



Algorithmentheorie

07 – Binomial Queues

Prof. Dr. S. Albers

Vorrangwarteschlangen: Operationen

(Vorrangswarte)schlange (queue) Q

Struktur zur Speicherung von Elementen, für die eine Prioritätsordnung definiert ist, und für die folgende Operationen ausführbar sind:

Operationen:

Q.initialize(): erstellt die leere Schlange Q

Q.isEmpty(): liefert true gdw. Q ist leer

Q.insert(e): fügt Eintrag e in Q ein und gibt einen Zeiger auf den Knoten, der Eintrag e enthält, zurück

Q.deletemin(): liefert den Eintrag aus Q mit minimalem Schlüssel und entfernt ihn

Q.min(): liefert den Eintrag aus Q mit minimalem Schlüssel

Q.decreasekey(v,k): verringert den Schlüssel von Knoten v auf k

Vorrangwarteschlangen: Operationen

Zusätzliche Operationen:

Q.delete(v): entfernt Knoten v mit Eintrag aus Q (ohne v zu suchen)

Q.meld(Q'): vereinigt Q und Q' (concatenable queue)

Q.search(k): sucht den Eintrag mit Schlüssel k in Q (searchable queue)

u.v.a., z.B. *predecessor, successor, max, deletemax*

Vorrangwarteschlangen Implementierungen



	Liste	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 ($m \leq n$)	$O(1)$	$O(n)$ od. $O(m \log n)$	$O(\log n)$	$O(1)$
decr.-key	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)^*$

*= amortisierte Kosten

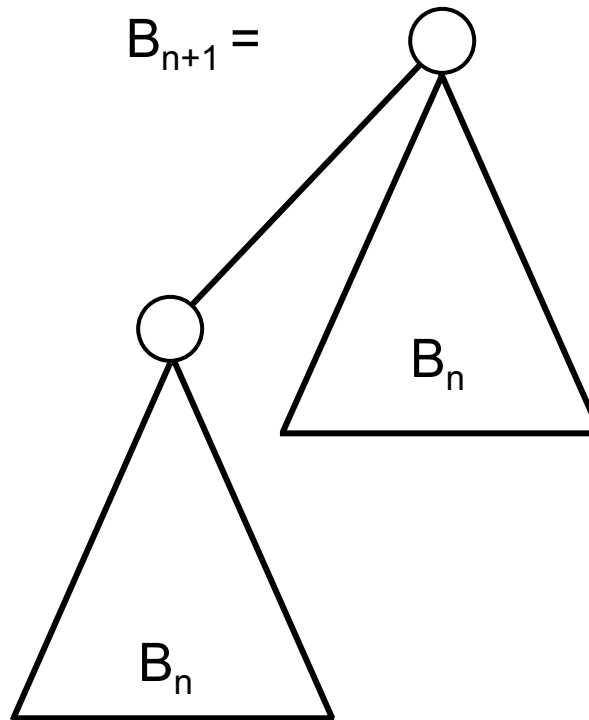
$$Q.delete(e) = Q.decreasekey(e, -\infty) + Q.deletemin()$$

Definition

n -ter Binomialbaum B_n , $n \geq 0$

$$B_0 = \bigcirc$$

$$B_{n+1} =$$



Binomial Bäume

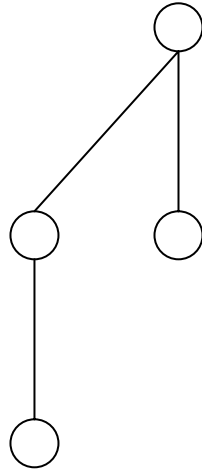
B_0



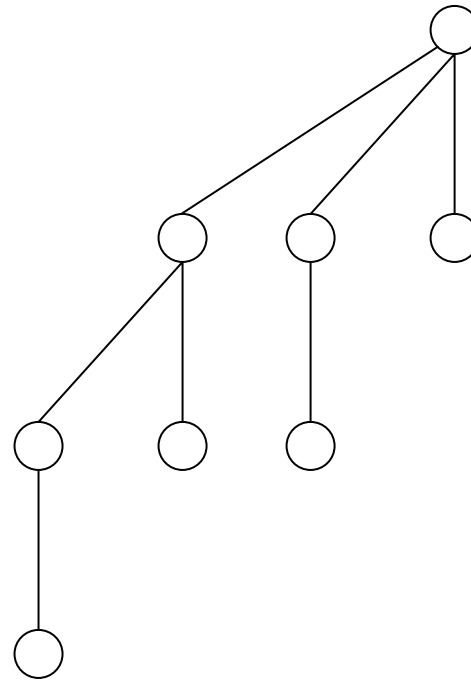
B_1



B_2

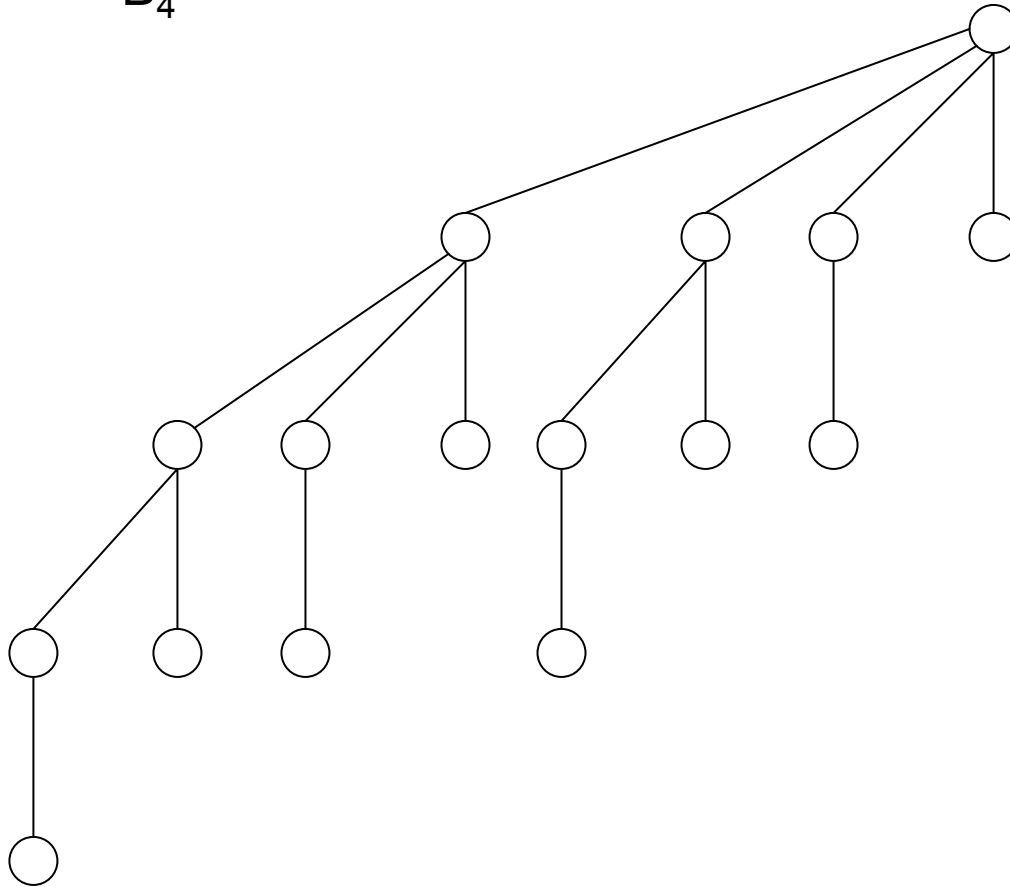


B_3



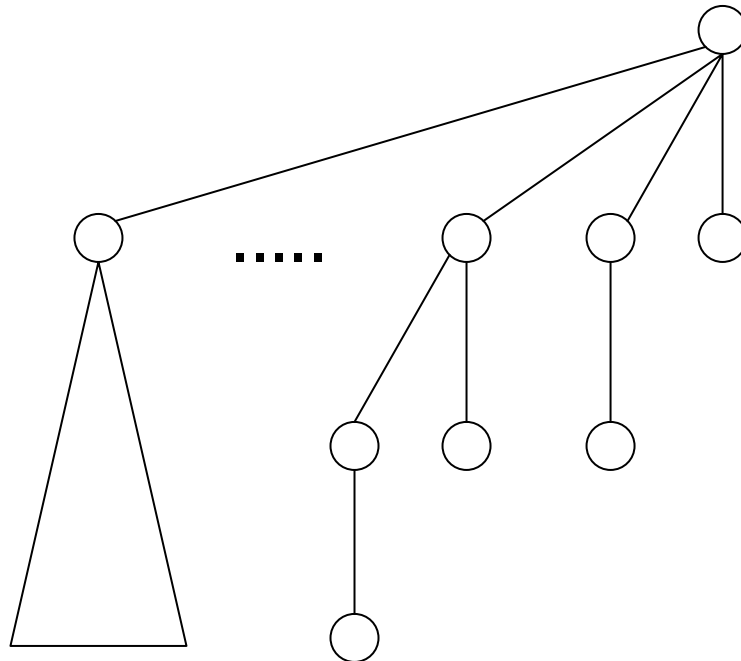
Binomial Bäume

B_4



Folgerung

1. B_n hat 2^n Knoten
2. B_n hat Höhe n
3. Wurzel von B_n hat Grad n (= Ordnung)
4. $B_n =$



5. Es gibt genau $\binom{n}{i}$ Knoten mit Tiefe i in B_n

Exkurs: Binomialkoeffizienten

$$\binom{n}{i} = \# \text{ Möglichkeiten, } i \text{ aus } n \text{ Objekten zu wählen}$$

Pascal'sches Dreieck:

				1					
				1		1			
			1		2		1		
		1		3		3		1	
	1		4		6		4		1

Anzahl Knoten mit Tiefe i in B_n

Es gibt genau $\binom{n}{i}$ Knoten mit Tiefe i in B_n

Binomial Queues

Binomialqueue Q:

Vereinigung **heapgeordneter** Binomialbäume verschiedener Ordnung zur Speicherung von Schlüsseln

n Schlüssel:

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

9 Schlüssel:

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

$$9 = (1001)_2$$

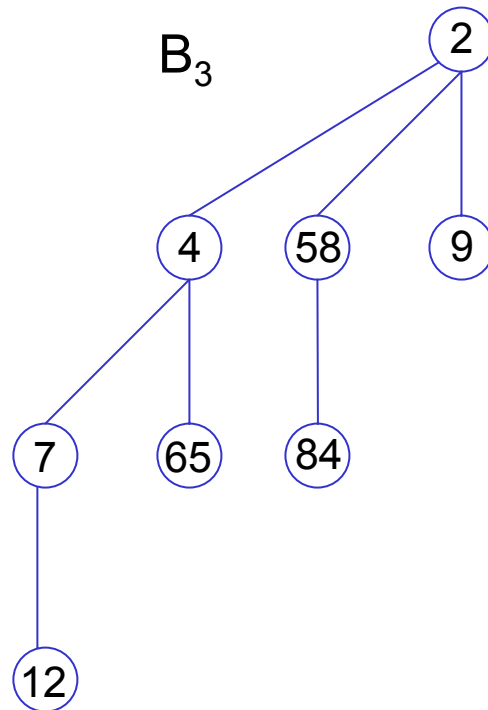
Binomial Queues: Beispiel 1

9 Schlüssel:

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

$9 = (1001)_2$

B_0
23



Min bestimmen in Zeit:
 $O(\log n)$

Binomial Queues: Beispiel 2

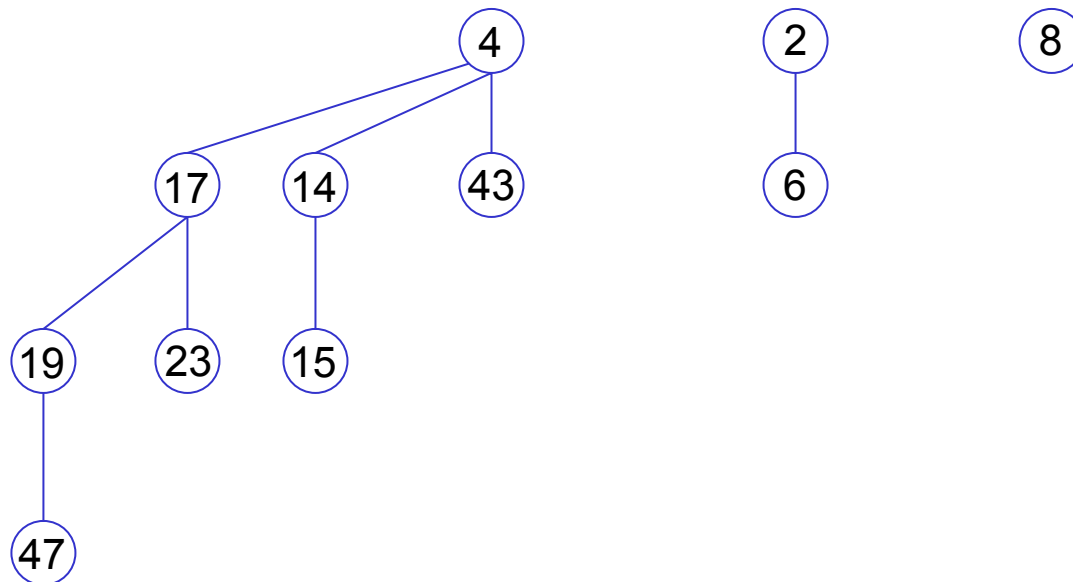
11 Schlüssel:

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

$11 = (1011)_2 \rightarrow 3$ Binomialbäume

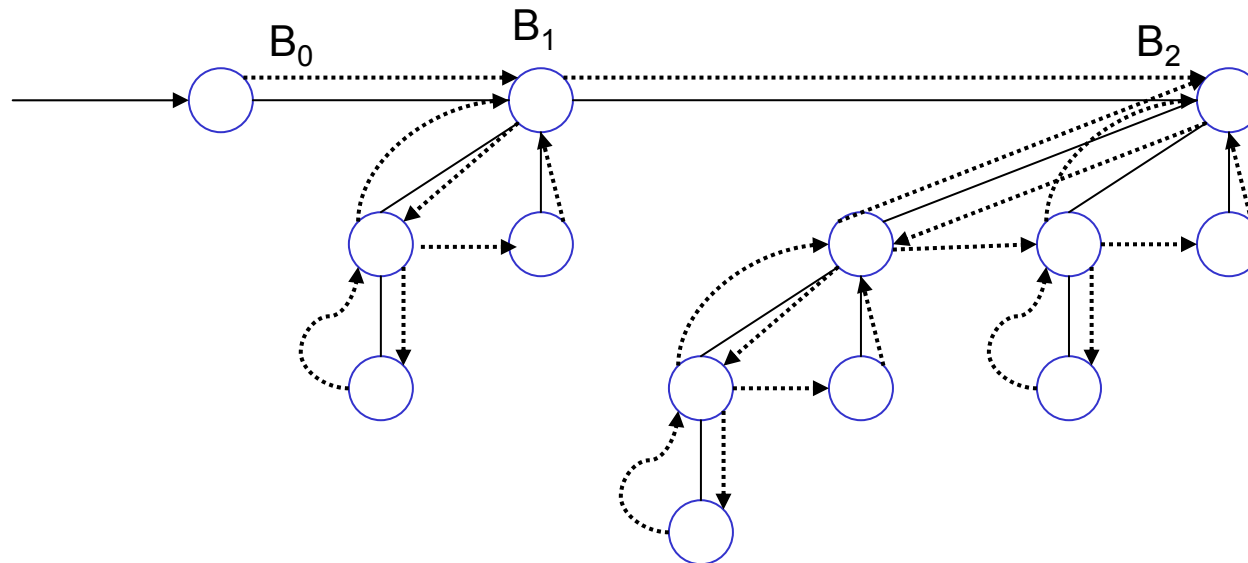
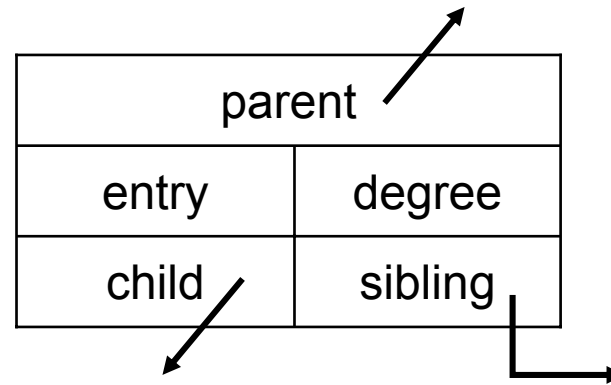
B_3 , B_1 , und B_0

Q_{11} :



Child - Sibling Darstellung

Knotenformat:



Binomialbäume: Vereinigung (Link)

Vereinigung zweier Binomialbäume B , B' von **gleicher** Ordnung

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

Link-Operation:

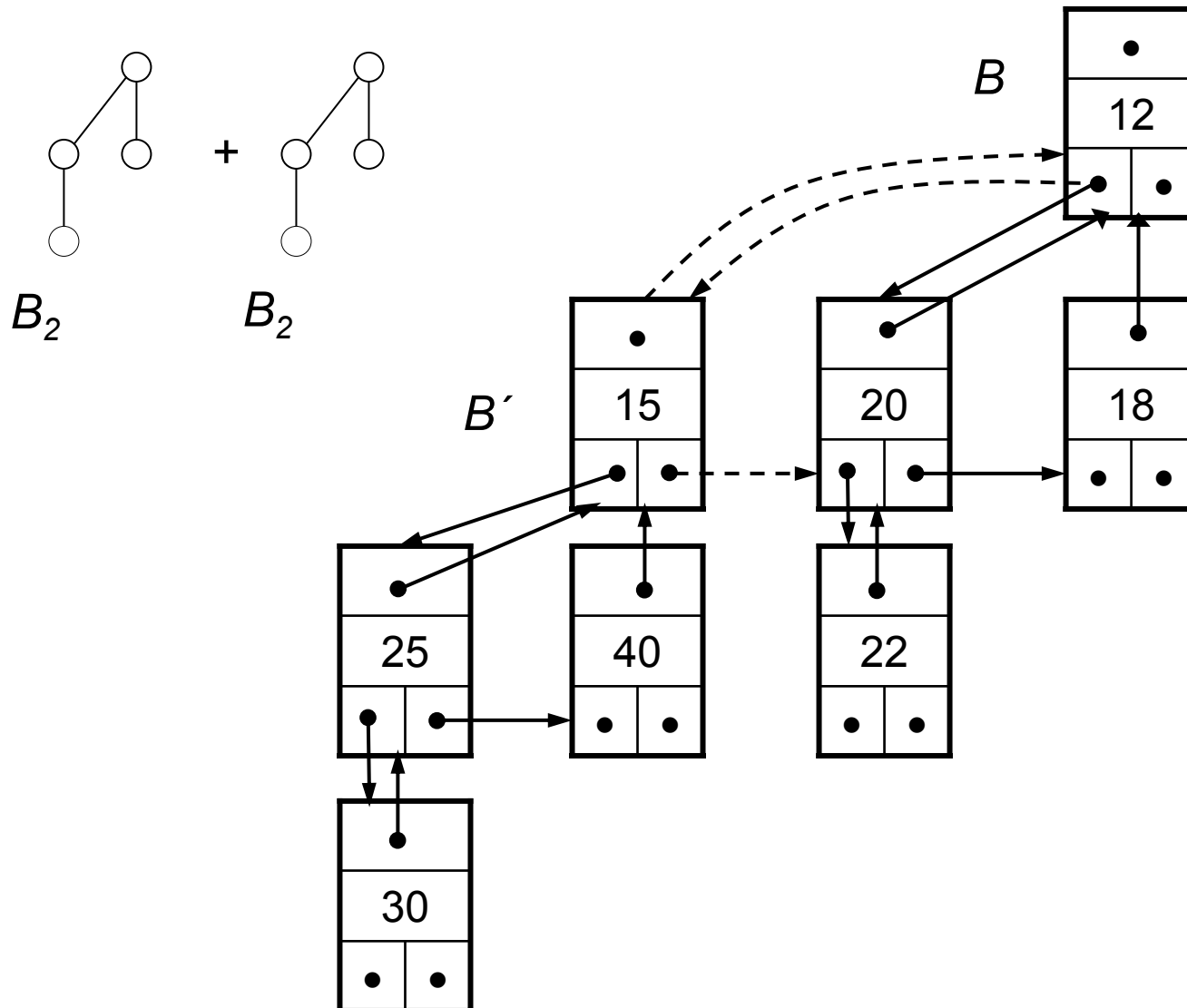
$B.Link(B')$

*/*Mache Wurzel des Baumes mit größerem Schlüssel zum Sohn des Baumes mit kleinerem Schlüssel */*

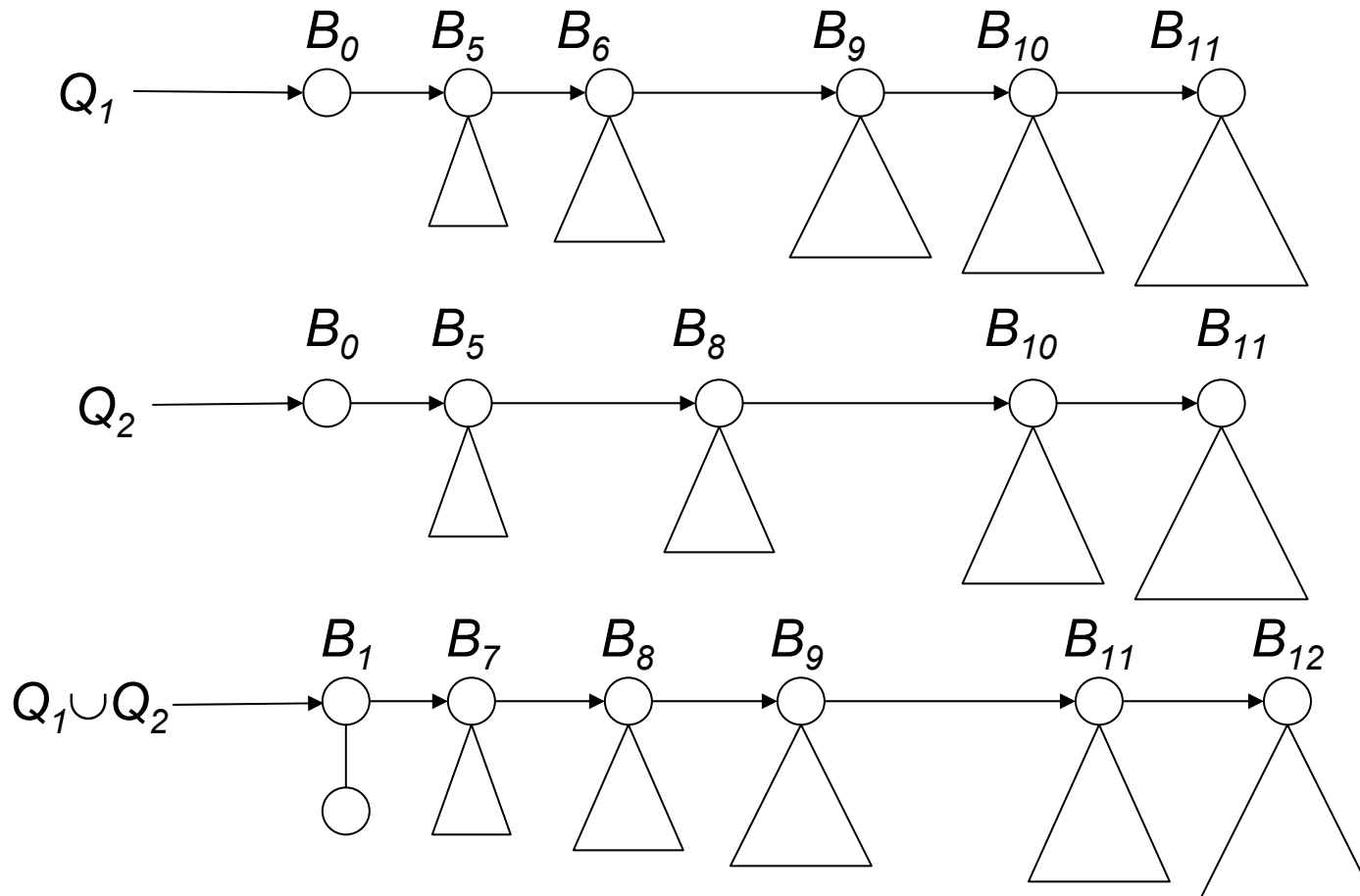
```
1 if  $B.key > B'.key$ 
2   then  $B'.Link(B)$ 
3   return
   /*  $B.key \leq B'.key$ */
4  $B'.parent = B$ 
5  $B'.sibling = B.child$ 
6  $B.child = B'$ 
7  $B.degree = B.degree + 1$ 
```

Zeit $O(1)$

Beispiel zur Link-Operation



Binomial Queues: Vereinigung (Meld)



Wird durch Vereinigung B_i erzeugt, wobei zwei B_i in Ausgangslisten, so vereinige letztere.

Zeit: $O(\log n)$

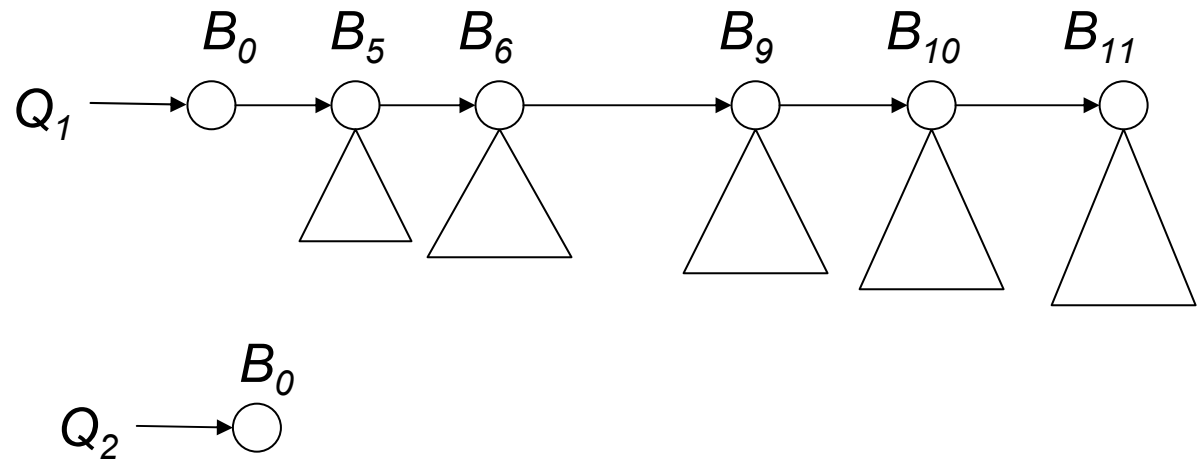
Binomial Queues: Operationen

Q.initialize:

$Q.root = null$

Q.insert(e):

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



Zeit = $O(\log n)$

Binomial Queues: Deletemin

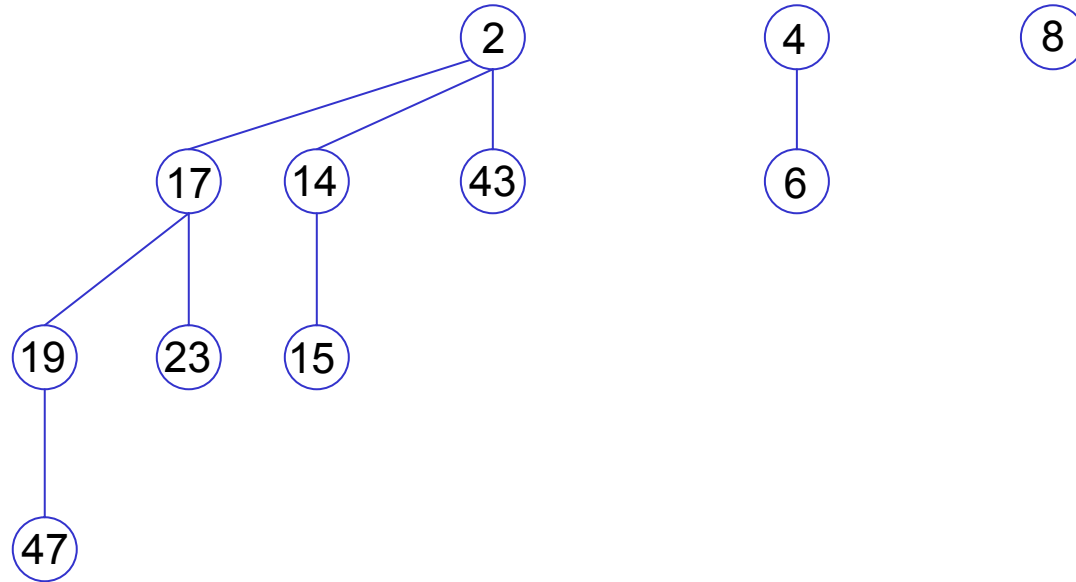
Q.deletemin():

1. Bestimme B_i mit minimalen Schlüssel in der Wurzelliste und entferne B_i aus Q (liefert Q')
2. Drehe die Reihenfolge der Söhne von B_i um, also zu $B_0, B_1, \dots, B_{i-1} \rightarrow Q''$
3. $Q'.meld(Q'')$

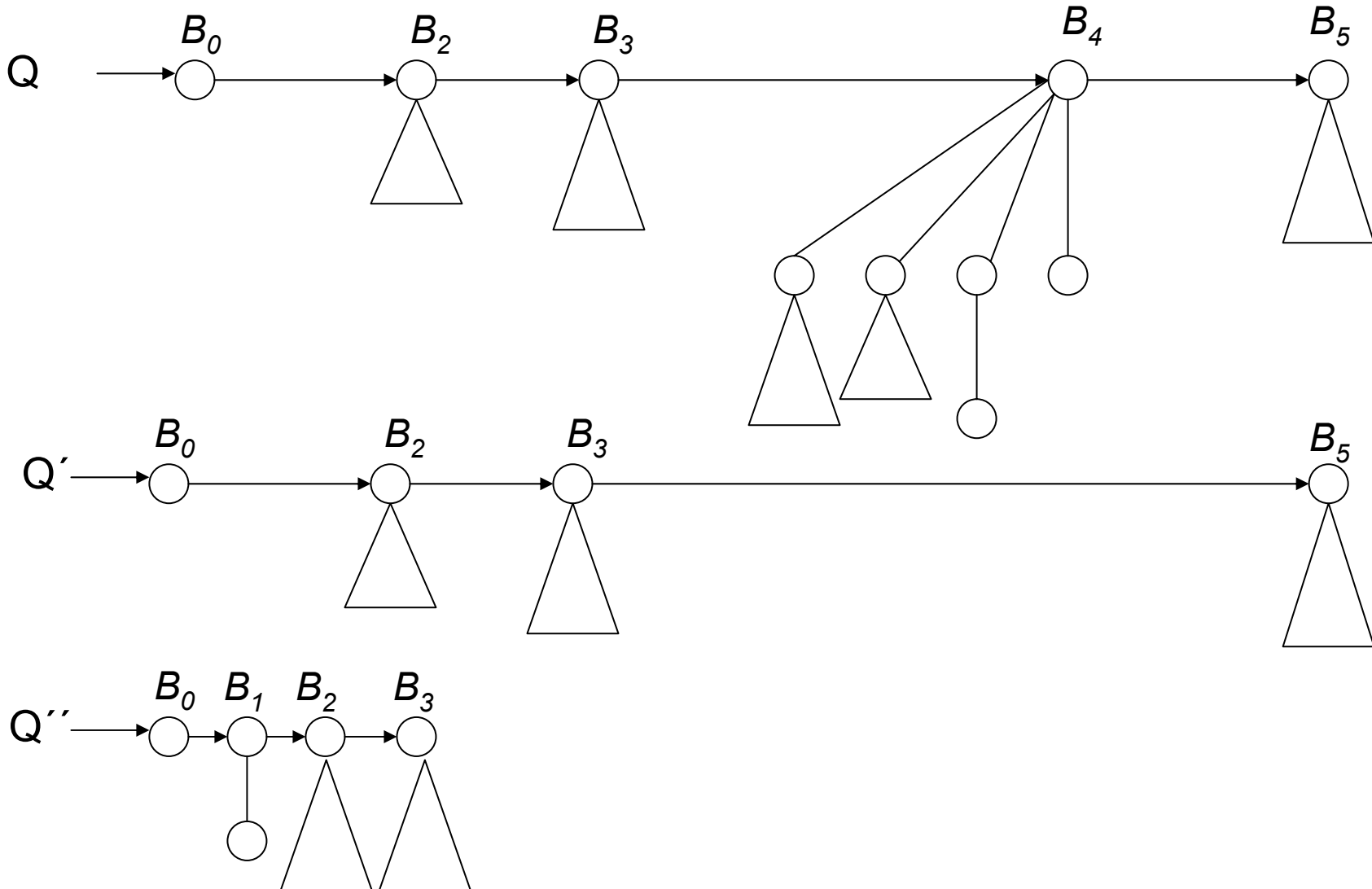
Zeit: $O(\log n)$

Binomial Queues: Deletemin Beispiel 1

Q_{11} :



Binomial Queues: Deletemin Beispiel 2

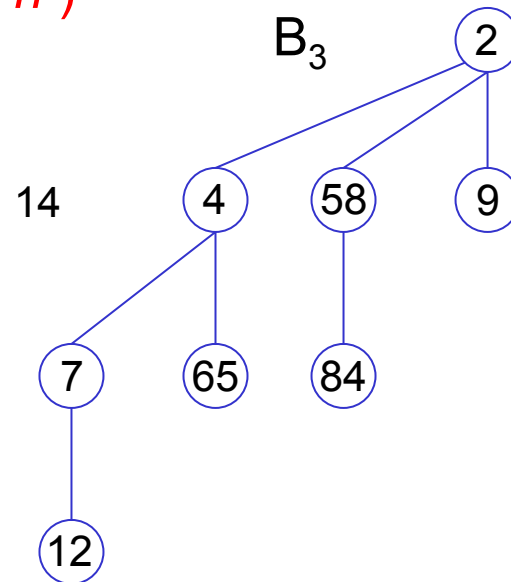


Binomialqueues: Decreasekey

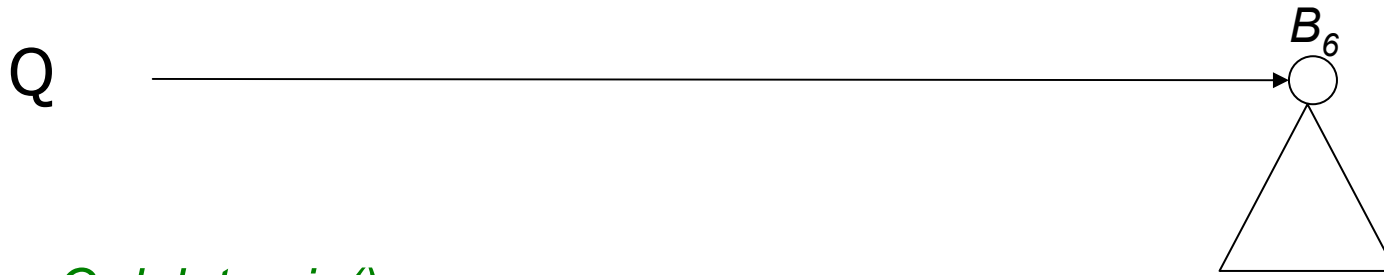
Q.decreasekey(v, k):

1. $v.entry.key := k$
2. $v.entry$ nach oben steigen lassen in dem geg. Baum, bis die Heapbedingung erfüllt ist.

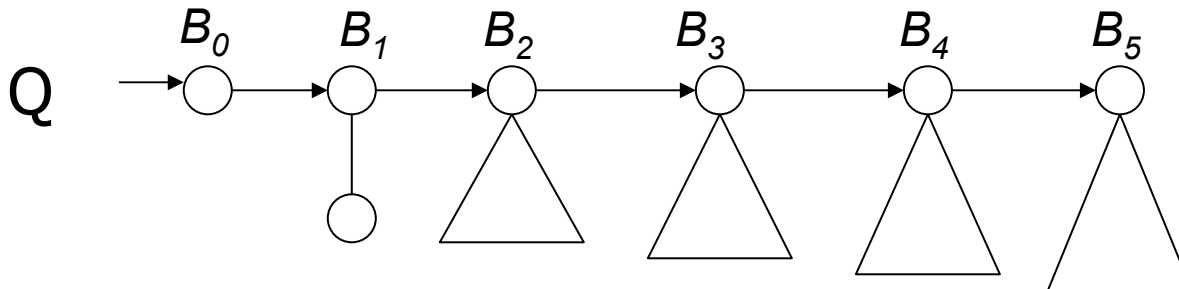
Zeit: $O(\log n)$



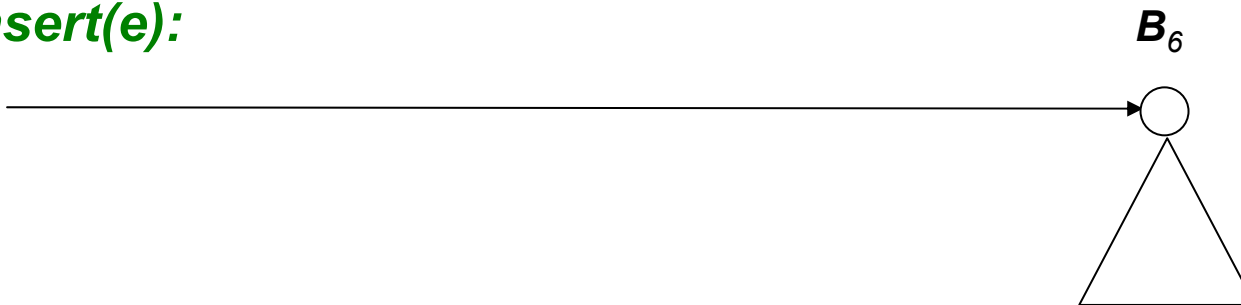
Binomial Queues: Worst Case Folge von Operationen



Q.delete_min():



Q.insert(e):



Zeit: $O(\log n)$