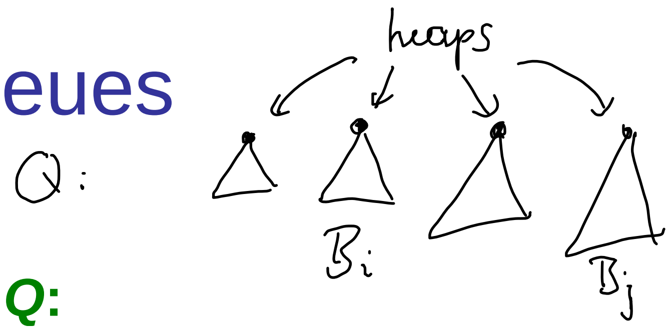
Binomial queues



$i \neq j$

Binomial queue *Q*:

Set of heap ordered binomial trees of different order to store keys.

$(n)_2 = b_k b_{k-1} \dots b_0$

$n = \sum_{i=0}^{k} b_i \cdot 2^i$

***n* keys:**

$B_i \in Q \iff i\text{-th bit in } (n)_2 = 1$

Do we have enough space to store n keys?
Yes: Each B_i has 2^i many nodes.

9 keys:

{2, 4, 7, 9,12, 23, 58, 65, 85}

$9 = (1001)_2$

$B_3 \quad B_0$

1 0 1 0 1 1 1 0

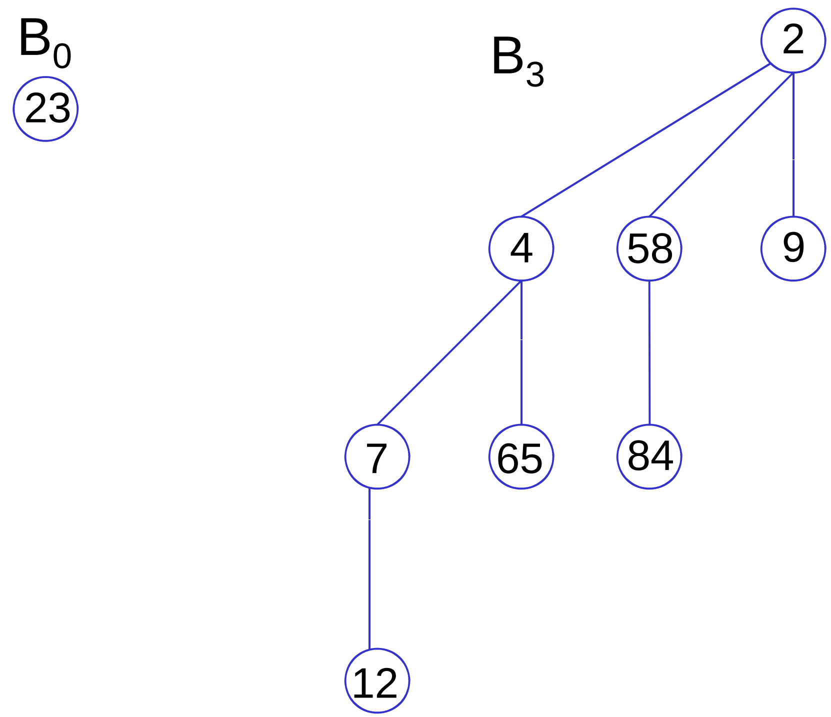
$B_7 \quad B_5 \quad B_3 B_2 B_1$

Binomial queues: 1st example

9 keys:

{2, 4, 7, 9,12, 23, 58, 65, 85}

$9 = (1001)_2$



Min can be determined in O(log *n*) time.

Binomial queues: 2nd example

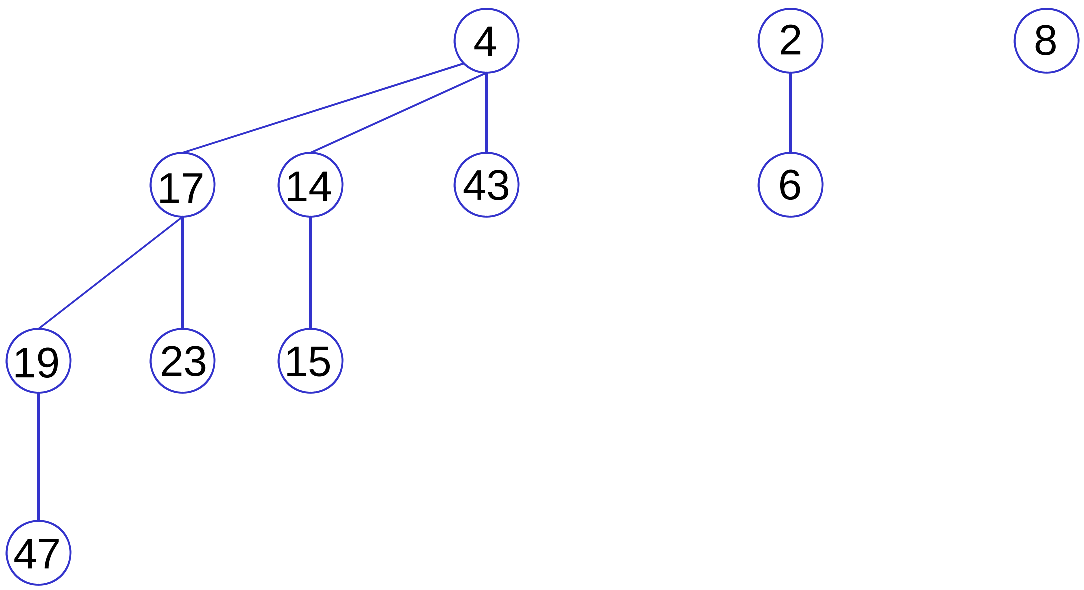
11 keys:

{2, 4, 6, 8, 14, 15, 17, 19, 23, 43, 47}

11 = $(1011)_2$ → 3 binomial trees

B_3, B_1 and B_0

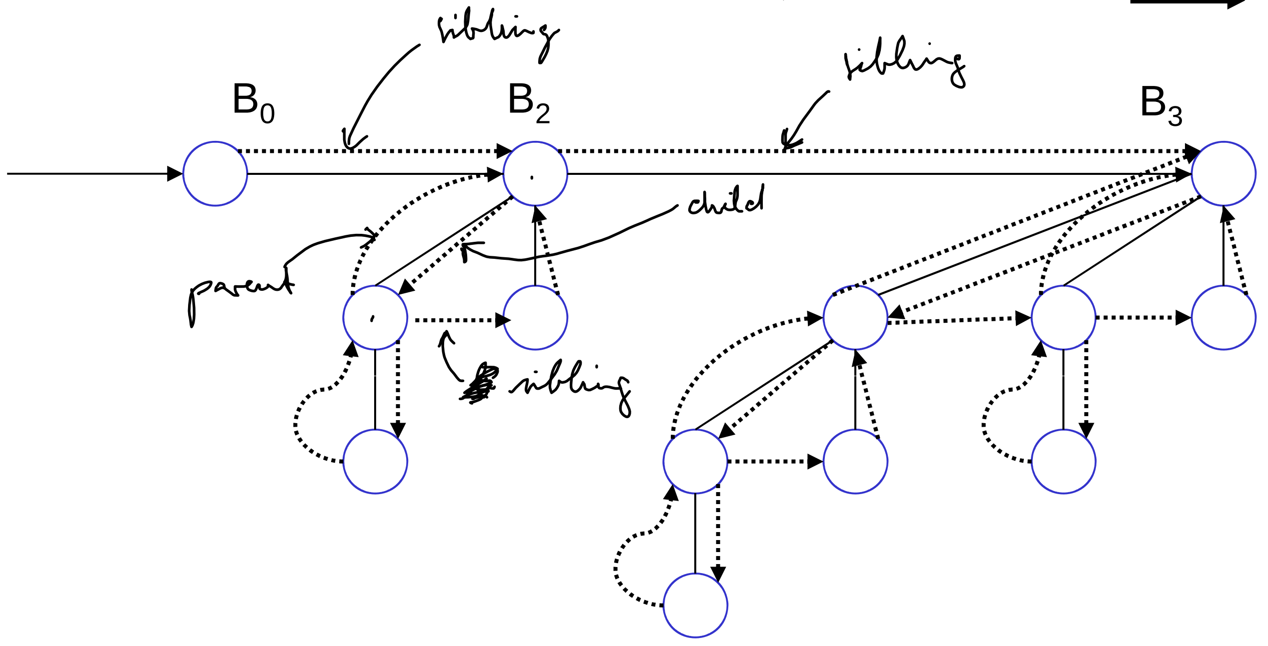
Q_{11}:



Child-sibling representation

Structure of a node:

parent	
key	degree
child	sibling

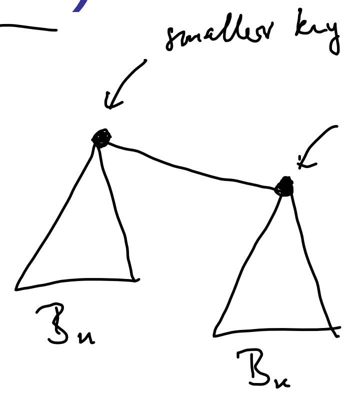


Binomial trees: operation 'meld' ('link')

link: Unite two binomial trees *B, B´* of the same order

$B_n + B_n \rightarrow B_{n+1}$

maintain heap property

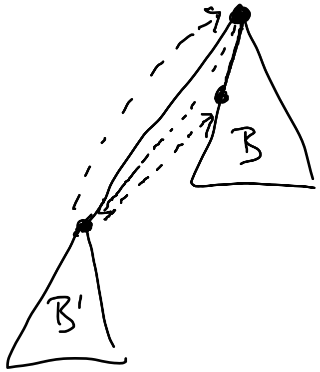


procedure Link:

B.Link(B´)

/* Make the root with the larger key a child of the root with the smaller key. */

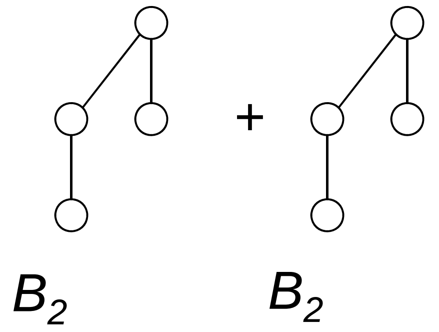
```
if B.key >  B´.key
   then B´.Link(B)
        return
/* B.key ≤ B´.key*/
B´.parent = B
B´.sibling = B.child
B.child = B´
B.degree = B.degree +1
```

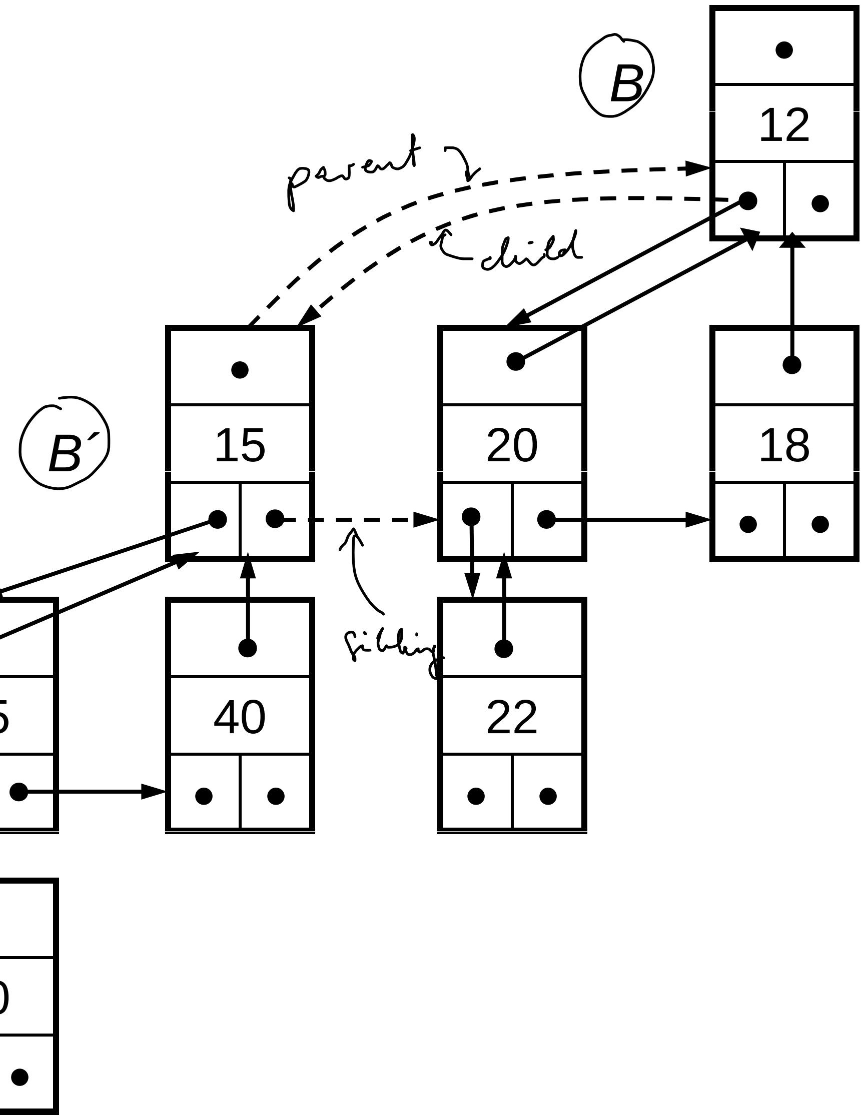


Running time O(1)

Example of the operation ‘link’

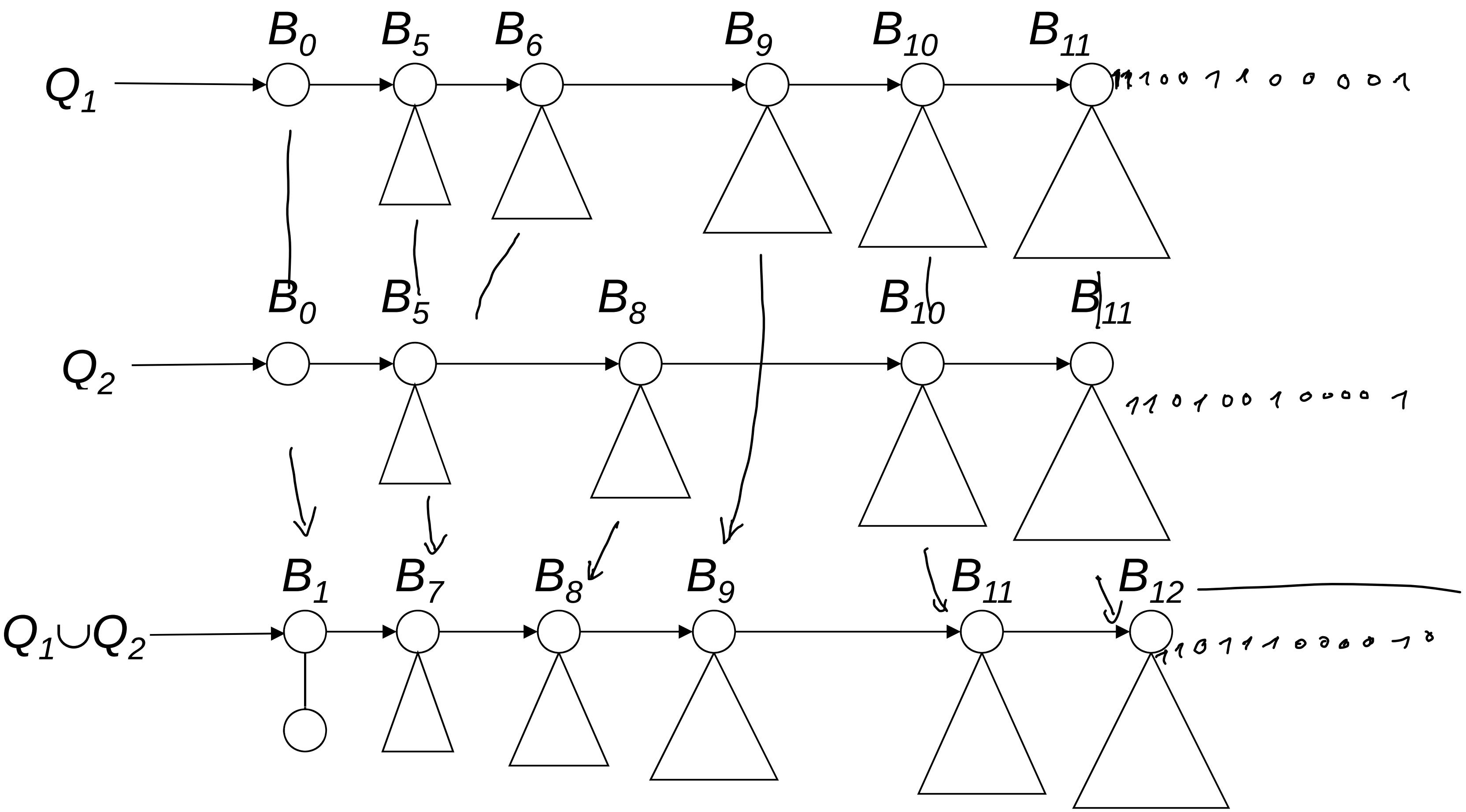
link





Binomial queues: operation 'meld'

Binary addition



If the operation yields a B_i and the initial lists both contain a B_i, then unite the initial B_i's.

Running time: O (log *n*)

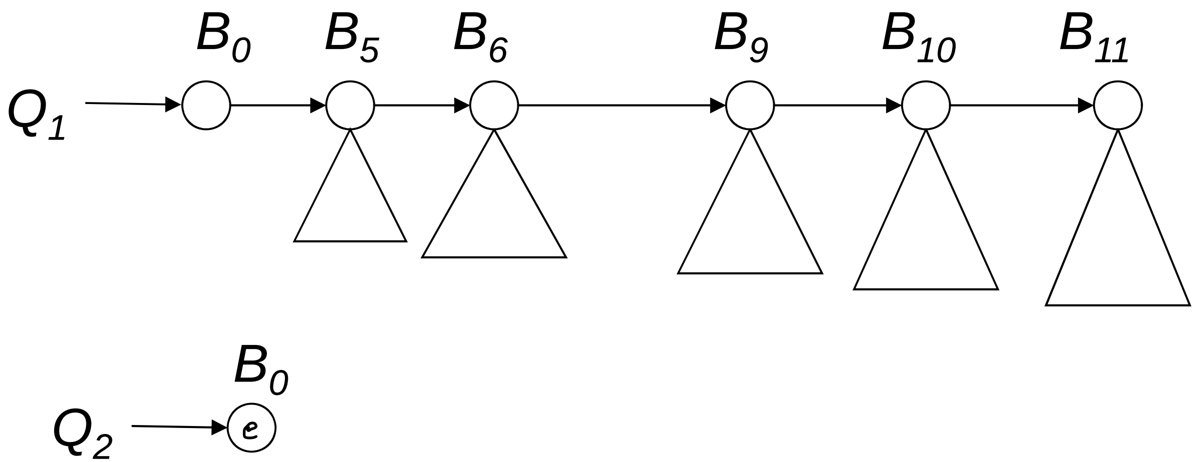
Binomial queues: operations

Q.initialize:

Q.root = null

Q.insert(e):

new B_0
B_0.*key* = e
Q.*meld*(B_0)



Running time: O(log *n*)

Binomial queues: 'deletemin'

Q.deletemin():

$O(\log n)$

1. Determine B_i whose root has the minimum key in the root list and delete B_i from Q (returns Q´)

$O(\log n)$

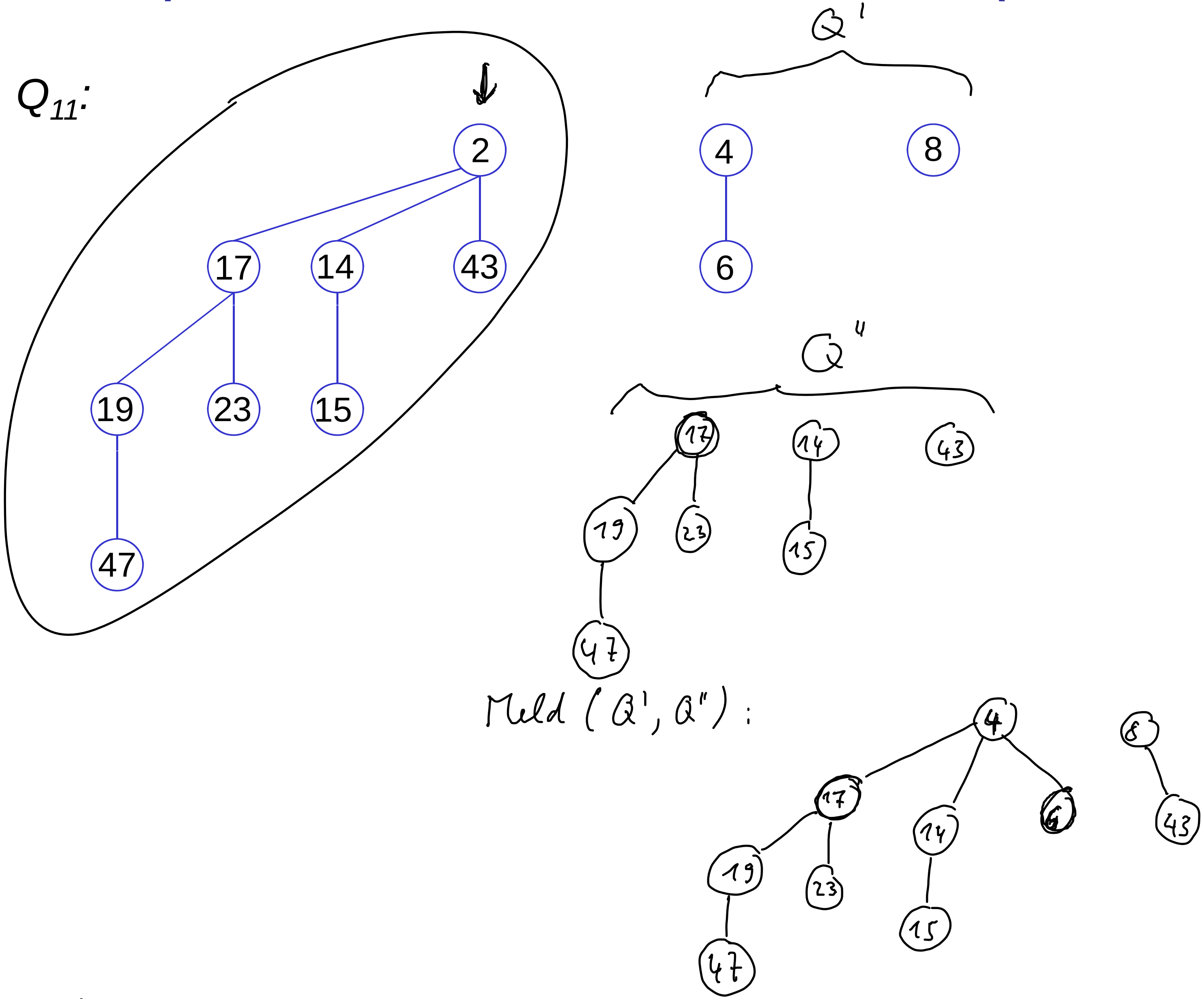
2. Insert the children of B_i in reverse order into a new queue : B_0 , B_1 , ... , B_{i-1} → Q''

$O(\log n)$

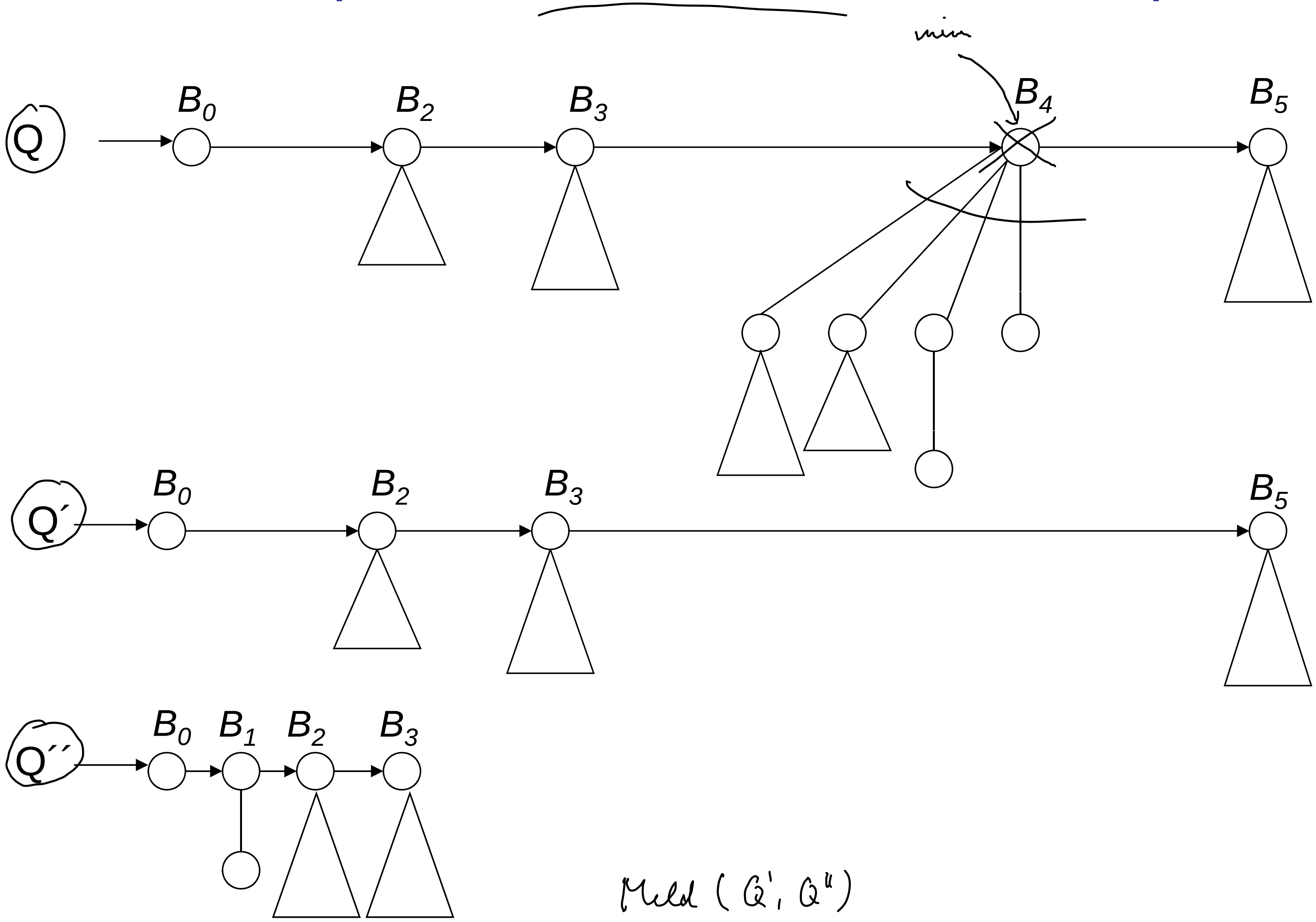
3. *Q´.meld(Q´´)*

Running time: O(log *n*)

Binomial queues: 'deletemin', 1st example



Binomial queues: ‘deletemin’, 2nd example



Binomial queues: ‘decreasekey’

Q.decreasekey(v, k):

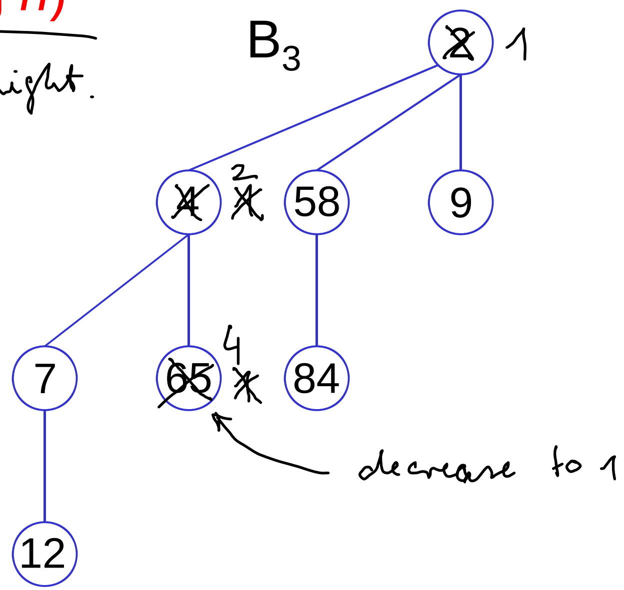
1. *v.element.key := k*
2. Repeatedly exchange *v.element* with the element of *v*‘s parent, until the heap property is restored.

Running time: O (log *n*)

Proportional to the height.

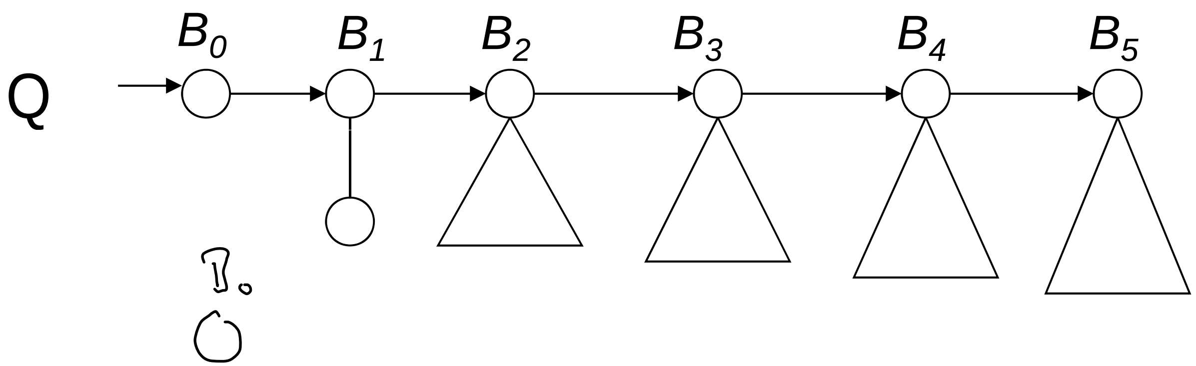
B_i : 2^i nodes

i height

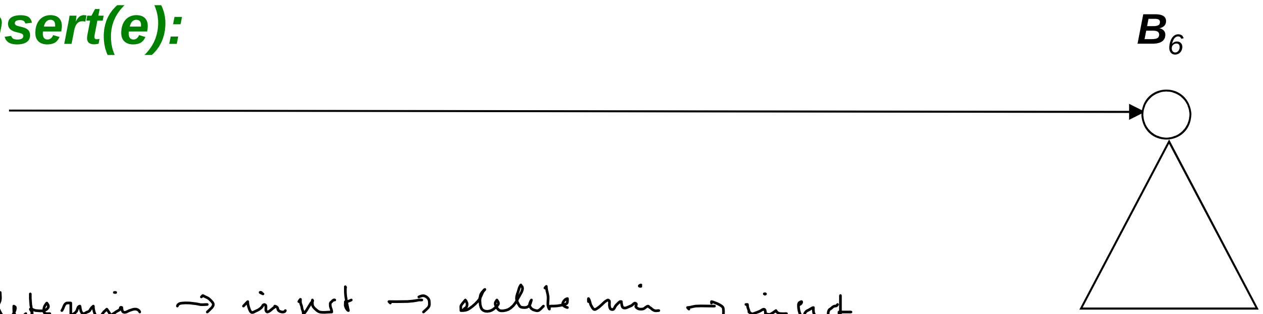


Binomial queues: worst case sequence

Q.deletemin():



Q.insert(e):



Running time:
$\Theta(\log n)$

deletemin → insert → deletemin → insert ...