Binomial Heaps

Manoj Kumar
DTU, Delhi

Binomial Heaps

- Known as ***mergeable heaps***.
- Supports following five operations:

1. **MAKE-HEAP():** creates and returns a new heap with no elements.
2. **INSERT(H, x)**: Inserts node x, into heap H.
3. **MINIMUM(H)**: returns a pointer to the node in heap H whose key is minimum.
4. **EXTRACT-MIN(H):** deletes the node from heap H whose key is minimum, returning pointer o the node.
5. **UNION(H1,H2)**: creates and returns new heap that contains all the elements of heaps H1 and H2. H1 and H2 are destroyed in this process.

Binomial Heaps: Definitions

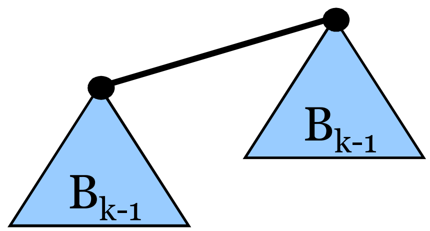
Binomial Heap: Collection of binomial trees (satisfying some properties).

Binomial Trees

- Definition is **inductive**.
- These are **ordered** trees, i.e., order of children is important.

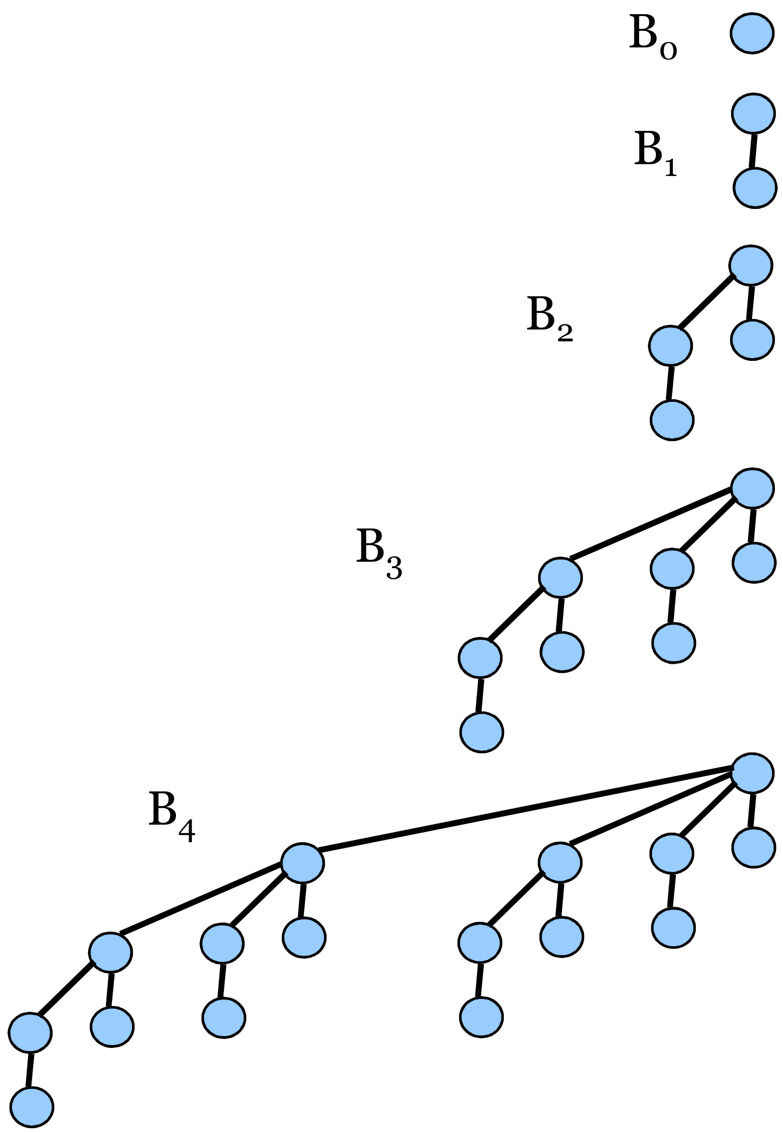
Binomial Trees

- **Base Case:** B_0 = single node is a binomial tree.
- **Inductive Step:**

B_k = 

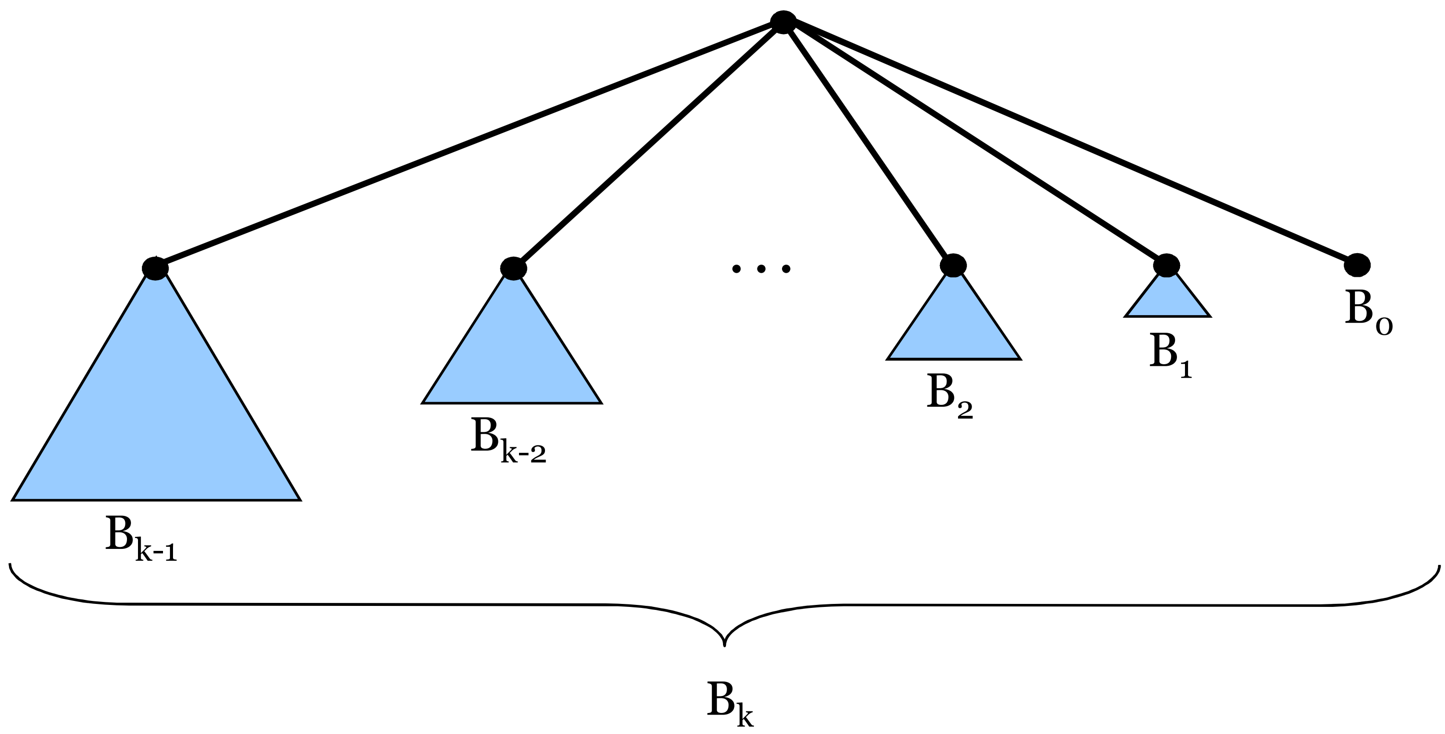
is a binomial tree.

Examples



depth	# nodes
0	1
1	4
2	6
3	4
4	1

Another Way to Look at B_k



Properties of Binomial Trees

Lemma 1: For the binomial tree B_k,

1. There are 2^k nodes.

2. Tree height is k.

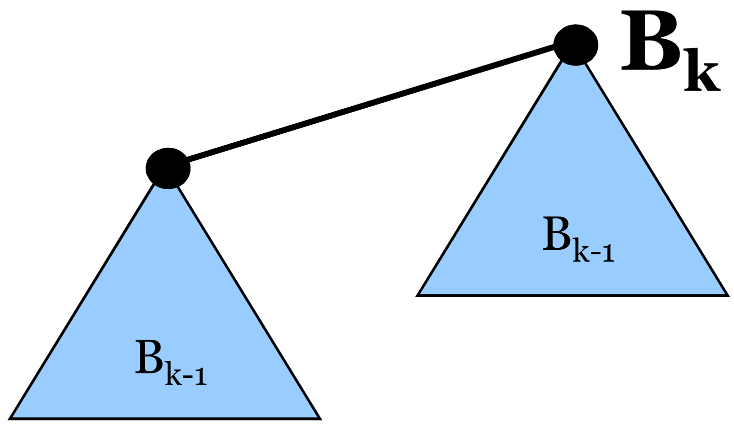
3. $\binom{\mathbf{k}}{\mathbf{i}}$ nodes at depth i, i = 0, 1, …, k **[binomial coefficients]**.

4. Root has degree k, other nodes have smaller degree. i^{th} child of root is root of subtree B_i, where i = k–1, k–2, …, 0 [B_{k-1} is Left Most, B_0 is Right Most].

Proof: Inductive

1. Binomial tree B_k consists of two copies of B_{k-1}, so B_k has $2^{k-1} + 2^{k-1} = 2^k$ nodes.

2. The way in which two copies of are connected, height of B_k is one greater than height of $B_{k-1.}$ So the height of B_k is (k-1)+1= k.



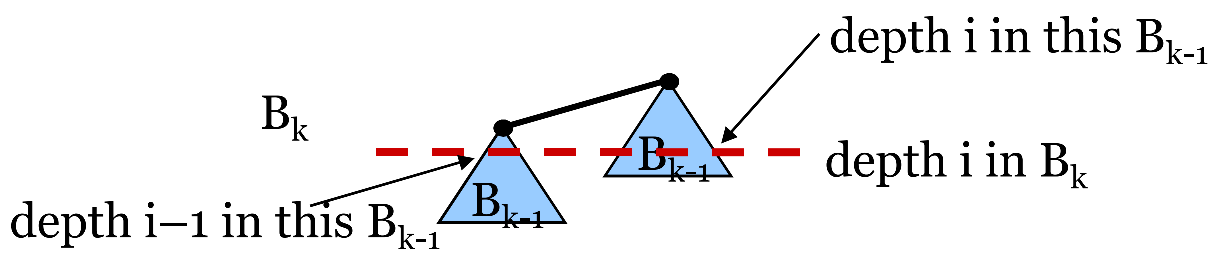
Proof: Inductive

3. Let D(k,i) be the number of nodes at depth I of binomial tree B_k. Since B_k is composed of two copies of B_{k-1} linked together, a node at depth i in B_{k-1} appears in B_k once at depth i and once at depth i+1.

Thus number of nodes in B_k at depth i is the number of nodes at depth i in B_{k-1} plus the number of nodes at depth i-1 in B_{k-1}. Thus

$$D(k,i) = D(k-1,i) + D(k-1,i-1)$$

$$= \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$$



proof

4. Root degree of B_k = 1 + root degree of B_{k-1}

   = 1 + k–1 , induction hypothesis

   = k

<u>Corollary :</u> The maximum degree in an n-node binomial tree is lg n.

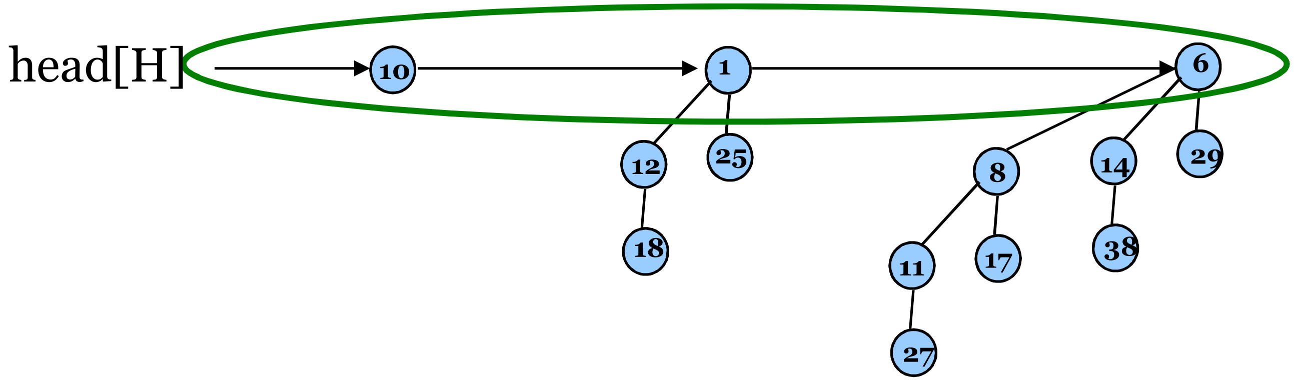
Binomial Heaps

- Binomial heap H is a set of binomial trees satisfying following **binomial-heap properties**:
  1. **Each binomial tree in H is Heap ordered:** the key of a node is greater than or equal to the key of it's parent.
     - Implies root of a binomial tree has the smallest key in that tree.
  2. There is at most one binomial tree in H whose root has a given degree.
     - Implies B.H. with n nodes has at most $\lfloor \lg n \rfloor + 1$ B.T.'s. Think of n in binary: $\langle b_{\lfloor \lg n \rfloor}, \ldots, b_o \rangle$, i.e.,

$$n = \sum_{i=1}^{\lfloor \lg n \rfloor} b_i 2^i$$

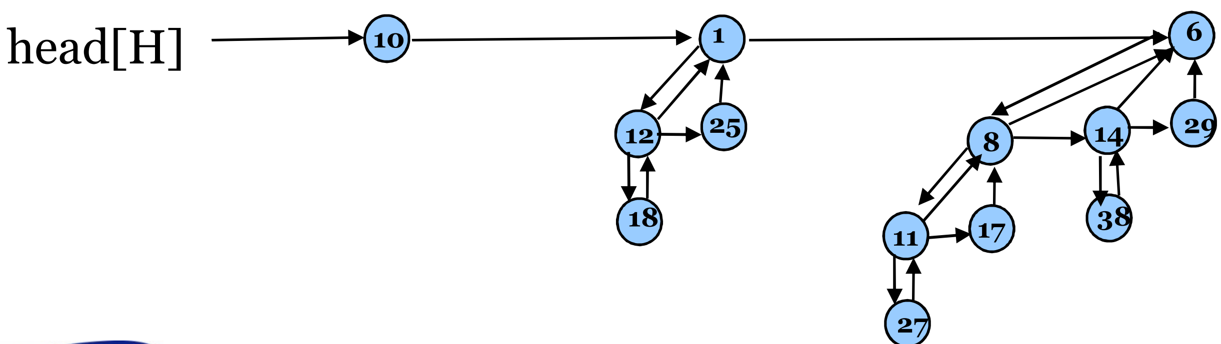
B.H contains B_i iff b_i is 1.

Representing Binomial Heaps



Each node is represented by a structure like this

parent	
key	
degree	
child	sibling



Operations on Binomial Heaps

MAKE-BINOMIAL-HEAP(): simply allocates and return an object H, where head[H] = NIL.

Running time is Θ(1)

```
MAKE-BINOMIAL-HEAP()
1.  Create new head node H
2.  head[H]←NIL
3.  return H
```

Operations on Binomial Heaps

```
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
```

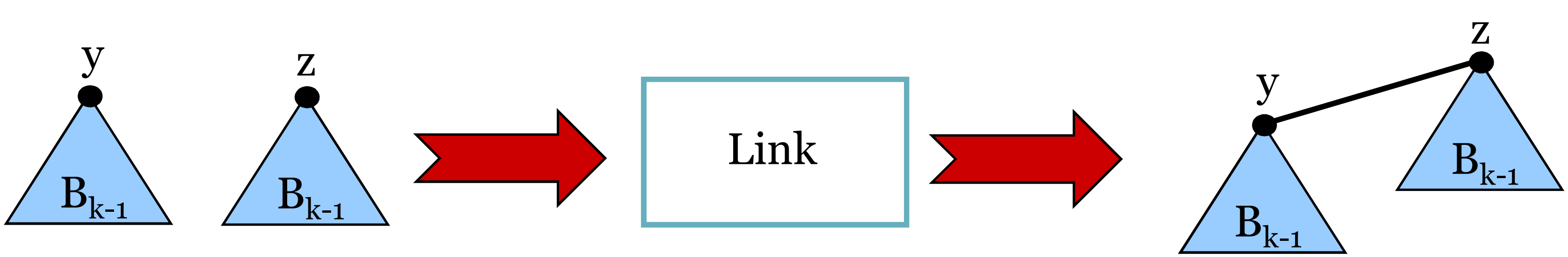
Time is O(lg n).

Linking Two Binomial Trees

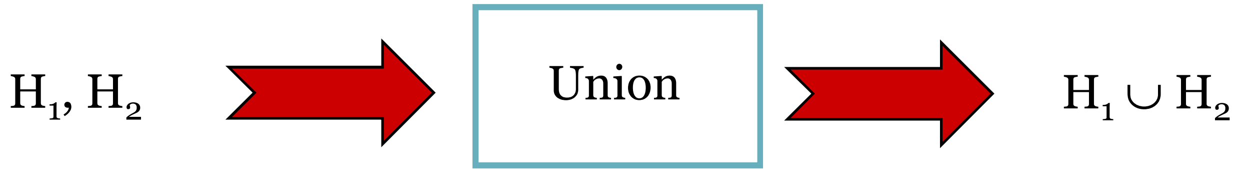
Linking two binomial trees whose roots have same degree.

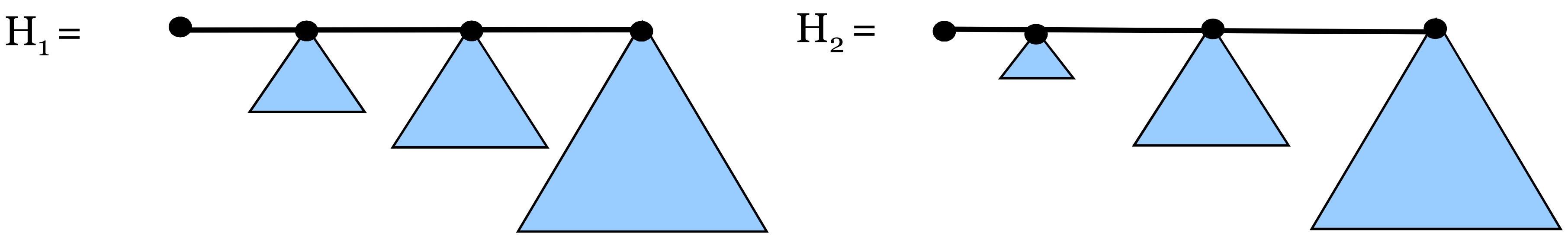
BINOMIAL-LINK(y,z)

1. p[y] ← z;
2. sibling[y] ← child[z];
3. child[z] ←y;
4. degree[z] ← degree[z] + 1

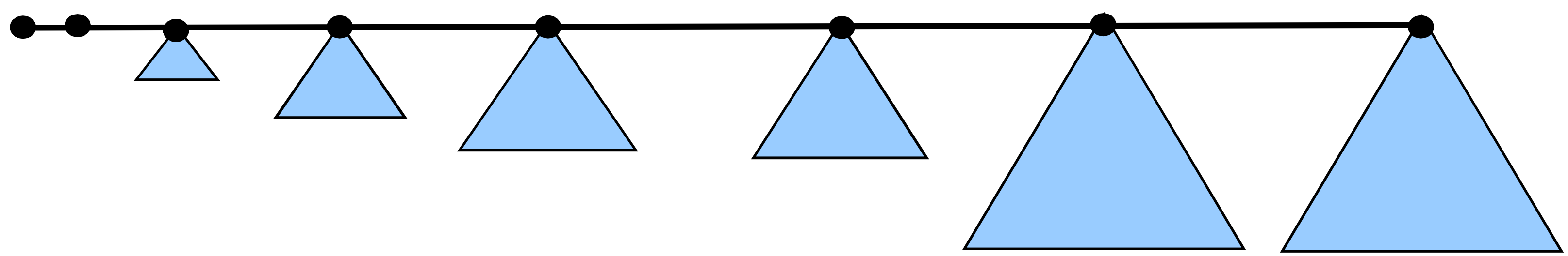


UNION





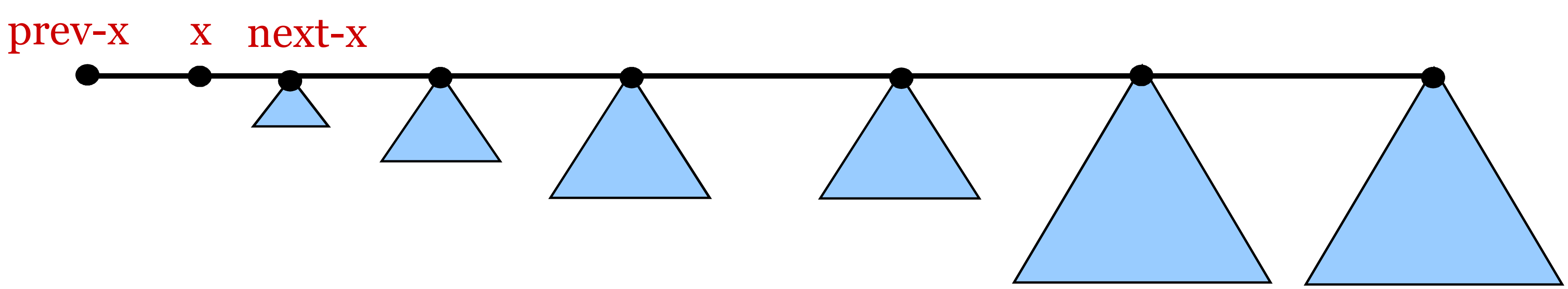
First, simply merge the two root lists by root degree (like merge sort).



Remaining Problem: Can have two trees with the same root degree.

UNION…

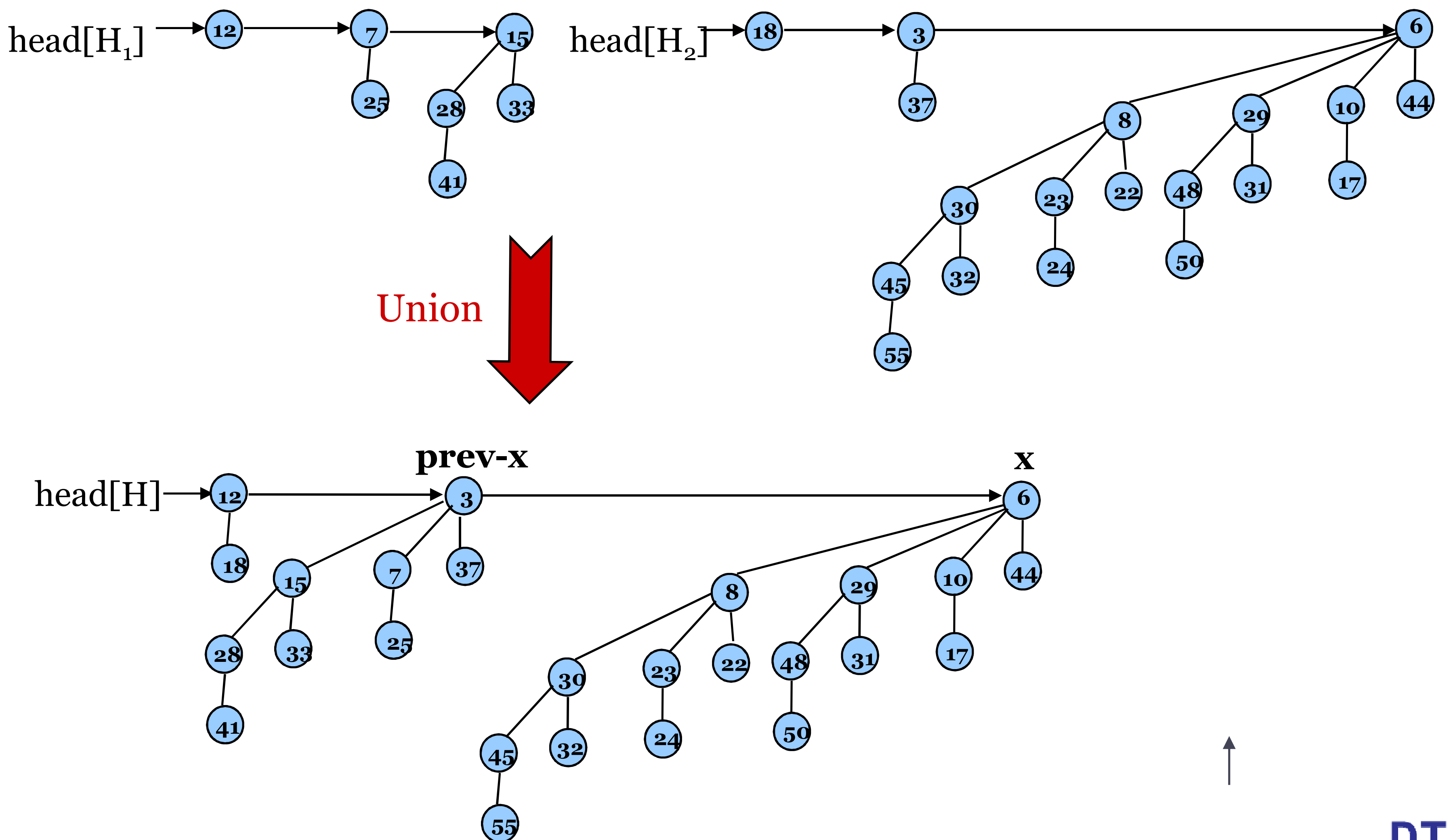
Union traverses the new root list like this:



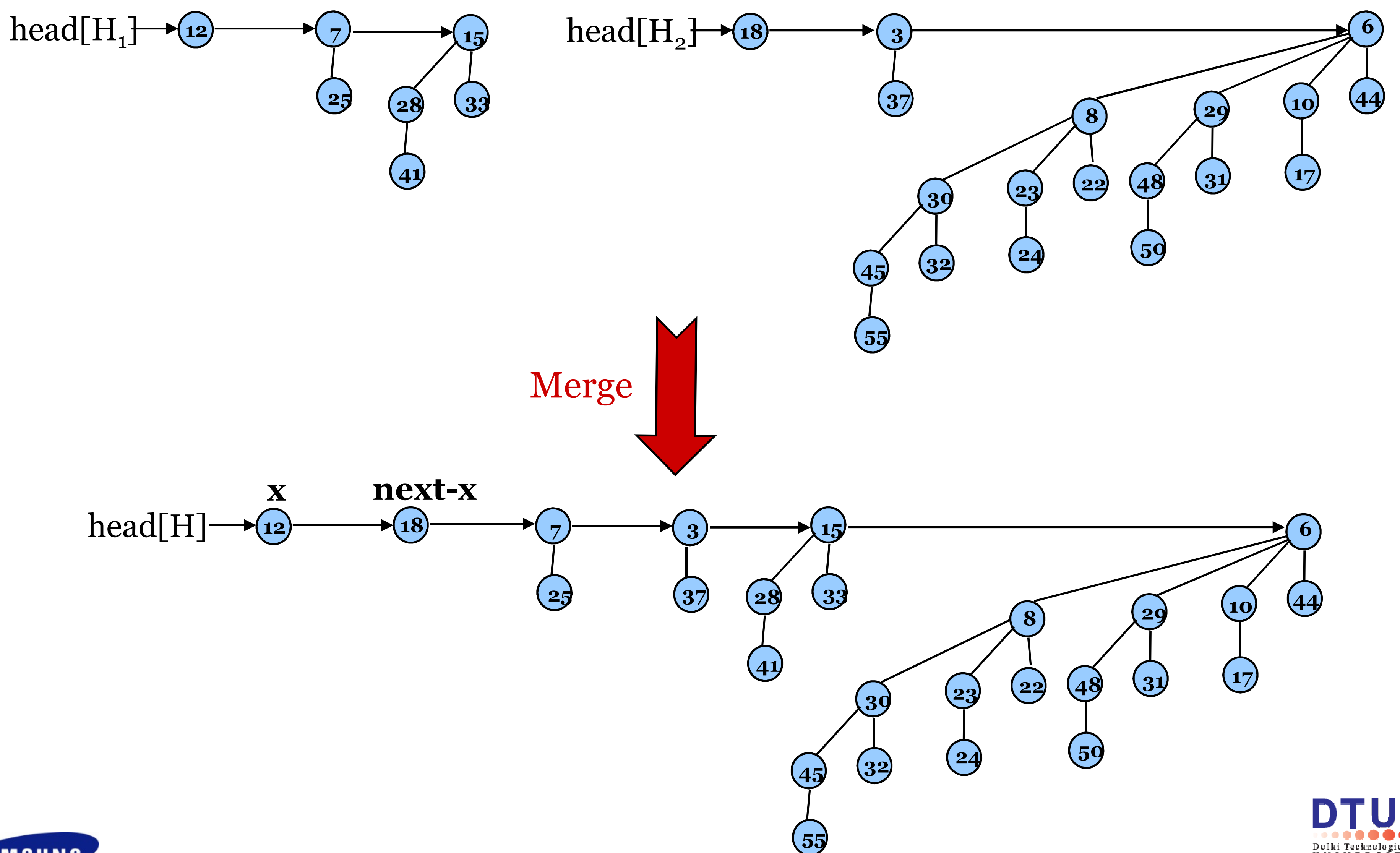
Depending on what x, next-x, and sibling[next-x] point to, Union links trees with the same root degree.

<u>Note:</u> We may temporarily create <u>three</u> trees with the same root degree.

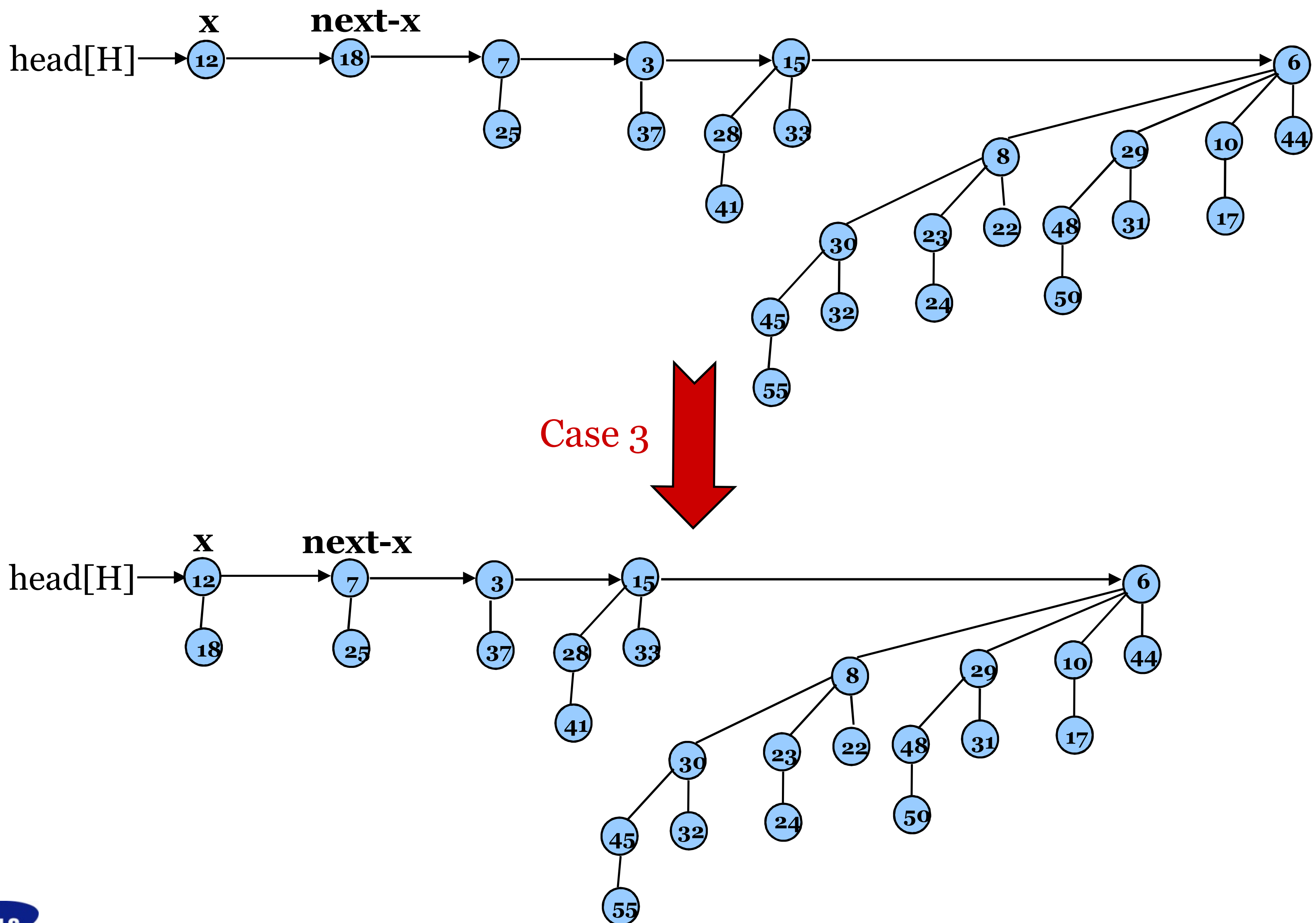
UNION…


head[H_1]
12
7
15
25
28
33
41
head[H_2]
18
3
6
37
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
Union
head[H]
prev-x
x
12
3
6
18
15
7
37
8
29
10
44
28
33
25
30
23
22
48
31
17
41
45
32
24
50
55

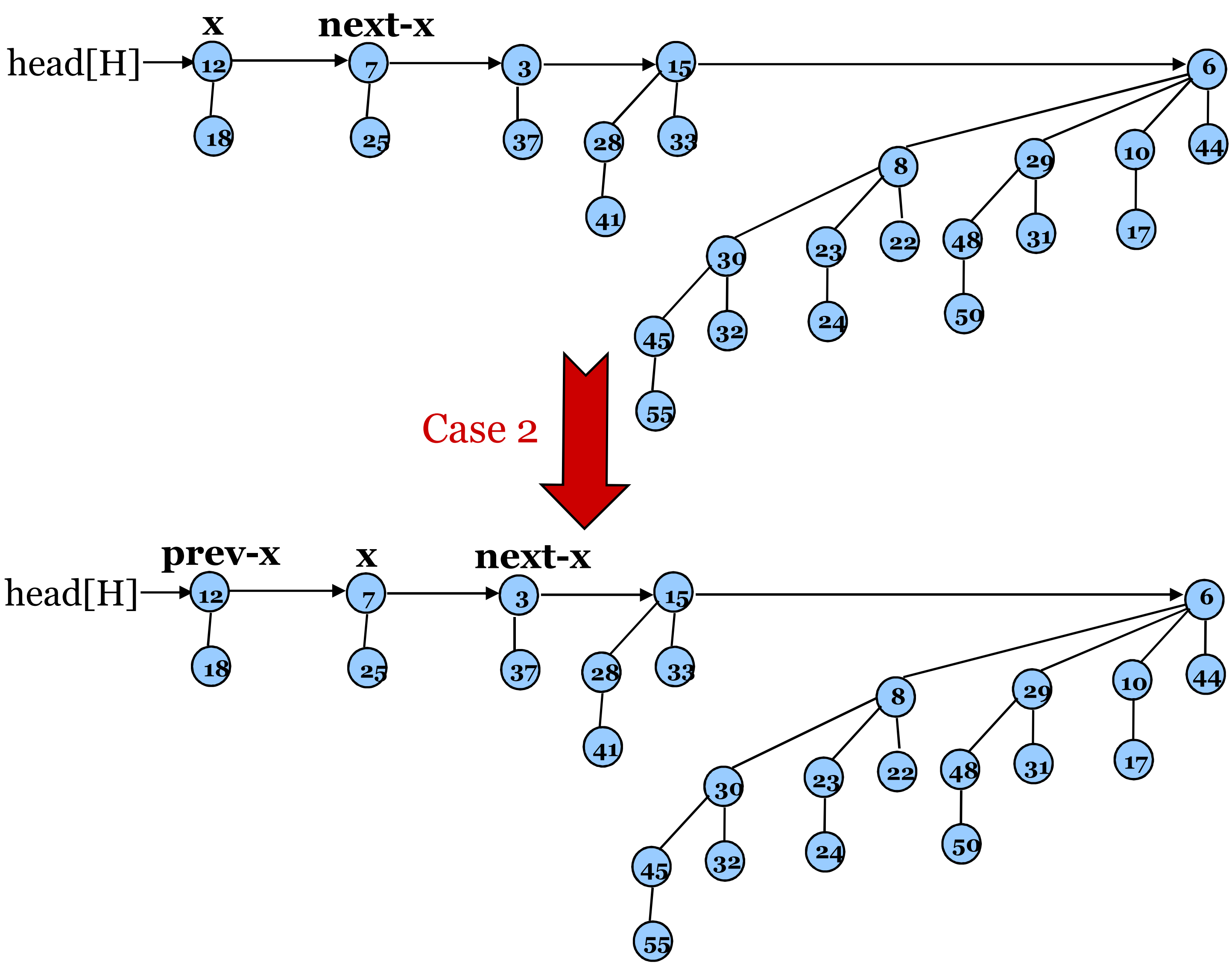
UNION:Example



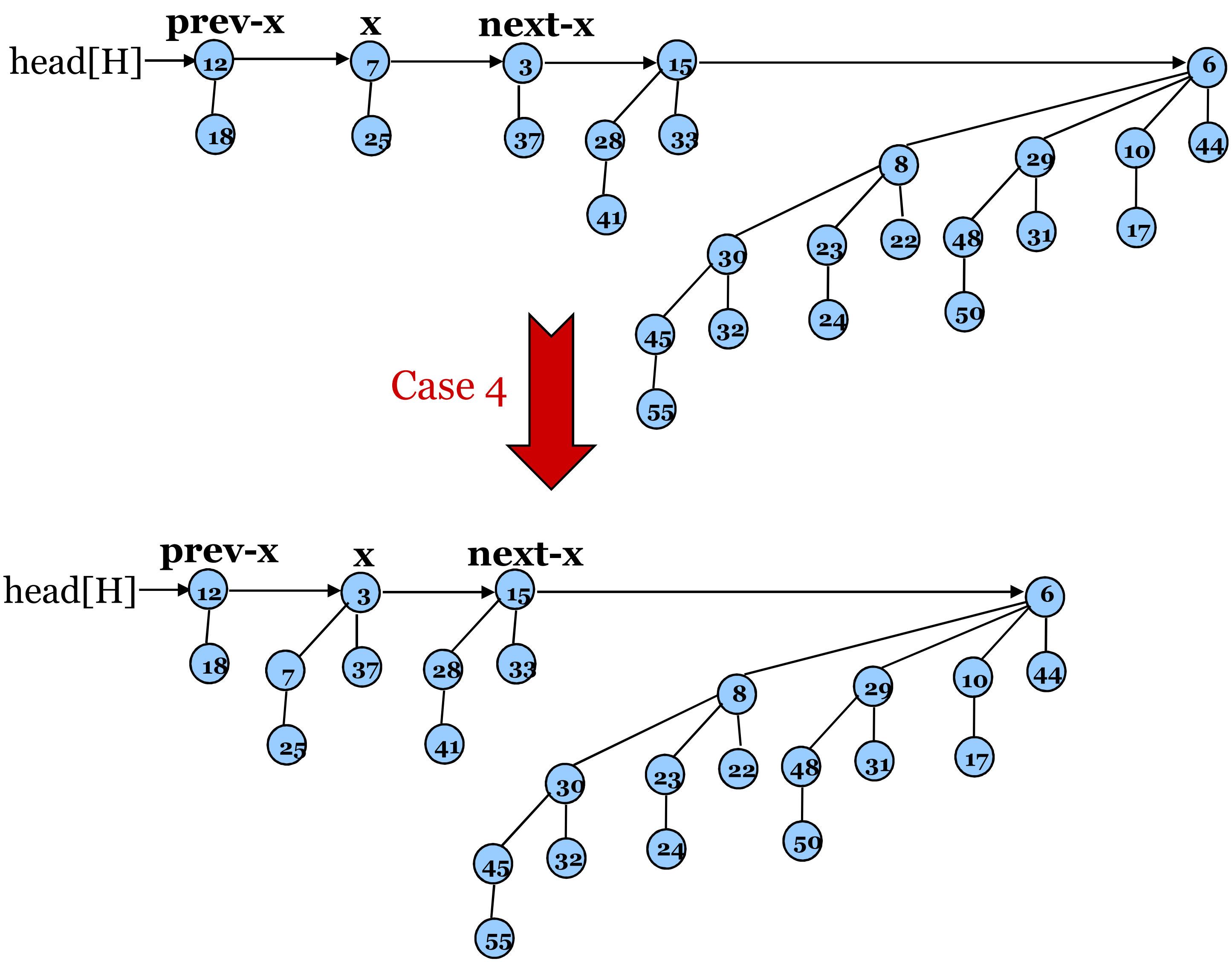
UNION:Example…



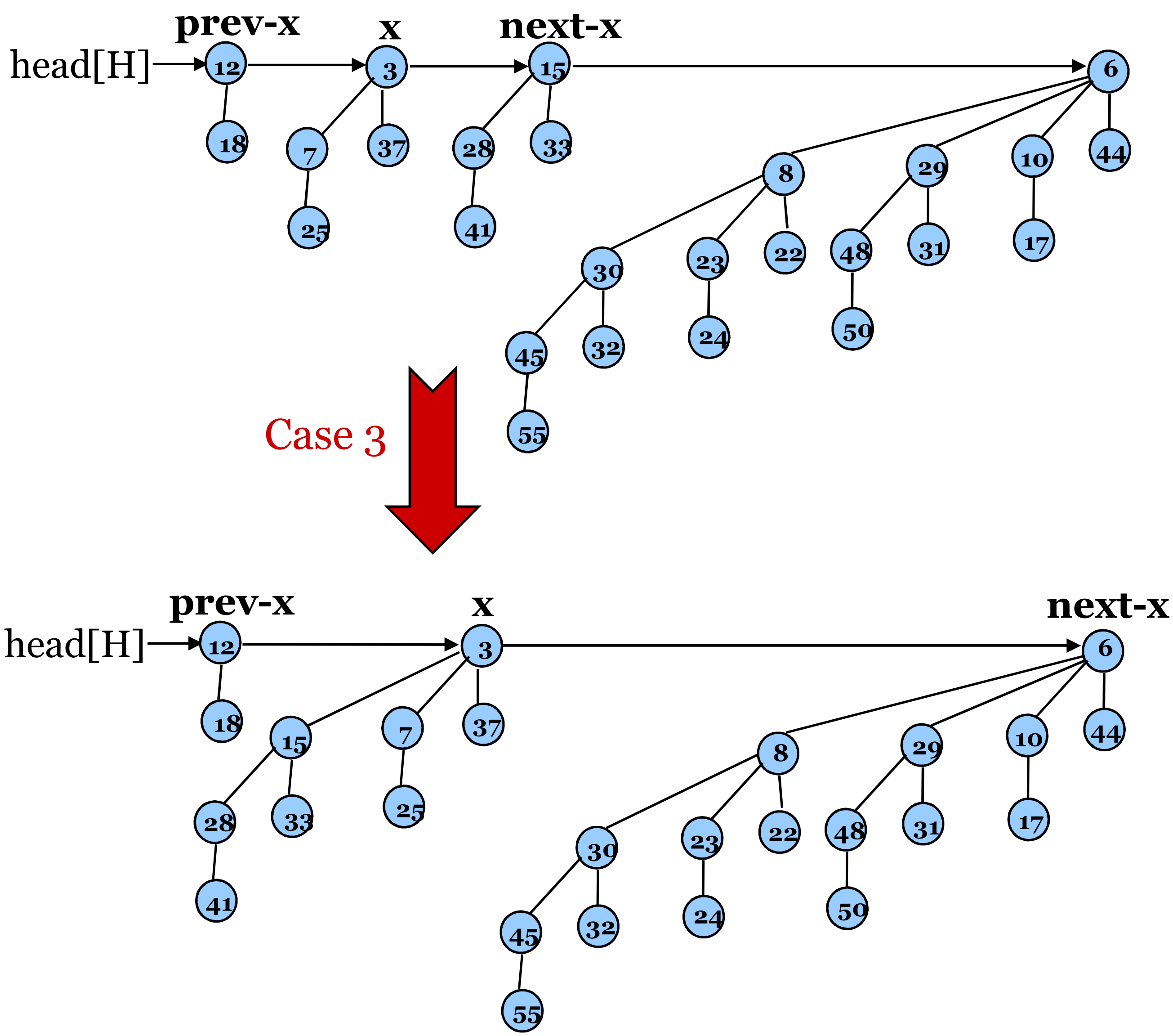
UNION:Example…



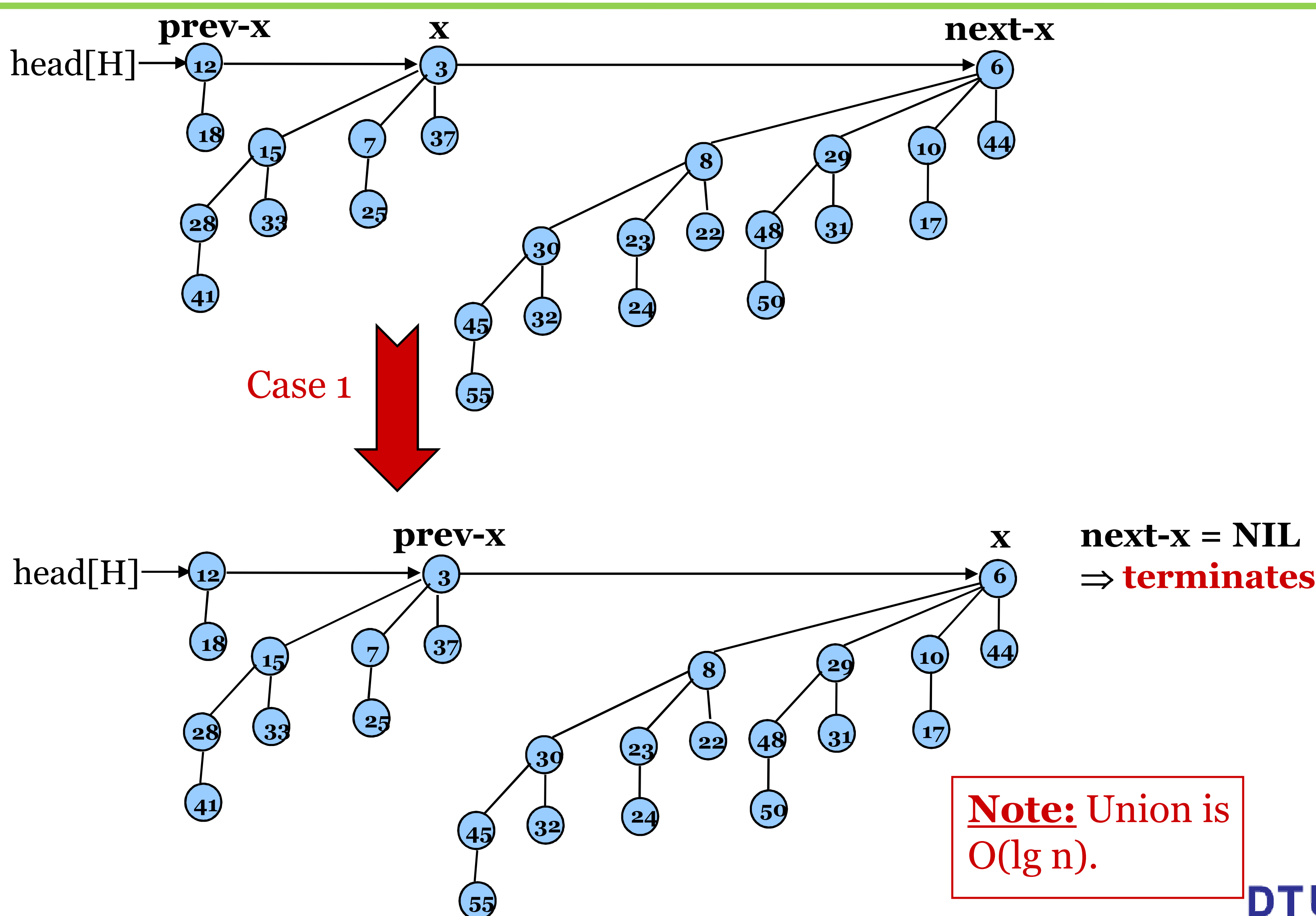
UNION:Example…



UNION:Example…



UNION:Example…

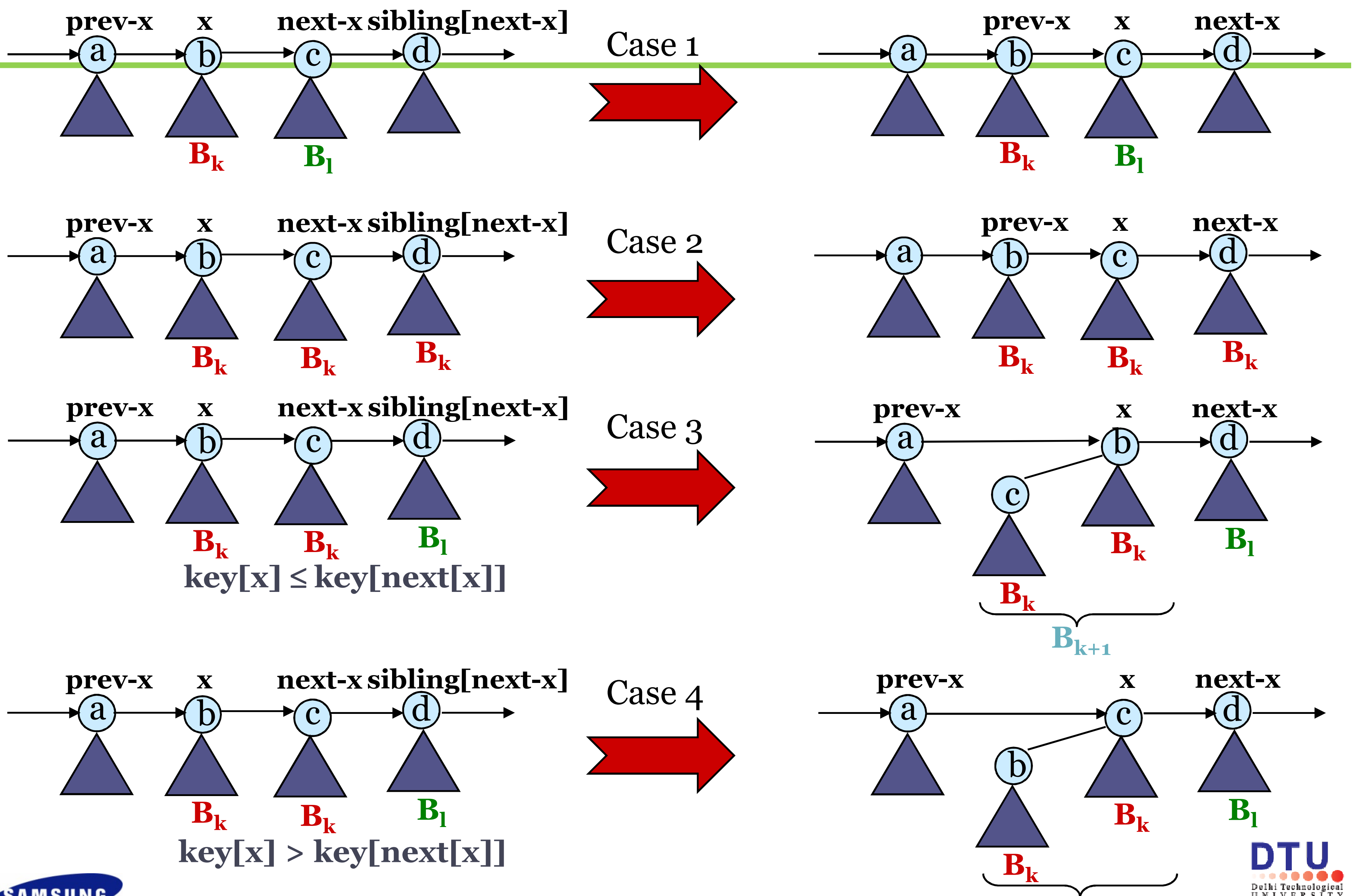


Note: Union is O(lg n).

Code for UNION

```
BINOMIAL-HEAP-UNION(H1, H2)
    H ← MAKE-BINOMIAL-HEAP();
    head[H] ← BINOMIAL-HEAP-MERGE(H1, H2);          /* simple merge of root lists */
    Free the objects H1 and H2, but not the list they point to
    if head[H] = NIL
      then return H
    prev-x ← NIL;
    x ← head[H];
    next-x ← sibling[x];
    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 & 2
                    x ← next-x;                         Cases 1 & 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
```

Delhi Technological University

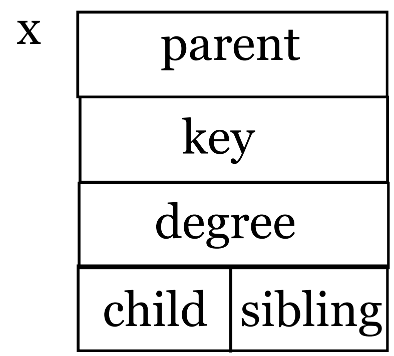


prev-x
x
next-x
sibling[next-x]
a
b
c
d
B_k
B_l
Case 1
prev-x
x
next-x
a
b
c
d
B_k
B_l
prev-x
x
next-x
sibling[next-x]
a
b
c
d
B_k
B_k
B_k
Case 2
prev-x
x
next-x
a
b
c
d
B_k
B_k
B_k
prev-x
x
next-x
sibling[next-x]
a
b
c
d
B_k
B_k
B_l
key[x] ≤ key[next[x]]
Case 3
prev-x
x
next-x
a
b
c
d
B_k
B_k
B_l
B_{k+1}
prev-x
x
next-x
sibling[next-x]
a
b
c
d
B_k
B_k
B_l
key[x] > key[next[x]]
Case 4
prev-x
x
next-x
a
c
b
d
B_k
B_k
B_l

INSERT

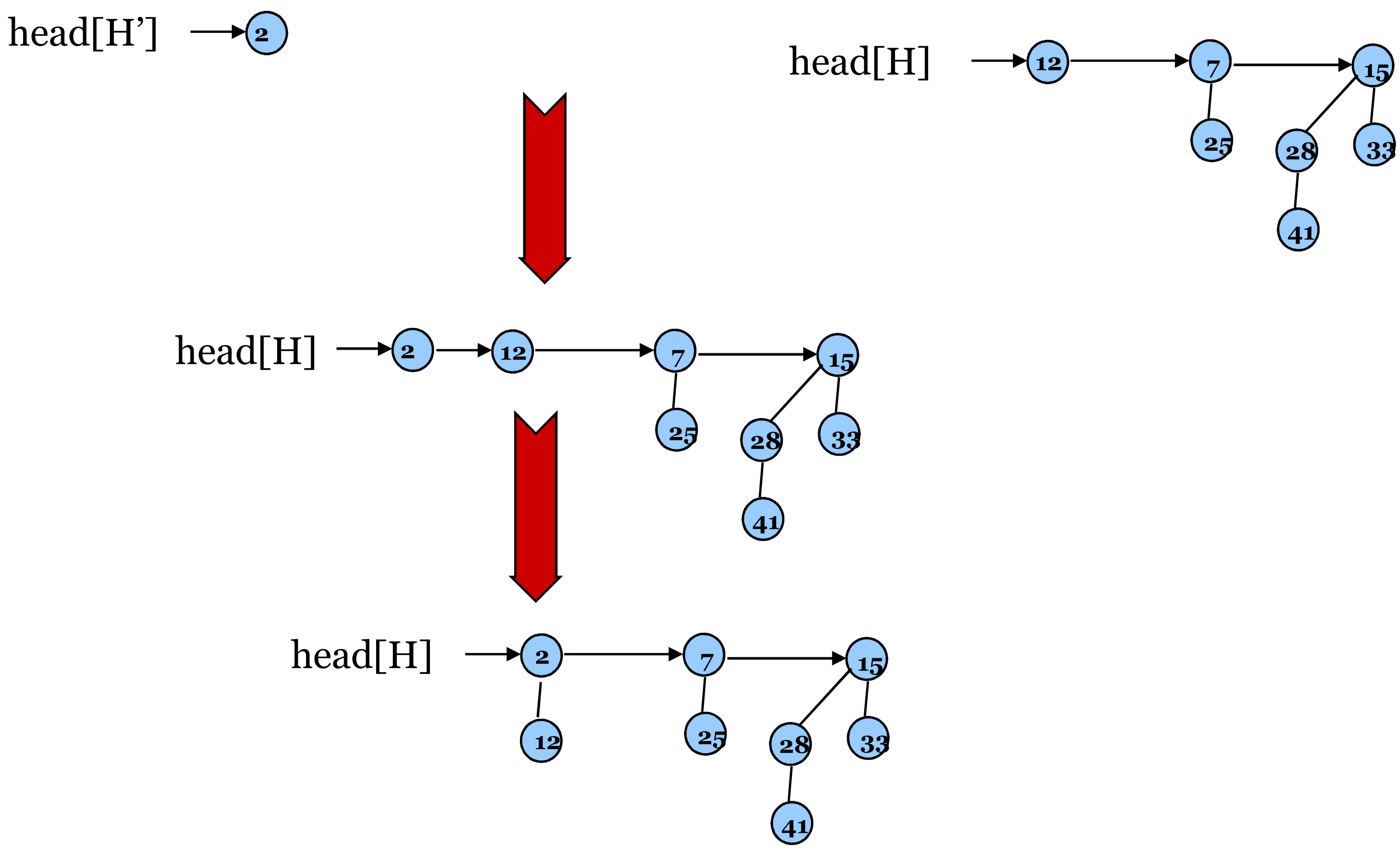
Inserts node x into binomial heap H, assuming that node x already been allocated and key[x] has already been filled.

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

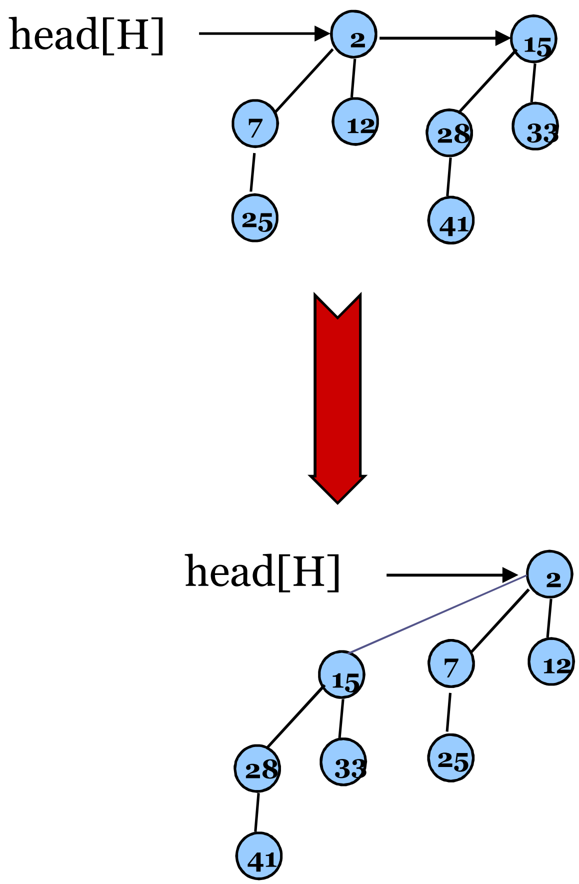


Time O(lg n).

INSERT:Example



INSERT: Example…



Extract-Minimum

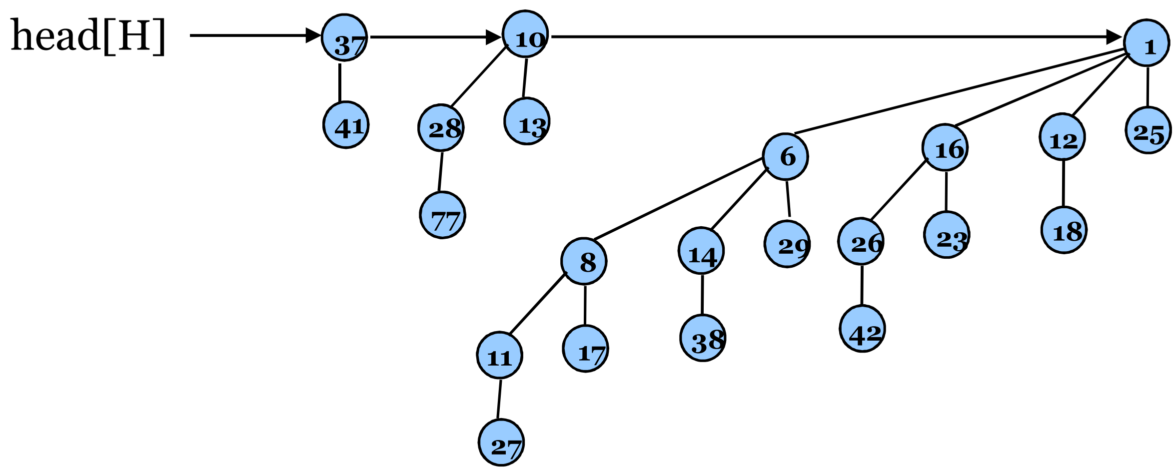
- Extracts the node with minimum key from binomial heap H, and returns a pointer to the extracted node

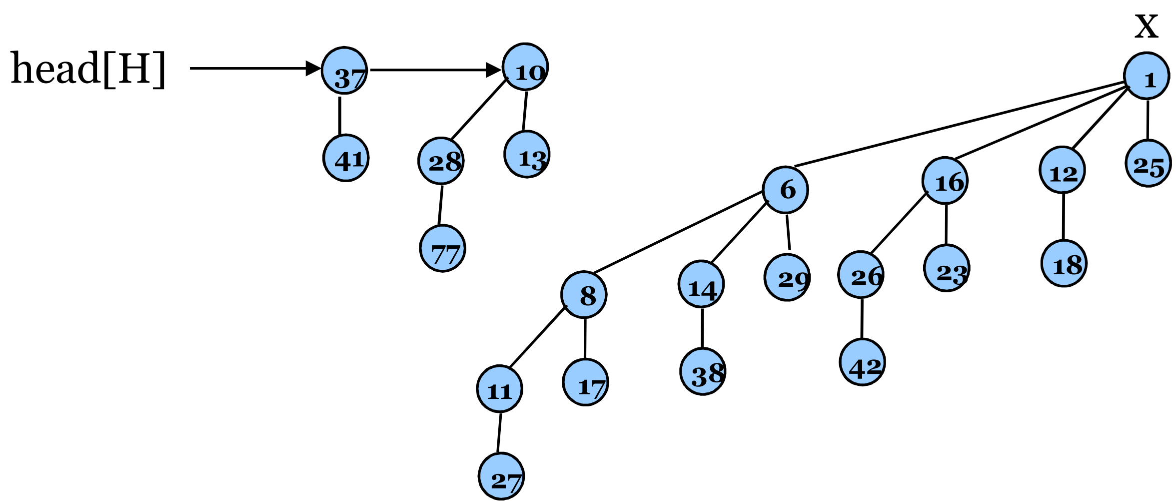
BINOMIAL-HEAP-EXTRACT-MIN(H)

1. remove minimum key root x from H's root list;
2. H′ ← Make-B-H();
3. root list of H′ = x's children in reverse order;
4. H ← Union(H, H′);
5. **return** x

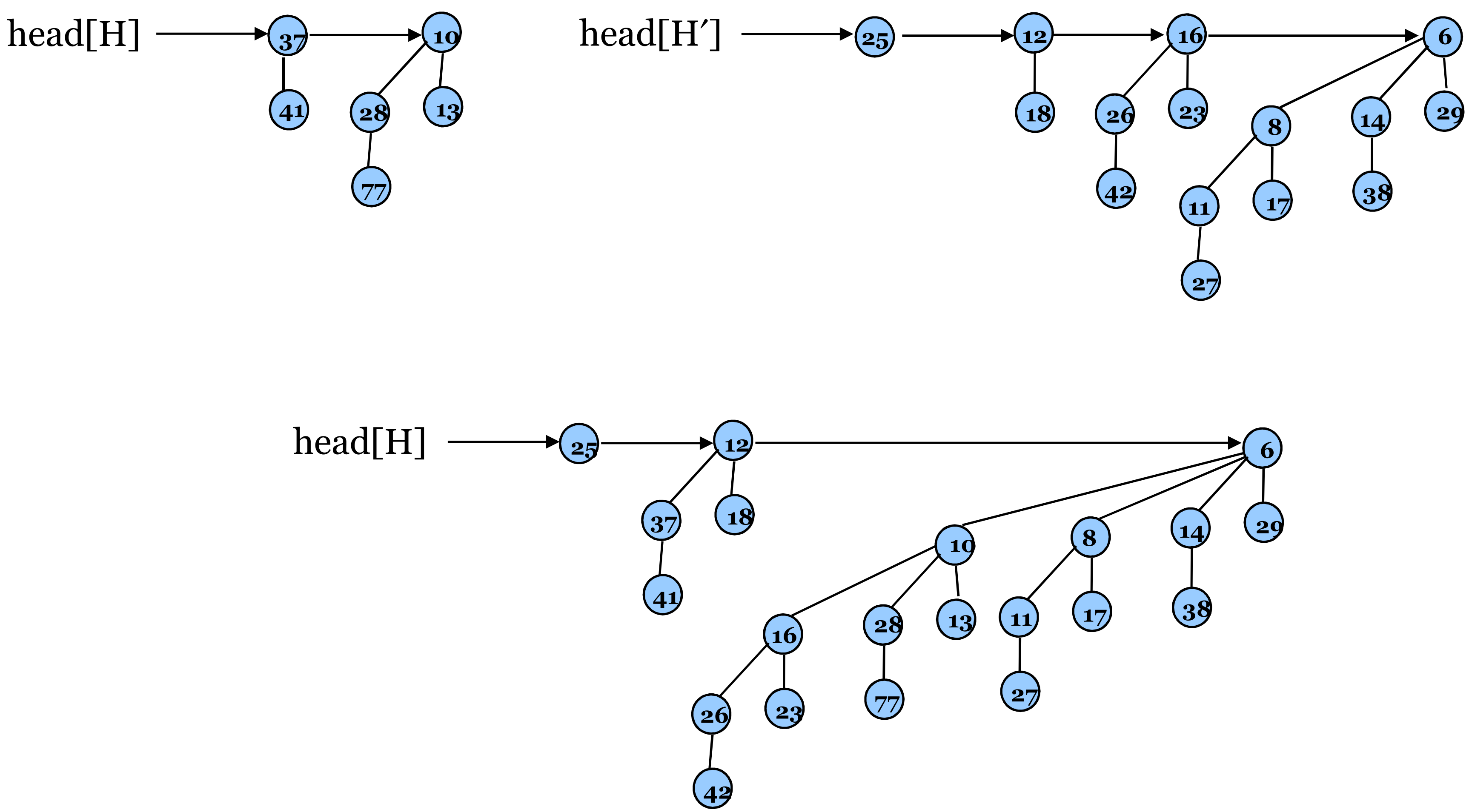
Time O(lg n).

Extract-Min: Example





Extract-Min: Example



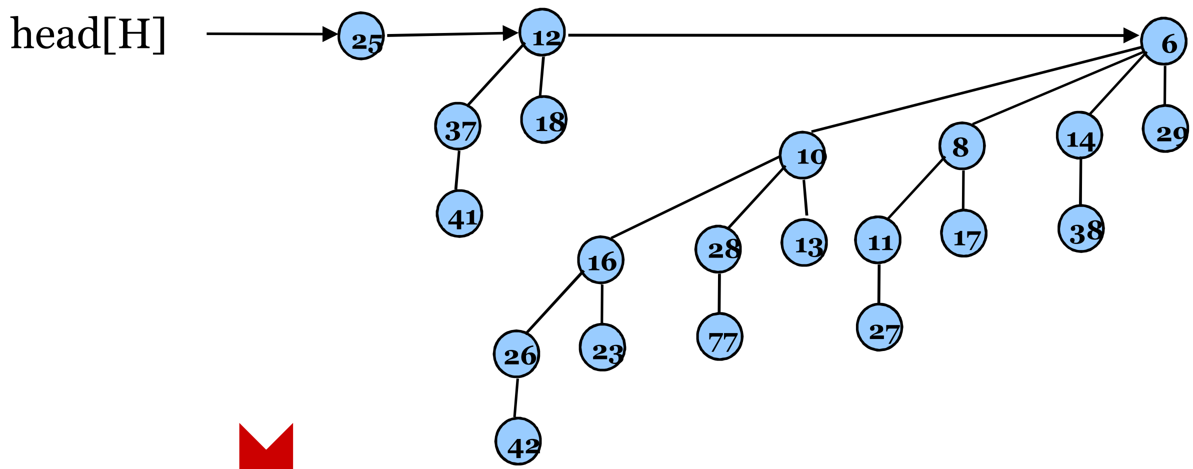
Decrease-Key

Decreases the key of a node x in a binomial heap H to a new value k.

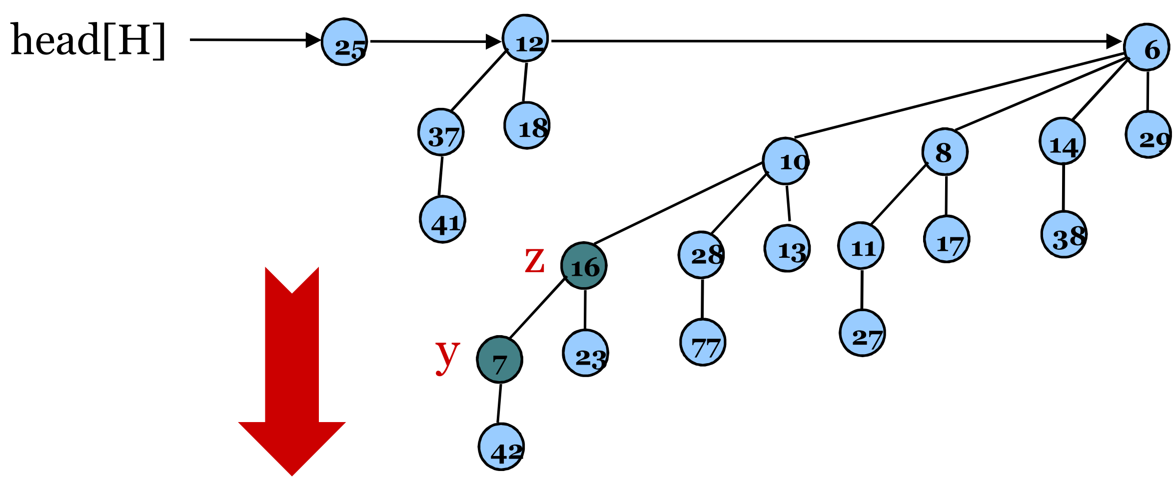
```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
if k > key[x]
        then error "new key is greater than current key"
key[x] ← k;
y ← x;
z ← p[y];
while z ≠ NIL and key[y] < key[z]
        do  exchange key[y] and key[z];
            exchange other satellite fields of y and z
            y ← z;
            z ← p[y]
```

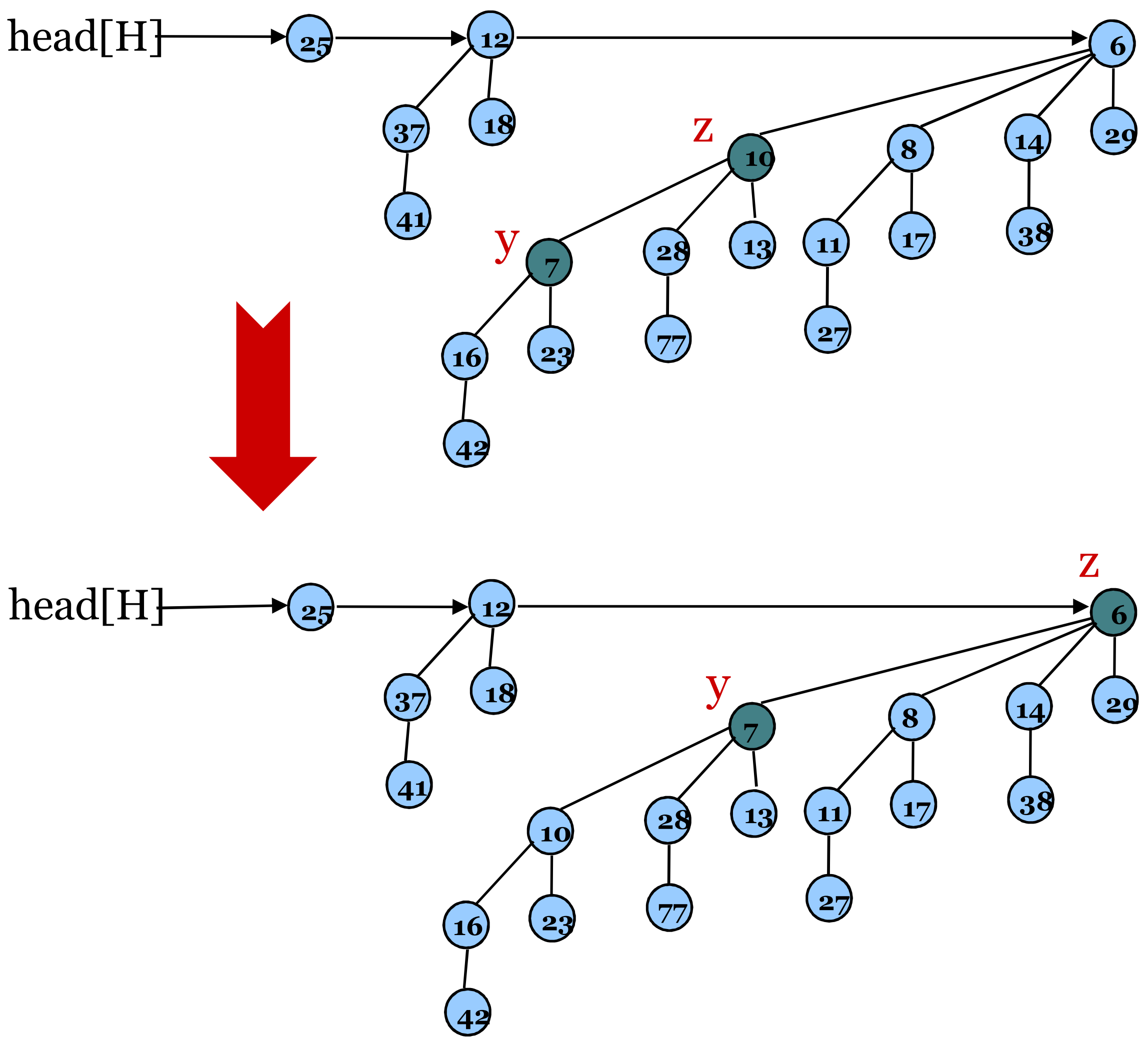
O(lg n)

Decrease-Key Example



Decrease key 26 to 7




head[H]
25
12
6
37
18
z
10
8
14
29
41
y
7
28
13
11
17
38
16
23
77
27
42
head[H]
25
12
z
6
37
18
y
7
8
14
29
41
10
28
13
11
17
38
16
23
77
27
42

DELETE a node

BINOMIAL-HEAP-DELETE(H, x)

1. BINOMIAL-HEAP-DECREASE-KEY(H, x, $-\infty$);
2. BINOMIAL-HEAP-EXTRACT-MIN(H)

Time is O(lg n)