Operations on Fibonacci heaps

Q.initialize(): *Q.rootlist = Q.min = null*

$O(1)$

Q.meld(Q´):

1. concatenate *Q.rootlist* and *Q´.rootlist*
2. update *Q.min* and size

$O(1)$

Q.insert(e):

1. generate a new node with element *e* → *Q´*
2. *Q.meld(Q´)*

$O(1)$

Q.min():

return *Q.min.key*

$O(1)$

Fibonacci heaps: ‘deletemin’

Q.deletemin()

/*Delete the node with minimum key from *Q* and return its element.*/

```
1   m = Q.min()
2   if Q.size() > 0
3         then remove Q.min() from Q.rootlist
4                   add Q.min.childlist to Q.rootlist
5         Q.consolidate()
        /* Repeatedly meld nodes in the root list having the same
           degree. Then determine the element with minimum key. */
6   return m
```

Fibonacci heaps: maximum degree of a node

rank(v) = degree of node *v* in *Q*

rank(Q) = maximum degree of any node in *Q*

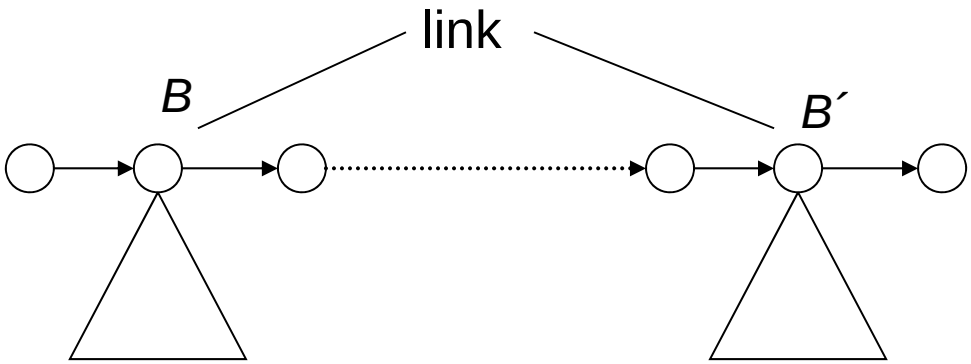
Assumption:

$$rank(Q) \leq 2 \log n,$$

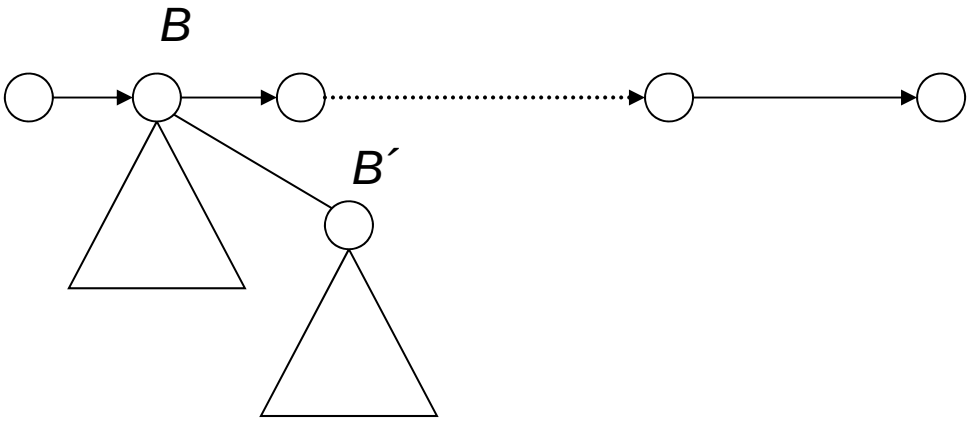
if $Q.size = n$.

Fibonacci heaps: operation ‘link’

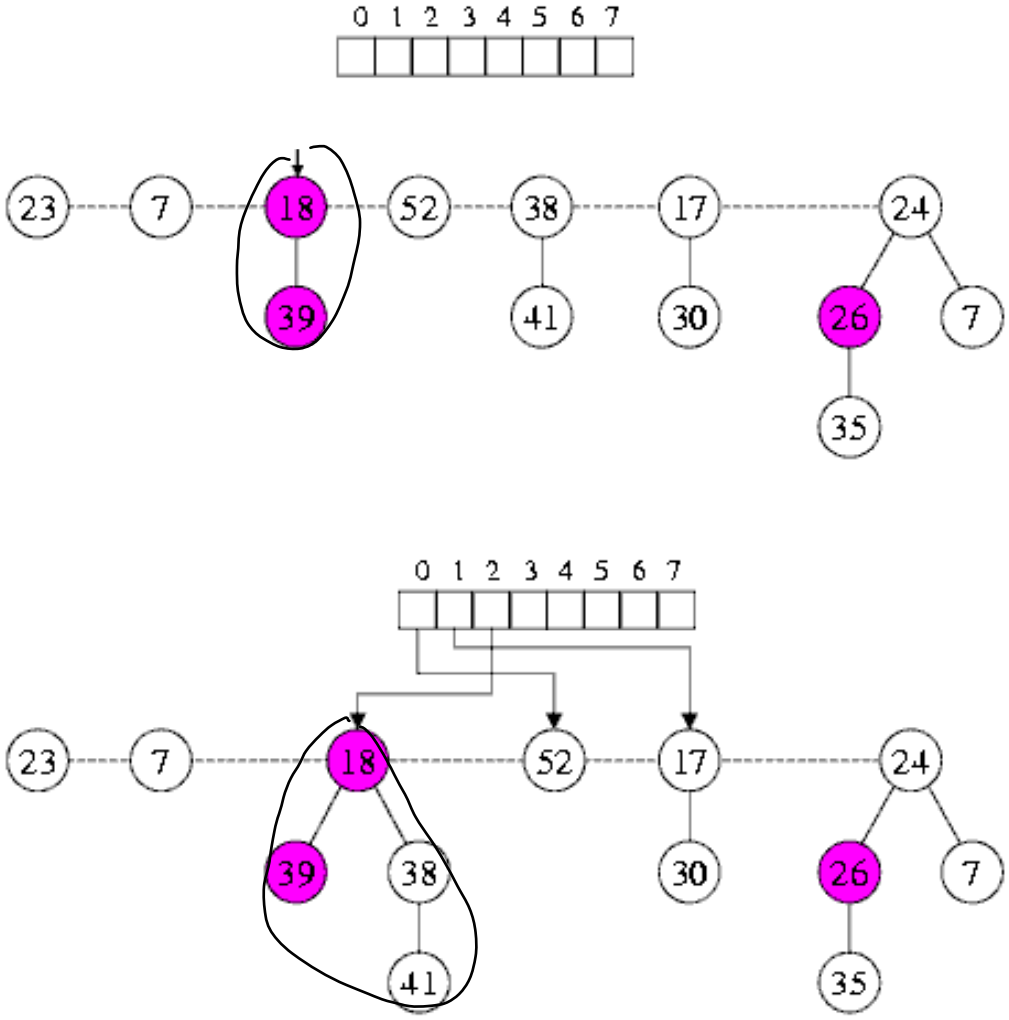
rank(B) = degree of the root of *B*
Heap-ordered trees *B,B´* with *rank(B) = rank(B´)*



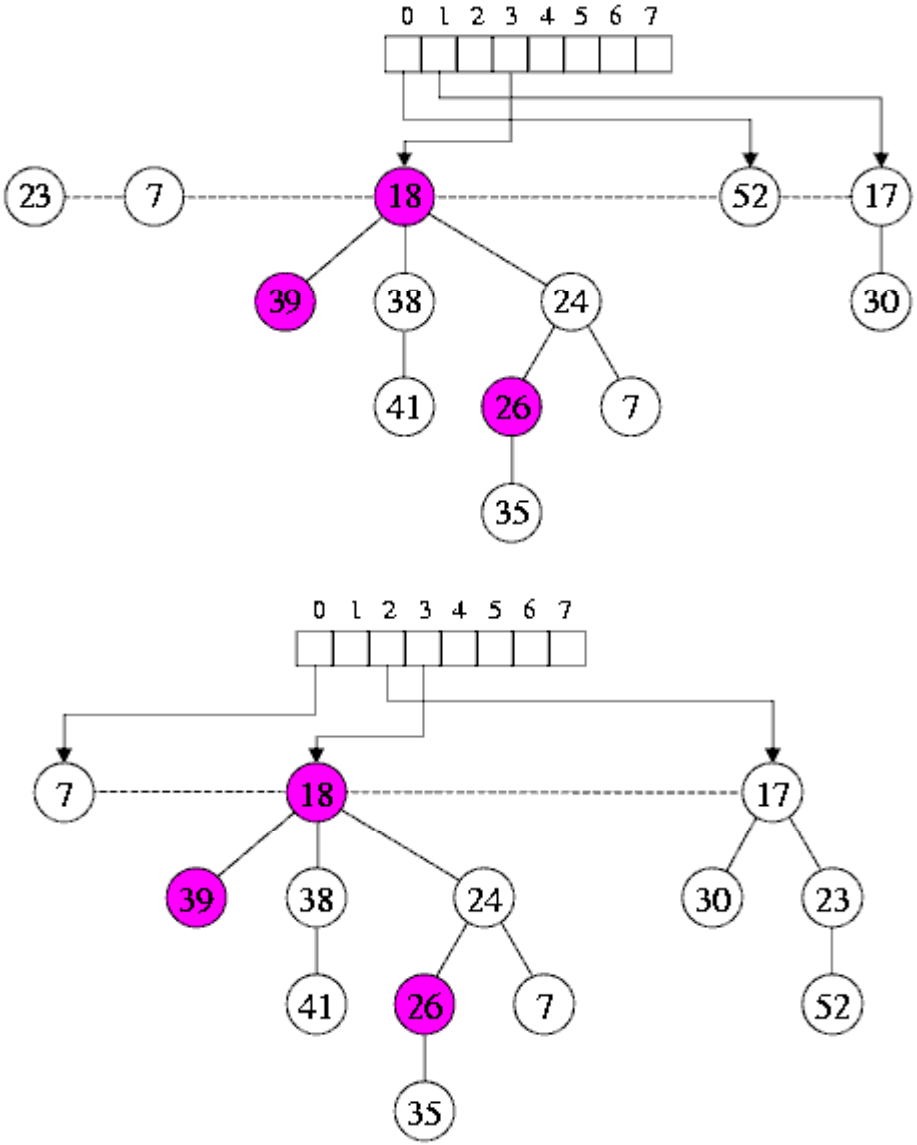
1. *rank(B) = rank(B)* + 1
2. *B´.mark = false*



Consolidation of the root list



Consolidation of the root list



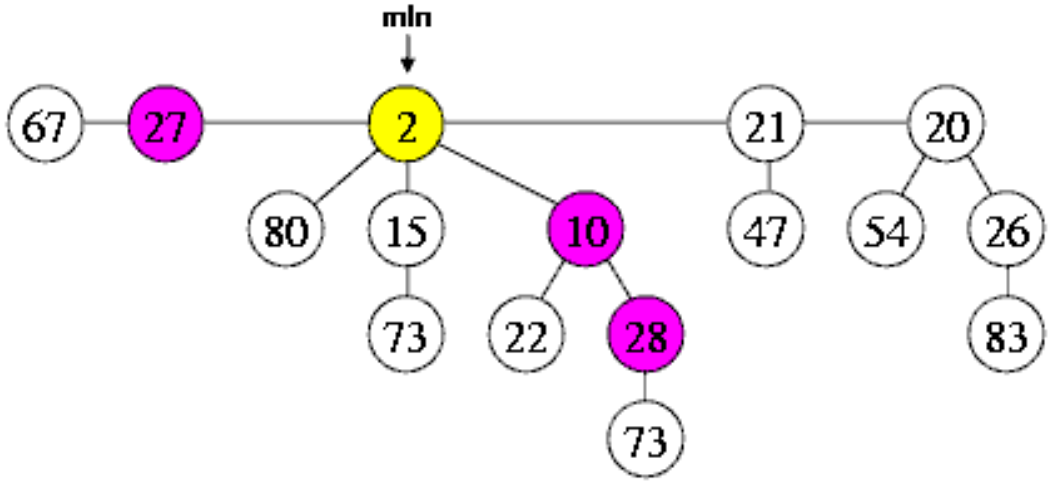
Fibonacci heaps: ‘deletemin’

Find roots having the same rank:
Array *A*:

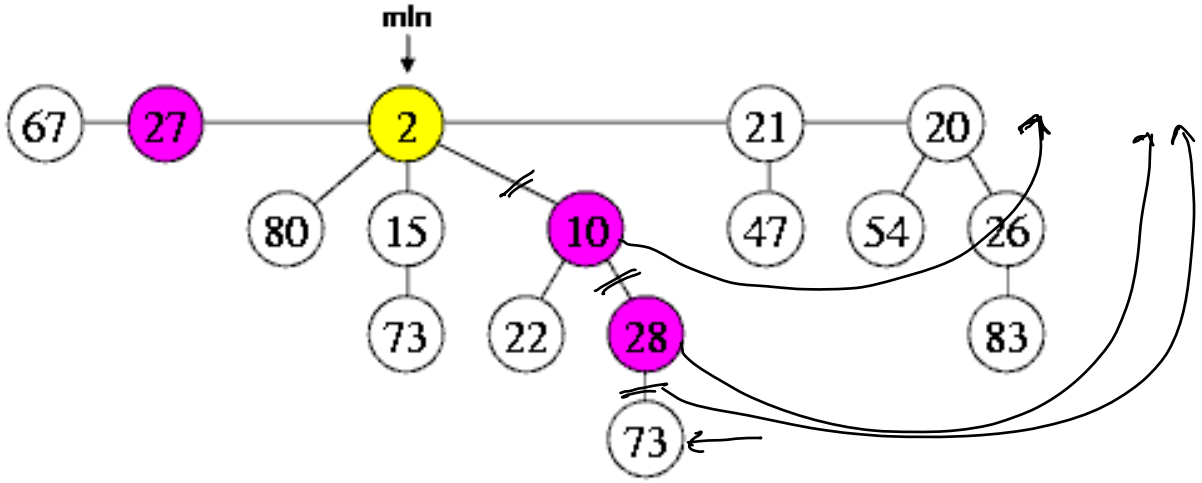
Q.consolidate()

```
1   A = array of length 2 log n pointing to Fibonacci heap nodes
2   for i = 0 to 2 log n do A[i] = null
3   while Q.rootlist ≠ ∅ do
4         B = Q.delete-first()
5         while A[rank(B)] is not null do
6               B´ = A[rank(B)];   A[rank(B)] = null;   B = link(B,B´)
7         end while
8         A[rank(B)] = B
9    end while
10  determine Q.min
```

Fibonacci heap: Example



Fibonacci heap: Example



Fibonacci heaps: ‘decreasekey’

```
Q.decreasekey(v,k)
 1   if k > v.key then return
 2   v.key = k
 3   update Q.min
 4   if v ∈ Q.rootlist or k ≥ v.parent.key then return
 5   do /* cascading cuts */
 6        parent = v.parent
 7        Q.cut(v)
 8        v = parent
 9   while v.mark and v ∉ Q.rootlist
10    if v ∉ Q.rootlist then v.mark = true
```

Fibonacci heaps: ‘cut’

Q.cut(v)

```
1  if v ∉ Q.rootlist
2    then  /* cut the link between v and its parent */
3          rank (v.parent) = rank (v.parent) – 1
4          remove v  from v.parent.childlist
5          v.parent = null
6          add v to Q.rootlist
```

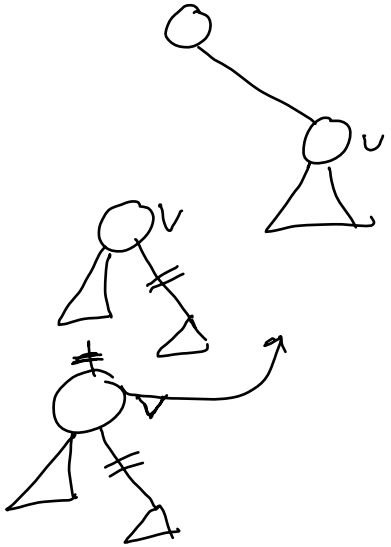
Fibonacci heaps: marks

History of a node:

(1) *v* is being linked to a node → *v.mark = false*

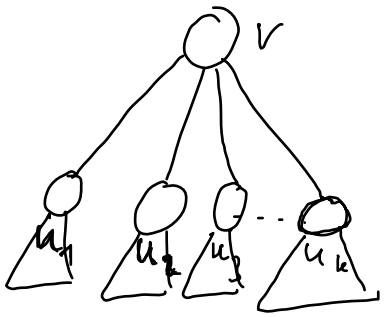
(2) a child of *v* is cut → *v.mark = true*

(3) a second child of *v* is cut → cut *v*



The boolean value *mark* indicates whether node *v* has lost a child since the last time *v* was made the child of another node.

Rank of the children of a node



Lemma

Let v be a node in a Fibonacci-Heap Q. Let $u_1,\ldots,u_k$ denote the children of v in the order in which they were linked to v. Then:

$$rank(u_i) \geq i - 2.$$

Proof:

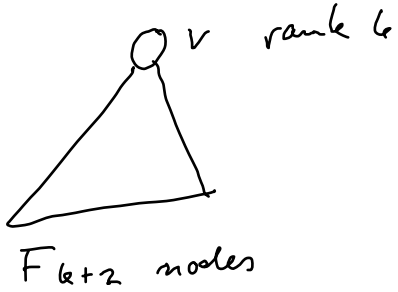
At the time when u_i was linked to v:

children of v ($rank(v)$): $\geq i - 1$

children of u_i ($rank(u_i)$): $\geq i - 1$

→ # children u_i may have lost: 1

Maximum rank of a node



Theorem

Let *v* be a node in a Fibonacci heap *Q*, and let *rank(v) = k* . Then *v* is the root of a subtree that has at least F_{k+2} nodes.

$F_0 = 0, \quad F_1 = 1, \quad F_k = F_{k-1} + F_{k-2} \quad k \geq 2, \quad F_{k+2} \geq \phi^k \quad \phi = \frac{1+\sqrt{5}}{2} \approx 1.618$

The number of descendants of a node grows exponentially in the number of children.

Implication:

The maximum rank *k* of any node *v* in a Fibonacci heap *Q* with *n* nodes satisfies: v root subtree with at least

$\phi^k \leq n$ nodes

$\Rightarrow k \leq \log_\phi n = \frac{\log_2 n}{\log_2 \phi} \approx 1.441 \cdot \log_2 n \leq 2 \cdot \log_2 n.$

$\boxed{k \leq 2 \cdot \log_2 n}$

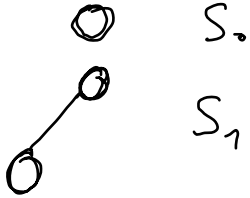
Maximum rank of a node

Proof

S_k = minimum possible size of a subtree whose root has rank k

$S_0 = 1$

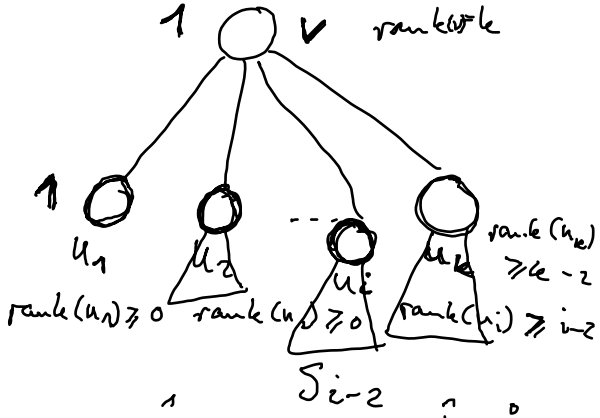
$S_1 = 2$



(1) There is:

$$S_k \geq 2 + \sum_{i=0}^{k-2} S_i \quad \text{for } k \geq 2 \qquad (1)$$

Lemma:



(2) Fibonacci numbers:

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i \qquad (2)$$

$$= 1 + F_0 + F_1 + \ldots + F_k$$

(1) + (2) + induction → $S_k \geq F_{k+2}$

$k=0$: $S_0 = 1 \geq F_2$

$k=1$: $S_1 = 2 \geq F_3$

$$S_k \overset{(1)}{\geq} 2 + \sum_{i=0}^{k-2} S_i \geq 2 + \sum_{i=0}^{k-2} F_{i+2} = 1 + F_0 + F_1 + \sum_{i=2}^{k} F_i \overset{(2)}{=} F_{k+2}$$

$k=0$: $F_2 = 1 + F_0 = F_1 + F_0$

$k=1$: $F_3 = 1 + F_0 + F_1 = 2$

k: $F_{k+2} = F_{k+1} + F_k$

$= F_{k+1} + 1 + F_0 + F_1 + \ldots + F_{k-2}$

$= F_k + F_{k-1} + 1 + F_0 + \ldots + F_{k-2}$

$= 1 + F_0 + \ldots + F_{k-2} + F_{k-1} + F_k$

Analysis of Fibonacci heaps

Potential method to analyze Fibonacci heap operations.

Potential Φ_Q of Fibonacci heap Q:

$$\Phi_Q = r_Q + 2 \cdot m_Q$$

where

r_Q = number of nodes in $Q.rootlist$

m_Q= number of all marked nodes in Q,
that are not in the root list.

Amortized analysis

Amortized cost a_i of the i-th operation:

actual cost

change in potential

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$= t_i + (r_i - r_{i-1}) + 2(m_i - m_{i-1})$$

Analysis of ‘insert’

insert

$t_i = 1$

generate new node, insert into roothist

$r_i - r_{i-1} = 1$

nodes in roothist increases by 1.

$m_i - m_{i-1} = 0$

$a_i = 1 + 1 + 0 = O(1)$

$t_i + (r_i - r_{i-1}) + 2 \cdot (m_i - m_{i-1})$

Analysis of ‘deletemin’

deletemin:

in consolidate: we traverse the rootlist and each node gets linked as a child to some other node at most once. We traverse the new rootlist again for finding the new min.

$r_i \leq 2 \log n$

$t_i = r_{i-1} + 2 \log n$

$r_i - r_{i-1} \leq 2 \log n - r_{i-1}$

$m_i - m_{i-1} \leq 0$

$a_i \leq \overbrace{r_{i-1} + 2 \log n}^{t_i} + \overbrace{2 \log n - r_{i-1}}^{r_i - r_{i-1} + 2 \cdot (m_i - m_{i-1})} + 0$

$= O(\log n)$

Analysis of ‘decreasekey’

decreasekey:

c : # marked nodes on the path from the node where decrease key happens to root

$t_i = c + 2$

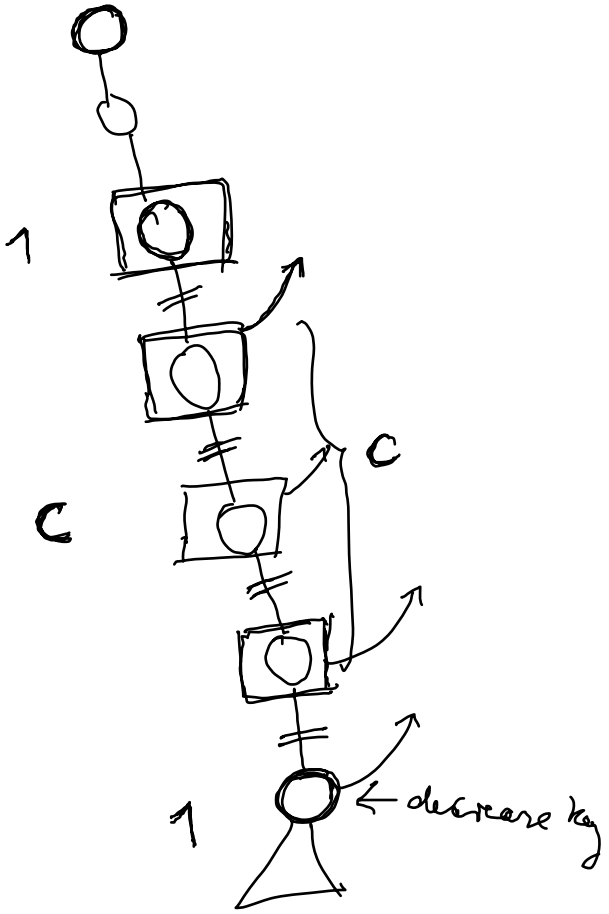
$r_i - r_{i-1} = c + 1$

$m_i - m_{i-1} \leq -c + 1$

$$a_i \leq \underbrace{c + 2}_{t_i} + \underbrace{c + 1}_{r_i - r_{i-1}} + \underbrace{2 \cdot (-c + 1)}_{m_i - m_{i-1}}$$

$= 2 \cdot c + 3 + 2 \cdot (-c + 1) = 5$

$= O(1)$



Priority queues: comparison

	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≤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