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 注意：Fibonacci是 amortized analysis

- If we don't need the UNION operation, ordinary binary heaps, as used in heapsort, work well. 不考慮 union → binary heap不錯☺
- The Fibonacci heaps are amortized time bounds.
- All of the three heaps are inefficient in their support of the operation SEARCH. search時都不是很有效率

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$_{1/2}$

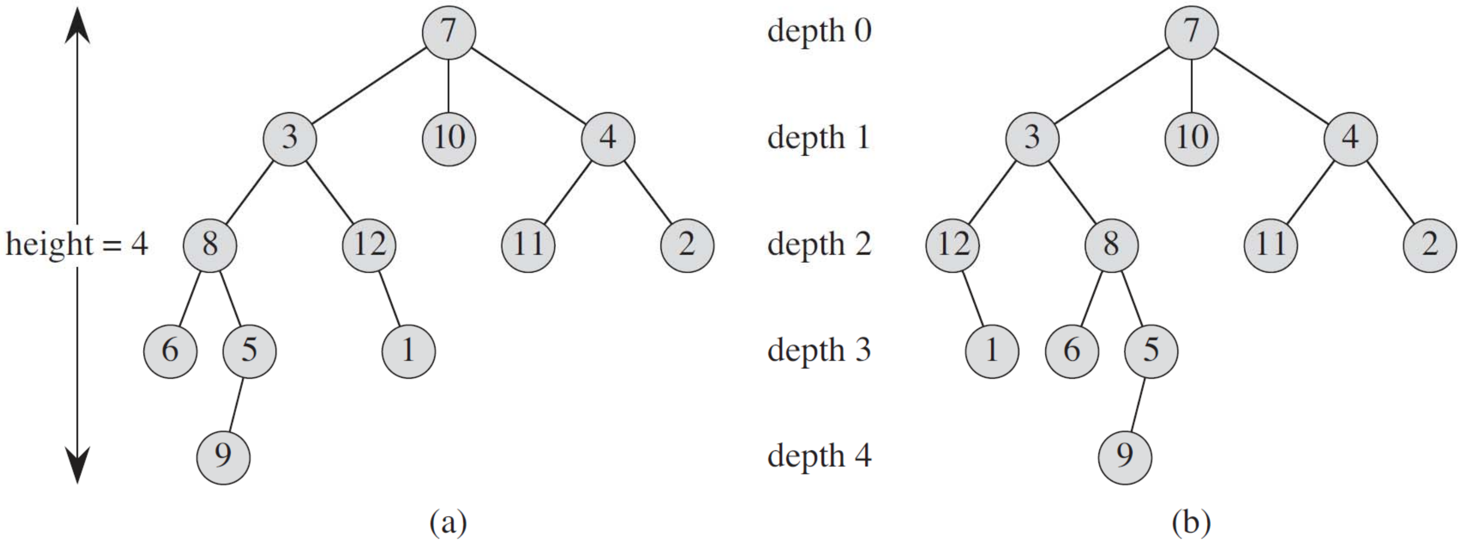
有個特別的點: root

- 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. root 的兒子有排序 ex: 第1個兒子

  2

  第k個兒子
- 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$_{2/2}$

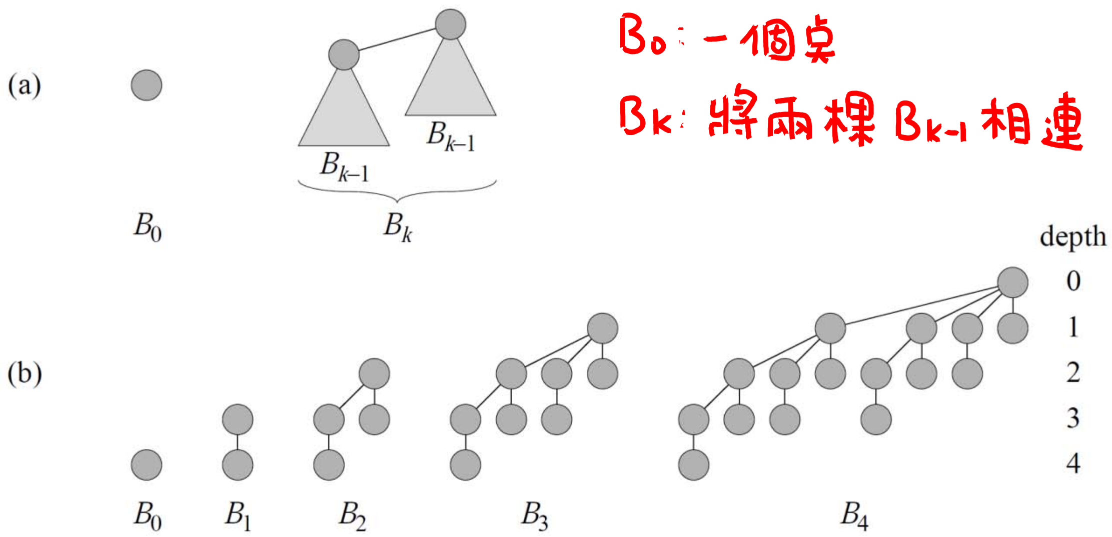


- The above two trees are different when considered to be ordered trees, but the same when considered to be just rooted trees.

以 ordered tree 角度看：不同
以 rooted tree 角度看：相同

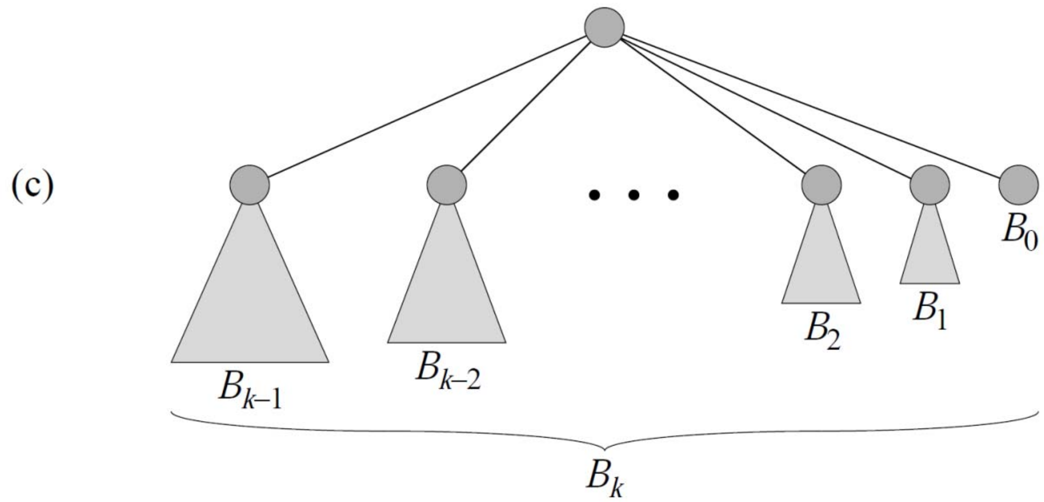
Binomial trees$_{1/2}$

- 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 $B_{k\text{-}1}$ that are **linked** together: the root of one is the leftmost child of the root of the other.



B_0: 一個点
B_k: 將兩棵 B_{k-1} 相連

Binomial trees$_{2/2}$



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,

  The term "binomial tree" comes from.

  1. there are 2^k nodes,
  2. the height of the tree is k, 樹的高度為k
  3. there are exactly $\binom{k}{i}$ nodes at depth i for $i = 0, 1, ..., k$, 深度為i有$\binom{k}{i}$個點
  4. the root has degree k, which is the largest, and root 的 degree為k為最大
  5. if the children of the root are numbered from left to right by $k - 1, k - 2, ..., 0$, child i is the root of a subtree B_i. 圖c

Proof: By induction on k. 歸納法

- Each property holds for B_0 is trivial. 以上5個性質在B_0時都成立
- Assume that the lemma holds for B_{k-1}. 假設B_{k-1}成立

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

- 1. There are 2^k nodes. B_k = 2 個 B_{k-1}
  - 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.

  B_k 比 B_{k-1} 高一層

  $(k-1)+1=k$
  - 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 $i = 0, 1, ..., k$.
  - Let $D(k, i)$ be the number of nodes at depth i of binomial tree B_k.

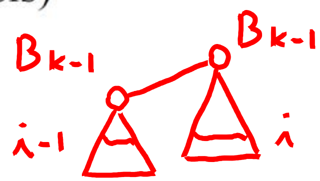
$$
\begin{aligned}
D(k, i) &= D(k-1, i) + D(k-1, i-1) \\
&= \binom{k-1}{i} + \binom{k-1}{i-1} && \text{(by the inductive hypothesis)} \\
&= \binom{k}{i} && \text{(by Exercise C.1-7)}\ .
\end{aligned}
$$

B_k 第 i 層的点数

= B_{k-1} 第 i 層

\+

B_{k-1} 第 $i-1$ 層



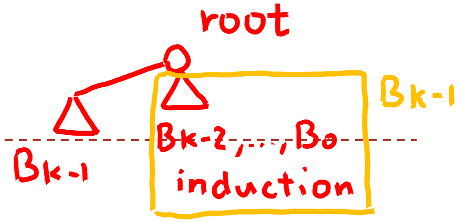
Properties of binomial trees$_{3/3}$

兩 B_{k-1} 相比，
唯一 degree 有变大的是 root
root 的 degree : $k-1+1$

- 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 $B_{k-1}, B_{k-2}, \ldots, B_0$.

n個點的 binomial tree，最大degree 為 lgn (性質1和性質4)

- **Corollary 19.2** The maximum degree of any node in an n-node binomial tree is $\lg n$. (From properties 1 and 4)



binomial heap = binomial trees + binomial heap properties

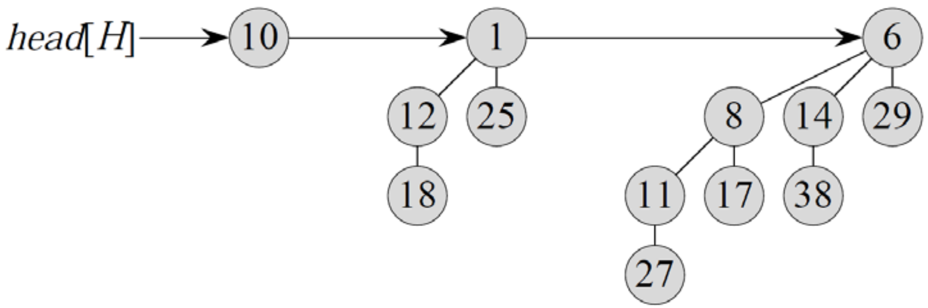
Binomial heaps

- A **binomial heap** *H* is a set of binomial trees that satisfies the following **binomial-heap properties**.

P_1 1. Each binomial tree in *H* is **min-heap ordered**:

$$key(x) \geq key(p(x)).$$

P_2 2. For any nonnegative integer *k*, there is at most one binomial tree in *H* whose root has degree *k*.



binomial heap properties

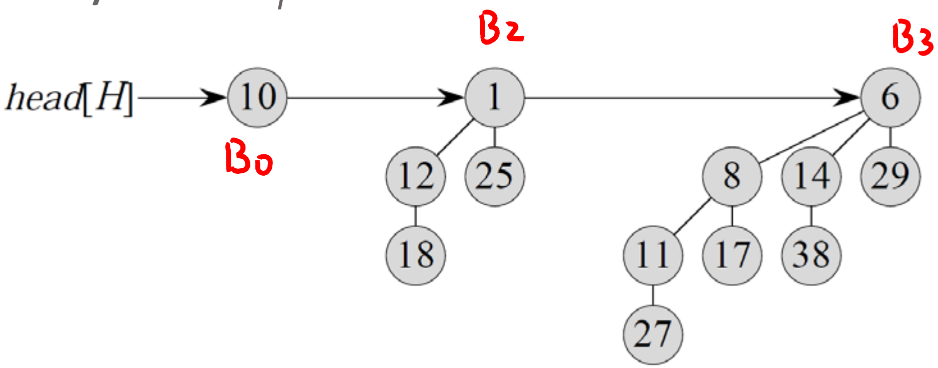
P_1: 父 ≤ 子

P_2: B_k 的 k 是唯一的

$H = \underbrace{B_5 + B_4}_{\text{不会重覆}} + B_1$

Observations ①+② ⇒ 找最小：最多檢查 $\lfloor \lg n \rfloor + 1$ 個 roots

- The first property tells us that the root of a min-heap-ordered tree contains the smallest key in the tree. P1：最小值在某一個 root
- The second property implies that an *n*-node binomial heap *H* consists of at most $\lfloor \lg n \rfloor + 1$ binomial trees. P2：binomial tree 最多有 $\lfloor \lg n \rfloor + 1$ 個
  - 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 $n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$.
    用二進位表示n，需要 $\lfloor \lg n \rfloor + 1$ bit
  - By property 1 of Lemma 19.1, binomial tree B_i appears in *H* if and only if bit $b_i = 1$.



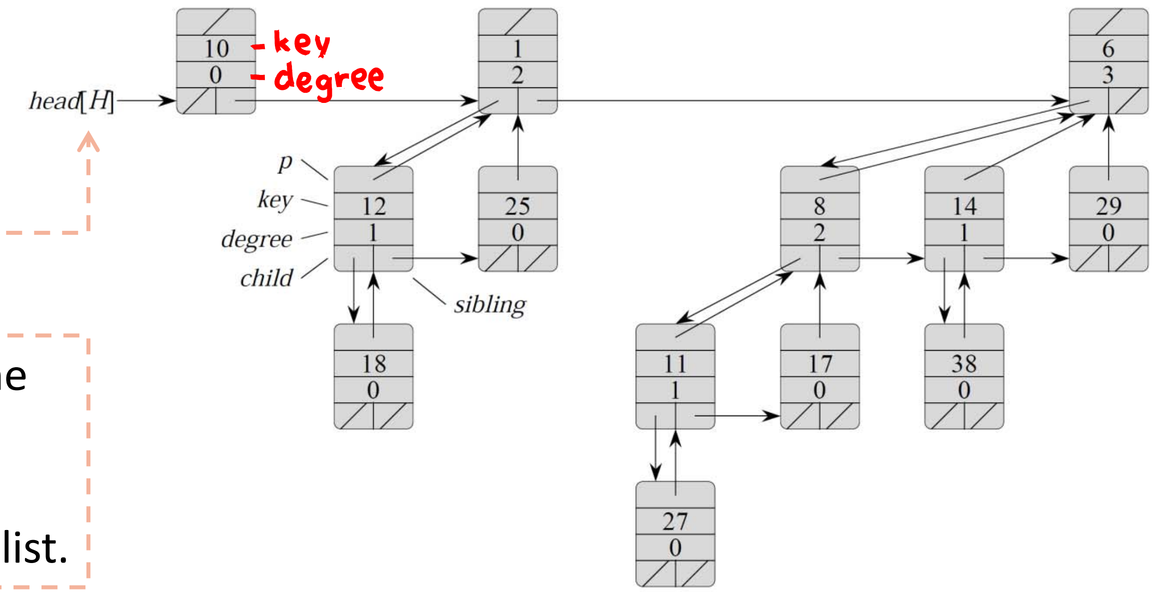
binary representation = <1, 1, 0, 1>

B3 B2 B0

用左子右弟表示 binomial tree 注意: degree由小到大

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

找最小

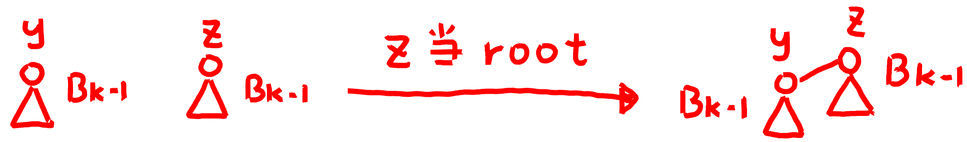
- BINOMIAL-HEAP-MINIMUM(*H*):
  - Since a binomial heap is min-heap-ordered, the minimum key must reside in a root node.
  - At most $\lfloor \lg n \rfloor + 1$ roots to check.
  - Running time = $O(\lg n)$.

```
BINOMIAL-HEAP-MINIMUM(H)
    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
```

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)
    p[y] ← z
    sibling[y] ← child[z]
    child[z] ← y
    degree[z] ← degree[z] + 1
```



- BINOMIAL-HEAP-MERGE(H_1, H_2): 将H_1和H_2合併，使degree由小到大排列
  - 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

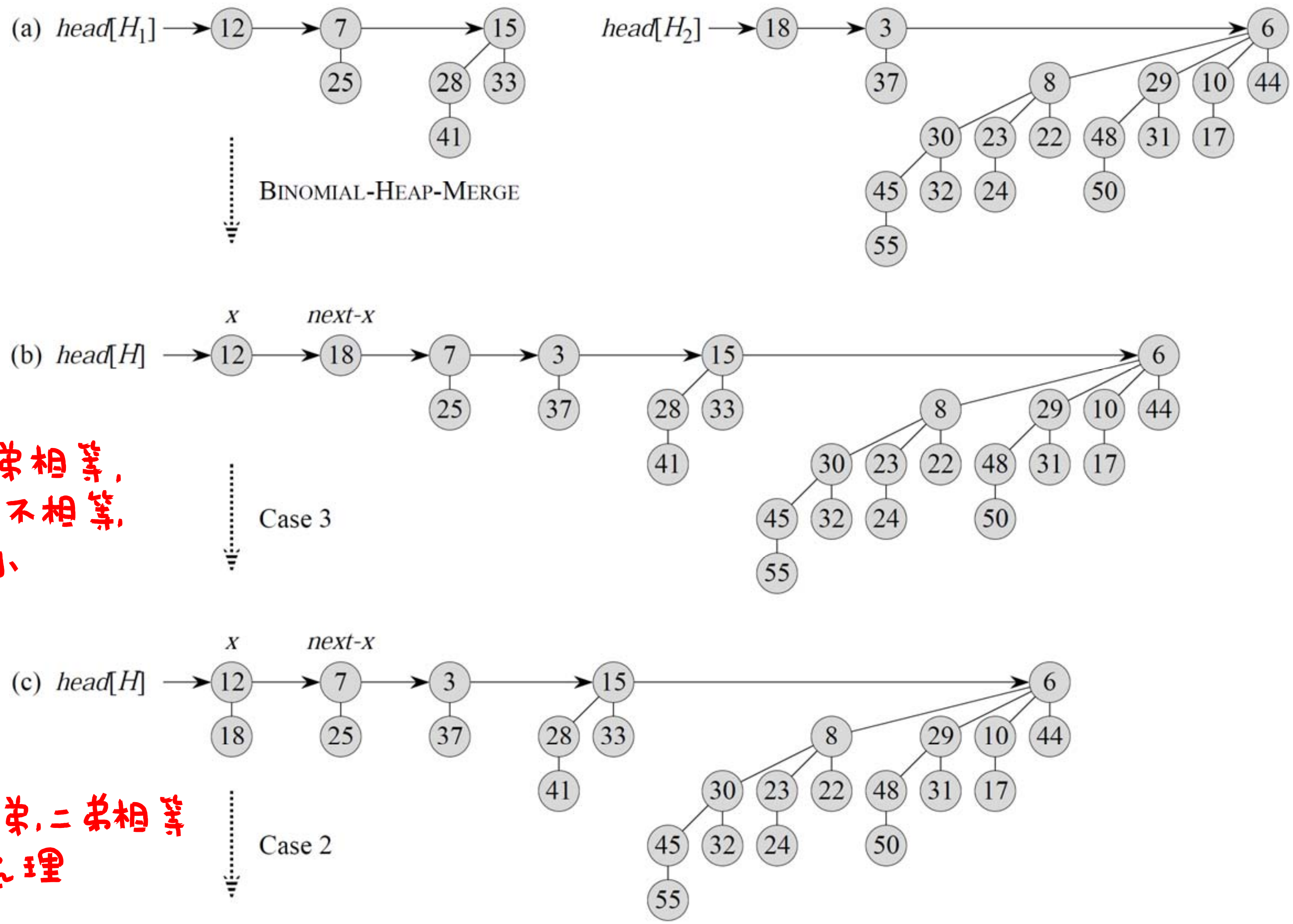
- BINOMIAL-HEAP-UNION(H_1, H_2): 第一步：先由小到大排 merge
  - Phase 1: merge the root lists of H_1 and H_2 into a single linked list H in monotonically increasing order.
  - Phase 2: link roots of equal degree until at most one root remains of each degree. 第二步：做進位
  - Running time = $O(\lg n)$.
- Phase 1:
  - 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 = $O(\lg n_1) + O(\lg n_2)$.

1 1 1 1
0 1 1 0

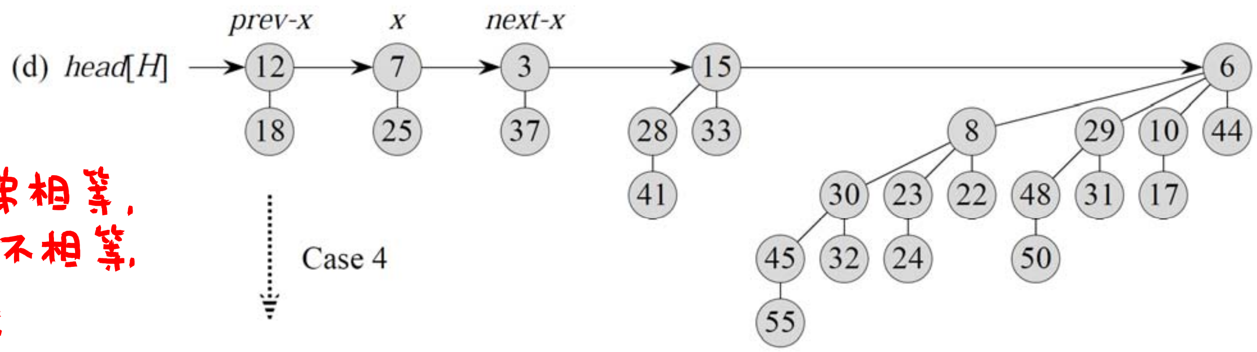
① 1 和 0，無進位
② 1 和 0，有進位
③ 1 和 1，無進位
④ 1 和 1，有進位

```
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]
    while next-x ≠ NIL                                    x和大弟不相等
          do if (degree[x] ≠ degree[next-x]) or           x和大弟二弟相等
                (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
                then prev-x ← x                 ⇒不處理          ▷ Cases 1 and 2
                     x ← next-x                                   ▷ Cases 1 and 2
                else if key[x] ≤ key[next-x]    x較小
                        then sibling[x] ← sibling[next-x]         ▷ Case 3
                             BINOMIAL-LINK(next-x, x)             ▷ Case 3
                        else if prev-x = NIL    x較大             ▷ 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
                                                 x和大弟相等，和二弟不相等。
             next-x ← sibling[x]
    return H
```

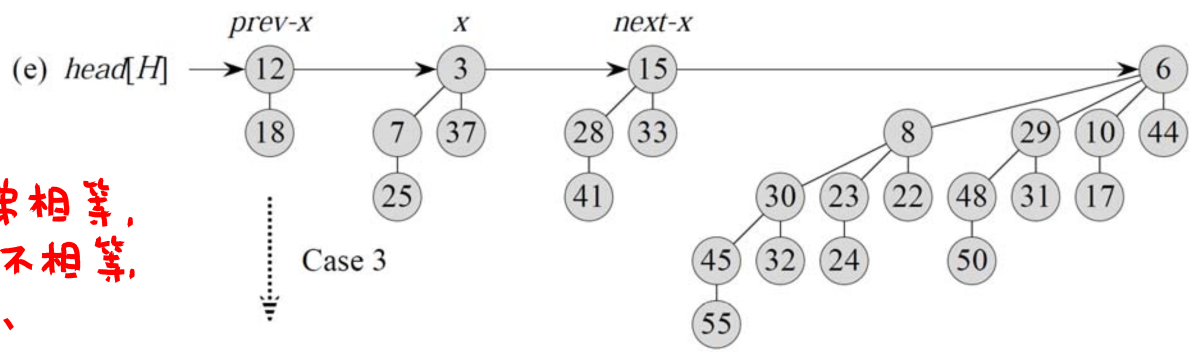


x和大弟相等，
和二弟不相等，
x較小

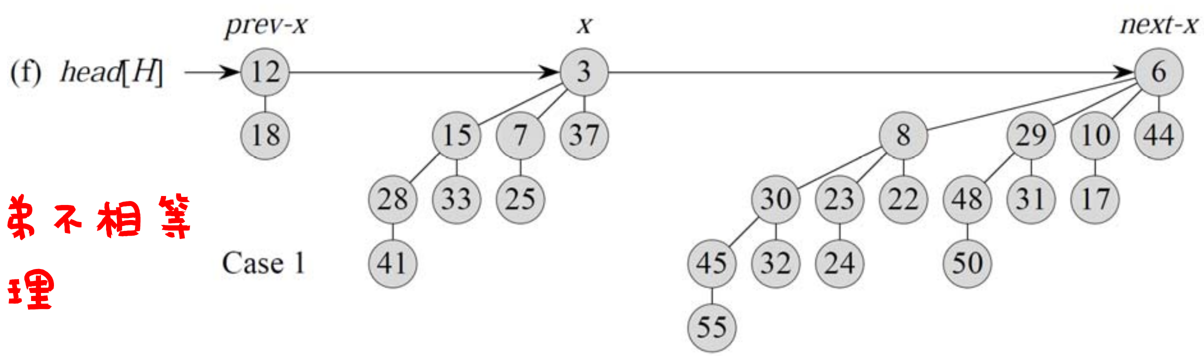
x和大弟，二弟相等
⇒不處理



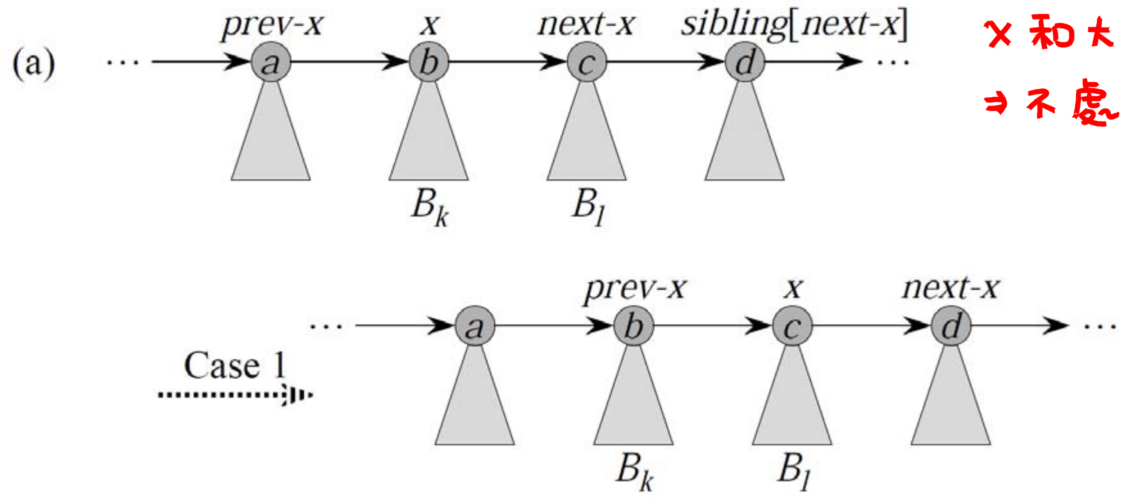
x和大弟相等,
和二弟不相等,
x較大



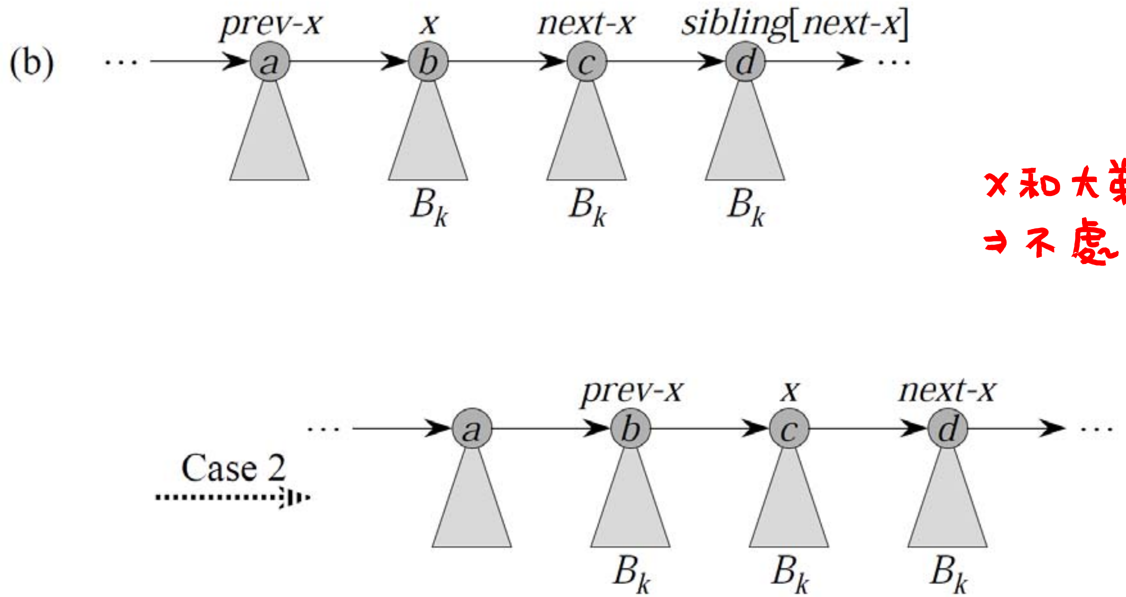
x和大弟相等,
和二弟不相等,
x較小



x和大弟不相等
⇒不處理

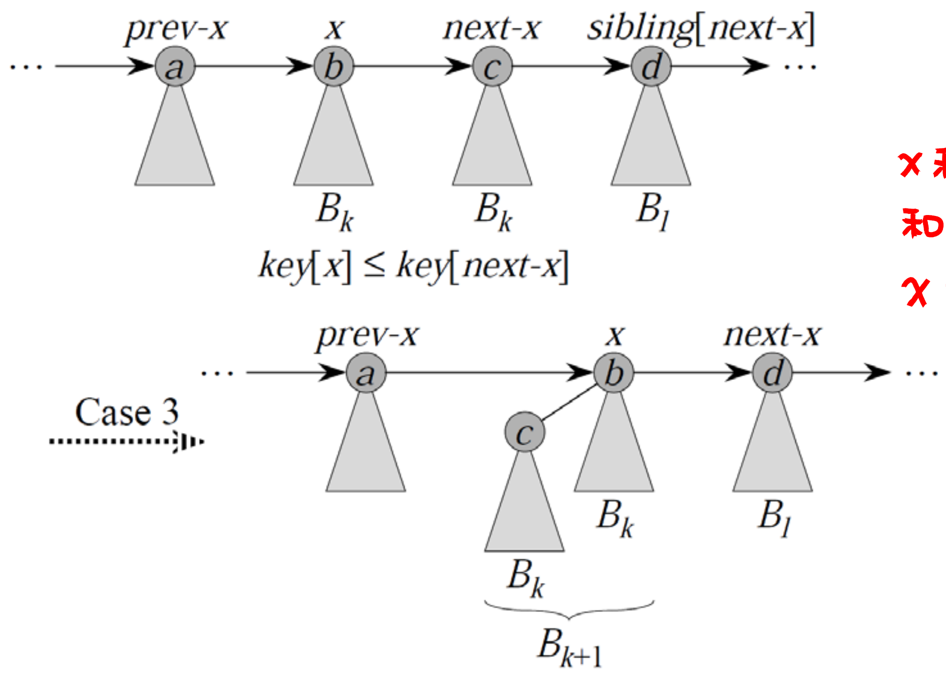


x和大弟不相等
⇒不處理



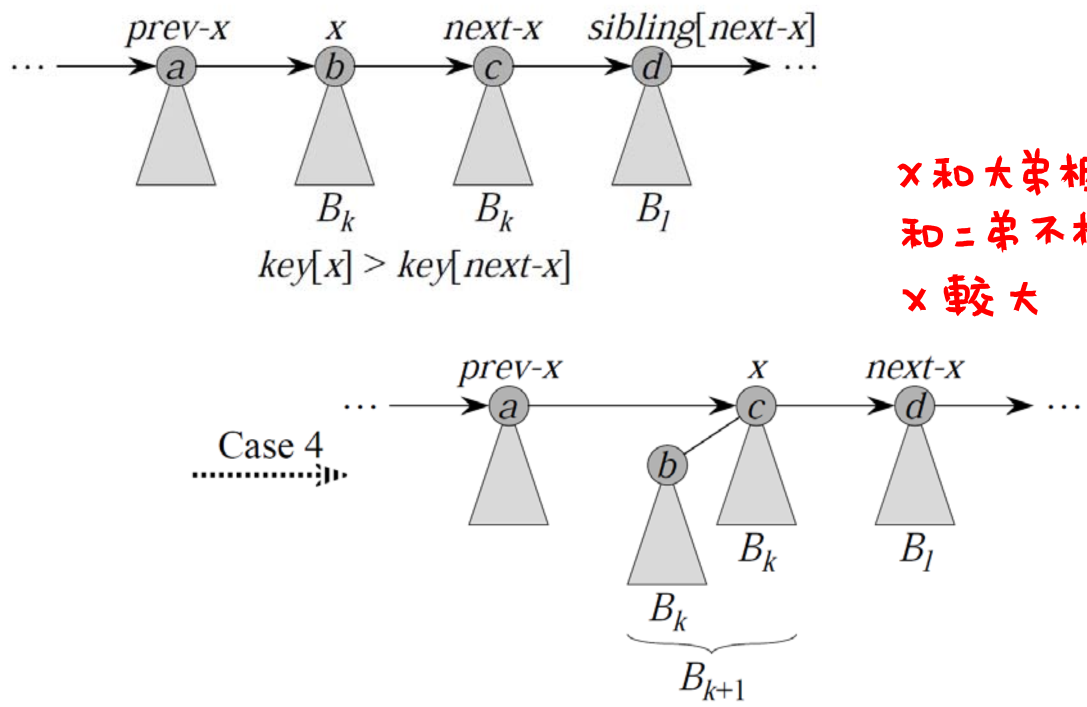
x和大弟,二弟相等
⇒不處理

(c)



x和大弟相等，
和二弟不相等，
x較小

(d)



x和大弟相等，
和二弟不相等，
x較大

Insert & Extract-Min

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

① 先做只有一個 node 的 heap H'

② 將 H' 和 H union

BINOMIAL-HEAP-INSERT(*H*, *x*)

make a one-node binomial heap *H*′ containing *x*

H ← BINOMIAL-HEAP-UNION(*H*, *H*′)

- BINOMIAL-HEAP-EXTRACT-MIN(*H*): 找最小，並將它移除
  - Running time = $O(\lg n)$.

BINOMIAL-HEAP-EXTRACT-MIN(*H*)

find the root *x* with the minimum key in the root list of *H*, and remove *x* from the root list of *H* 找最小，分割

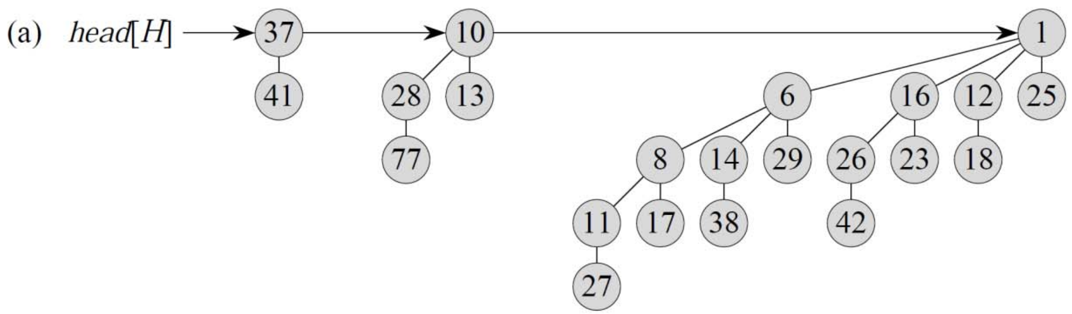
H′ ← MAKE-BINOMIAL-HEAP()

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

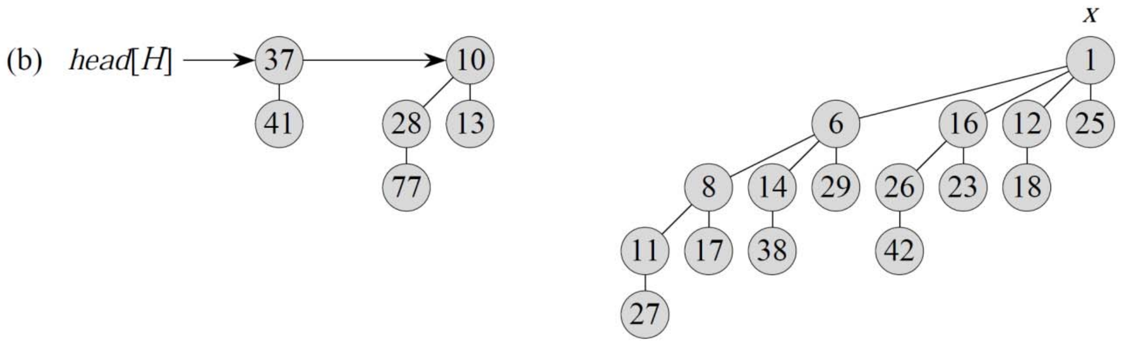
H ← BINOMIAL-HEAP-UNION(*H*, *H*′) 合併

return *x*

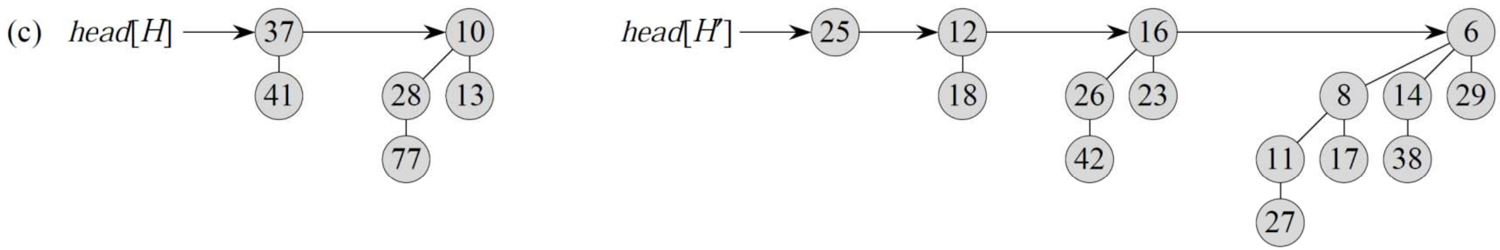
找最小


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

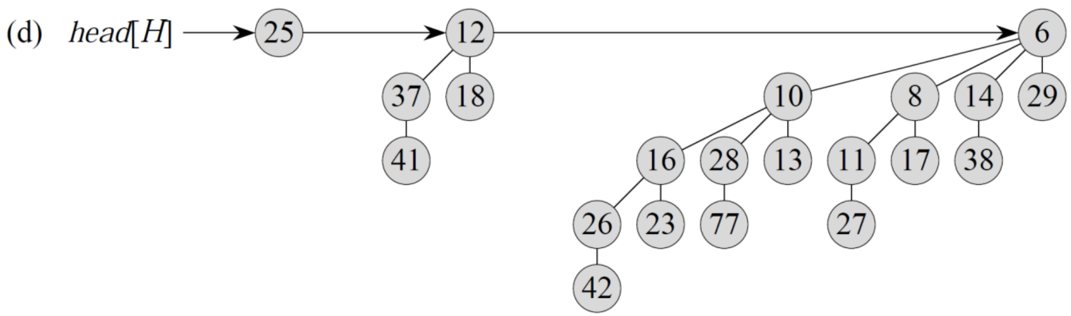
分割


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

反轉


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

合併


(d)
head[H]
25 12 37 18 41 6 10 8 14 29 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*).

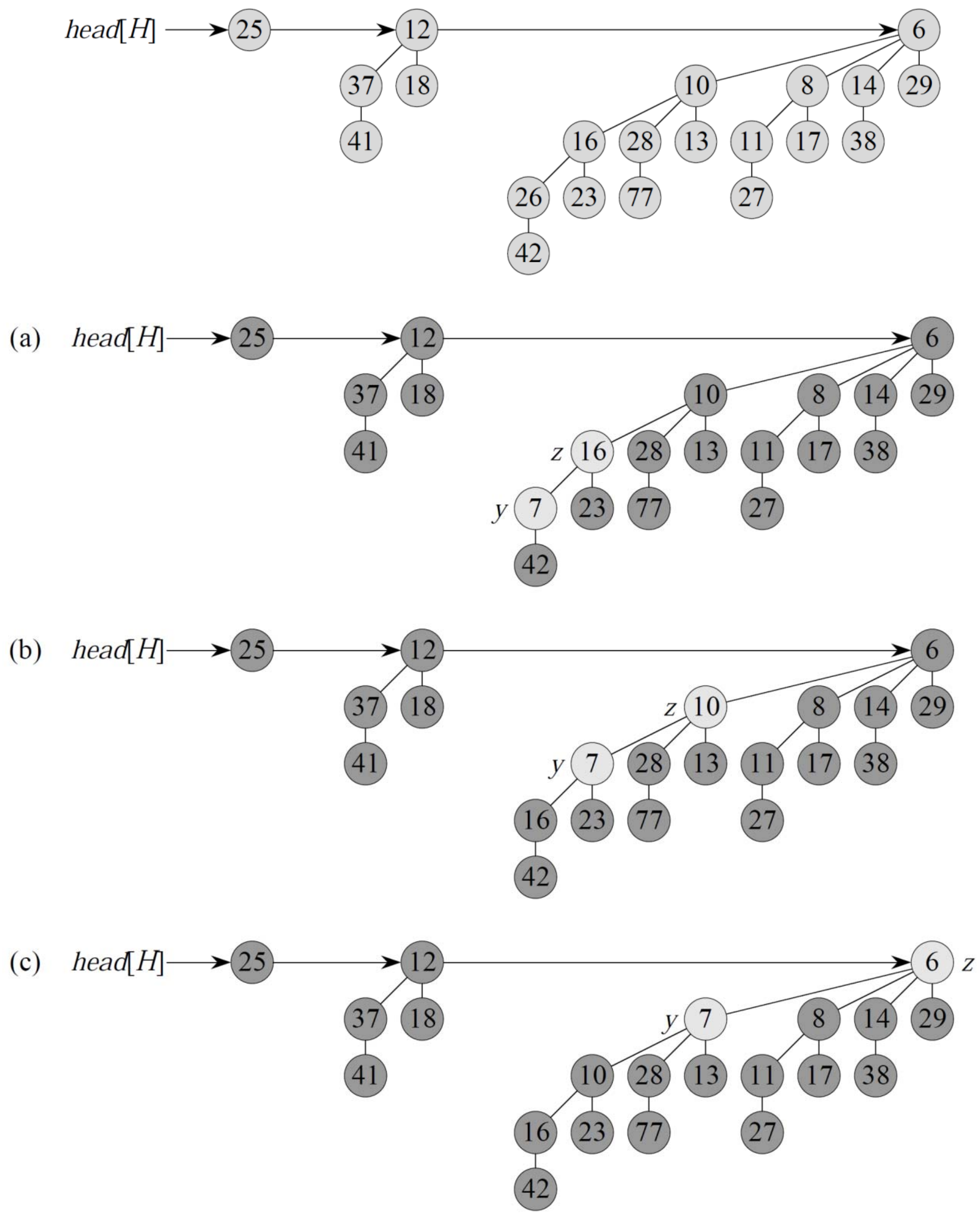
```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
    if k > key[x]
        then error "new key is greater than current key"
    key[x] ← k
    y ← x
    z ← p[y]
    while z ≠ NIL and key[y] < key[z]
        do exchange key[y] ↔ key[z]
           y ← z
           z ← p[y]
```

跟父親比，做交換

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

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

① 將x變成最小
② delete 最小


head[H]
(a)
head[H]
z
y
(b)
head[H]
z
y
(c)
head[H]
z
y