Chapter 19

Binomial Heaps

Lee, Hsiu-Hui

Ack: This presentation is based on the lecture slides from Prof. Tsai, Shi-Chun as well as various materials from the web.

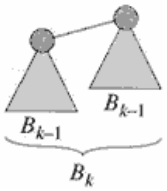
.

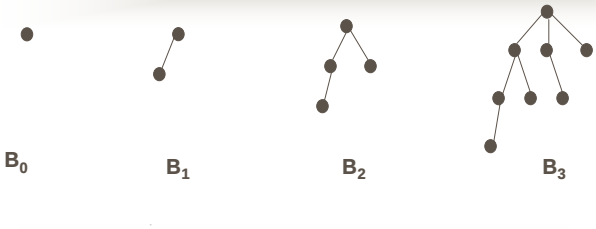
Binomial trees

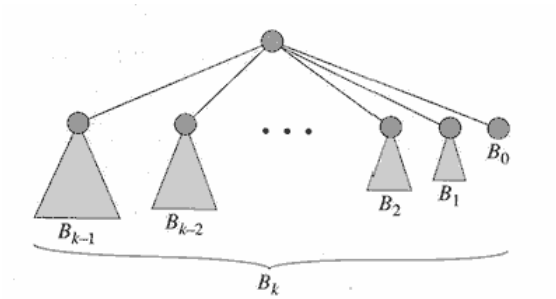
- The *binomial tree* ***B(k)*** **is an ordered tree:**

***B*(0) consists of a single node.**

***B(k)* consists of two binomial trees *B*(*k*-1) that are *linked* together: the root of one is the leftmost child of the root of the other.**







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 $i= 0,1, ..., k$
4. the root has degree k, which is greater than that of any other node; moreover 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

Proof:

1.By induction, $2^{k-1} + 2^{k-1} = 2^k$

2.By induction, $1 + (k-1) = k$

3 $\binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$, by induction

Corollary

The maximum degree of any node in an n-node binomial tree is $\lg(n)$

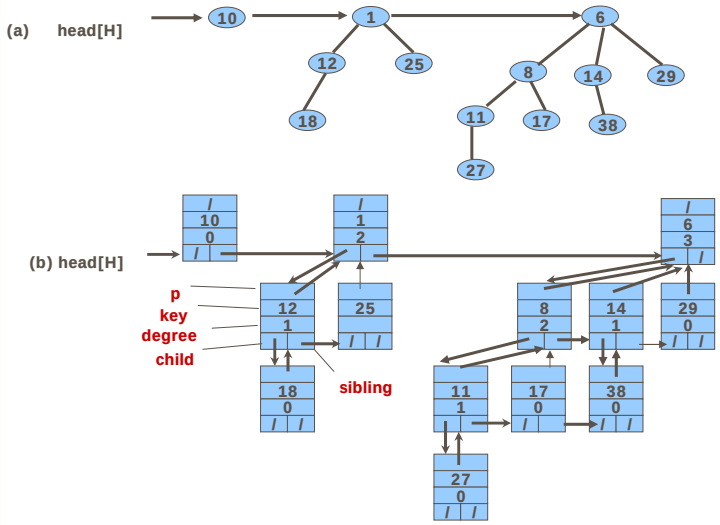
Binomial heaps

***H*: a set of binomial trees satisfying the following:**

1. Each binomial tree in *H* is heap-ordered:
   the key of a node is greater than or equal to the key of its parent
2. There is at most one binomial tree in *H* whose root has a given degree

By 2. an *n*-node binomial heap *H* consists of at most $\lfloor \lg n \rfloor + 1$ binomial trees

Representation of binomial heaps



Operations on binomial heaps

- **Creating a new binomial heap**

$\text{head}[H] = \text{NIL}, \ \theta(1) : \text{time}$

Time: $\Theta(1)$

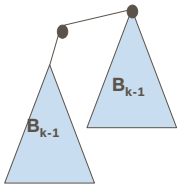
- **Finding the minimum key**

```
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
```

Time: O(lg n)

- **Linking 2 binomial trees**

```
BINOMIAL-LINK(y, z)
1  p[y] ← z
2  sibling[y] ← child[z]
3  child[z] ← y
4  degree[z] ← degree[z] + 1
```

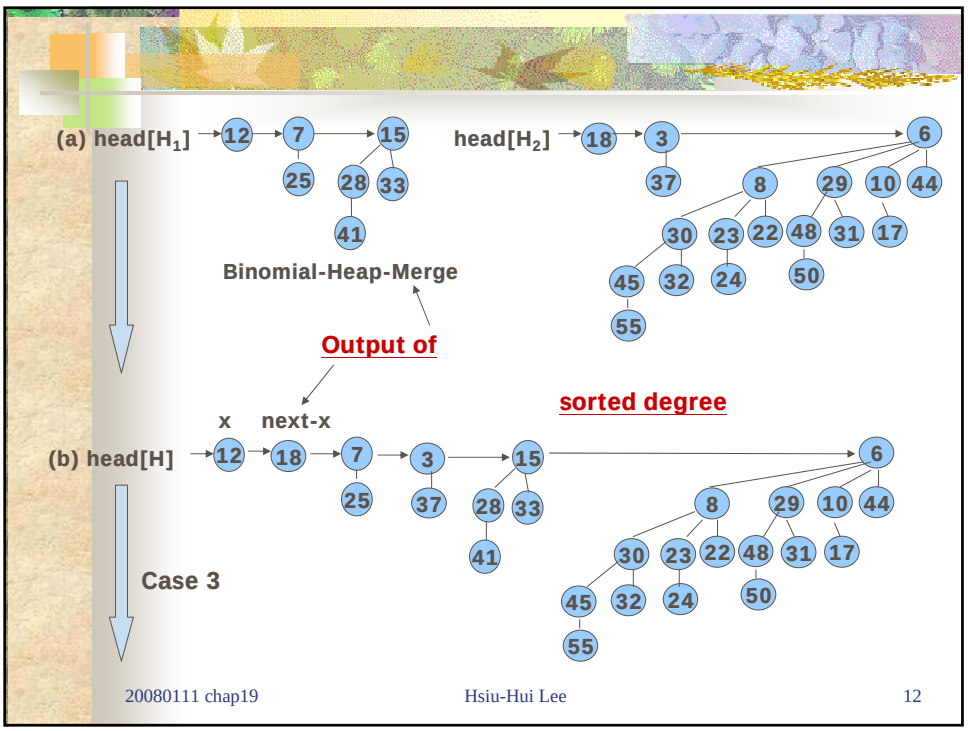


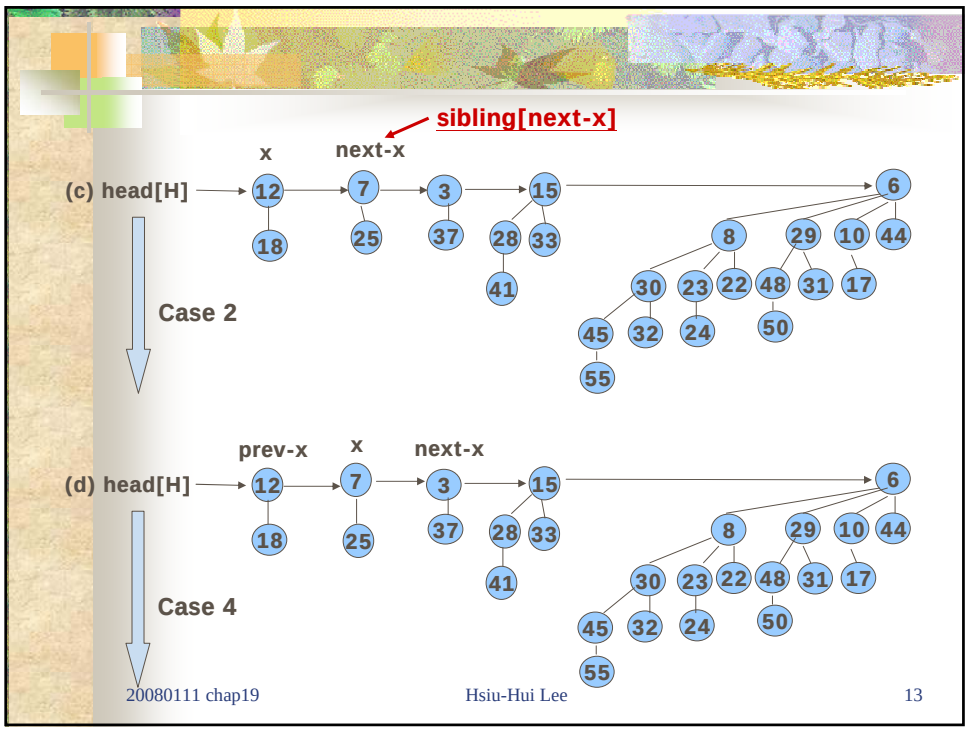
Y, z : B_{k-1} trees

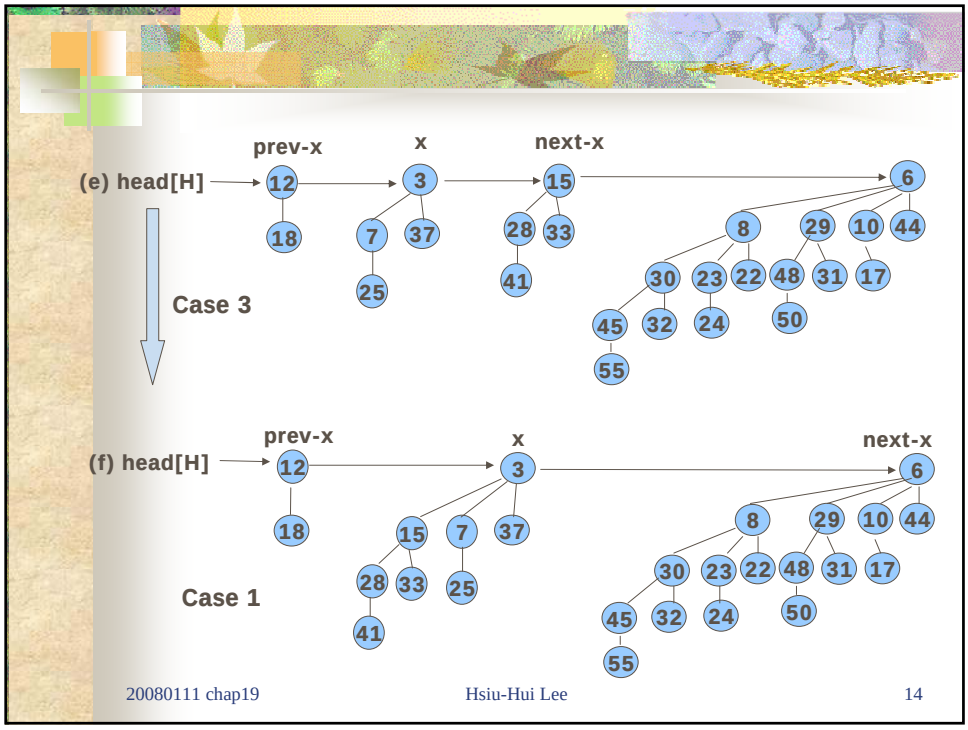
Uniting 2 binomial heaps

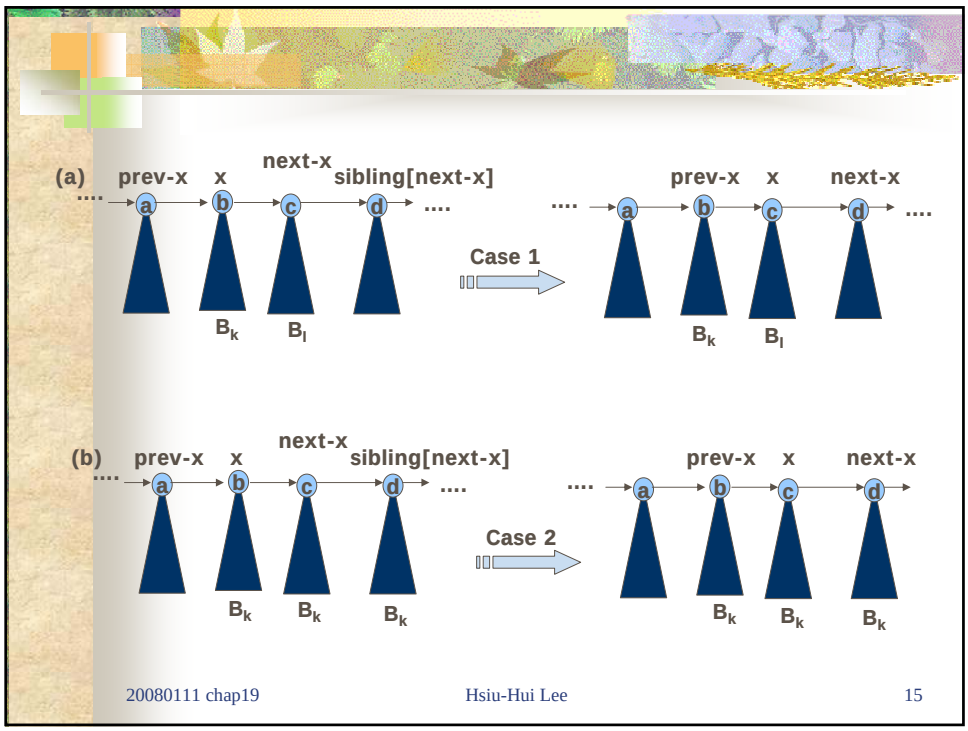
```
BINOMIAL-HEAP-UNION(H1, H2)
1  H ← MAKE-BINOMIAL-HEAP()
2  head[H] ← BINOMIAL-HEAP-MERGE(H1, H2)
3  free the objects H1 and H2 but not the lists they point to
4  if head[H] = NIL
5     then return H
6  prev-x ← NIL
7  x ← head[H]
8  next-x ← sibling[x]
9  while next-x ≠ NIL
10       do if (degree[x] ≠ 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
```

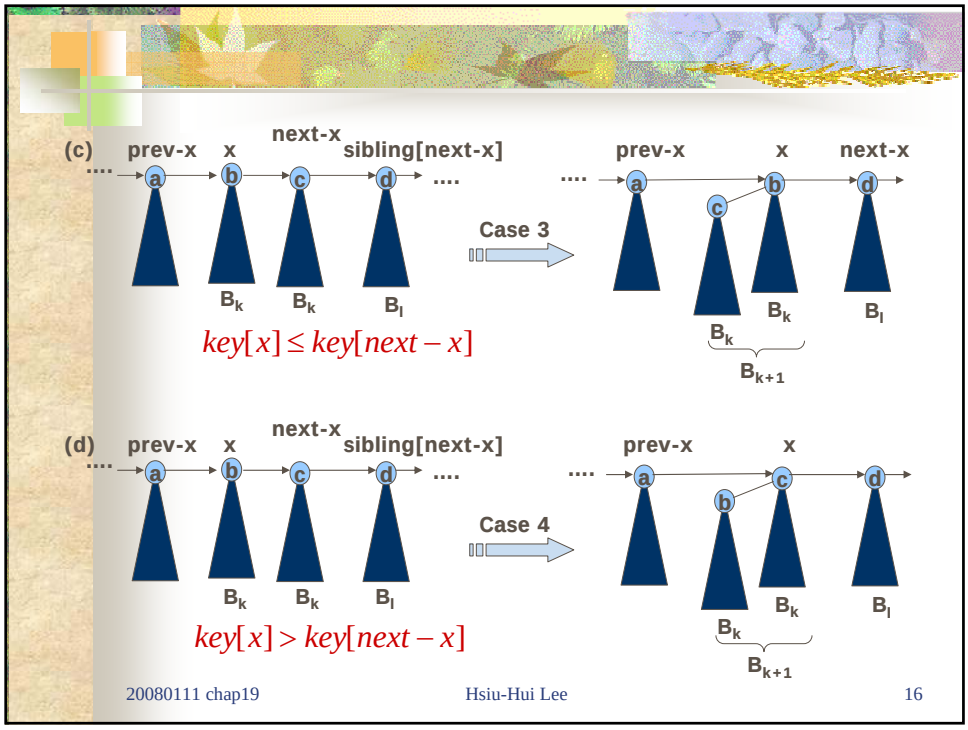
Time: O(lg n)




sibling[next-x]
x
next-x
(c) head[H]
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]
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 4
20080111 chap19
Hsiu-Hui Lee
13


prev-x
x
next-x
(e) head[H]
12
3
15
6
18
7
37
28
33
8
29
10
44
25
41
30
23
22
48
31
17
45
32
24
50
55
Case 3
prev-x
x
next-x
(f) head[H]
12
3
6
18
15
7
37
8
29
10
44
28
33
30
23
22
48
31
17
25
41
45
32
24
50
55
Case 1
20080111 chap19
Hsiu-Hui Lee
14


(a)
prev-x
x
next-x
sibling[next-x]
Case 1
(b)
Case 2
20080111 chap19
Hsiu-Hui Lee
15


(c)
prev-x
x
next-x
sibling[next-x]
Case 3
$key[x] \le key[next - x]$
(d)
Case 4
$key[x] > key[next - x]$
20080111 chap19
Hsiu-Hui Lee
16

- **Insert a node**

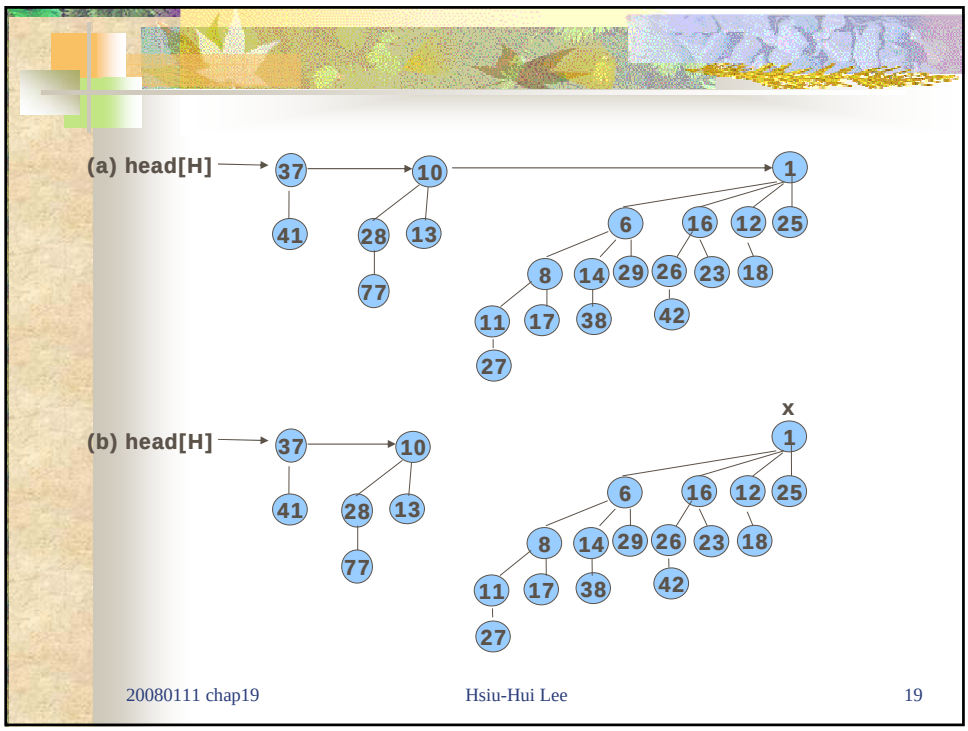
```
BINOMIAL-HEAP-INSERT(H, x)
1  H' ← MAKE-BINOMIAL-HEAP()
2  p[x] ← NIL
3  child[x] ← NIL
4  sibling[x] ← NIL
5  degree[x] ← 0
6  head[H'] ← x
7  H ← BINOMIAL-HEAP-UNION(H, H')
```

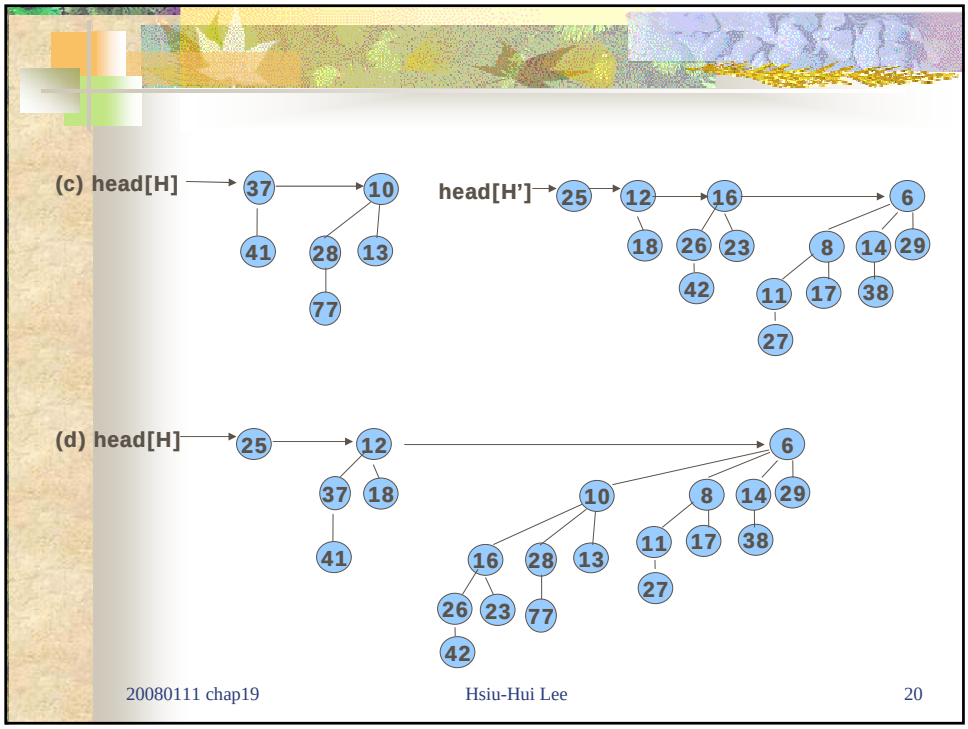
Time: O(lg n)

- **Extracting the node with minimum key**

```
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' ← 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 ← BINOMIAL-HEAP-UNION(H, H')
5  return x
```

Time: Θ (lg n)


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

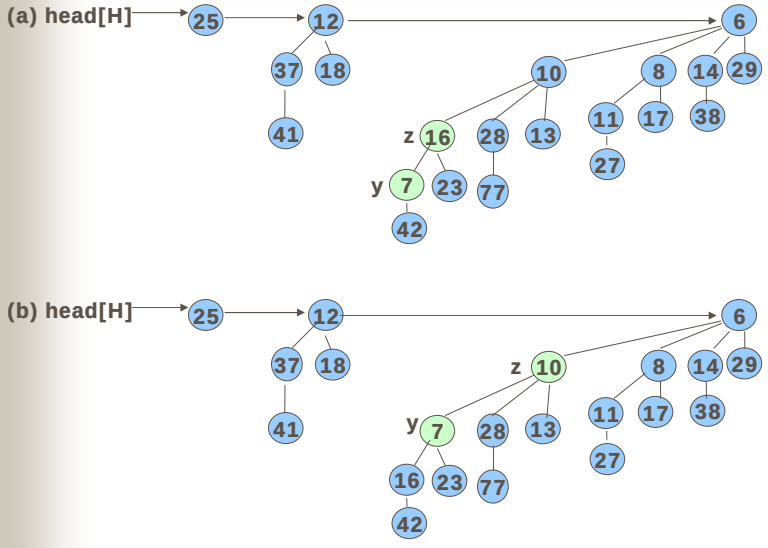

(c) head[H]
37
10
41
28
13
77
head[H']
25
12
16
6
18
26
23
8
14
29
42
11
17
38
27
(d) head[H]
25
12
6
37
18
10
8
14
29
41
16
28
13
11
17
38
27
26
23
77
42

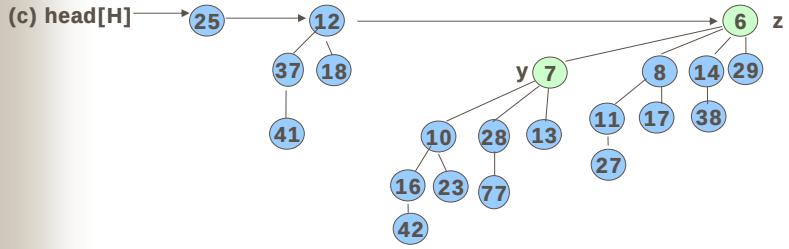
- **Decreasing a key**

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] ← k
4  y ← x
5  z ← p[y]
6  while z ≠ NIL and key[y] < key[z]
7        do exchange key[y] ↔ key[z]
8           ▷ If y and z have satellite fields, exchange them, too.
9           y ← z
10          z ← p[y]
```

Time: Θ (lg n)





- **Deleting a key**

BINOMIAL-HEAP-DELETE(H, x)

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

Time: Θ (lg n)