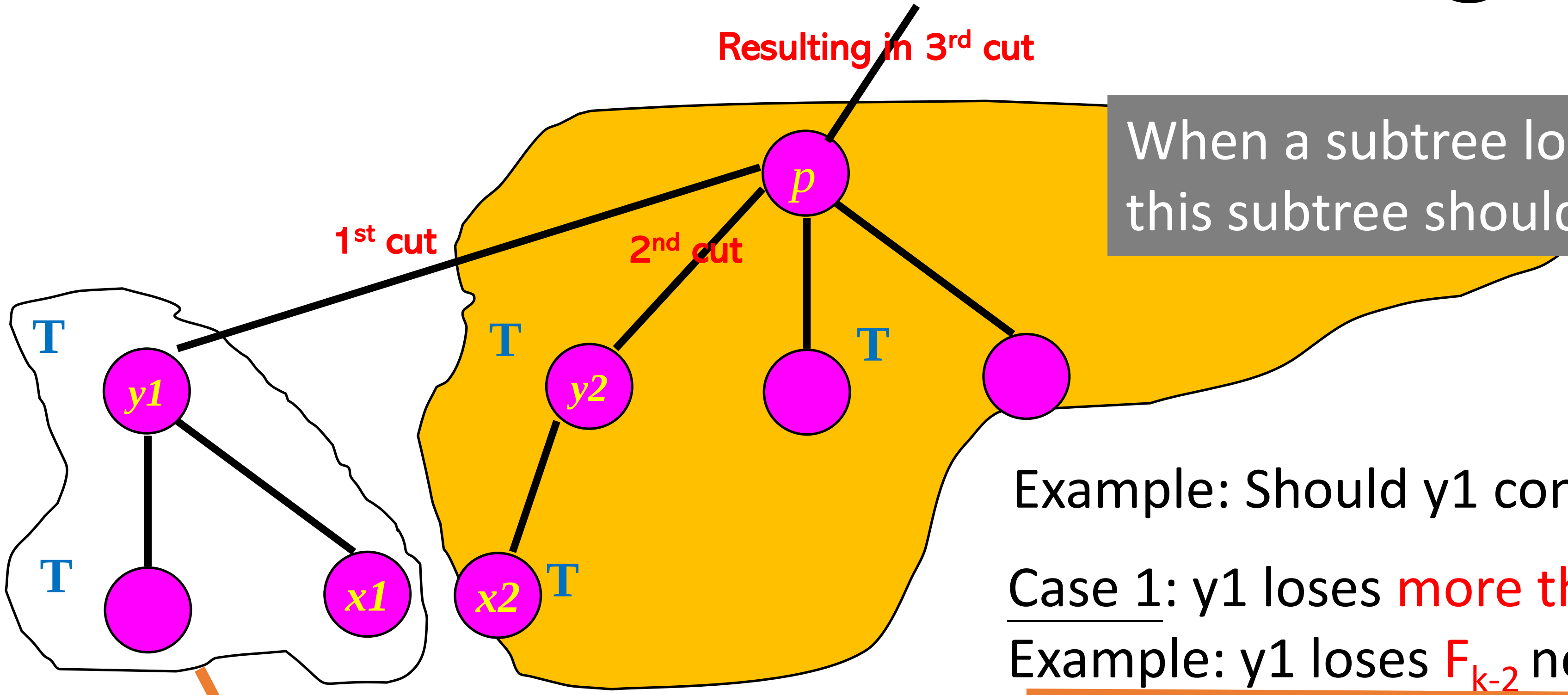
$F_{k-1}=F_{k-2}+F_{k-3}$

y1

$F_k=F_{k-2}+F_{k-3}+\ldots+F_1+1$

y2

More on cascading cut



When a subtree loses more than ½ of nodes, this subtree should be removed. Why?

Recall: $F_i \approx 1.618 F_{i-1}$ (for any i)

$F_{k-1}= \cancel{F_{k-2}}+F_{k-3}$ ↖ *y1*

$F_k=F_{k-2}+F_{k-3}+\dots+F_1+1$ ↖ *y2*

Example: Should y1 continue to be the subtree of p?

Case 1: y1 loses more than ½ of nodes.

Example: y1 loses F_{k-2} nodes.

The number of nodes in p subtree becomes

$F_{k-2}+F_{k-3}+F_{k-3}+\dots$ $\quad F_{k-1}=F_{k-2}+F_{k-3}$

$=F_{k-1}+(F_{k-3}+\dots)$

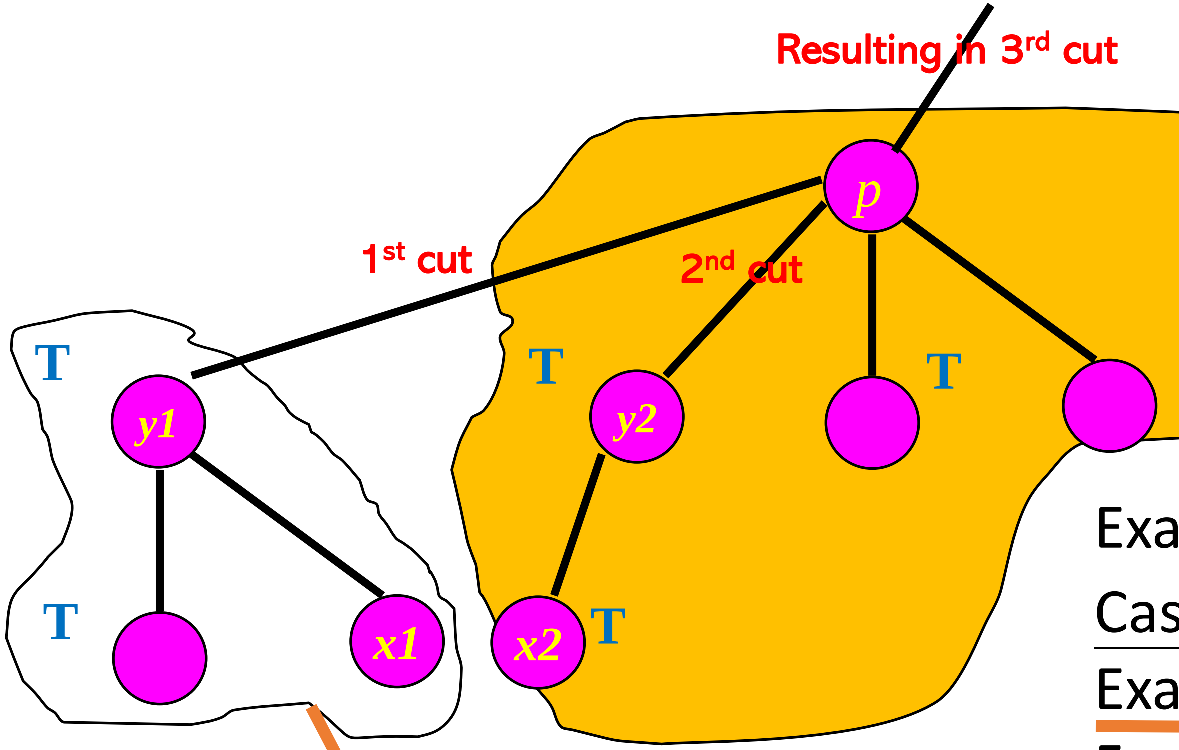
$=F_{k-1}+F_{k-1}$ $\quad F_{k-1}=F_{k-3}+F_{k-4}+F_{k-5}+\dots$

$=2F_{k-1} >(1.618)F_{k-1}$

Not the expected number of nodes for p subtree.

y1 shouldn't be the subtree of p.

More on cascading cut



When a subtree loses more than ½ of nodes, this subtree should be removed. Why?

$F_{k-1}=F_{k-2}+\cancel{F_{k-3}}$

y1

$F_k=F_{k-2}+F_{k-3}+\ldots+F_1+1$

y2

Recall: $F_i \approx 1.618F_{i-1}$ (for any i)

Example: Should y1 continue to be the subtree of p?

Case 2: y1 loses less than ½ of nodes.

Example: y1 loses F_{k-3} nodes.

$F_{k-2}+F_{k-2}+F_{k-3}+\ldots$

$\approx(1.618)^2 F_{k-2}+(F_{k-3}+\ldots)$

$=(1.618)^2 F_{k-2}+F_{k-1}$

$\approx(1.618)F_{k-1}+F_{k-1}$

$=(1.618)^2 F_{k-1}$

$\approx(1.618)F_k$

$\approx F_{k+1}$

$(1.618)^2 = 2.617924$

$F_{k-1} = F_{k-3} + F_{k-4} + F_{k-5} + \ldots$

$F_{k-1} \approx 1.618\, F_{k-2}$

$(1.618)^2 = 2.617924 = 1.618+1$

$F_k \approx 1.618\, F_{k-1}$

It is the expected number of nodes for p subtree.
y1 can be the subtree of p.

Lemma 9.4 (p.446)

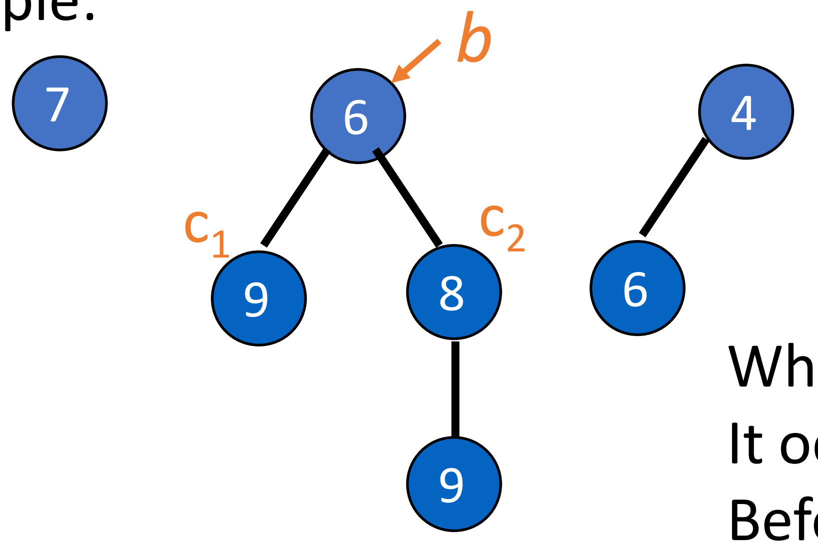
- Prove that a sequence of operations generates the tree of degree *k* with minimum number of nodes = F_{k+2}, $k > 0$.

b: any node in any of the min-trees of an F-heap

N_i: minimum number of elements in the subtree with root *b* and degree of *b* is *i*.

$c_1, c_2, .., c_i$: the j-th children of *b*, c_j became a child earlier than c_{j+1}.

Example:

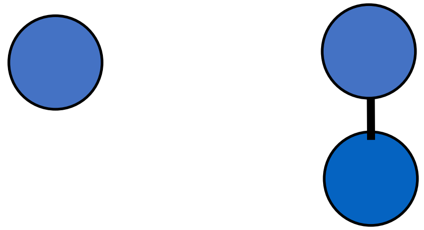


Degree = 2

N_2 = ?

Proof:

$N_0 = 1$ $N_1 = 2$



When c_k becomes a child of *b*, the degree of *b* is at least *k*-1.

It occurs only during min-tree joining for delete-min operation.

Before joining, c_k has the same degree as *b*.

After joining, c_k may lose one child. So, degree of c_k>=max{0, k-2}

$$N_i = \overset{c1}{N_0} + \overset{c2,\ldots,\,ci}{\sum_{k=0}^{i-2} N_k} + \overset{\text{root } b}{1} = \sum_{k=0}^{i-2} N_k + 2 = F_{i+2}$$

Consolidation due to delete/delete min

- Recall: merge 2 Bi’s with identical degree in B-heap, iteratively.

- How about consolidation in F-heap?
  - Say, merge F_5 and F_4, resulting in F_6
    - Refer to “Fibonacci number def.”
  - In fact, you may also merge 2 F_5’s, resulting in something larger than F_6 (in terms of # of nodes)
    - Prior merging algorithm remains to work

Fibonacci heaps

- Time complexity for different operations

	Actual	Amortized
Insert	O(1)	O(1)
Delete min (or max)	O(n)	O(log n)
Meld	O(1)	O(1)
Delete "any"	O(n)	O(log n)
Decrease key (or increase)	O(n) Due to cascading cut	O(1)

See Theorem 9.2 (P.447)

Similar with delete-min

One to cascading cut
One to delete-min

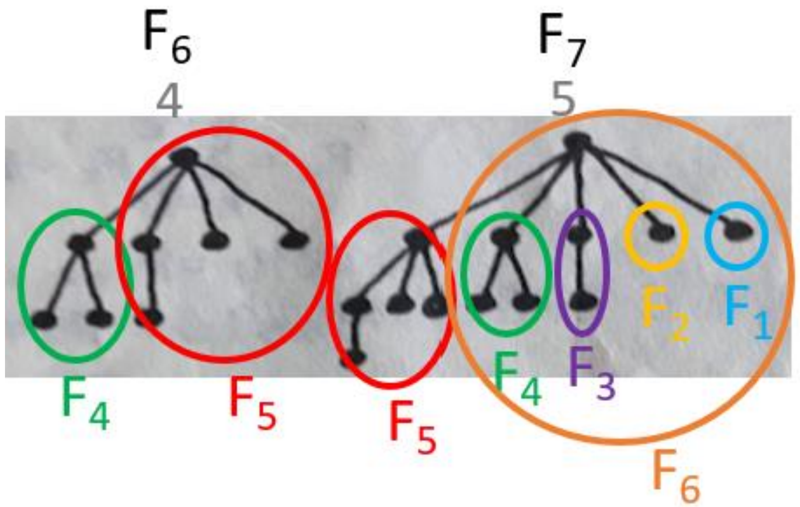
Amortized cost of delete-min

See Theorem 9.2 (P.447)

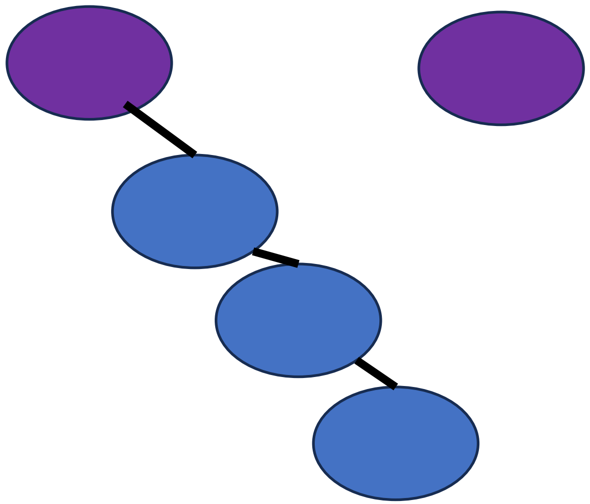
- Actual cost of delete-min $O(\log n + s)$
- $s = \underbrace{\#insert + lastSize}_{\text{Scan all of the min trees}} + \underbrace{u - 1}_{\text{Perform min-tree joining}}$
  - *lastSize*: changes in the number of min trees
  - *u*: the degree of the min node (Each subtree becomes new min trees in top-level.)
- Amortization:
  - *# insert*: forward to each of previous insert operations
  - *lastSize*: forward to previous delete-min, delete, and decrease key operations.
  - $u < \log_2(n)$
- Amortized cost of delete-min $O(\log n)$

Height of F-heap

- Most of the time, it is O(log n) if there are many subtrees.



- After a sequence of operations, when will be the height =O(n)?
  - Between two delete (or delete-min):
    - Number of insert and decrease key operations are Θ(n).
    - No cascading cut occurs.



Note: O(k) equal or less than ak+b
Θ(k) equal to ak+b
k can be a large value. a and b are constant numbers.

Summary

- Fibonacci heaps
  - Node structure
  - Operations: Delete, Decrease key.
- Cascading Cut
- Consolidation
- Time complexity
- Height of F-heap