Mergeable Heaps

Supportano le operazioni

MAKE-HEAP()

INSERT(H, x)

MINIMUM(H)

EXTRACT-MIN(H)

UNION(H_1, H_2)

DECREASE-KEY(H, x, k)

DELETE(H, x)

Considereremo due implementazioni

- Heap binomiali (J. Vuillemin, 1978)
- Heap di Fibonacci (M. Fredman, R. Tarjan 1984–1987)

MAKE-HEAP()	creates and returns a new heap containing no elements.
INSERT(H, x)	inserts node x, whose *key* field has already been filled in, into heap H.
MINIMUM(H)	returns a pointer to the node in heap H whose key is minimum.
EXTRACT-MIN(H)	deletes the node from heap H whose key is minimum, returning a pointer to the node.
UNION(H_1, H_2)	creates and returns a new heap that contains all the nodes of heaps H_1 and H_2. Heaps H_1 and H_2 are "destroyed" by this operation.
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.
DELETE(H, x)	deletes node x from heap H.

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

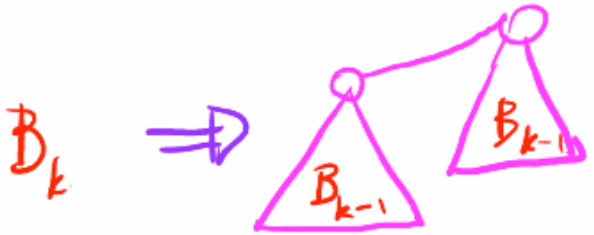
(*) Costi ammortizzati

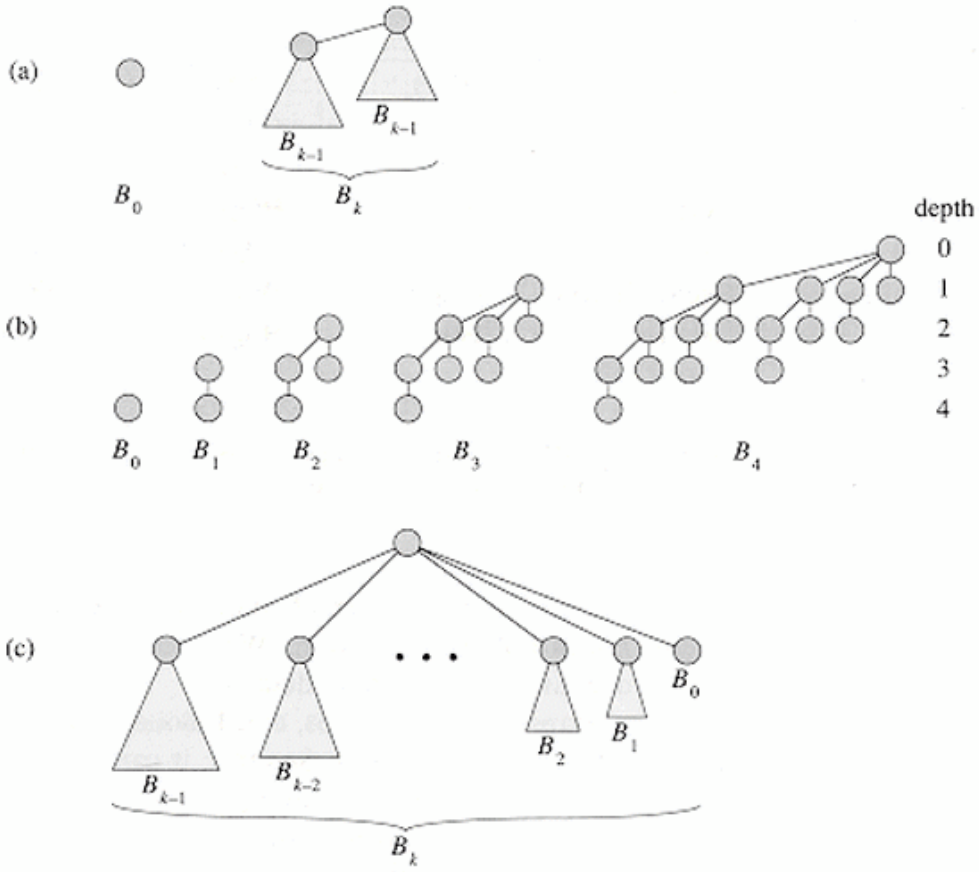
ALBERI BINOMIALI

DEFINIZIONE

PER OGNI $k \in \mathbb{N}$ ESISTE UN ALBERO BINOMIALE B_k DI GRADO k, DEFINITO IN BASE ALLA SEGUENTE RICORSIONE:

- B_0 E' L'ALBERO FORMATO DA UN SOLO NODO
- DATO B_{k-1} DEFINIAMO B_k COMBINANDO DUE COPIE DI B_{k-1} NELLA SEGUENTE MANIERA:




(a)
B_0
B_{k-1}
B_{k-1}
B_k
(b)
depth
0
1
2
3
4
B_0
B_1
B_2
B_3
B_4
(c)
B_{k-1}
B_{k-2}
B_2
B_1
B_0
B_k

LEMMA (PROPRIETÀ DEGLI ALBERI BINOMIALI)

PER OGNI $k = 0, 1, 2, \ldots$ VALGONO LE SEGUENTI PROPRIETÀ:

1. B_k HA 2^k NODI
2. L'ALTEZZA DI B_k È k
3. B_k HA $\binom{k}{i}$ NODI A PROFONDITÀ i $(i = 0, 1, \ldots, k)$
4. LA RADICE DI B_k HA GRADO k ED OGNI ALTRO NODO IN B_k HA GRADO $< k$, INOLTRE I FIGLI DELLA RADICE DI B_k SONO RADICI DI $B_{k-1}, B_{k-2}, \ldots, B_2, B_1, B_0$, NELL'ORDINE DATO.

DIM.

PER INDUZIONE

CASO BASE $k=0$

1. B_0 HA $1 = 2^0$ NODI
2. L'ALTEZZA DI B_0 E' 0
3. B_0 HA $1 = \binom{0}{0}$ NODI A PROFONDITA' 0
4. LA RADICE DI B_0 HA GRADO 0

PASSO INDUTTIVO

SUPPONIAMO CHE IL LEMMA SIA VERO PER $k-1$, CON $k \geq 1$

1. B_k HA $\#(B_{k-1}) + \#(B_{k-1}) = 2 \cdot 2^{k-1} = 2^k$ NODI
2. L'ALTEZZA DI B_k E' UGUALE A
   $height(B_{k-1}) + 1 = (k-1) + 1 = k$
3. SIA $1 \leq i \leq k-1$.
   IL NUMERO DI NODI DI B_k A PROFONDITA' i È UGUALE A $\binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$.
   INOLTRE B_k HA $-1 = \binom{k}{0}$ NODI A PROFONDITA' 0
   $-\binom{k-1}{k-1} = \binom{k}{k} = 1$ NODI A PROFONDITA' k

4. • IL GRADO DELLA RADICE DI B_k E' UGUALE A

$$\text{degree}(B_{k-1}) + 1 = (k-1) + 1 = k$$

• INOLTRE CIASCUN ALTRO NODO APPARTIENE AD UN B_{k-1} E PERTANTO PER IPOTESI INDUTTIVA HA GRADO $\leq k-1$, CIOE' $< k$

• IL PRIMO FIGLIO DELLA RADICE DI B_k E' RADICE DI B_{k-1}. INOLTRE, PER INDUTTIVA, I SUCCESSIVI $k-1$ FIGLI DELLA RADICE DI B_k SONO RADICI DI $B_{k-2}, \ldots, B_1, B_0$.

■

COROLLARIO

SIA B UN ALBERO BINOMIALE CON n NODI. ALLORA OGNI NODO IN B HA GRADO AL PIÙ $\lg n$.

DIM.

PER QUALCHE $k \in \mathbb{N}$ SI HA $B = B_k$. QUINDI $n = 2^k$. OGNI NODO IN B_k HA GRADO $\leq k$, MA $k = \lg n$, DA CUI LA TESI. ∎

HEAP BINOMIALI

DEFINIZIONE

UNO HEAP BINOMIALE H E' UN INSIEME DI ALBERI BINOMIALI TALE CHE

- CIASCUN ALBERO BINOMIALE IN H GODE DELLA PROPRIETA' min-heap
- PER OGNI $k \in \mathbb{N}$, H CONTIENE AL PIÙ UN SOLO ALBERO BINOMIALE DI GRADO k

CONSEGUENZE IMMEDIATE:

- IN CIASCUN ALBERO BINOMIALE IN UNO HEAP BINOMIALE, LA RADICE CONTIENE LA CHIAVE MINIMA DELL'ALBERO
- UNO HEAP BINOMIALE H CON n NODI E' FORMATO DA AL PIÙ $\lfloor \lg n \rfloor + 1$ ALBERI BINOMIALI

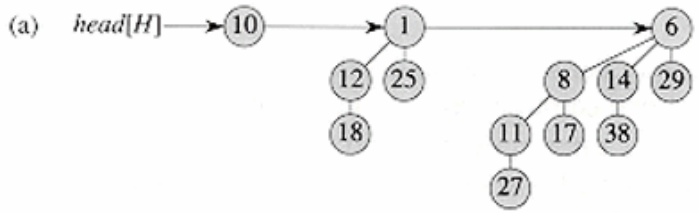
  INFATTI, SIA B_k L'ALBERO BINOMIALE DI GRADO MASSIMO IN H. SI HA $2^k \leq n$, DA CUI $k \leq \lg n$ E QUINDI $k \leq \lfloor \lg n \rfloor$.

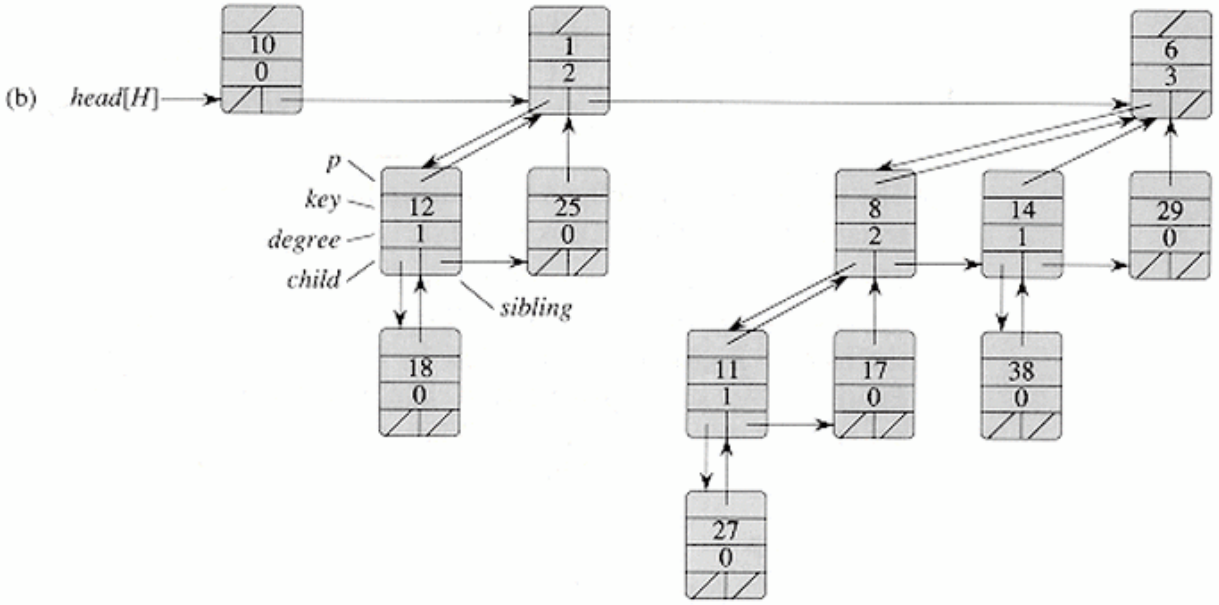
  PERTANTO IL NUMERO DI ALBERI BINOMIALI IN H E' AL PIÙ $k+1 \leq \lfloor \lg n \rfloor + 1$

ESEMPIO DI HEAP BINOMIALE

$13 = 2^3 + 2^2 + 2^0$

$B_3 \quad B_2 \quad B_0$





```
MAKE_BINOMIAL_HEAP()
    H := allocate_heap();
    head[H] := NIL;
    return H;
```

COMPLESSITA'

$O(1)$

```
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
```

COMPLESSITA'

O(LUNGHEZZA DELLA LISTA DELLE RADICI)

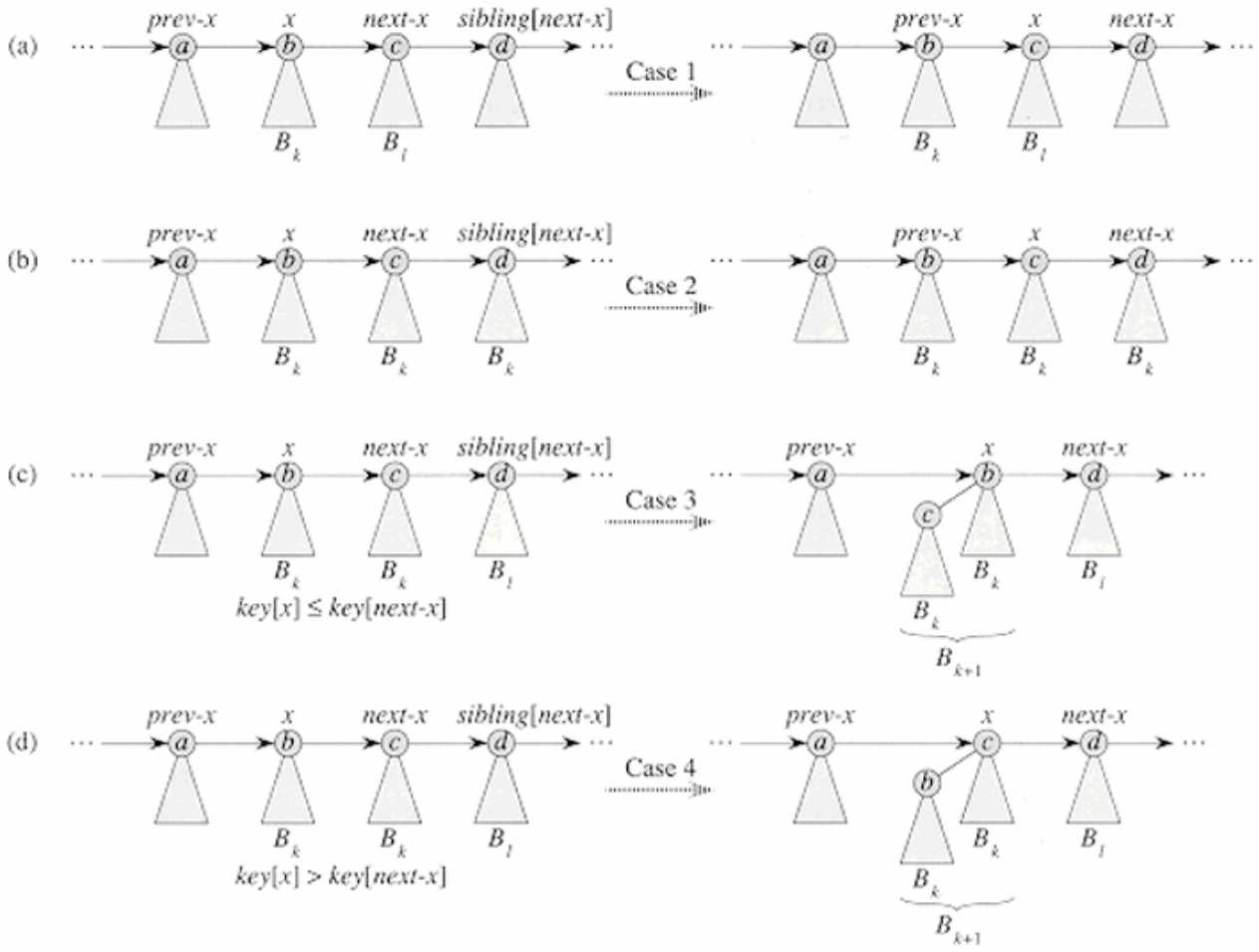
$= O(\lg n)$

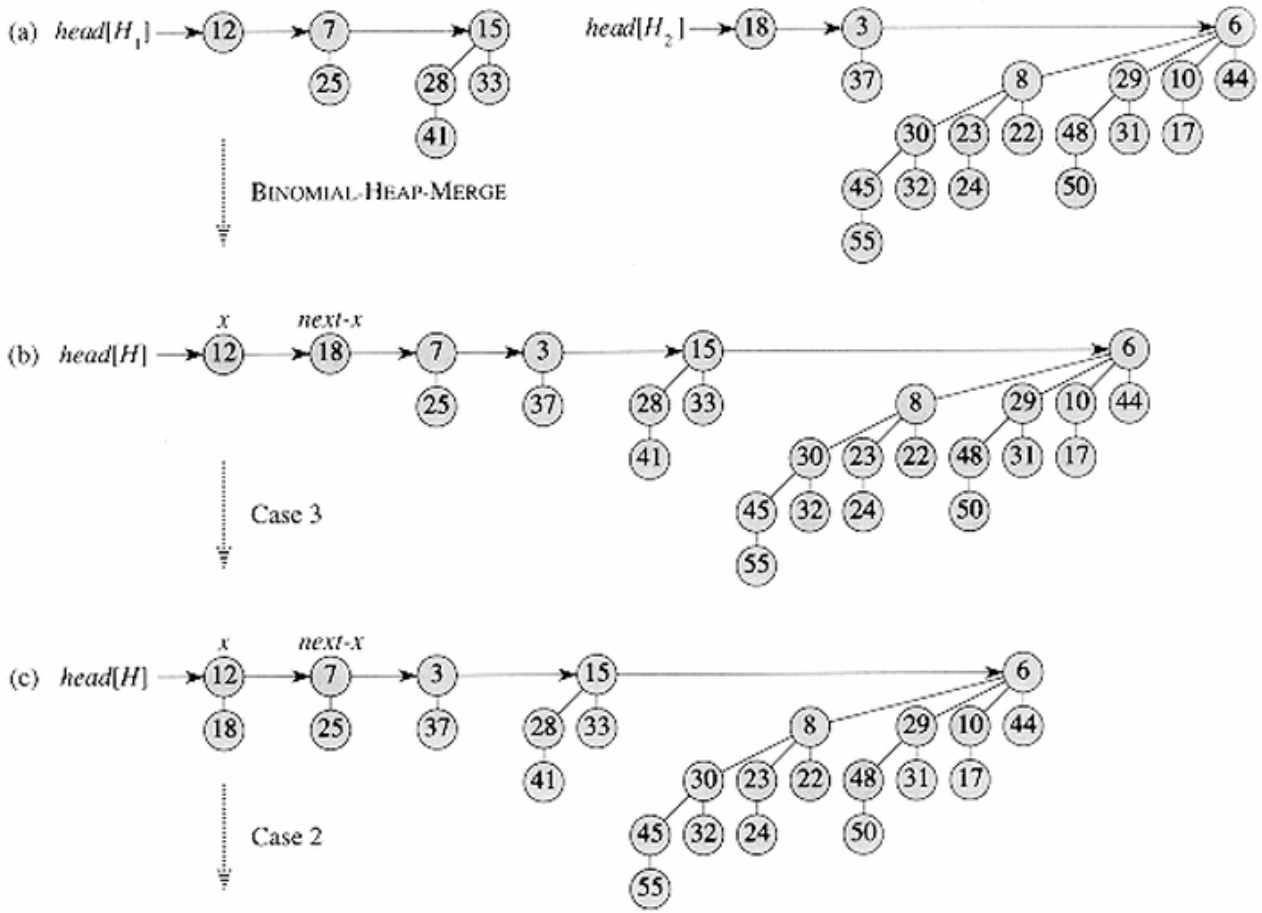
```
BINOMIAL-LINK(y,z)
1  p[y] ← z
2  sibling[y] ← child[z]
3  child[z] ← y
4  degree[z] ← degree[z]+1
```

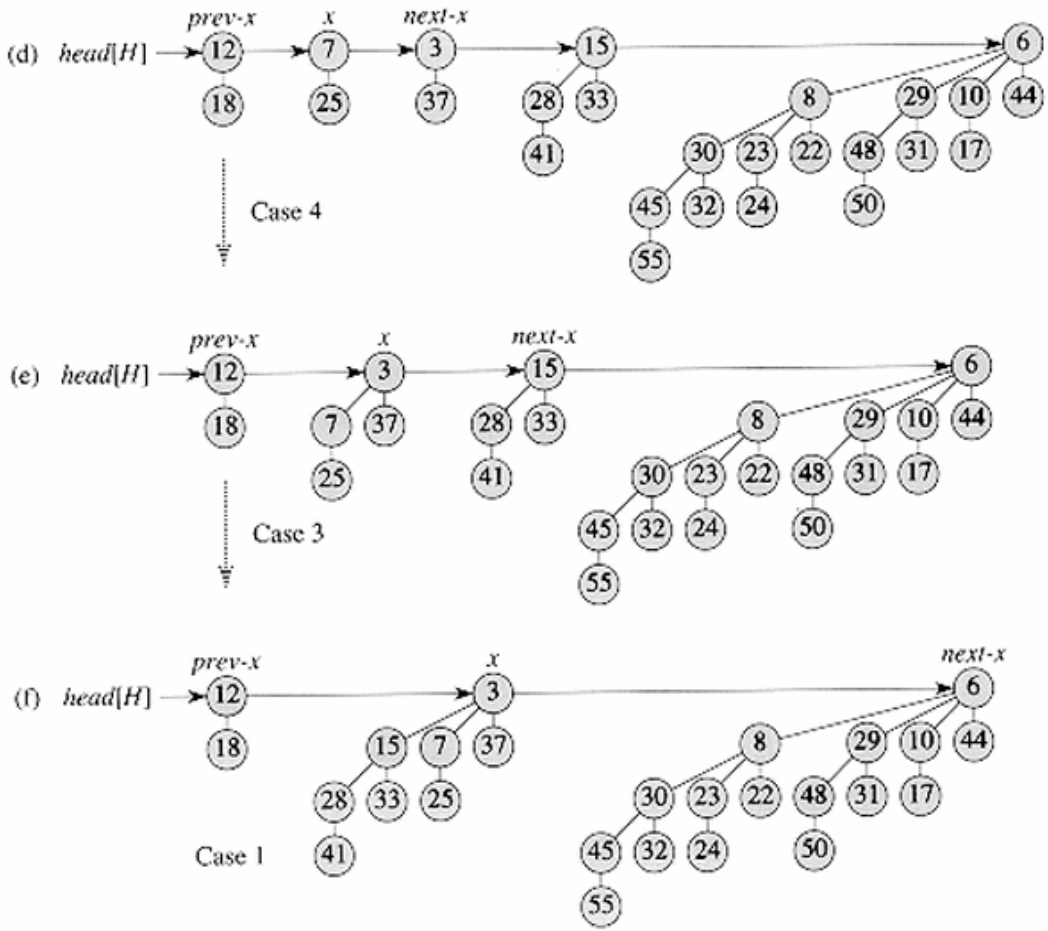
COMPLESSITA'

O(1)

```
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)
prev-x
x
next-x
sibling[next-x]
Case 1
(b)
Case 2
(c)
Case 3
key[x] ≤ key[next-x]
(d)
Case 4
key[x] > key[next-x]


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


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

COMPLESSITÀ DI BINOMIAL-HEAP-UNION

$n_1 = \#\text{nodi}(H_1) \rightarrow \#\text{alberi}(H_1) \leq \lfloor \log n_1 \rfloor + 1$

$n_2 = \#\text{nodi}(H_2) \rightarrow \#\text{alberi}(H_2) \leq \lfloor \log n_2 \rfloor + 1$

$n = n_1 + n_2$

$$\begin{aligned} \#\text{alberi}(H_1 \cup H_2) &\leq \lfloor \log n_1 \rfloor + \lfloor \log n_2 \rfloor + 2 \\ &\leq 2 \lfloor \log n \rfloor + 2 \end{aligned}$$

COMPLESSITÀ DI BINOMIAL-HEAP-MERGE: $O(\log n)$

SI OSSERVI CHE AD OGNI ITERAZIONE DEL CICLO WHILE 9-21

- IL PUNTATORE x AVANZA DI UNA POSIZIONE,

OPPURE

- VIENE RIMOSSA UNA RADICE DALLA LISTA DELLE RADICI

INOLTRE, CIASCUNA ITERAZIONE DEL CICLO WHILE PRENDE TEMPO COSTANTE.

PERTANTO LA COMPLESSITA' DI BINOMIAL-HEAP-UNION E' $O(\lg n)$

```
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')
```

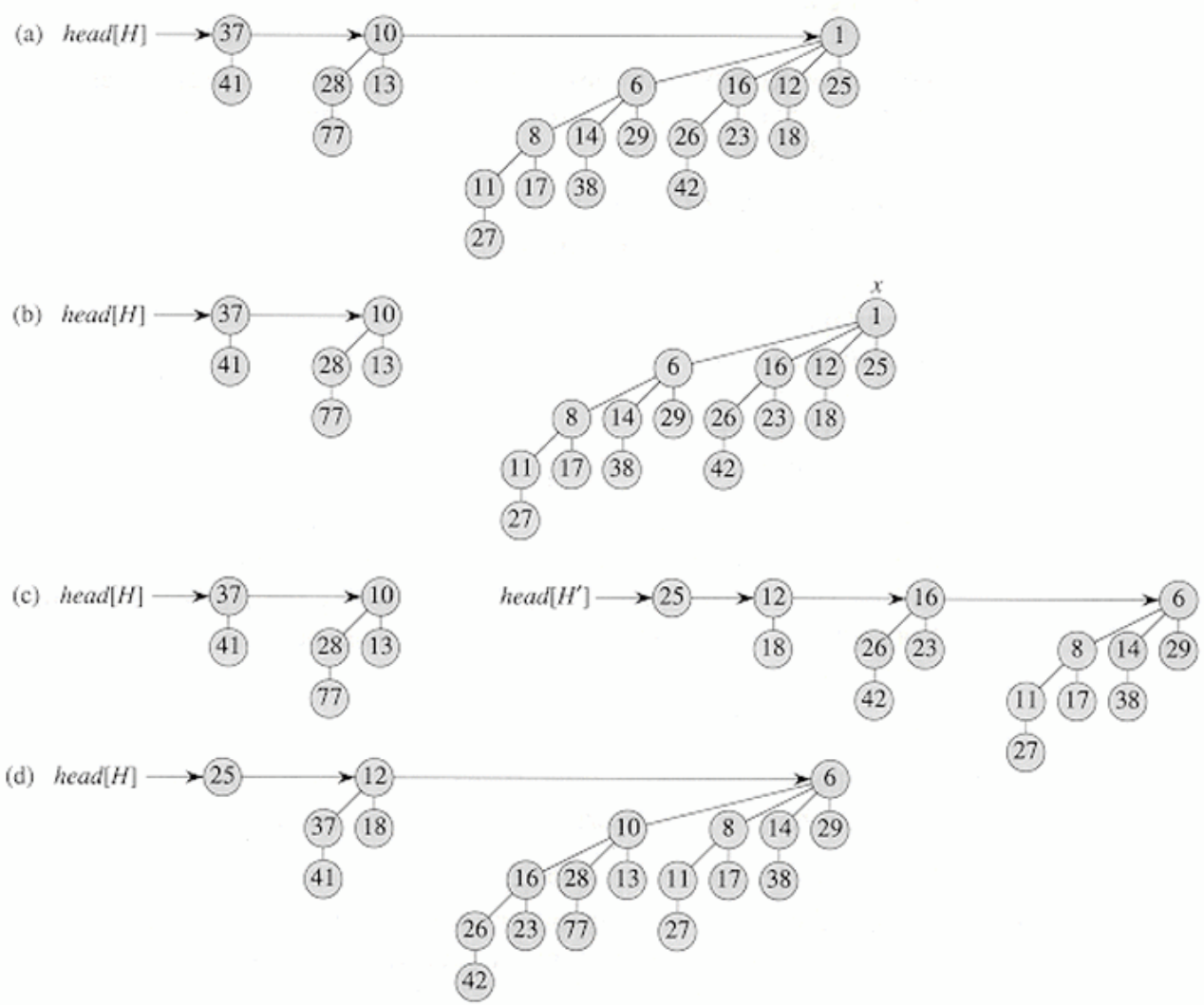
SIA $n = \#\text{nodi}(H)$.
ALLORA LA COMPLESSITA' DI BINOMIAL-HEAP-INSERT E' $O(\log n)$

```
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' ← 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
```

COMPLESSITÀ' ($n = \#\text{nodi}(H)$)

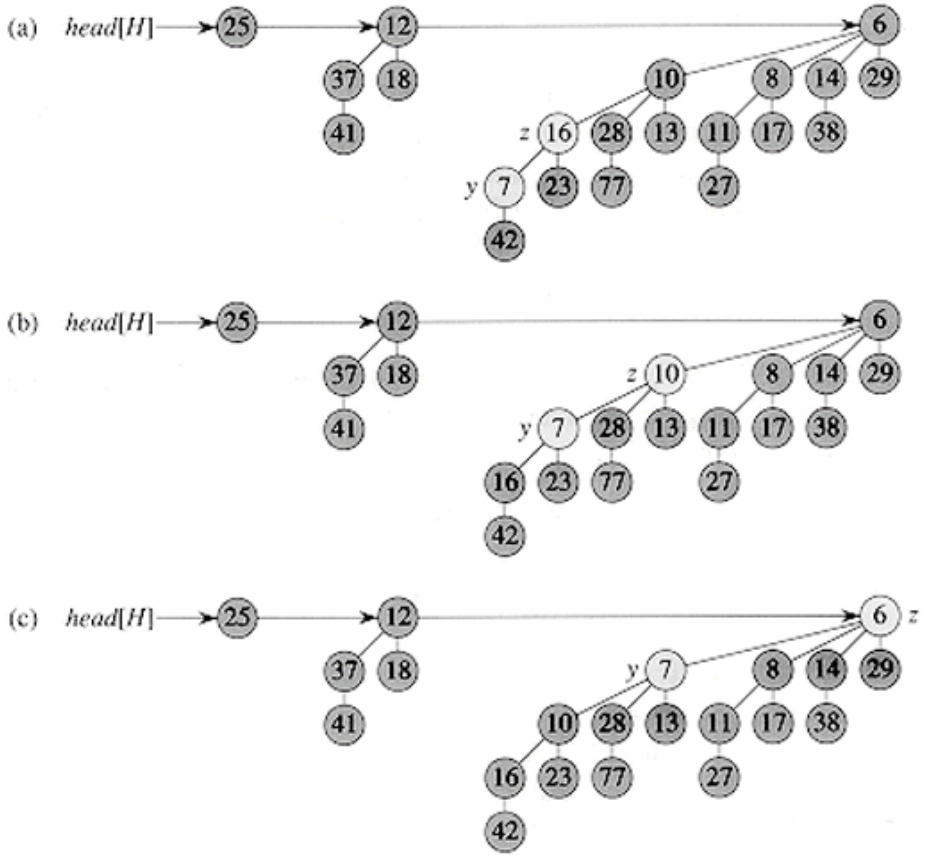
1 → $O(\lg n)$
2 → $O(1)$
3 → $O(\lg n)$
4 → $O(\lg n)$

QUINDI BINOMIAL-HEAP-EXTRACT-MIN HA COMPLESSITÀ' $O(\lg n)$


(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]
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
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
26
23
77
27
42

```
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]
```

COMPLESSITA': $O(\lg n)$


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

```
BINOMIAL-HEAP-DELETE(H,x)

1 BINOMIAL-HEAP-DECREASE-KEY(H,x,-∞)

2 BINOMIAL-HEAP-EXTRACT-MIN(H)
```

COMPLESSITA': $O(\lg n)$, CON $n = \#\,\text{nodi}(H)$