Design and Analysis of Algorithm

Advanced Data Structure
(Binomial Heap)

LECTURE 41 - 44

Overview

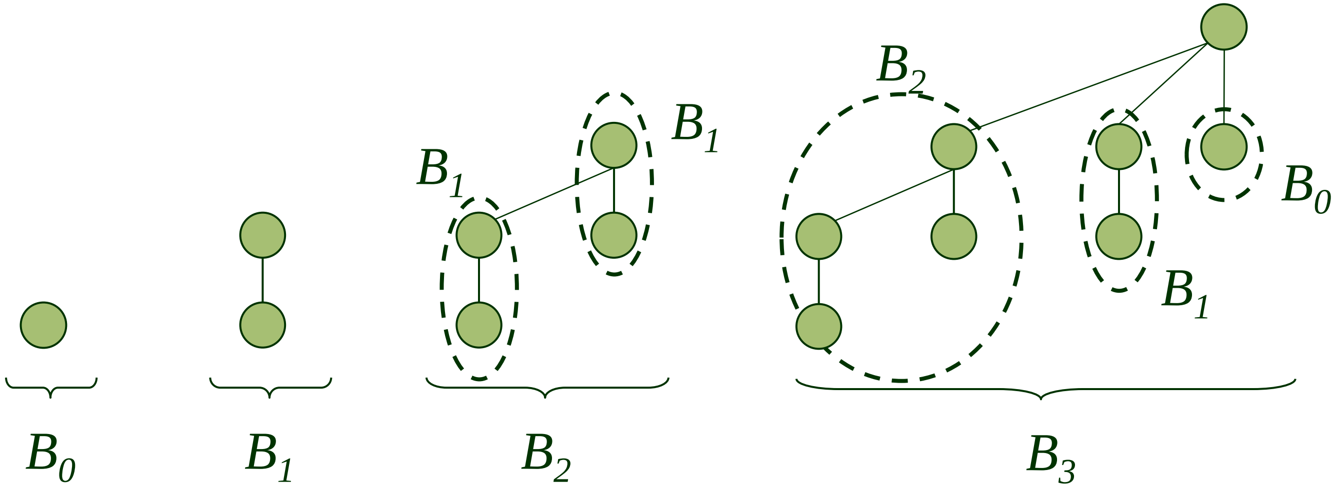
- This section present a data structure known as mergeable heaps, which support the following seven operations.
  - MAKE-BINOMIAL-HEAP
  - BINOMIAL-HEAP-INSERT
  - BINOMIAL-HEAP-MINIMUM
  - BINOMIAL-HEAP-EXTRACT-MIN
  - BINOMIAL-HEAP-UNION
  - BINOMIAL-HEAP DECREASE-KEY
  - BINOMIAL-HEAP-DELETE

Binomial Heap

- Binomial heap was introduced in 1978 by Jean Vuillemin.
- Jean Vuillemin is a professor in mathematics and computer science.
- The other name of Binomial Heap is Mergeable heaps.
- A binomial heap is a collection of binomial trees.
  - Lets learn What is Binomial Tree?

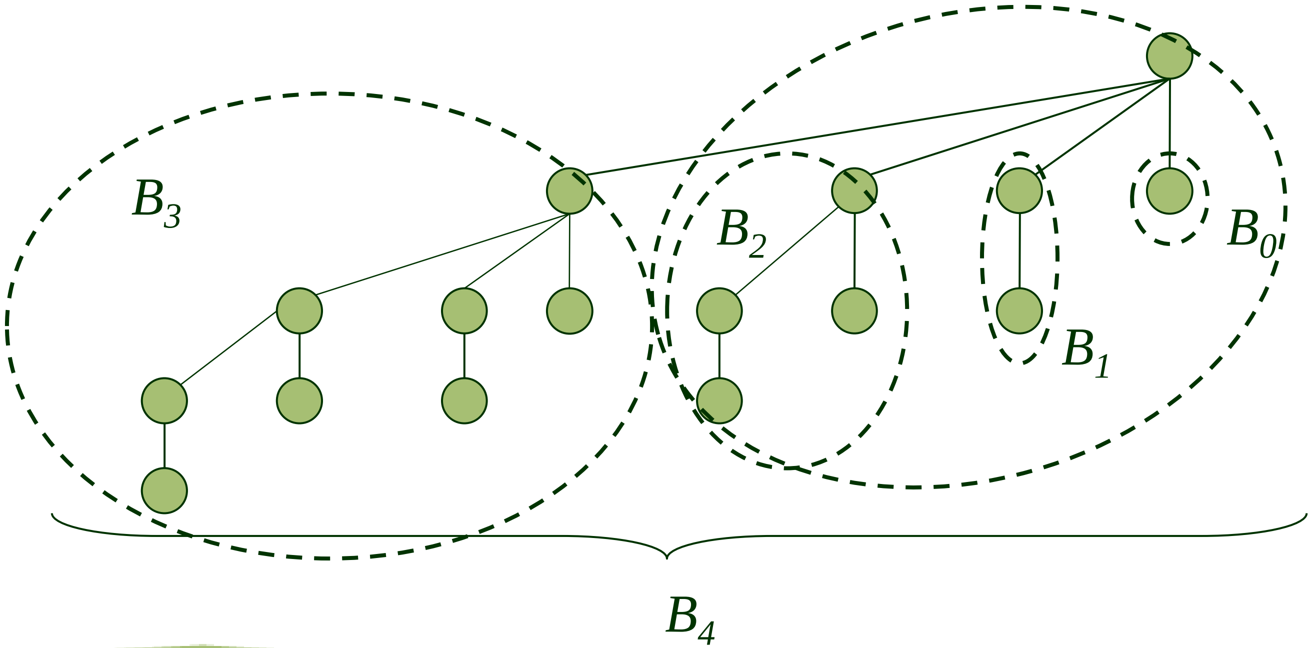
Binomial Tree

- Binomial tree B_k is an ordered tree defined recursively.
- The binomial tree B_0 has one node.
- The binomial tree B_k consists of two binomial trees B_{k-1} and they are connected such that the root of one tree is the leftmost child of the other.



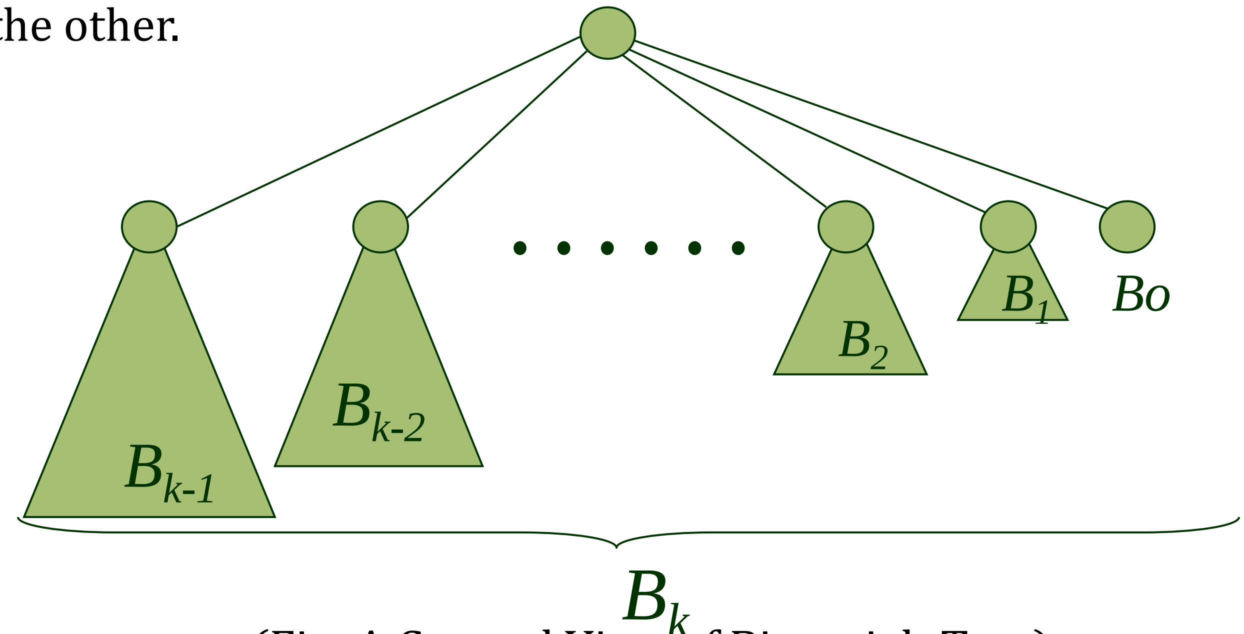
Binomial Tree

- Binomial tree B_k is an ordered tree defined recursively.
- The binomial tree B_0 has one node.
- The binomial tree B_k consists of two binomial trees B_{k-1} and they are connected such that the root of one tree is the leftmost child of the other.



Binomial Tree

- Binomial tree B_k is an ordered tree defined recursively.
- The binomial tree B_0 has one node.
- The binomial tree B_k consists of two binomial trees B_{k-1} and they are connected such that the root of one tree is the leftmost child of the other.



(Fig: A General View of Binomial -Tree)

Binomial Tree (Property)

A Binomial tree satisfy the following properties:

- B_k has 2^knodes
- B_k has height k
- There are exactly $\binom{k}{i}$ combination of nodes at depth i for i=0, 1, 2,…,k.

  For Example:

  Lets check in B_4 and depth $2 (i.e. k = 4 \; and \; i = 2)$

  $$\binom{k}{i} = \frac{k!}{i!(k-i)!} = \binom{4}{2} = \frac{4!}{2! \; (4-2)!} = \frac{4 \; X \; 3 \; X \; 2 \; X \; 1}{2 \; X \; 1 \; X \; 2 \; X \; 1} = 6$$

  Hence 6 numbers of nodes are available in depth 2 of B_4
- The root has degree k which is greater than other node in the tree. Each of the root's child is the root of a subtree Bi.

Binomial Heap (Property)

A Binomial Heap H is a set of binomial trees that satisfies the following properties:

- P1. Each binomial tree in H obeys the **min heap property.**

  (i.e. key of a node is greater or equal to the key of its parent. Hence the root has the smallest key in the tree).

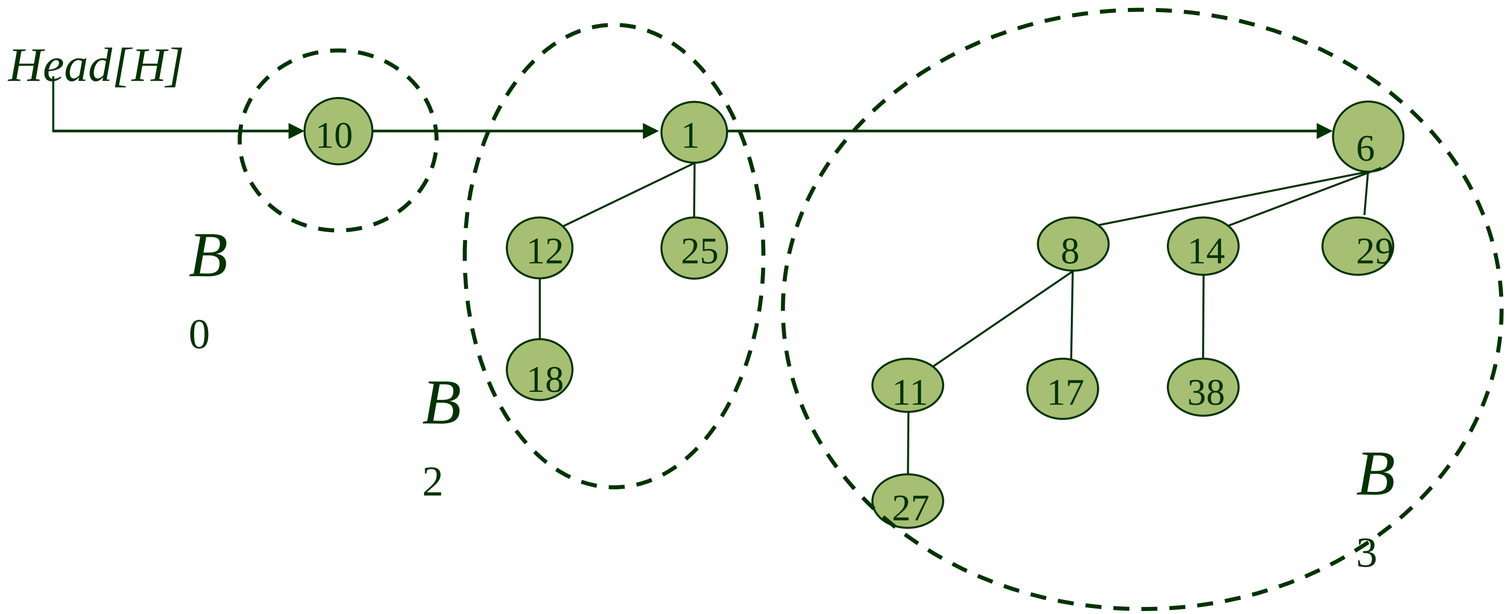
- P2. For any non negative integer k, there is at most one binomial tree whose root has degree k.

  (e.g. it implies that an n node Binomial heap H consists of at most $\lfloor \log n \rfloor + 1$ binomial Tree. (Fig. is available in next page)

- P3. The binomial trees in the binomial heap are arranged in increasing order of degree.

Binomial Heap (Property)

- Example:



Here n=13

So $\lfloor \log n \rfloor + 1 = \lfloor \log 13 \rfloor + 1 = 3 + 1 = 4$ (So at most 4)

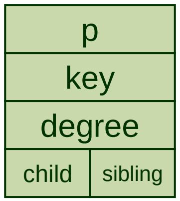
The Above Figure of Binomial Heap consists of B_0 , B_2 and B_3

Binomial Heap (Representation)

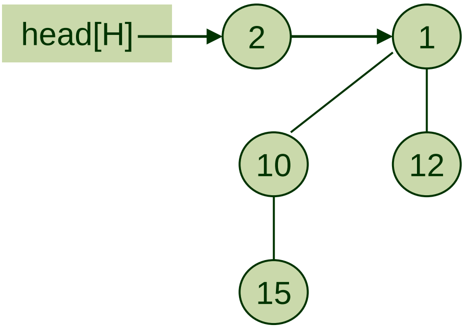
- Each binomial tree within a binomial heap is stored in the left-child, right-sibling representation
- Each node x contains POINTERS
  - $p[x] \rightarrow parent$ to its parent
  - $key[x] \rightarrow key$ to its key value
  - $child[x] \rightarrow child$ to its leftmost child
  - $sibling[x] \rightarrow sibling$ to its immediately right sibling
  - $degree[x] \rightarrow degree$ to its degree value (i.e. denotes the number of children of x)

Binomial Heap (Representation)

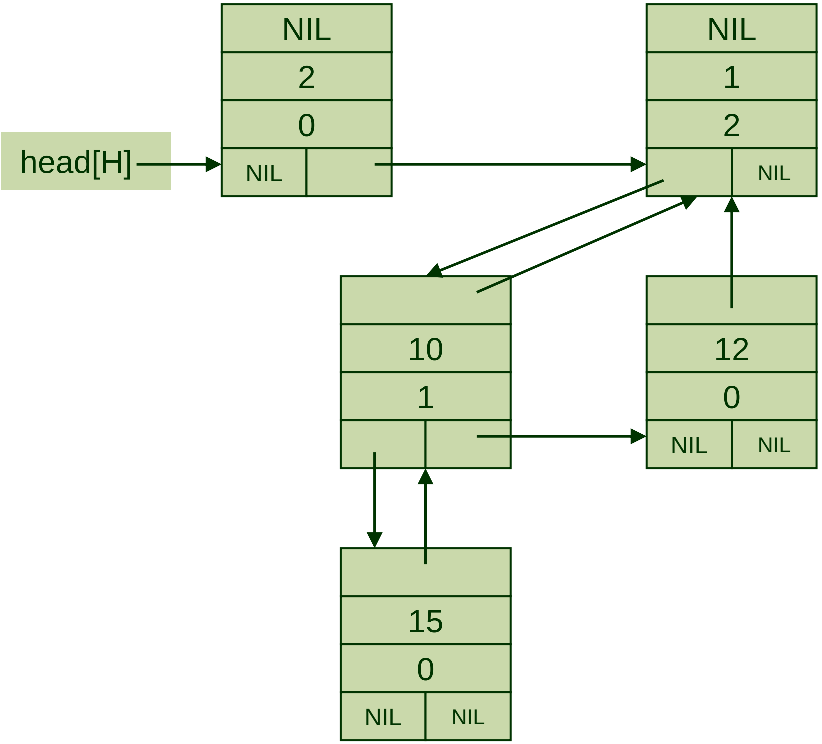
a)



b)



c)



Binomial Heap (Operations)

Binomial Heap support the following five operations:

1. **MAKE-HEAP()** creates and returns a new heap containing no elements.
2. **MINIMUM(H)** returns a pointer to the node in heap H whose key is minimum.
3. **UNION(H1, H2)** creates and returns a new heap that contains all the nodes of heaps H1 and H2. Heaps H1 and H2 are "destroyed" by this operation.
4. **EXTRACT-MIN(H)** deletes the node from heap H whose key is minimum, returning a pointer to the node.

Binomial Heap (Operations)

5. **INSERT(H, x)** inserts node x, whose key field has already been filled in, into heap H.
6. **DECREASE-KEY(H, x, k)** assigns to node x within heap H the new key value k, which is assumed to be no greater than its current key value.[1]
7. **DELETE(H, x)** deletes node x from heap H

Binomial Heap (Operations_1)

1. **MAKE-HEAP() :** creates and returns a new heap containing no elements.

To make an empty binomial heap, the MAKE-BINOMIAL-HEAP procedure simply allocates and returns an object H , where $head[H] = NIL$.

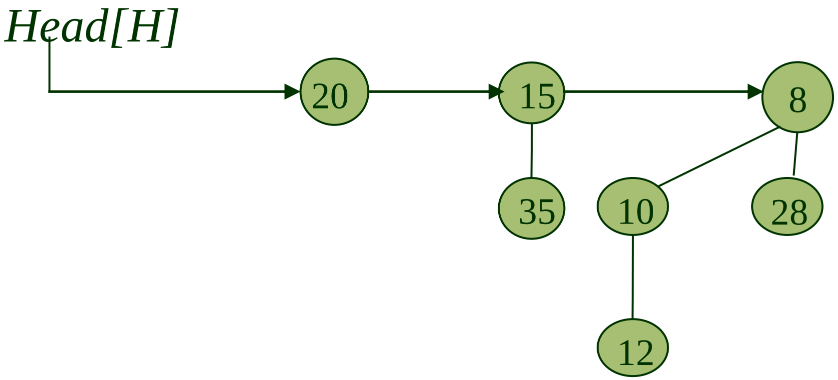
The running time is $\Theta(1)$.

Binomial Heap (Operations_2)

2. MINIMUM(H) : Since the binomial heap is a min-heap-order, the minimum key of each binomial tree must be at the root. This operation checks all the roots to find the minimum key.

Binomial Heap (Operations_2)

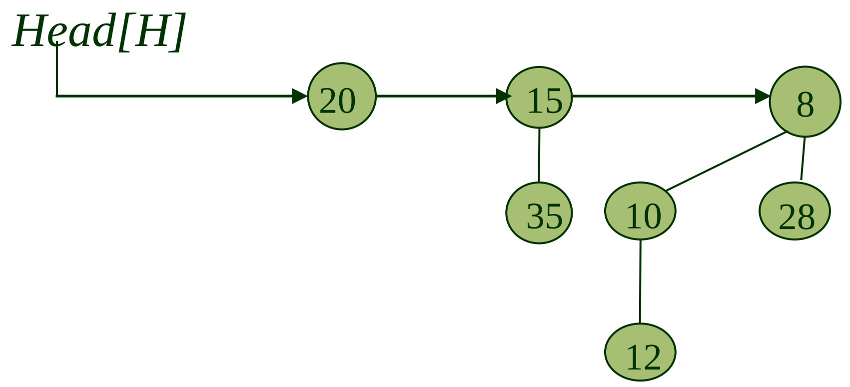
2. MINIMUM(H) : Since the binomial heap is a min-heap-order, the minimum key of each binomial tree must be at the root. This operation checks all the roots to find the minimum key.



Binomial Heap (Operations_2)

2. MINIMUM(H) : Since the binomial heap is a min-heap-order, the minimum key of each binomial tree must be at the root. This operation checks all the roots to find the minimum key.

Pseudocode: This implementation assumes that there are no keys with value ∞.




```
BINOMIAL-HEAP-MINIMUM(H)
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL
      do if key[x] < min
            then min ← key[x]
                 y ← x
      x ← sibling[x]
return y
```

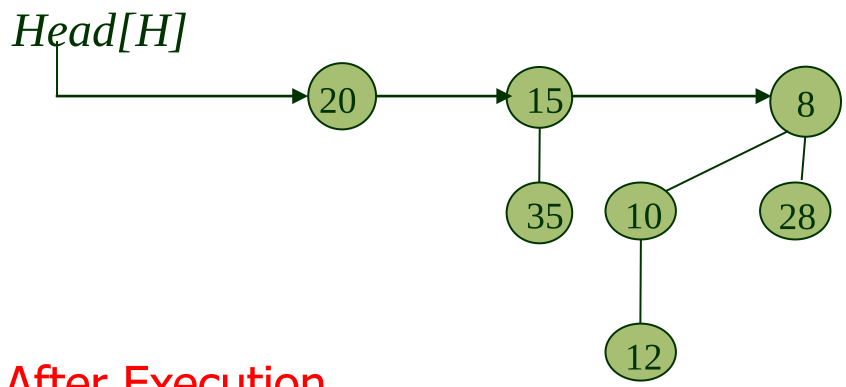
Binomial Heap (Operations_2)

2. MINIMUM(H) : Since the binomial heap is a min-heap-order, the minimum key of each binomial tree must be at the root. This operation checks all the roots to find the minimum key.

Pseudocode: This implementation assumes that there are no keys with value ∞.

BINOMIAL-HEAP-MINIMUM(H)

```
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL
      do if key[x] < min
            then min ← key[x]
            y ← x
      x ← sibling[x]
return y
```



After Execution
Return y=8

Binomial Heap (Operations_2)

Since binomial heap is Heap-ordered and the minimum key must reside in a ROOT node. The **BINOMIAL-HEAP-MINIMUM(H)** checks all roots in $O\ (lgn)$.

Because,

$$Number\ of\ Roots\ in\ Binomial\ Heap\ is\ at\ least\ \lfloor \log n \rfloor + 1(\text{ property 2})$$

$$\text{Hence } RUNNING\text{–}TIME\ =\ O\ (lgn)$$

Binomial Heap (Operations_3)

3. UNION(H1, H2)

This operation consists of the following steps

- Merge two binomial heaps H1 and H2. The resulting heap has the roots in increasing order of degree
- For each tree in the binomial heap H, if it has the same order with another tree, link the two trees together such that the resulting tree obeys min-heap-order.

For this there is an requirement of 3 pointers into the root list

x = points to the root currently being examined

prev-*x* = points to the root PRECEDING *x* on the root list
sibling [prev-*x*] = *x*

next-*x* = points to the root FOLLOWING *x* on the root list
sibling [*x*] = next-*x*

Binomial Heap (Operations_3)

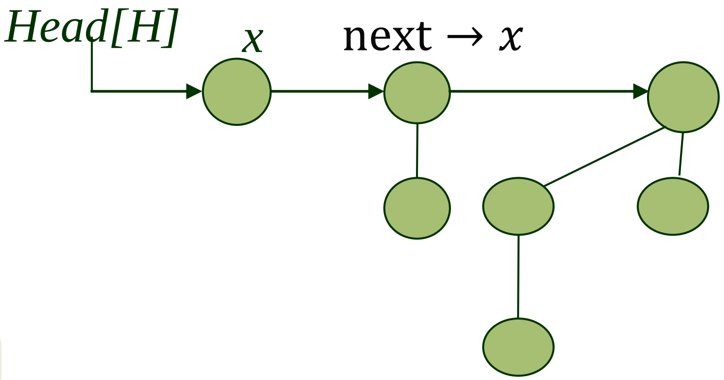
This operation perform by the help of 4(four) number of cases.

Case 1: $\boldsymbol{if}\ (\boldsymbol{degree}[\boldsymbol{x}] \neq \boldsymbol{degree}[\boldsymbol{next} \rightarrow \boldsymbol{x}])$

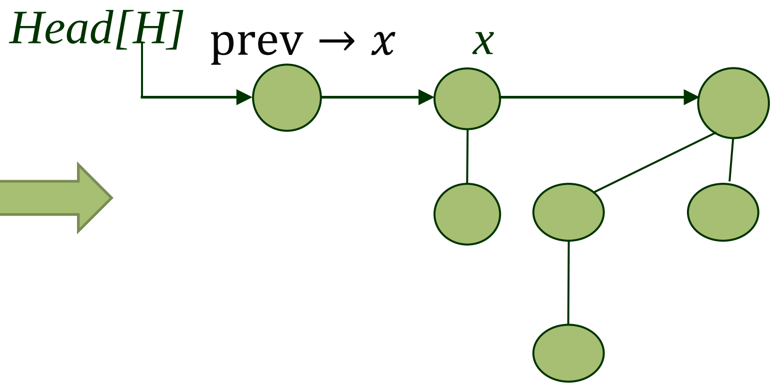
$\text{prev} \rightarrow x = x$

x=next → x

Example:



Before Case 1



After Case 1

Binomial Heap (Operations_3)

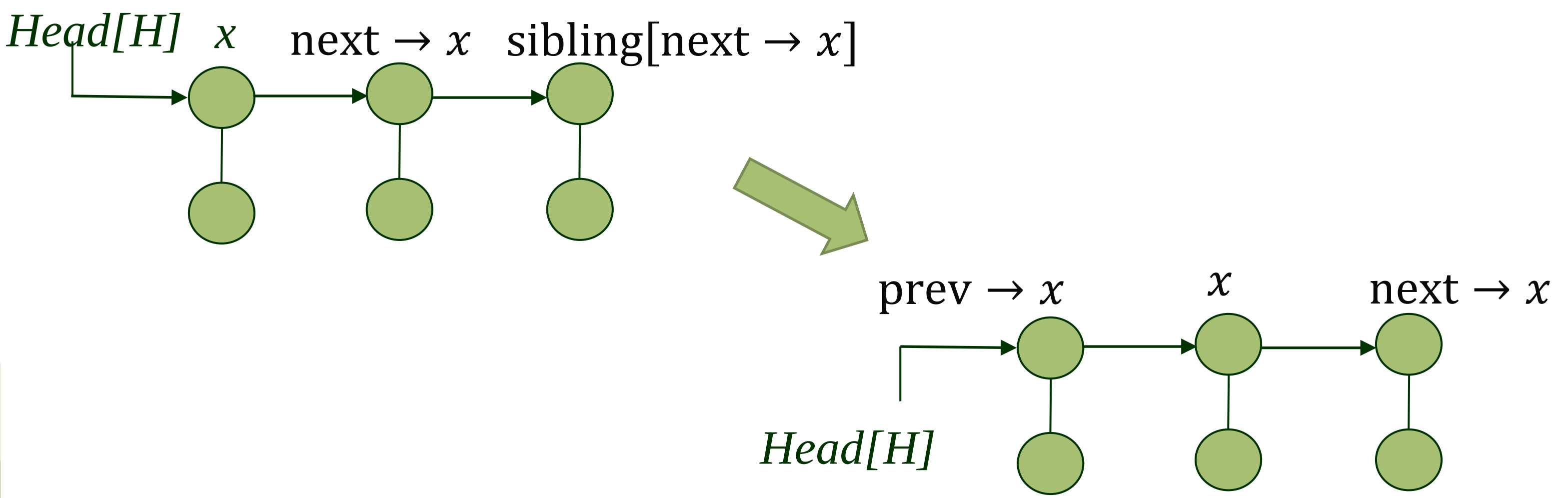
This operation perform by the help of 4(four) number of cases.

Case 2: $\boldsymbol{if}\ (\boldsymbol{degree}[\boldsymbol{x}] = \boldsymbol{degree}[\boldsymbol{next} \rightarrow \boldsymbol{x}] = \boldsymbol{degree}[\boldsymbol{sibling}[\boldsymbol{next} \rightarrow \boldsymbol{x}]])$

$\text{prev} \rightarrow x = x$

x=next → x

Example:



Before Case 2

After Case 2

Binomial Heap (Operations_3)

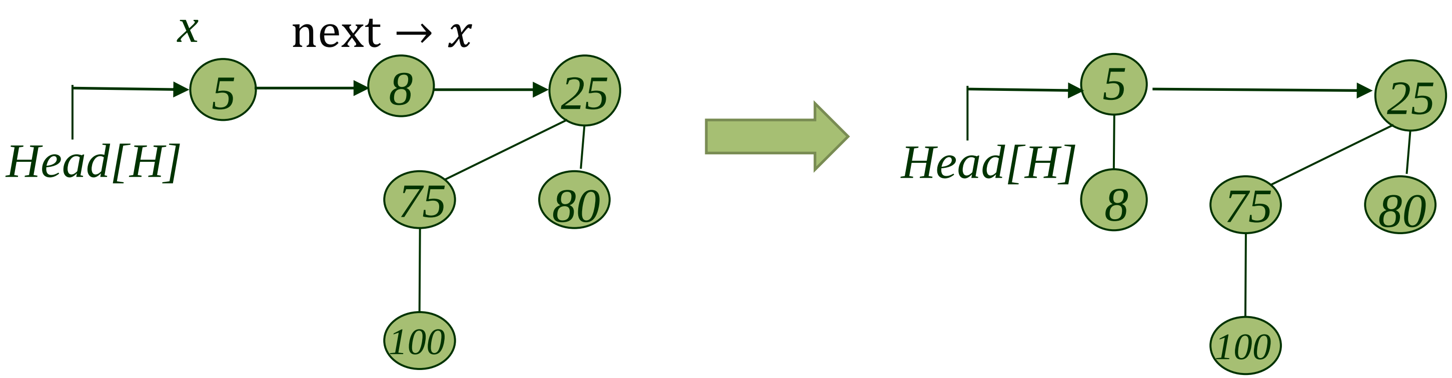
This operation perform by the help of 4(four) number of cases.

Case 3: $\boldsymbol{if}\ (\boldsymbol{degree}[\boldsymbol{x}] = \boldsymbol{degree}[\boldsymbol{next} \to \boldsymbol{x}])$ **and (key[x] ≤ key[next])**

sibling[x]=sibling[next → x]

Binomial Link(next → x, x)

Example:



Before Case 3

After Case 3

Binomial Heap (Operations_3)

This operation perform by the help of 4(four) number of cases.

Case 4: $\boldsymbol{if}\ (\boldsymbol{degree}[\boldsymbol{x}] = \boldsymbol{degree}[\boldsymbol{next} \rightarrow \boldsymbol{x}])$ **and (key[x] ≥ key[next])**

if(prev → x== Null)

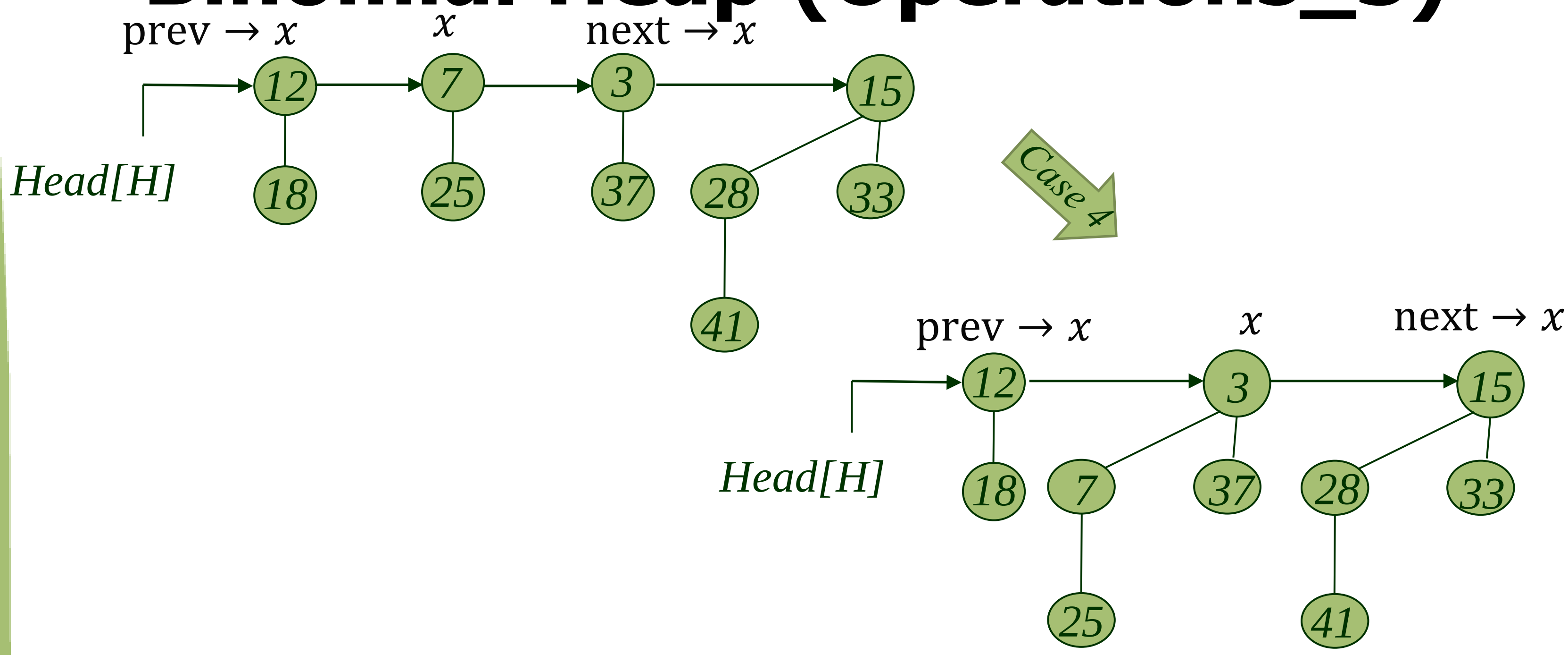
Head[H]= next → x

else

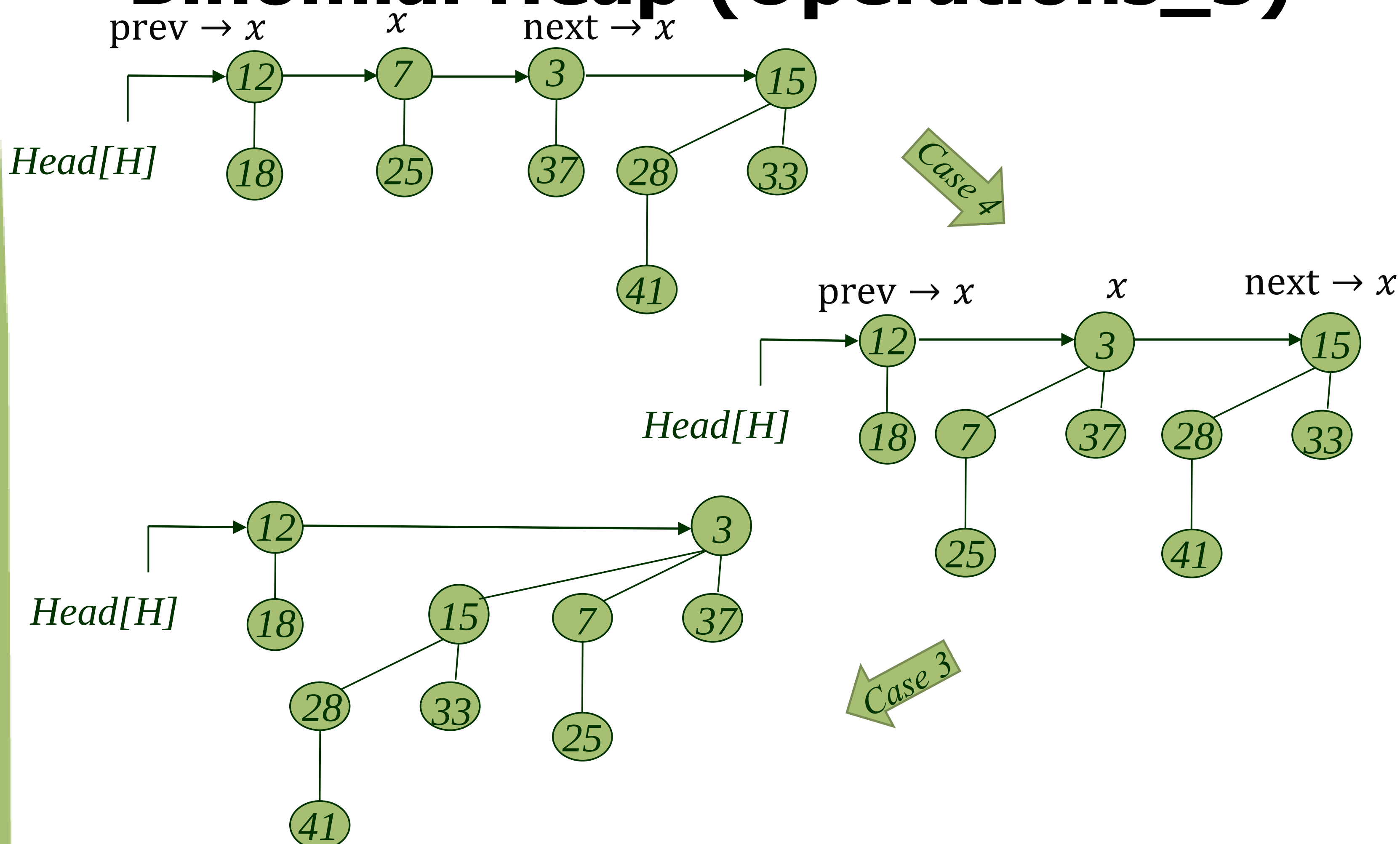
sibling[prev → x]= next → x

Binomial Link(x, next → x)

Binomial Heap (Operations_3)



Binomial Heap (Operations_3)



Binomial Heap (Operations_3)

BINOMIAL-LINK(y, z)

1 p[y] = z

2 sibling[y] = child[z]

3 child[z] = y

4 degree[z] = degree[z] + 1

Binomial Heap (Operations_3)

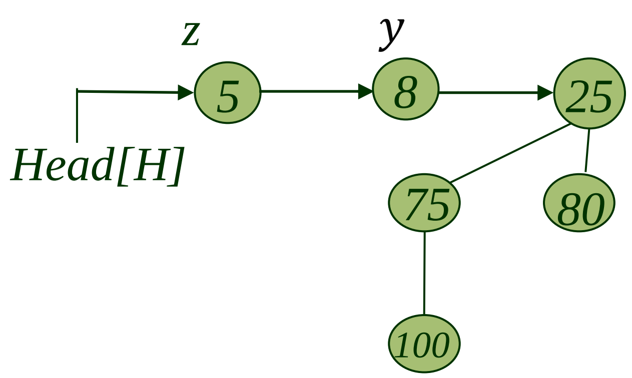
BINOMIAL-LINK(y, z)

1 p[y] = z

2 sibling[y] = child[z]

3 child[z] = y

4 degree[z] = degree[z] + 1



Binomial Heap (Operations_3)

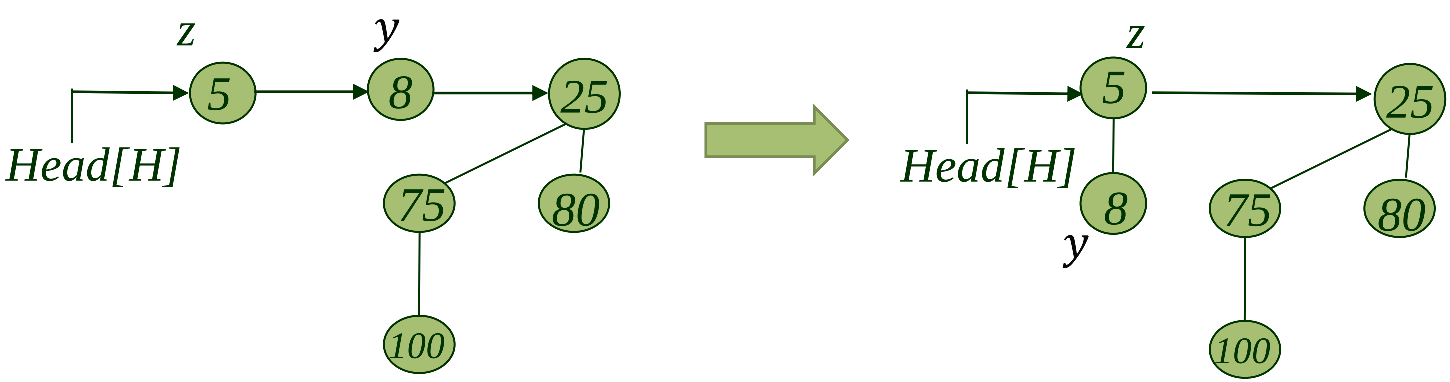
BINOMIAL-LINK(y, z)

1 p[y] = z

2 sibling[y] = child[z]

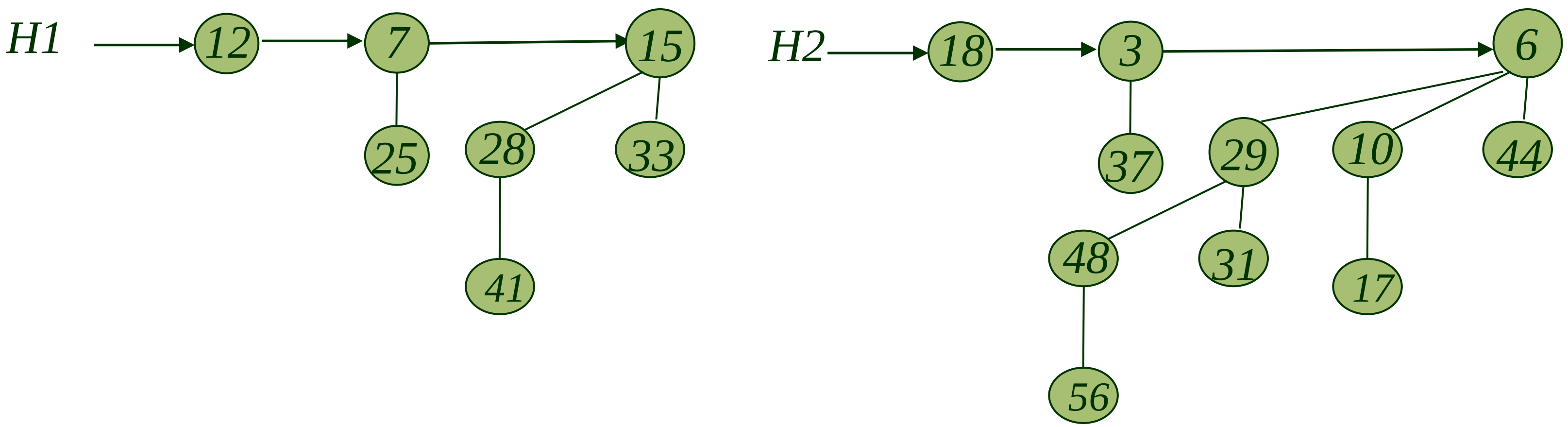
3 child[z] = y

4 degree[z] = degree[z] + 1



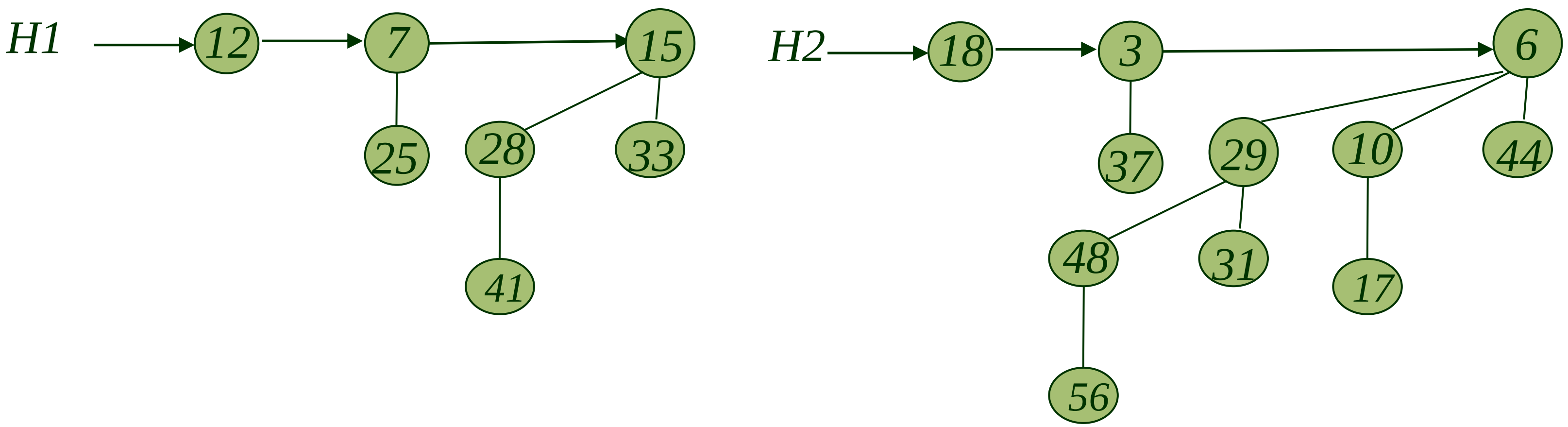
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



Binomial Heap (Operations_3)

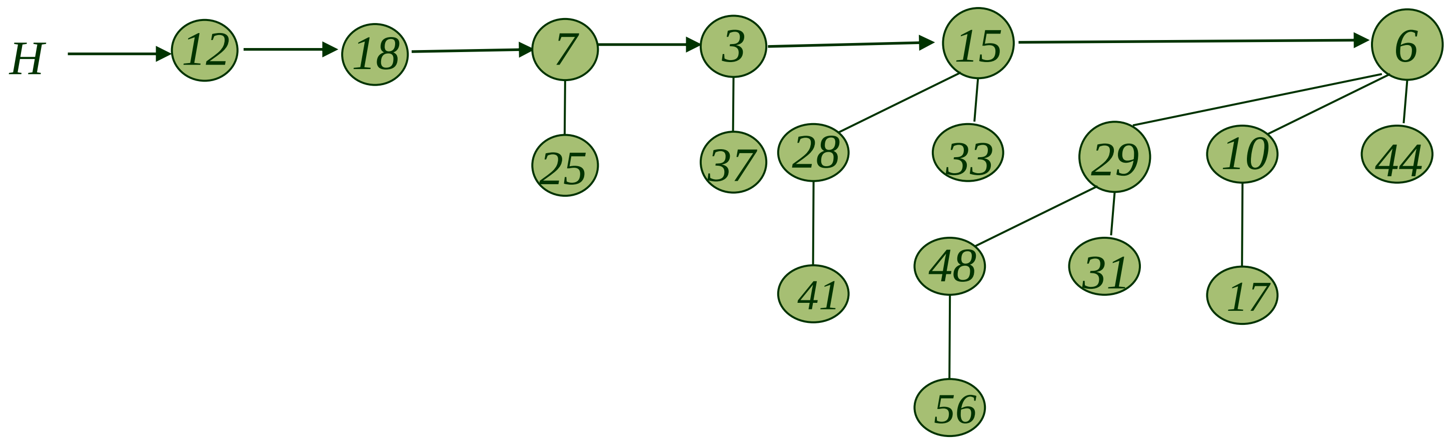
Example 1: Merge the following two Binomial heap H1 and H2.



After Merging…….

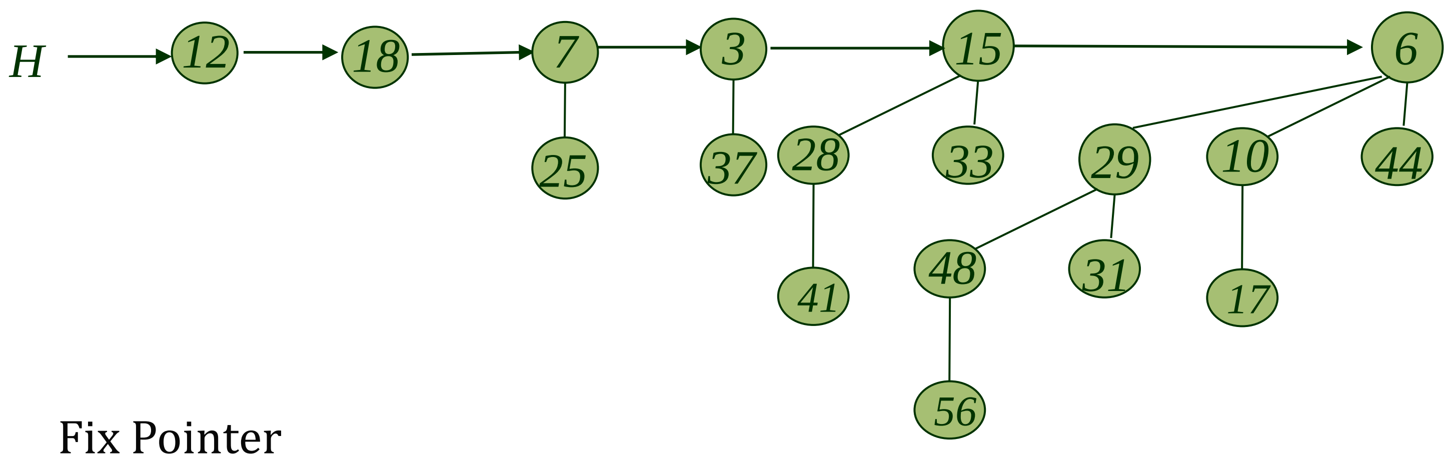
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



Binomial Heap (Operations_3)

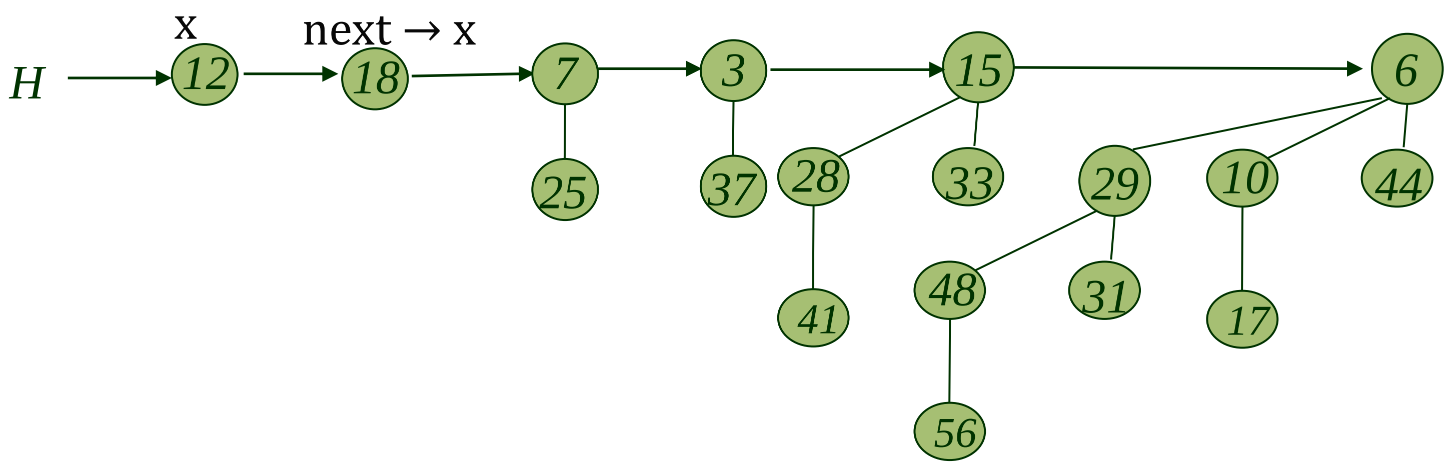
Example 1: Merge the following two Binomial heap H1 and H2.



Fix Pointer

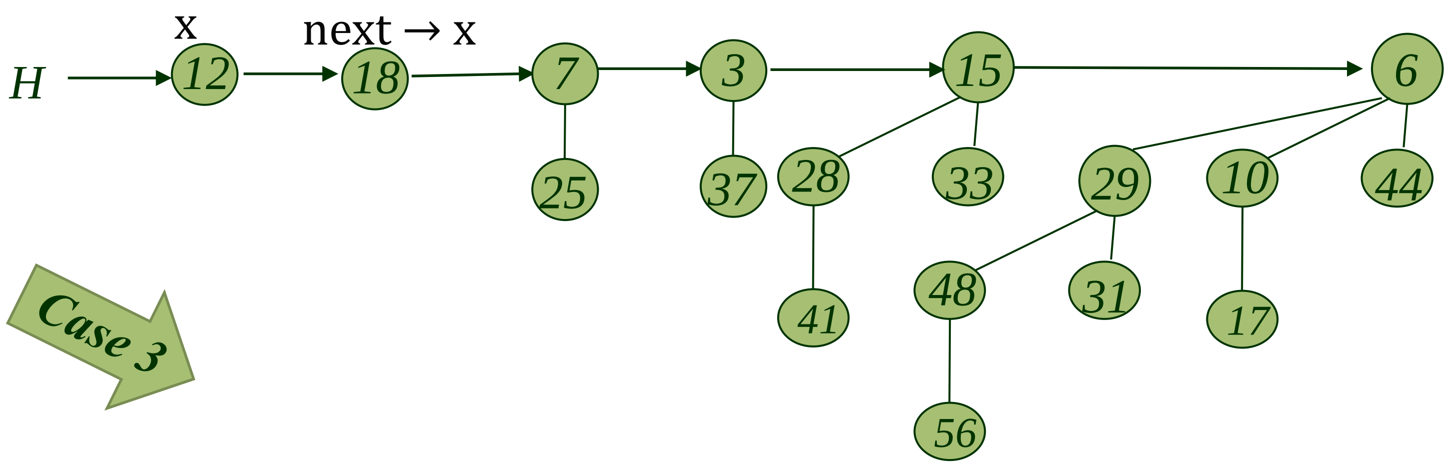
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



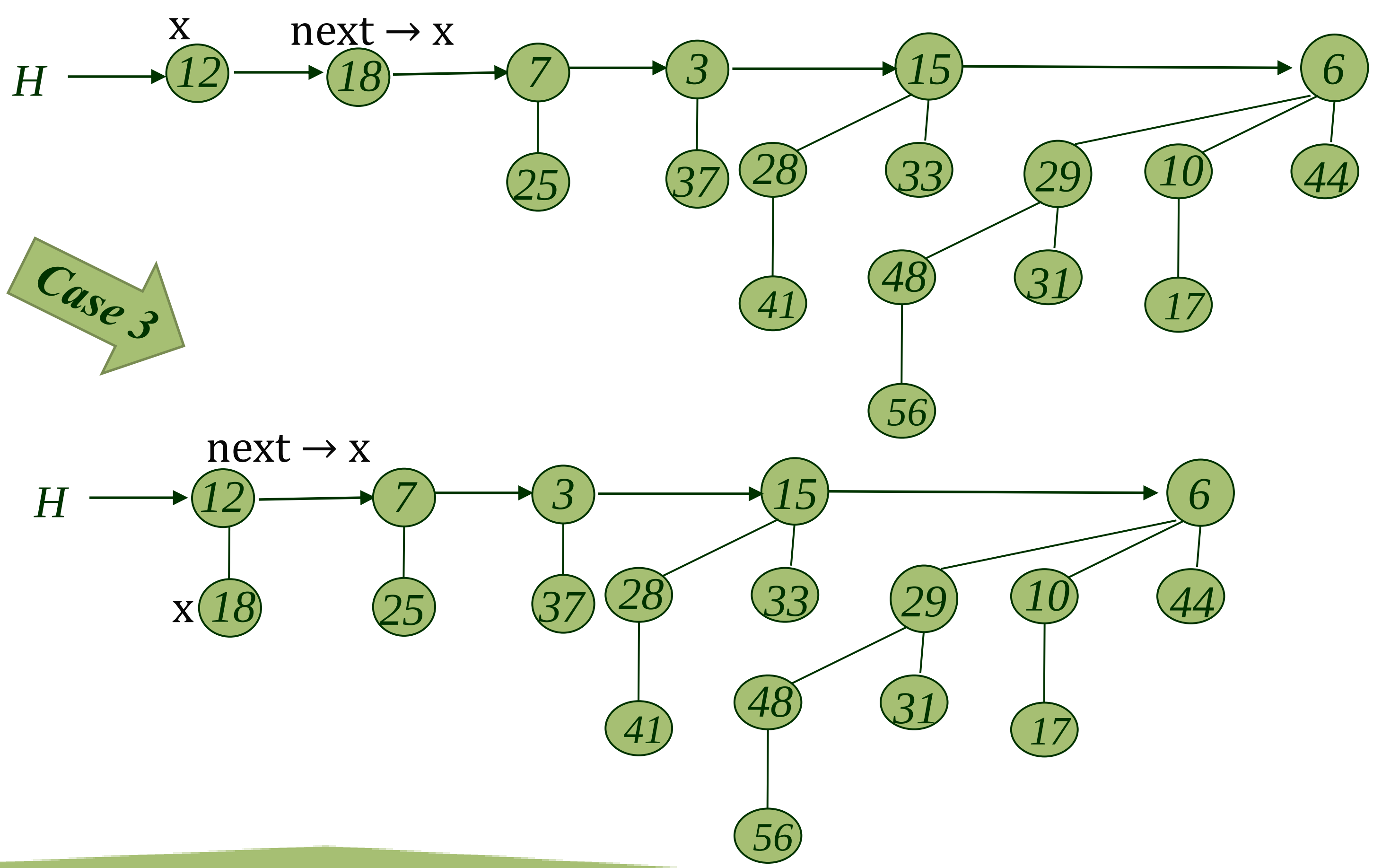
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



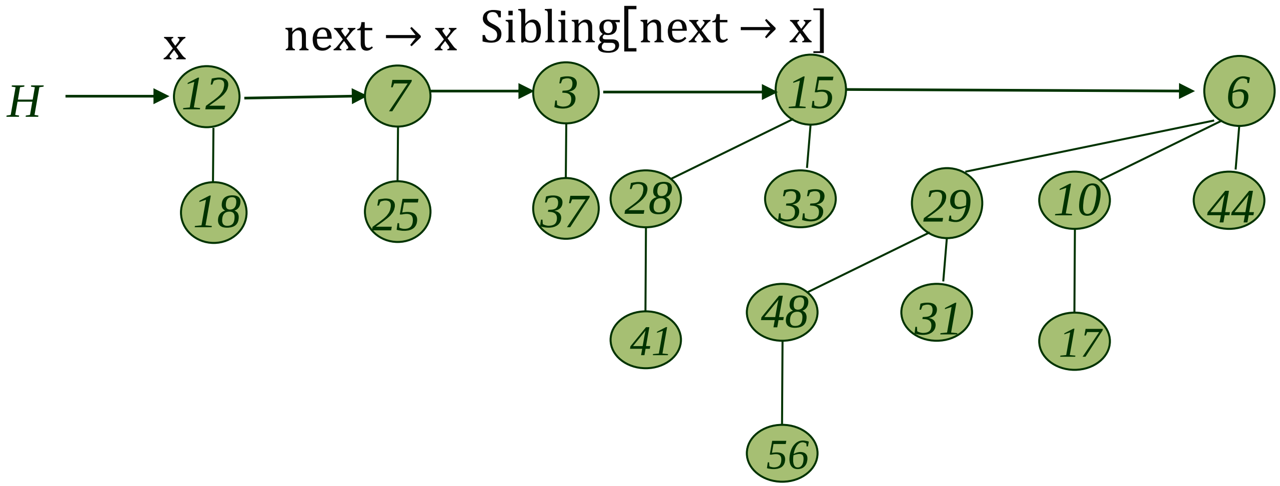
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



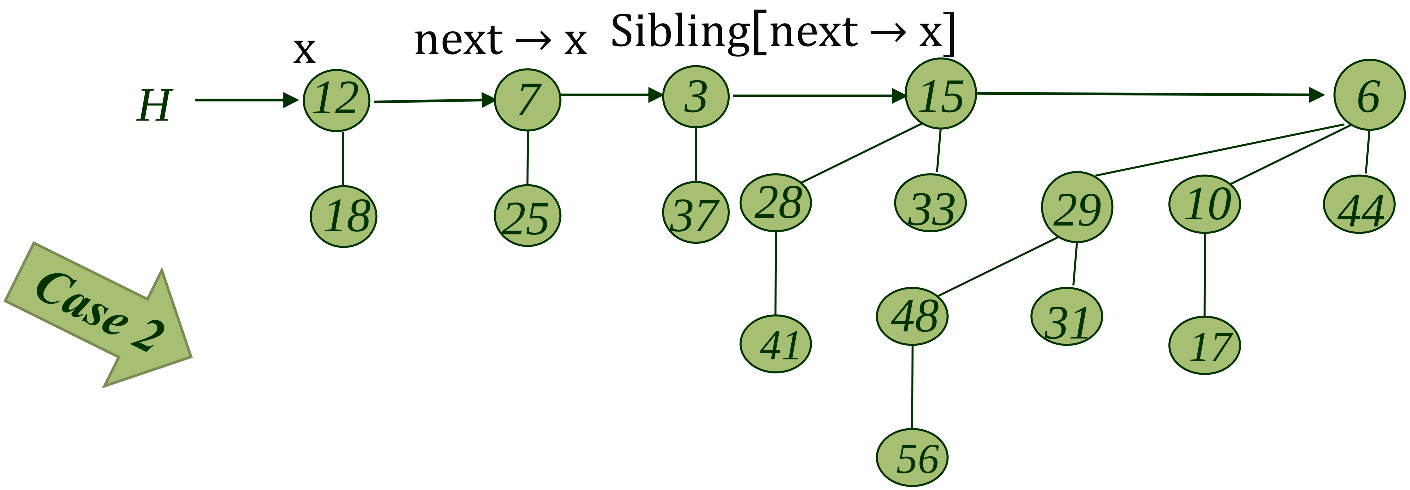
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



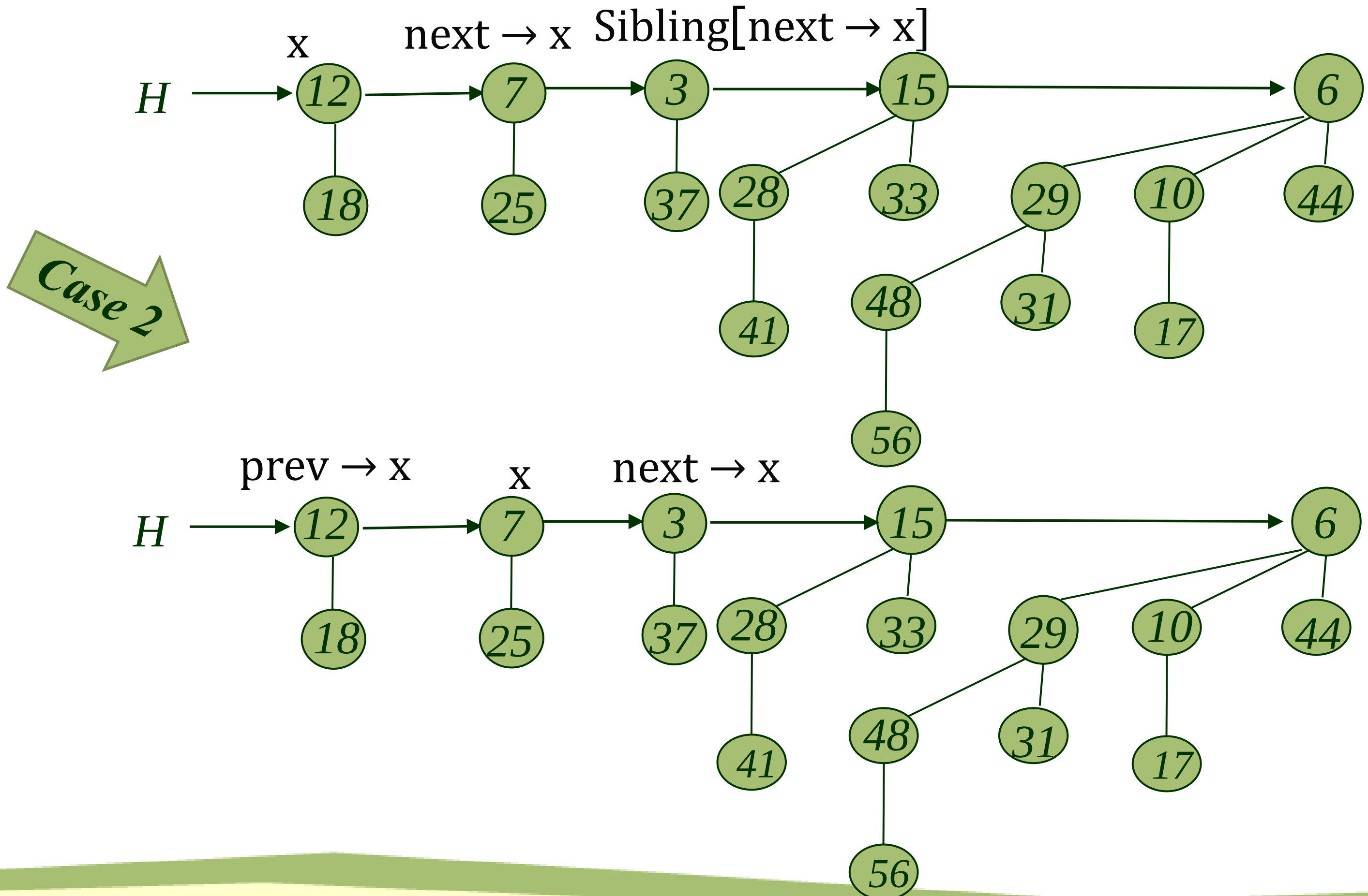
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



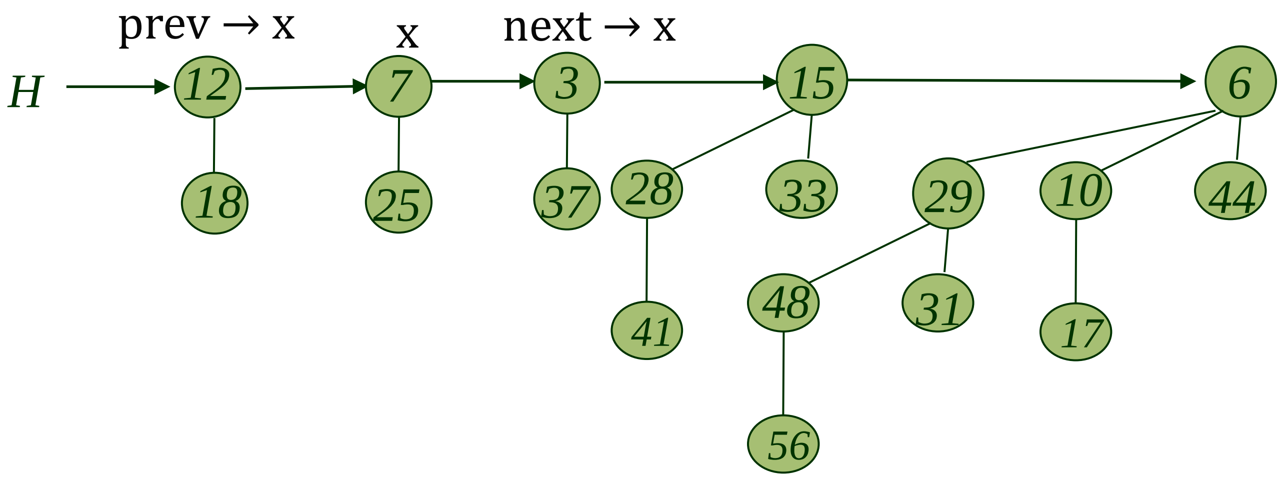
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



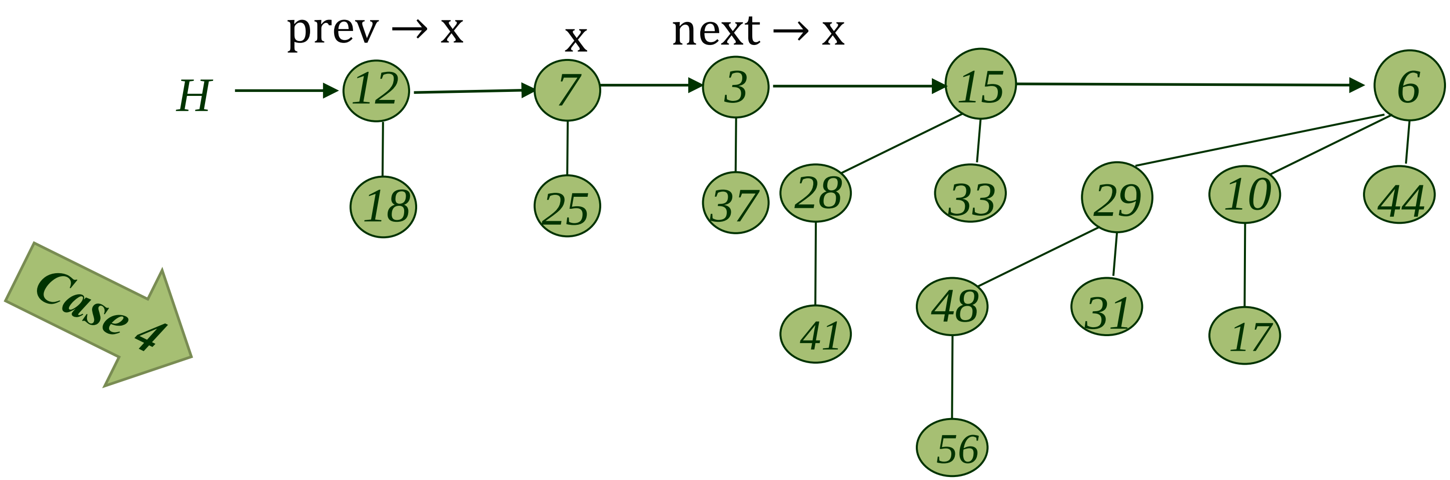
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



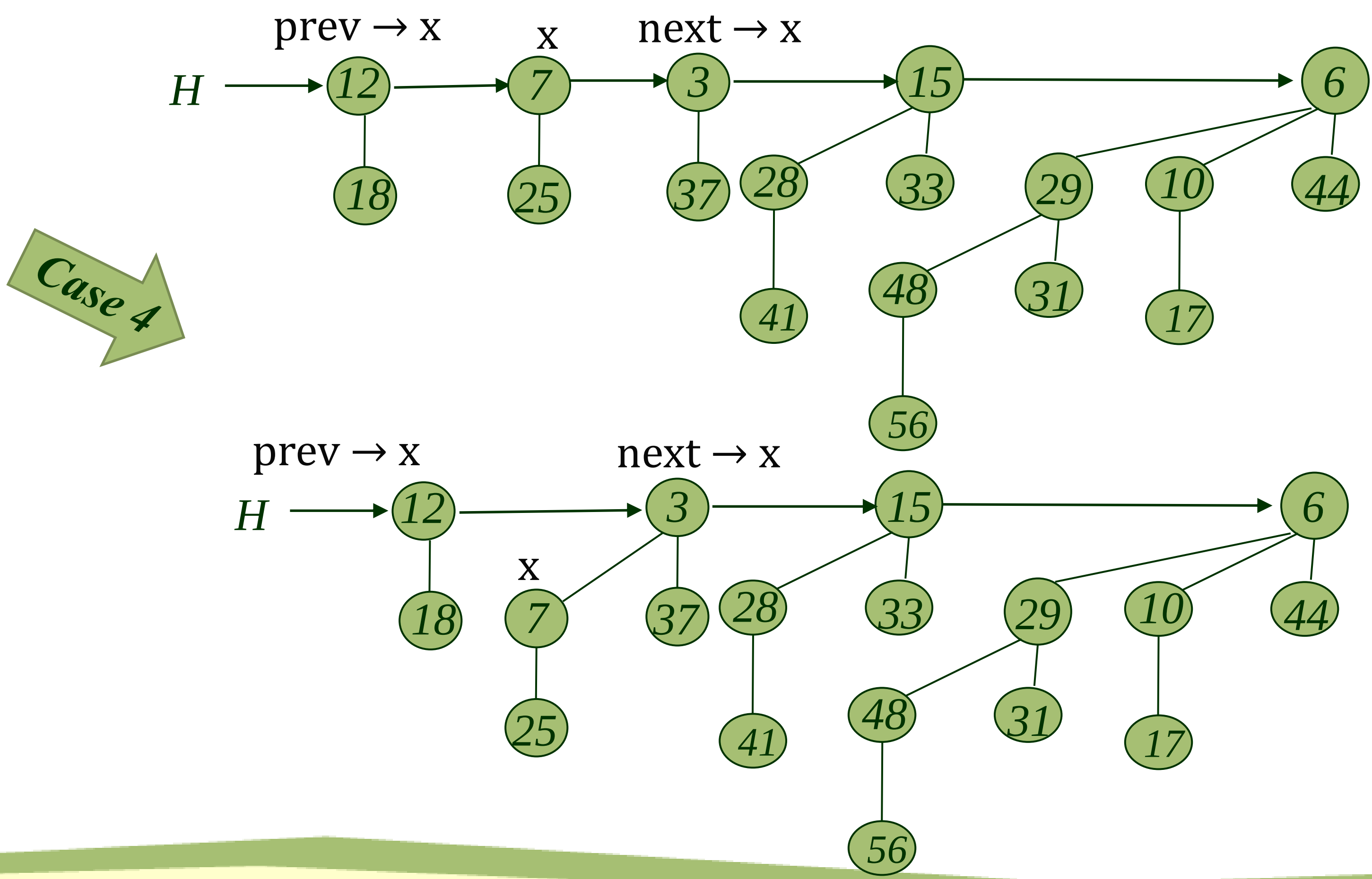
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



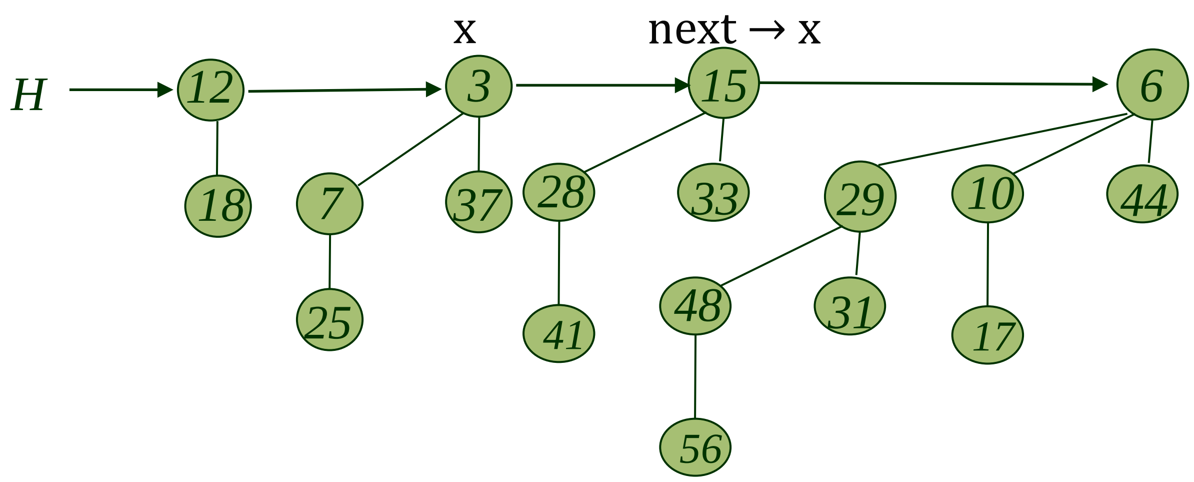
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



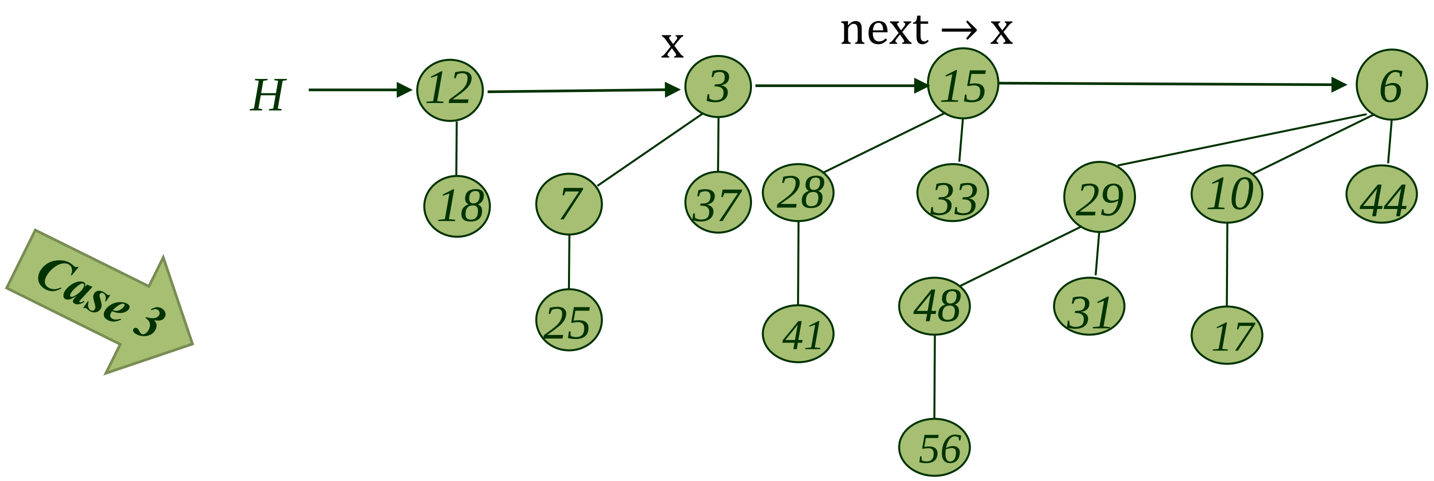
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



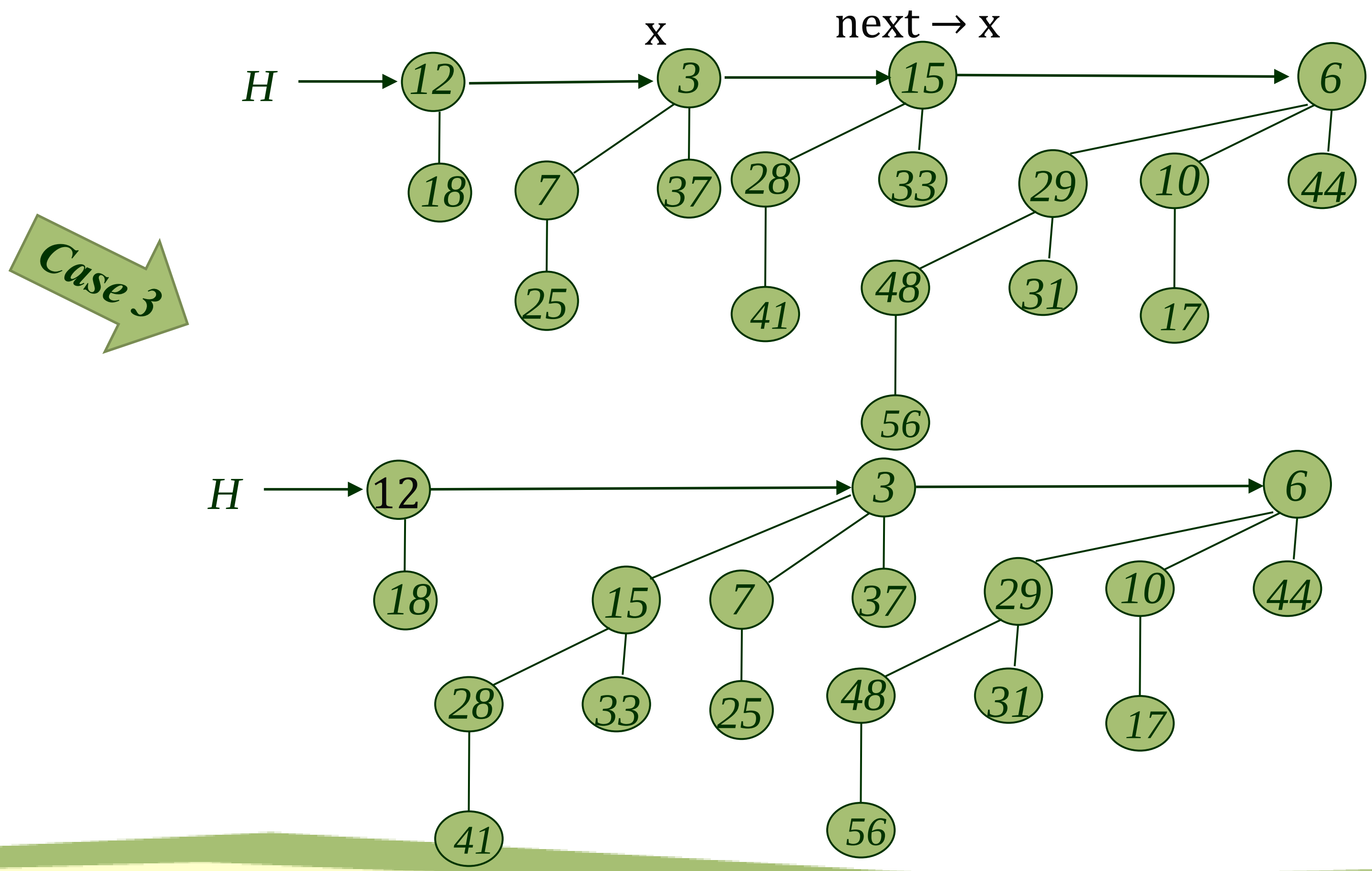
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



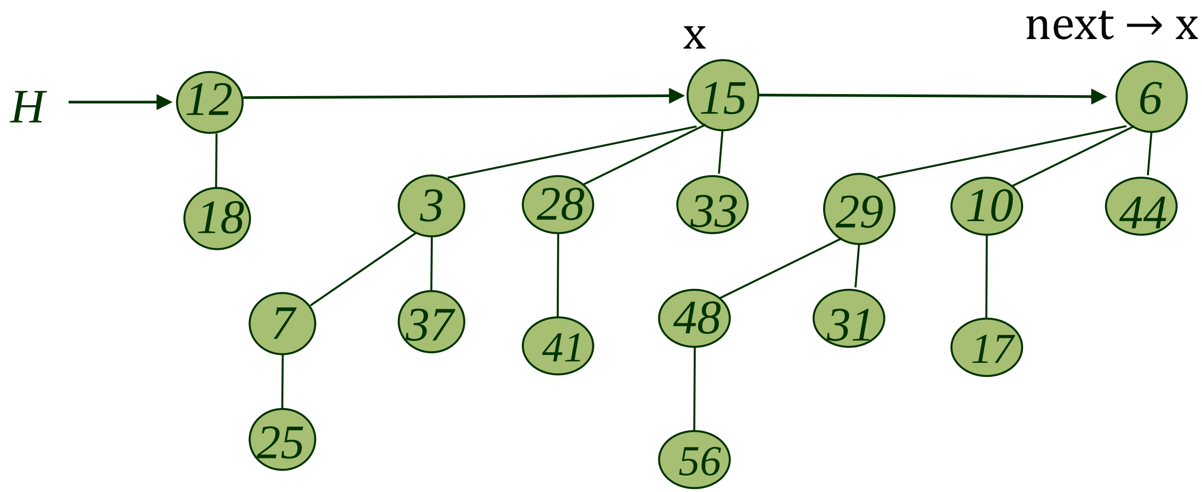
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



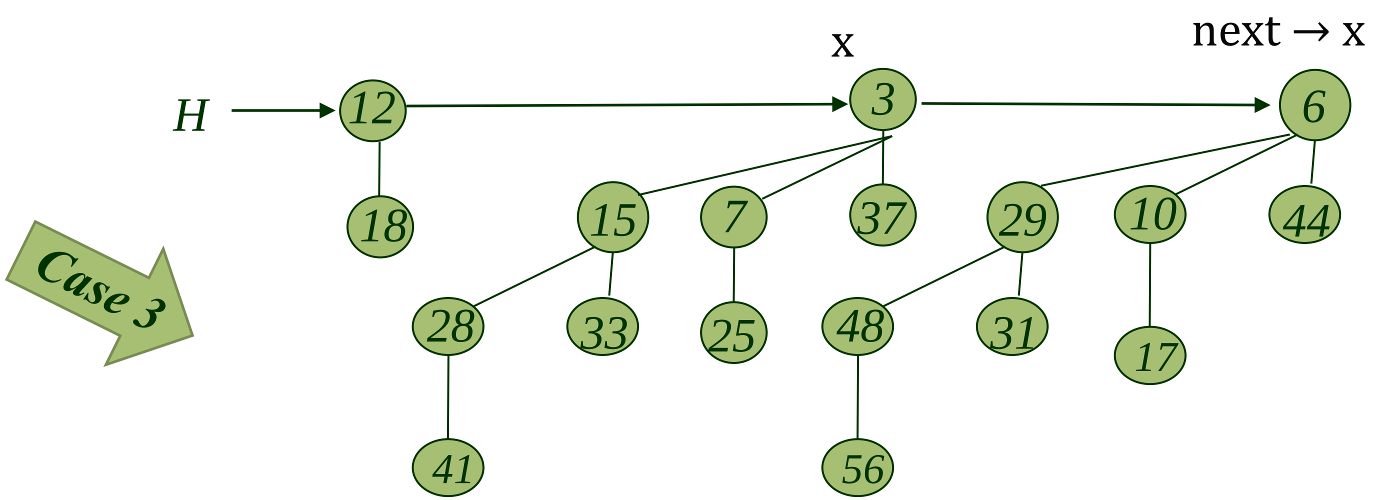
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



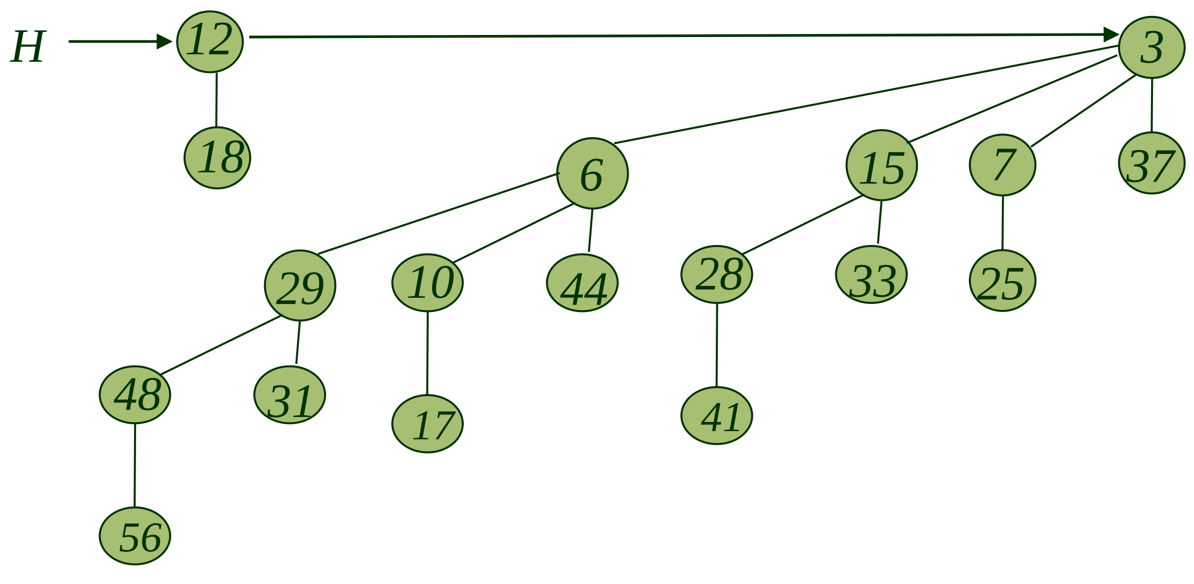
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



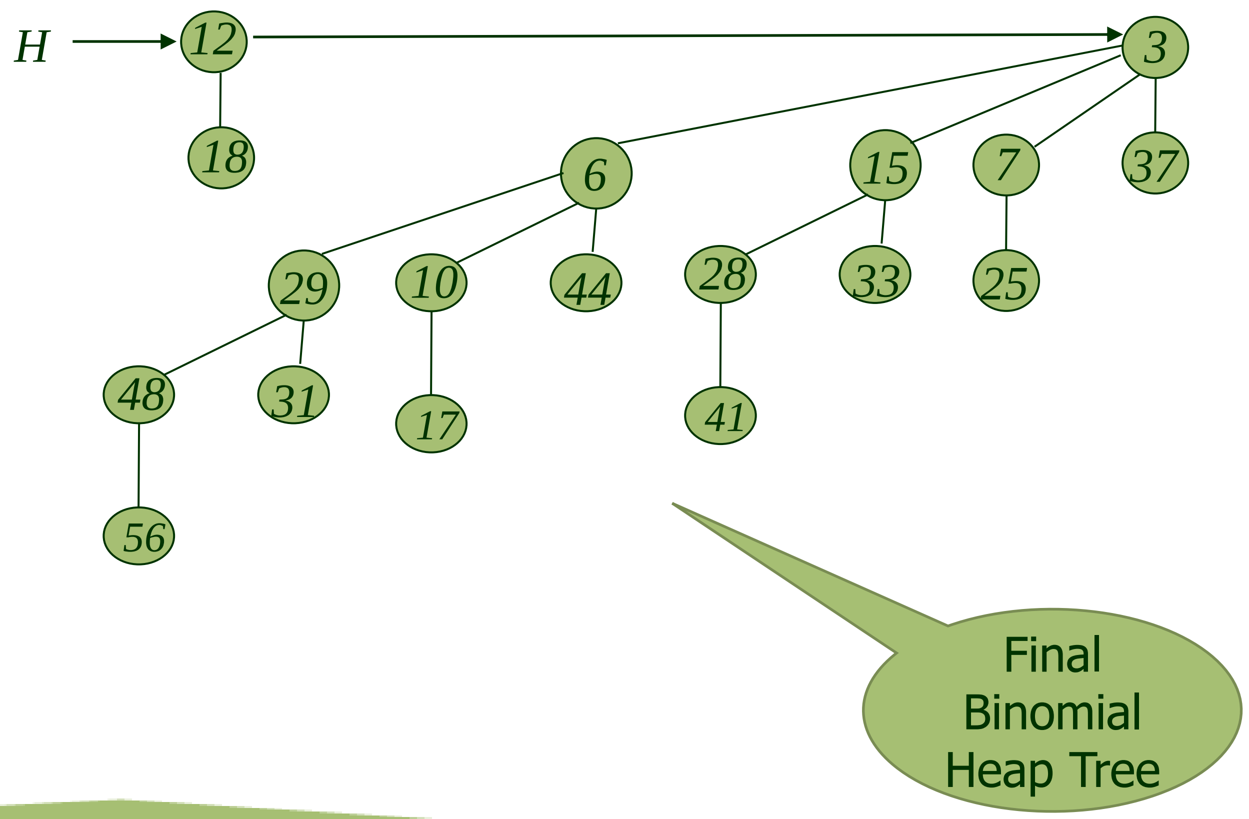
Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



Binomial Heap (Operations_3)

Example 1: Merge the following two Binomial heap H1 and H2.



Binomial Heap (Operations_3)

```
BINOMIAL-HEAP-UNION(H1, H2)
H = MAKE-BINOMIAL-HEAP()
head[H] = BINOMIAL-HEAP-MERGE(H1, H2)
free the objects H1 and H2 but not the lists they point to
if head[H] = NIL
    then return H
prev-x ← NIL
x ← head[H]
next-x ← sibling[x]
```

Binomial Heap (Operations_3)

```
while next-x ≠ NIL
    do if (degree[x] ≠ degree[next-x]) or
    (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
        then prev-x ← x                          ▹ Cases 1 and 2
             x ← next-x                          ▹ Cases 1 and 2
    else if key[x] ≤ key[next-x]
        then sibling[x] ← sibling[next-x]        ▹ Case 3
             BINOMIAL-LINK(next-x, x)            ▹ Case 3
        else if prev-x = NIL                     ▹ Case 4
                then head[H] ← next-x            ▹ Case 4
                else sibling[prev-x] ← next-x    ▹ Case 4
             BINOMIAL-LINK(x, next-x)            ▹ Case 4
             x ← next-x                          ▹ Case 4
    next-x ← sibling[x]
return H
```

Binomial Heap (Operations_3)

```
while next-x ≠ NIL
    do if (degree[x] ≠ degree[next-x]) or
    (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
        then prev-x ← x                          ▹ Cases 1 and 2
            x ← next-x                           ▹ Cases 1 and 2
    else if key[x] ≤ key[next-x]
        then sibling[x] ← sibling[next-x]        ▹ Case 3
            BINOMIAL-LINK(next-x, x)             ▹ Case 3
        else if prev-x = NIL                     ▹ Case 4
            then head[H] ← next-x                ▹ Case 4
            else sibling[prev-x] ← next-x        ▹ Case 4
            BINOMIAL-LINK(x, next-x)             ▹ Case 4
            x ← next-x                           ▹ Case 4
    next-x ← sibling[x]
return H
```

Binomial Heap (Operations_3)

Analysis of BINOMIAL-HEAP-UNION(H1, H2)

The running time of BINOMIAL-HEAP-UNION is $O(\lg n)$, where n is the total number of nodes in binomial heaps H1 and H2.

We can see this as follows.

- Let H1 contain n1 nodes and H2 contain n2 nodes,

  Hence, $n = n1 + n2$.
- Then H1 contains at most $\lfloor \lg n1 \rfloor + 1$ roots.
- and H2 contains at most $\lfloor \lg n2 \rfloor + 1$ roots,
- Hence H contains at most $\lfloor \lg n1 \rfloor + \lfloor \lg n2 \rfloor + 2 \leq 2\lfloor \lg n \rfloor + 2 = O(\lg n)$ roots immediately after the call of BINOMIAL-HEAP-MERGE.
- The time required to perform BINOMIAL-HEAP-MERGE is thus $O(\lg n)$.

Binomial Heap (Operations_3)

Analysis of BINOMIAL-HEAP-UNION(H1, H2)

- Each iteration of the while loop takes O(1) time, and there are at most $\lfloor \lg n1 \rfloor + \lfloor \lg n2 \rfloor + 2$ iterations.

  (because each iteration either advances the pointers one position down the root list of H or removes a root from the root list.)
- Hence the total time required to execute BINOMIAL-HEAP-UNION is O(lg n).

Binomial Heap (Operations_4)

4. EXTRACT-MIN(H)

The following procedure extracts the node with the minimum key from binomial heap H and returns a pointer to the extracted node.

BINOMIAL-HEAP-EXTRACT-MIN(H)

1 find the root x with the minimum key in the root list of H, and remove x from the root list of H

2 H′ ← call MAKE-BINOMIAL-HEAP()

3 reverse the order of the linked list of x's children, and set head[H′] to point to the head of the resulting list

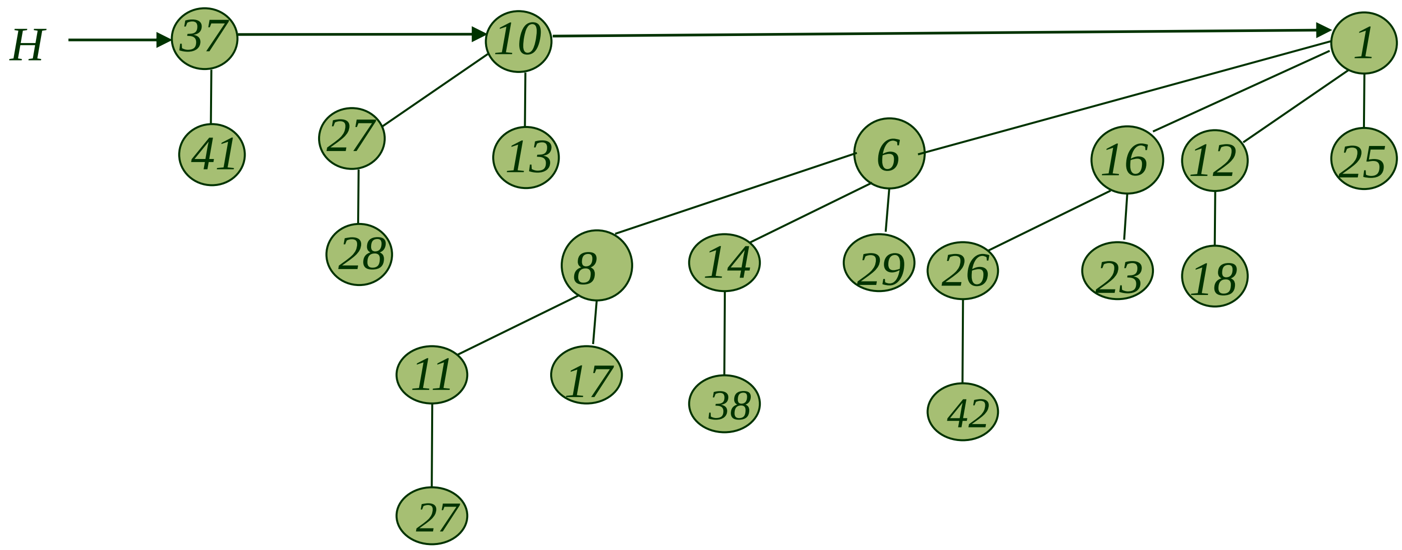
4 H ← call BINOMIAL-HEAP-UNION(H, H′)

5 return x

Binomial Heap (Operations_4)

Example:

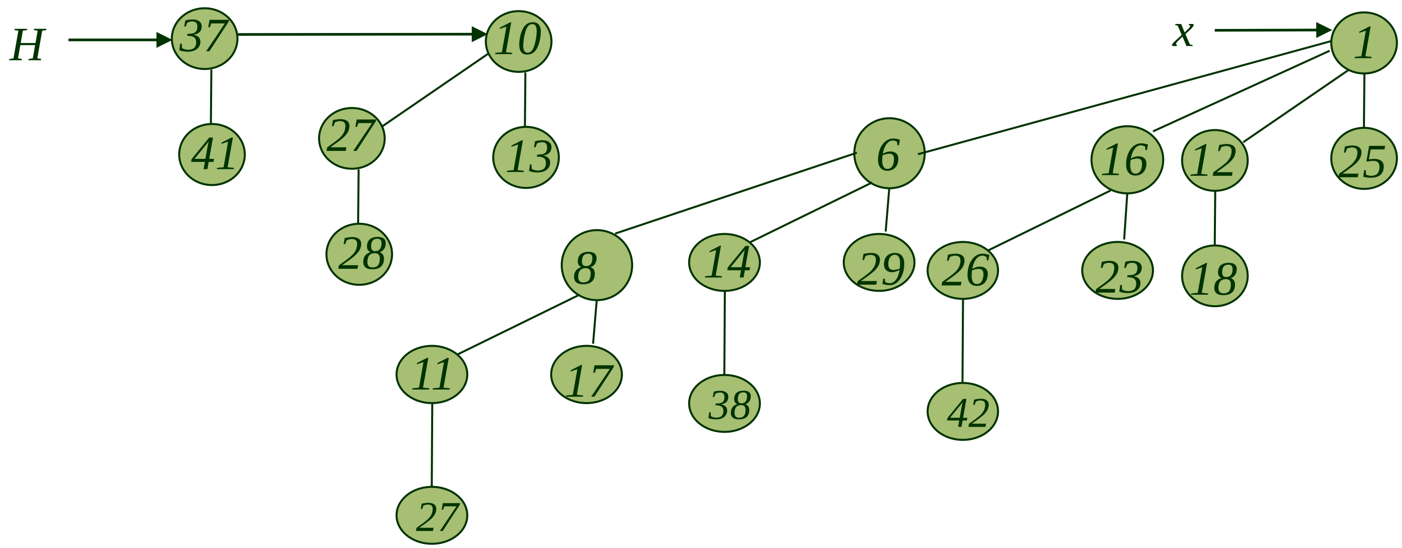
Extract the node with minimum key from following Binomial Heap.



Binomial Heap (Operations_4)

Example:

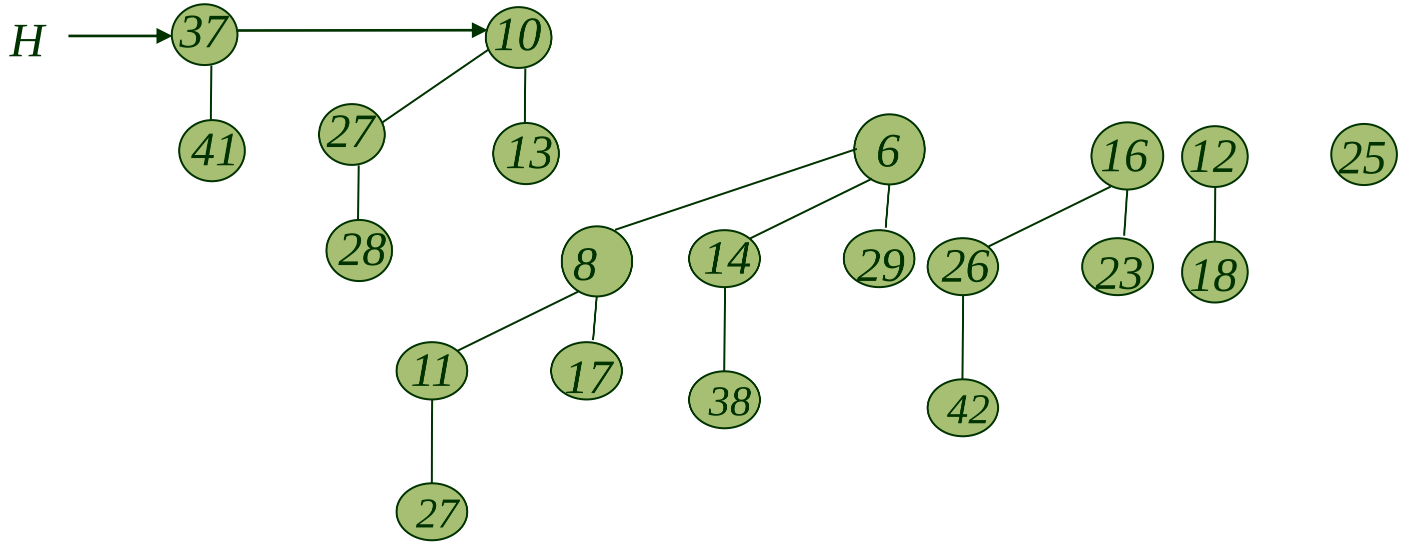
Hear minimum is 1(i.e. x), so remove it



Binomial Heap (Operations_4)

Example:

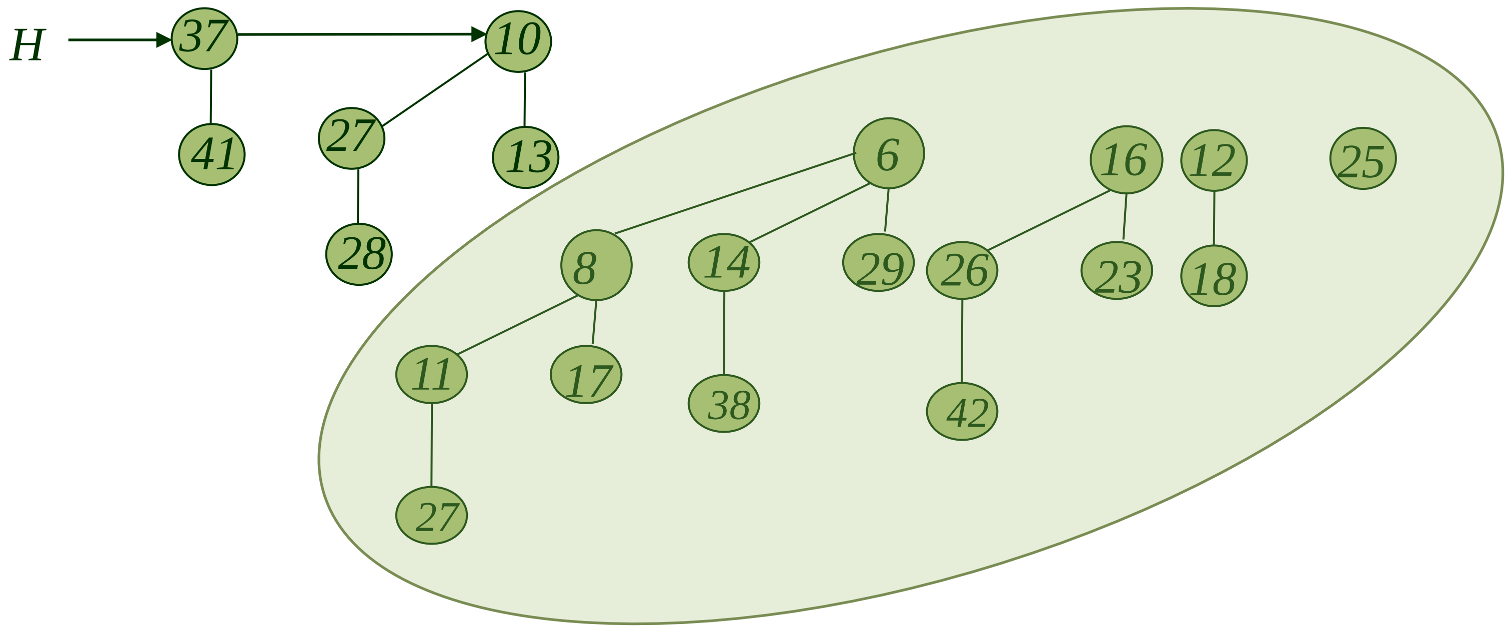
Hear minimum is 1(i.e. x), so remove it



Binomial Heap (Operations_4)

Example:

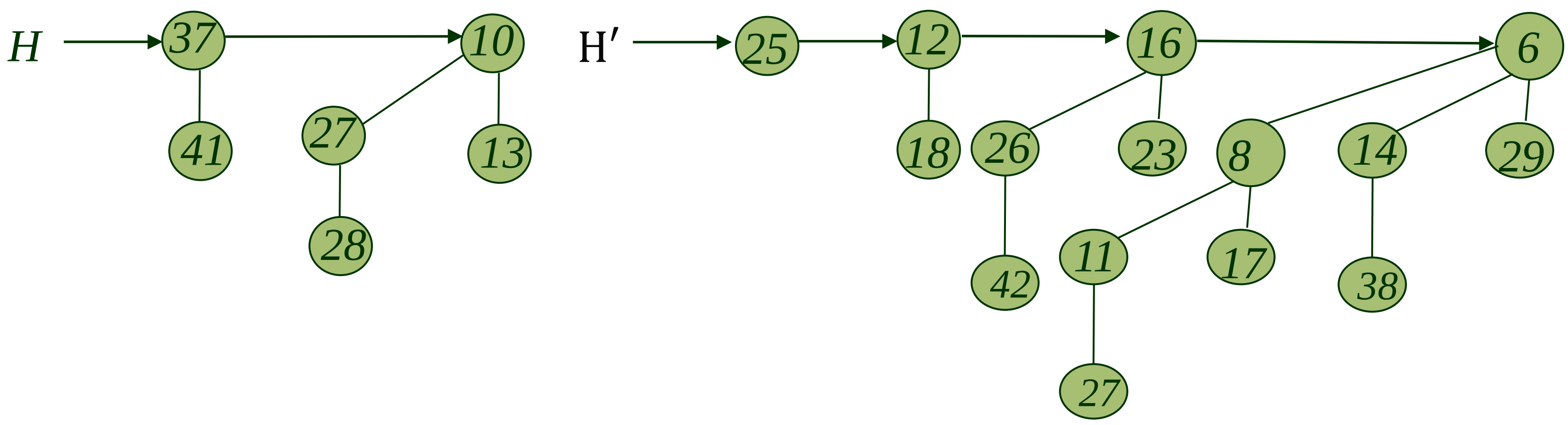
Hear minimum is 1(i.e. x), so remove it



Binomial Heap (Operations_4)

Example:

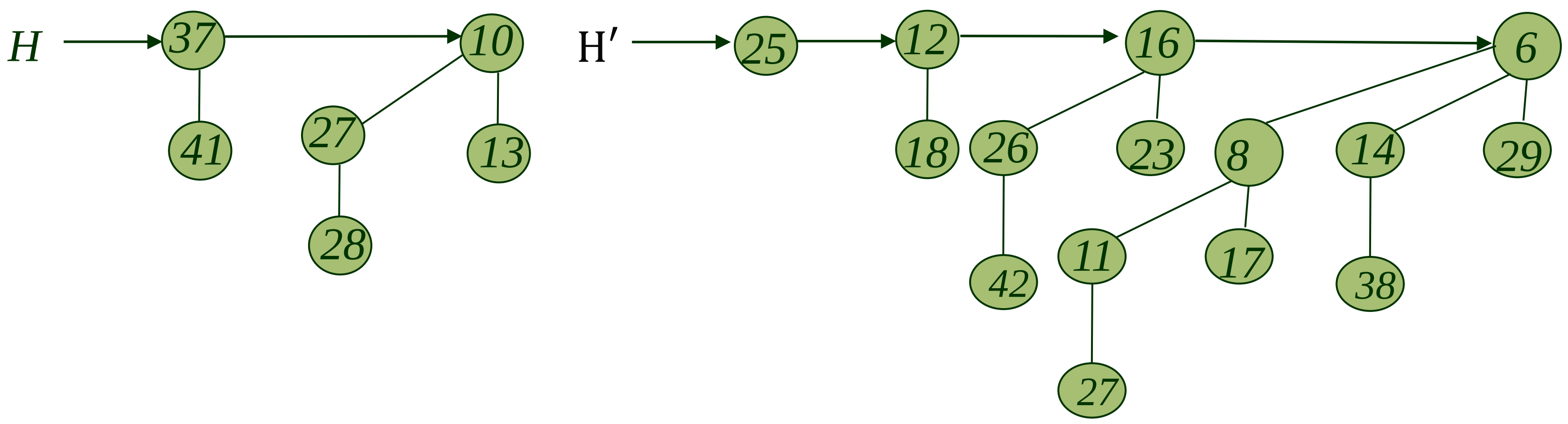
After remove x reverse the order of the list and put it in H′



Binomial Heap (Operations_4)

Example:

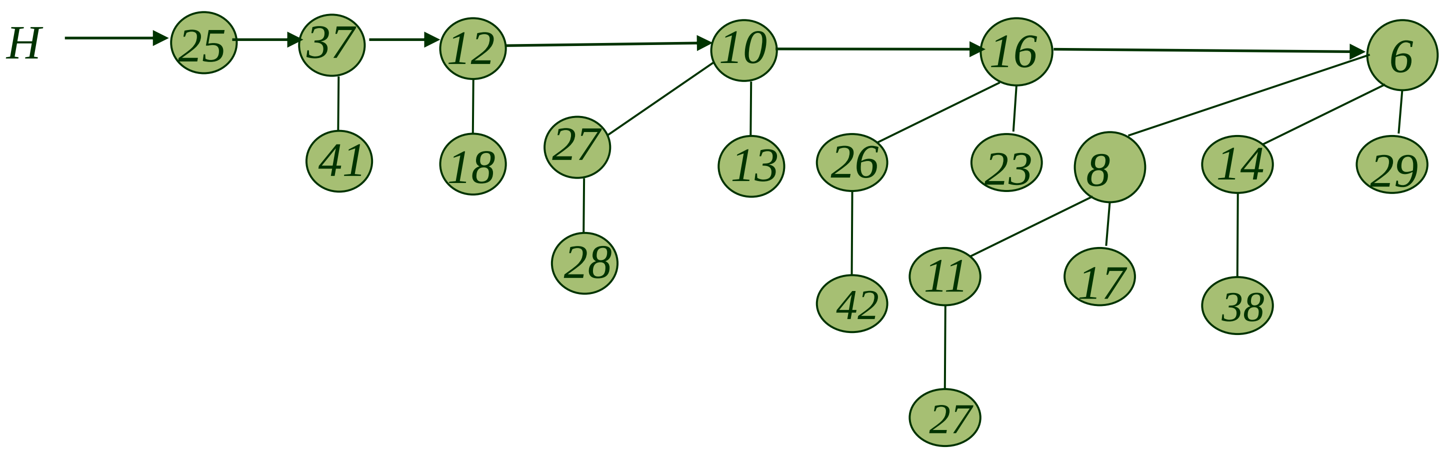
Apply BINOMIAL-HEAP-UNION(H, H′) on the following two Binomial Heap



Binomial Heap (Operations_4)

Example:

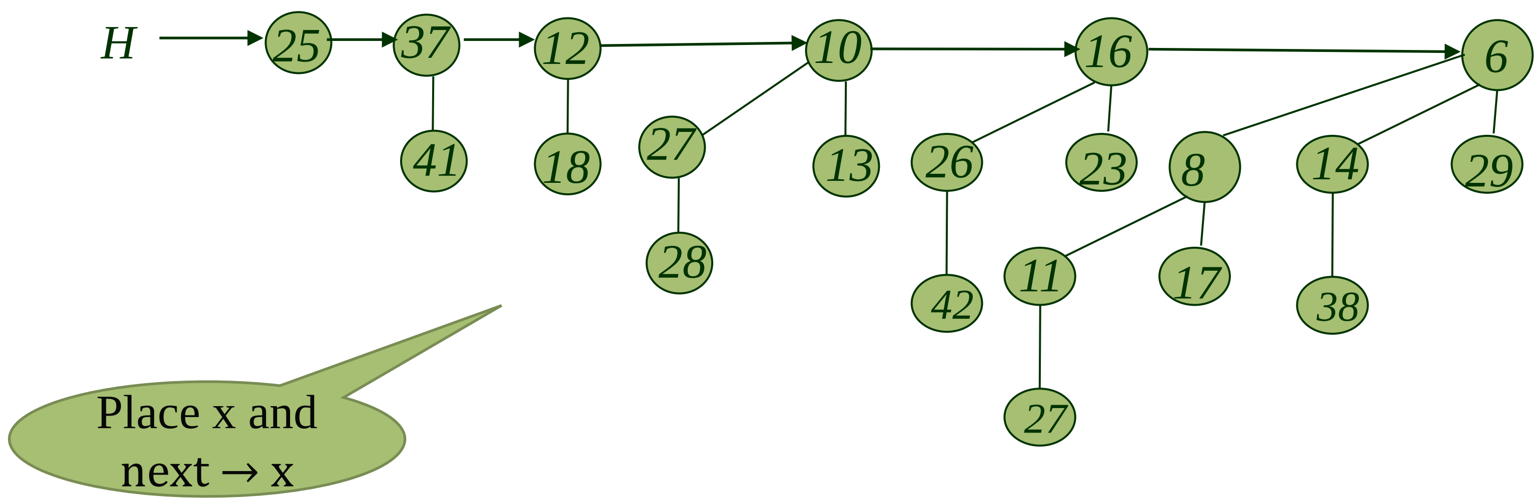
After merging of two binomial heap H and H′



Binomial Heap (Operations_4)

Example:

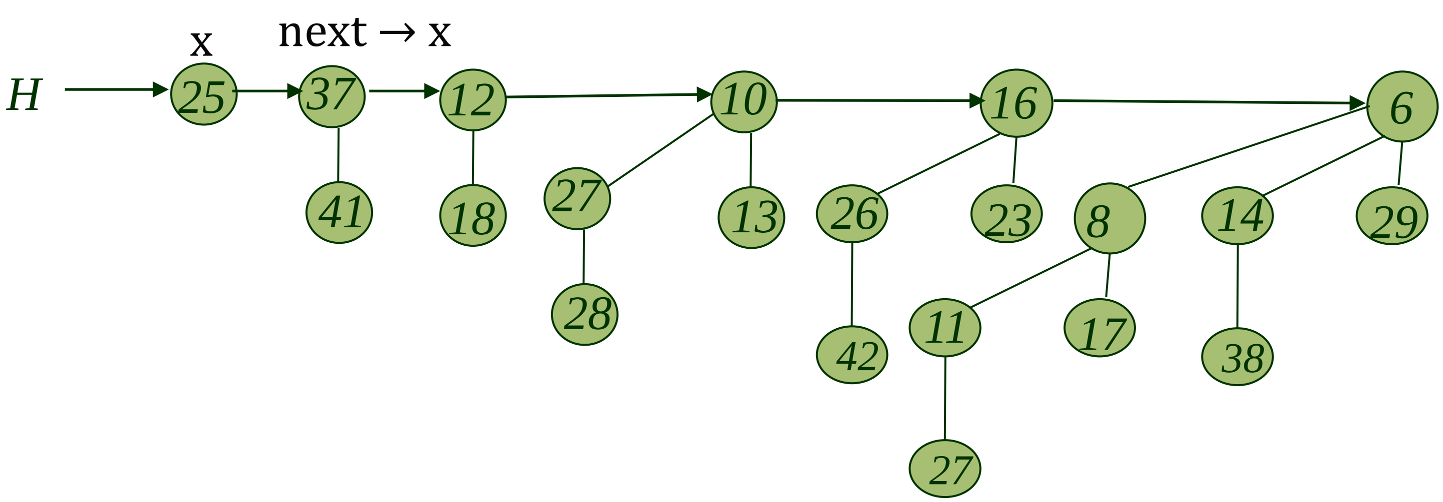
After merging of two binomial heap H and H′



Binomial Heap (Operations_4)

Example:

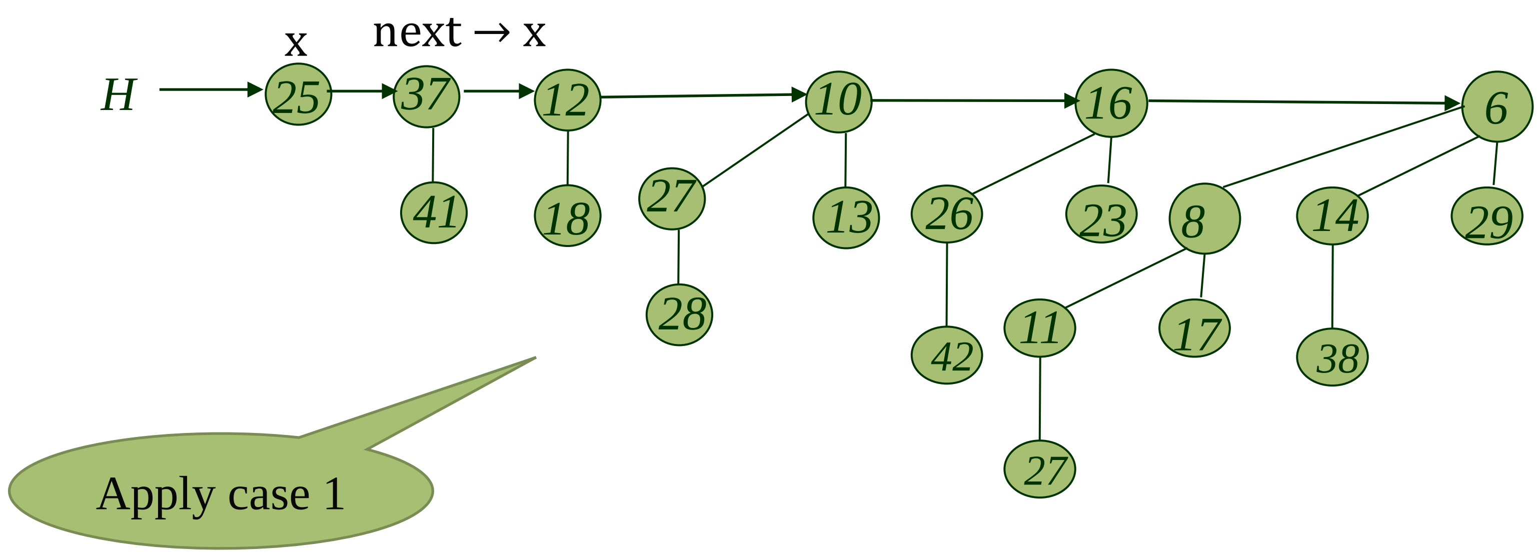
After placing x and next → x



Binomial Heap (Operations_4)

Example:

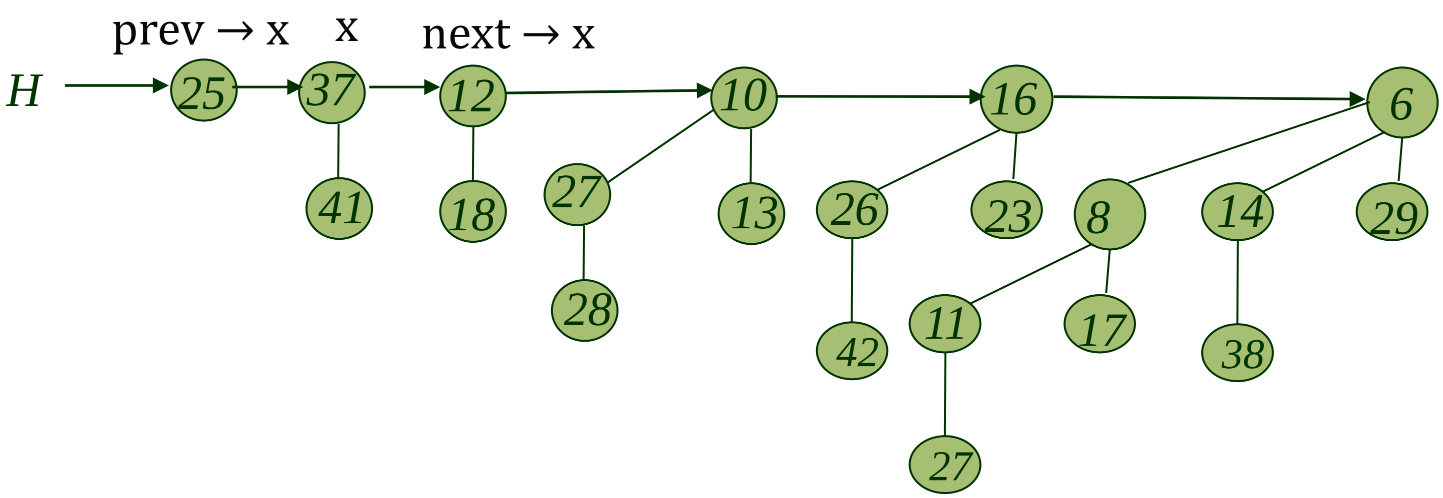
After placing x and next → x



Binomial Heap (Operations_4)

Example:

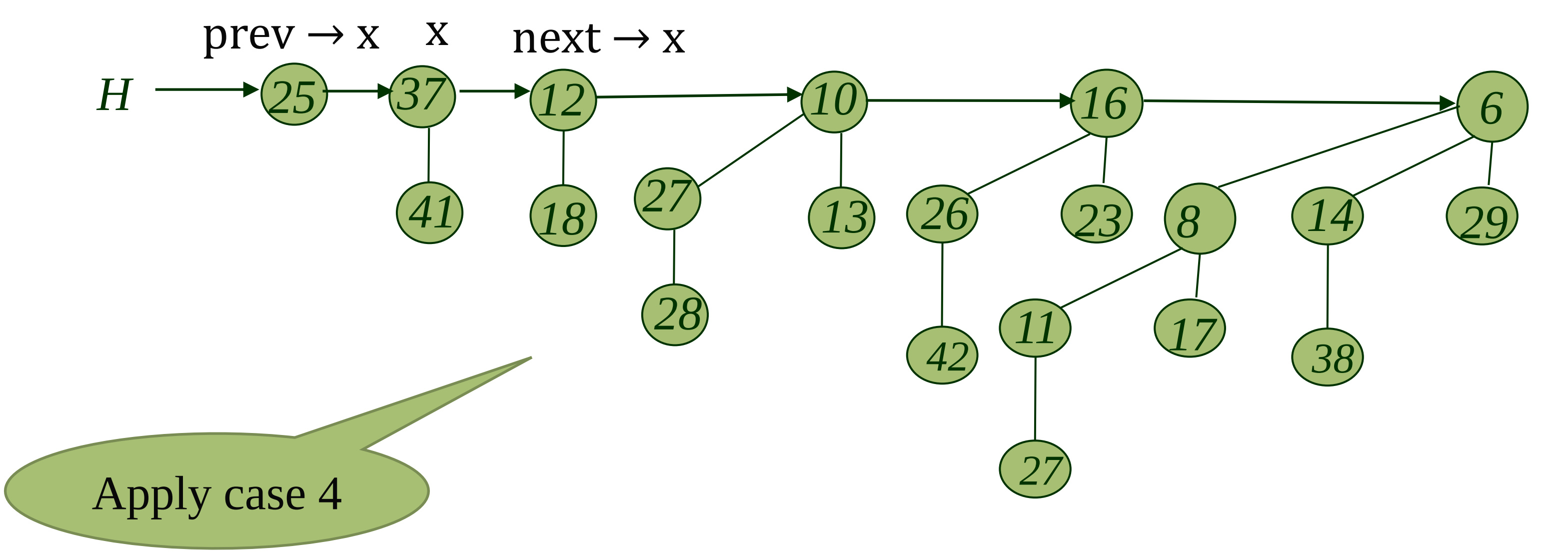
After applying case 1



Binomial Heap (Operations_4)

Example:

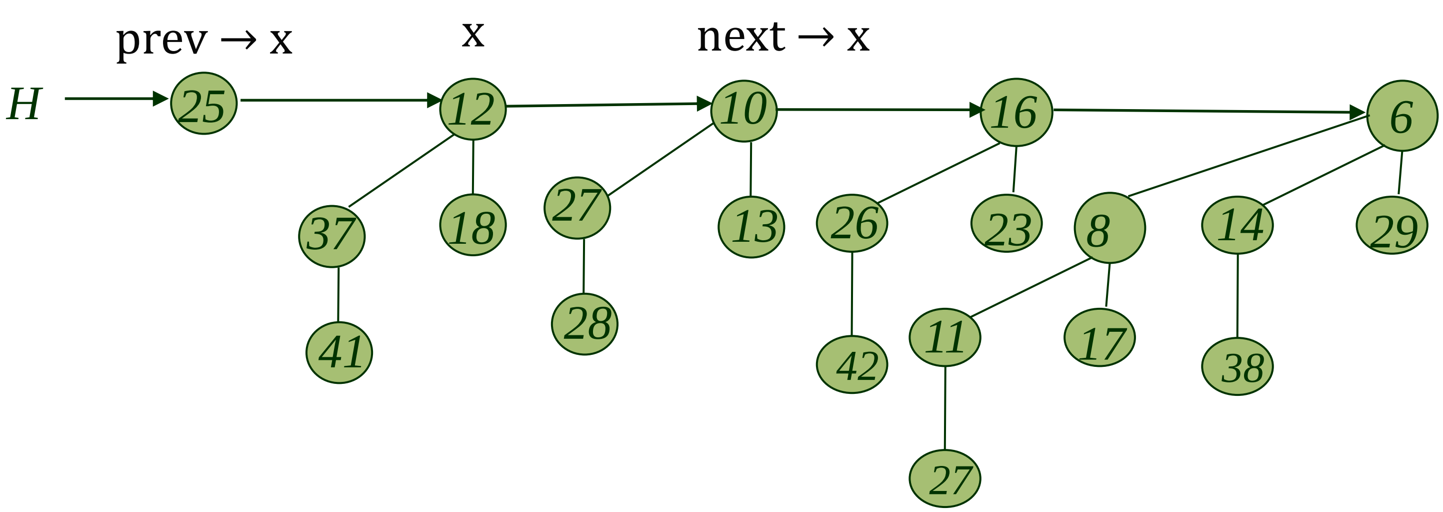
After applying case 1



Binomial Heap (Operations_4)

Example:

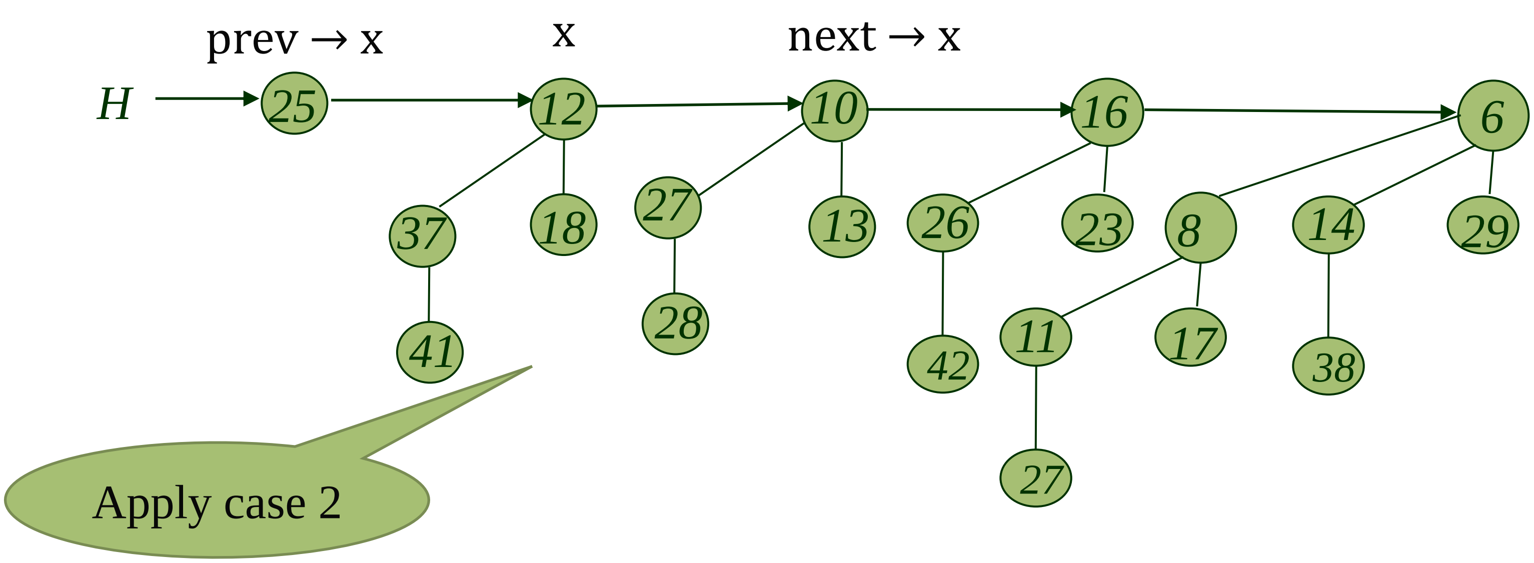
After applying case 4



Binomial Heap (Operations_4)

Example:

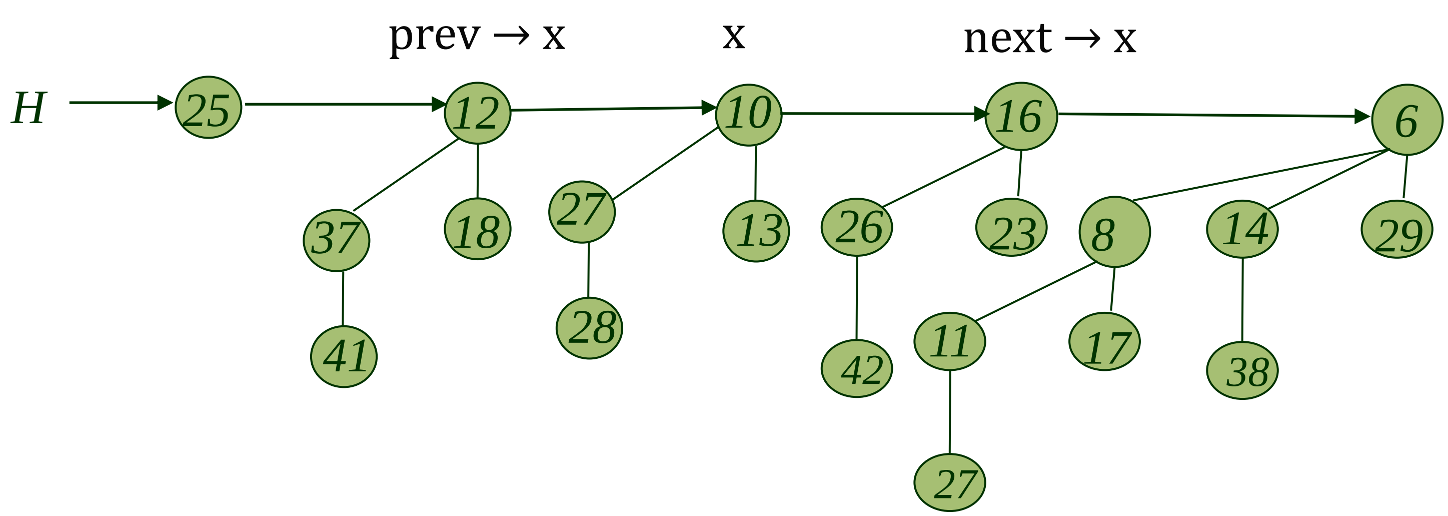
After applying case 4



Binomial Heap (Operations_4)

Example:

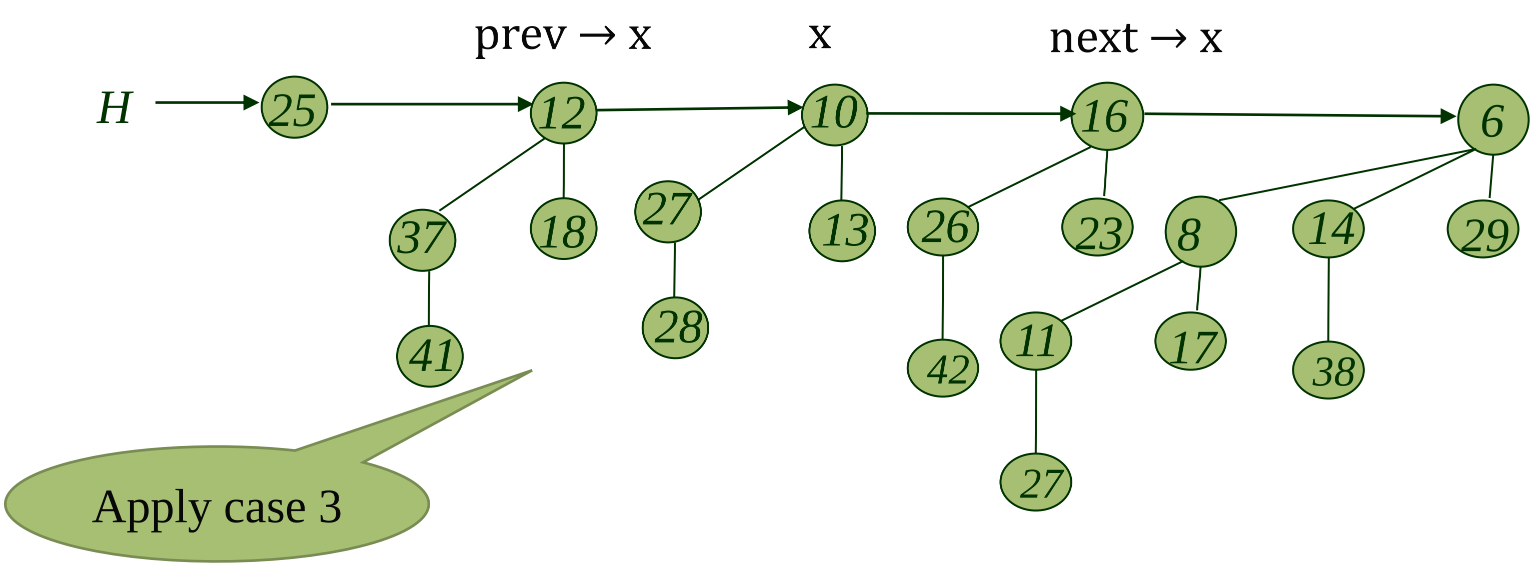
After applying case 2



Binomial Heap (Operations_4)

Example:

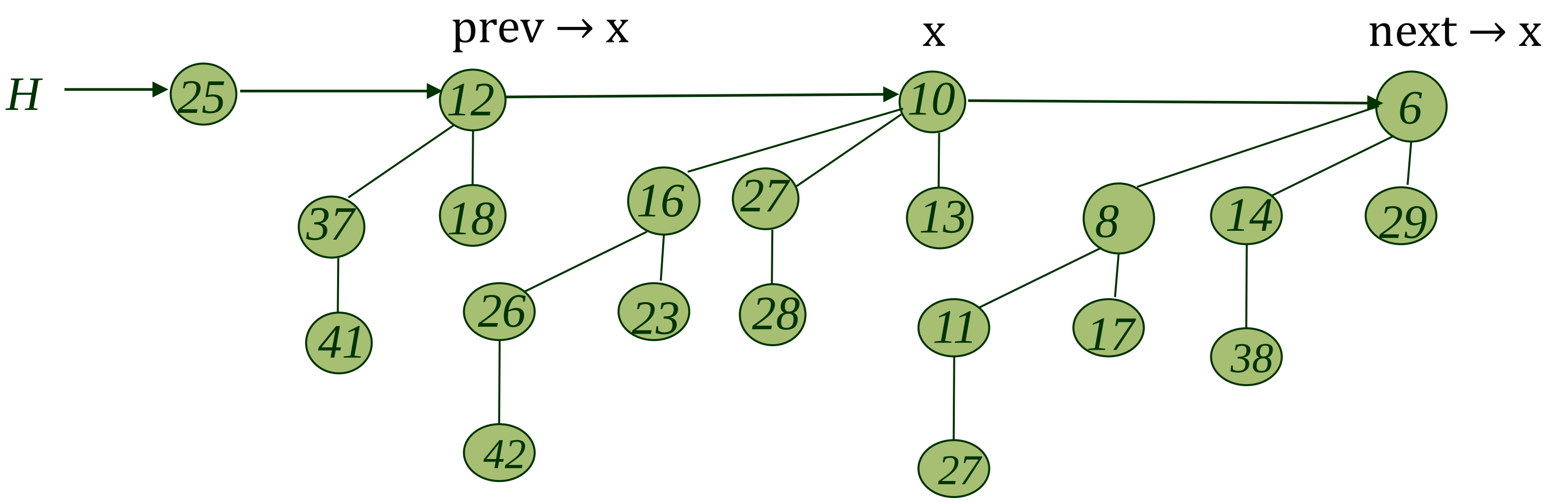
After applying case 2



Binomial Heap (Operations_4)

Example:

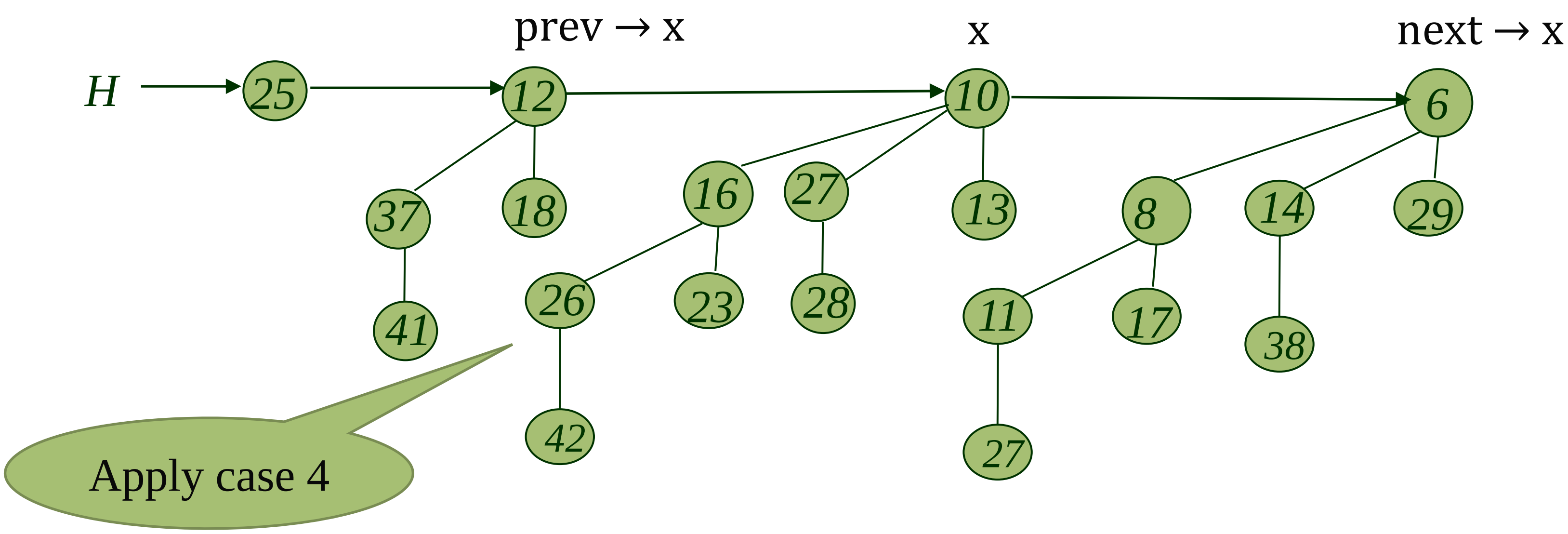
After applying case 3



Binomial Heap (Operations_4)

Example:

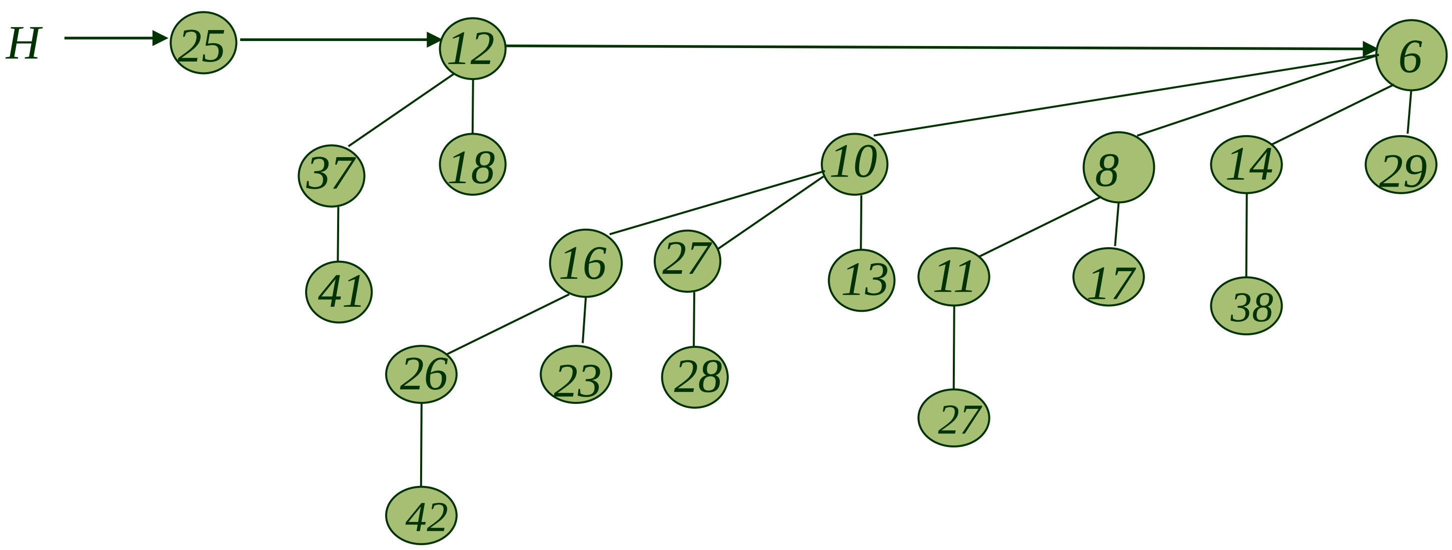
After applying case 3



Binomial Heap (Operations_4)

Example:

After applying case 4



Binomial Heap (Operations_5)

5. INSERT (H, x)

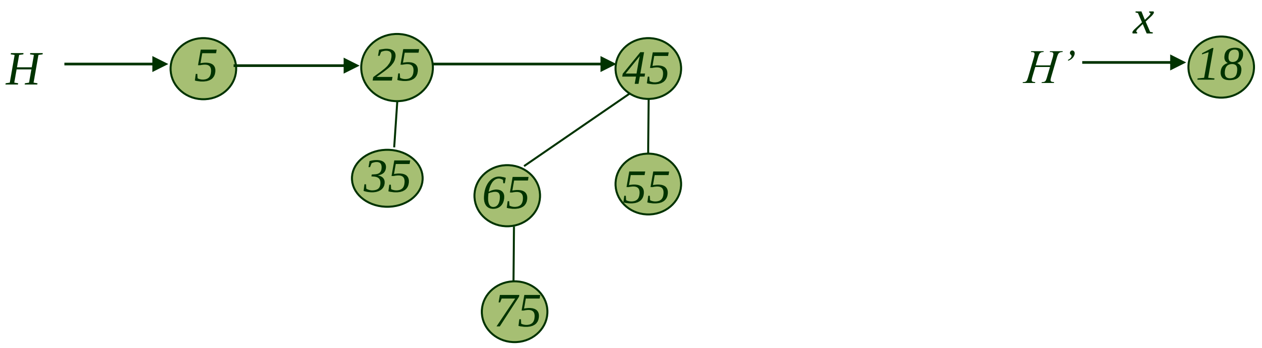
The BINOMIAL-HEAP-INSERT procedure inserts node x into binomial heap H , assuming that x has already been allocated and key[x] has already been filled in.

```
BINOMIAL-HEAP-INSERT(H, x)
    H′ ← call MAKE-BINOMIAL-HEAP()
    p[x] ← NIL
    child[x] ← NIL
    sibling[x] ← NIL
    degree[x] ← 0
    head[H′] ← x
    H ← call BINOMIAL-HEAP-UNION(H, H′)
```

Binomial Heap (Operations_5)

Example:

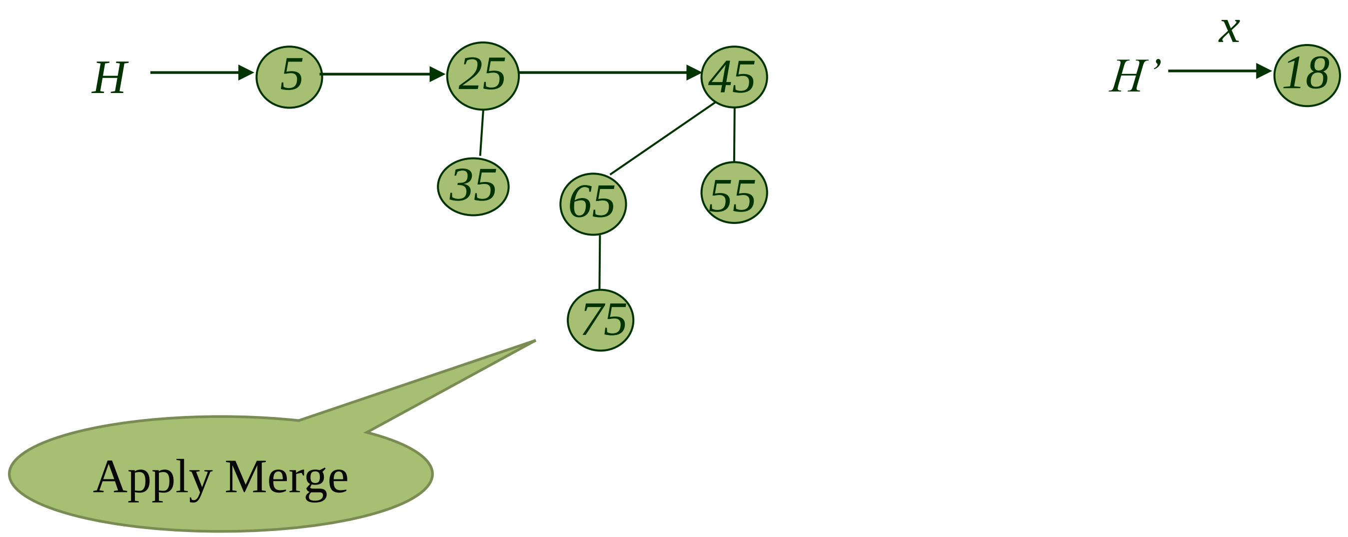
inserts node x into binomial heap H



Binomial Heap (Operations_5)

Example:

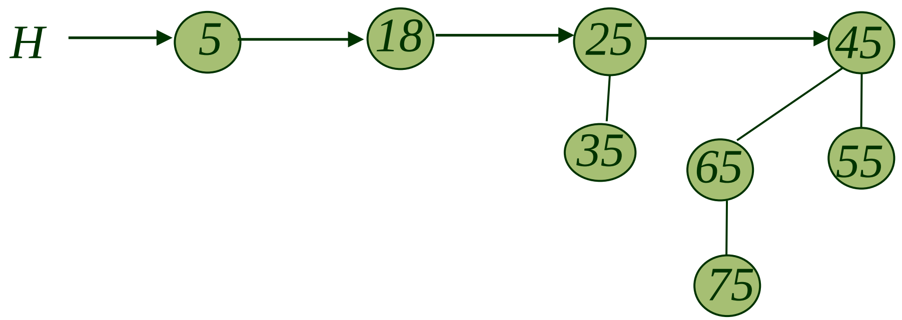
inserts node x into binomial heap H



Binomial Heap (Operations_5)

Example:

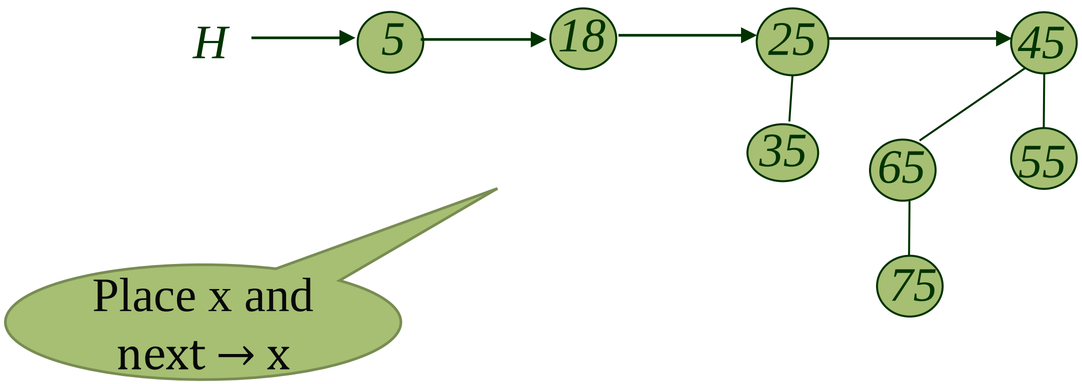
After Merging



Binomial Heap (Operations_5)

Example:

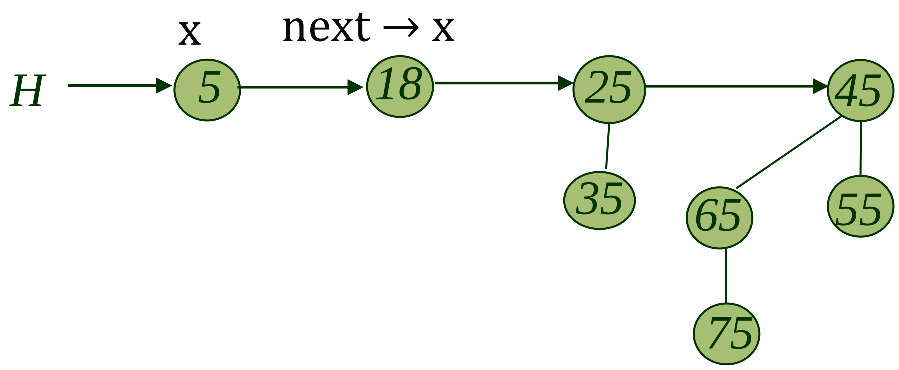
After Merging



Binomial Heap (Operations_5)

Example:

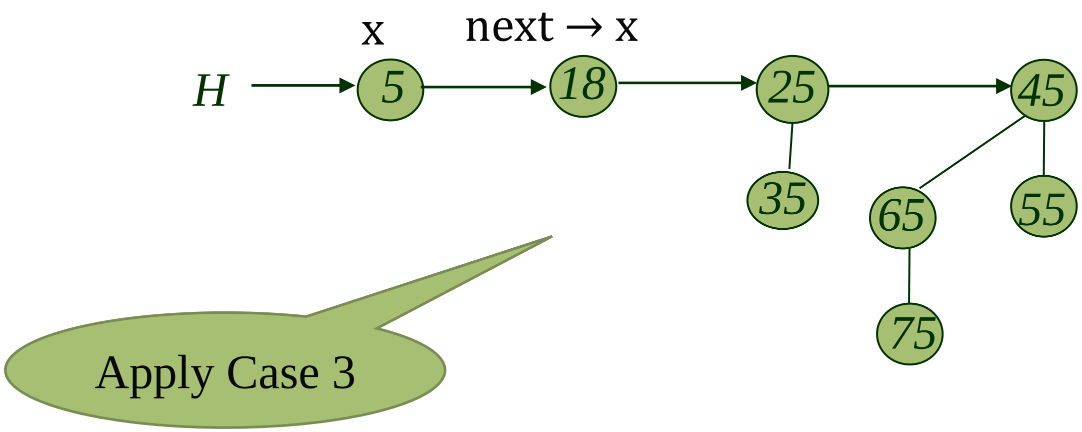
After placing x and next → x



Binomial Heap (Operations_5)

Example:

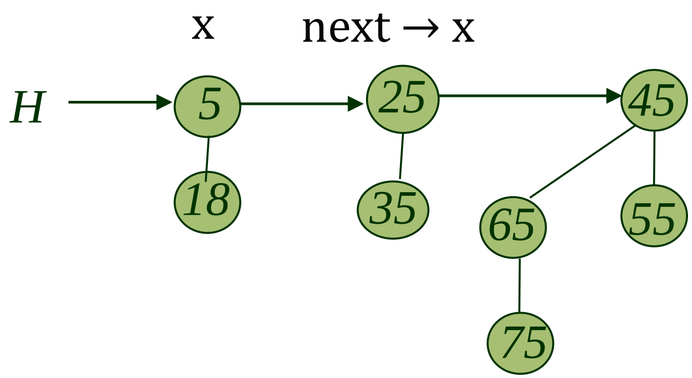
After placing x and next → x



Binomial Heap (Operations_5)

Example:

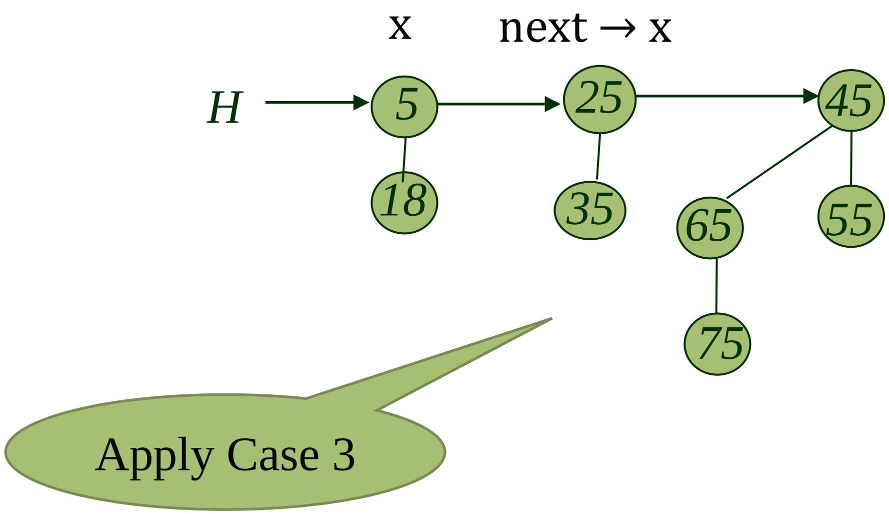
After applying case 3



Binomial Heap (Operations_5)

Example:

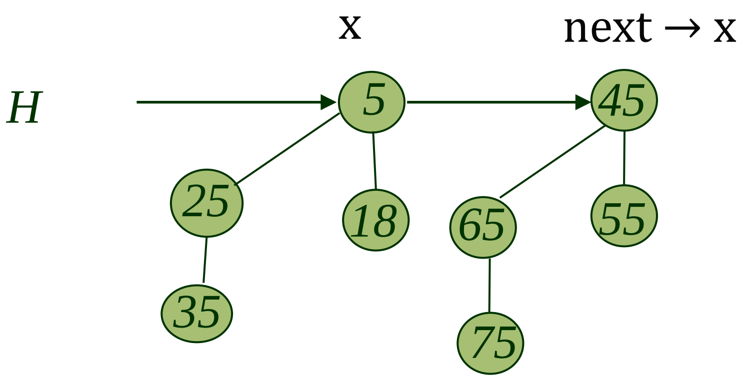
After applying case 3



Binomial Heap (Operations_5)

Example:

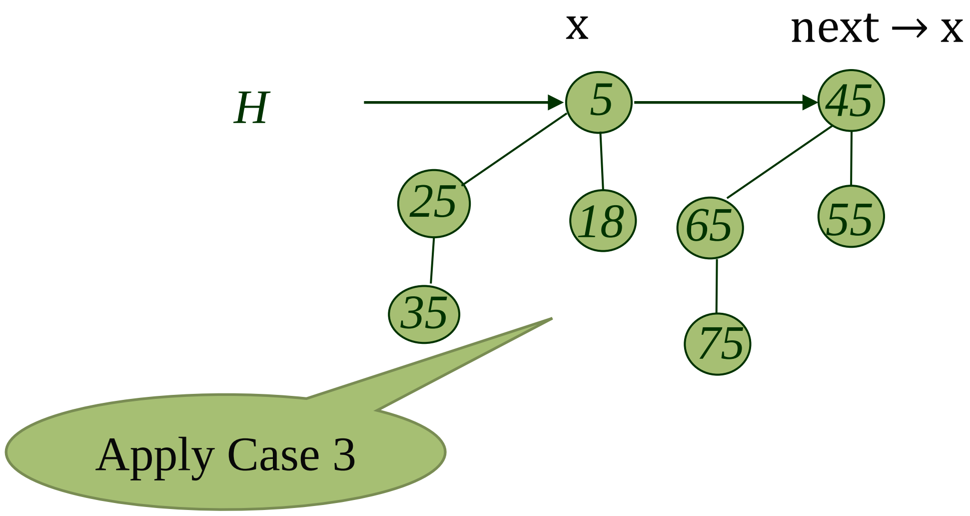
After applying case 3



Binomial Heap (Operations_5)

Example:

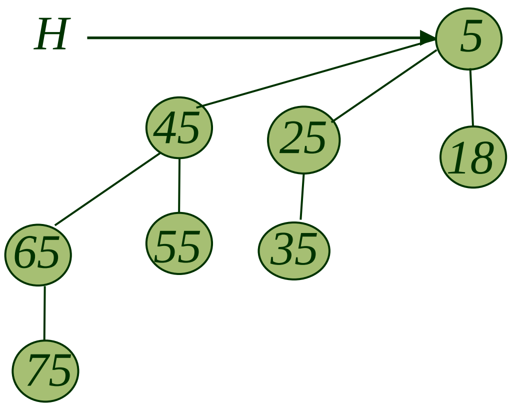
After applying case 3



Binomial Heap (Operations_5)

Example:

After applying case 3



Binomial Heap (Operations_6)

6. DECREASE KEY (H, x, k)

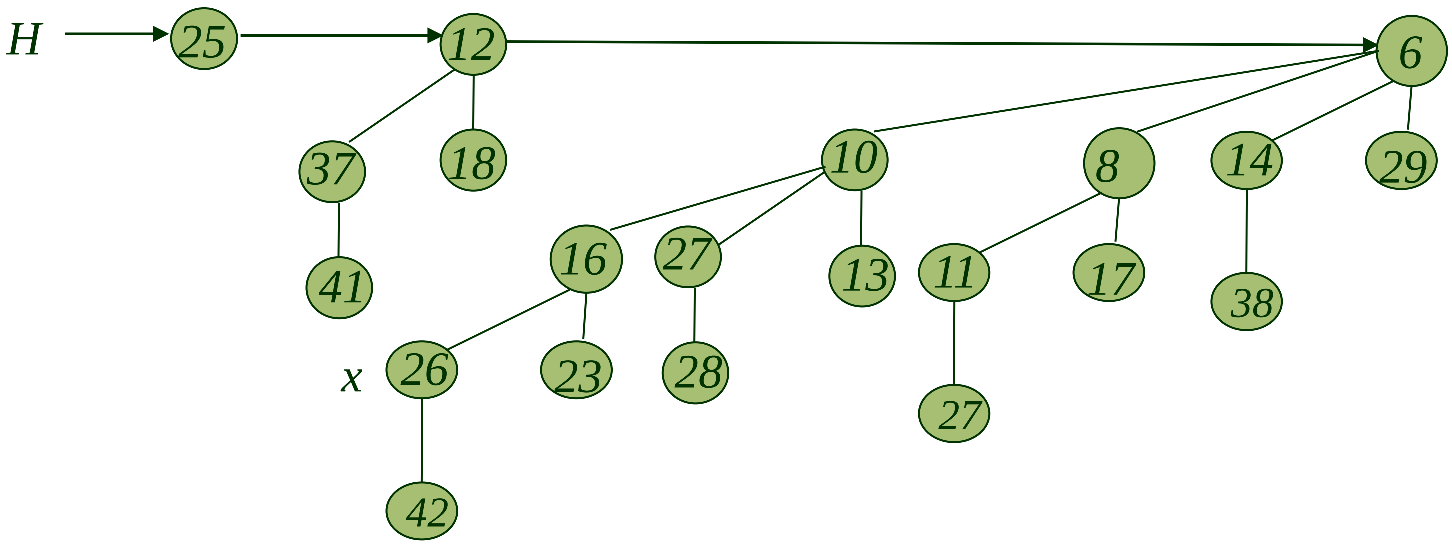
The DECREASE KEY procedure decreases the key of a node x in a binomial heap H to a new value k. It signals an error if k is greater than x's current key.

```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
if k > key[x]
        then error "new key is greater than current key"
key[x] ← k
y ← x
z ← p[y]
while z ≠ NIL and key[y] < key[z]
        do exchange key[y] ↔ key[z]
                ▸ If y and z have satellite fields, exchange them, too.
                y ← z
                z ← p[y]
```

Binomial Heap (Operations_6)

Example:

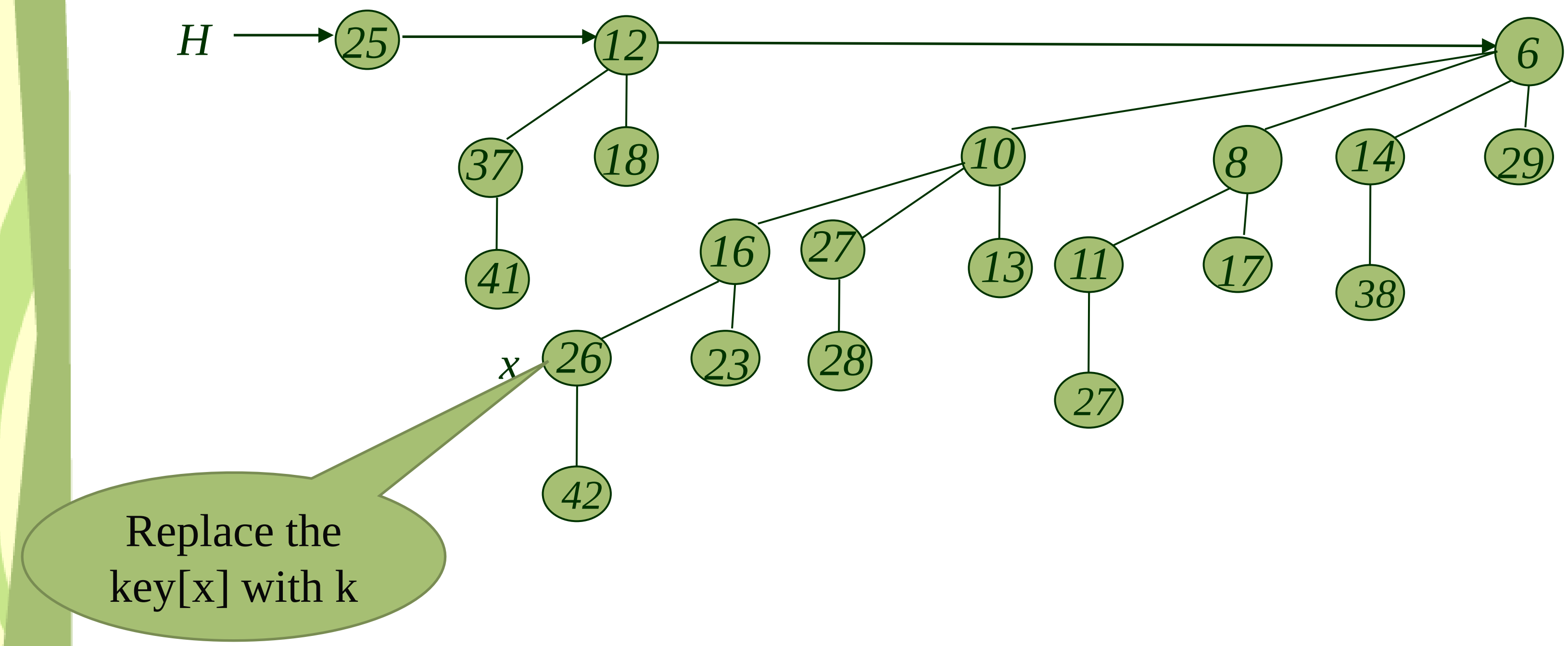
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

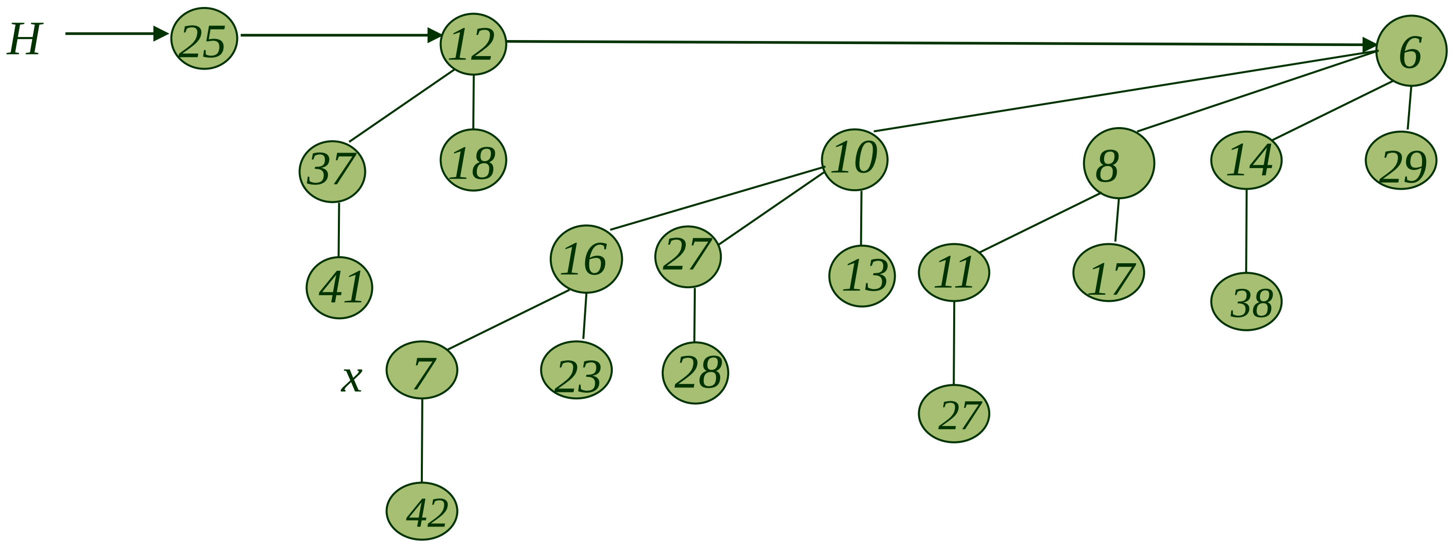
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

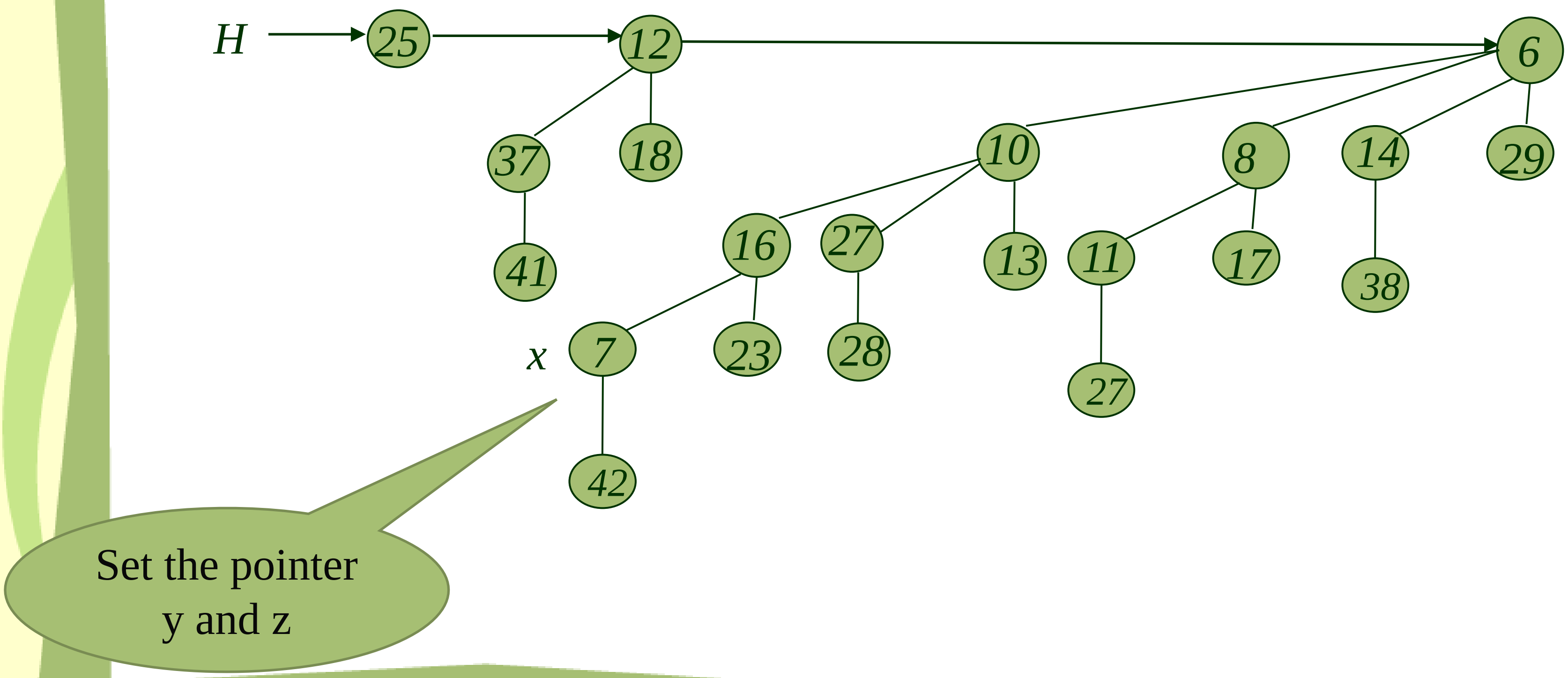
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

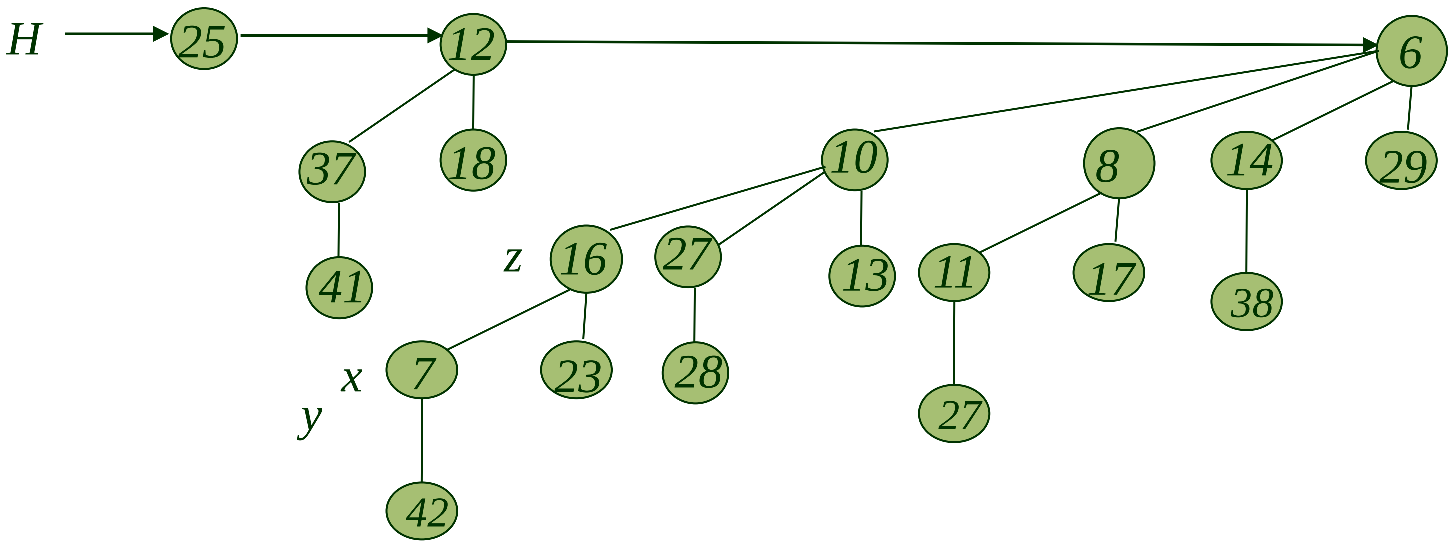
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

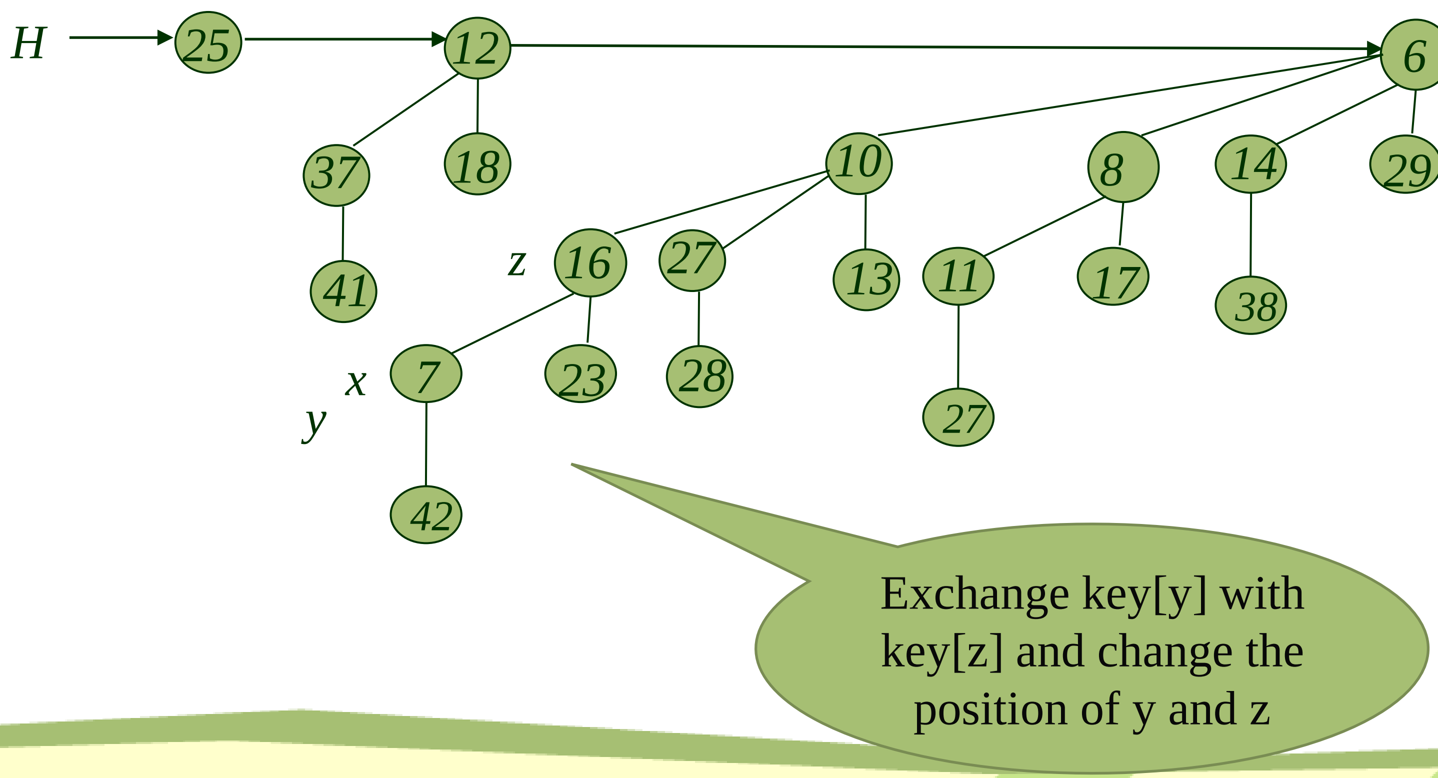
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

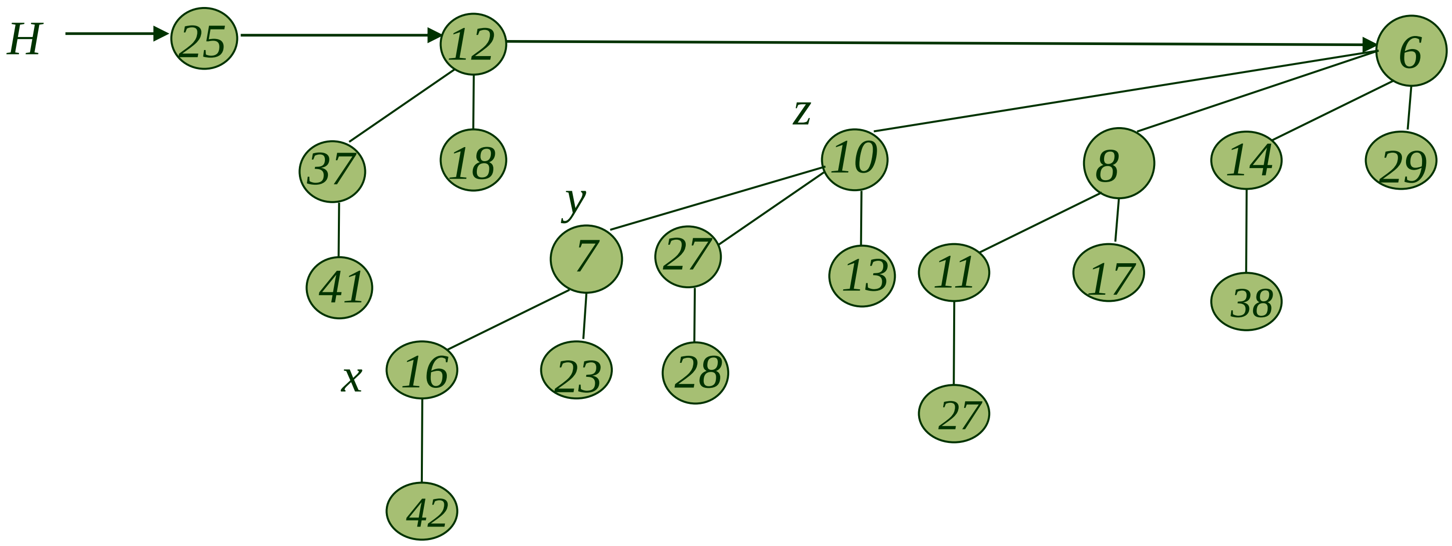
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

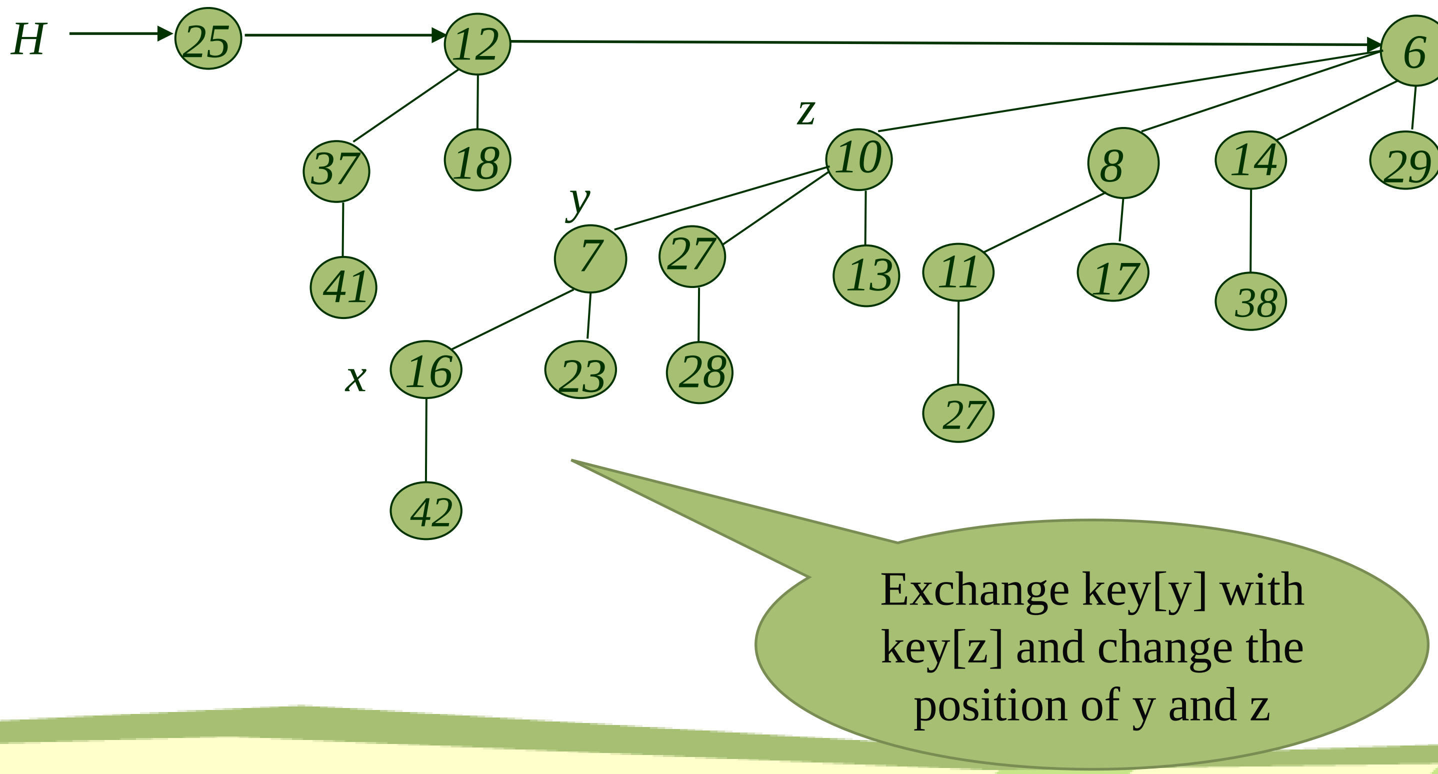
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

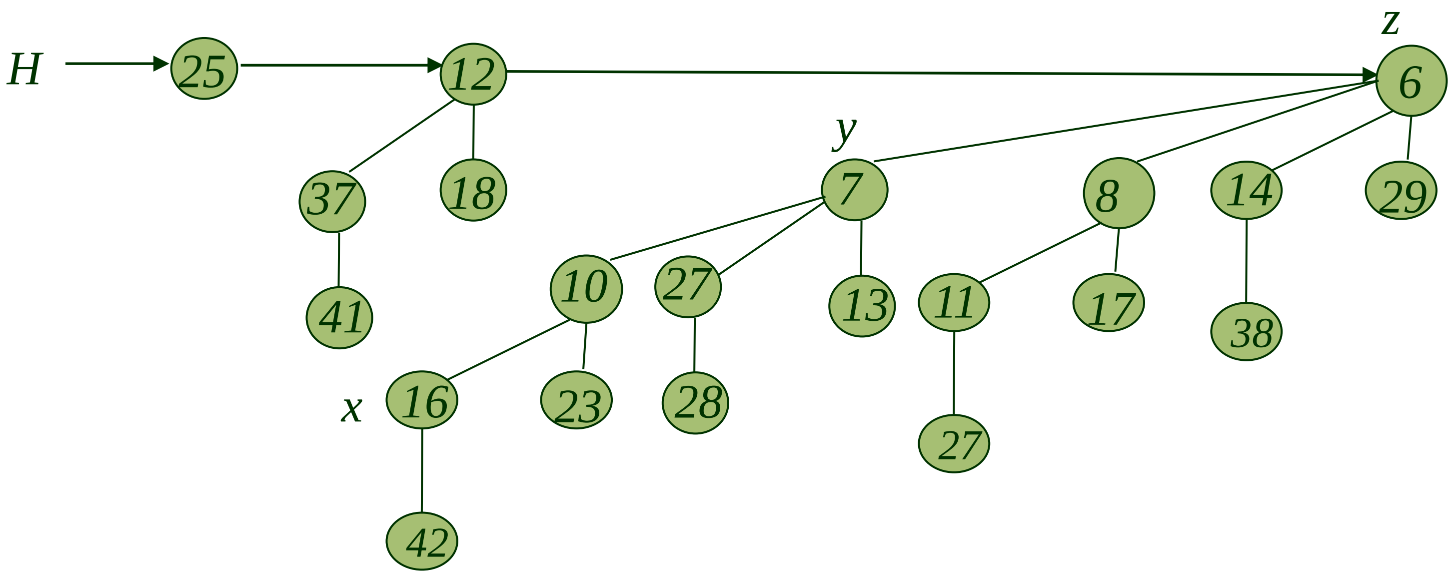
Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_6)

Example:

Decreases the key of a node x (i.e. 26) in a binomial heap H to a new value k (i.e. 7).



Binomial Heap (Operations_7)

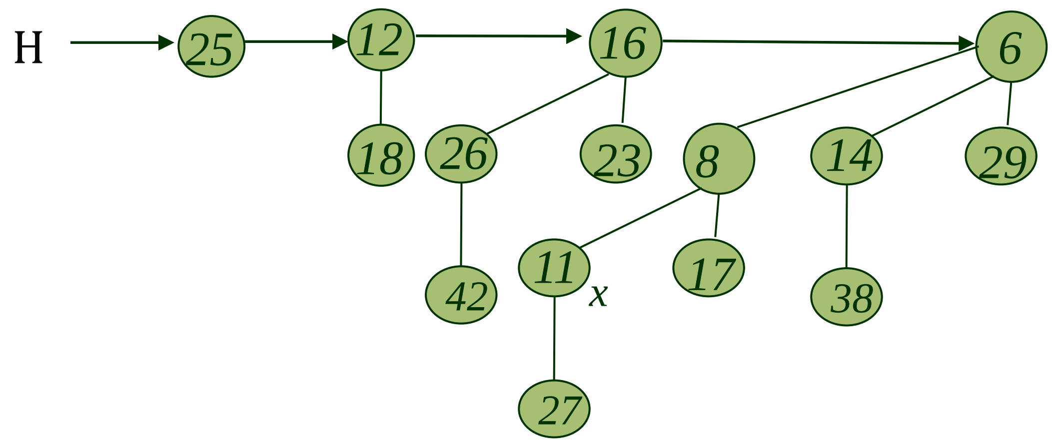
7. DELETE(H, x)

The delete procedure delete the key x with an assumption that no node currently in the binomial heap has a key of -∞.

```
BINOMIAL-HEAP-DELETE(H, x)
    BINOMIAL-HEAP-DECREASE-KEY(H, x, -∞)
    BINOMIAL-HEAP-EXTRACT-MIN(H)
```

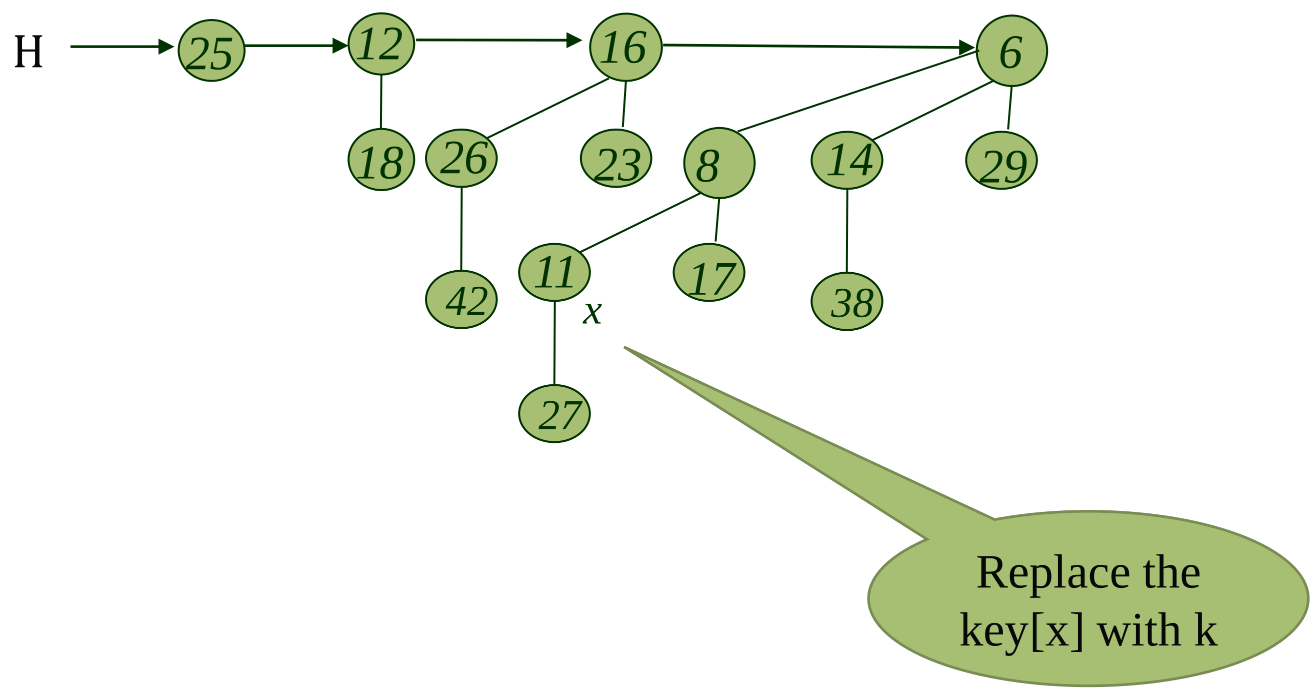
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞.



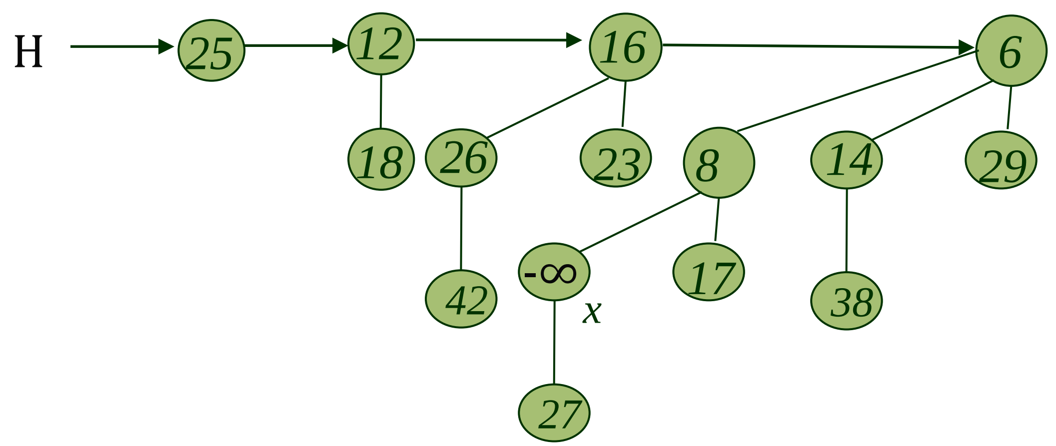
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



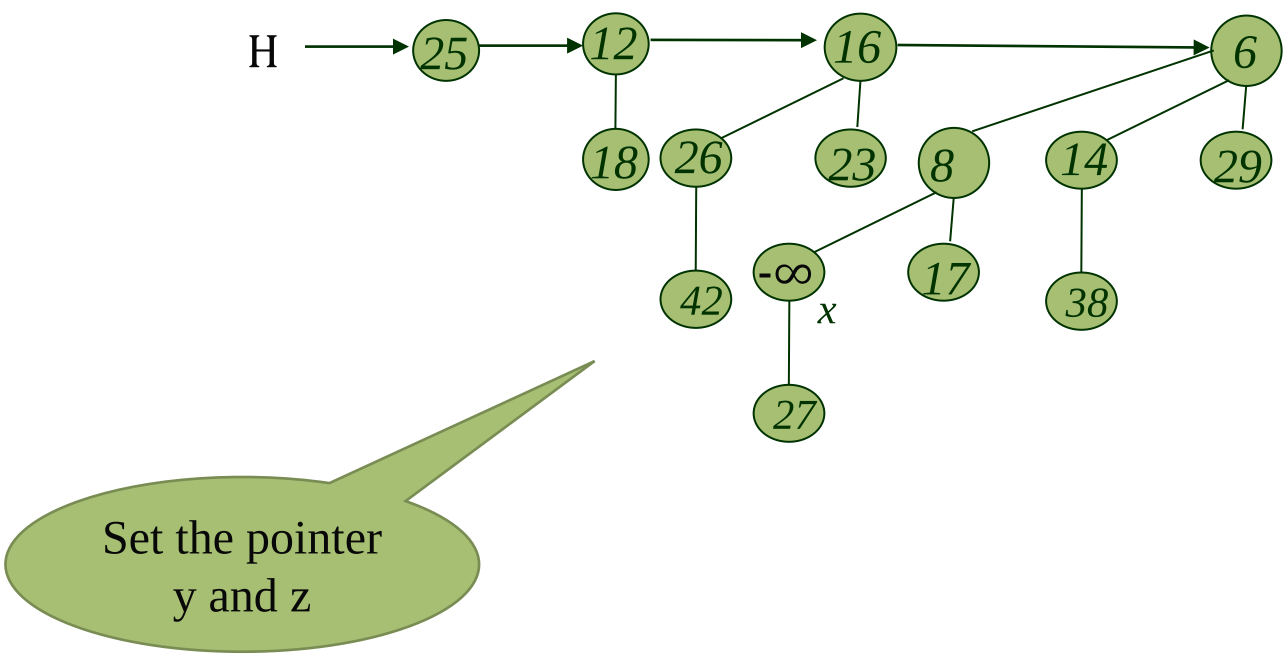
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



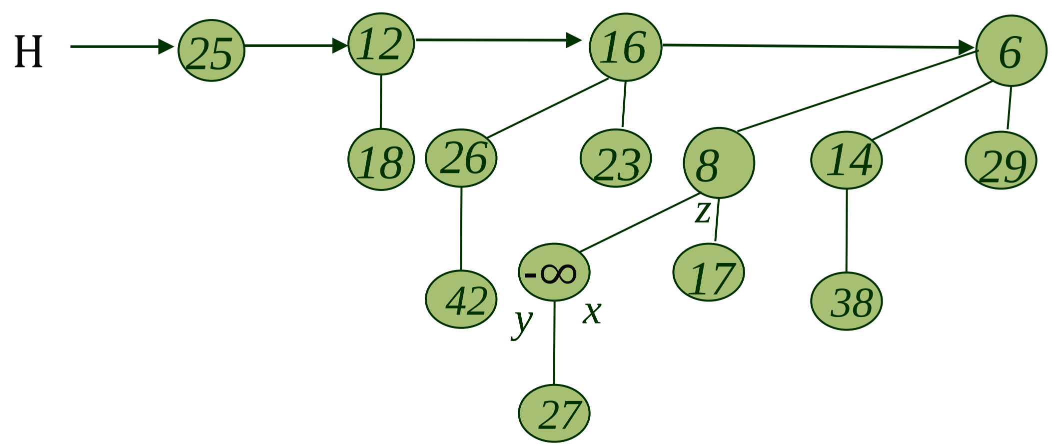
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



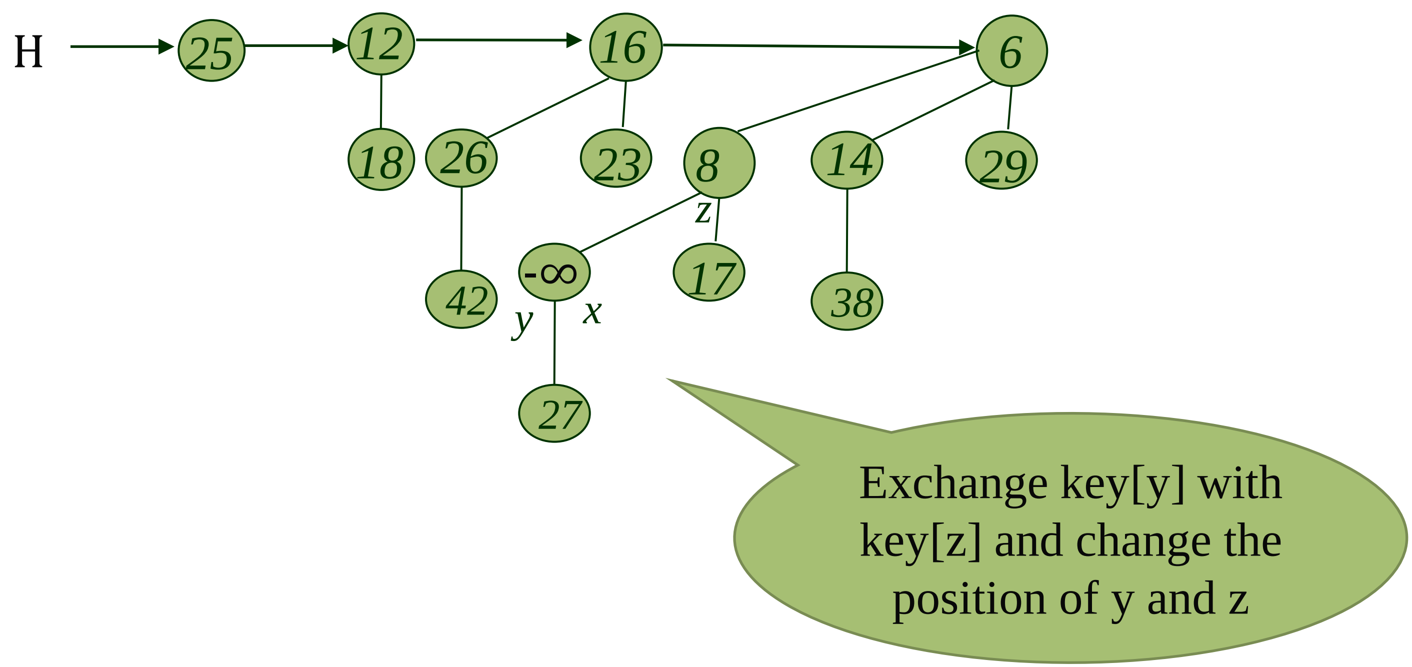
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



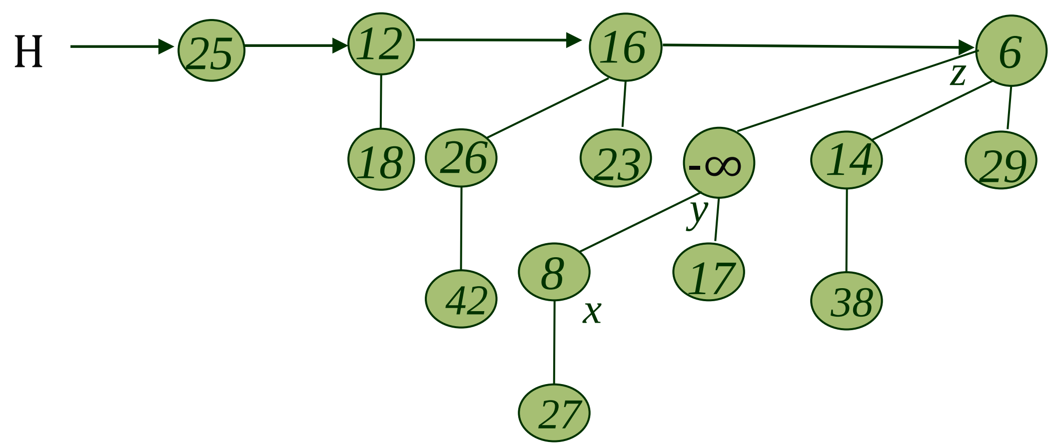
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



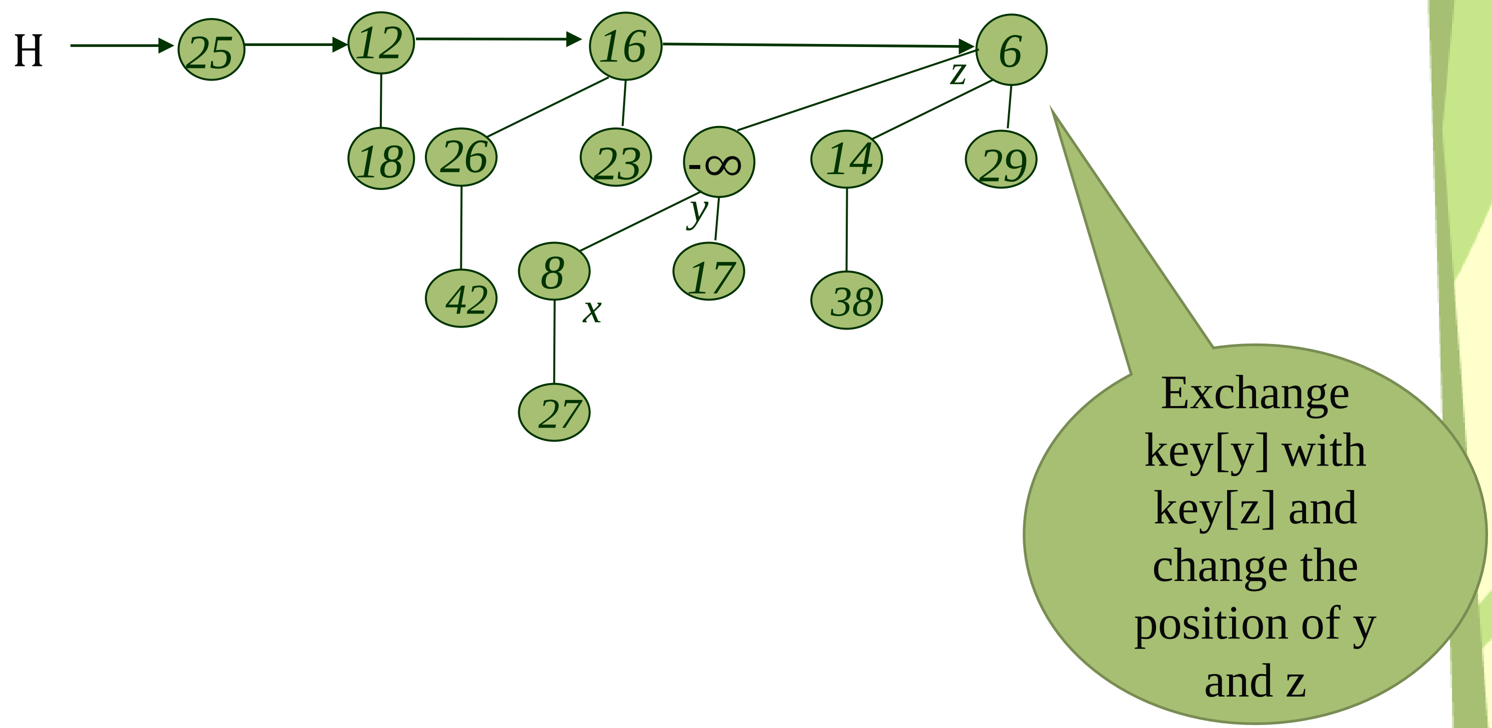
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



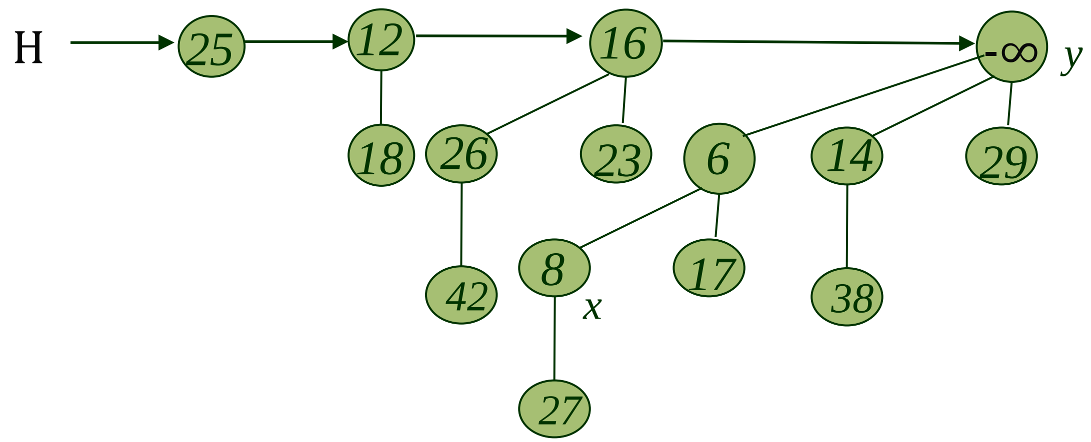
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



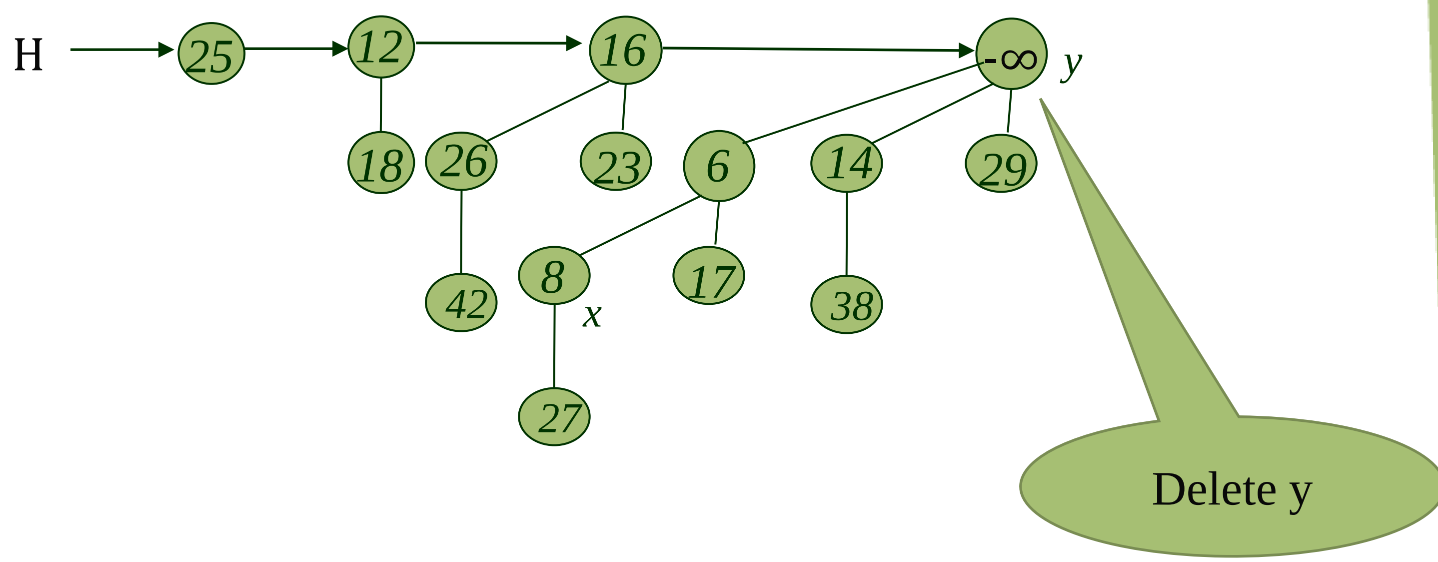
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



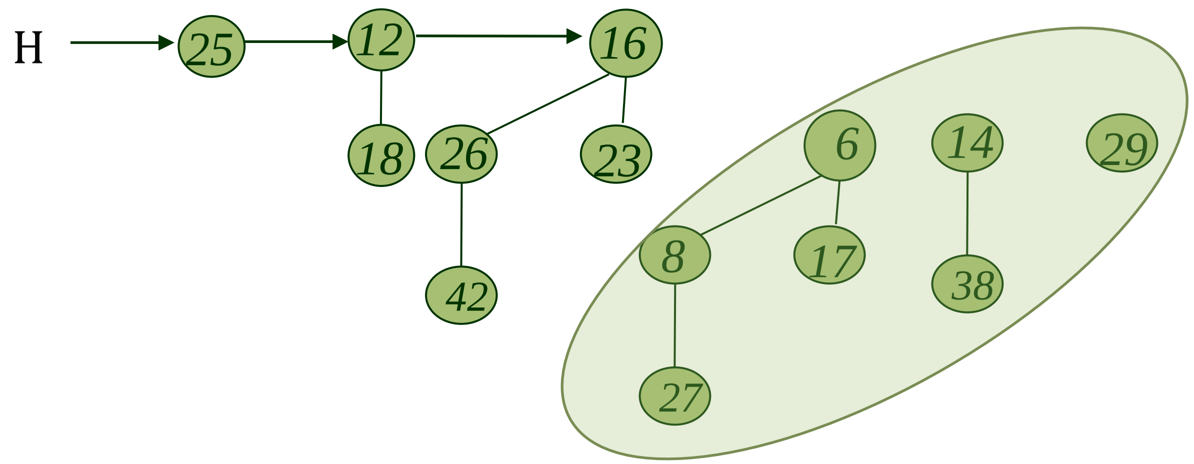
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



Binomial Heap (Operations_7)

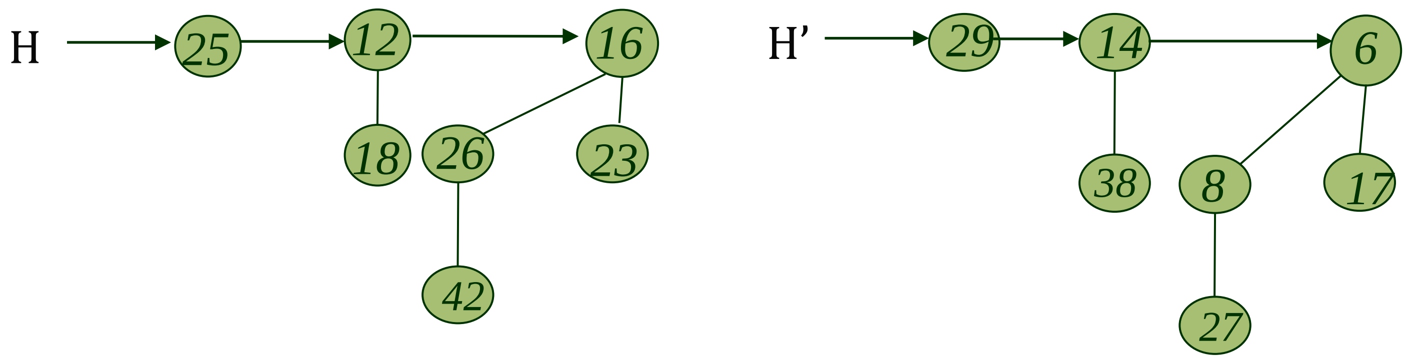
Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



After remove x reverse the order of the list and put it in H′

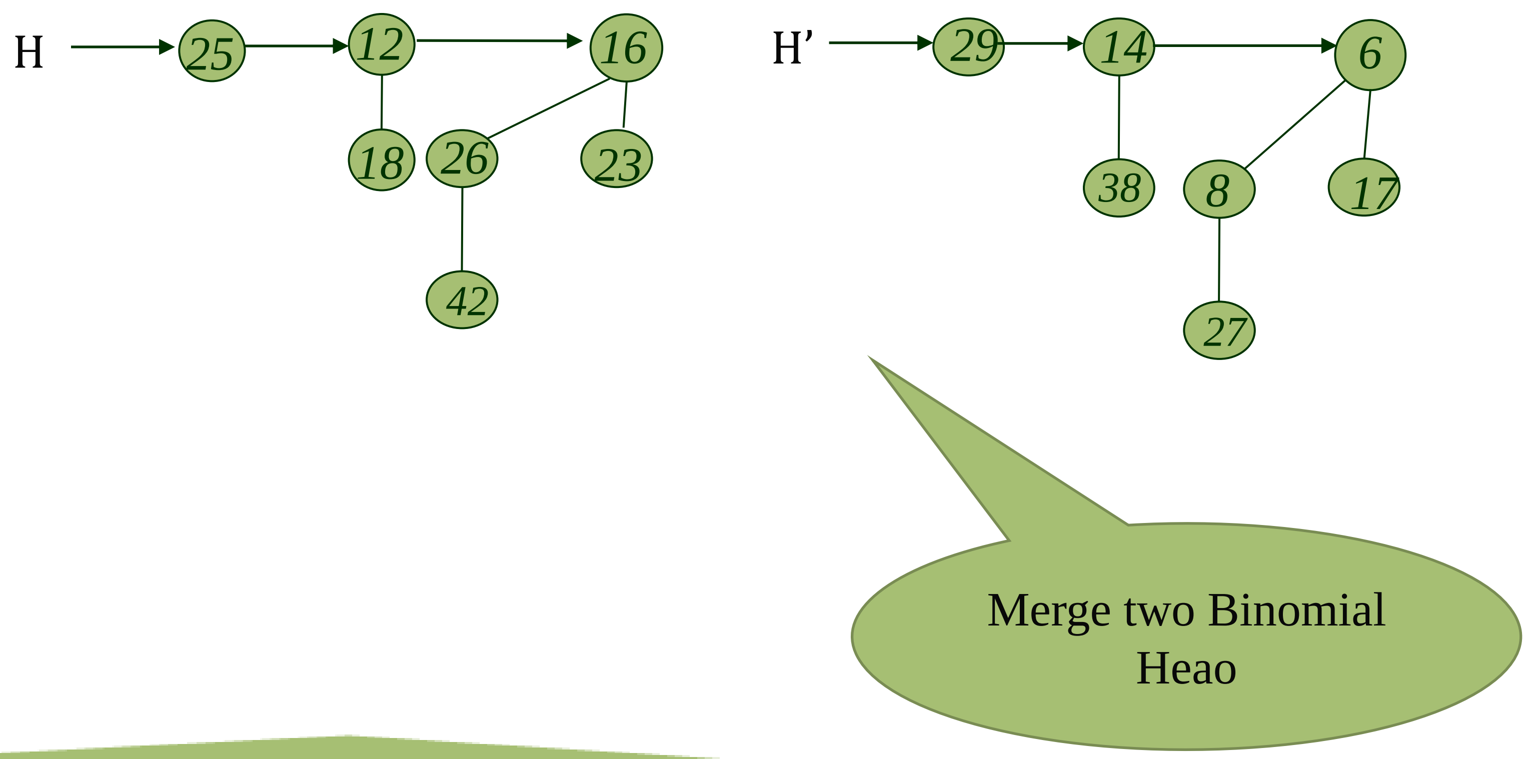
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



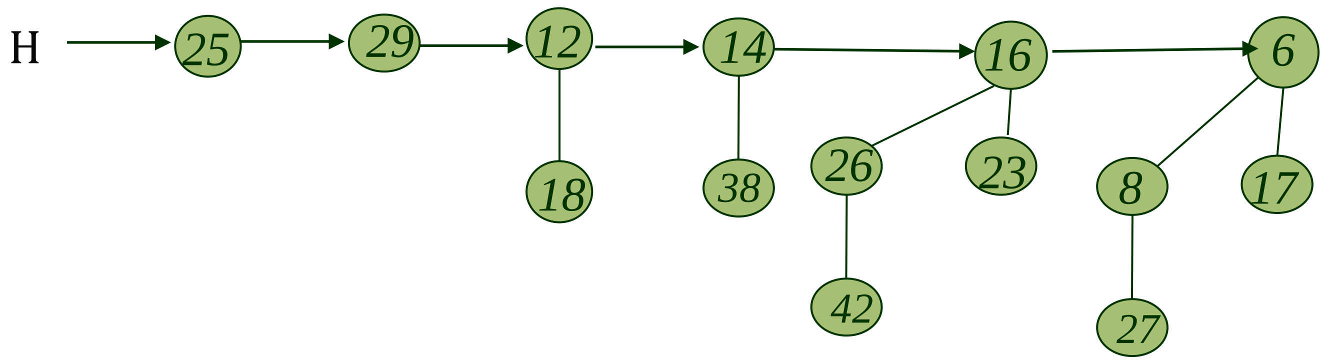
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



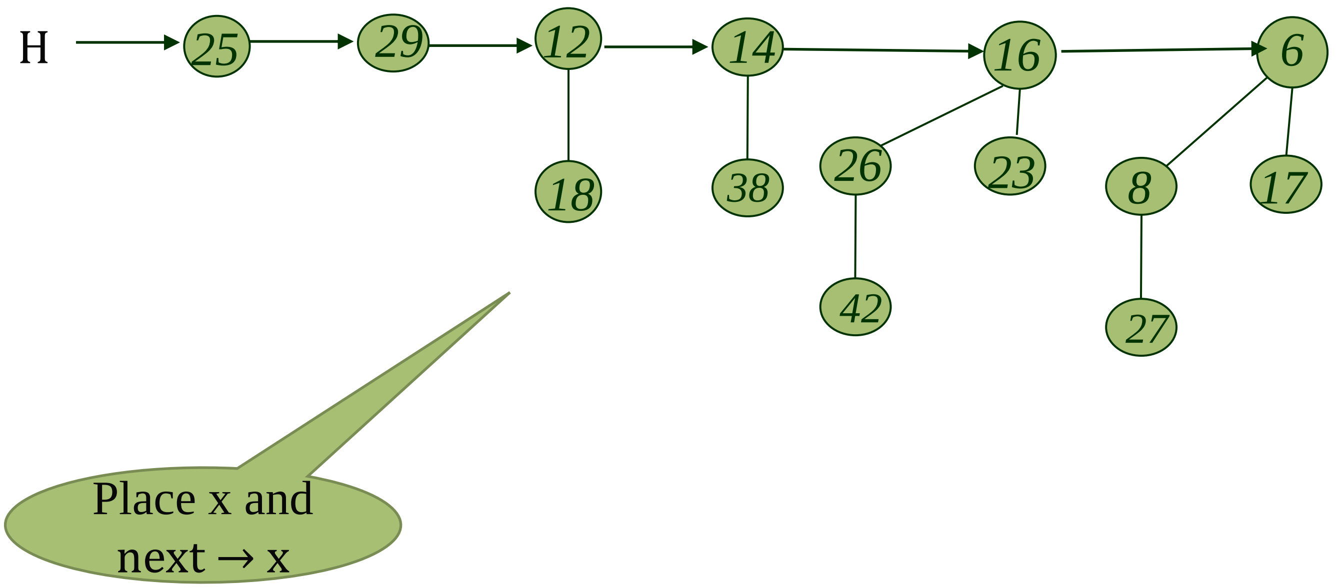
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



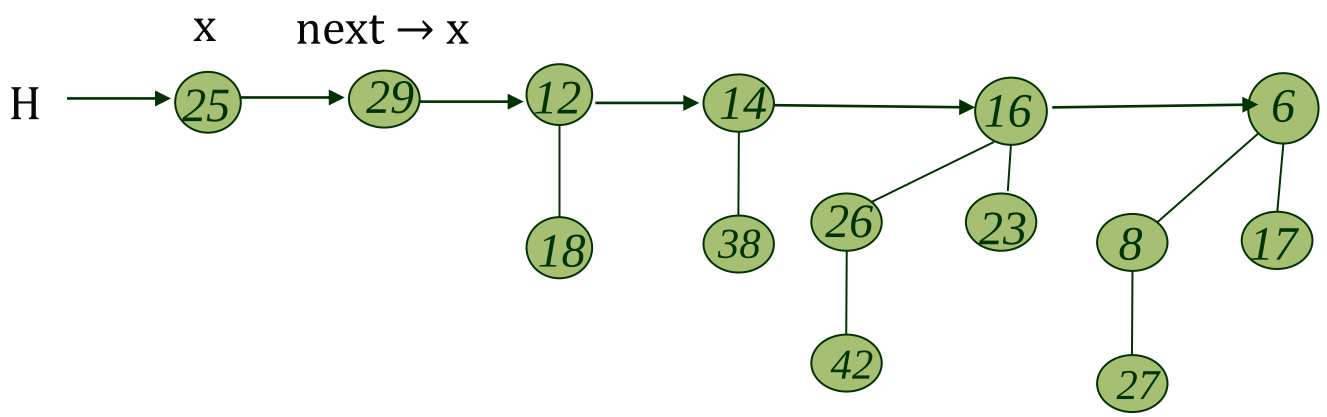
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



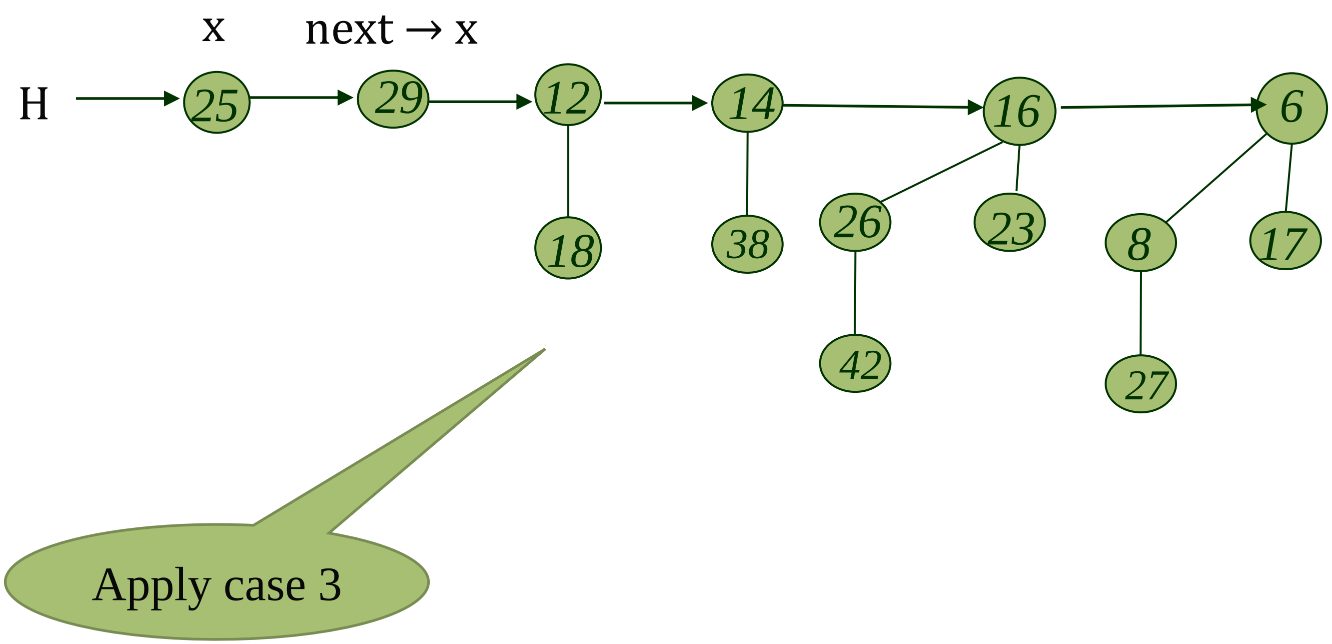
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



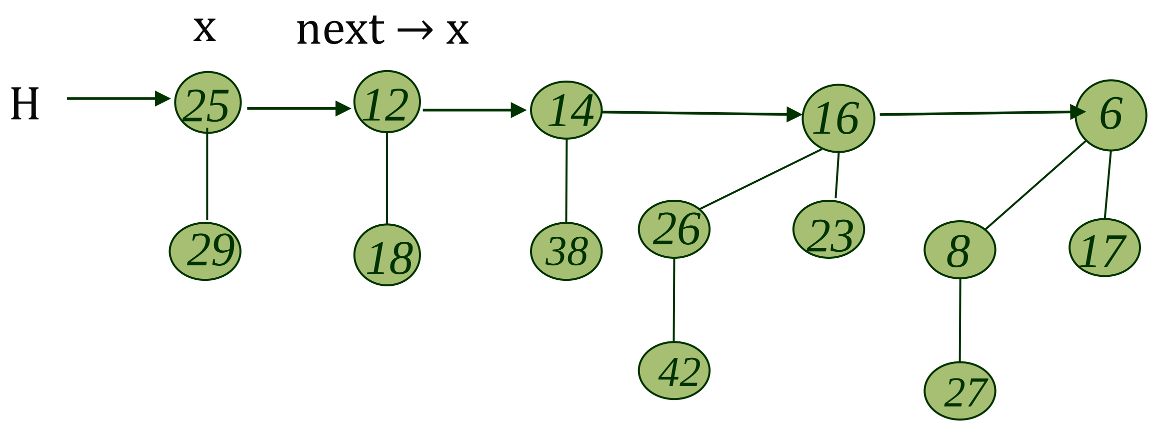
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



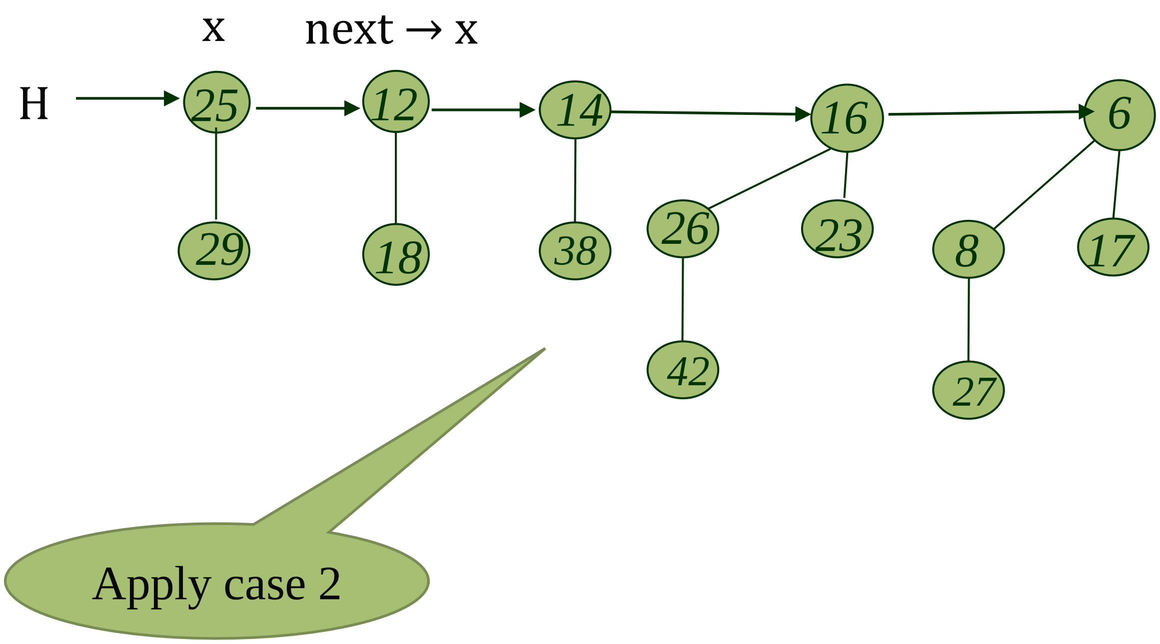
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



Binomial Heap (Operations_7)

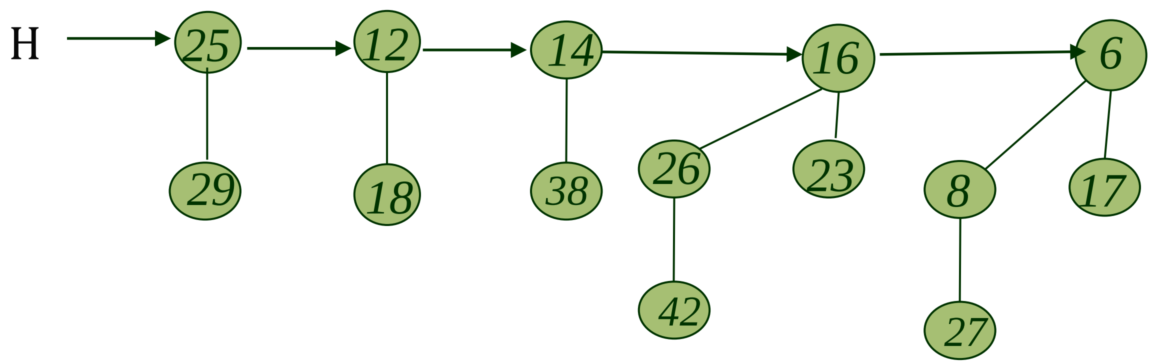
Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).

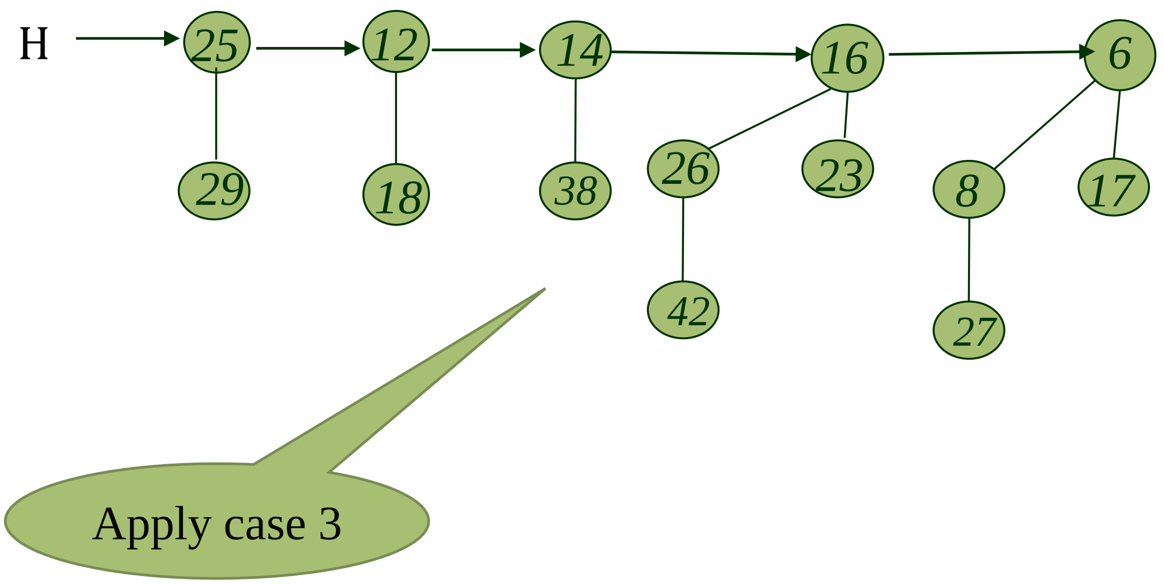
prev → x   x   next → x



Binomial Heap (Operations_7)

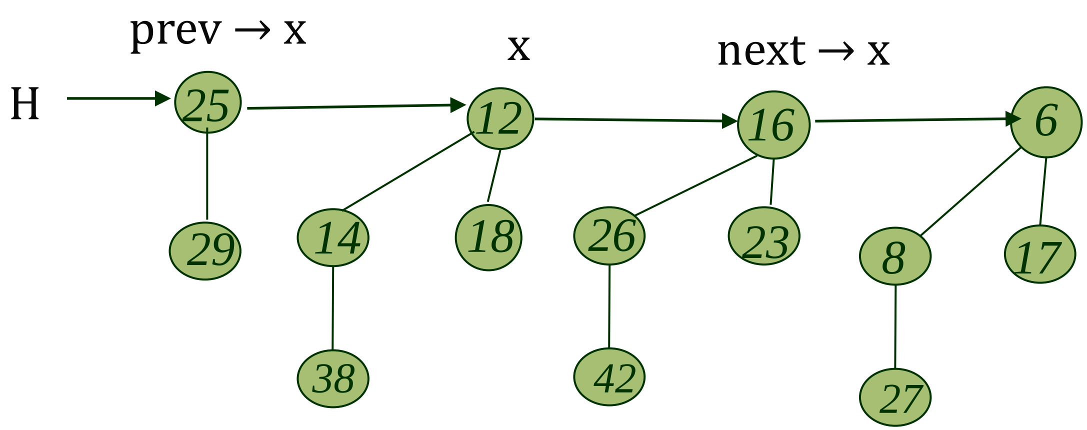
Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).

prev → x   x   next → x



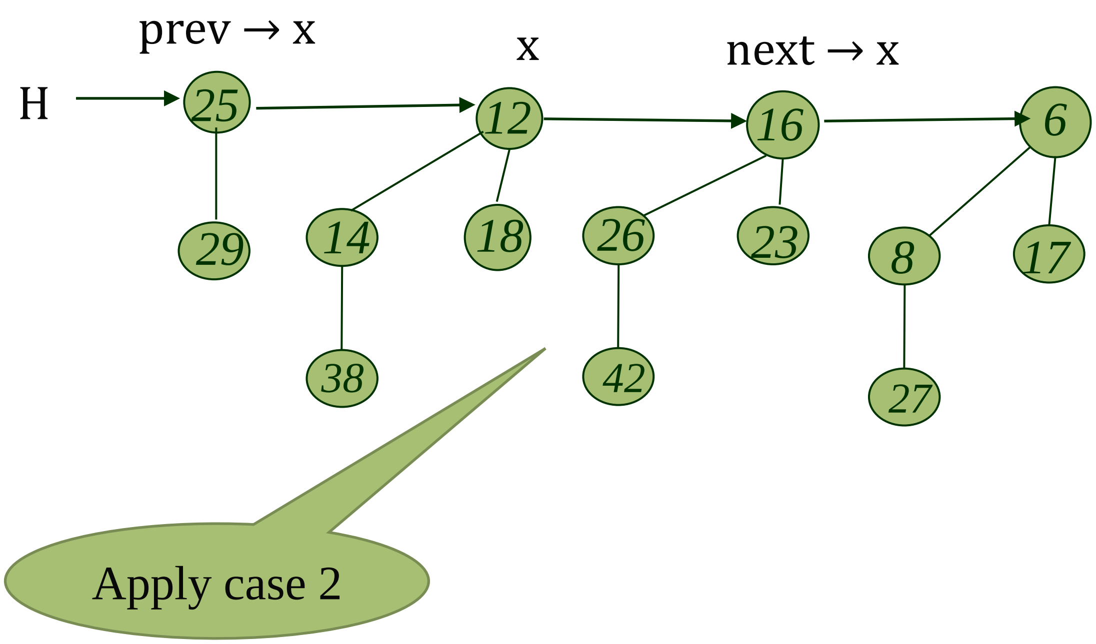
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



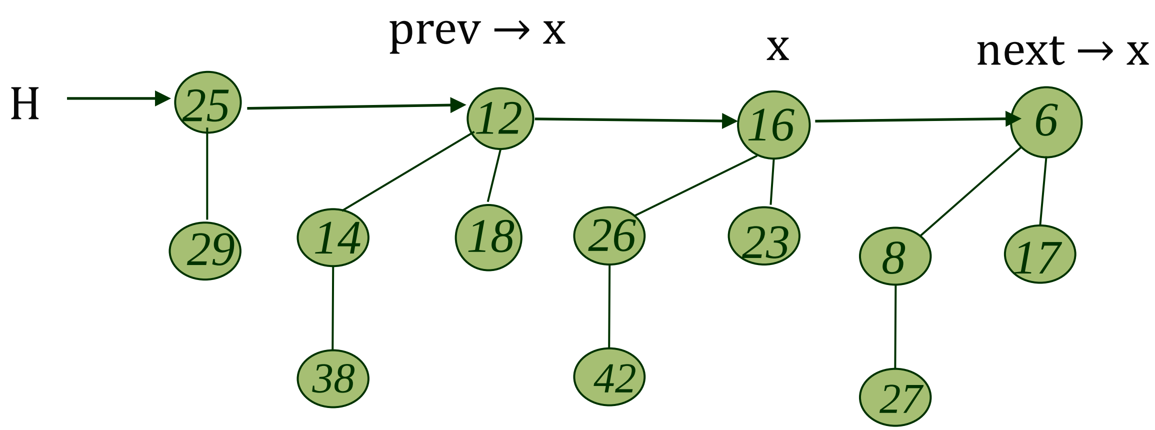
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



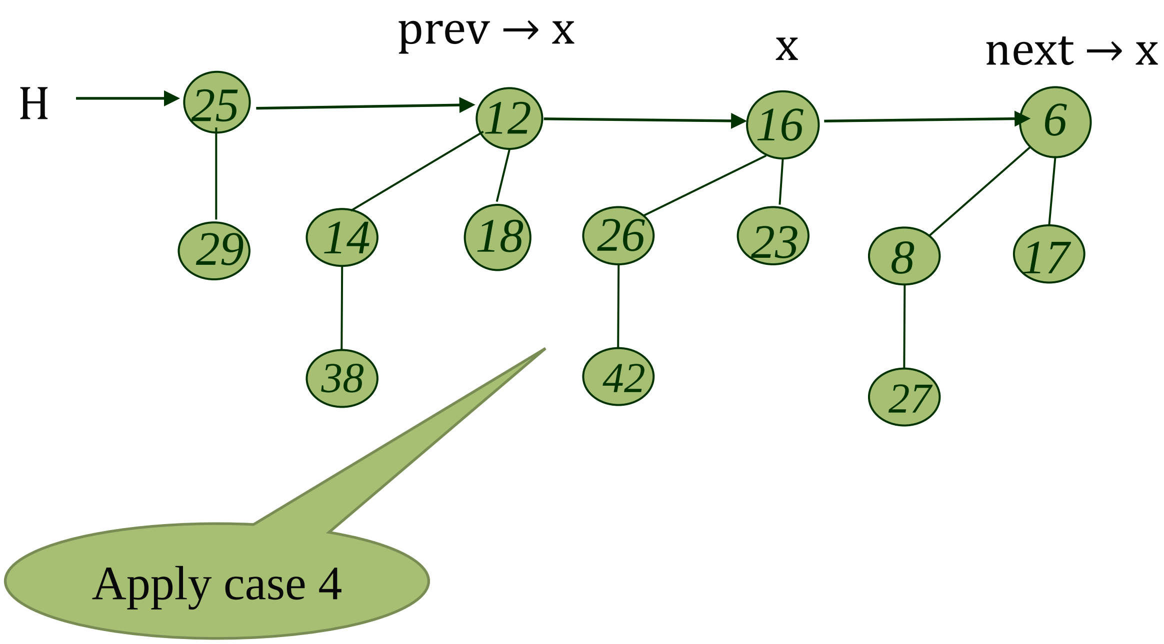
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



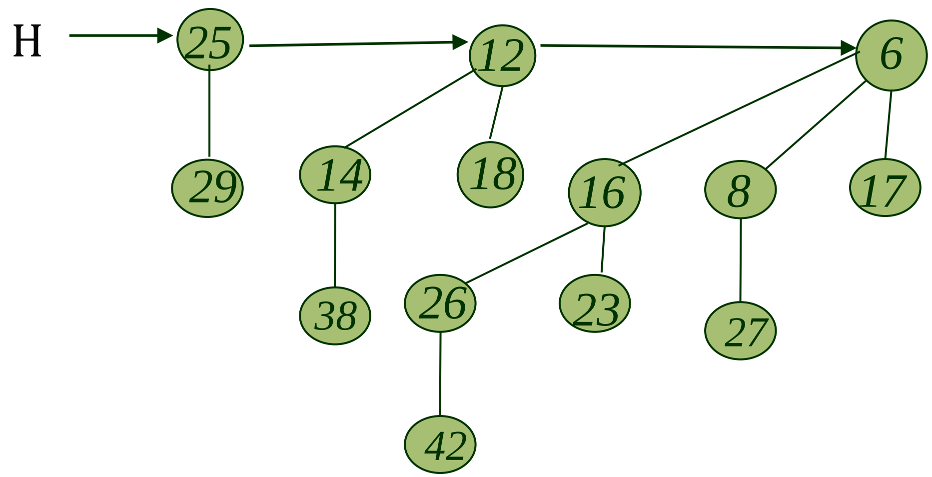
Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



Binomial Heap (Operations_7)

Example: Delete the key x with an assumption that no node currently in the binomial heap has a key of -∞(i.e. k).



Binomial Heap

The running time of binomial Heap w.r.t Binary Heap is given below

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)
INSERT	$\Theta(\lg n)$	$O(\lg n)$
MINIMUM	$\Theta(1)$	$O(\lg n)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$
DECREASEKEY	$\Theta(\lg n)$	$\Theta(\lg n)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$

Thank u