algorithm BINOMIALLINK(y, z)

parent$[y] = z$

brother$[y] =$ child$[z]$

child$[z] = y$

deg$[z] =$ deg$[z] + 1$

end algorithm

algorithm BinomialHeapUnion(H_1, H_2)

Verkette die Wurzellisten von H_1 und H_2 zu einer Wurzelliste H, die aufsteigend nach dem Grad der Bäume sortiert ist.

Durchlaufe H und verkette die Bäume von gleichem Grad in folgender Weise:

- Fall 1: vom Grad i gibt es genau 1 Baum: erhalte ihn.
- Fall 2: vom Grad i gibt es genau 2 Bäume: verlinke beide, wobei der Knoten mit der kleineren Wurzel die neue Wurzel wird.
- Fall 3: vom Grad i gibt es genau 3 Bäume: erhalte den ersten, verlinke die beiden anderen, wobei der Knoten mit der kleineren Wurzel die neue Wurzel wird.

end algorithm

algorithm BinomialHeapInsert(H, x)
    $H' = \text{MakeBinomialHeap}(H, x)$
    $\text{parent}[x] = \text{NIL}$
    $\text{child}[x] = \text{NIL}$
    $\text{brother}[x] = \text{NIL}$
    $\deg[x] = 0$
    $\text{head}[H'] = x$
    $H = \text{BinomialHeapUnion}(H, H')$
end algorithm

algorithm BinomialHeapExtractMin(H)
    Finde Wurzel x mit minimalem Schlüssel in der Wurzelliste von H und entferne x aus der Wurzelliste.
    $H' =$ MakeBinomialHeap(x)
    Verkette die Kinder von x in umgekehrter Reihenfolge. Setze den Vater jedes Kindes auf NIL. Setze head[H'] auf den Kopf der resultierenden Liste.
    $H =$ BinomialHeapUnion(H, H')
    return x
end algorithm

```
algorithm BinomialHeapDecreaseKey(H, x, k)
    if k > key[x] then
        Error
    end if
    key[x] = k
    y = x
    z = parent[y]
    while z ≠ NIL and key[y] < key[z] do
        swap(key[y], key[z])
        ▷ auch Satellitendaten tauschen
        y = z
        z = parent[y]
    end while
end algorithm
```

algorithm BinomialHeapDelete(H, x)
    BinomialHeapDecreaseKey(H, x, $-\infty$)
    BinomialHeapExtractMin(H)
end algorithm