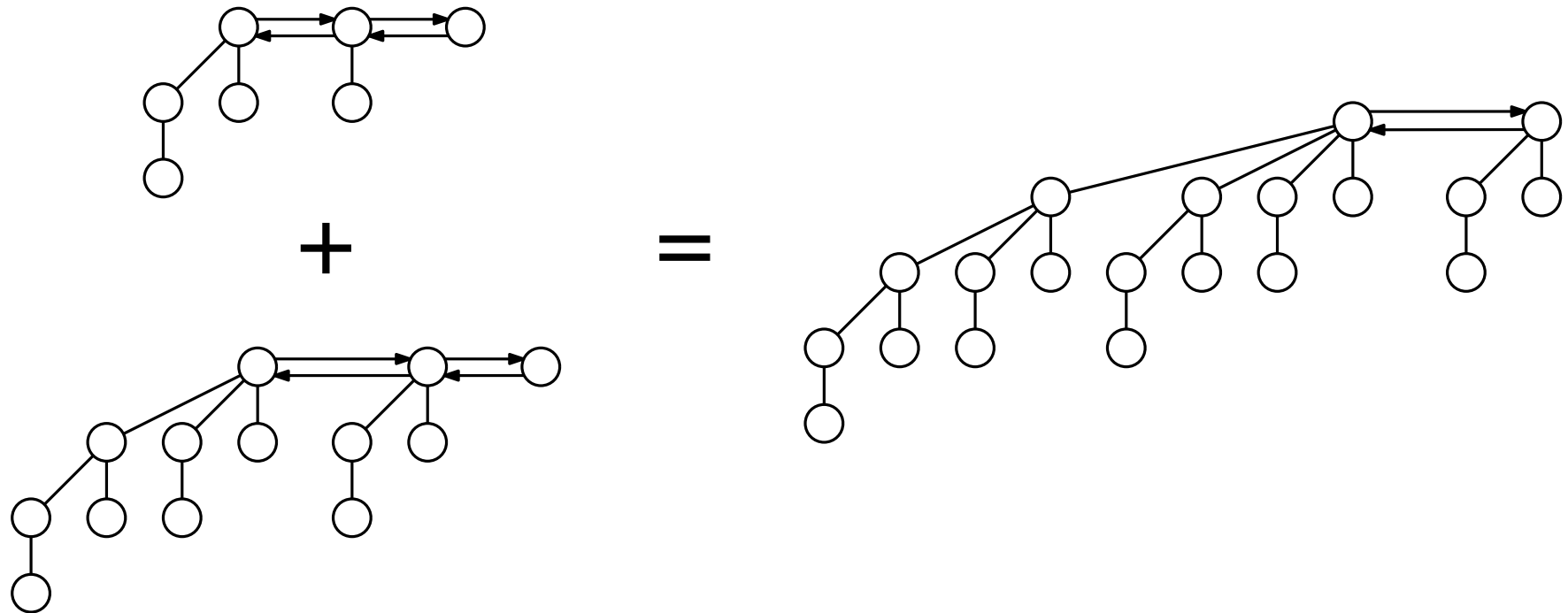




ICS 621: Analysis of Algorithms

Prof. Nodari Sitchinava



Mergeable Priority Queues: Binomial Heaps

Priority Queue

PQ Abstract Data Type (ADT):

- MAKE()
- INSERT(Q, x)
- MINIMUM(Q)
- EXTRACT-MIN(Q)
- DECREASE-KEY(Q, x, k)
- DELETE(Q, x)

Priority Queue

PQ Abstract Data Type (ADT):

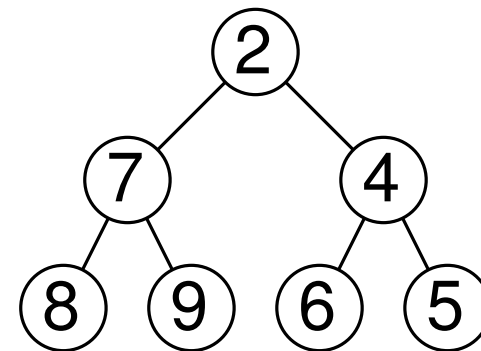
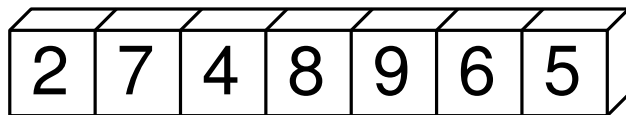
- MAKE()
- INSERT(Q, x)
- MINIMUM(Q)
- EXTRACT-MIN(Q)
- DECREASE-KEY(Q, x, k)
- DELETE(Q, x)

- UNION(Q_1, Q_2)

Binary Heap (from ICS 311)

- MAKE()
- INSERT(Q, x)
- MINIMUM(Q)
- EXTRACT-MIN(Q)
- DECREASE-KEY(Q, x, k)
- DELETE(Q, x)

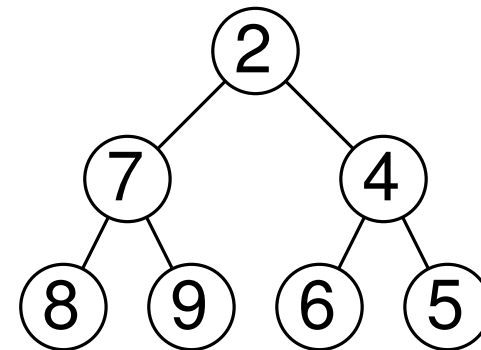
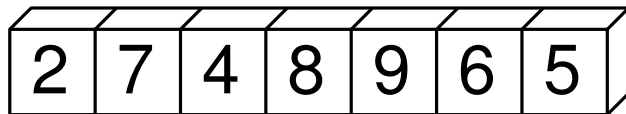
- UNION(Q_1, Q_2)



Binary Heap (from ICS 311)

- MAKE() $O(1)$
- INSERT(Q, x) $O(\log n)$
- MINIMUM(Q) $O(1)$
- EXTRACT-MIN(Q) $O(\log n)$
- DECREASE-KEY(Q, x, k) $O(\log n)$
- DELETE(Q, x) $O(\log n)$

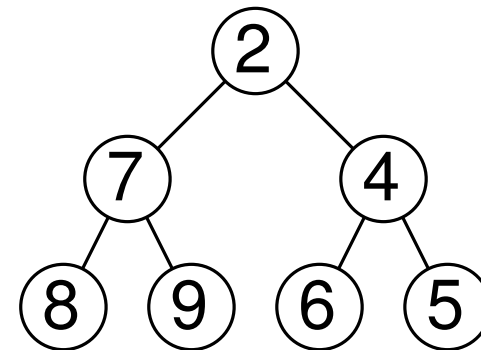
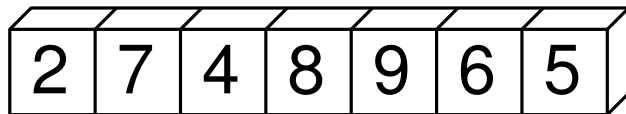
- UNION(Q_1, Q_2)



Binary Heap (from ICS 311)

- MAKE() $O(1)$
- INSERT(Q, x) $O(\log n)$
- MINIMUM(Q) $O(1)$
- EXTRACT-MIN(Q) $O(\log n)$
- DECREASE-KEY(Q, x, k) $O(\log n)$
- DELETE(Q, x) $O(\log n)$

- UNION(Q_1, Q_2) $O(n)$



Today: Binomial Heap

- MAKE()
- INSERT(Q, x)
- MINIMUM(Q)
- EXTRACT-MIN(Q)
- DECREASE-KEY(Q, x, k)
- DELETE(Q, x)

- UNION(Q_1, Q_2)

Today: Binomial Heap

- MAKE()
- INSERT(Q, x)
- MINIMUM(Q)
- EXTRACT-MIN(Q)
- DECREASE-KEY(Q, x, k)
- DELETE(Q, x)

- UNION(Q_1, Q_2)

Standard

$O(1)$
 ~~$O(\log n)$~~ $O(1)^*$
 $O(1)$
 $O(\log n)$
 $O(\log n)$
 $O(\log n)$

 ~~$O(n)$~~ $O(\log n)$

* Amortized cost

Today: Binomial Heap

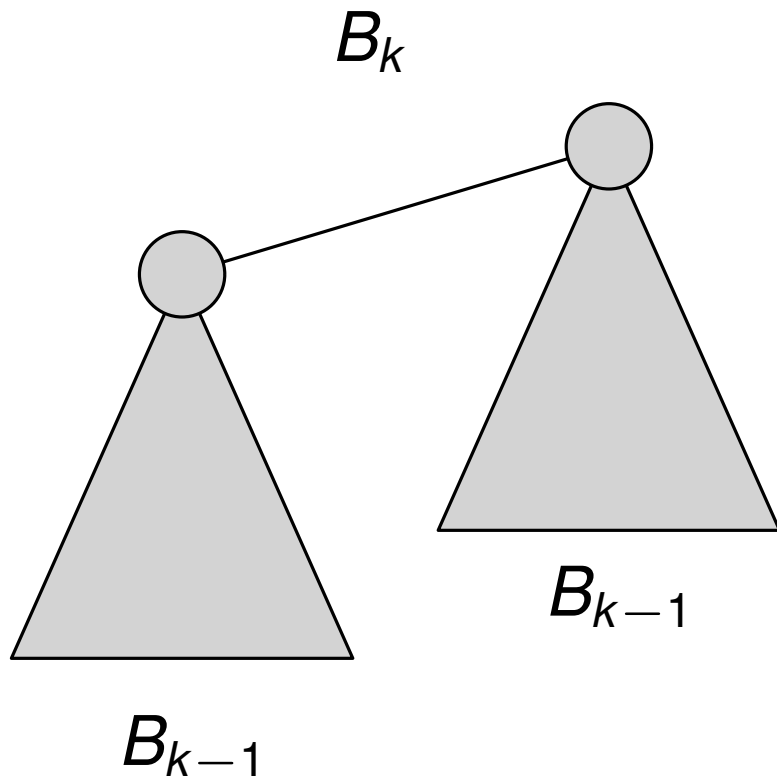
- MAKE()
- INSERT(Q, x)
- MINIMUM(Q)
- EXTRACT-MIN(Q)
- DECREASE-KEY(Q, x, k)
- DELETE(Q, x)

- UNION(Q_1, Q_2)

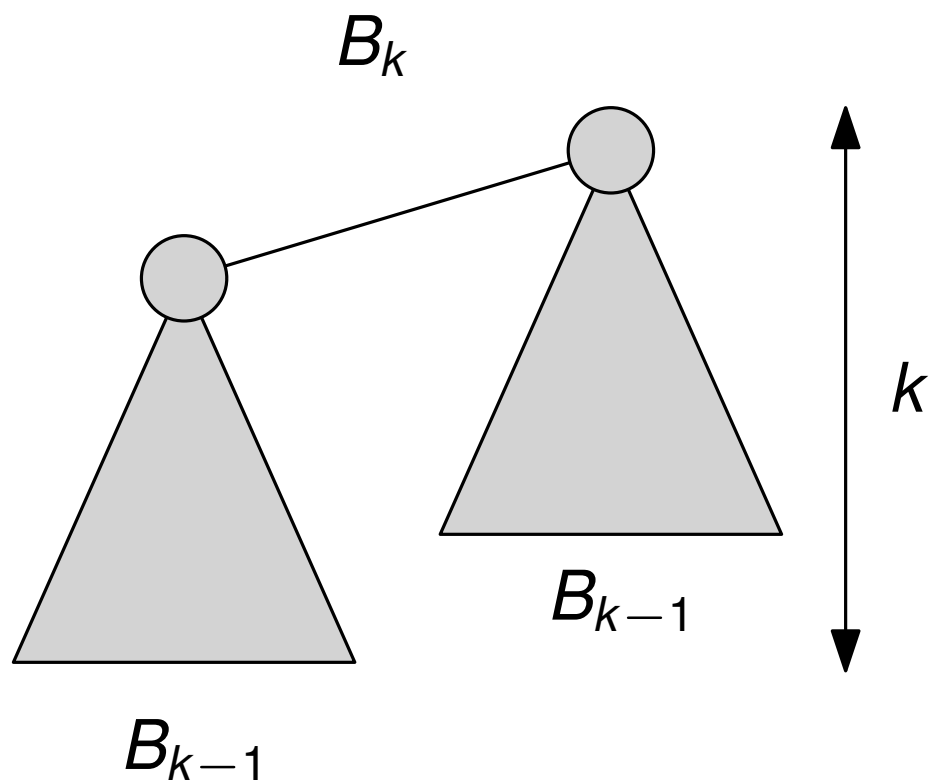
	Standard		Lazy
	$O(1)$		
	$O(\log n)$	$O(1)^*$	$O(1)$
	$O(1)$		$O(1)$
	$O(\log n)$		$O(\log n)^*$
	$O(\log n)$		
	$O(\log n)$		$O(\log n)^*$
	$O(n)$	$O(\log n)$	$O(1)$

* Amortized cost

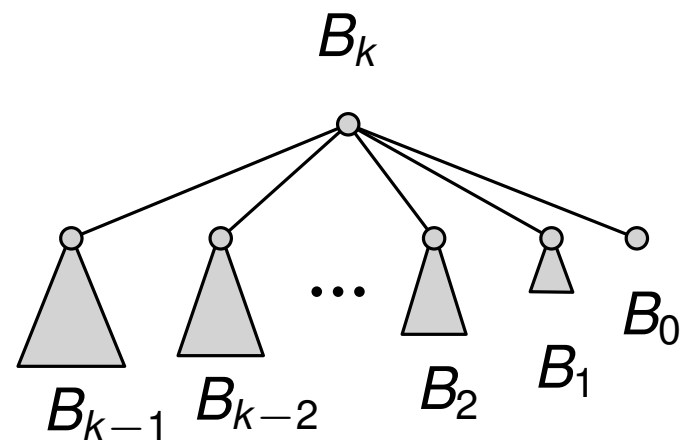
Reminder: Binomial Tree



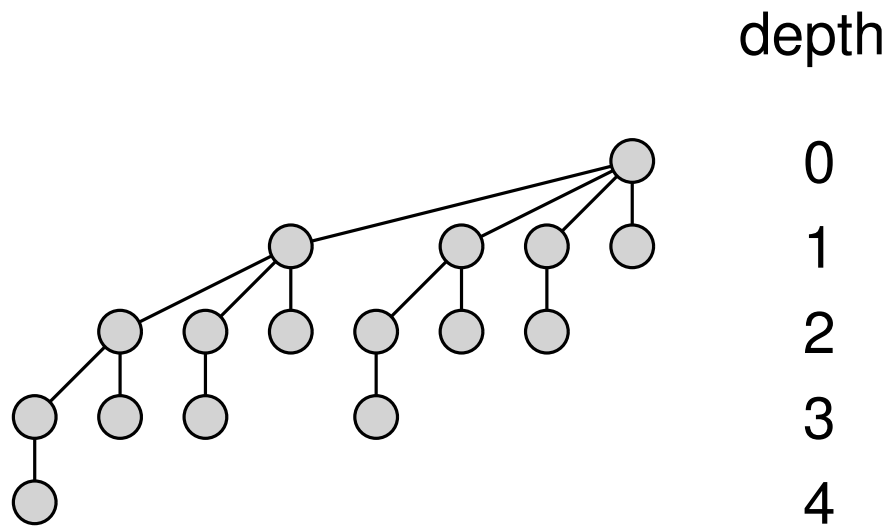
Reminder: Binomial Tree



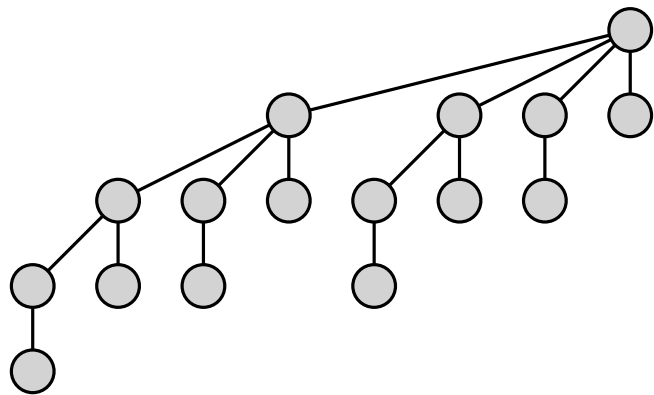
1. *size* = 2^k nodes
2. *height* = k
3. $\binom{k}{i}$ nodes at depth i
4. *degree*(root) = k



Tree representation

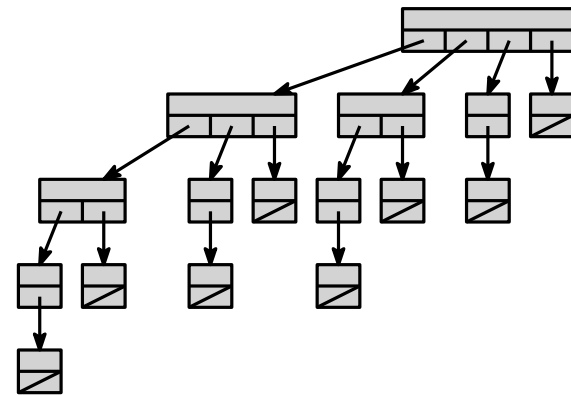


Tree representation

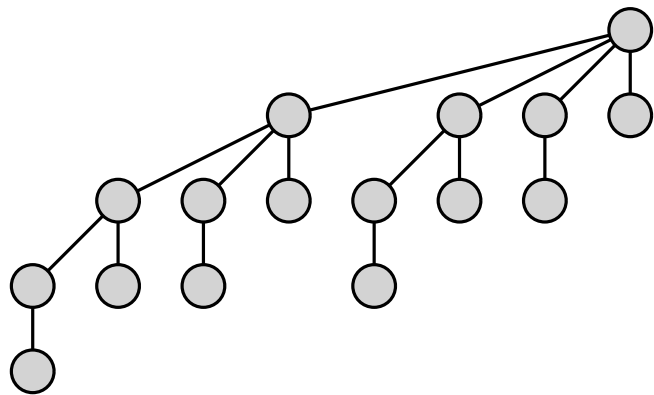


depth

0
1
2
3
4

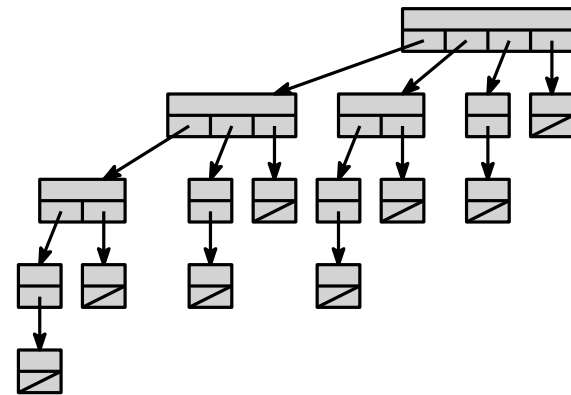


Tree representation



depth

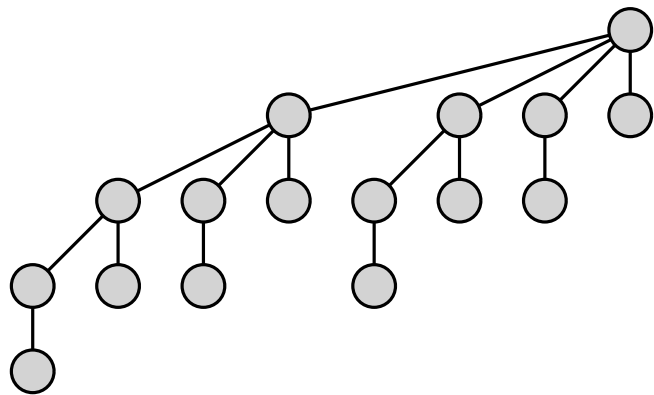
0
1
2
3
4



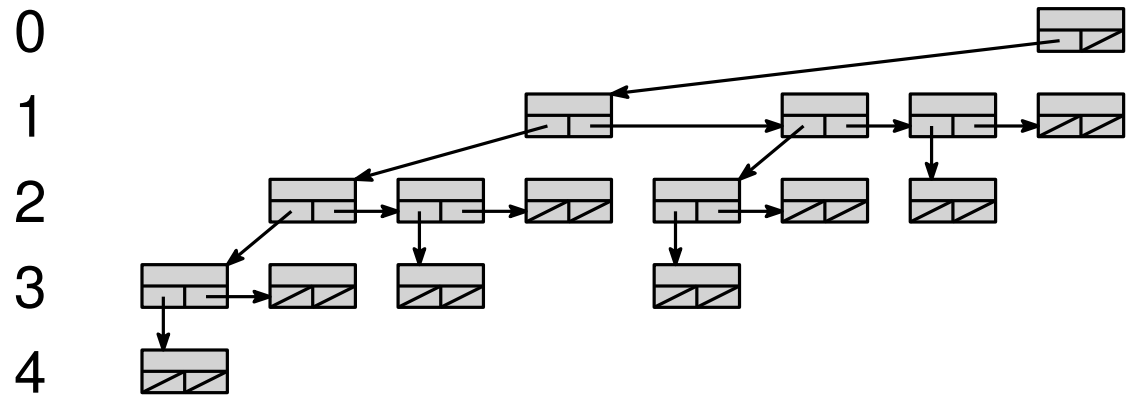
Variable number of children
at each node!

Tree representation

Left child, right sibling representation

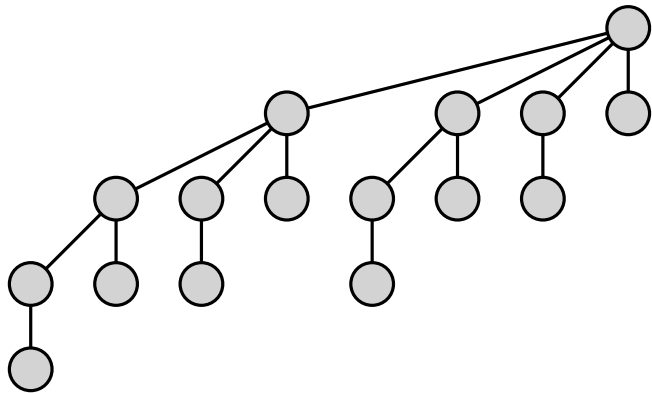


depth

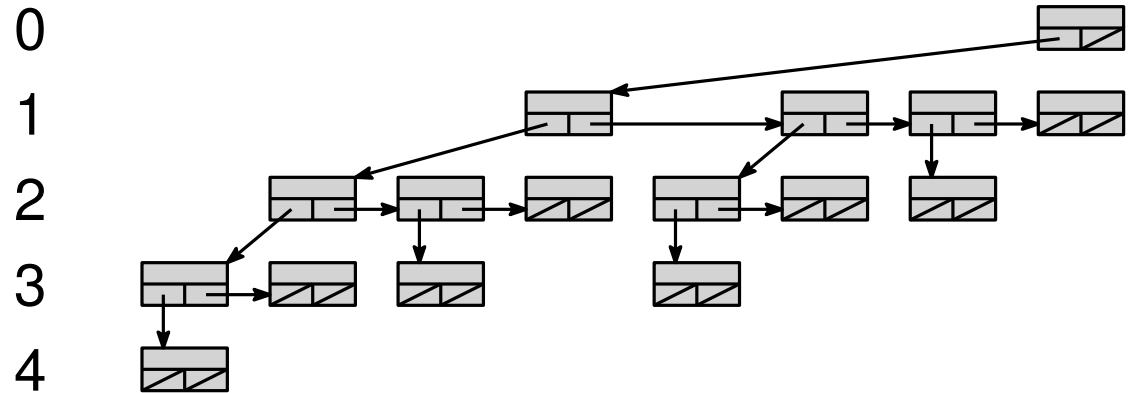


Tree representation

Left child, right sibling representation

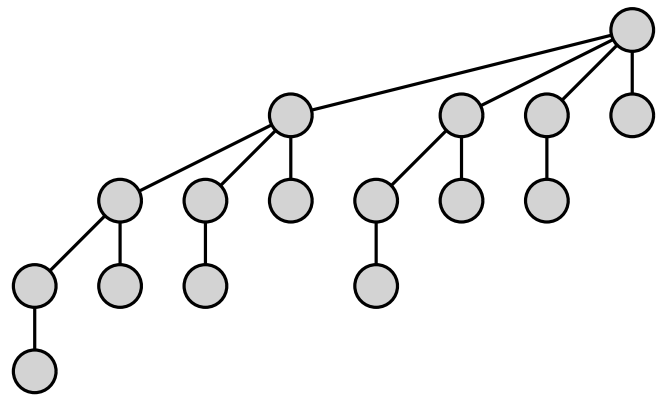


depth



Each node is identical

Tree representation



Left child, right sibling representation

- add parent pointers

depth

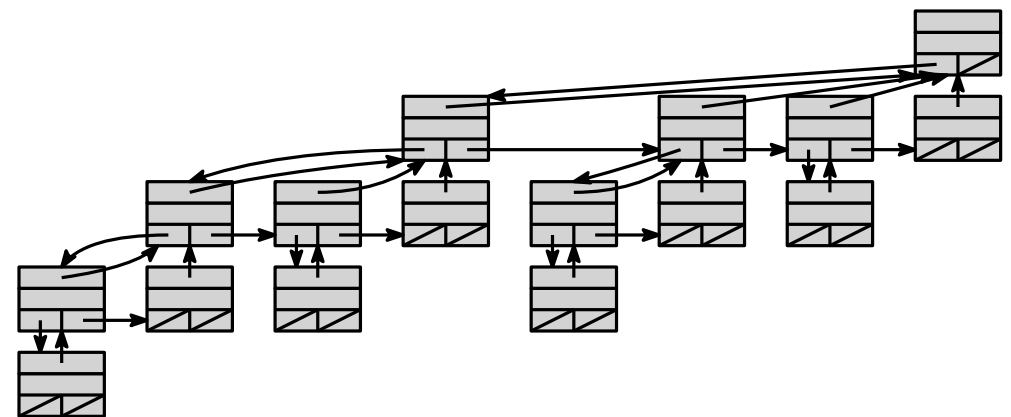
0

1

2

3

4



Each node is identical

Binomial heap

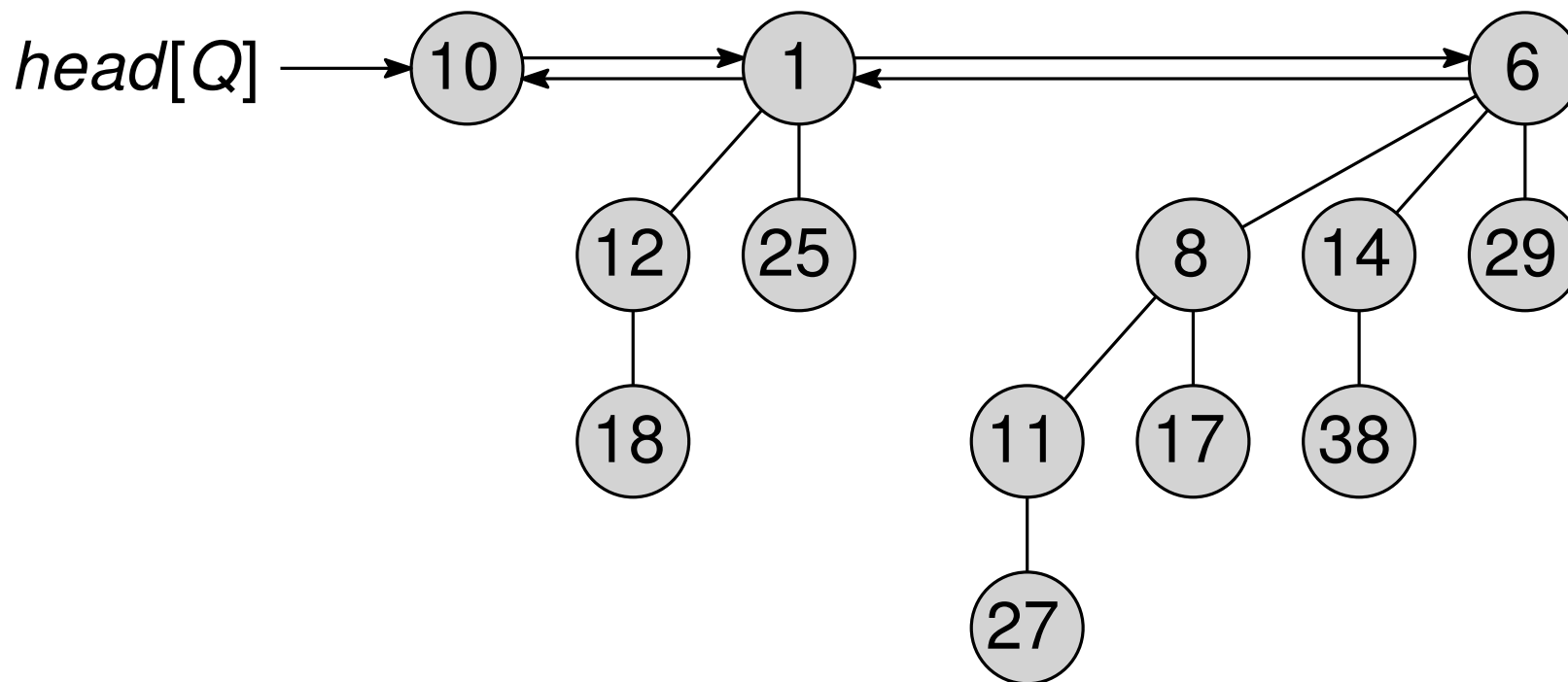
Collection of heap-ordered binomial trees:

- Each tree is heap-ordered
- At most **one** tree B_k , for $k = 0, 1, 2, \dots, \lfloor \log n \rfloor$

Binomial heap

Collection of heap-ordered binomial trees:

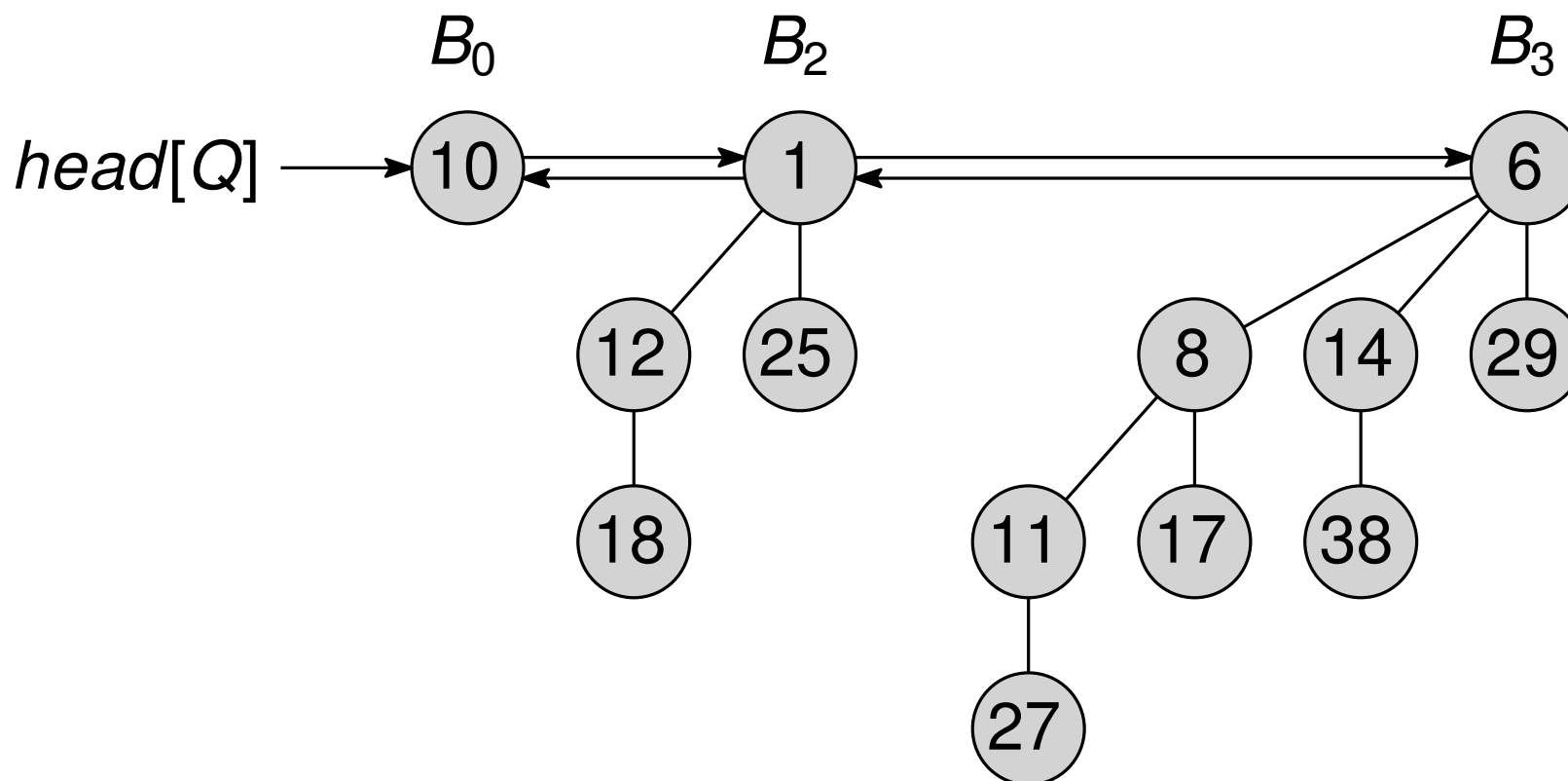
- Each tree is heap-ordered
- At most **one** tree B_k , for $k = 0, 1, 2, \dots, \lfloor \log n \rfloor$



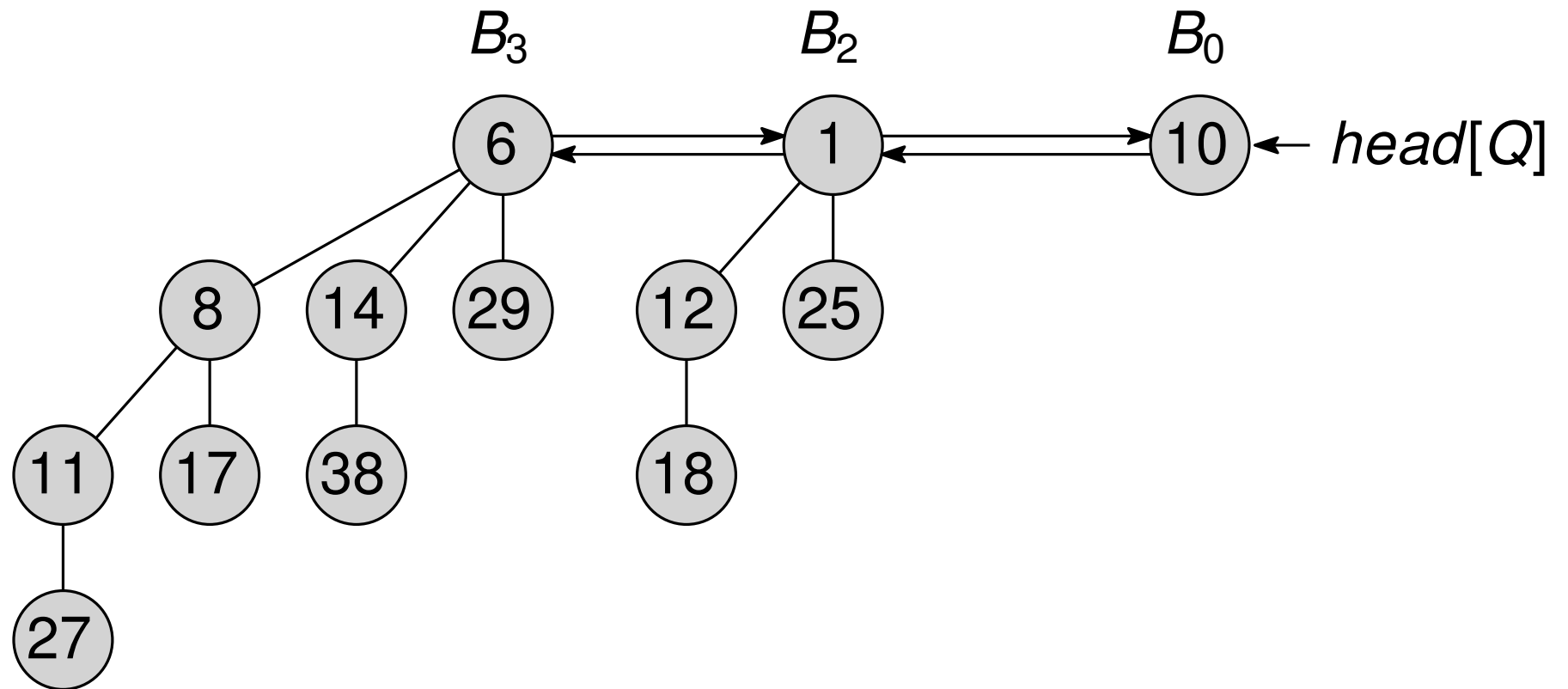
Binomial heap

Collection of heap-ordered binomial trees:

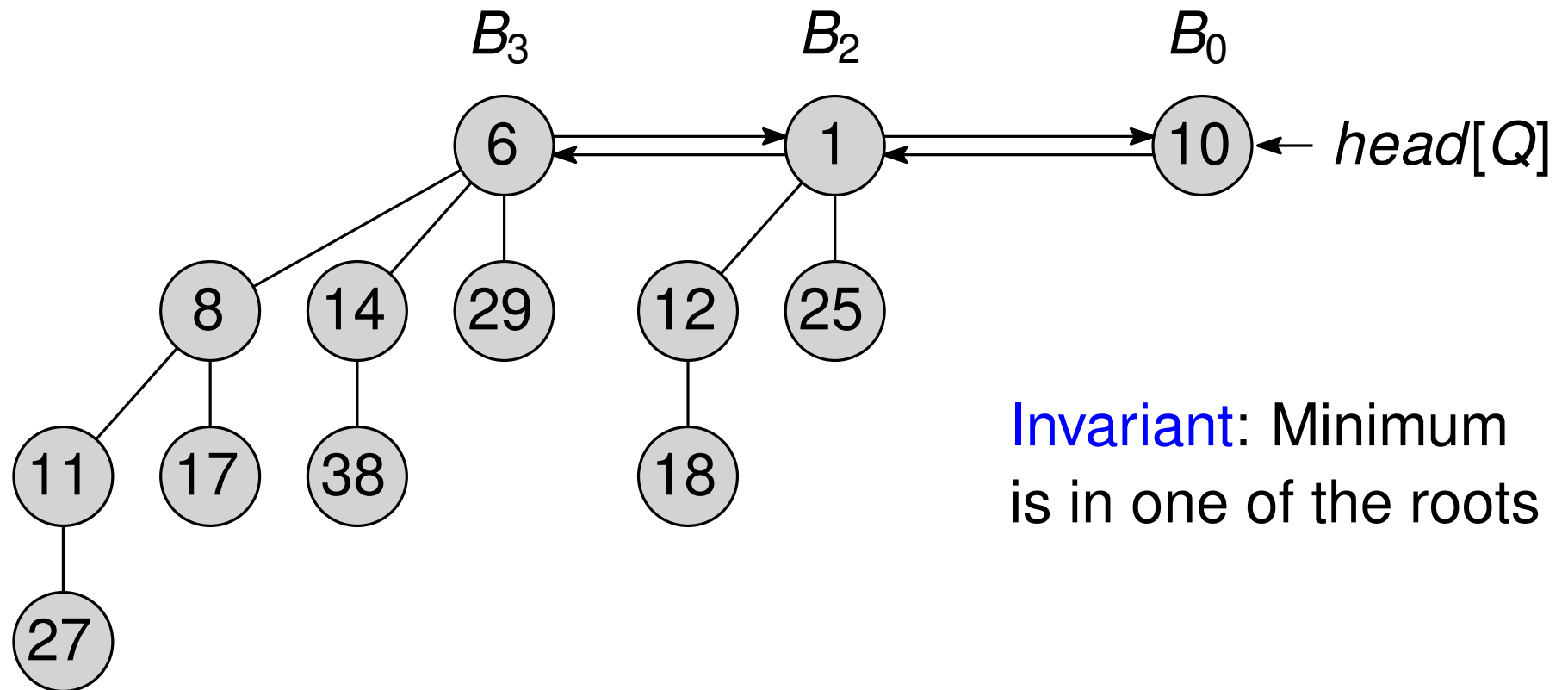
- Each tree is heap-ordered
- At most **one** tree B_k , for $k = 0, 1, 2, \dots, \lfloor \log n \rfloor$



MINIMUM(Q)

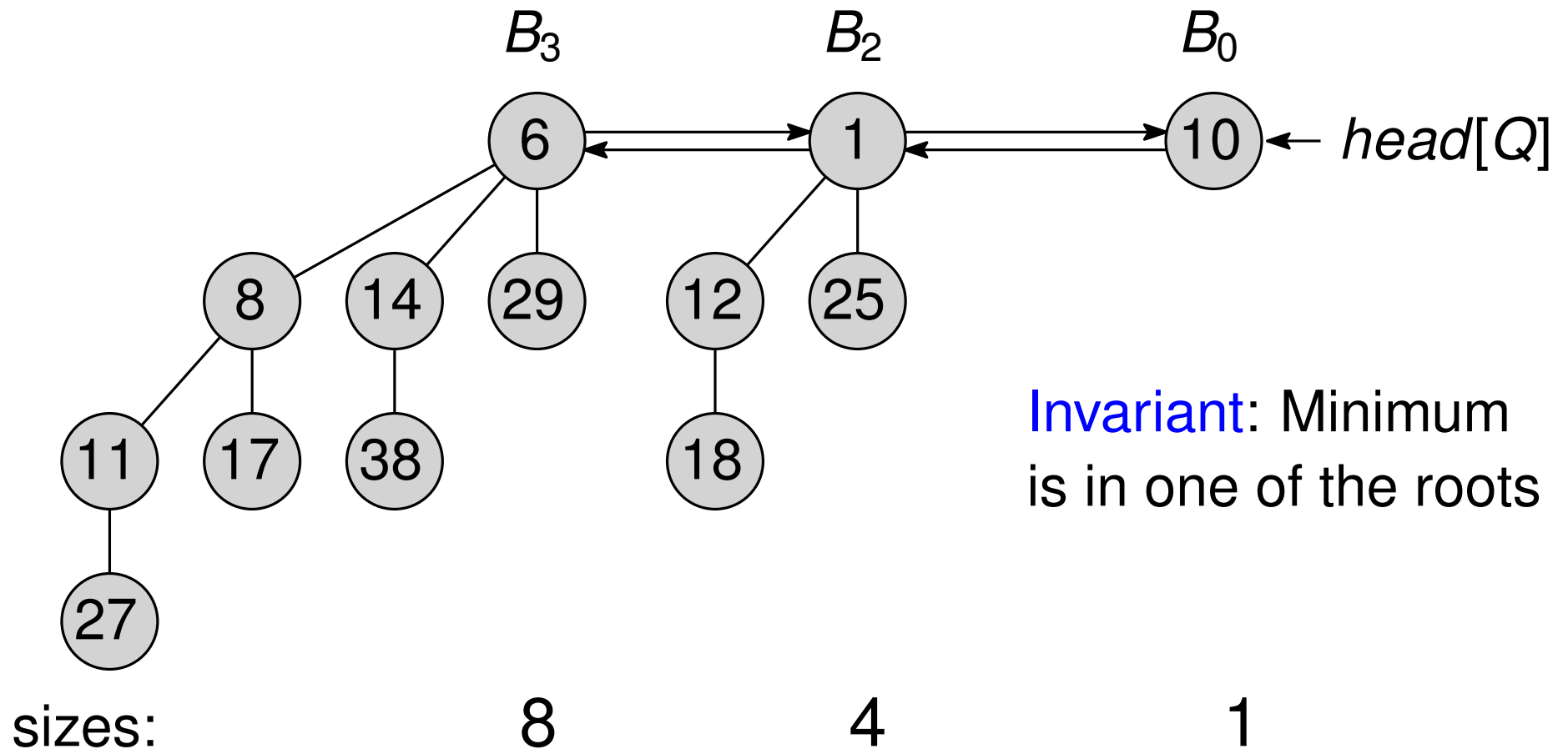


MINIMUM(Q)

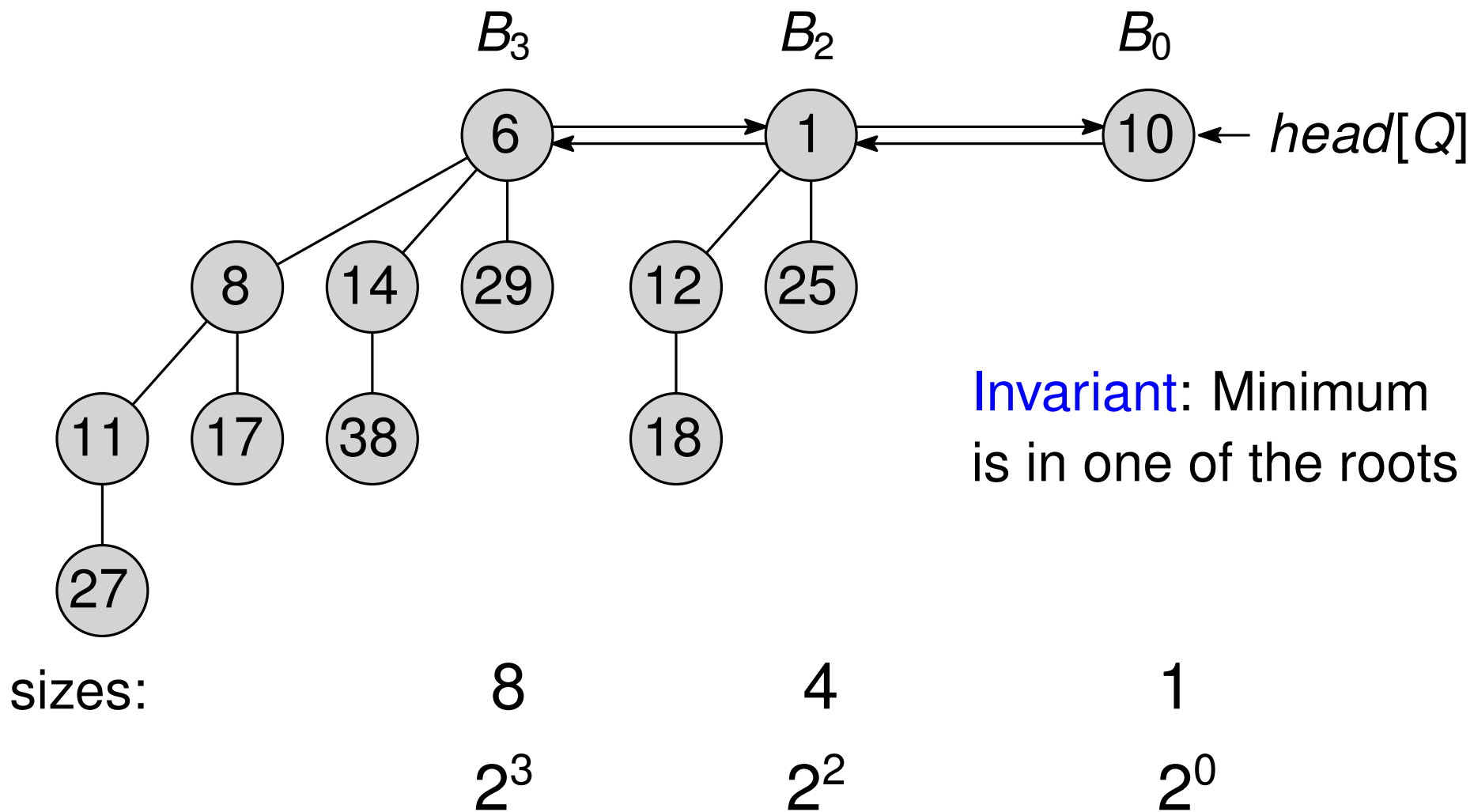


Invariant: Minimum is in one of the roots

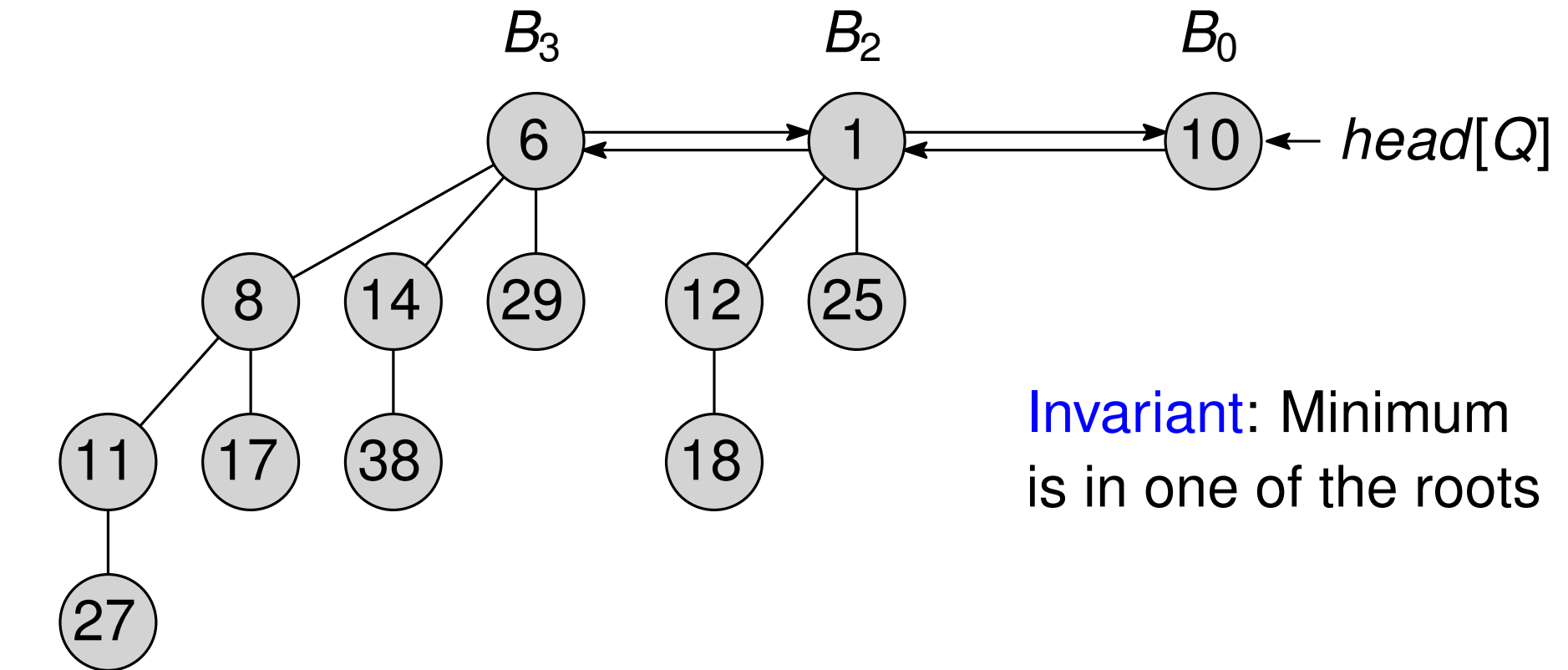
MINIMUM(Q)



MINIMUM(Q)



MINIMUM(Q)

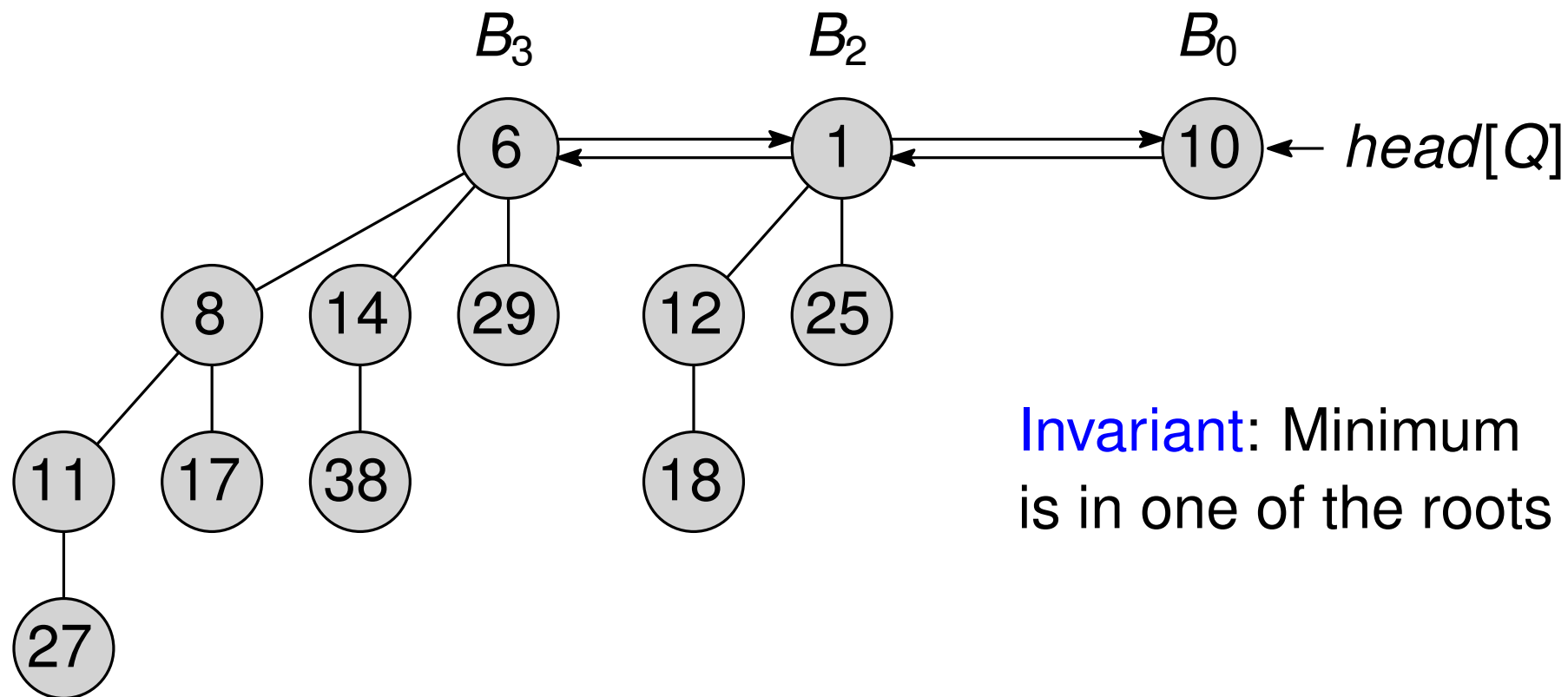


Invariant: Minimum is in one of the roots

sizes: 8 4 1

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

MINIMUM(Q)



Invariant: Minimum is in one of the roots

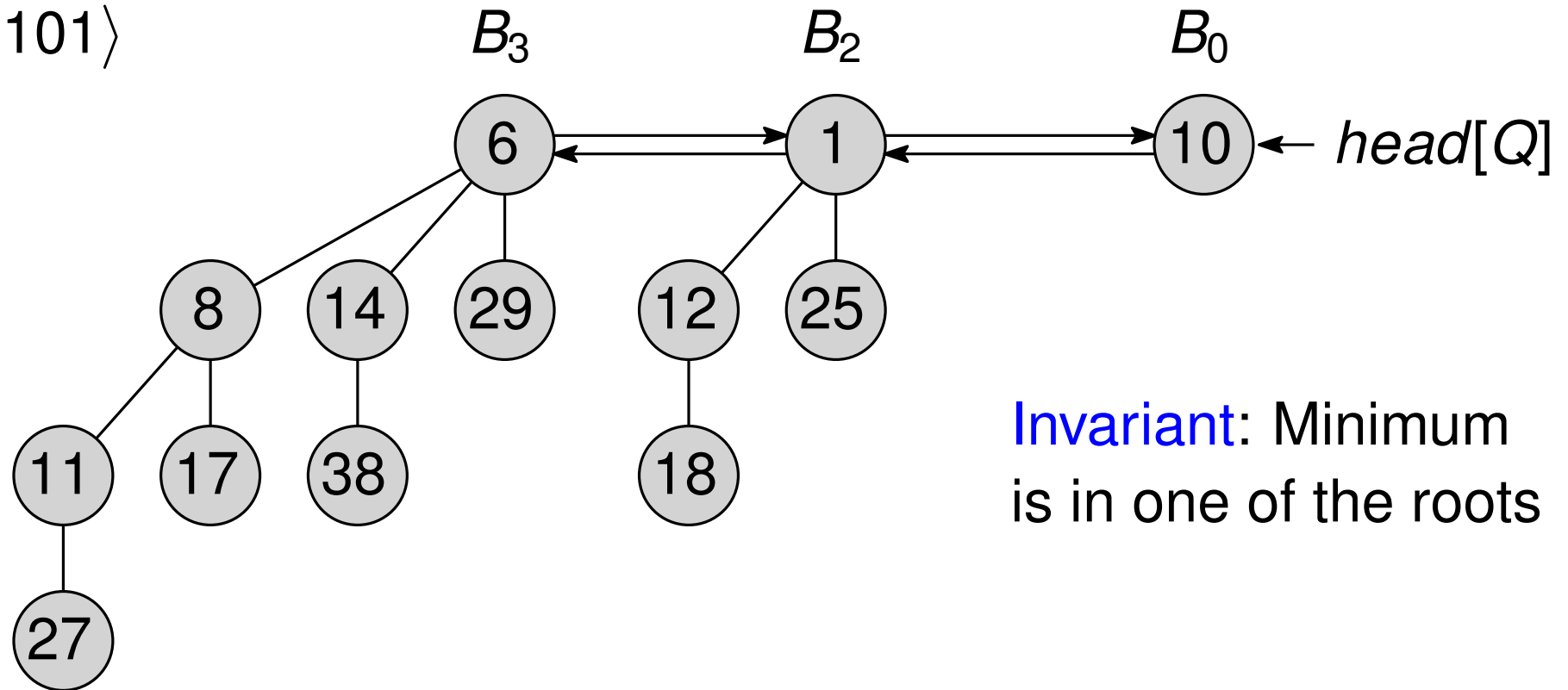
sizes:

$$N = 13 = 8 + 4 + 1$$
$$N = 13_{10} = 2^3 + 2^2 + 2^0$$

$$N = 13_{10} = 1101_2$$

MINIMUM(Q)

$\langle 1101 \rangle$



Invariant: Minimum is in one of the roots

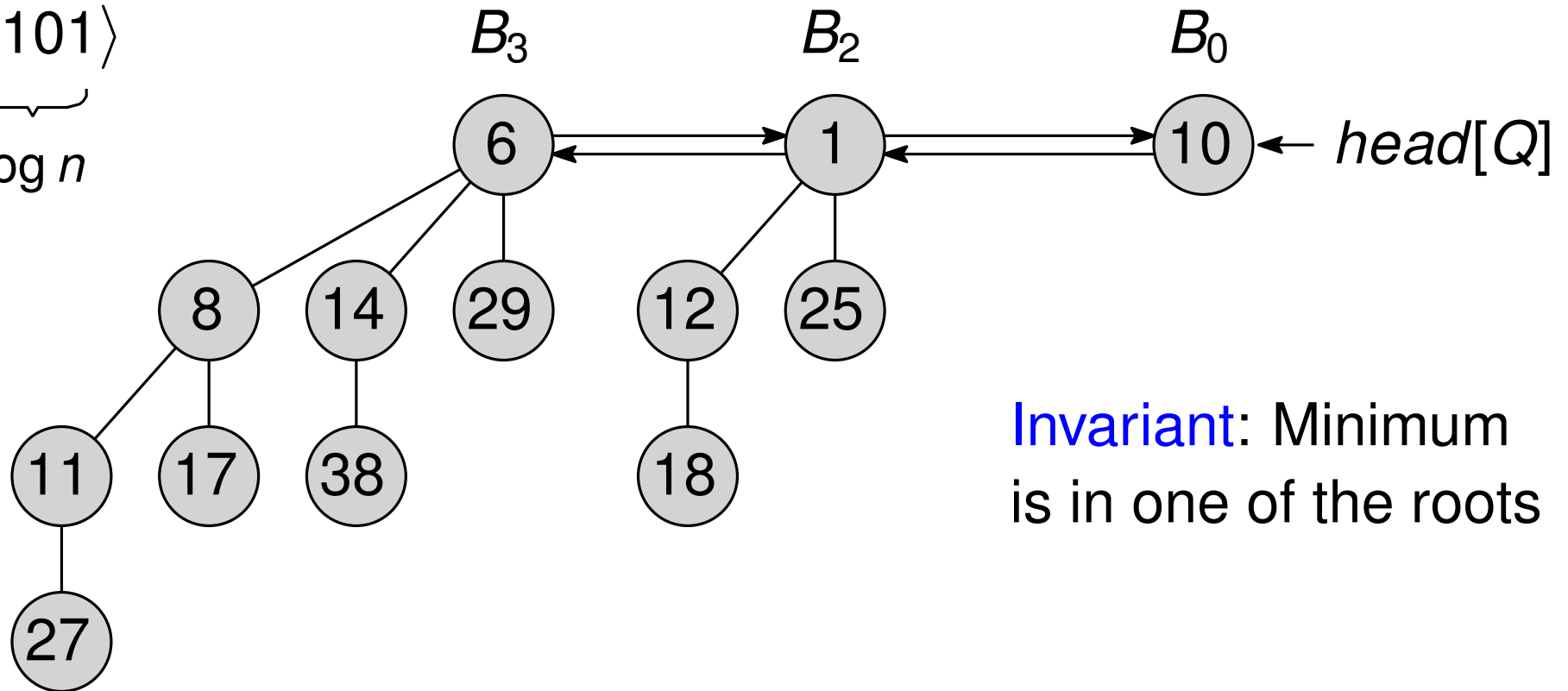
sizes:

$$N = 13 = 8 + 4 + 1$$

$$N = 13_{10} = 1101_2$$

MINIMUM(Q)

$\langle 1101 \rangle$
└───┘
 $\log n$



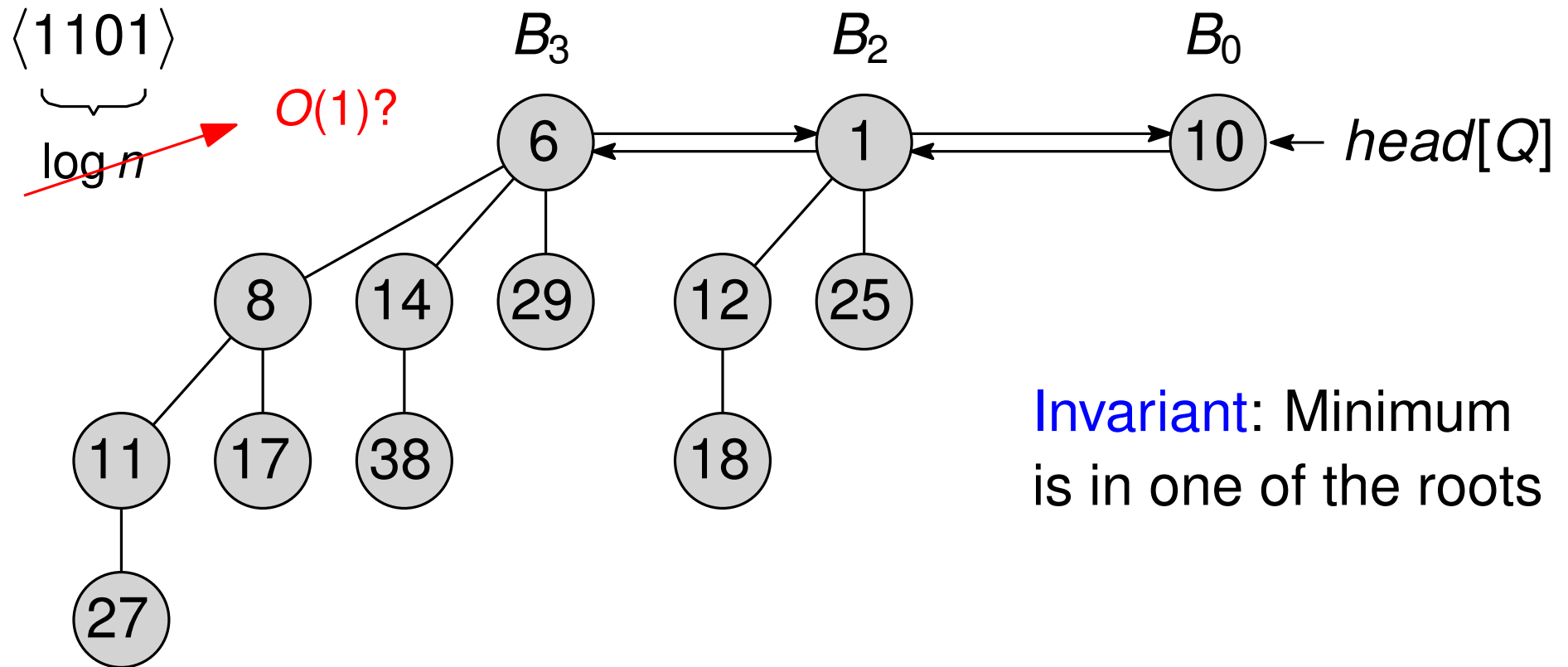
Invariant: Minimum is in one of the roots

sizes: 8 4 1

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

$$N = 13_{10} = 1101_2$$

MINIMUM(Q)



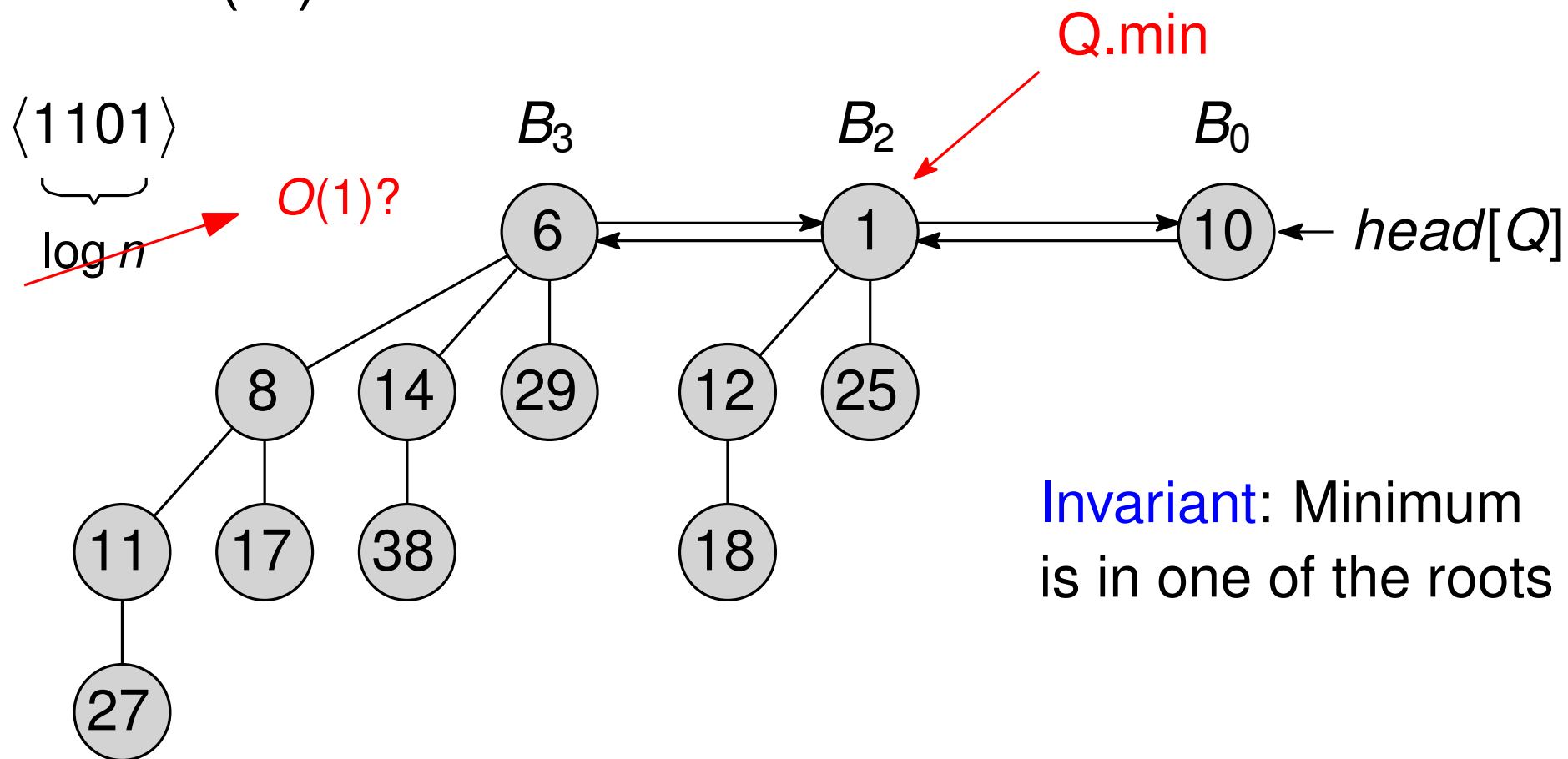
Invariant: Minimum is in one of the roots

sizes:

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

$$N = 13_{10} = 1101_2$$

MINIMUM(Q)



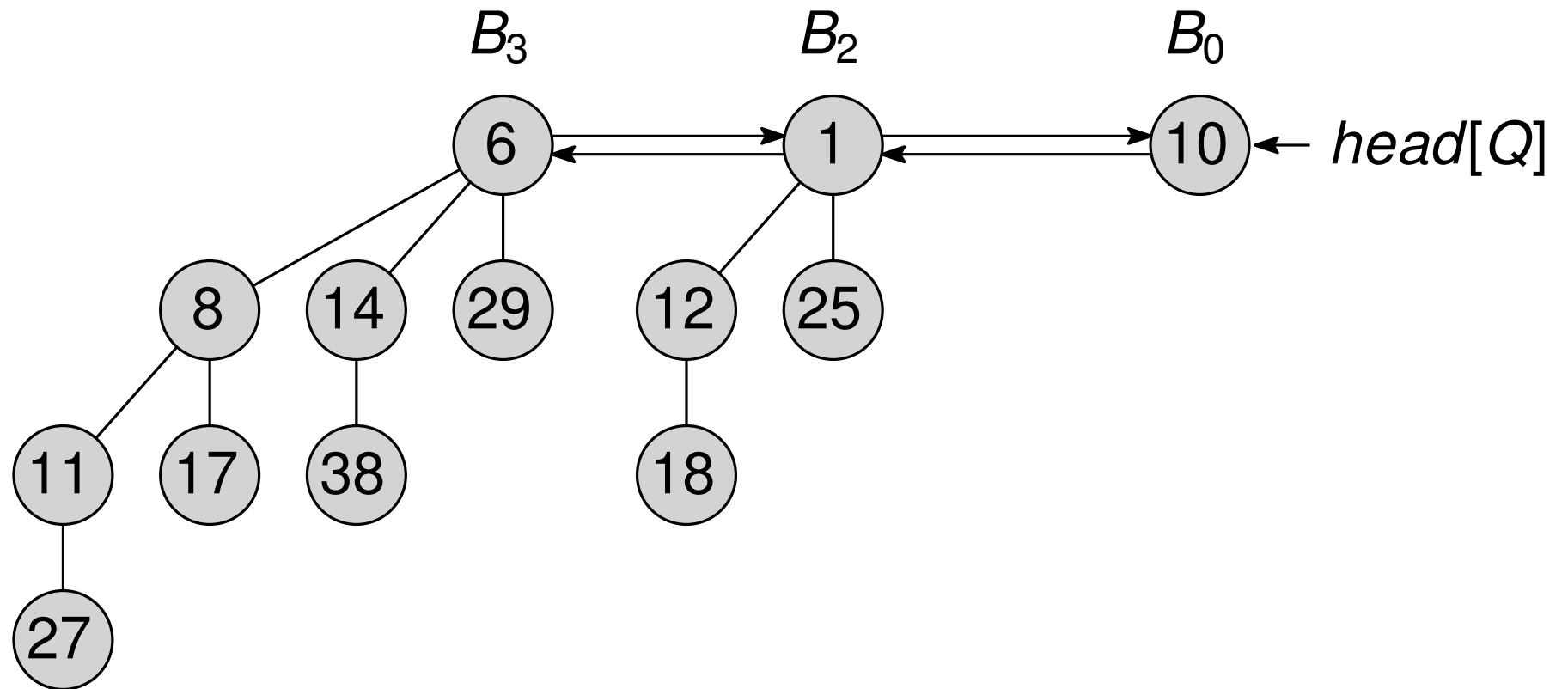
Invariant: Minimum is in one of the roots

sizes: 8 4 1

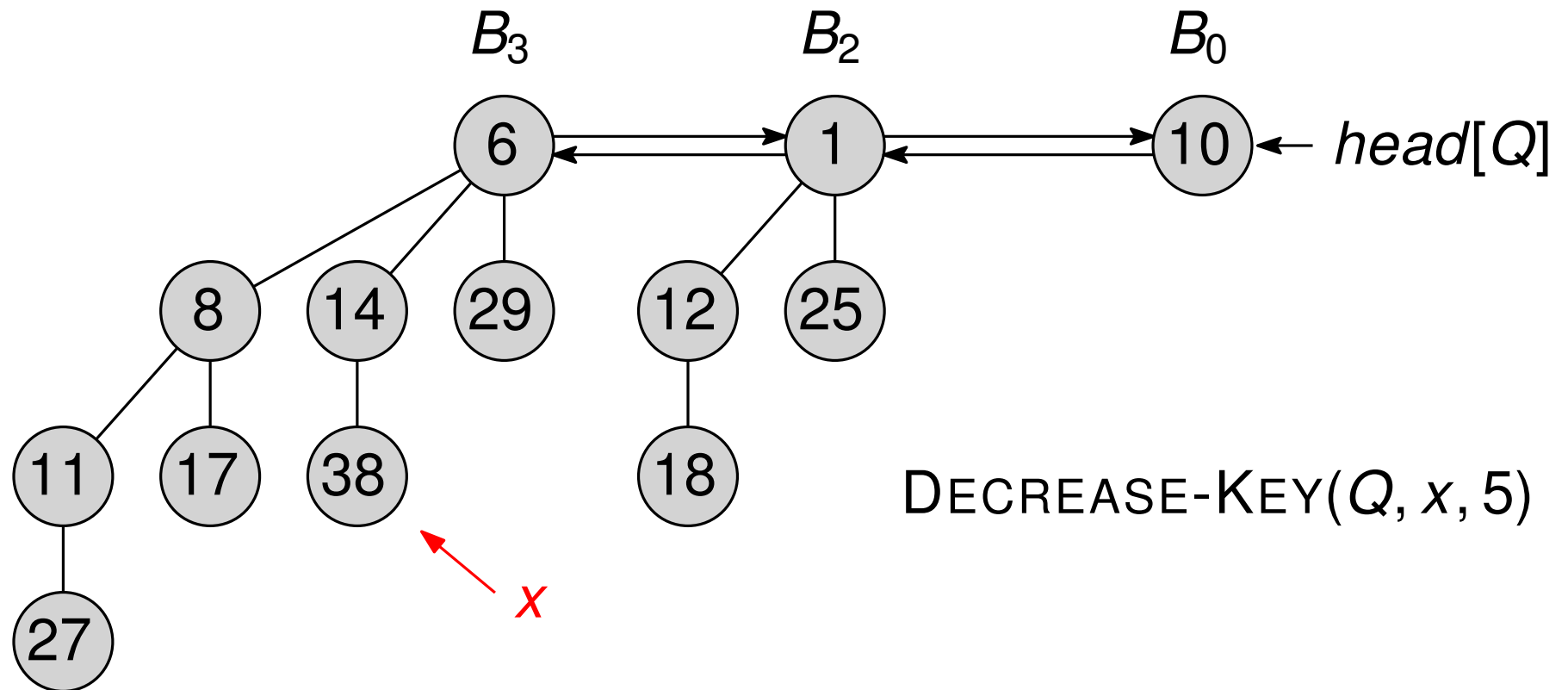
$N = 13 = 2^3 + 2^2 + 2^0$

$N = 13_{10} = 1101_2$

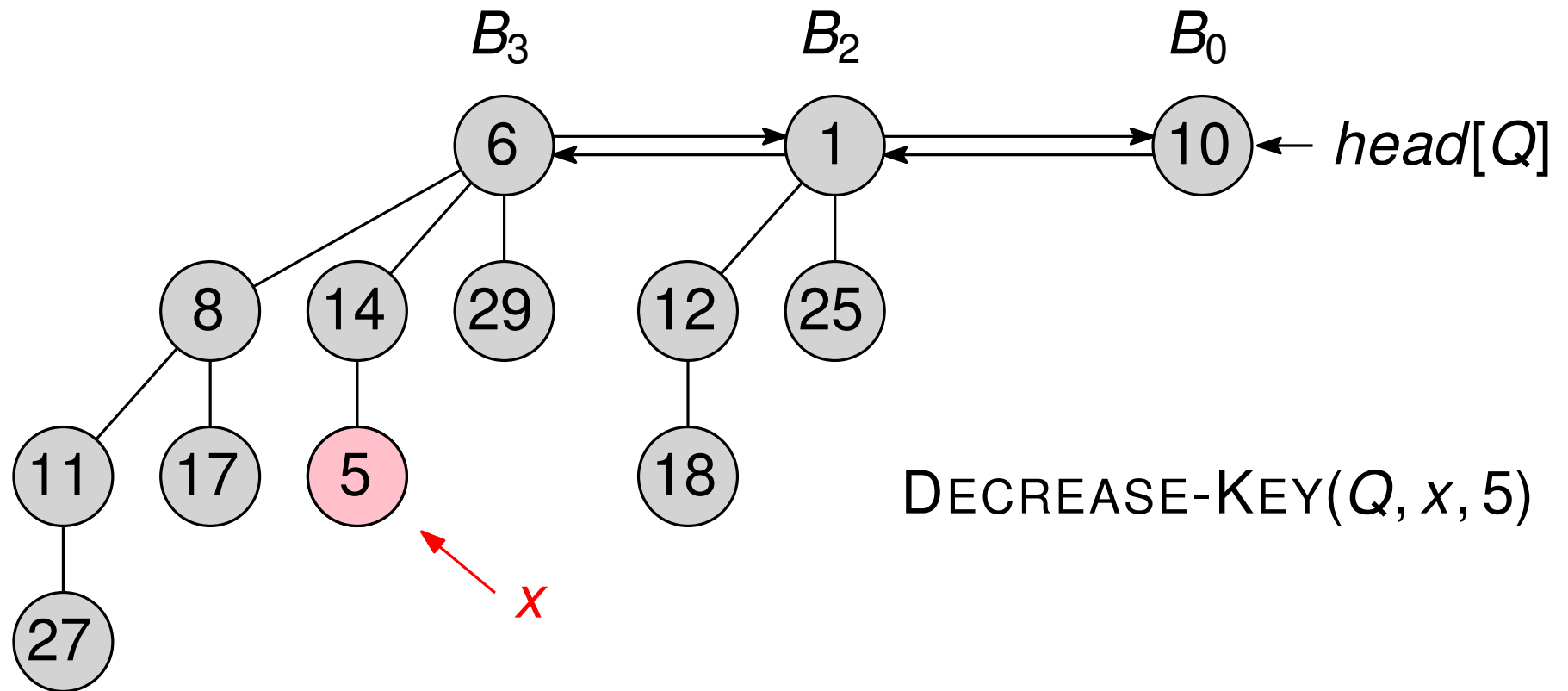
DECREASE-KEY(Q, x, k)



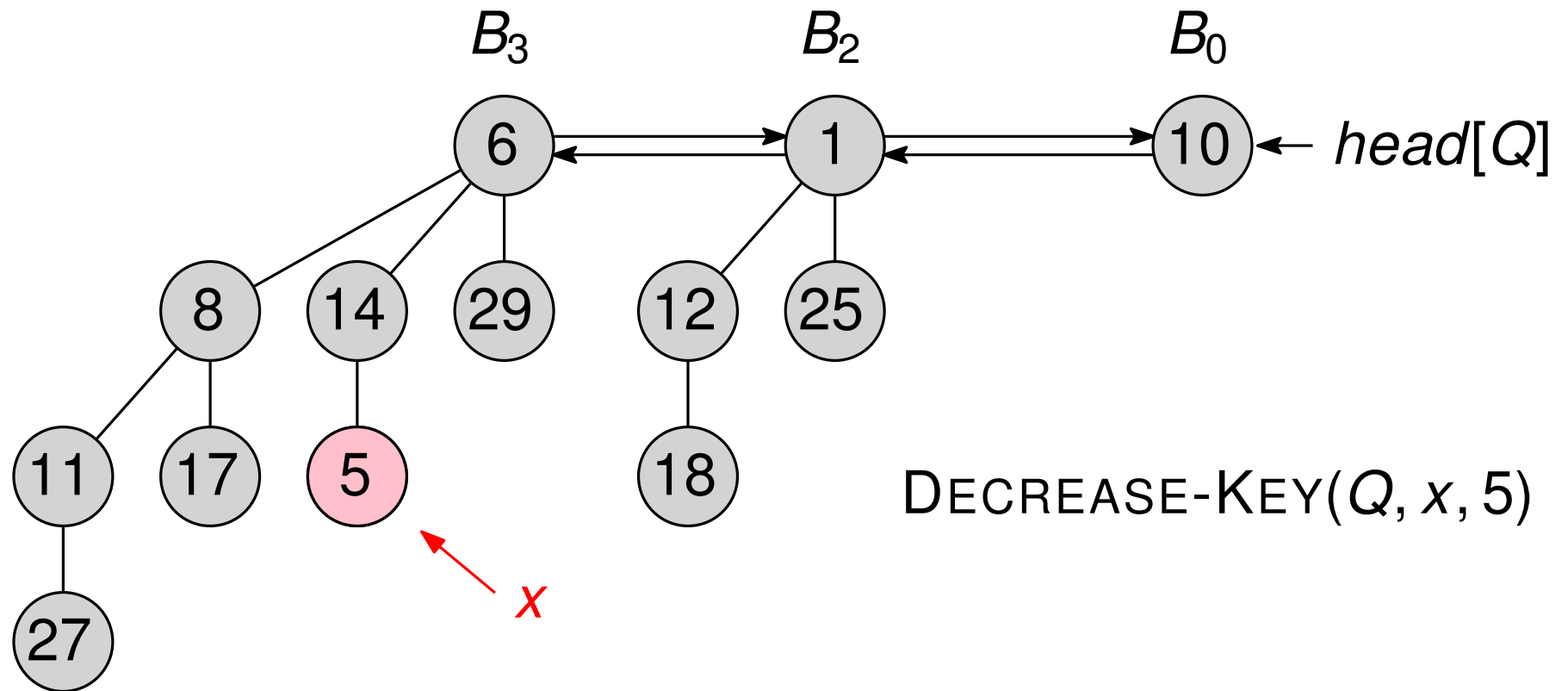
DECREASE-KEY(Q, x, k)



DECREASE-KEY(Q, x, k)

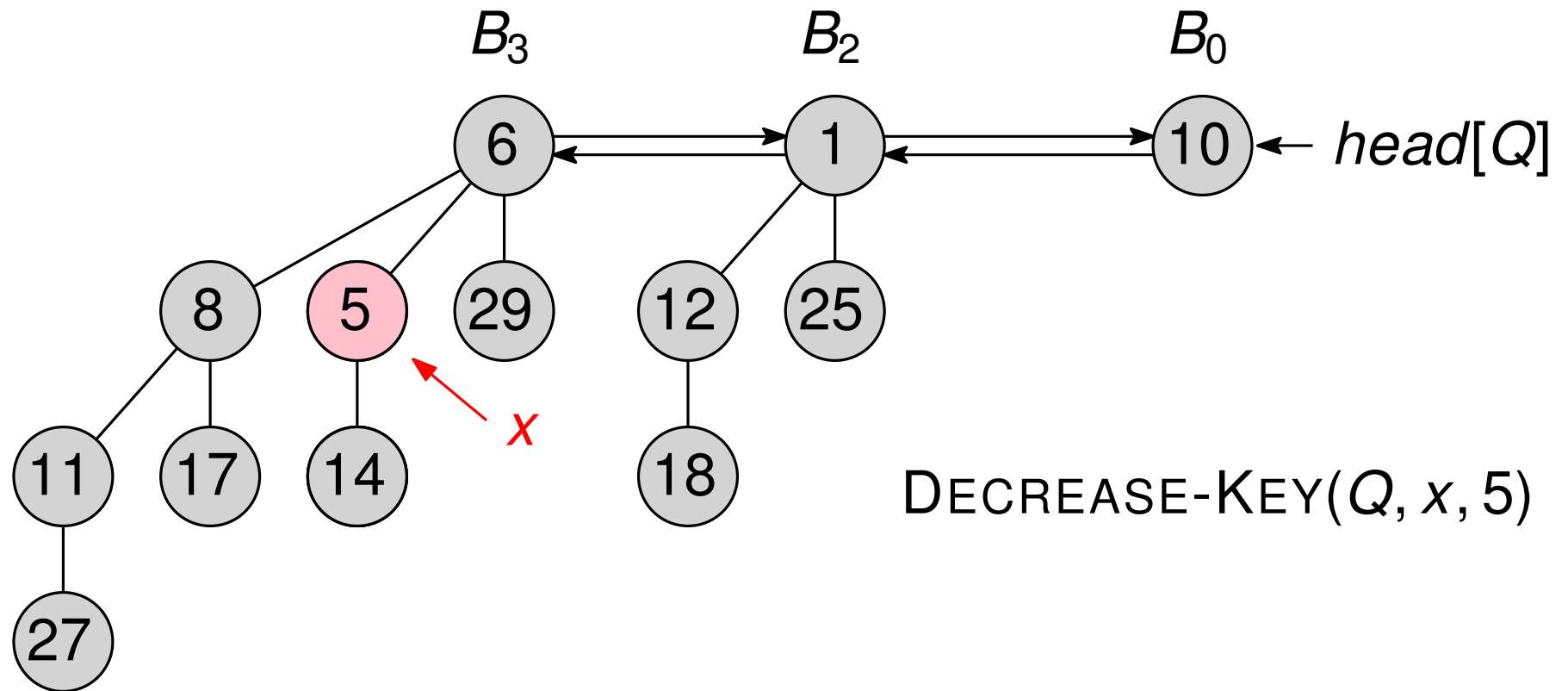


DECREASE-KEY(Q, x, k)



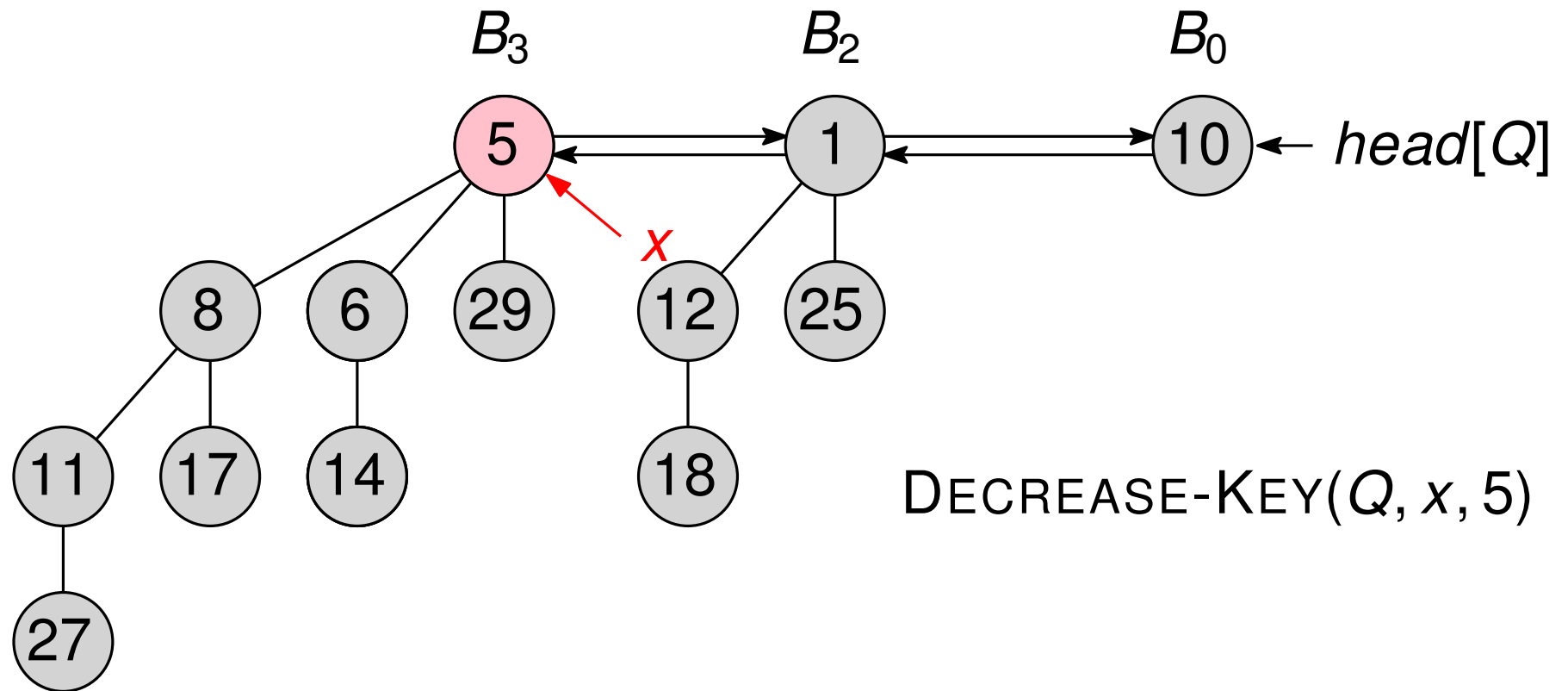
- Fix heap order

DECREASE-KEY(Q, x, k)



- Fix heap order

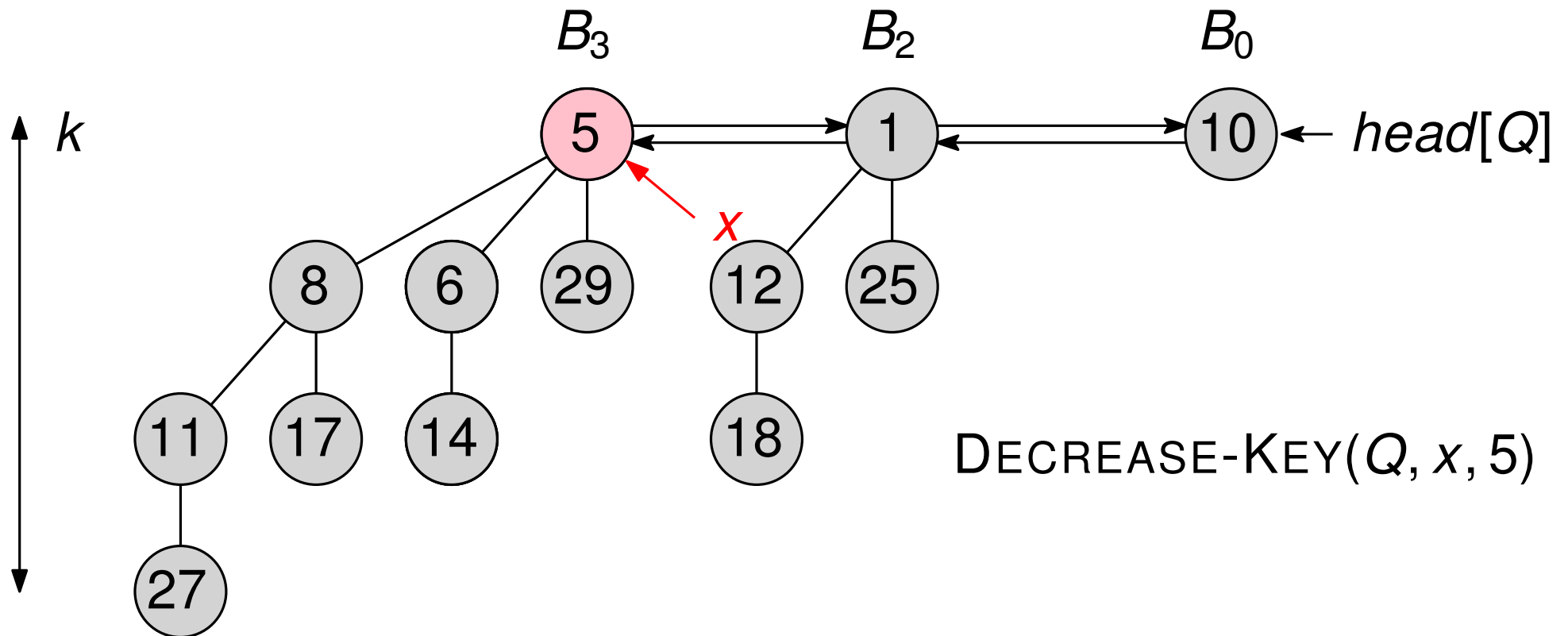
DECREASE-KEY(Q, x, k)



- Fix heap order

DECREASE-KEY(Q, x, k)

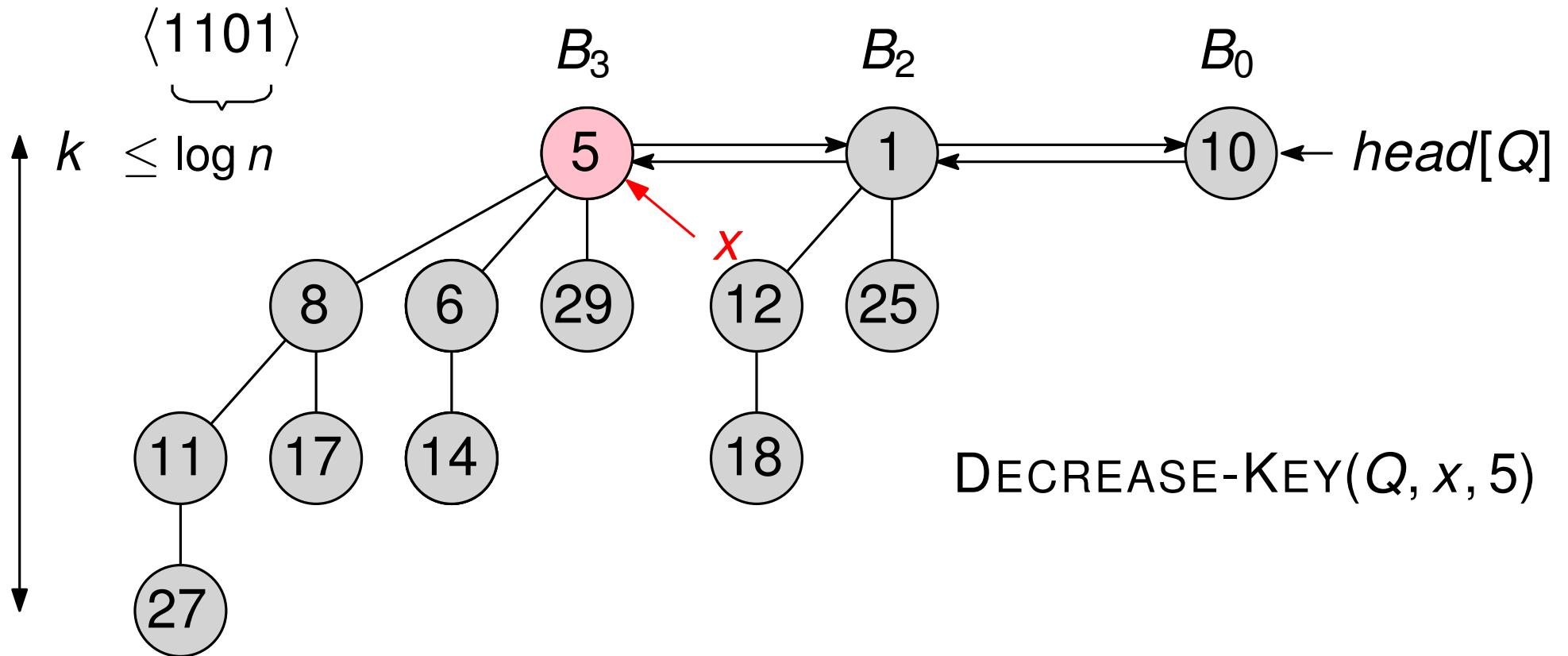
Depth of B_k is k



- Fix heap order

DECREASE-KEY(Q, x, k)

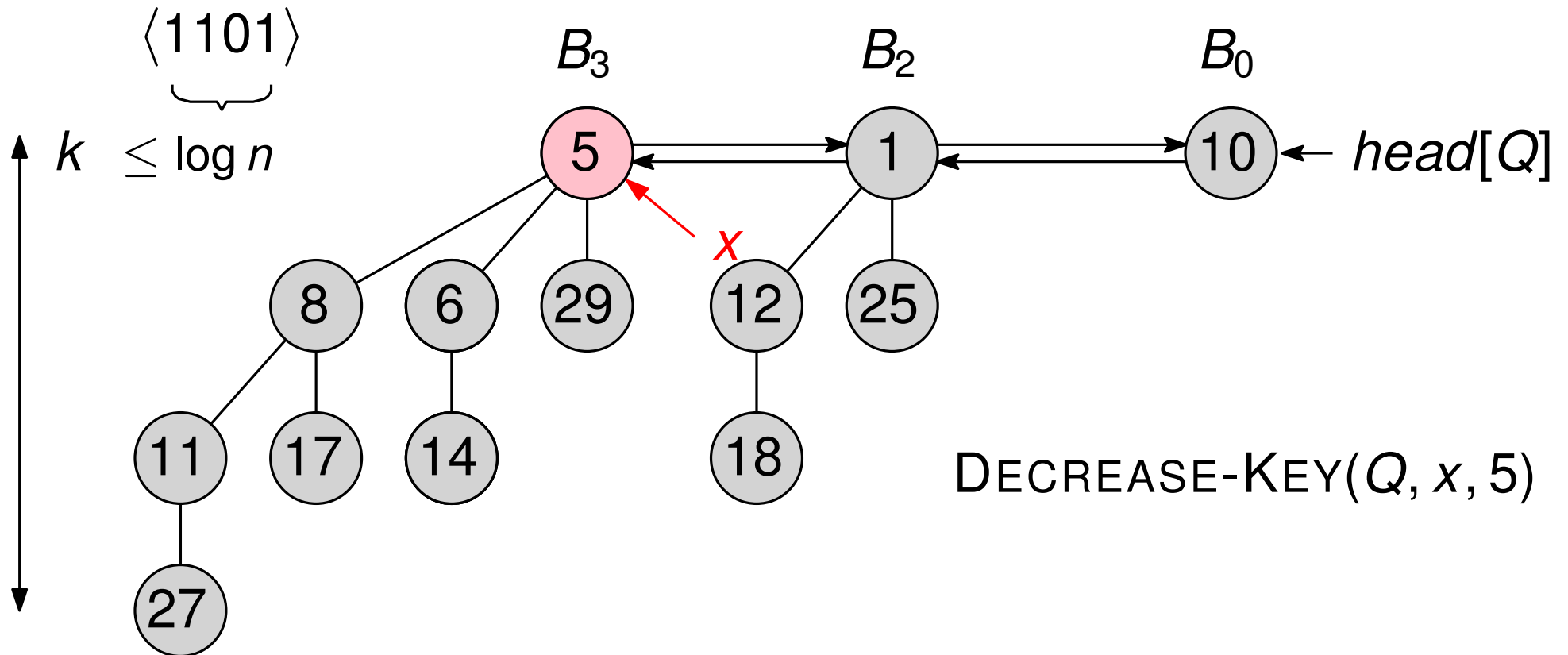
Depth of B_k is k



- Fix heap order

DECREASE-KEY(Q, x, k)

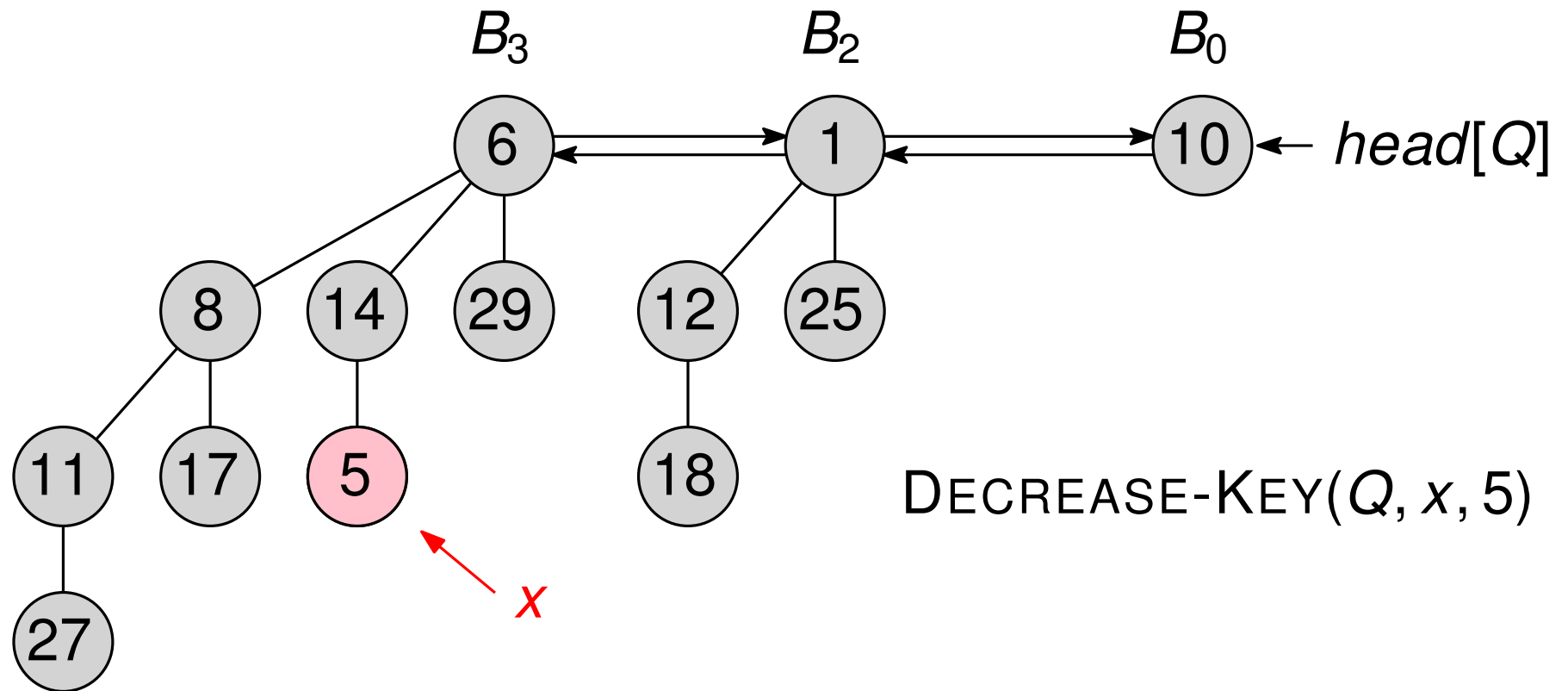
Depth of B_k is k



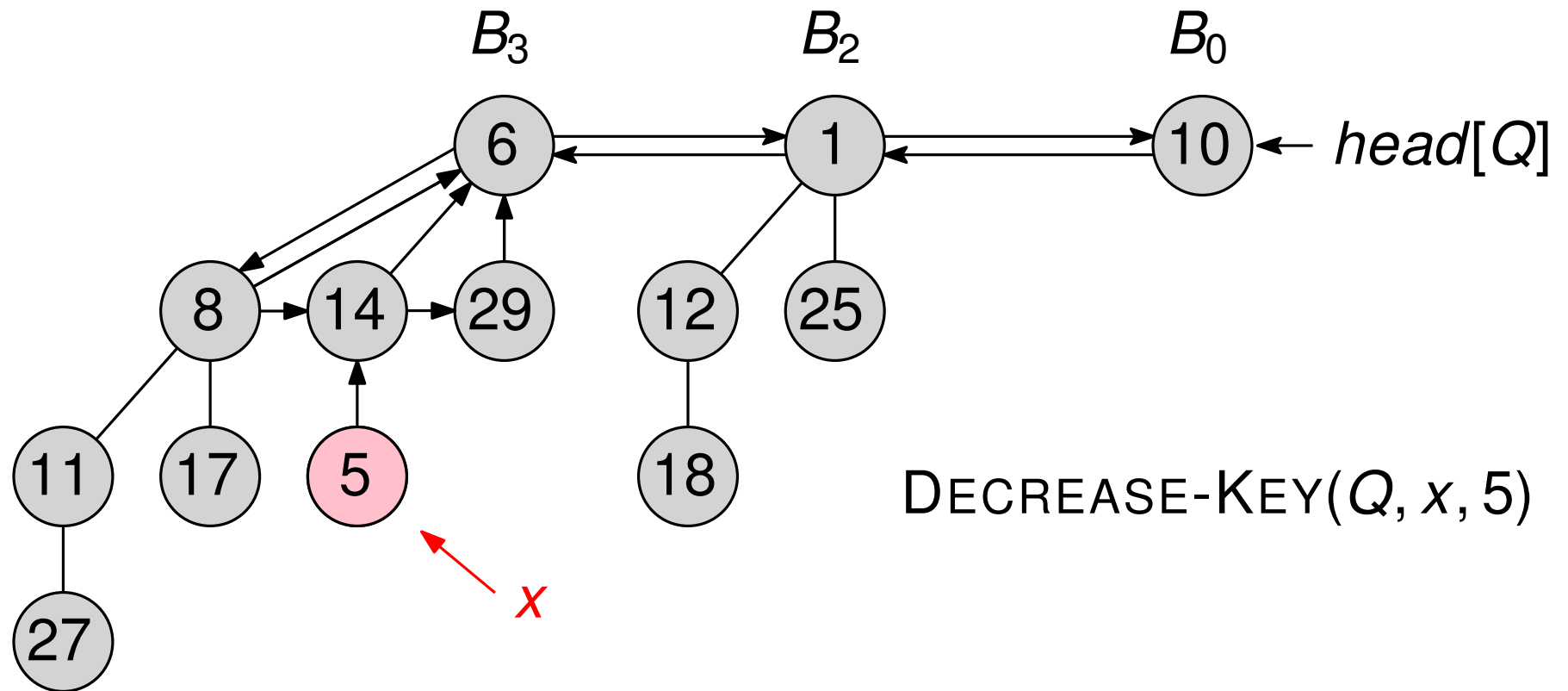
- Fix heap order

Can you implement DECREASE-KEY(Q, x, k) ?

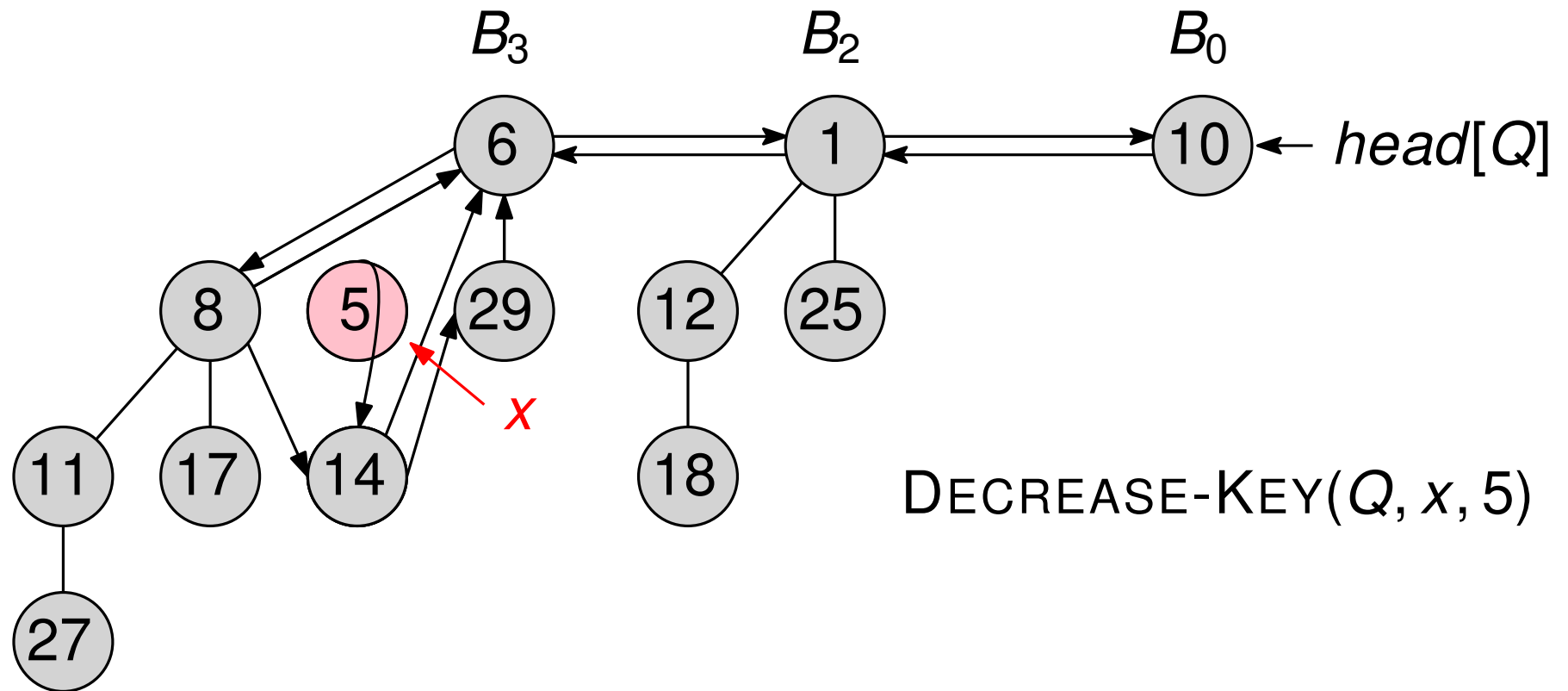
DECREASEKEY(Q, x, k)



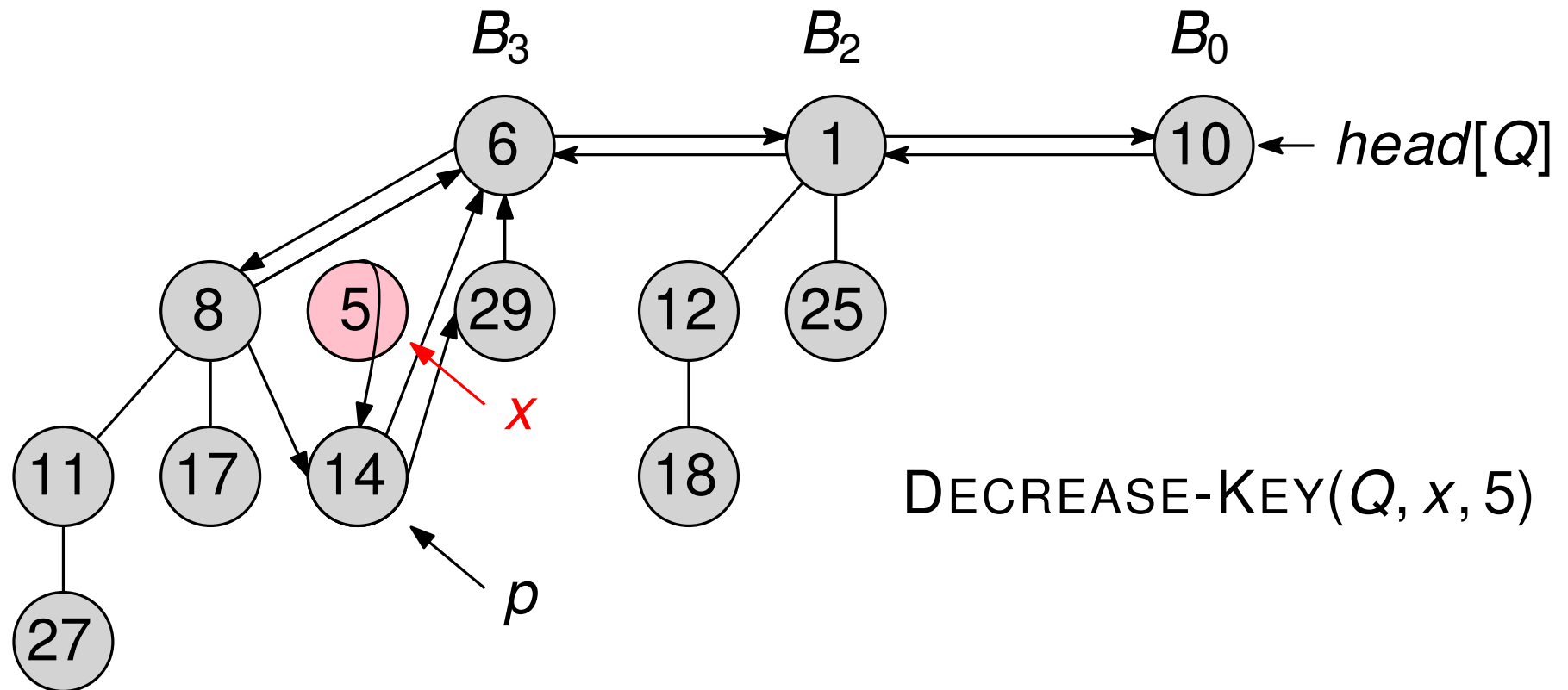
DECREASEKEY(Q, x, k)



DECREASEKEY(Q, x, k)



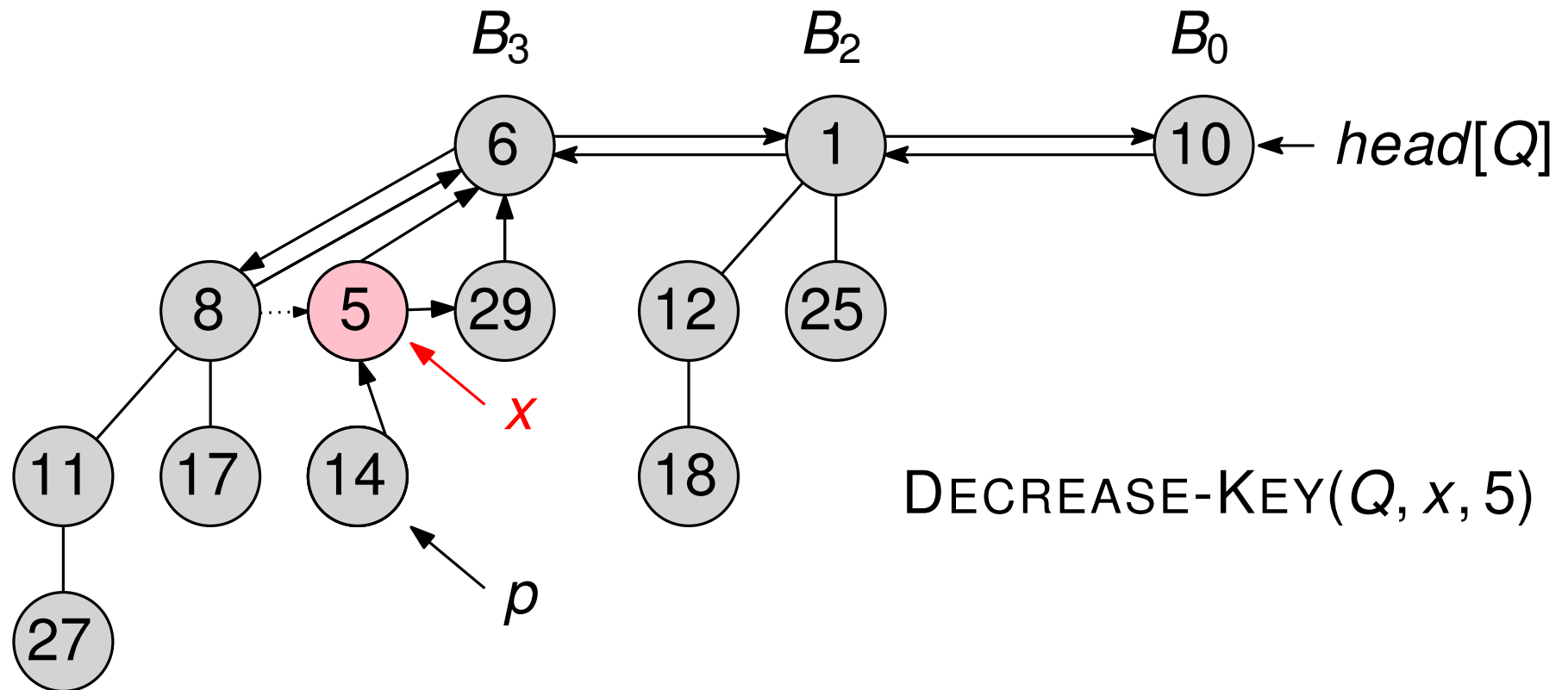
DECREASEKEY(Q, x, k)



DECREASE-KEY($Q, x, 5$)

- $p = x.parent$
- $x.parent = p.parent$
- $p.parent = x$
- $x.sibling = p.sibling$
- $???.sibling = x$

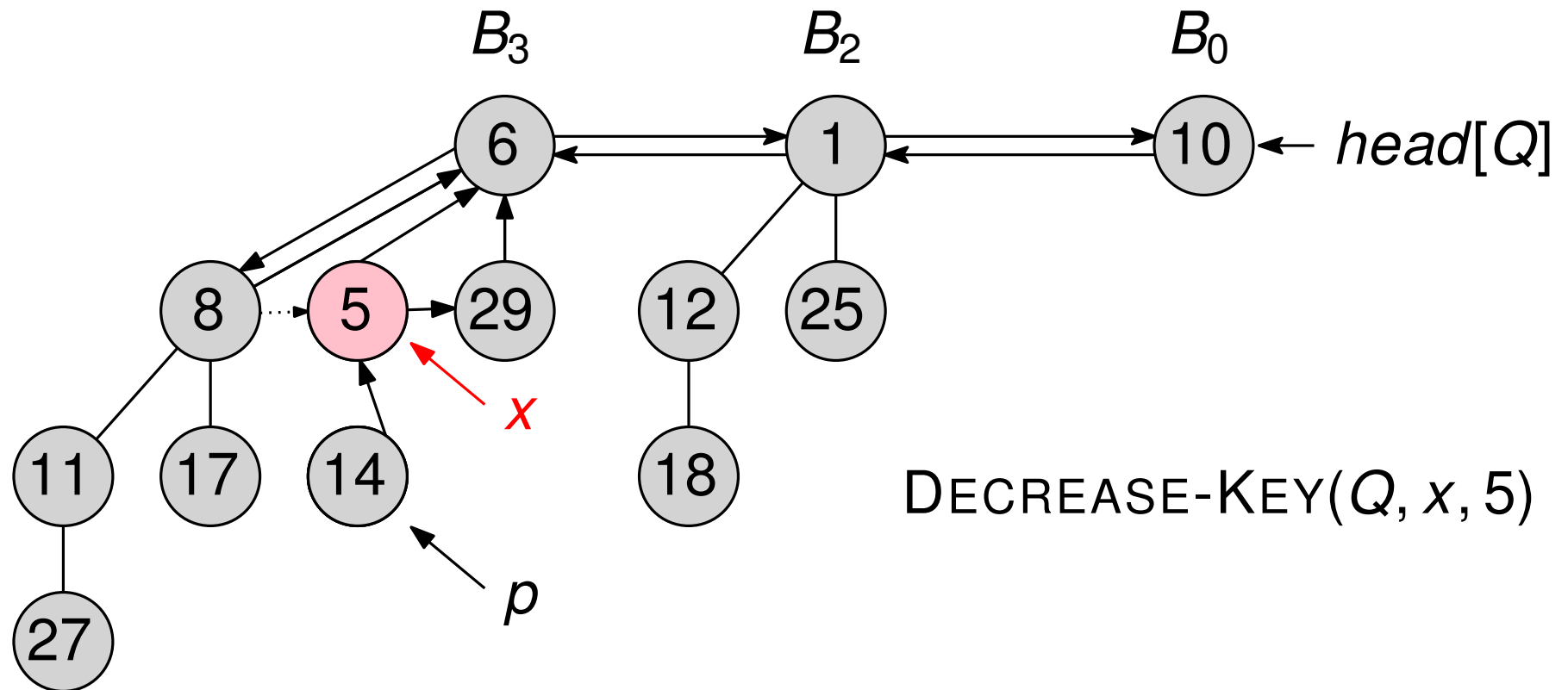
DECREASEKEY(Q, x, k)



DECREASE-KEY($Q, x, 5$)

- $p = x.parent$
- $x.parent = p.parent$
- $p.parent = x$
- $x.sibling = p.sibling$
- $???.sibling = x$

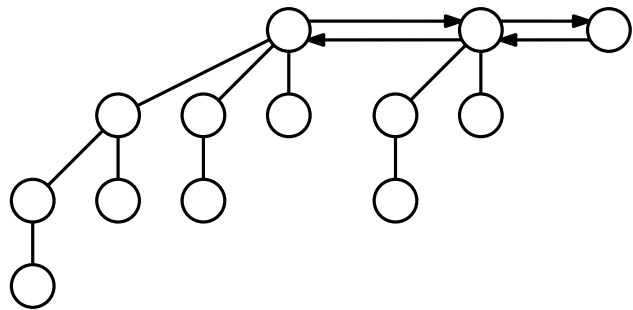
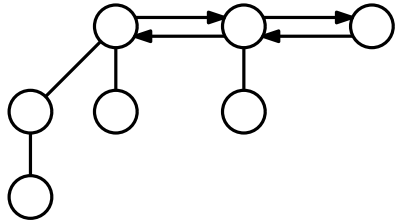
DECREASEKEY(Q, x, k)



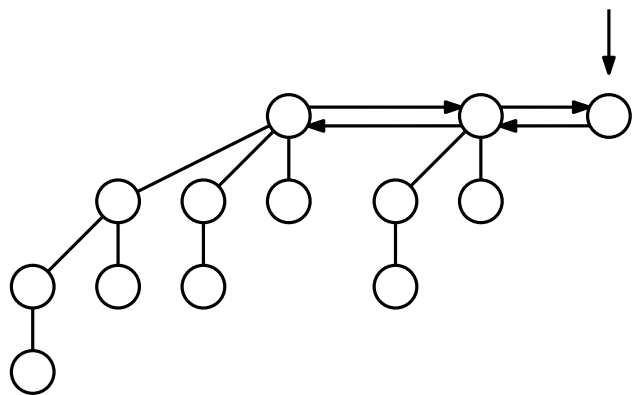
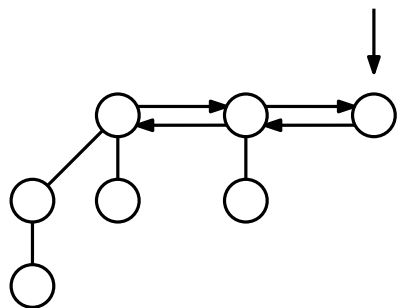
- $p = x.parent$
- $x.parent = p.parent$
- $p.parent = x$
- $x.sibling = p.sibling$
- $???.sibling = x$

NEED:
doubly-linked sibling list

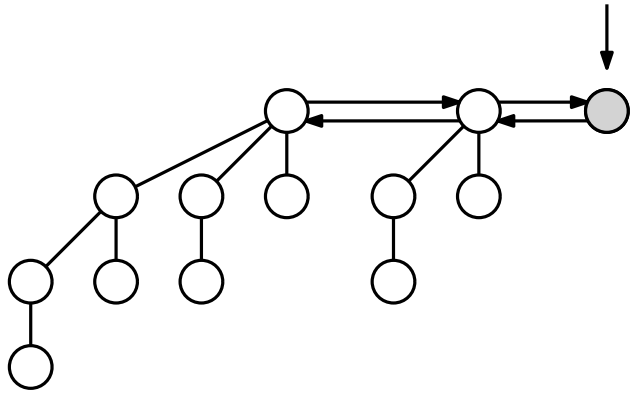
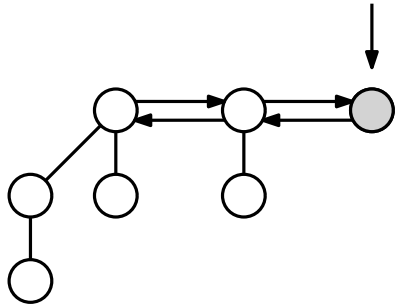
UNION(Q_1, Q_2)



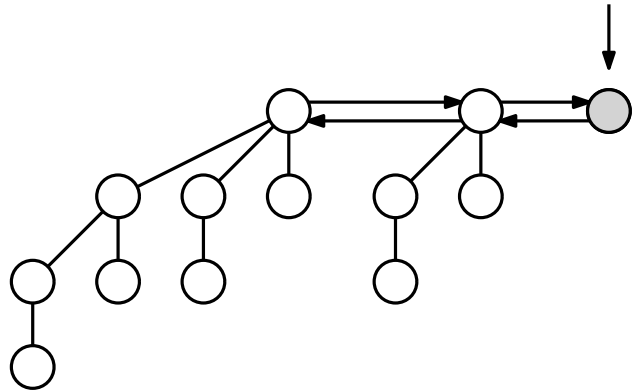
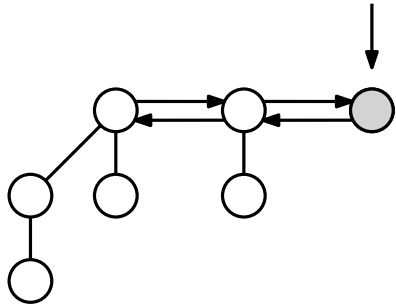
UNION(Q_1, Q_2)



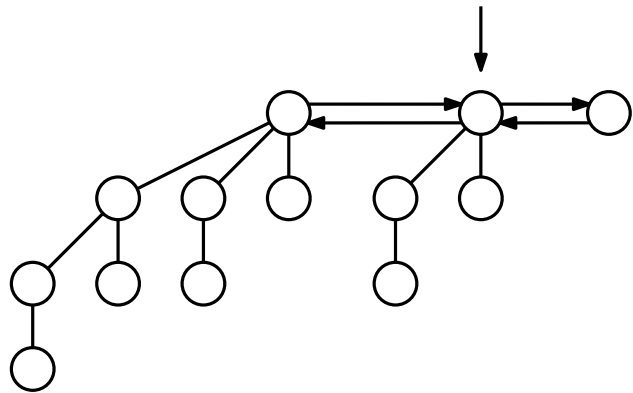
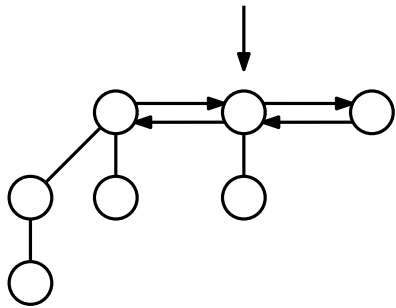
UNION(Q_1, Q_2)



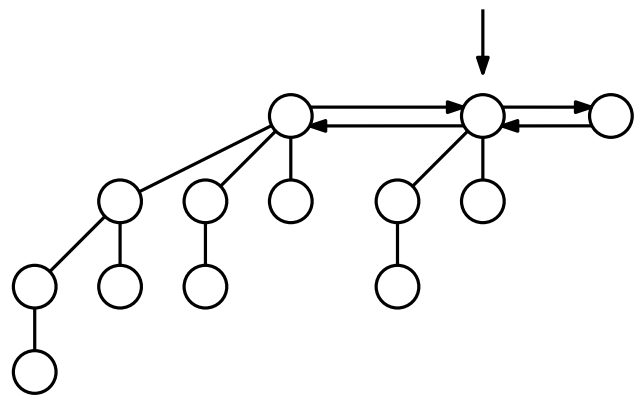
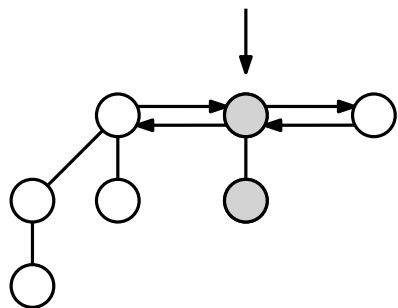
UNION(Q_1, Q_2)



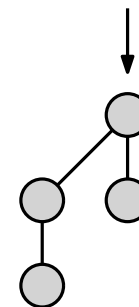
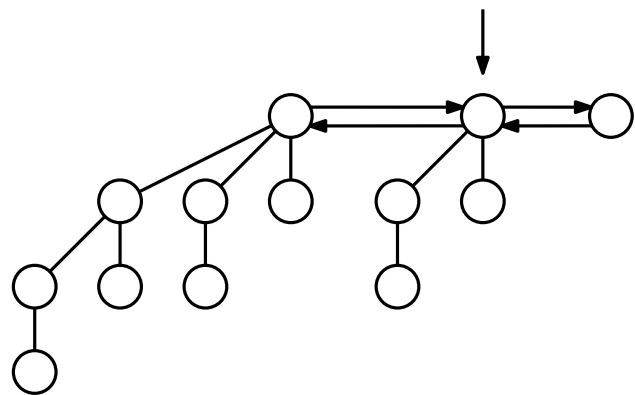
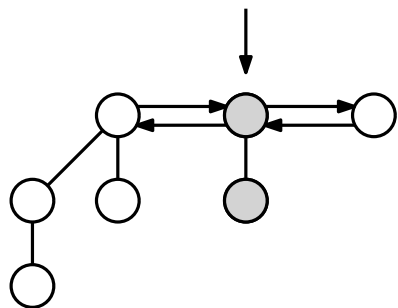
UNION(Q_1, Q_2)



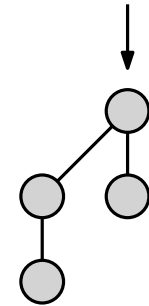
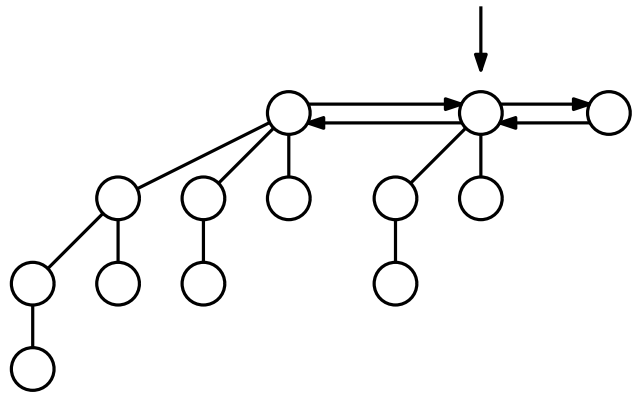
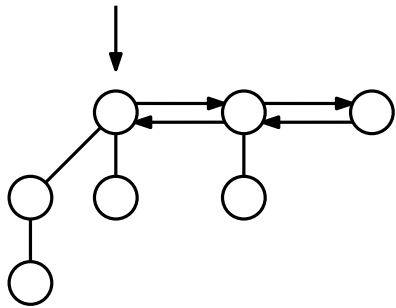
UNION(Q_1, Q_2)



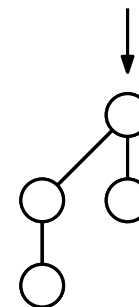
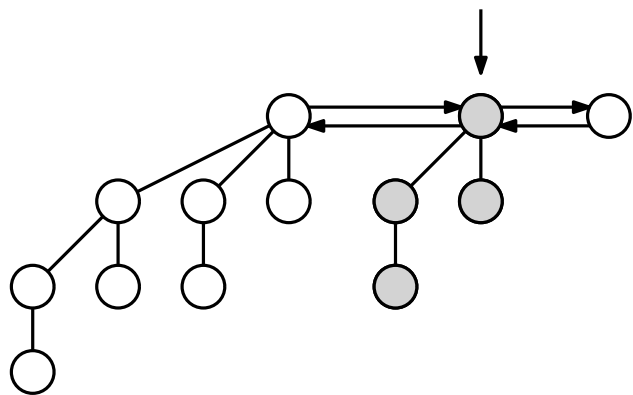
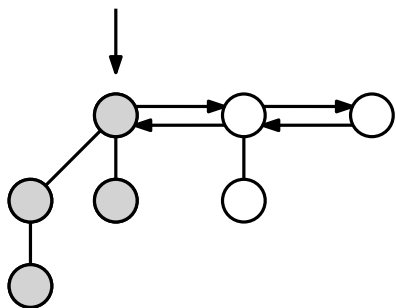
UNION(Q_1, Q_2)



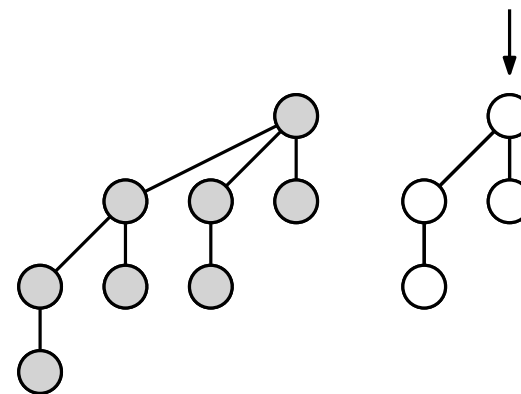
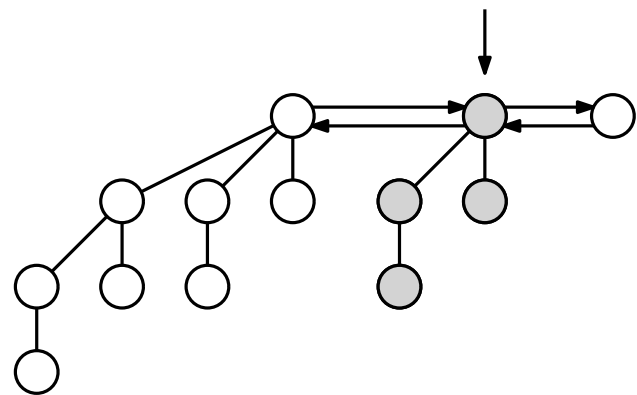
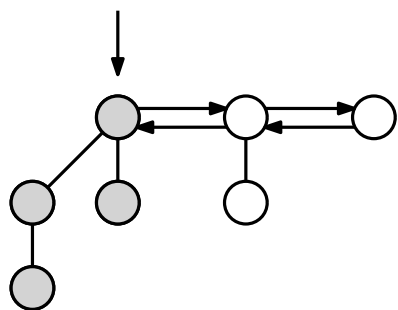
UNION(Q_1, Q_2)



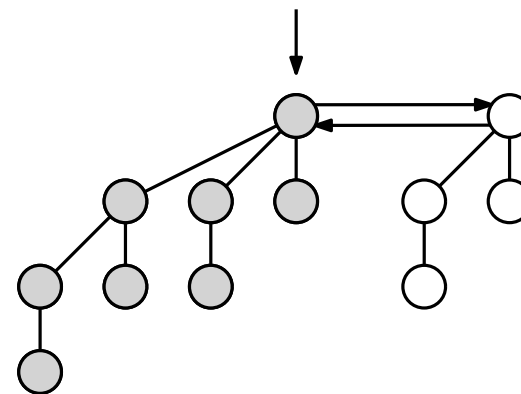
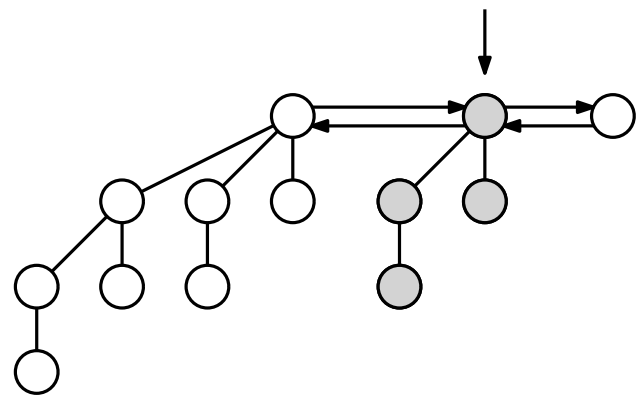
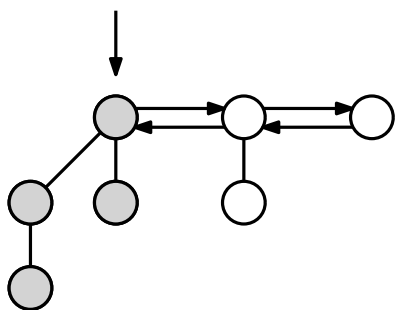
UNION(Q_1, Q_2)



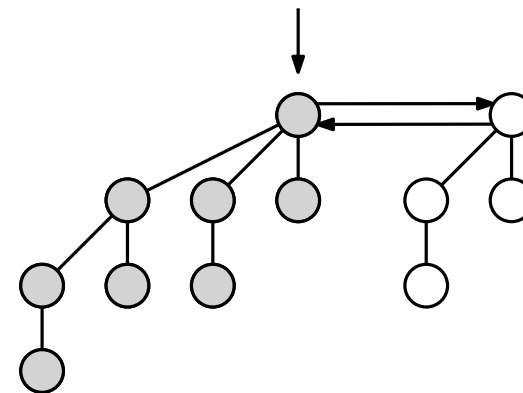
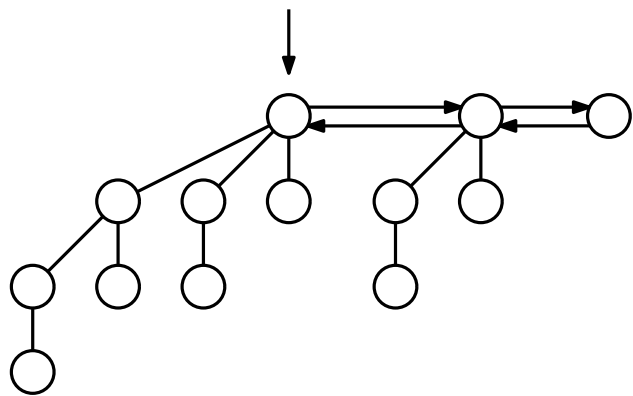
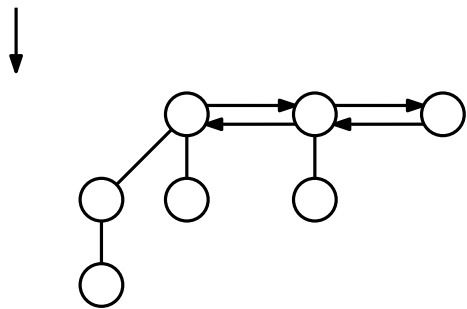
UNION(Q_1, Q_2)



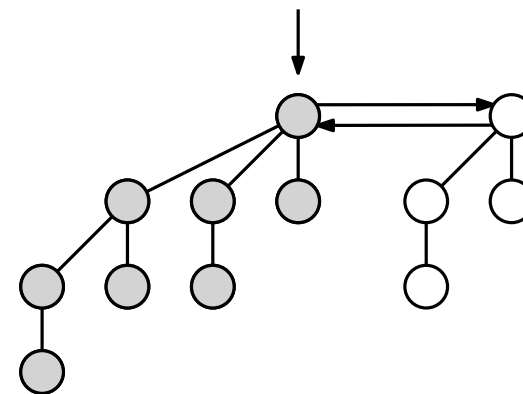
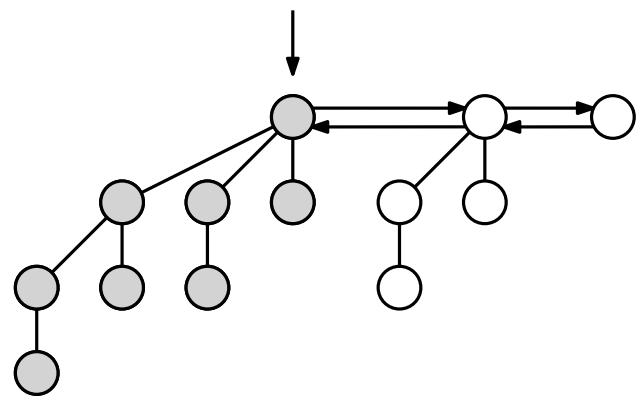
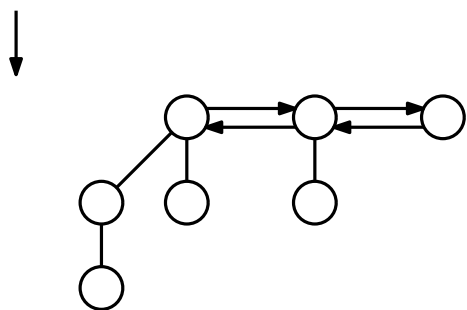
UNION(Q_1, Q_2)



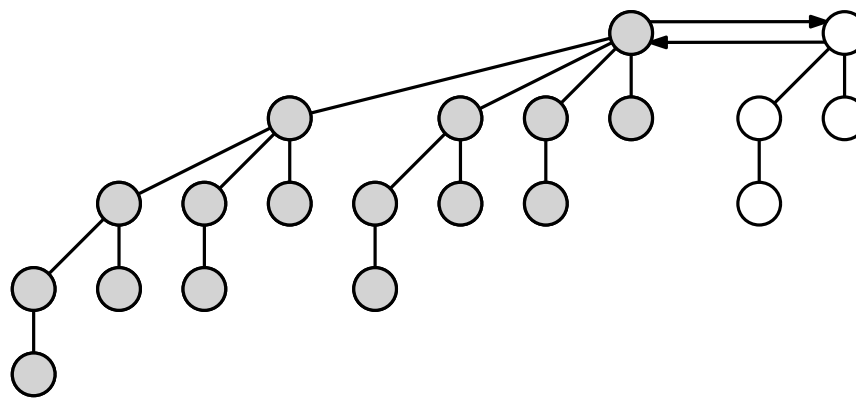
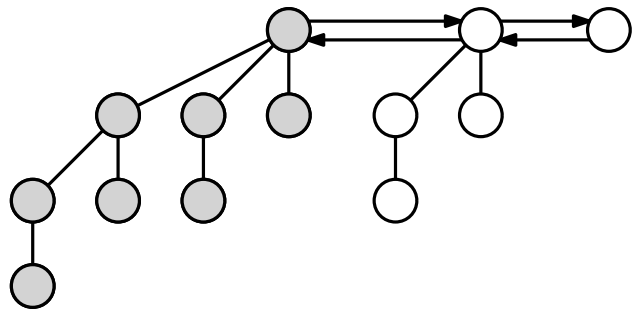
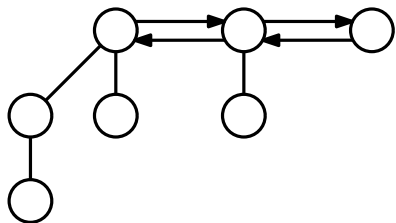
UNION(Q_1, Q_2)



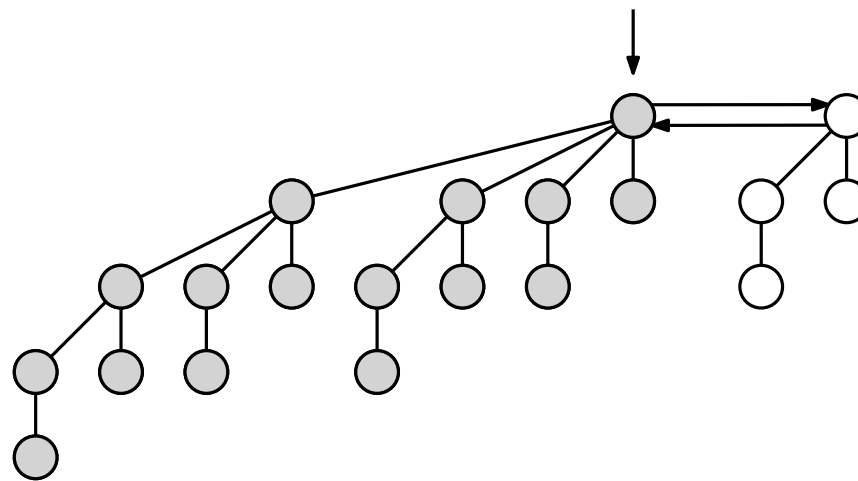
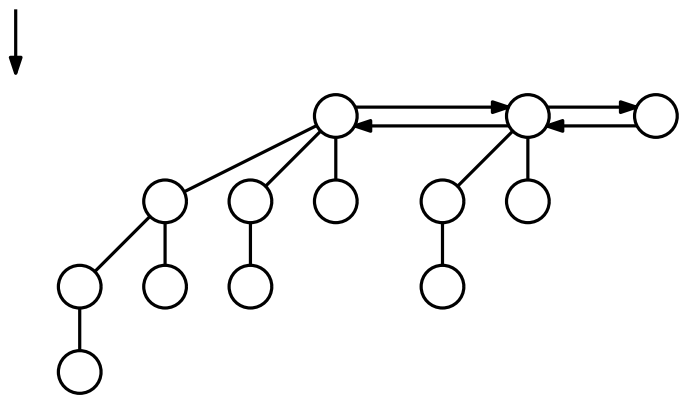
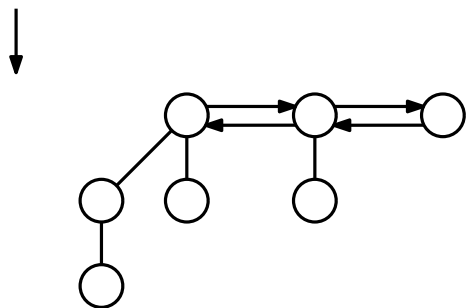
UNION(Q_1, Q_2)



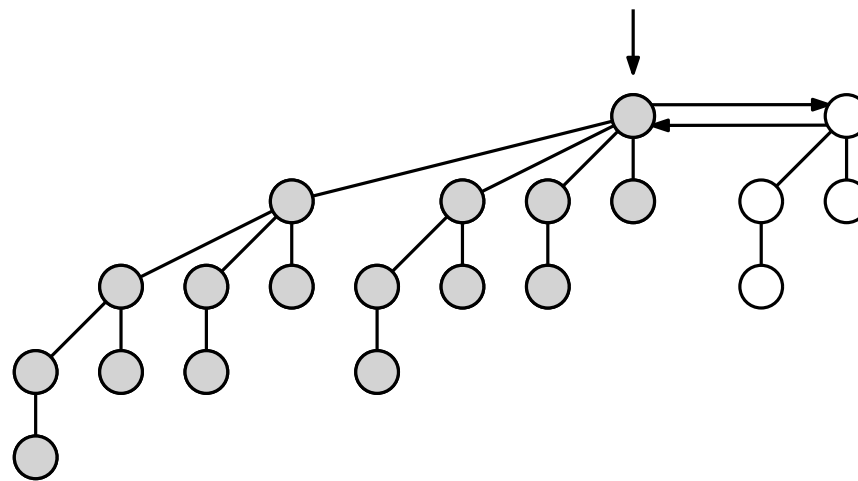
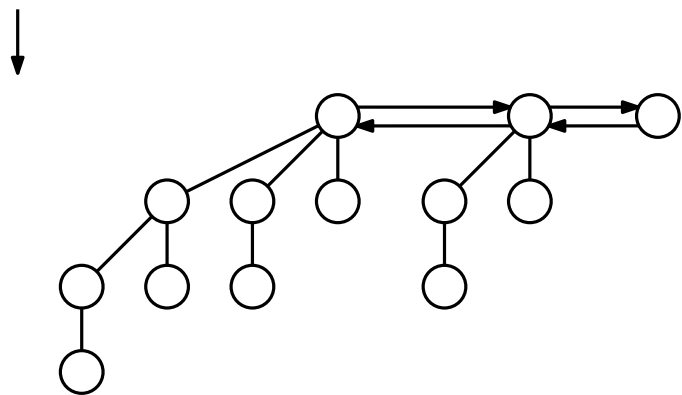
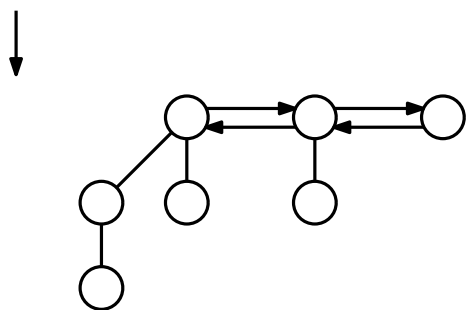
UNION(Q_1, Q_2)



UNION(Q_1, Q_2)

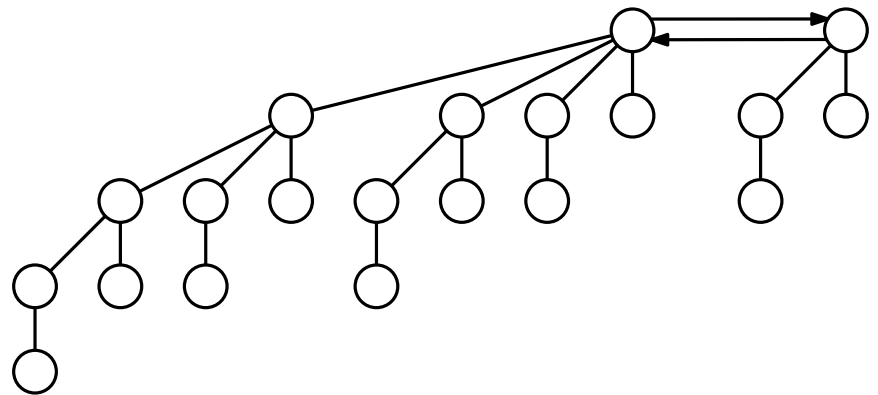
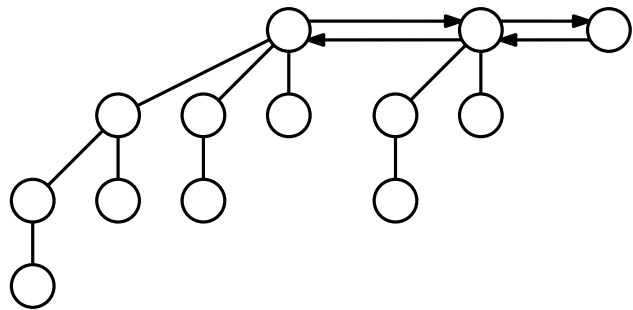
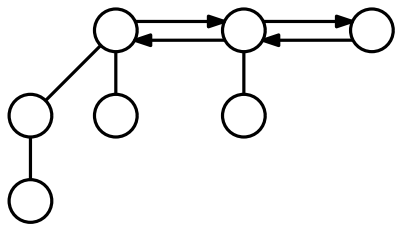


UNION(Q_1, Q_2)



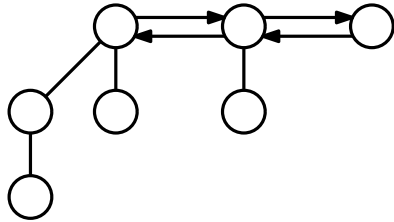
Runtime: $O(\max k) = O(\log n)$

UNION(Q_1, Q_2)



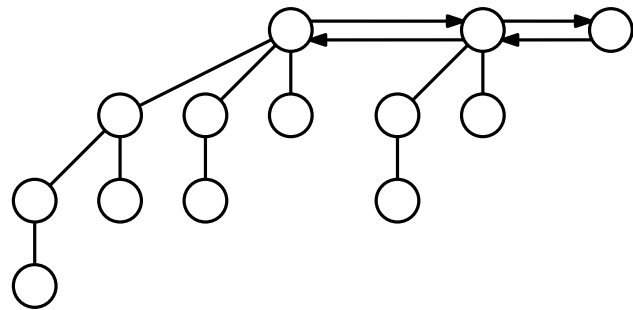
UNION(Q_1, Q_2)

$\langle 0111 \rangle$

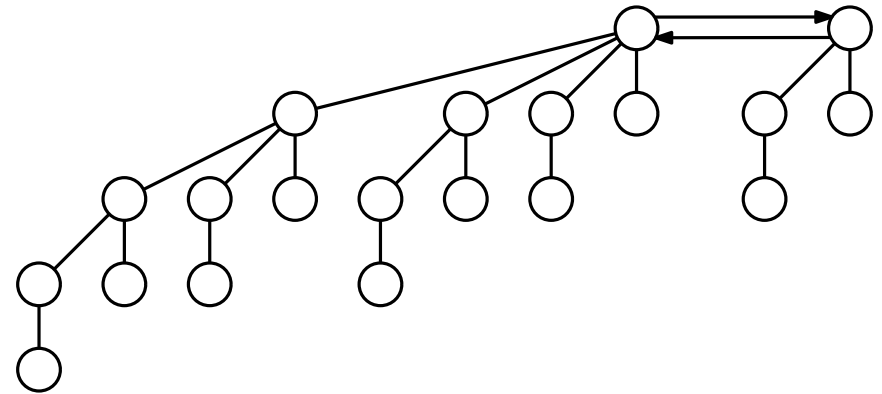


+

=



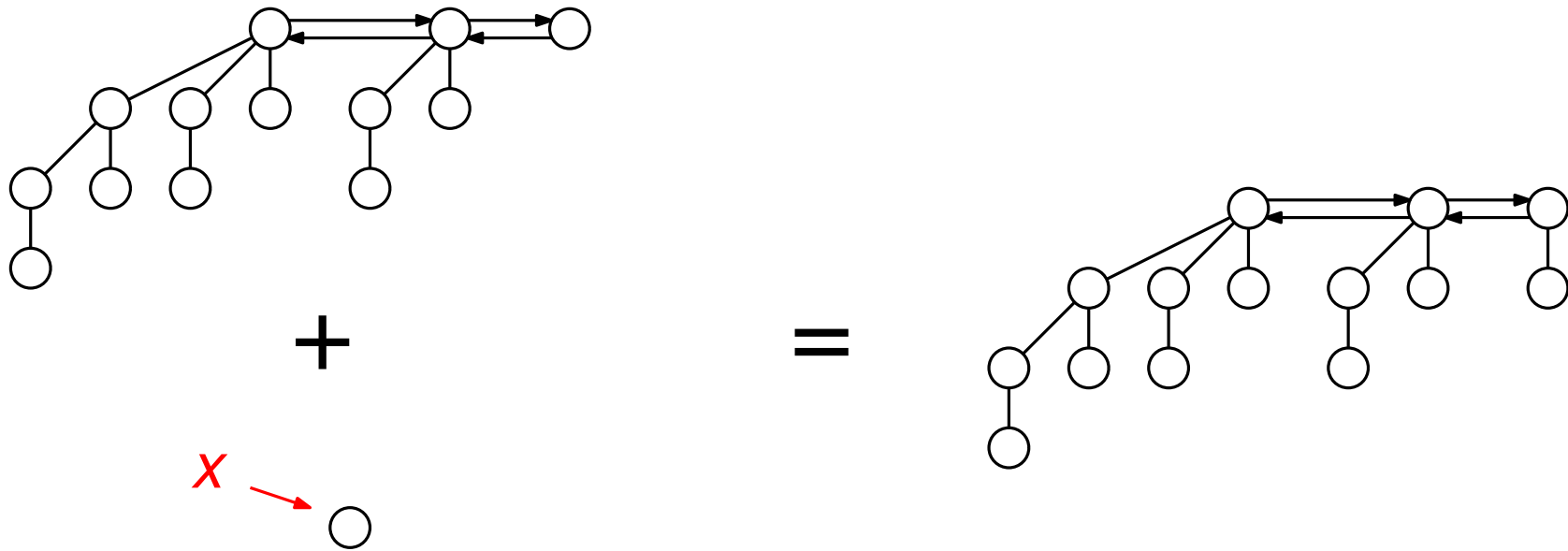
$\langle 1101 \rangle$



$\langle 10100 \rangle$

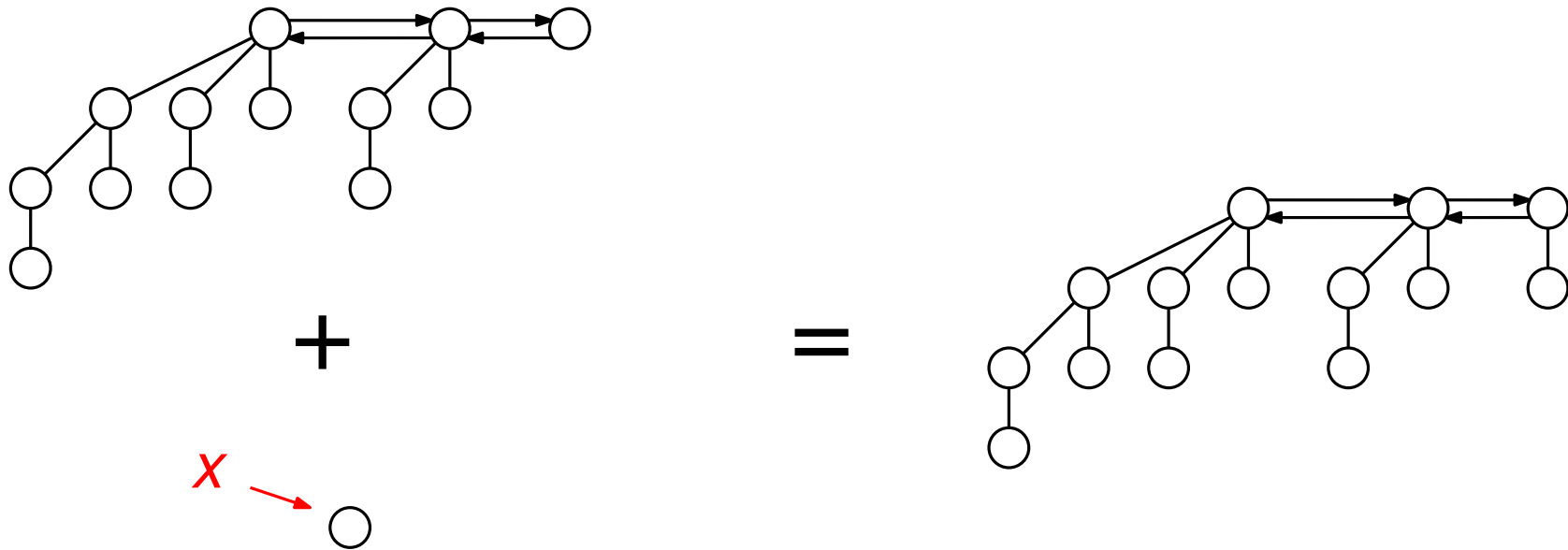
INSERT(Q, x)

Reduces to Union



INSERT(Q, x)

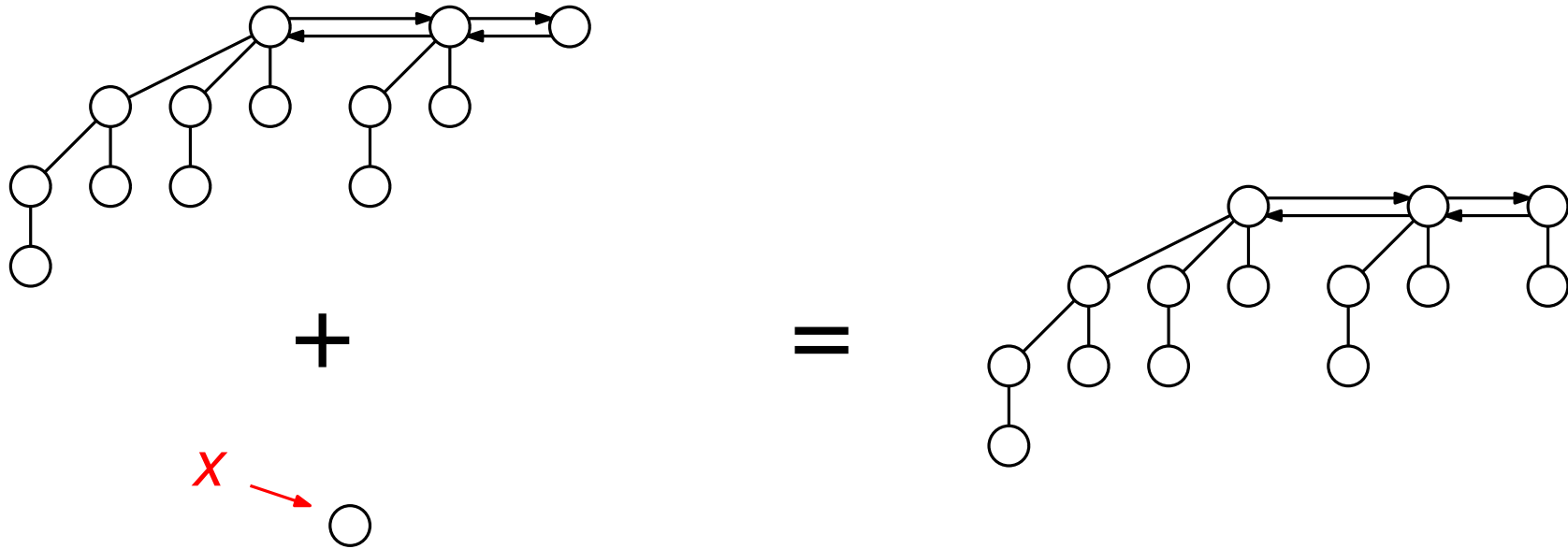
Reduces to Union



Runtime: $O(\log n)$ worst-case

INSERT(Q, x)

Reduces to Union

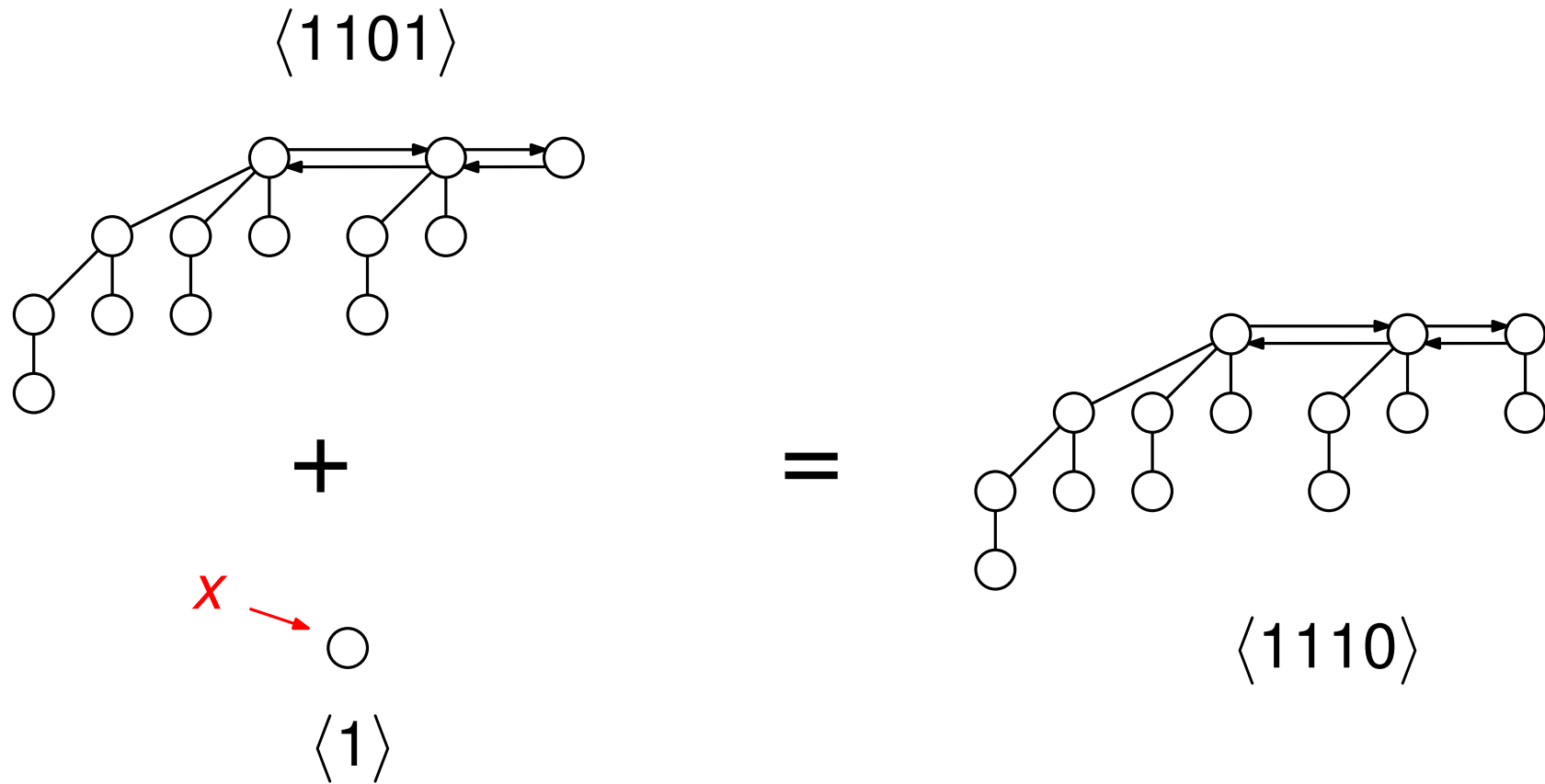


Runtime: $O(\log n)$ worst-case

Tighter analysis?

INSERT(Q, x)

Equivalent to incrementing counter



Runtime: $O(\log n)$ worst-case

Tighter analysis?

$O(1)$ amortized

EXTRACT-MIN(Q)

```
function EXTRACT-MIN( $Q$ )  
   $x$  = MINIMUM( $Q$ )  
   $Q'$  = MAKE()  
   $Q'.head$  =  $x.leftchild$   
  LINKEDLIST-EXTRACT( $x$ )  
  for each child  $y$  of  $x$  do  
     $y.parent$  = NIL  
   $Q$  = UNION( $Q$ ,  $Q'$ )  
  return  $x$ 
```

EXTRACT-MIN(Q)

function EXTRACT-MIN(Q)

$x = \text{MINIMUM}(Q)$

$Q' = \text{MAKE}()$

$Q'.head = x.leftchild$

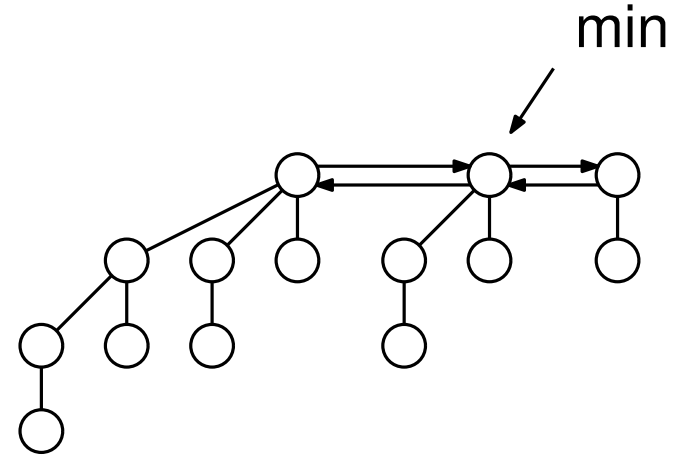
LINKEDLIST-EXTRACT(x)

for each child y of x **do**

$y.parent = \text{NIL}$

$Q = \text{UNION}(Q, Q')$

return x



EXTRACT-MIN(Q)

function EXTRACT-MIN(Q)

$x = \text{MINIMUM}(Q)$

$Q' = \text{MAKE}()$

$Q'.head = x.leftchild$

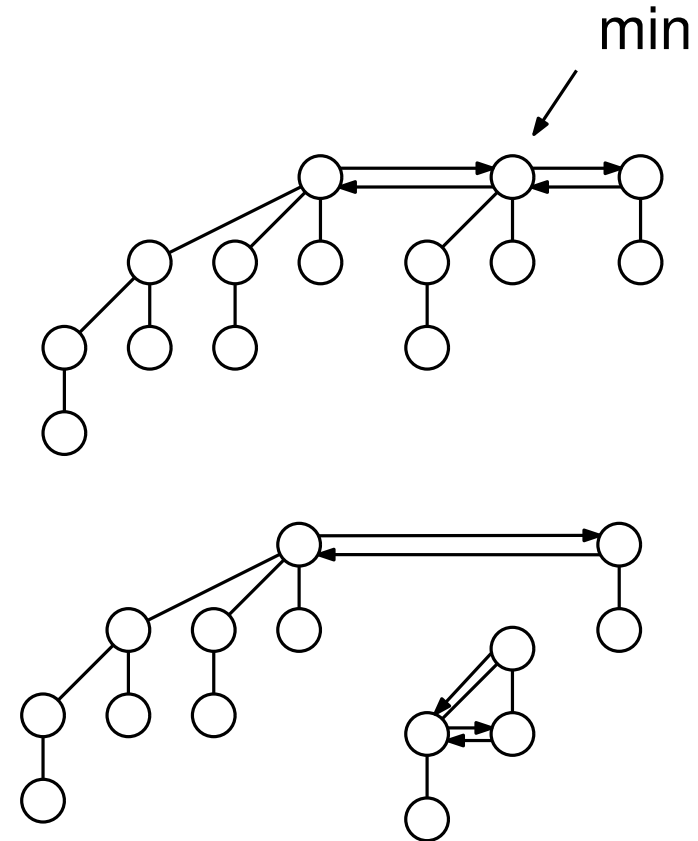
LINKEDLIST-EXTRACT(x)

for each child y of x **do**

$y.parent = \text{NIL}$

$Q = \text{UNION}(Q, Q')$

return x



EXTRACT-MIN(Q)

function EXTRACT-MIN(Q)

$x = \text{MINIMUM}(Q)$

$Q' = \text{MAKE}()$

$Q'.head = x.leftchild$

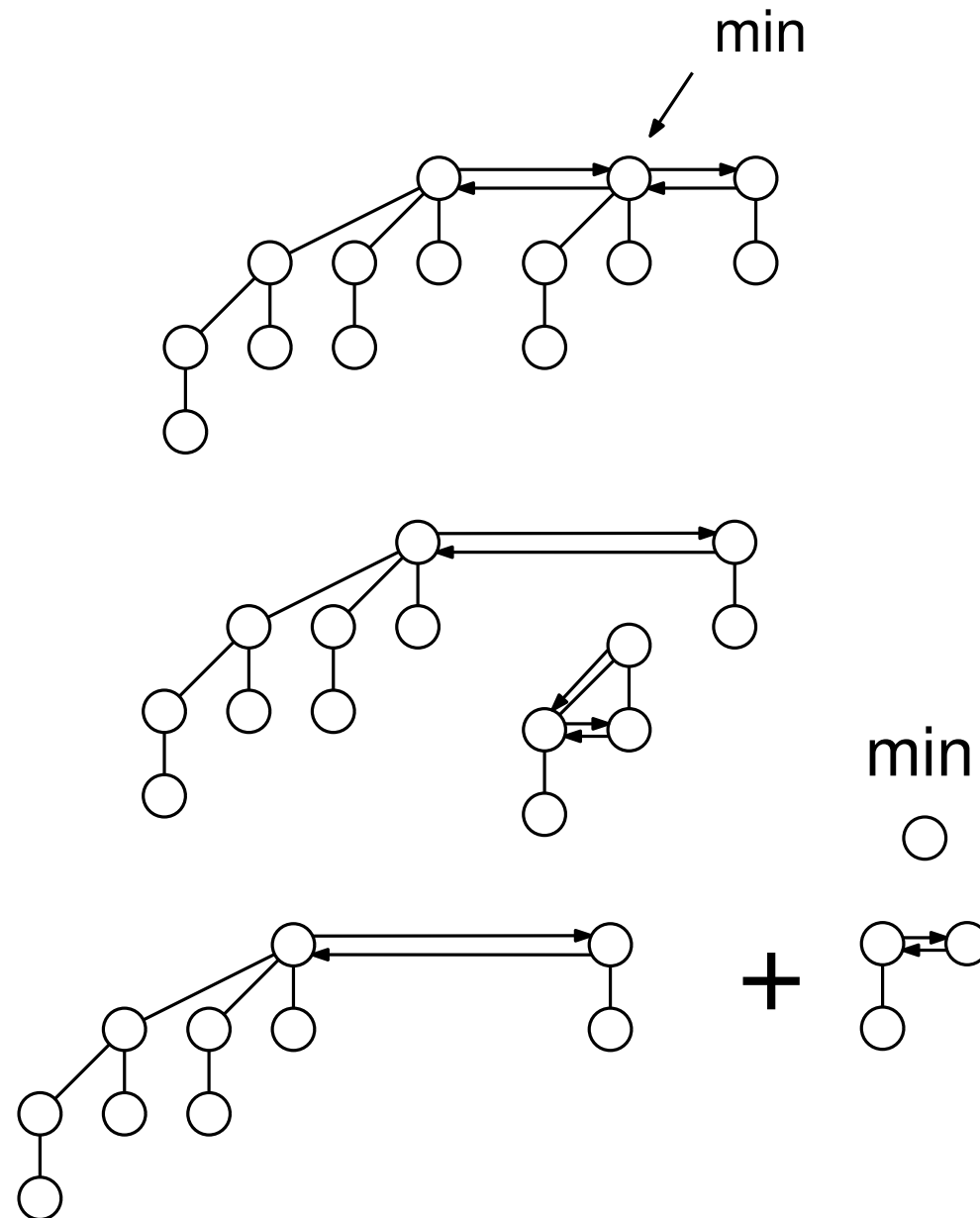
LINKEDLIST-EXTRACT(x)

for each child y of x **do**

$y.parent = \text{NIL}$

$Q = \text{UNION}(Q, Q')$

return x



EXTRACT-MIN(Q)

function EXTRACT-MIN(Q)

$x = \text{MINIMUM}(Q)$

$Q' = \text{MAKE}()$

$Q'.\text{head} = x.\text{leftchild}$

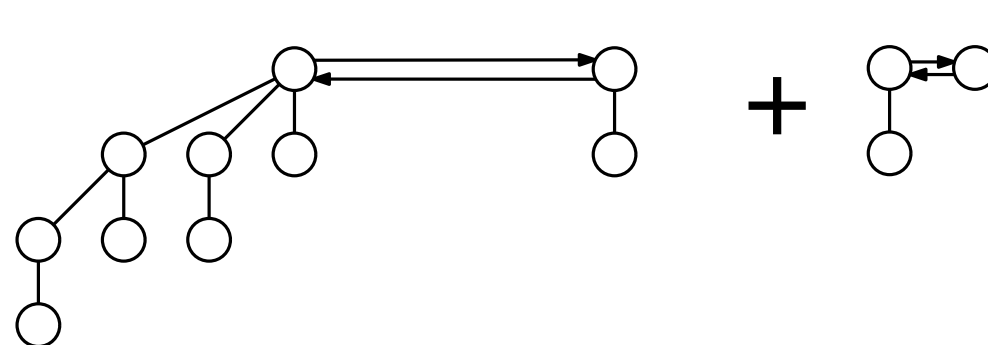
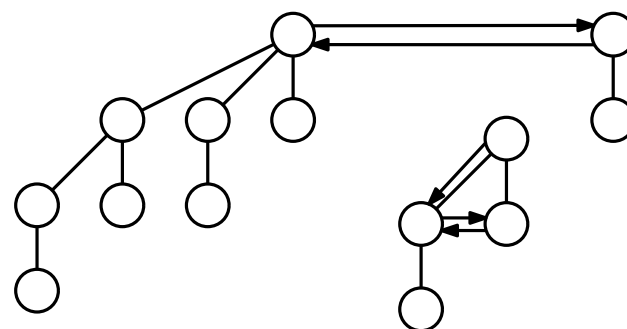
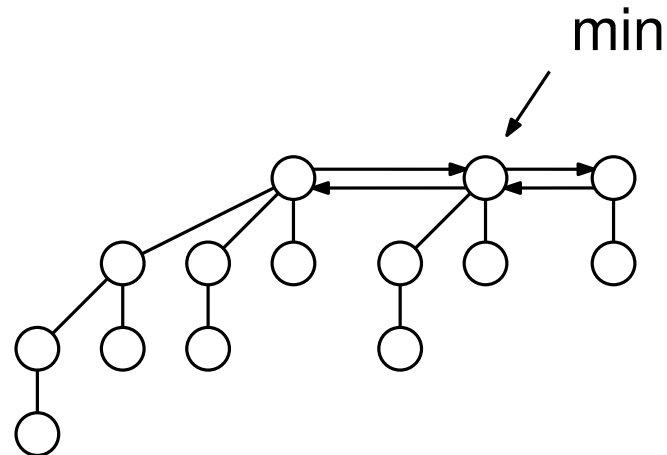
LINKEDLIST-EXTRACT(x)

for each child y of x **do**

$y.\text{parent} = \text{NIL}$

$Q = \text{UNION}(Q, Q')$

return x



Analysis: $O(\log n)$

DELETE(x)

function DELETE(x)

 DECREASE-KEY($Q, x, -\infty$)

 EXTRACT-MIN(Q)

Analysis: $O(\log n)$

Binomial Heap Summary

	Binary	Binomial
■ MAKE()	$O(1)$	$O(1)$
■ INSERT(Q, x)	$O(\log n)$	$O(1)^*$
■ MINIMUM(Q)	$O(1)$	$O(1)$
■ EXTRACT-MIN(Q)	$O(\log n)$	$O(\log n)$
■ DECREASE-KEY(Q, x, k)	$O(\log n)$	$O(\log n)$
■ DELETE(Q, x)	$O(\log n)$	$O(\log n)$
■ UNION(Q_1, Q_2)	$O(n)$	$O(\log n)$

* Amortized cost

Binomial Heap Summary

	Binary	Binomial	Lazy Binomial
■ MAKE()	$O(1)$	$O(1)$	$O(1)$
■ INSERT(Q, x)	$O(\log n)$	$O(1)^*$	$O(1)$
■ MINIMUM(Q)	$O(1)$	$O(1)$	$O(1)$
■ EXTRACT-MIN(Q)	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
■ DECREASE-KEY(Q, x, k)	$O(\log n)$	$O(\log n)$	$O(\log n)$
■ DELETE(Q, x)	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
■ UNION(Q_1, Q_2)	$O(n)$	$O(\log n)$	$O(1)$

* Amortized cost

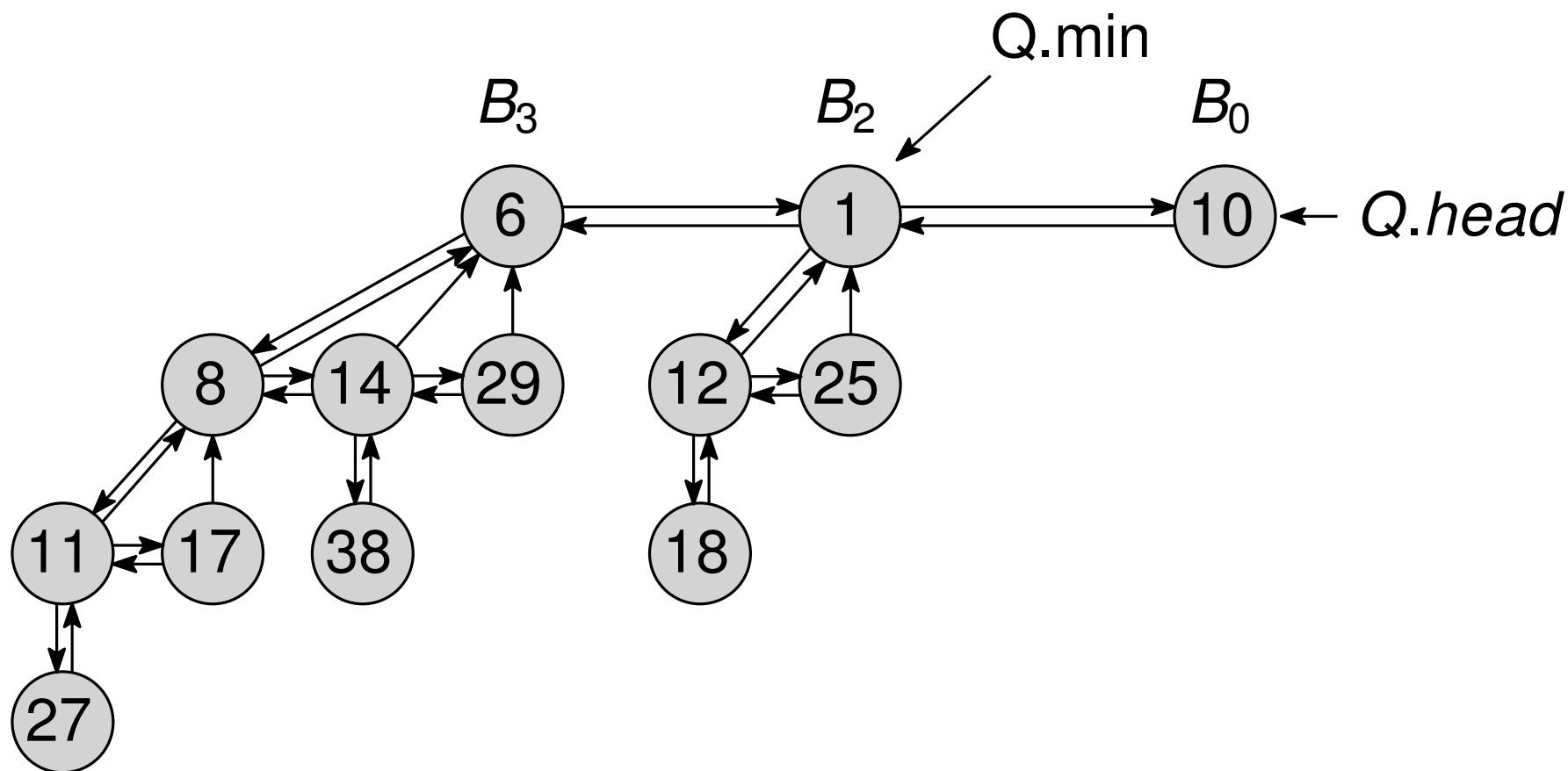
Lazy Binomial Heaps (Homework)

- Don't consolidate trees during UNION
- Consolidate during EXTRACT-MIN(Q)

Binomial Heaps

Collection of heap-ordered binomial trees:

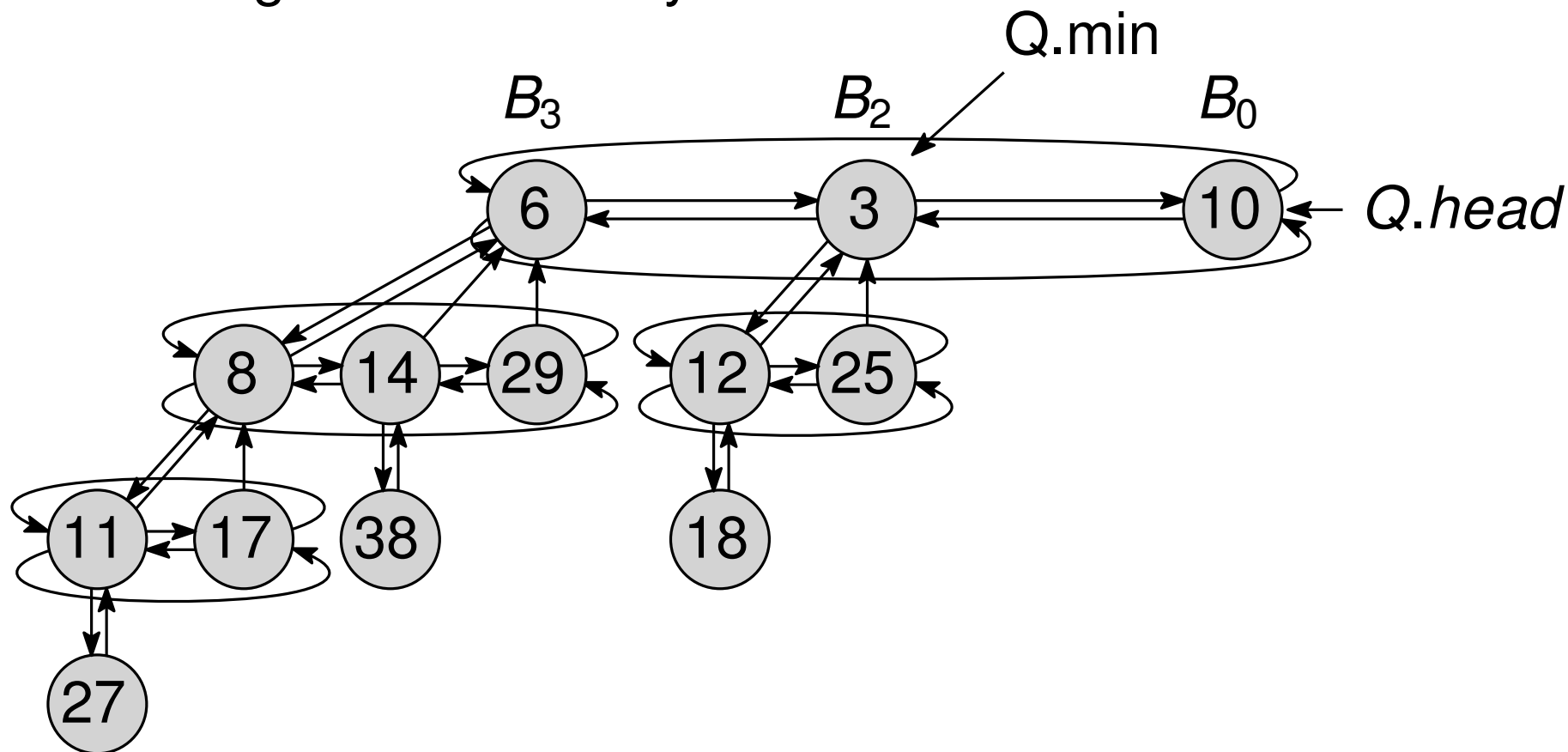
- Each tree is heap-ordered
- At most **one** tree B_k , for $k = 0, 1, 2, \dots, \lfloor \log n \rfloor$



Lazy Binomial Heaps

Collection of heap-ordered binomial trees:

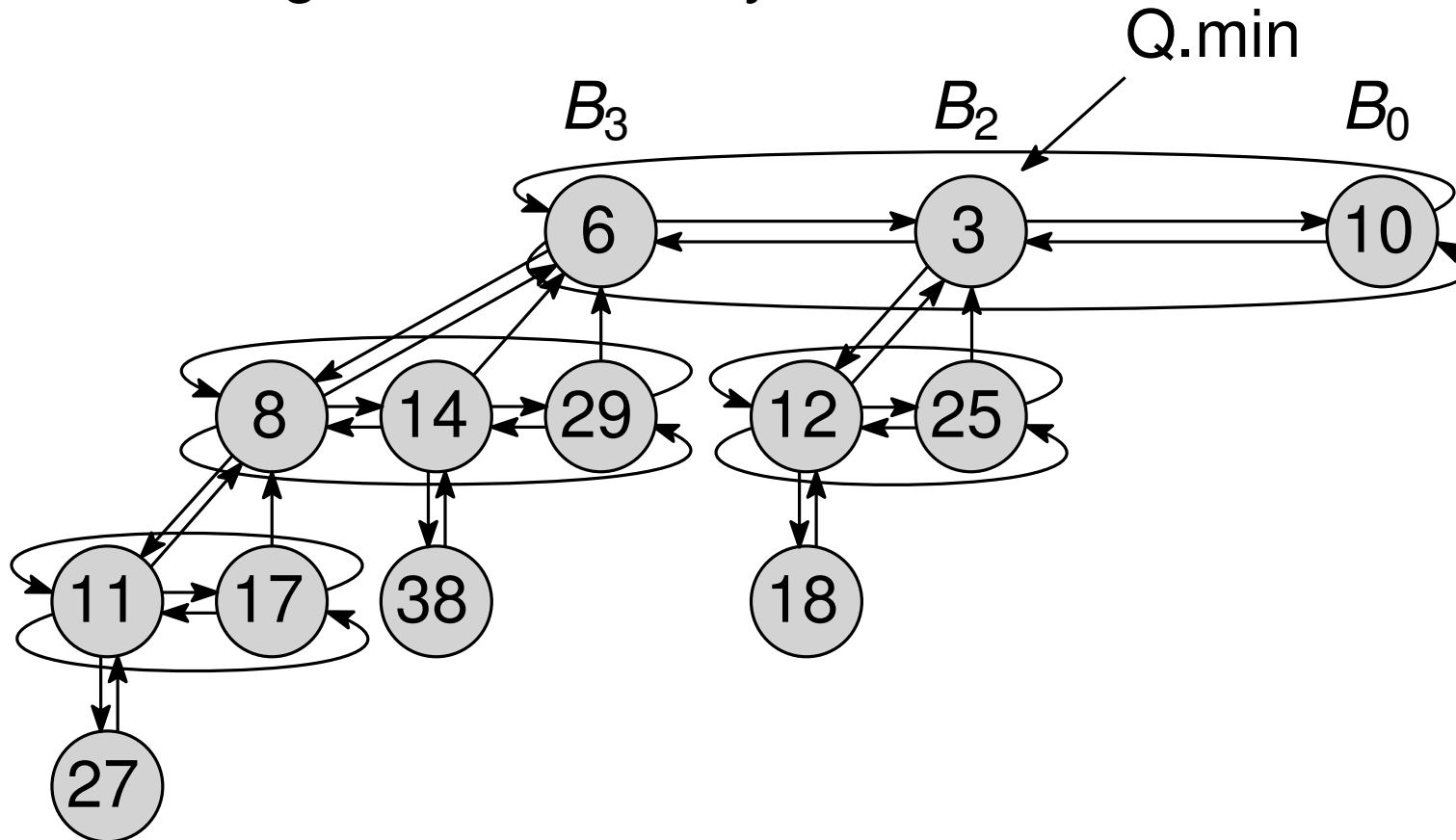
- Each tree is heap-ordered
- *Arbitrary* number of trees in the root list
- Sibling lists are doubly-linked *circular* lists



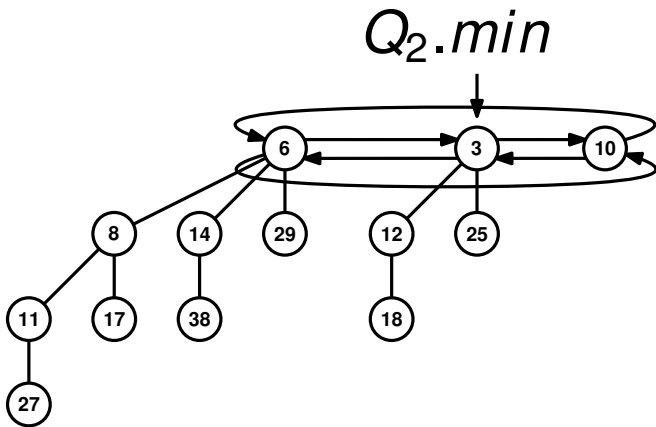
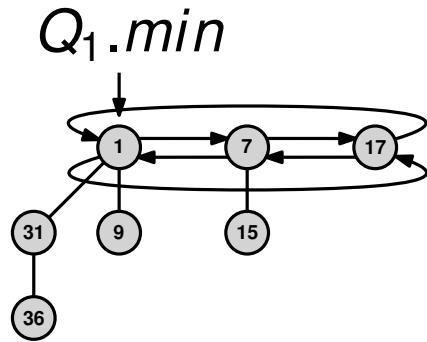
Lazy Binomial Heaps

Collection of heap-ordered binomial trees:

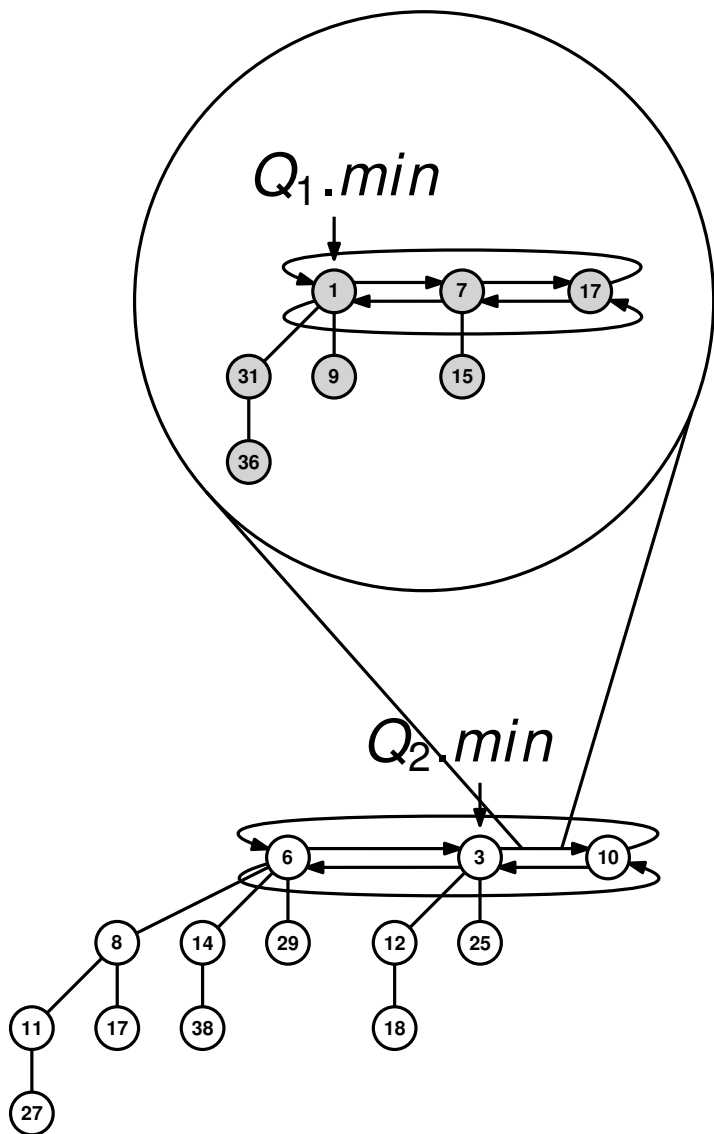
- Each tree is heap-ordered
- *Arbitrary* number of trees in the root list
- Sibling lists are doubly-linked *circular* lists



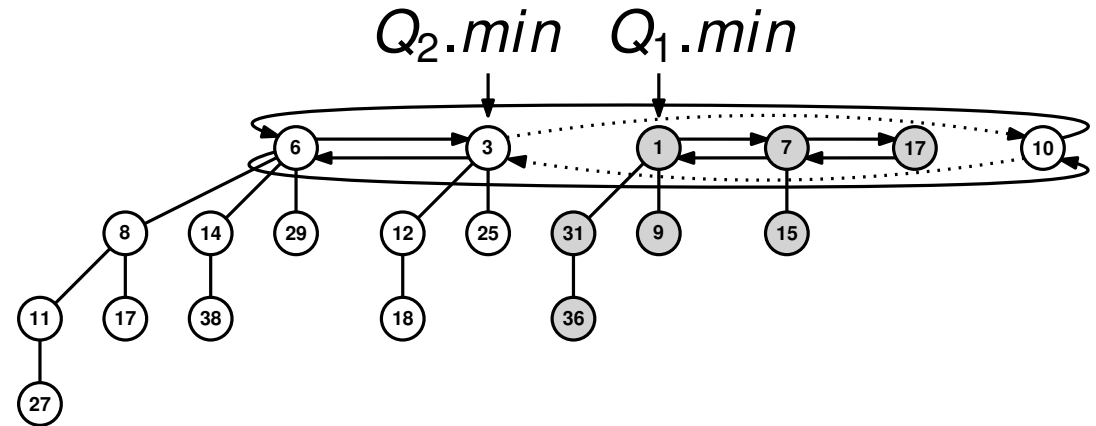
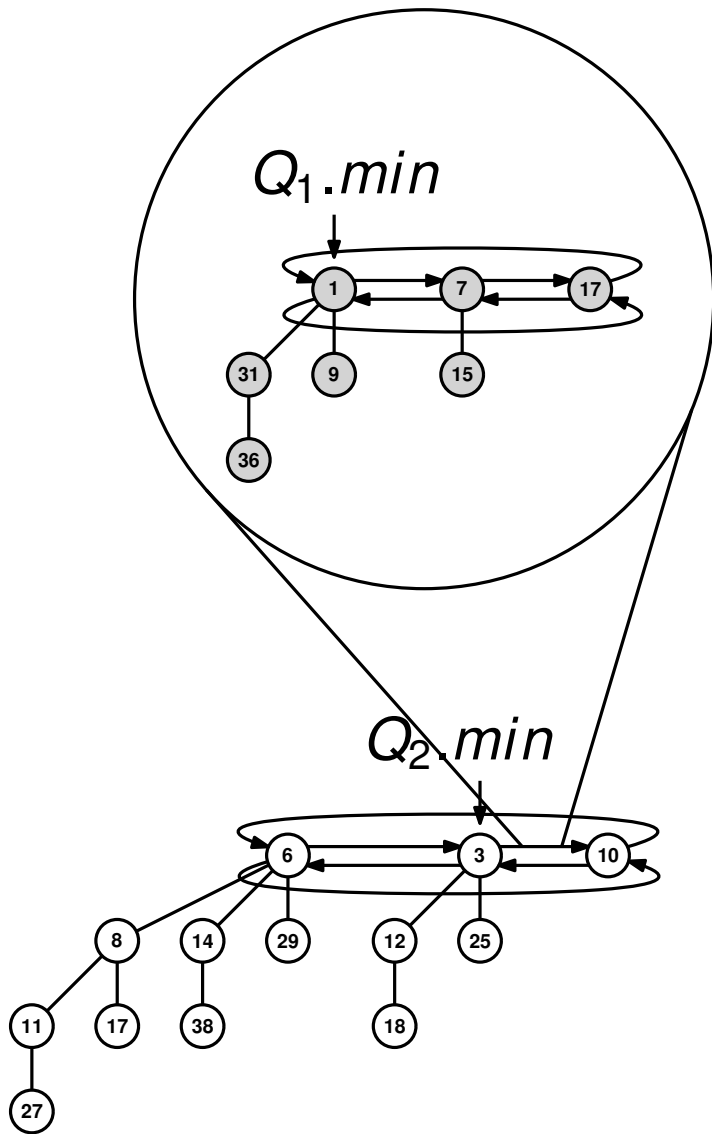
Lazy UNION(Q_1, Q_2)



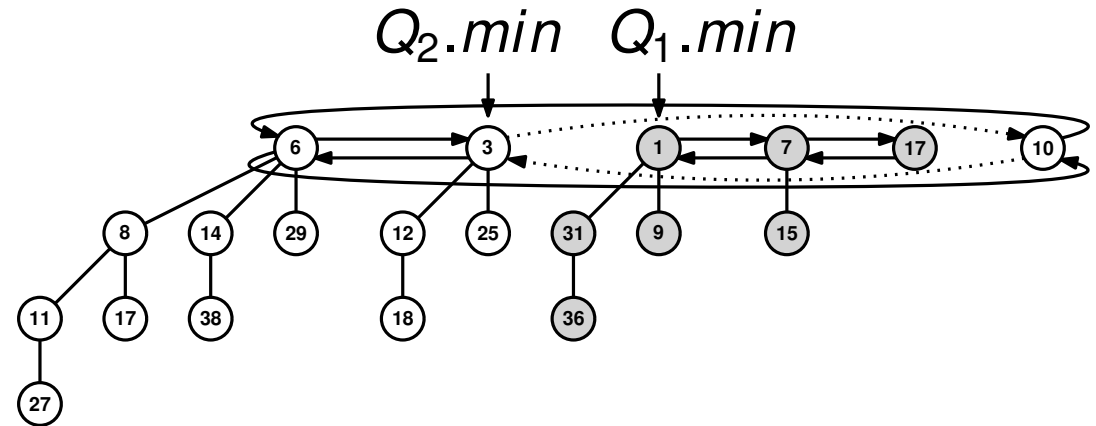
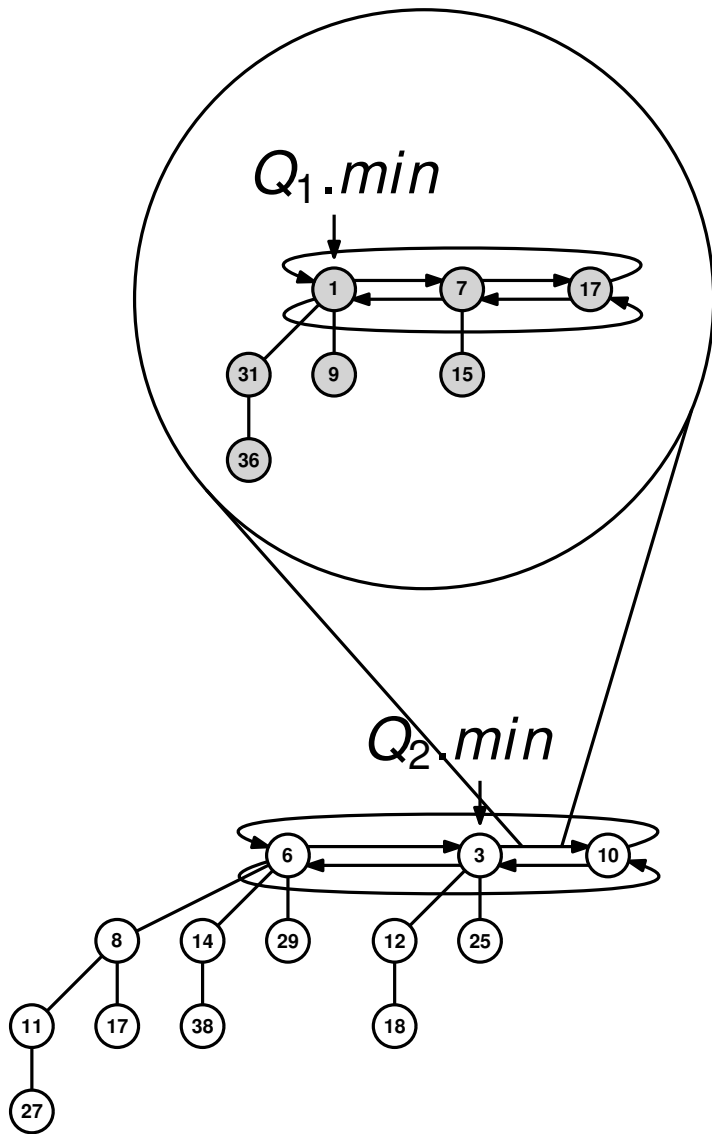
Lazy UNION(Q_1, Q_2)



Lazy UNION(Q_1, Q_2)



Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

$L_1 \leftarrow Q_1.min.left$

$R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

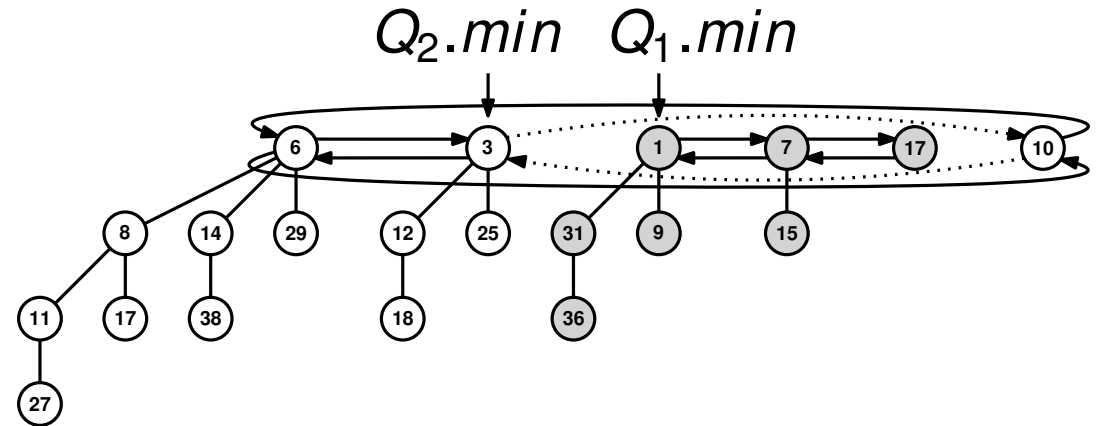
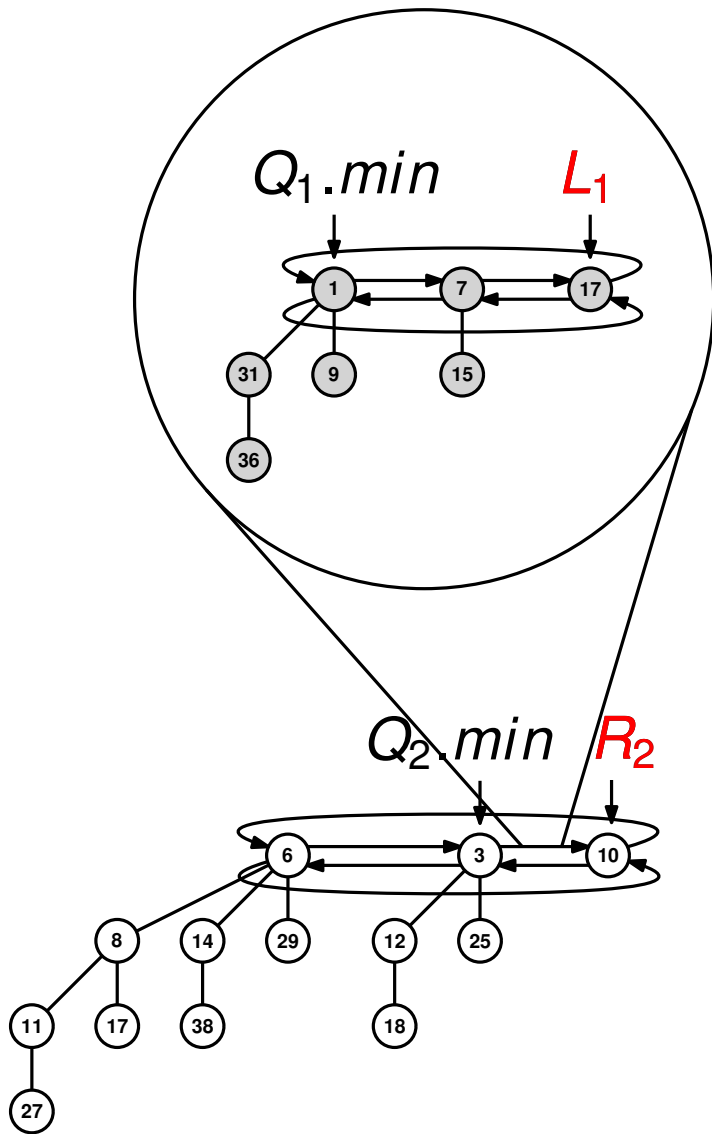
$Q_1.min.left \leftarrow Q_2.min$

if $Q_1.min.key < Q_2.min.key$ **then**

$Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

→ $L_1 \leftarrow Q_1.min.left$

→ $R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

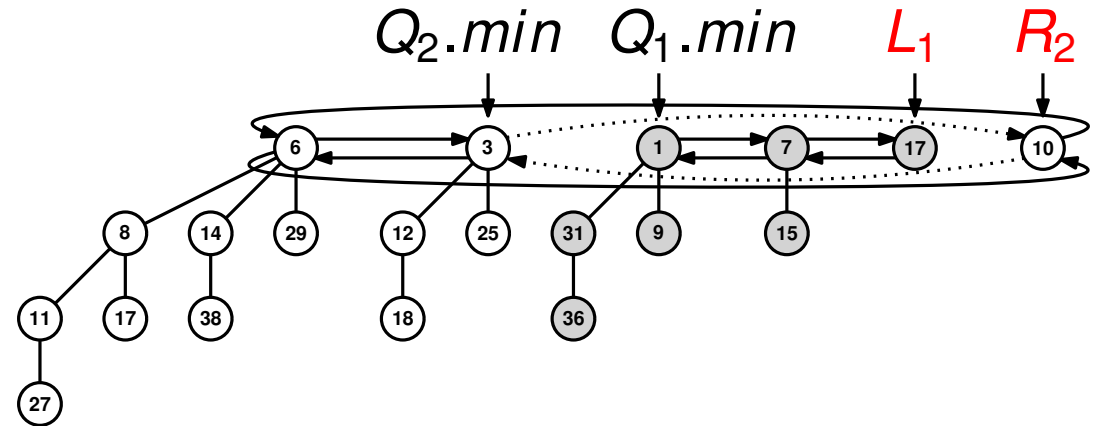
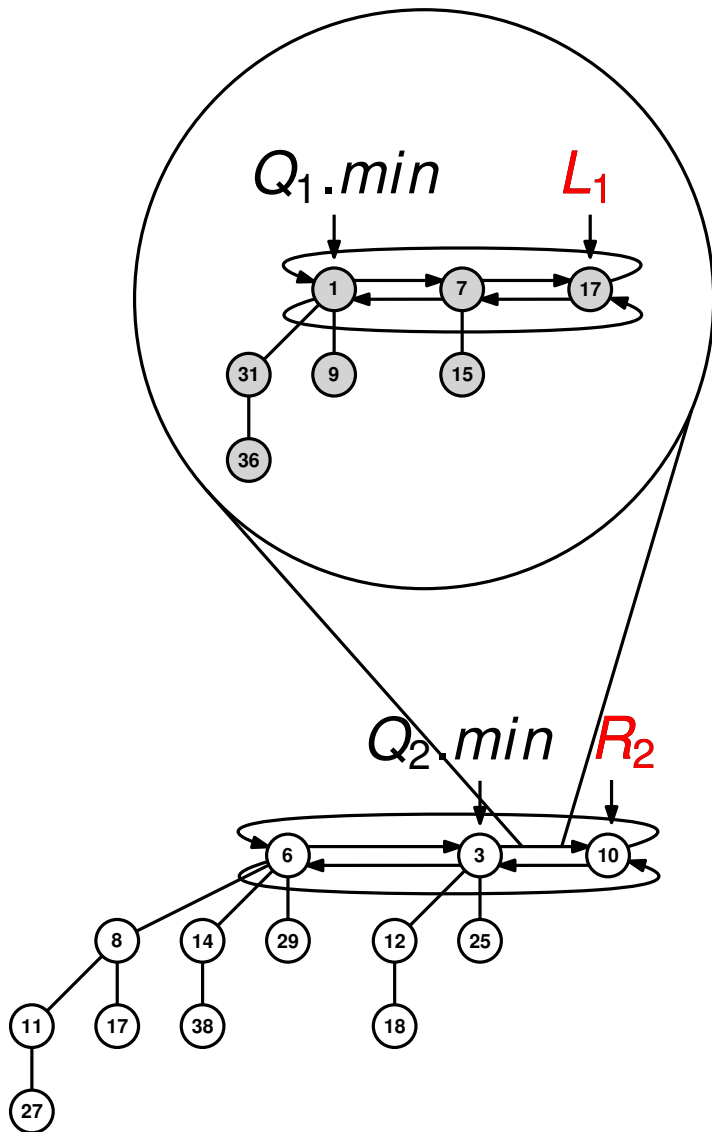
$Q_1.min.left \leftarrow Q_2.min$

if $Q_1.min.key < Q_2.min.key$ **then**

$Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

→ $L_1 \leftarrow Q_1.min.left$

→ $R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

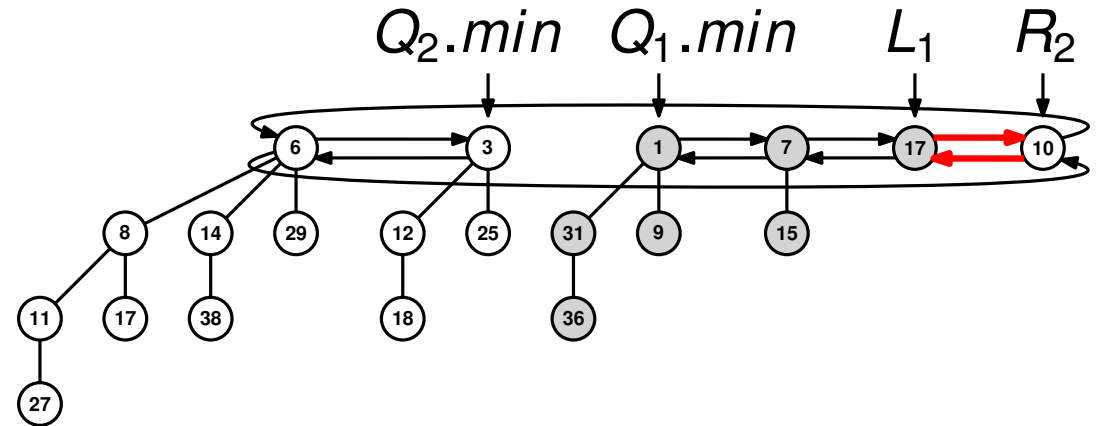
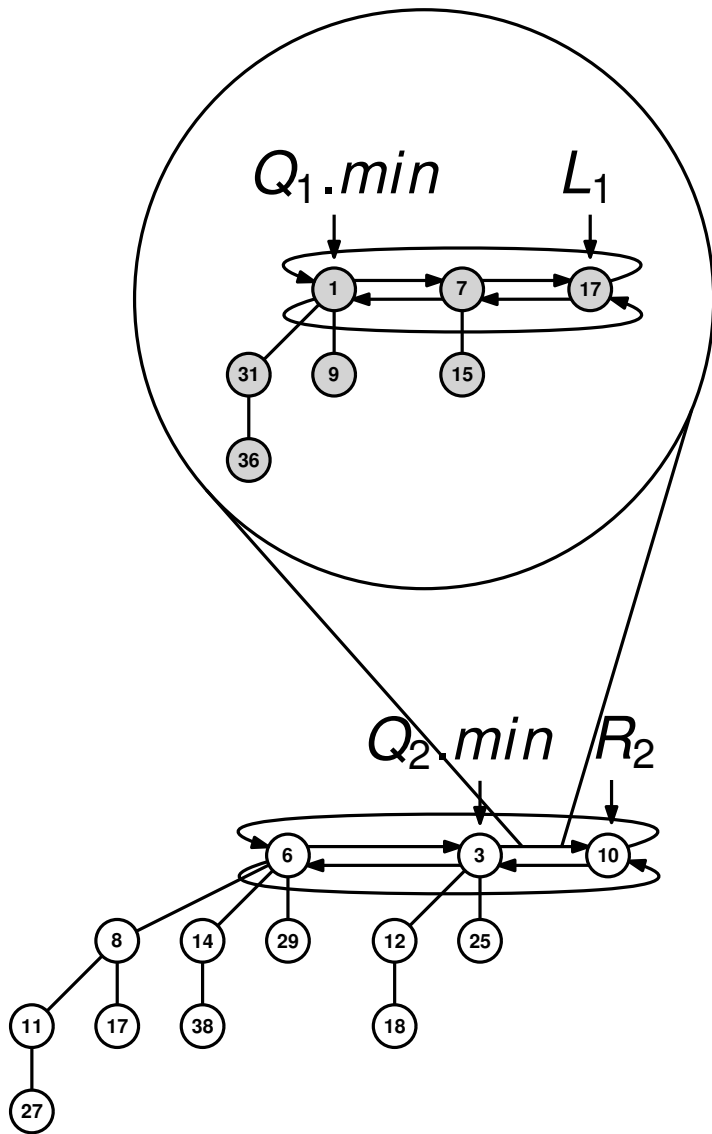
$Q_1.min.left \leftarrow Q_2.min$

if $Q_1.min.key < Q_2.min.key$ **then**

$Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

$L_1 \leftarrow Q_1.min.left$

$R_2 \leftarrow Q_2.min.right$

$\rightarrow L_1.right \leftarrow R_2$

$\rightarrow R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

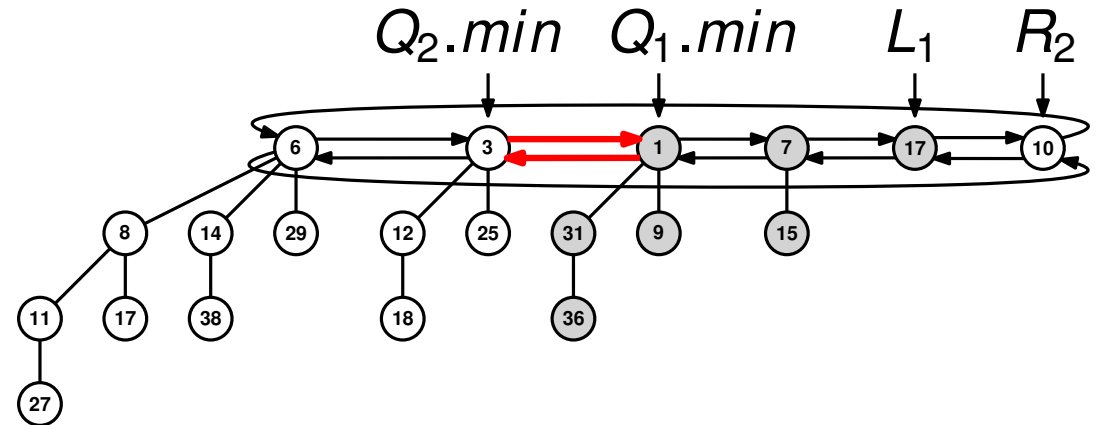
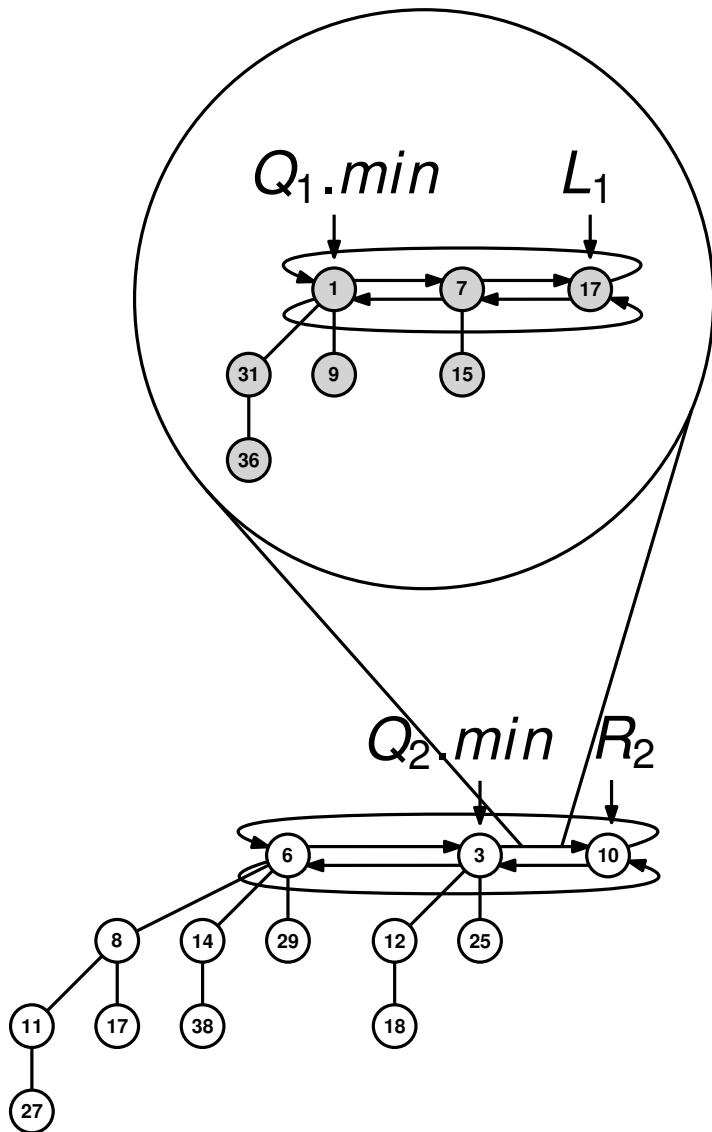
$Q_1.min.left \leftarrow Q_2.min$

if $Q_1.min.key < Q_2.min.key$ **then**

$Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

$L_1 \leftarrow Q_1.min.left$

$R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$\rightarrow Q_2.min.right \leftarrow Q_1.min$

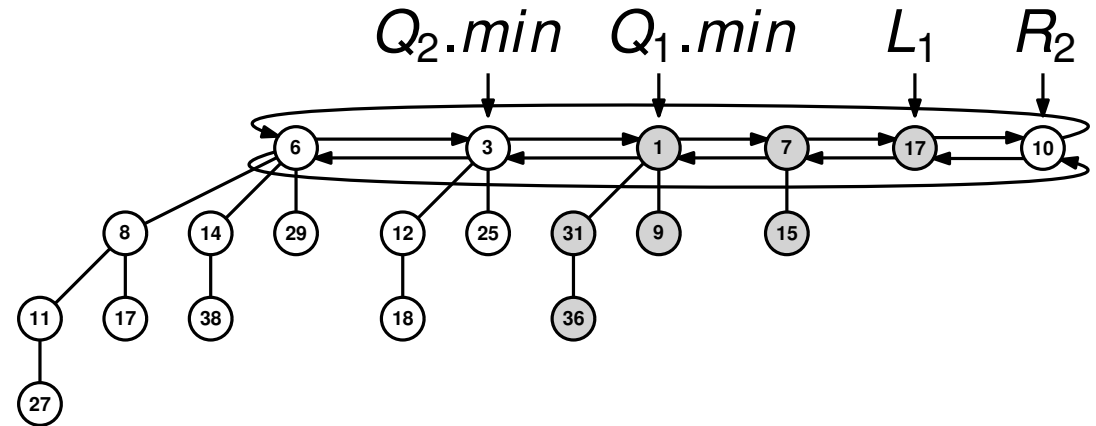
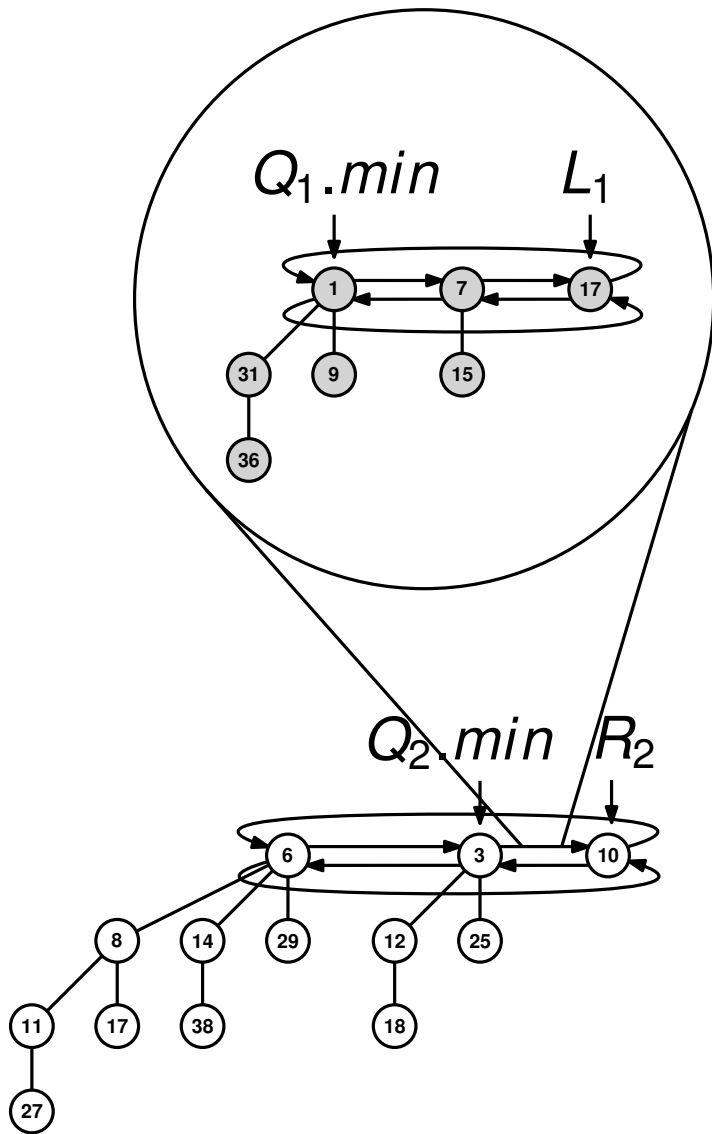
$\rightarrow Q_1.min.left \leftarrow Q_2.min$

if $Q_1.min.key < Q_2.min.key$ **then**

$Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

$L_1 \leftarrow Q_1.min.left$

$R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

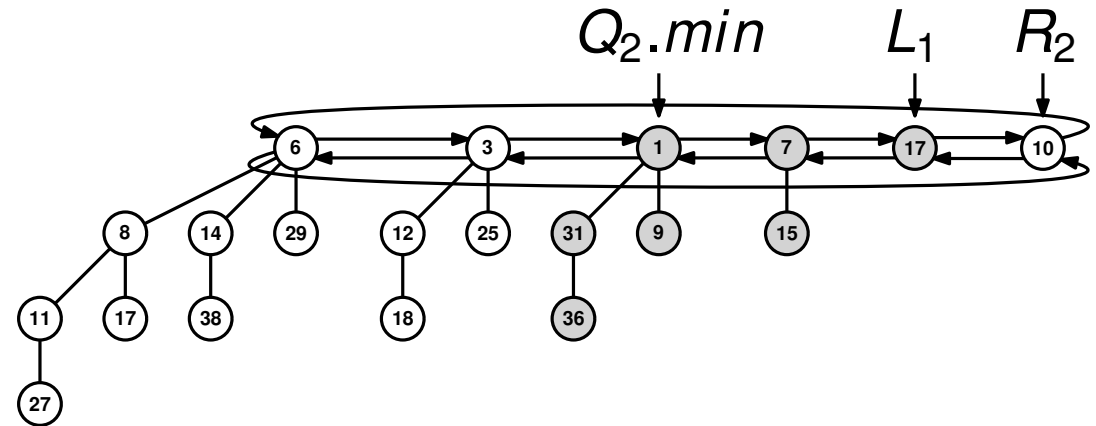
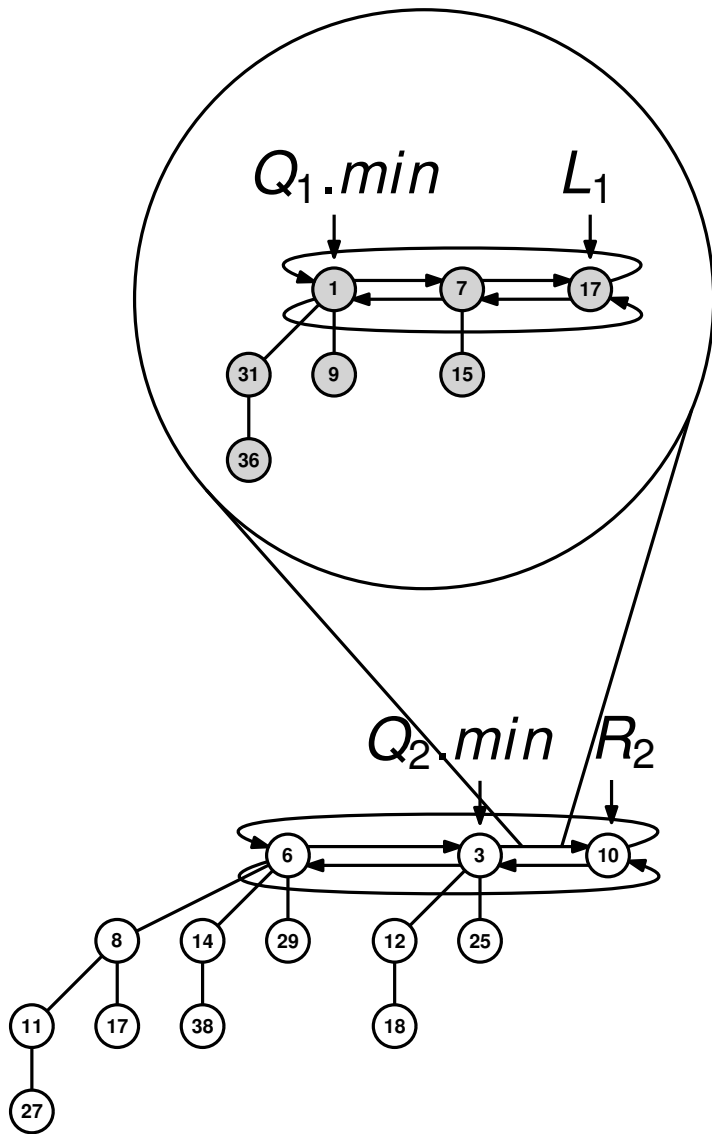
$Q_1.min.left \leftarrow Q_2.min$

→ if $Q_1.min.key < Q_2.min.key$ **then**

→ $Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



function UNION(Q_1, Q_2)

$L_1 \leftarrow Q_1.min.left$

$R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

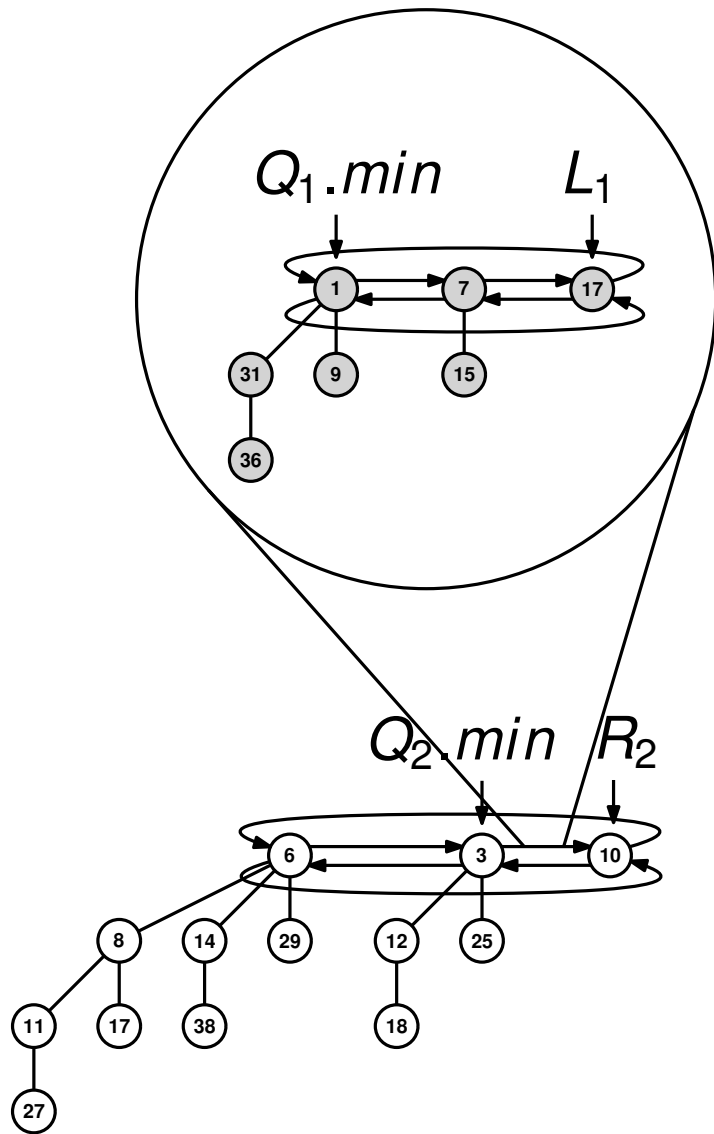
$Q_1.min.left \leftarrow Q_2.min$

→ if $Q_1.min.key < Q_2.min.key$ **then**

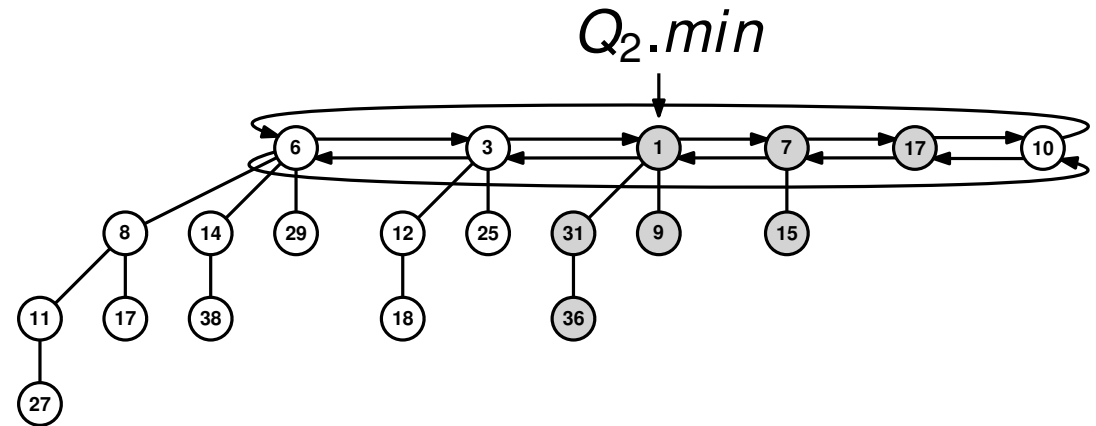
→ $Q_2.min \leftarrow Q_1.min$

return Q_2

Lazy UNION(Q_1, Q_2)



$O(1)$ time worst-case



function UNION(Q_1, Q_2)

$L_1 \leftarrow Q_1.min.left$

$R_2 \leftarrow Q_2.min.right$

$L_1.right \leftarrow R_2$

$R_2.left \leftarrow L_1$

$Q_2.min.right \leftarrow Q_1.min$

$Q_1.min.left \leftarrow Q_2.min$

if $Q_1.min.key < Q_2.min.key$ **then**

$Q_2.min \leftarrow Q_1.min$

return Q_2

Extract-Min

Extract-Min

In the homework