ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

15 Binomial Queues

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe
Freiburg

IIF
INSTITUT FÜR
INFORMATIK
FREIBURG

Priority Queues: Operations

- **Priority queue Q**
  - Data structure for maintaining a set of elements, each having an associated priority
- **Operations:**
  - Q.initialize():
    - creates empty queue Q
  - Q.isEmpty():
    - returns true iff Q is empty
  - Q.insert(e):
    - inserts element e in to Q and returns a pointer to a 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

Priority Queues: Operations

- **Additional Operations:**
  - Q.delete(v):
    - deletes node v and its elements from Q
    - v is a pointer to the element (no search)
  - Q.meld(Q´):
    - unites Q and Q´ (concatenable queue)
  - Q.search(k):
    - searches for the element with key k in Q (searchable queue)
- **possibly many more,**
  - e.g. predecessor, successor, max, deletemax

Priority Queues: Implementations

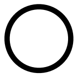
	List	Heap	Binomial Queue	Fibonacci Heap
insert	O(1)	O(log n)	O(log n)	O(1)
min	O(n)	O(1)	O(log n)	O(1)
delete-min	O(n)	O(log n)	O(log n)	O(log n)*
meld (m≤n)	O(1)	O(n) or O(m log n)	O(log n)	O(1)
decrease-key	O(1)	O(log n)	O(log n)	O(1)*

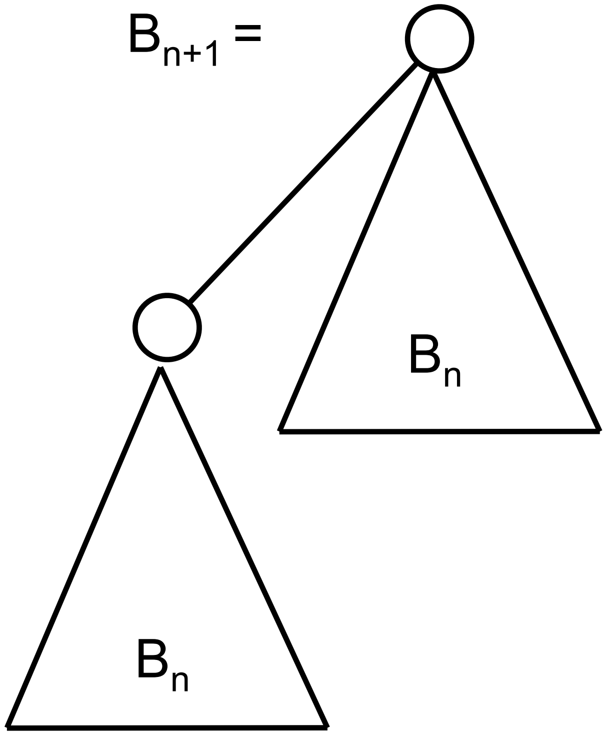
* = amortized cost

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

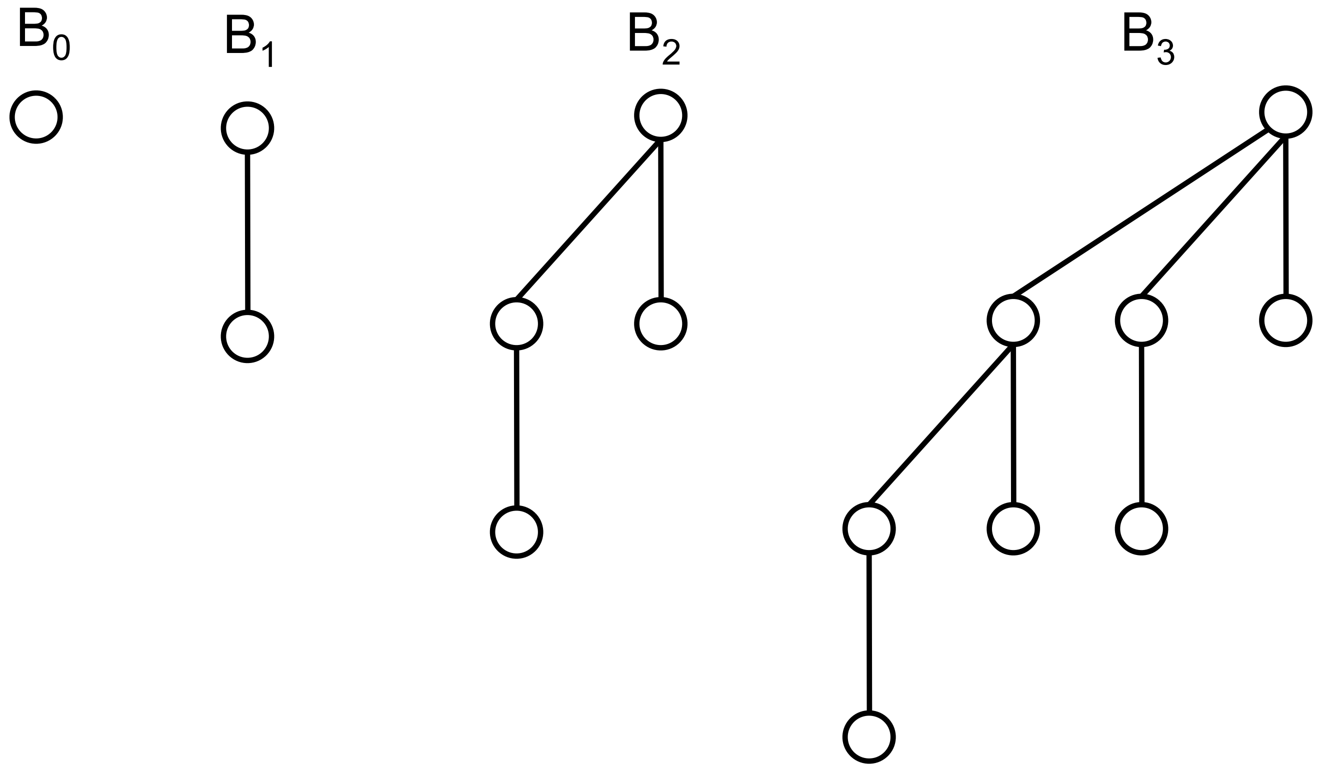
Definition

Binomial tree B_n of order n, $n \geq 0$

B_0 = 

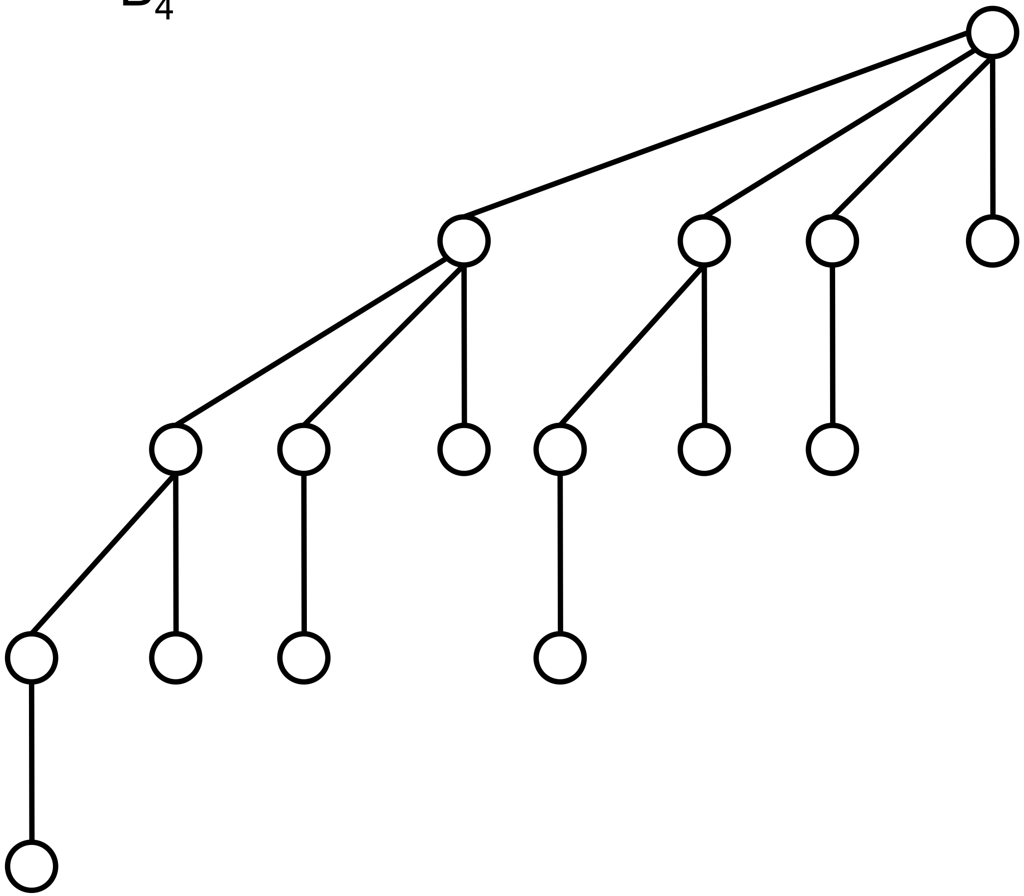
B_{n+1} = 

Binomial Trees



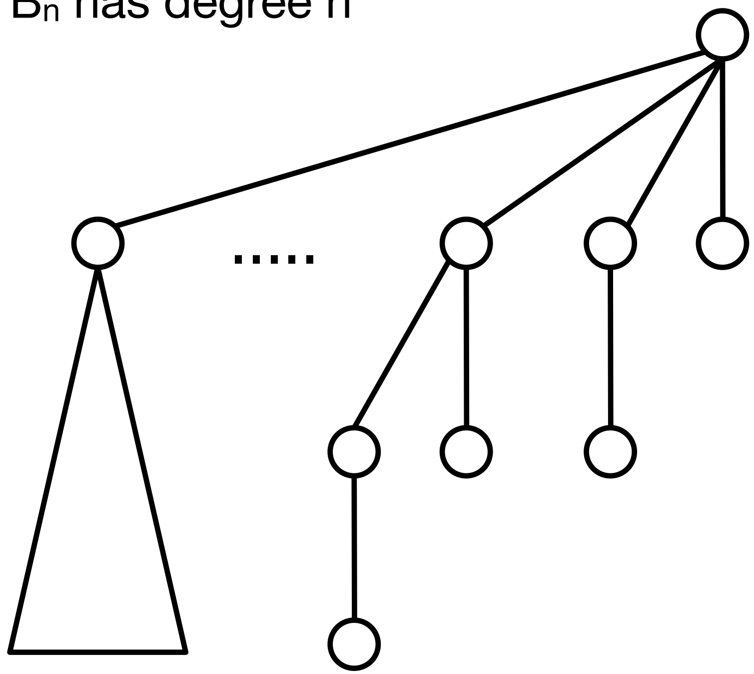
Binomial Trees

B_4



Properties

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



5. There are $\binom{n}{i}$ nodes in 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:

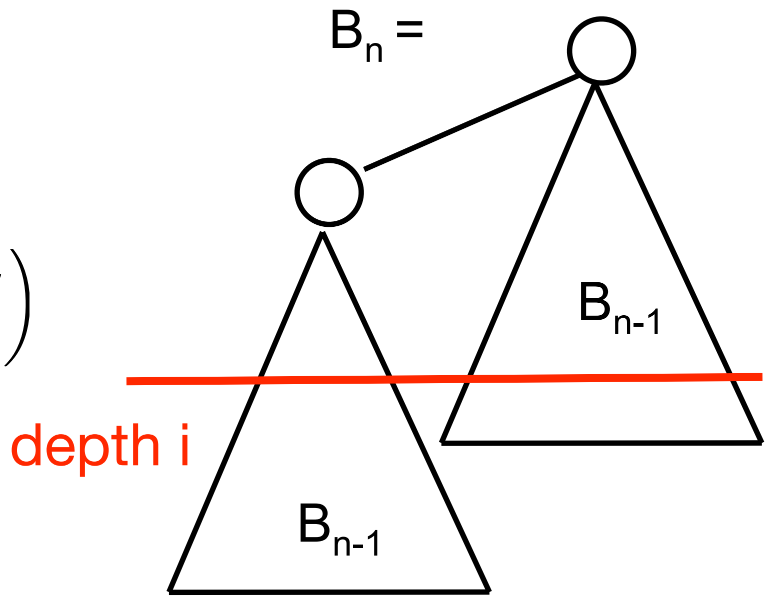
$$
\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1
\end{array}
$$

Number of Nodes at Depth i in B_n

- There are exactly $\binom{n}{i}$ nodes at depth i in B_n
- Proof by induction:
  - n=0

    $$\binom{0}{0} = 1$$

  - n>0
    - $$\binom{n-1}{i} + \binom{n-1}{i-1} = \binom{n}{i}$$



Binomial Queues

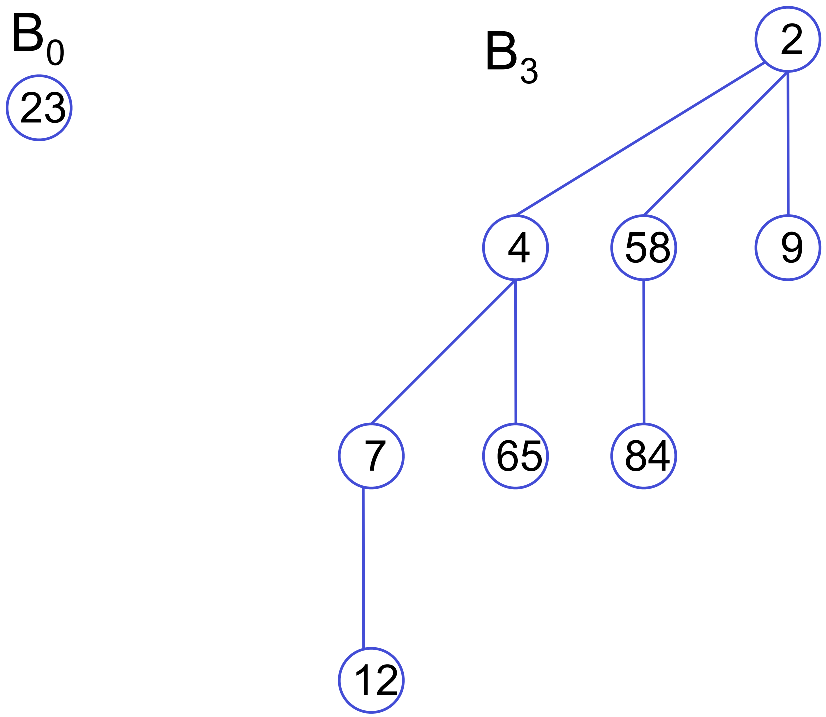
- **Binomial queue Q:**
  - Set of heap ordered binomial trees of different order to store keys.
- **n keys**
  - $B_i \in Q$ $\Leftrightarrow$ i-th Bit in $(n)_2 = 1$
- **9 keys:**
  - {2, 4, 7, 9,12, 23, 58, 65, 85}
  - $9 = (1001)_2$

Binomial Queues: 1st Example

9 keys:

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

$9 = (1001)_2$



Min can be computed in time O(log *n*)

Binomial Queues: 2nd Example

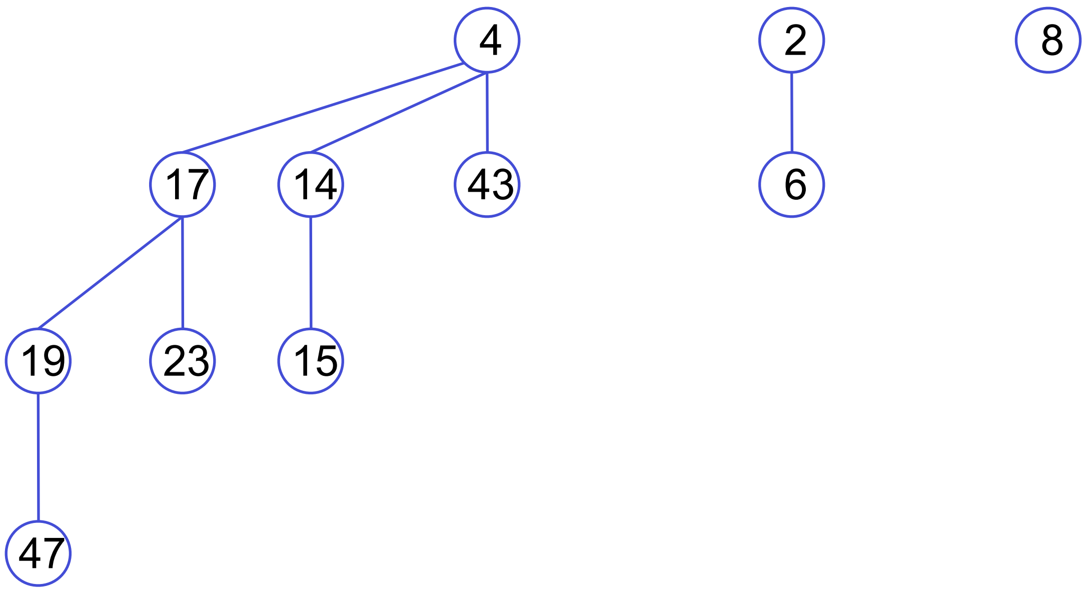
11 keys:

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

$11 = (1011)_2$ → 3 binomial trees

B_3, B_1, and B_0

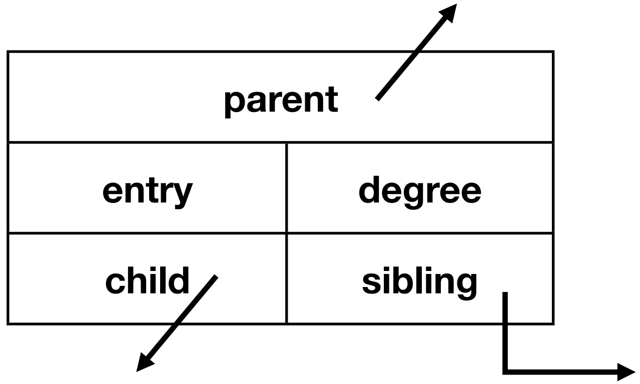
Q_{11}:

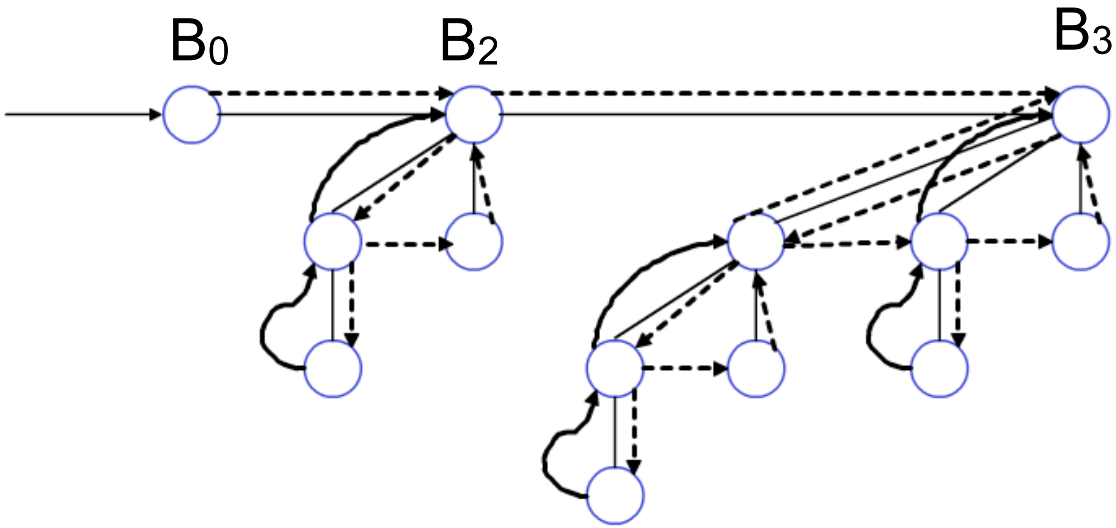


Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

Child - Sibling Representation

Structure of a node:





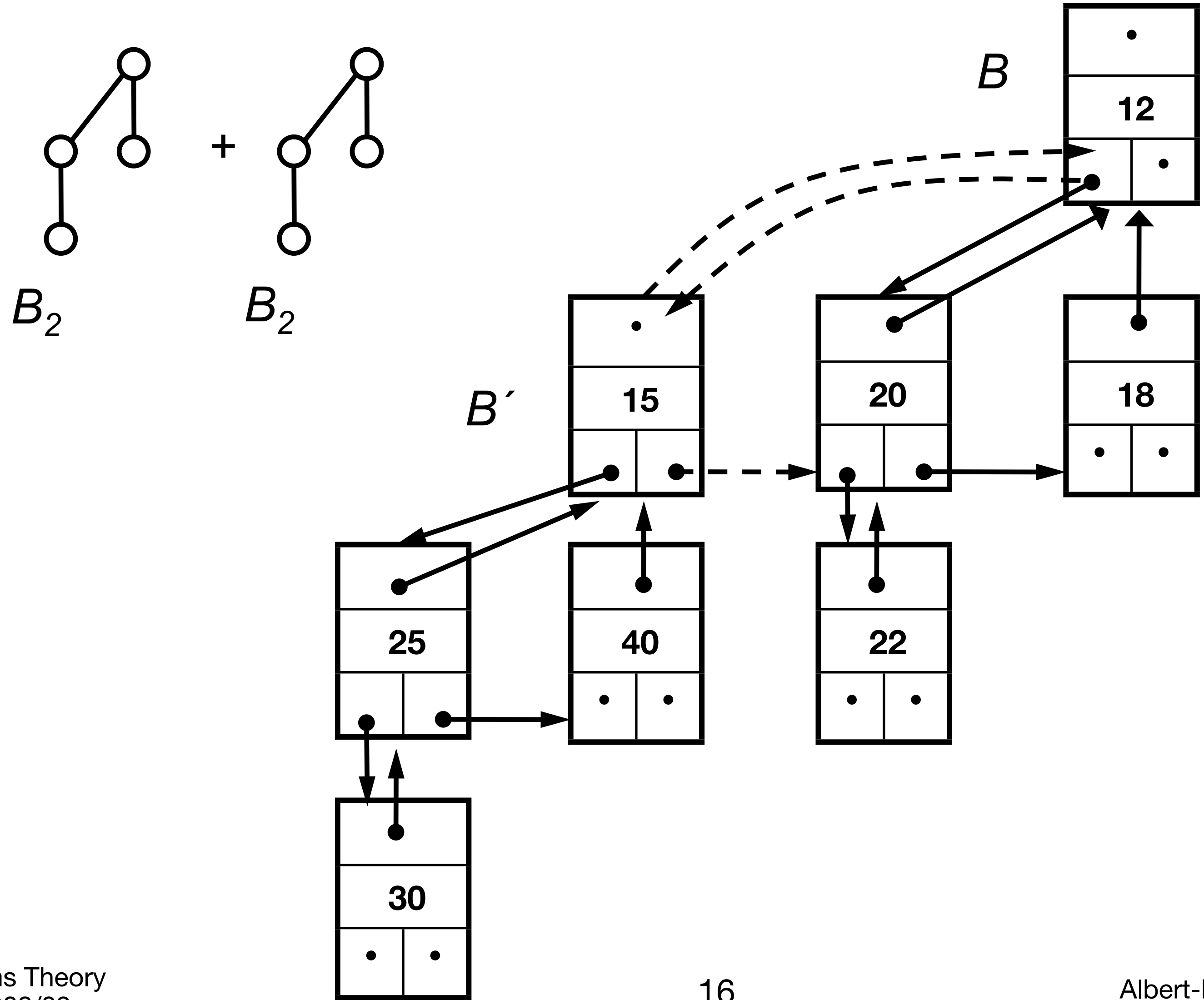
Binomial Trees: Operation Meld (Link)

- Unite two binomial trees B, B´ of same order
  - $B_n + B_n \rightarrow B_{n+1}$
- Procedure Link:
- B.Link(B´)

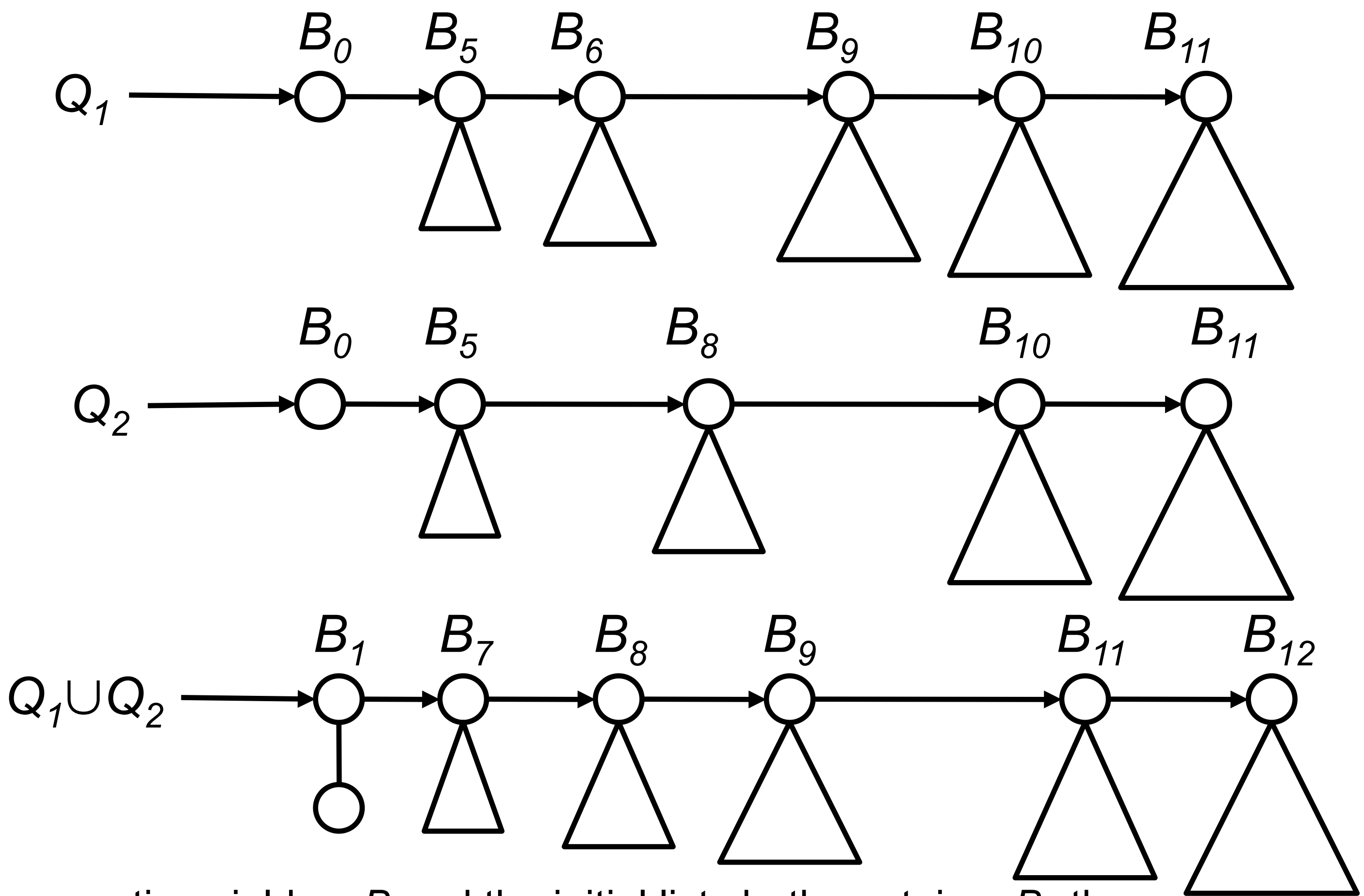
```
/*Make the root with the larger key a child of the root with the smaller key. */
if B.key >  B´.key
   then B´.Link(B)
   return
  /* B.key ≤ B´.key*/
B´.parent = B
B´.sibling = B.chlid
B.child = B
B.degree = B.degree +1
```

- Running Time: O(1)

Example of the Link-Operation



Binomial Queues: Meld-Operation



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

Time: O (log n)

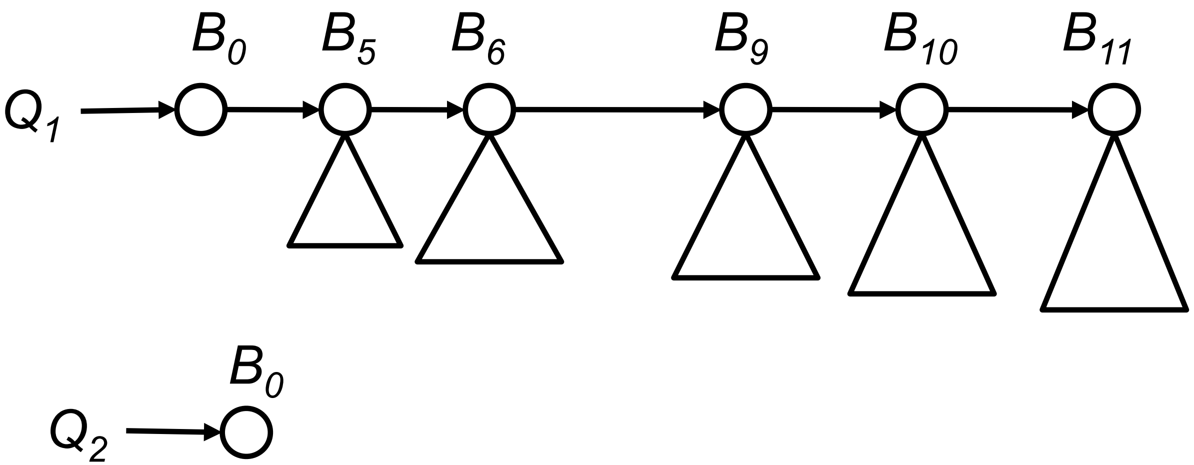
Binomial Queues: Operations

Q.initialize:

Q.root = null

Q.insert(e):

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



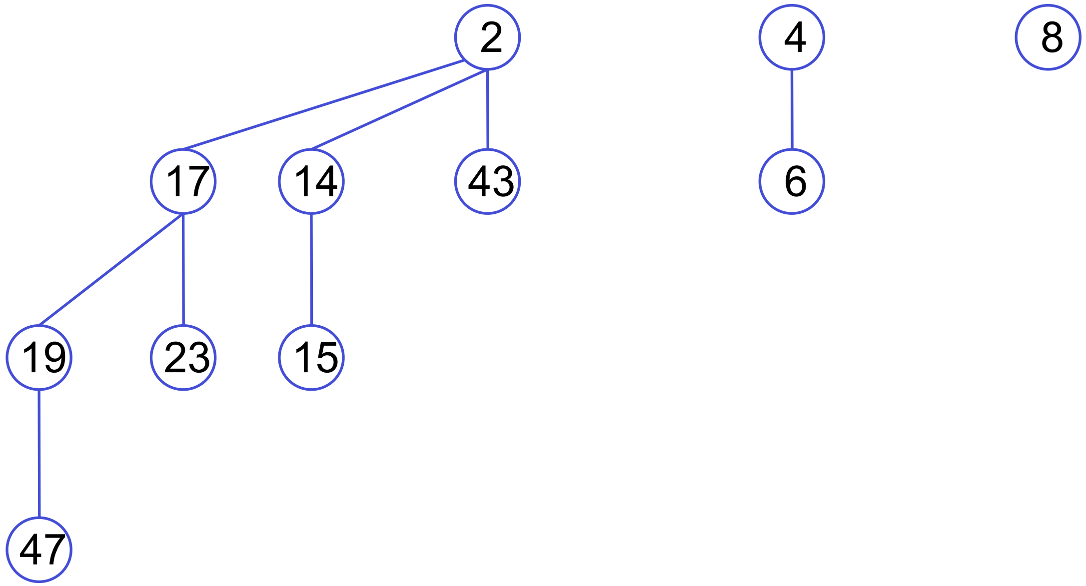
Time = O(log *n*)

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)**

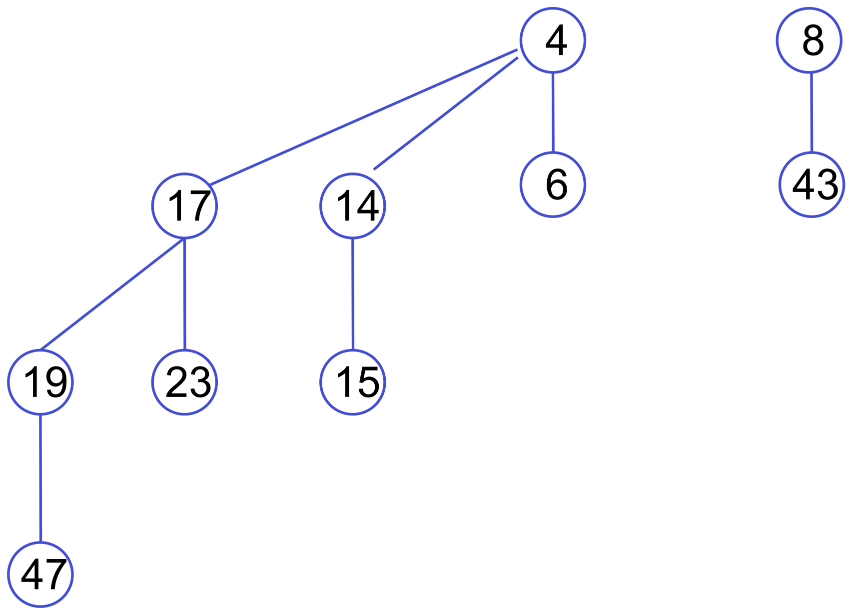
Binomial Queues: Deletemin
1st Example

Q_{11}:

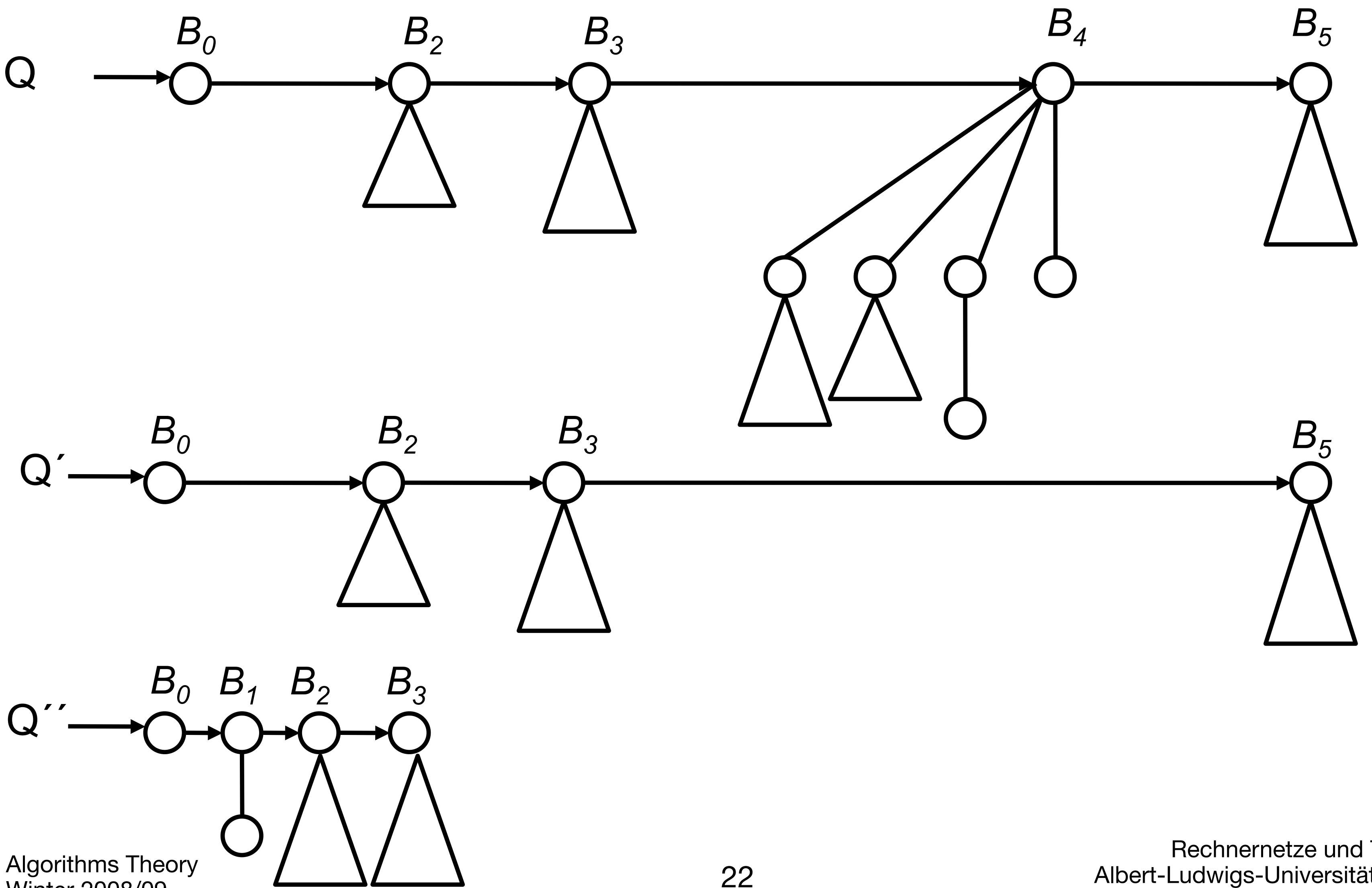


Binomial Queues: Deletemin
1st Example

Q_{11}:

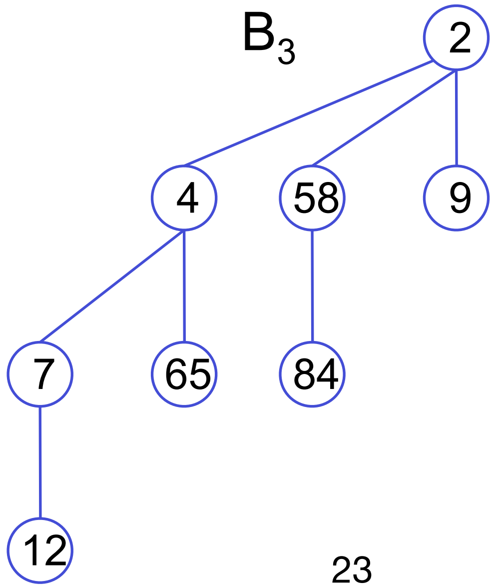


Binomial Queues: Deletemin 2nd Example

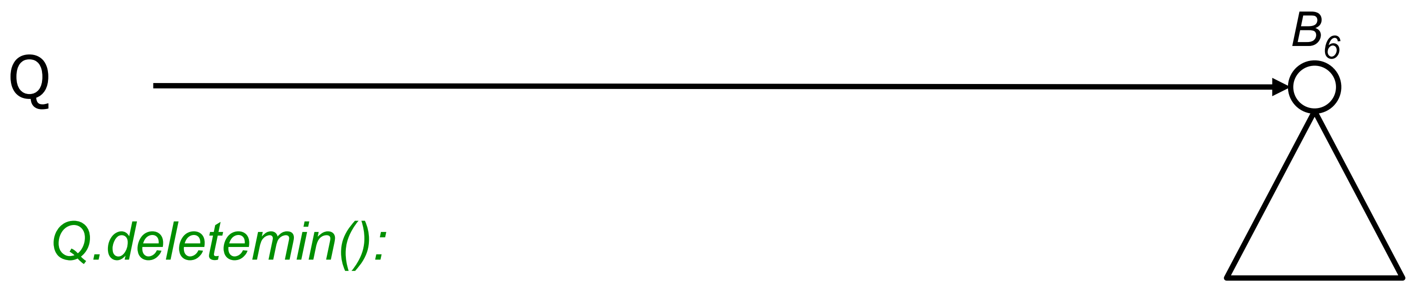


Binomial Queues: Decreasekey

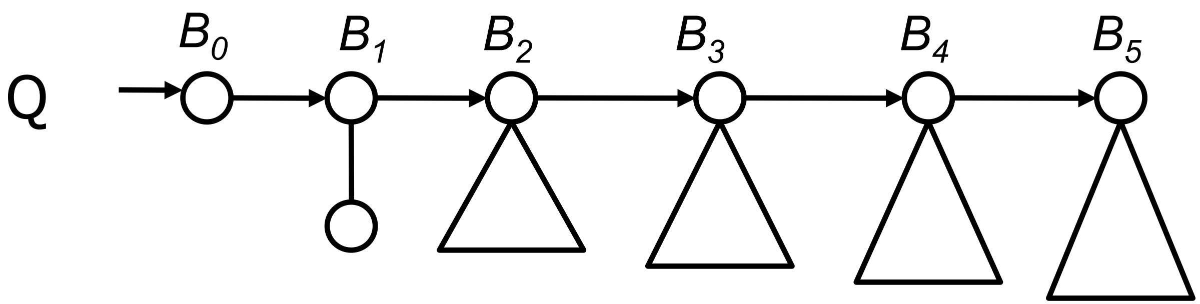
- **Q.decreasekey(v, k):**
  1. v.element.key := k
  2. Repeatedly exchange v.element with the element of v's parent, until the heap property is restored.
- **Running Time: O(log n)**



Binomial Queues: Worst Case Sequence of Operations

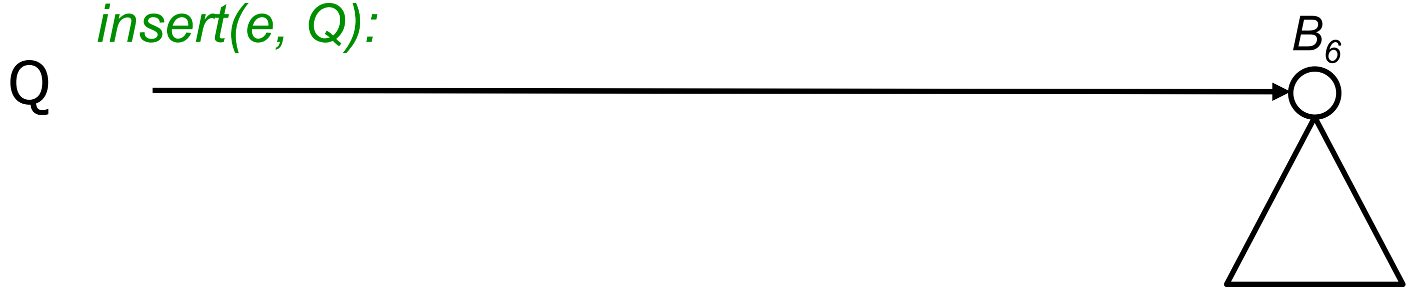


Q.deletemin():



Running Time: O(log n)

insert(e, Q):



ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

15 Binomial Queues

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik

Wintersemester 2007/08

CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG