algorithm FHeapInsert(H, x)
    deg[x] = 0
    parent[x] = NIL
    child[x] = NIL
    left[x] = x
    right[x] = x
    label[x] = false
    Verkette die Wurzelliste, die x enthält, mit der Wurzelliste von H
    if min[H] = NIL **or** key[x] < key[min[H]] **then**
        min[H] = x
    end if
    n[H] = n[H] + 1
end algorithm

algorithm FHeapUnion(H_1, H_2)
    H = MakeFHeap()
    min[H] = min[H_1]
    Verkette die Wurzelliste von H_2 mit der Wurzelliste von H
    if min[H_1] = NIL **or** (min[H_2] ≠ NIL **and** key[min[H_2]] < key[min[H_1]])
    then
        min[H] = min[H_2]
    end if
    n[H] = n[H_1] + n[H_2]
    Gib die Objekte H_1 und H_2 frei
    return H
end algorithm

```
algorithm FHeapExtractMin(H)
    z = min[H]
    if z ≠ NIL then
        for all Kind x von z do
            Füge x zur Wurzelliste hinzu
            parent[x] = NIL
            ▷ Markierung bleibt
        end for
        Entferne z aus der Wurzelliste
        if z = right[z] then
            min[H] = NIL
        else
            min[H] = right[z]
            FHeapConsolidate(H)
        end if
        n[H] = n[H] − 1
    end if
    return z
end algorithm
```

algorithm FHeapLink(H, y, x)
    Entferne y aus der Wurzelliste von H
    Mache y zu einem Kind von x, inkrementiere deg$[x]$
    label$[y] =$ **false**
    $\triangleright$ *Wenn y Kind von x wird, verliert es seine Markierung.*
end algorithm

```
algorithm FHeapConsolidate(H)
    for i = 0 to D(n[H]) do
        A[i] = NIL
    end for
    for all Knoten w der Wurzelliste von H do
        x = w
        d = deg[x]
        while A[d] ≠ NIL do
            y = A[d]          ▷ anderer Baum mit selbem Grad wie x
            if key[x] > key[y] then
                swap(x, y)
            end if
            FHeapLink(H, y, x)
            A[d] = NIL
            d = d + 1
        end while
        A[d] = x
    end for
    min[H] = NIL
    for i = 0 to D(n[H]) do
        if A[i] ≠ NIL then
            Füge A[i] zur Wurzelliste von H hinzu.
            if min[H] = NIL or key[A[i]] < key[min[H]] then
                min[H] = A[i]
            end if
        end if
    end for
end algorithm
```

```
algorithm FHeapDecreaseKey(H, x, k)
    if k > key[x] then
        Error
    end if
    key[x] = k
    y = parent[x]
    if y ≠ NIL and key[x] < key[y] then
        Cut(H, x, y)
        CascadingCut(H, y)
    end if
    if key[x] < key[min[H]] then
        min[H] = x
    end if
end algorithm
```

algorithm CUT(H, x, y)
    Entferne x aus der Kinderliste von y, dekrementiere deg[y].
    Füge x zur Wurzelliste von H hinzu.
    parent[x] = NIL
    label[x] = **false**
end algorithm

```
algorithm CascadingCut(H, y)
    z = parent[y]
    if z ≠ NIL then
        if label[y] = false then
            label[y] = true
        else
            Cut(H, y, z)
            CascadingCut(H, z)
        end if
    end if
end algorithm
```