# Algorithms
# Chapter 19*
# Binomial Heaps

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

# Outline

- **Binomial trees and binomial heaps**
- Operations on binomial heaps

# Overview

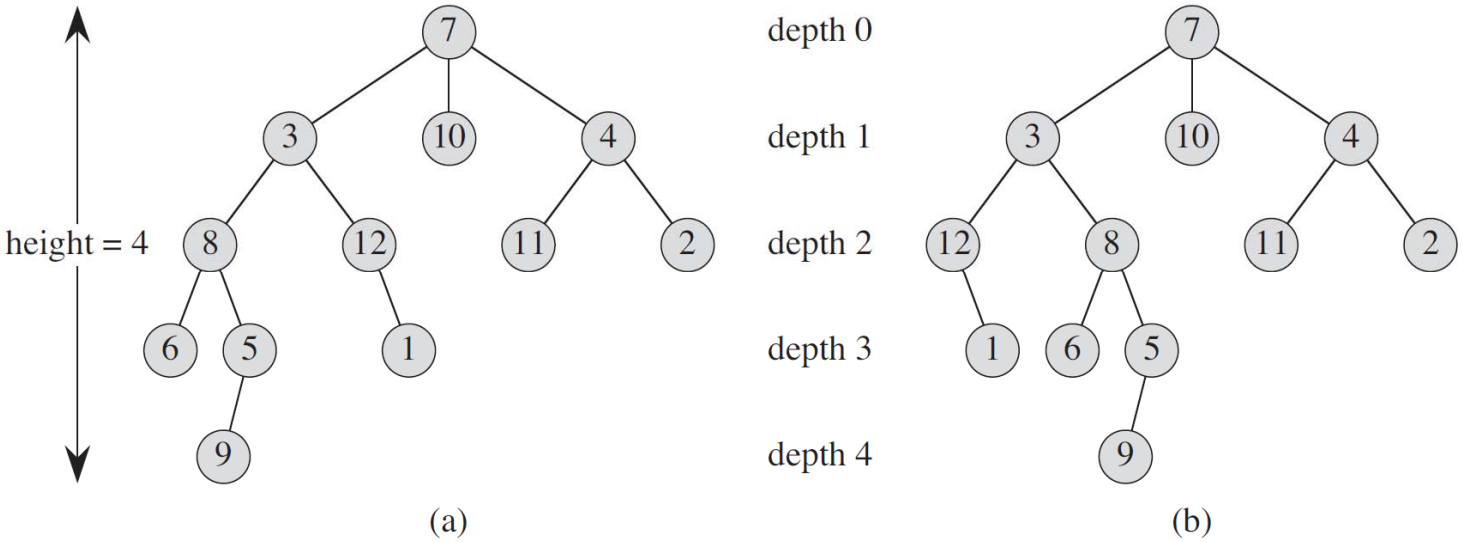
- If we don't need the UNION operation, ordinary binary heaps, as used in heapsort, work well.
- The Fibonacci heaps are amortized time bounds.
- All of the three heaps are inefficient in their support of the operation SEARCH.

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

# Rooted and ordered trees$_{1/2}$

- A **rooted tree** is a tree in which one of the vertices is distinguished from the others.
- The distinguished vertex is called the **root** of the tree.
- An **ordered tree** is a rooted tree in which the children of each node are ordered.
- That is, if a node has *k* children, then there is a first child, a second child,..., and a *k*th child.

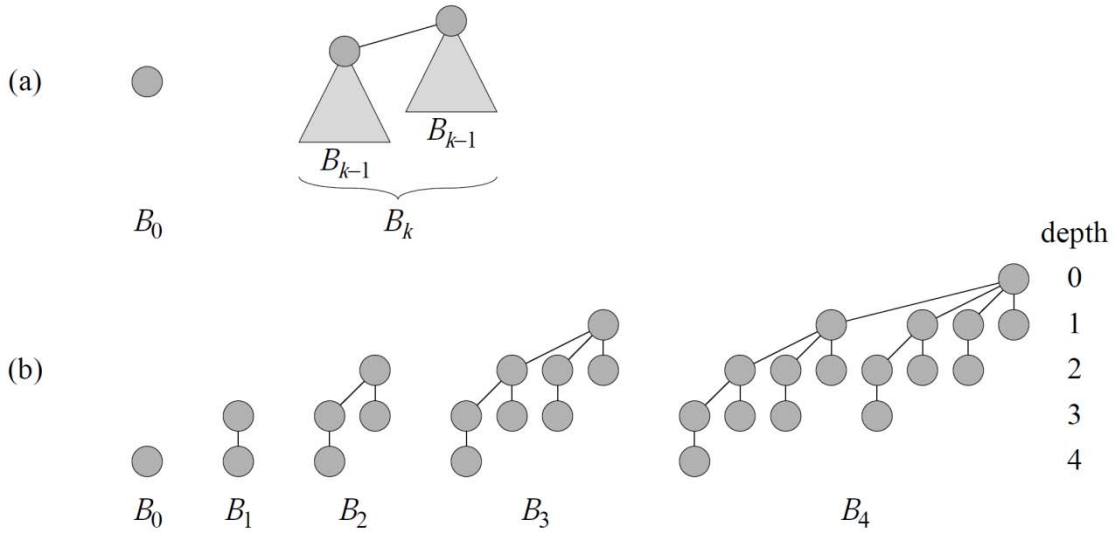
# Rooted and ordered trees$_{2/2}$




- The above two trees are different when considered to be ordered trees, but the same when considered to be just rooted trees.

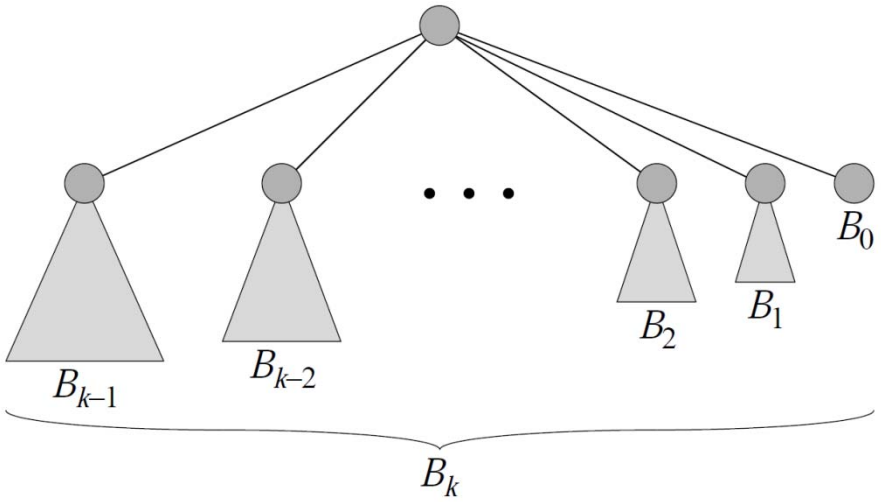
# Binomial trees$_{1/2}$

- The **binomial tree** $B_k$ is an ordered tree defined recursively as follows.
  - the binomial tree $B_0$ consists of a single node.
  - $B_k$ consists of two $B_{k-1}$ that are **linked** together: the root of one is the leftmost child of the root of the other.



# Binomial trees$_{2/2}$

(c)




Another way of looking at the binomial tree $B_k$.

# Properties of binomial trees$_{1/3}$

- **Lemma 19.1: (Properties of binomial trees)**
  For the binomial tree $B_k$,
  1. there are $2^k$ nodes,
  2. the height of the tree is $k$,
  3. there are exactly $\binom{k}{i}$ nodes at depth $i$ for $i = 0, 1, \ldots, k$,
  4. the root has degree $k$, which is the largest, and
  5. if the children of the root are numbered from left to right by $k - 1, k - 2, \ldots, 0$, child $i$ is the root of a subtree $B_i$.

The term "binomial tree" comes from.

**Proof:** By induction on $k$.

- Each property holds for $B_0$ is trivial.
- Assume that the lemma holds for $B_{k-1}$.

# Properties of binomial trees$_{2/3}$

- 1. There are $2^k$ nodes.
  - $B_k$ = two copies of $B_{k-1}$, and so $B_k$ has $2^{k-1} + 2^{k-1} = 2^k$ nodes.
- 2. The height of the tree is $k$.
  - Two copies of $B_{k-1}$ are linked to form $B_k$.
  - Maximum depth in $B_k$ = Maximum depth in $B_{k-1}$ + 1.
  - By the inductive hypothesis, this maximum depth is $(k-1) + 1 = k$.
- 3. There are exactly $\binom{k}{i}$ nodes at depth $i$ for $i = 0, 1, \ldots, k$.
  - Let $D(k, i)$ be the number of nodes at depth $i$ of binomial tree $B_k$.

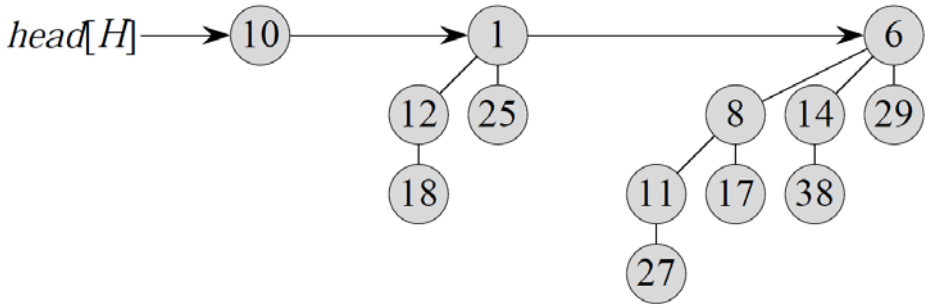
$$
\begin{aligned}
D(k,i) &= D(k-1,i) + D(k-1,i-1) \\
&= \binom{k-1}{i} + \binom{k-1}{i-1} && \text{(by the inductive hypothesis)} \\
&= \binom{k}{i} && \text{(by Exercise C.1-7)} .
\end{aligned}
$$

# Properties of binomial trees$_{3/3}$

- 4. The root has degree $k$, which is the largest.
  - The only node with greater degree in $B_k$ than in $B_{k-1}$ is the root, which has one more child than in $B_{k-1}$.
  - Since the root of $B_{k-1}$ has degree $k-1$, the root of $B_k$ has degree $k$.
- 5. If the children of the root are numbered from left to right by $k-1, k-2, \ldots, 0$, child $i$ is the root of a subtree $B_i$.
  - By the inductive hypothesis, the children of the root of $B_{k-1}$ are roots of $B_{k-2}, B_{k-3}, \ldots, B_0$.
  - When $B_{k-1}$ is linked to $B_{k-1}$, the children of the resulting root are roots of $B_{k-1}, B_{k-2}, \ldots, B_0$.
- **Corollary 19.2** The maximum degree of any node in an $n$-node binomial tree is $\lg n$. (From properties 1 and 4)

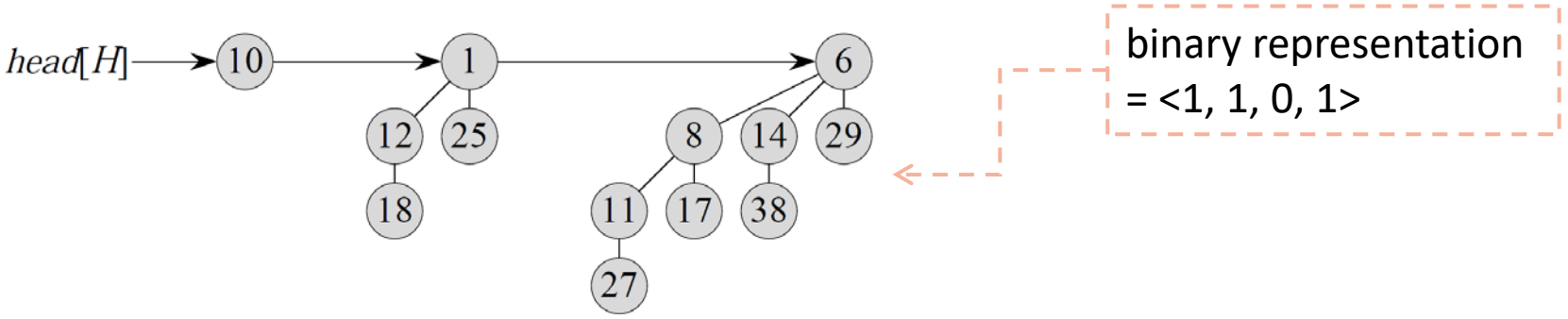
# Binomial heaps

- A **binomial heap** $H$ is a set of binomial trees that satisfies the following **binomial-heap properties**.
  1. Each binomial tree in $H$ is **min-heap ordered**:
     $key(x) \geq key(p(x))$.
  2. For any nonnegative integer $k$, there is at most one binomial tree in $H$ whose root has degree $k$.



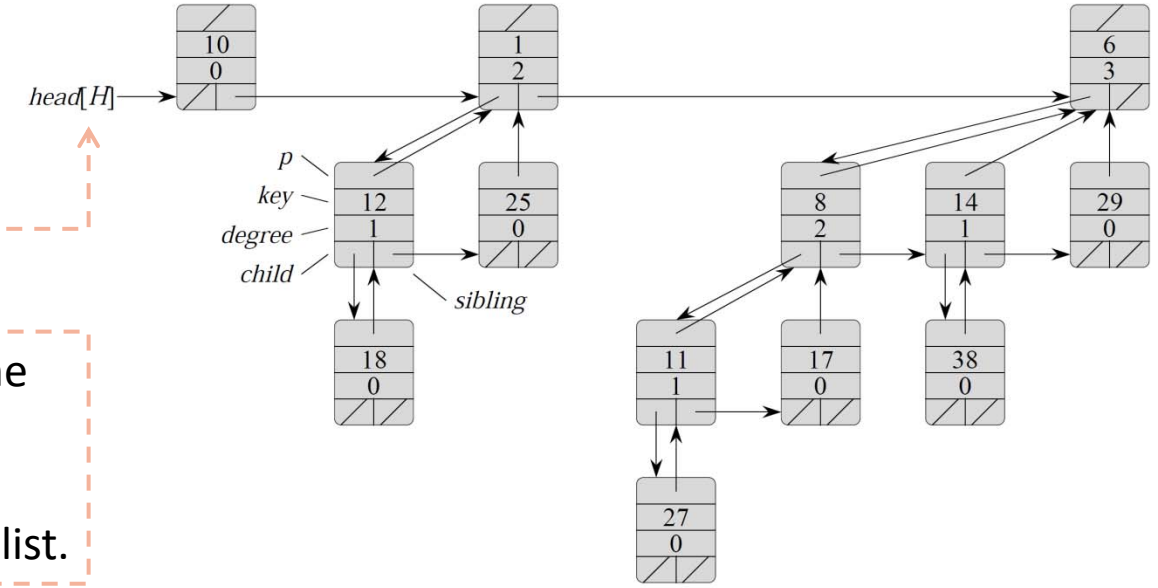
# Observations

- The first property tells us that the root of a min-heap-ordered tree contains the smallest key in the tree.
- The second property implies that an $n$-node binomial heap $H$ consists of at most $\lfloor \lg n \rfloor + 1$ binomial trees.
  - the binary representation of $n$ has $\lfloor \lg n \rfloor + 1$ bits, say $\langle b_{\lfloor \lg n \rfloor}, b_{\lfloor \lg n \rfloor - 1}, \ldots, b_0 \rangle$, so that $n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$.
  - By property 1 of Lemma 19.1, binomial tree $B_i$ appears in $H$ if and only if bit $b_i = 1$.



# Representing binomial heaps

- In a binomial heap:
  - binomial tree is stored in the left-child, right-sibling representation.
  - *key*[*x*]: key; *p*[*x*]: parent; *child*[*x*]: leftmost children.
  - *sibling*[*x*]: immediately right sibling;
  - *degree*[*x*]: the number of children.




The degrees of the roots strictly increase as we traverse the root list.

# Outline

- Binomial trees and binomial heaps
- **Operations on binomial heaps**

# Operations on binomial heaps

- MAKE-BINOMIAL-HEAP():
  - Allocate and return an object $H$, where $head[H]$ = NIL.
  - Running time = $\Theta(1)$.

- BINOMIAL-HEAP-MINIMUM($H$):
  - Since a binomial heap is min-heap-ordered, the minimum key must reside in a root node.
  - At most $\lfloor \lg n \rfloor + 1$ roots to check.
  - Running time = $O(\lg n)$.

BINOMIAL-HEAP-MINIMUM($H$)

```
1.  y ← NIL
2.  x ← head[H]
3.  min ← ∞
4.  while x ≠ NIL
5.      do if key[x] < min
6.             then min ← key[x]
7.                  y ← x
8.                  x ← sibling[x]
9.  return y
```

# Operations on binomial heaps

- BINOMIAL-LINK($y$, $z$):
  - $B_{k-1}$ rooted at $y$ + $B_{k-1}$ rooted at $z$ ➔ $B_k$ rooted at $z$.
  - Running time = $\Theta(1)$.

BINOMIAL-LINK($y$, $z$)

1. $p[y] \leftarrow z$
2. $sibling[y] \leftarrow child[z]$
3. $child[z] \leftarrow y$
4. $degree[z] \leftarrow degree[z] + 1$

- BINOMIAL-HEAP-MERGE($H_1$, $H_2$):
  - Merges the root lists of $H_1$ and $H_2$ into a single linked list that is sorted by degree into monotonically increasing order.
  - Pseudocode is left as Exercise 19.2-1.

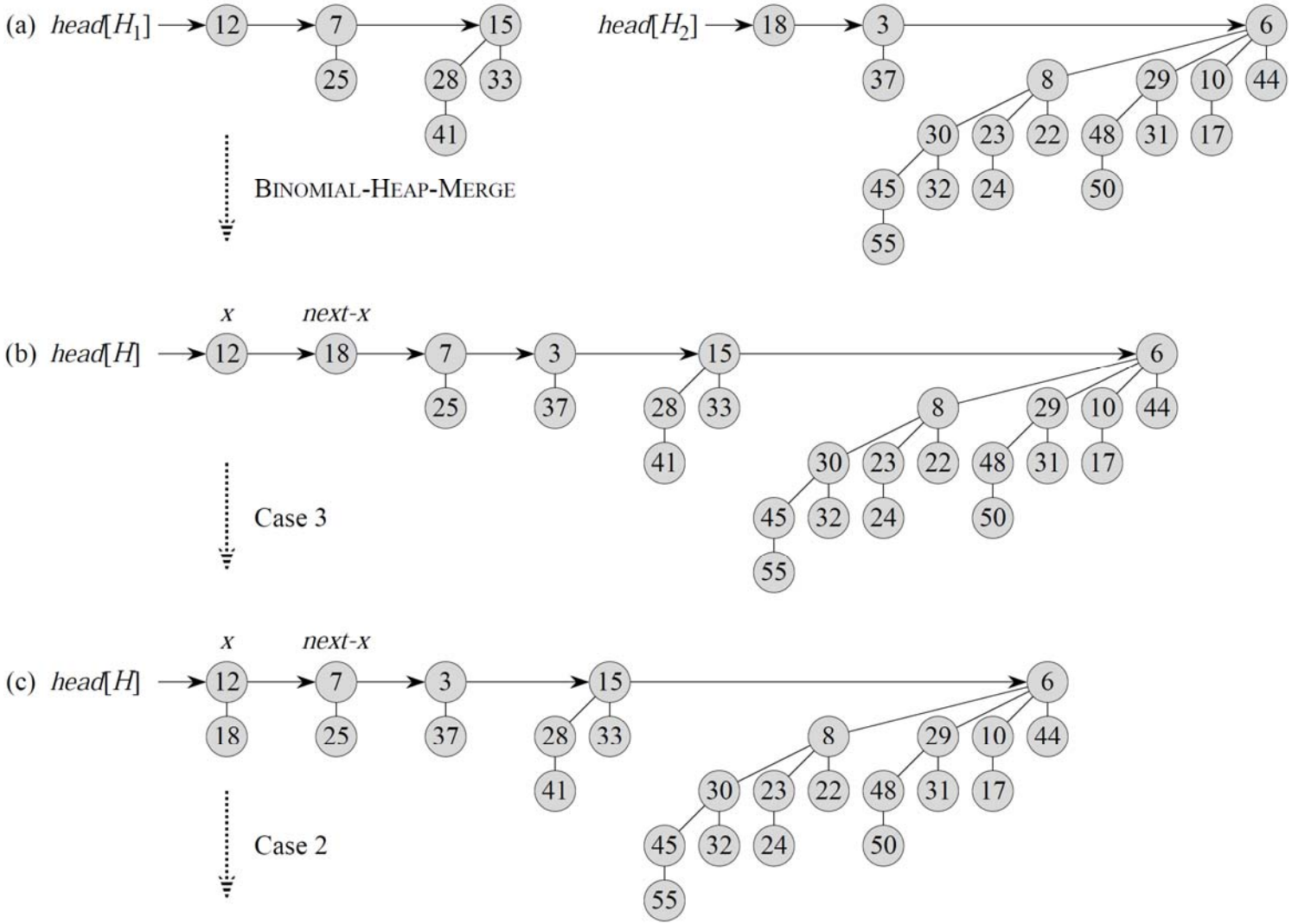
# Uniting two binomial heaps

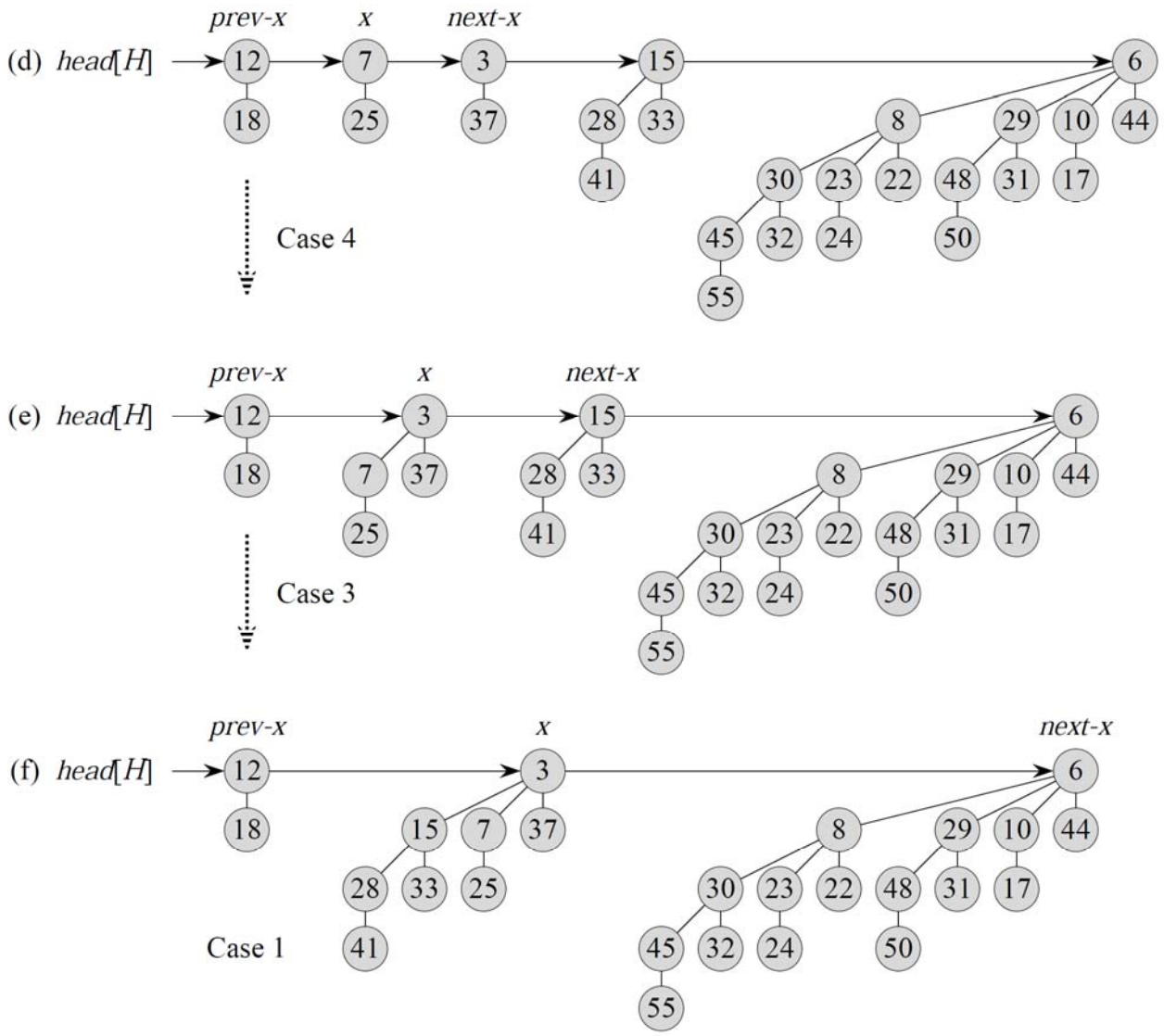
- BINOMIAL-HEAP-UNION($H_1$, $H_2$):
  - Phase 1: merge the root lists of $H_1$ and $H_2$ into a single linked list $H$ in monotonically increasing order.
  - Phase 2: link roots of equal degree until at most one root remains of each degree.
  - Running time = $O(\lg n)$.
- Phase 1:
  - Running time = $O(\lg n_1) + O(\lg n_2)$.

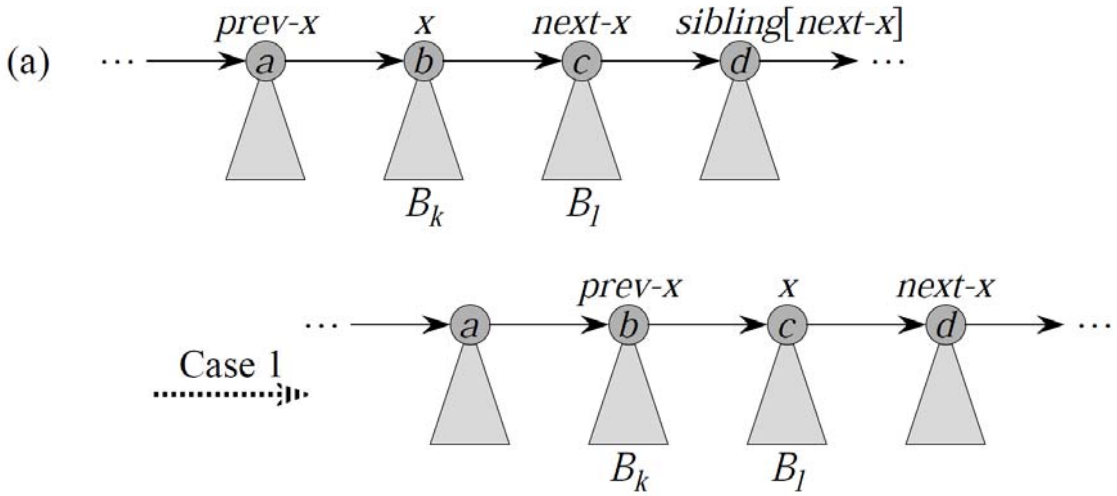
$H_1$ : $n_1$ nodes;
$H_2$ : $n_2$ nodes.

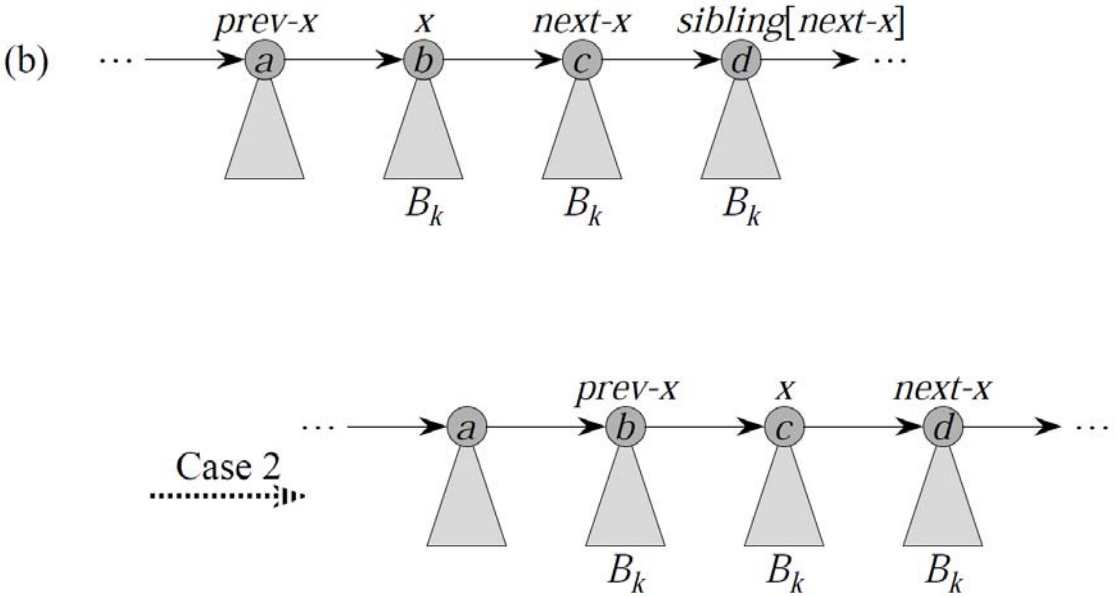
- Phase 2:
  - Each iteration of the **while** loop takes $O(1)$ time.
  - There are at most $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2$ iterations .
    - Each iteration either advances the pointers one position or removes a root.
  - Running time = $O(\lg n_1) + O(\lg n_2)$.

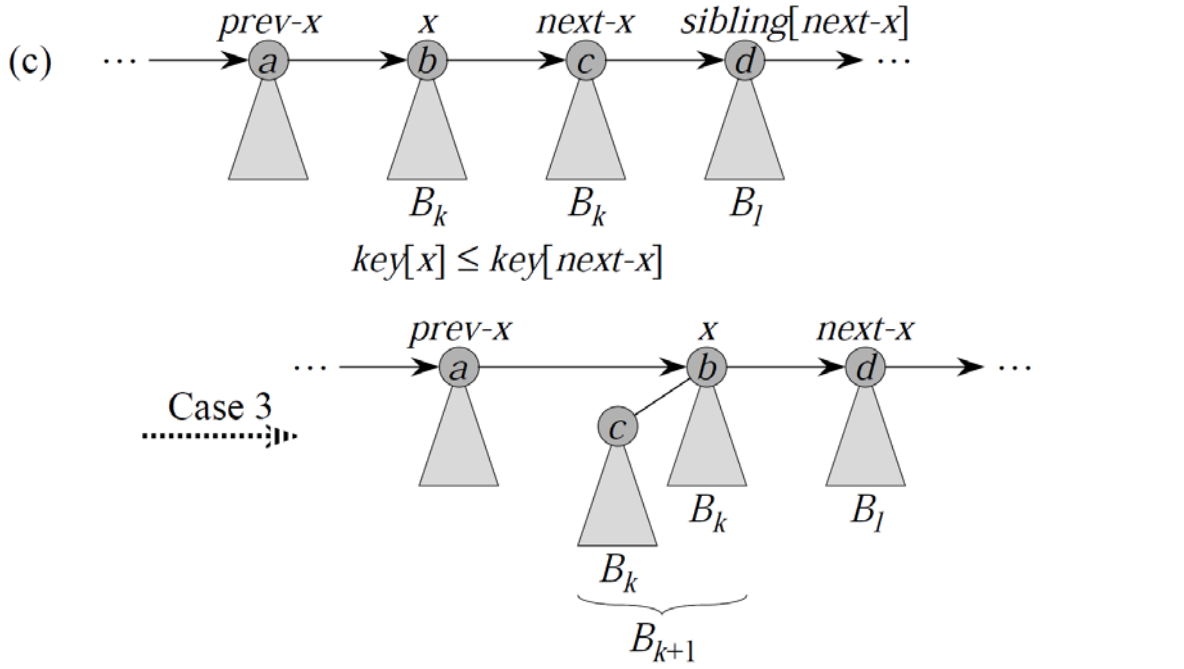
```
BINOMIAL-HEAP-UNION(H1, H2)
1.  H ← MAKE-BINOMIAL-HEAP()
2.  head[H] ← BINOMIAL-HEAP-MERGE(H1, H2)
3.  free the objects H1 and H2 but not the lists they point to
4.  if head[H] = NIL
5.      then return H
6.  prev-x ← NIL
7.  x ← head[H]
8.  next-x ← sibling[x]
9.  while next-x ≠ NIL
10.     do if (degree[x] ≠ degree[next-x]) or
              (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
11.         then prev-x ← x                                  ▷ Cases 1 and 2
12.              x ← next-x                                  ▷ Cases 1 and 2
13.         else if key[x] ≤ key[next-x]
14.                 then sibling[x] ← sibling[next-x]        ▷ Case 3
15.                      BINOMIAL-LINK(next-x, x)            ▷ Case 3
16.                 else if prev-x = NIL                     ▷ Case 4
17.                         then head[H] ← next-x            ▷ Case 4
18.                         else sibling[prev-x] ← next-x    ▷ Case 4
19.                      BINOMIAL-LINK(x, next-x)            ▷ Case 4
20.                      x ← next-x                          ▷ Case 4
21.         next-x ← sibling[x]
22. return H
```

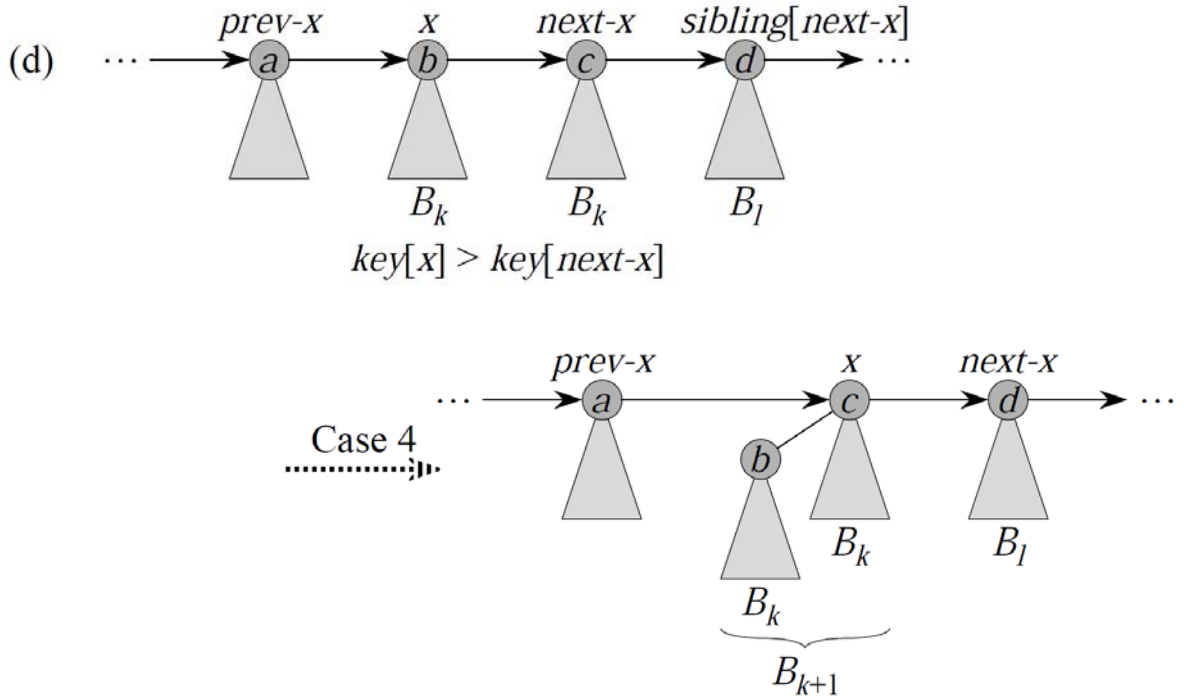

(a) head[H1]
12
7
15
25
28
33
41
BINOMIAL-HEAP-MERGE
head[H2]
18
3
6
37
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
(b) head[H]
x
next-x
12
18
7
3
15
6
25
37
28
33
8
29
10
44
41
30
23
22
48
31
17
45
32
24
50
55
Case 3
(c) head[H]
x
next-x
12
7
3
15
6
18
25
37
28
33
8
29
10
44
41
30
23
22
48
31
17
45
32
24
50
55
Case 2


prev-x
x
next-x
(d) head[H]
Case 4
(e) head[H]
Case 3
(f) head[H]
Case 1


(a)
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
$B_k$
$B_l$


(b)
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
$B_k$
$B_k$
$B_k$


(c)
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
d
c
$B_k$
$B_l$
$B_k$
$B_{k+1}$


(d)
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
d
b
$B_k$
$B_l$
$B_k$
$B_{k+1}$

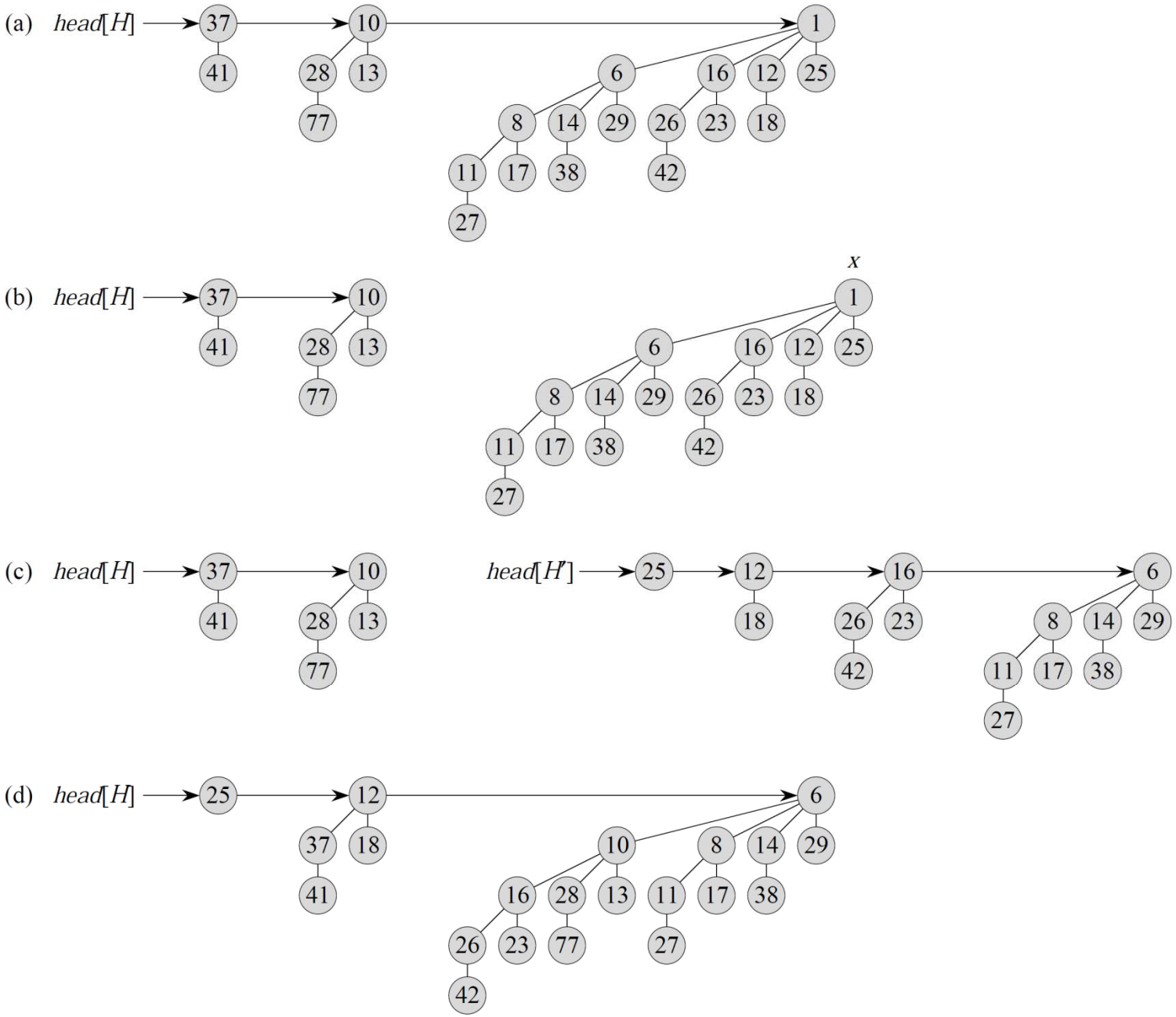
# Insert & Extract-Min

- BINOMIAL-HEAP-INSERT($H$, $x$):
  - Running time = $O(\lg n)$.

BINOMIAL-HEAP-INSERT($H$, $x$)
1. make a one-node binomial heap $H'$ containing $x$
2. $H \leftarrow$ BINOMIAL-HEAP-UNION($H$, $H'$)

- BINOMIAL-HEAP-EXTRACT-MIN($H$):
  - Running time = $O(\lg n)$.

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. $H' \leftarrow$ 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. $H \leftarrow$ BINOMIAL-HEAP-UNION($H$, $H'$)
5. **return** $x$


(a) head[H]
37
10
1
41
28
13
6
16
12
25
77
8
14
29
26
23
18
11
17
38
42
27
(b) head[H]
37
10
x
1
41
28
13
6
16
12
25
77
8
14
29
26
23
18
11
17
38
42
27
(c) head[H]
37
10
head[H']
25
12
16
6
41
28
13
18
26
23
8
14
29
77
42
11
17
38
27
(d) head[H]
25
12
6
37
18
10
8
14
29
41
16
28
13
11
17
38
26
23
77
27
42

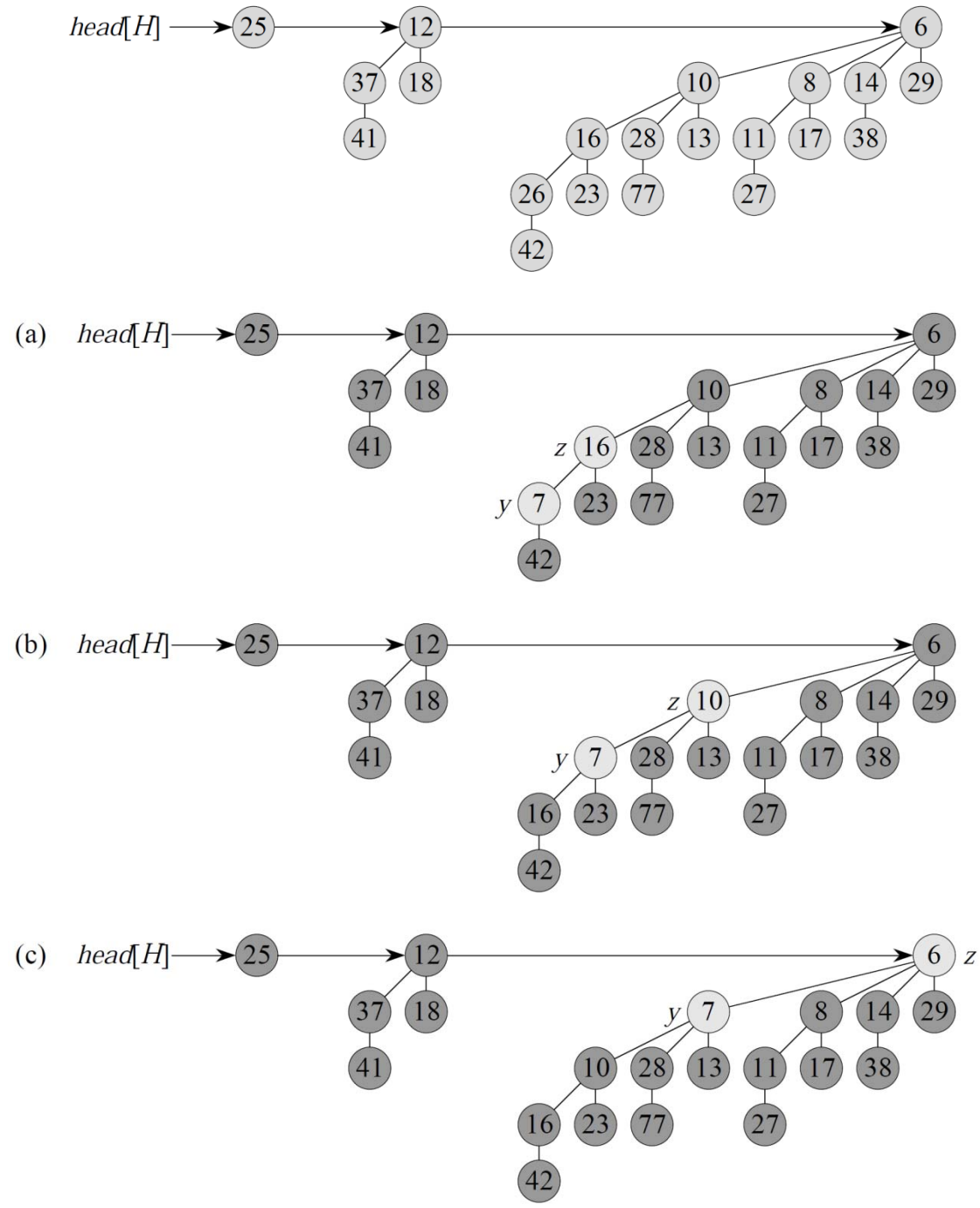
# Decrease-Key & Delete

- BINOMIAL-HEAP-DECREASE-KEY($H$, $x$, $k$):
  - Running time = $O$(depth of $x$) = $O(\lg n)$.

BINOMIAL-HEAP-DECREASE-KEY($H$, $x$, $k$)
1. **if** $k > key[x]$
2. **then error** "new key is greater than current key“
3. $key[x] \leftarrow k$
4. $y \leftarrow x$
5. $z \leftarrow p[y]$
6. **while** $z \neq$ NIL and $key[y] < key[z]$
7. **do** exchange $key[y] \leftrightarrow key[z]$
8. $y \leftarrow z$
9. $z \leftarrow p[y]$

- BINOMIAL-HEAP-DELETE($H$):
  - Running time = $O(\lg n)$.

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


head[H]
25
12
6
37
18
10
8
14
29
41
16
28
13
11
17
38
26
23
77
27
42
(a)
head[H]
z
y
7
(b)
head[H]
z
y
7
(c)
head[H]
z
y
7