# Algorithm Theory

# 07 – Binomial Queues

Dr. Alexander Souza

## Priority queues: operations

(Priority) queue *Q*

Data structure for maintaining a set of elements, each having an associated priority from a totally ordered universe. The following operations are supported.

**Operations:**

*Q.initialize():* initializes an empty queue *Q*

*Q.isEmpty():* returns true iff *Q* is empty

*Q.insert(e):* inserts element *e* into *Q* and returns a pointer to the node containing *e*

*Q.deletemin():* returns the element of *Q* with minimum key and deletes it

*Q.min():* returns the element of *Q* with minimum key

*Q.decreasekey(v,k):* decreases the value of *v*'s key to the new value *k*

# Priority queues: operations

**Additional operations:**

*Q.delete(v)*: deletes node *v* and its element from *Q* (without searching for *v*)

*Q.meld(Q´):* unites *Q* and *Q´* (concatenable queue)

*Q.search(k)*: searches for the element with key *k* in *Q* (searchable queue)

And many more, e.g. *predecessor, successor, max, deletemax*

# Priority queues: implementations

| | List | Heap | Bin. – Q. | Fib.-Hp. |
|---|---|---|---|---|
| insert | O(1) | O(log n) | O(log n) | O(1) |
| min | O(n) | O(1) | O(log n) | O(1) |
| delete-min | O(n) | O(log n) | O(log n) | O(log n)* |
| meld ($m \le n$) | O(1) | O(n) or O(m log n) | O(log n) | O(1) |
| decr.-key | O(1) | O(log n) | O(log n) | O(1)* |

*= amortized cost

***Q.delete(e) = Q.decreasekey(e, -∞ ) + Q.deletemin( )***

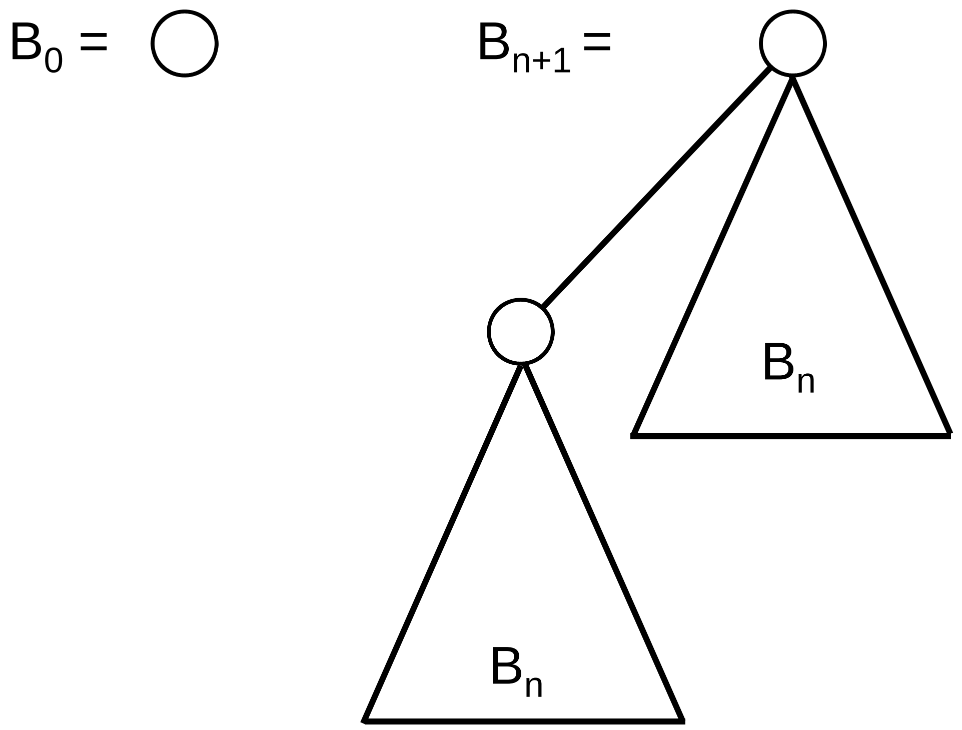
## Unsorted List & Heap

Unsorted List

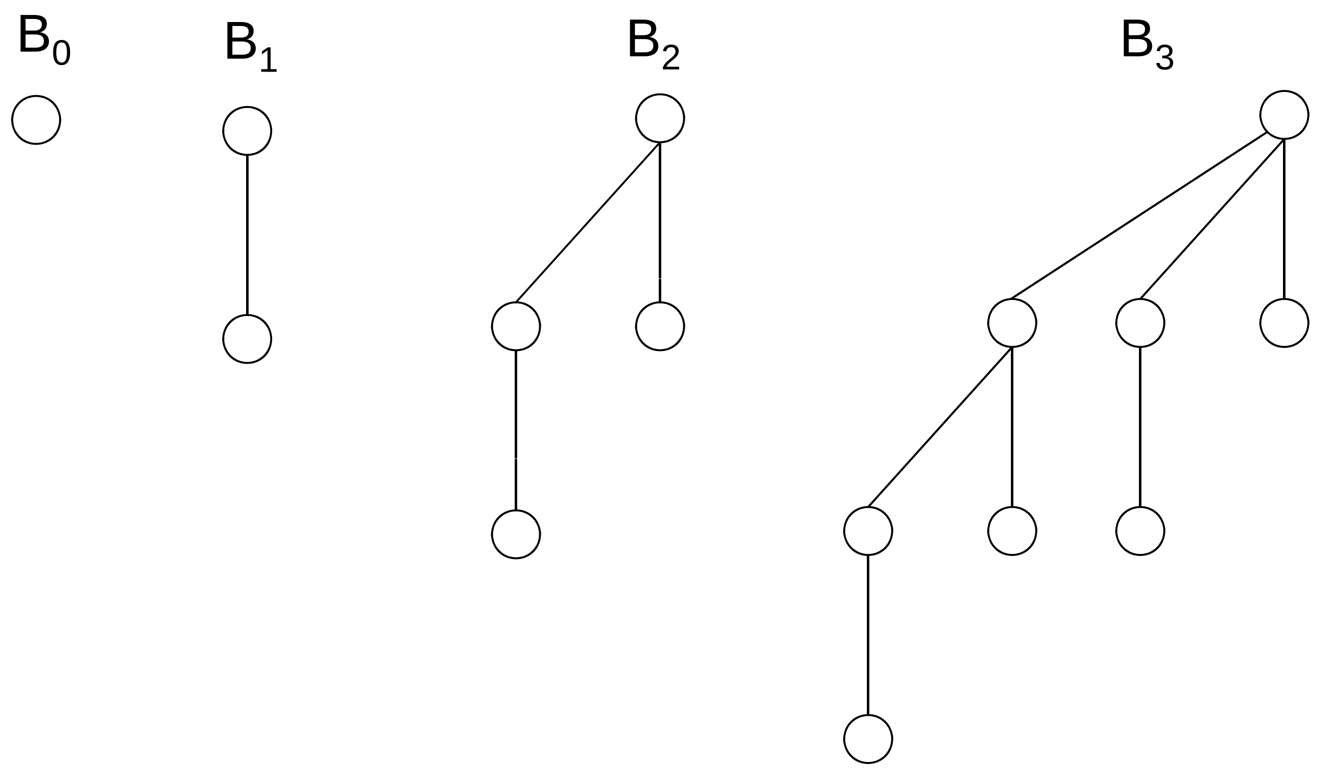
Heap (Min-Heap)

## Definition

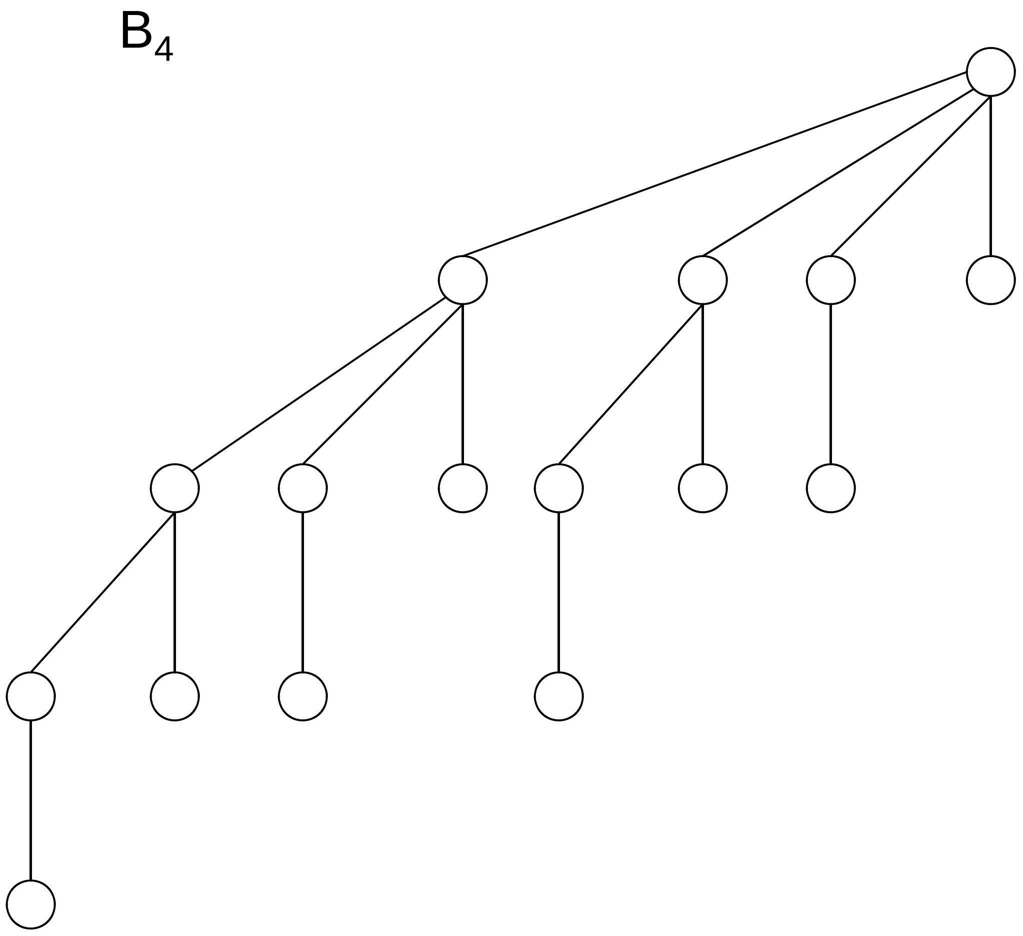
Binomial tree $B_n$ of order *n* $(n \geq 0)$



## Binomial trees

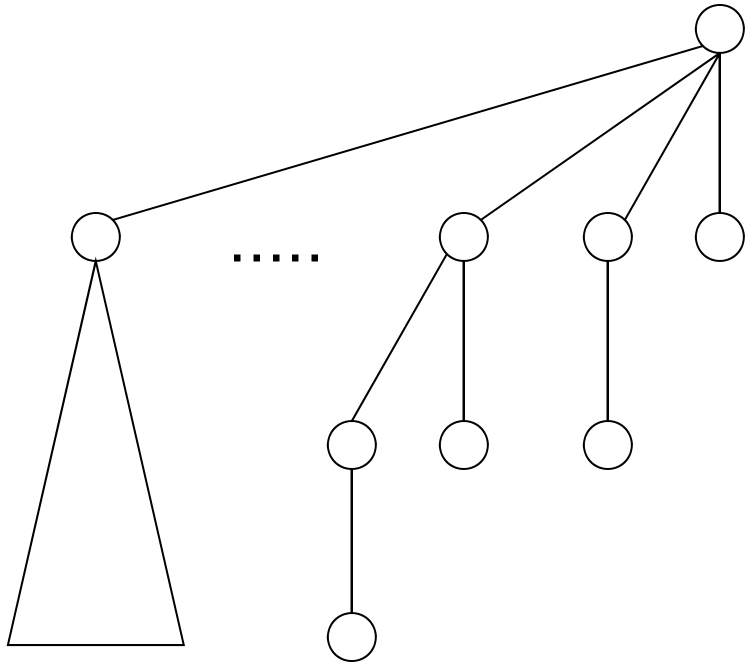


## Binomial trees



## Properties

1. $B_n$ contains $2^n$ nodes.
2. The height of $B_n$ is $n$.
3. The root of $B_n$ has degree $n$.
4. $B_n$ =



5. There are exactly $\binom{n}{i}$ nodes at depth $i$ in $B_n$ .

## Binomial coefficients

$\binom{n}{i}$ = # *i*-element subsets that can be chosen from an *n*-element set

Pascal‘s triangle:

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1
\end{array}$$

## Number of nodes at depth *i* in $B_n$

There are exactly $\binom{n}{i}$ nodes at depth *i* in $B_n$ .

# Binomial queues

**Binomial queue *Q*:**

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

***n* keys:**

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

**9 keys:**

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

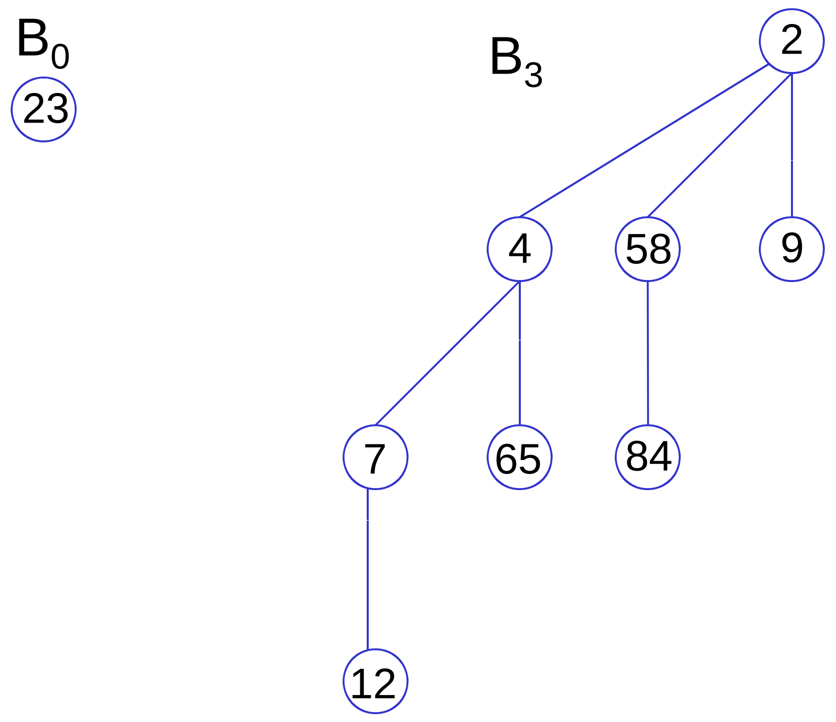
$9 = (1001)_2$

# 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

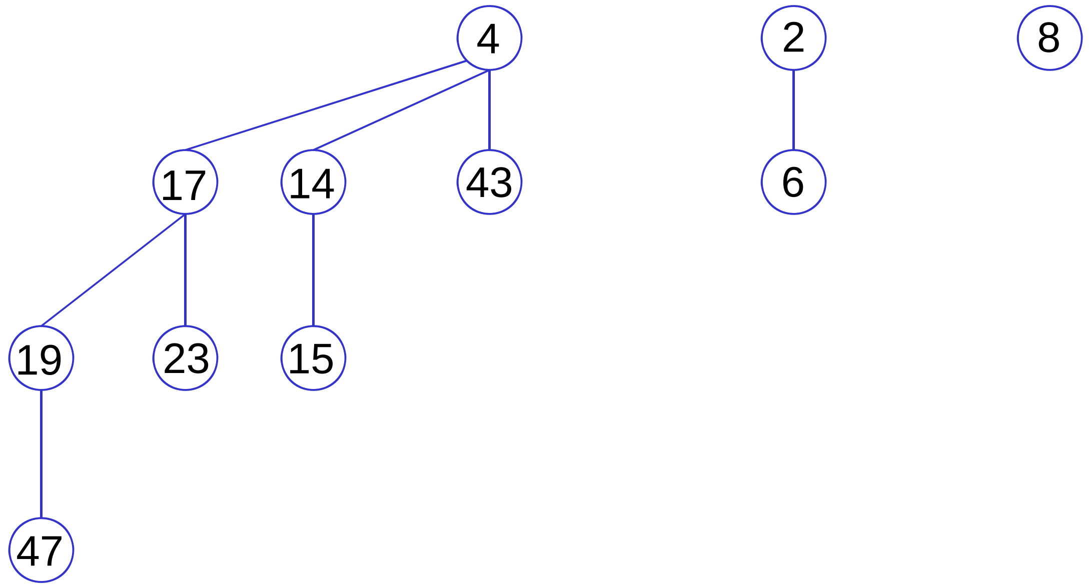
11 keys:

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

$11 = (1011)_2 \rightarrow$ 3 binomial trees

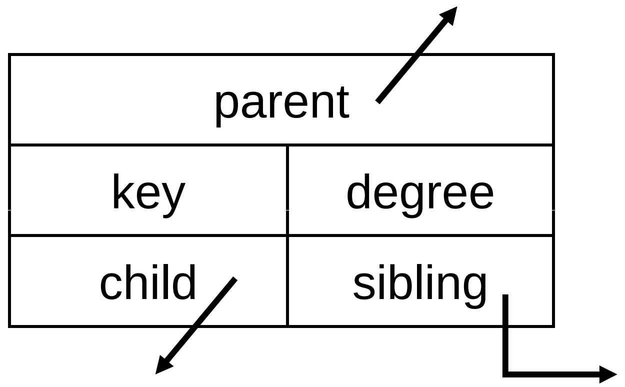
$B_3$, $B_1$ and $B_0$

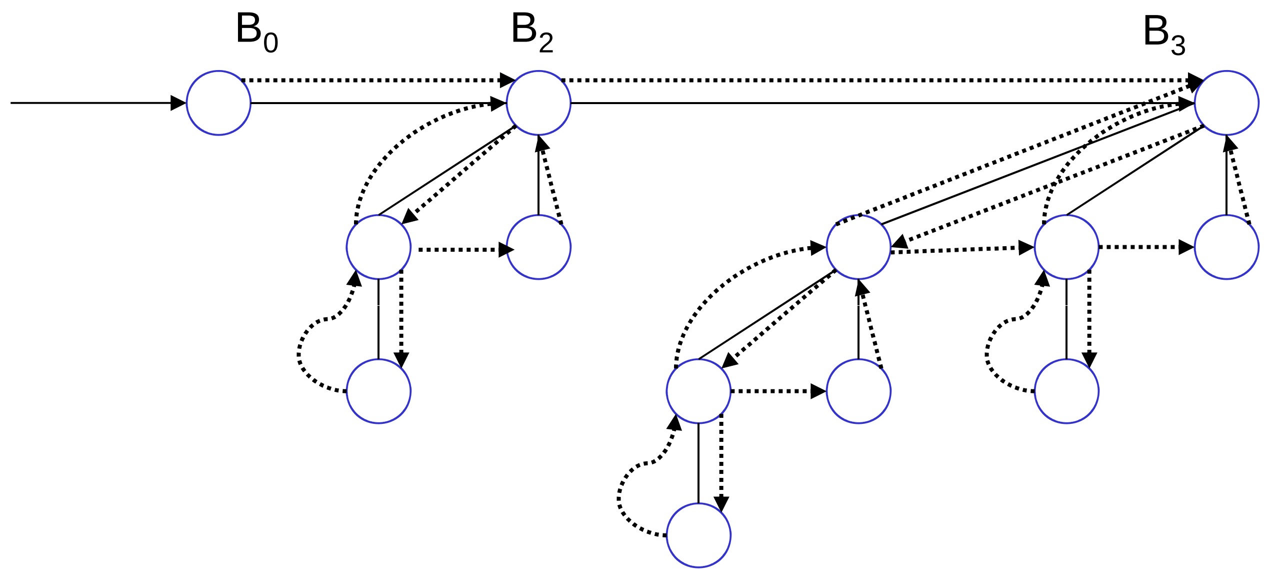
$Q_{11}$:



# Child-sibling representation

Structure of a node:





## Binomial trees: operation ‘meld’ (‘link’)

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

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

**procedure Link:**

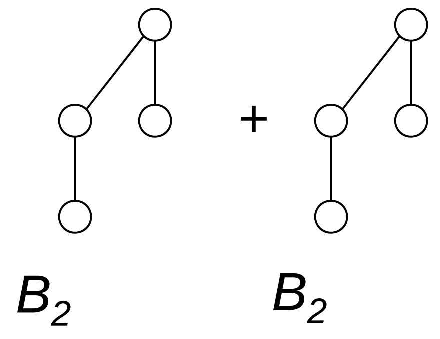
*B.Link(B´)*

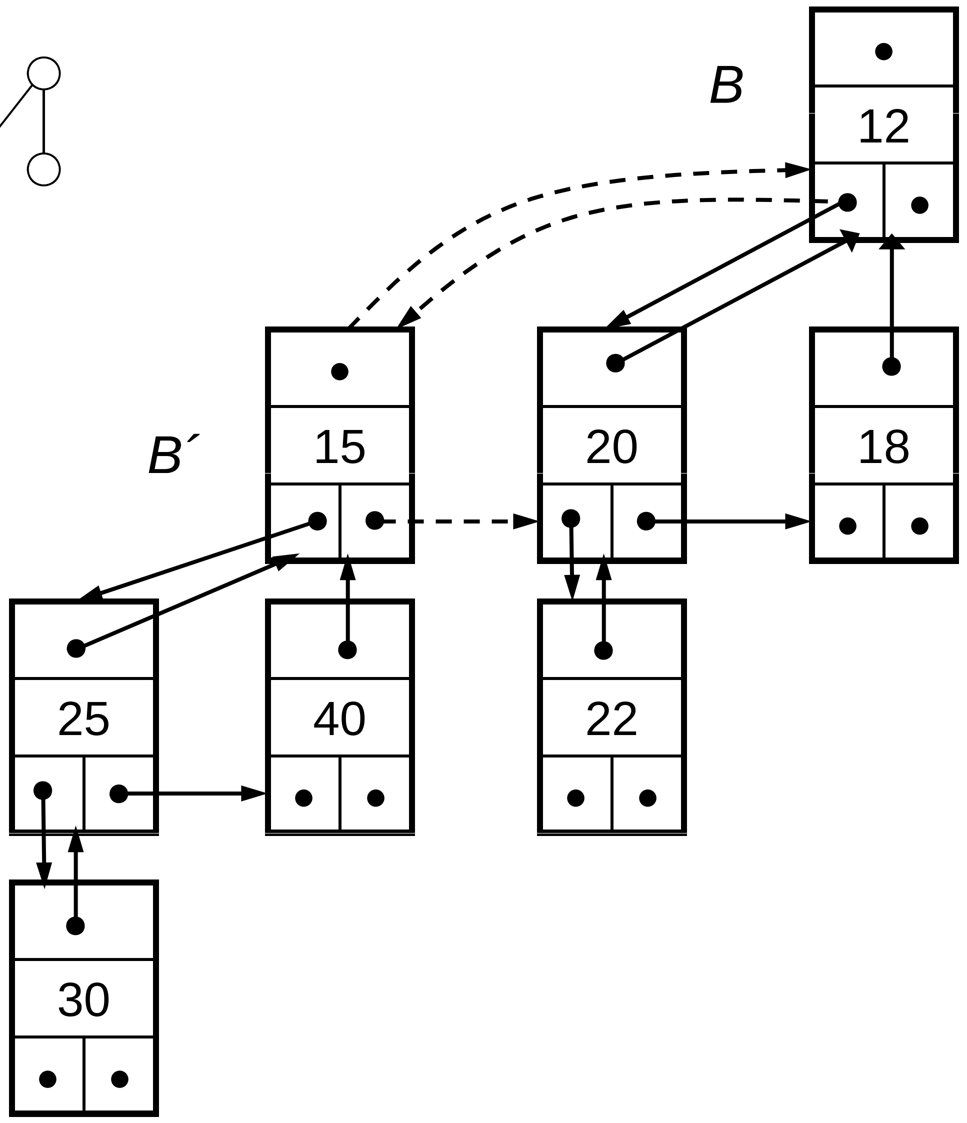
/* Make the root with the larger key a child of the root with the smaller key. */

```
1   if $B.key > B´.key$
2      then $B´.Link(B)$
3              return
       /* $B.key \le B´.key$ */
4   $B´.parent = B$
5   $B´.sibling = B.child$
6   $B.child = B´$
7   $B.degree = B.degree + 1$
```

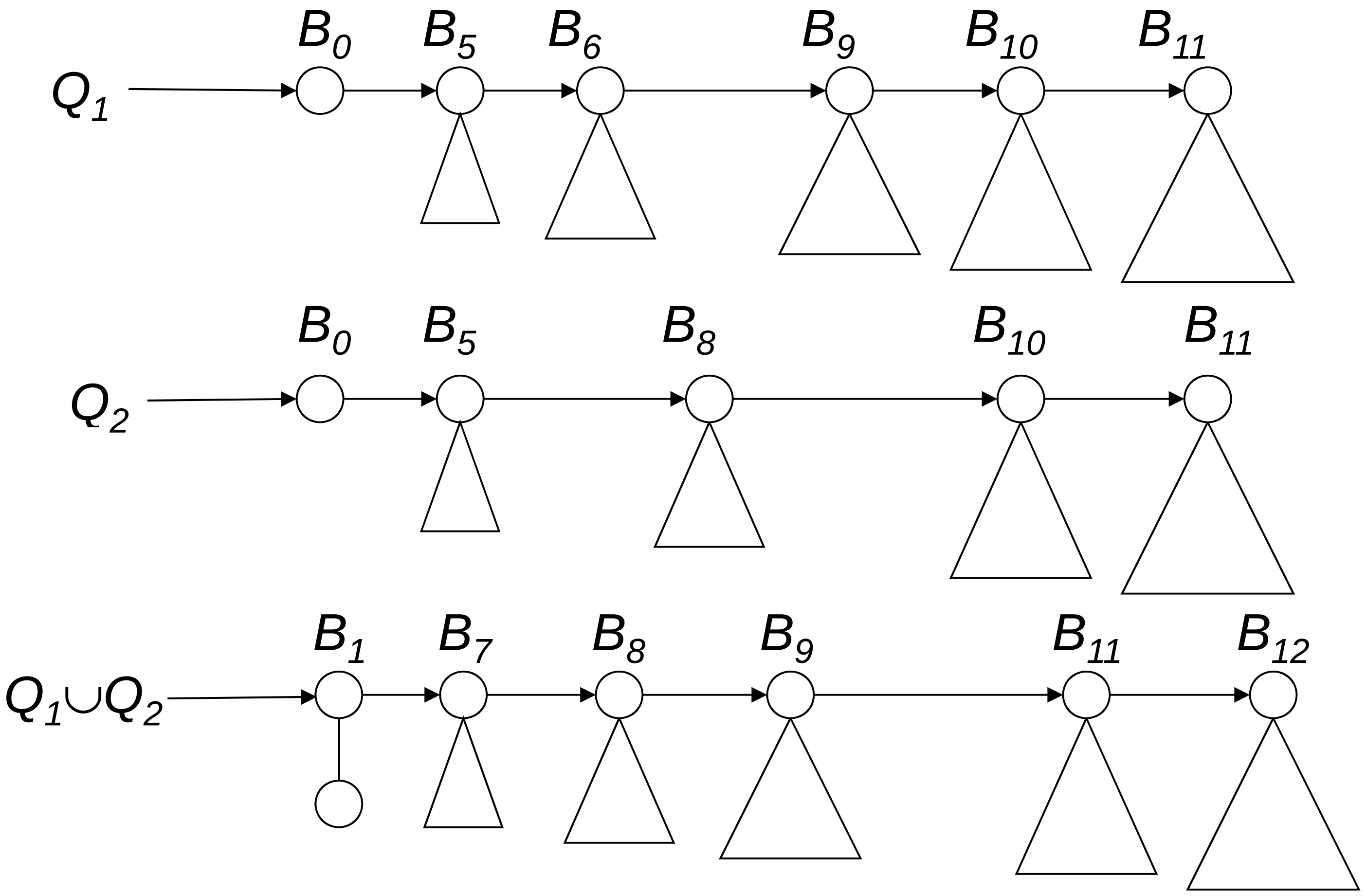
Running time O(1)

# Example of the operation 'link'





# Binomial queues: operation 'meld'



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 $n$)

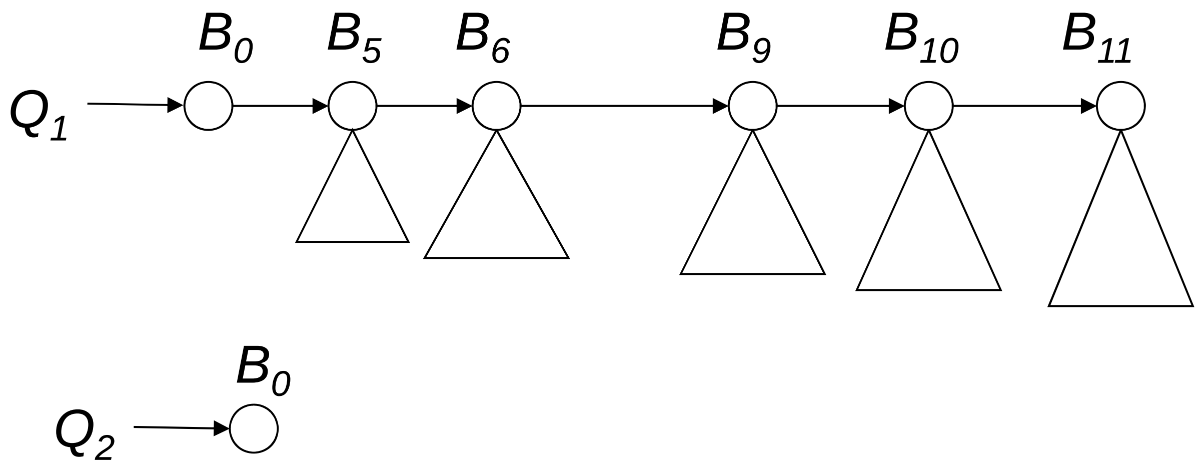
## Binomial queues: operations

*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’

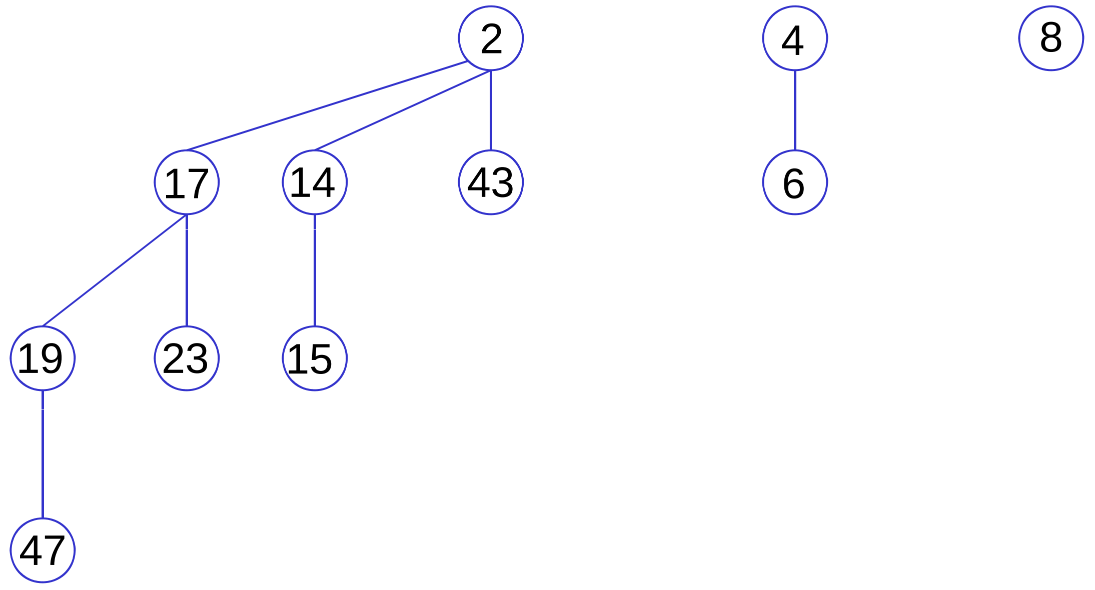
*Q.deletemin():*

1. Determine $B_i$ whose root has the minimum key in the root list and delete $B_i$ from $Q$ (returns Q´)
2. Insert the children of $B_i$ in reverse order into a new queue : $B_0$ , $B_1$ , ..... , $B_{i-1}$ $\rightarrow$ *Q´´*
3. *Q´.meld(Q´´)*

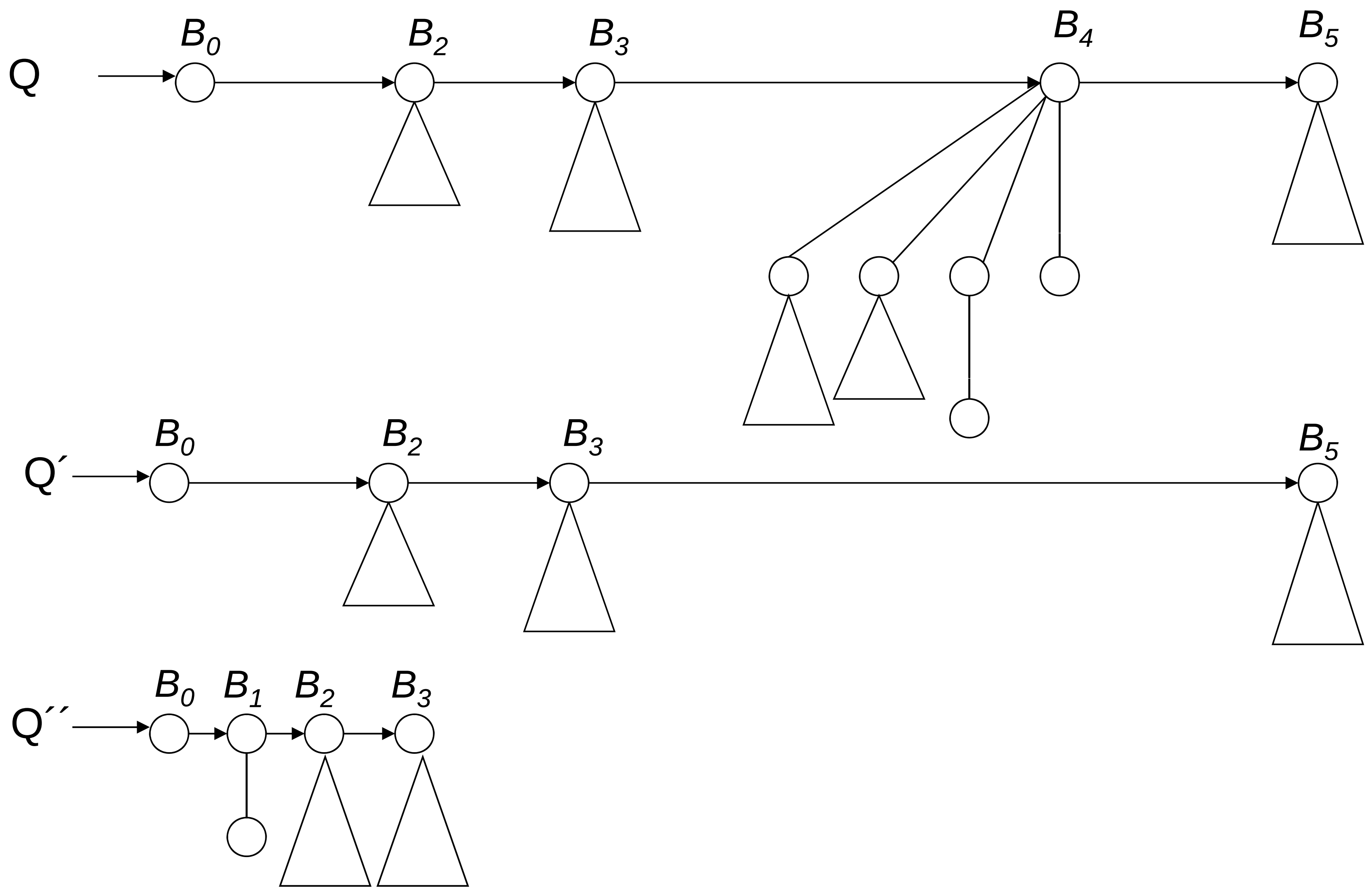
Running time: O(log *n*)

# Binomial queues: ‘deletemin’, 1st example

$Q_{11}$:



## Binomial queues: ‘deletemin’, 2nd example

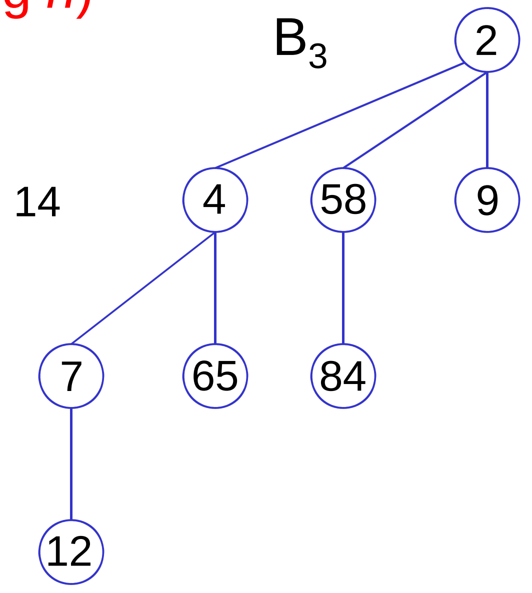


# Binomial queues: ‘decreasekey’

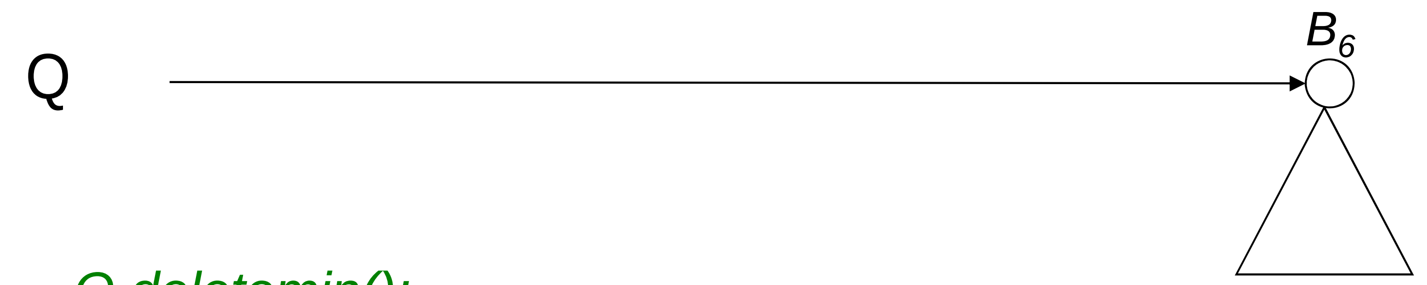
*Q.decreasekey(v, k):*

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

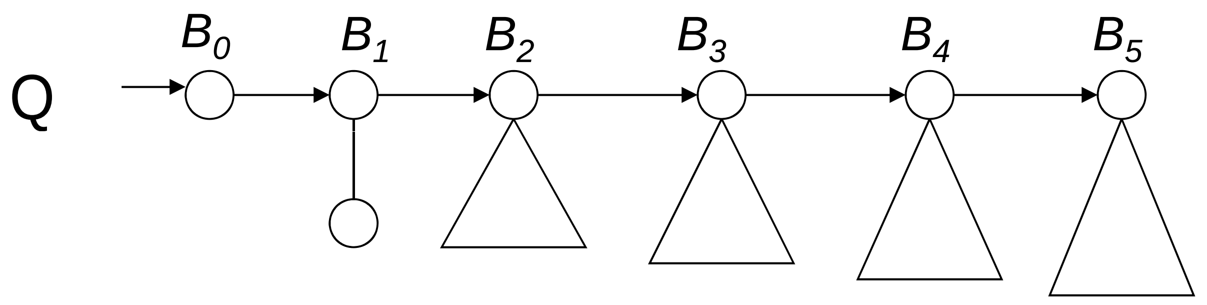
Running time: O (log *n*)



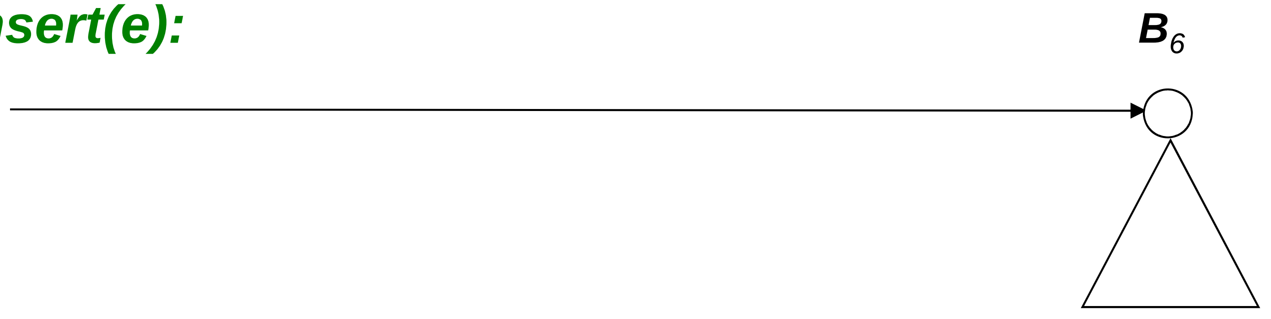
# Binomial queues: worst case sequence



*Q.deletemin():*



***Q.insert(e):***



Running time:
O(log *n*)