# COS 423 Lecture 7
# Rank-Pairing Heaps

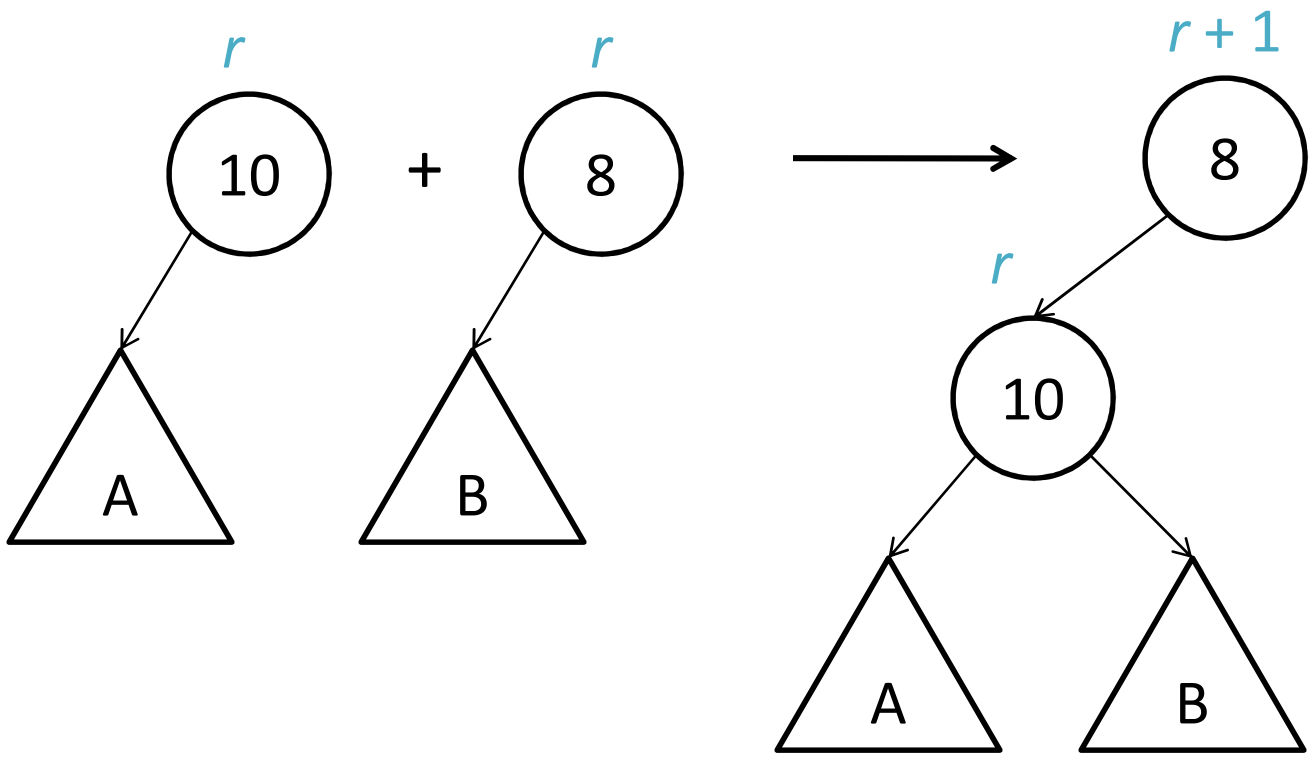
Inefficiency in pairing heaps:

- Links during *inserts*, *melds*, *decrease-keys* can combine two trees of very different sizes, increasing the potential by $\Theta(\lg n)$.

To avoid such bad links, we use *ranks*:

- Each node has a non-negative integer rank
- Missing nodes have rank –1
- Rank of root = 1 + rank of left child
- *rank of tree* = rank of root

Store ranks with nodes. Link only roots of equal rank. Rank of winning root increases by one:



A heap is a set of half trees, not just one half tree (can't link half trees of different ranks)

Representation of a heap: a circular singly-linked list of roots of half trees, with the root of minimum key (the *min-root*) first

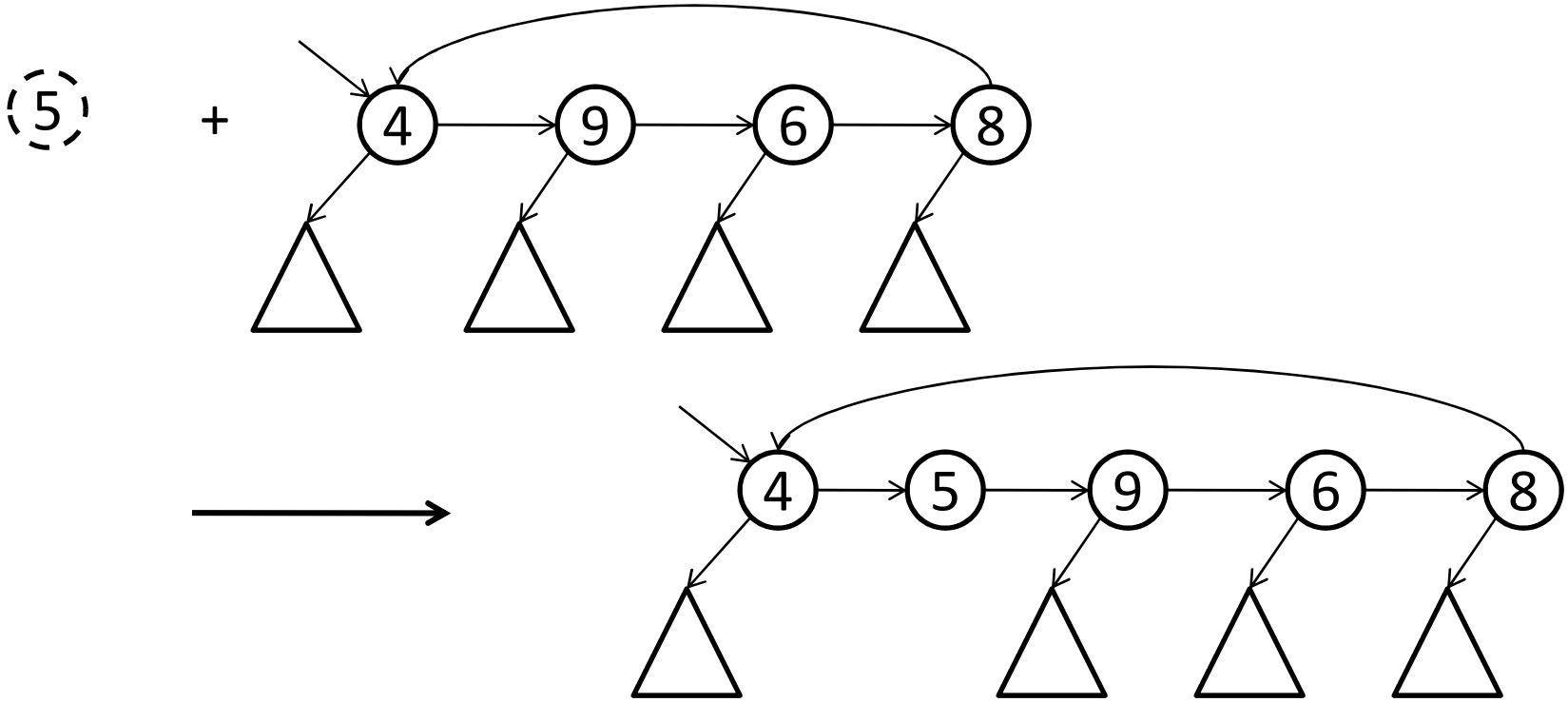
Circular linking → catenation takes O(1) time

Node ranks depend on how operations are done

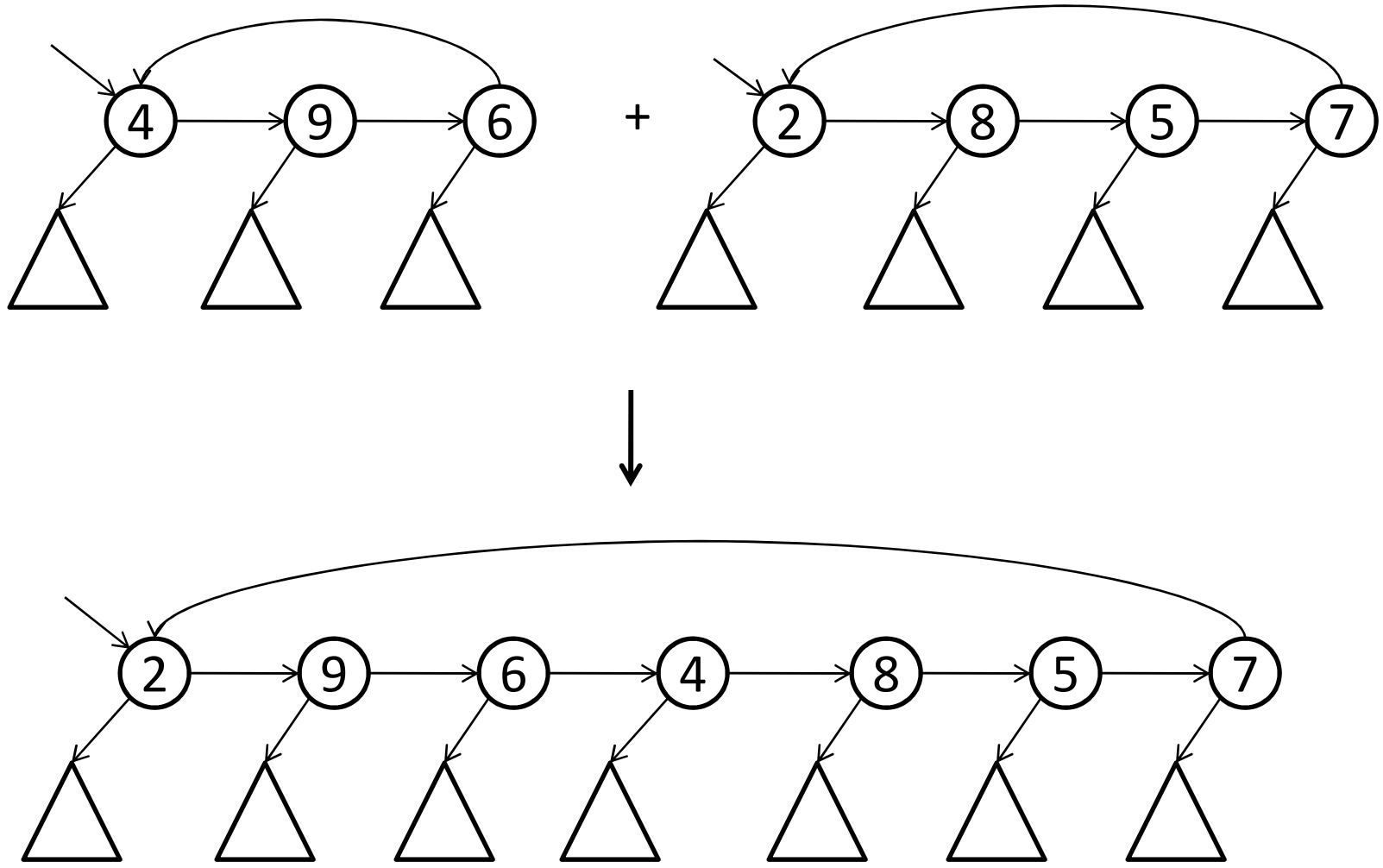
*find-min*: return item in min-root

*make-heap*: return a new, empty list

*insert*: create new one-node half tree of rank 0, insert in list, update min-root



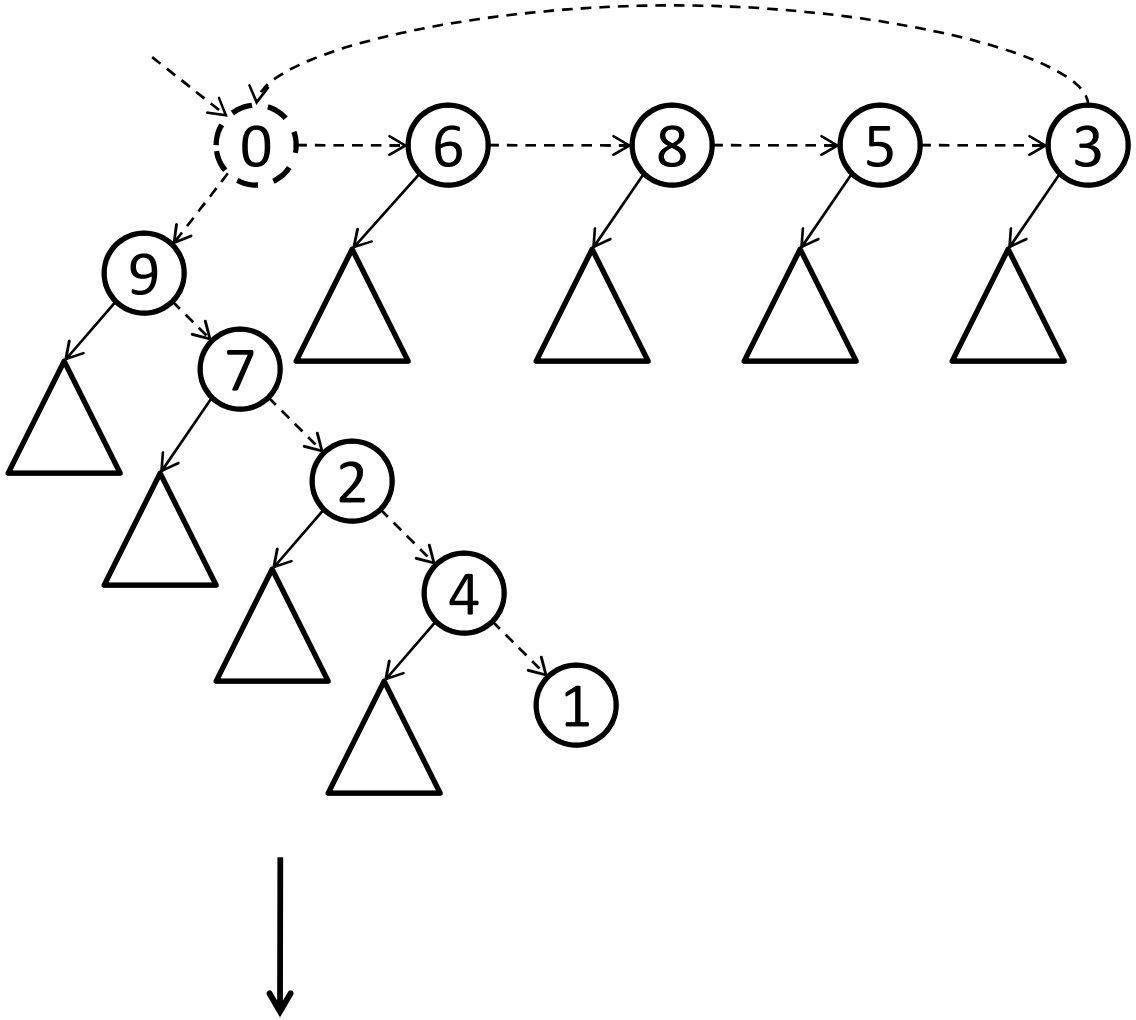
*meld*: catenate lists, update min-root



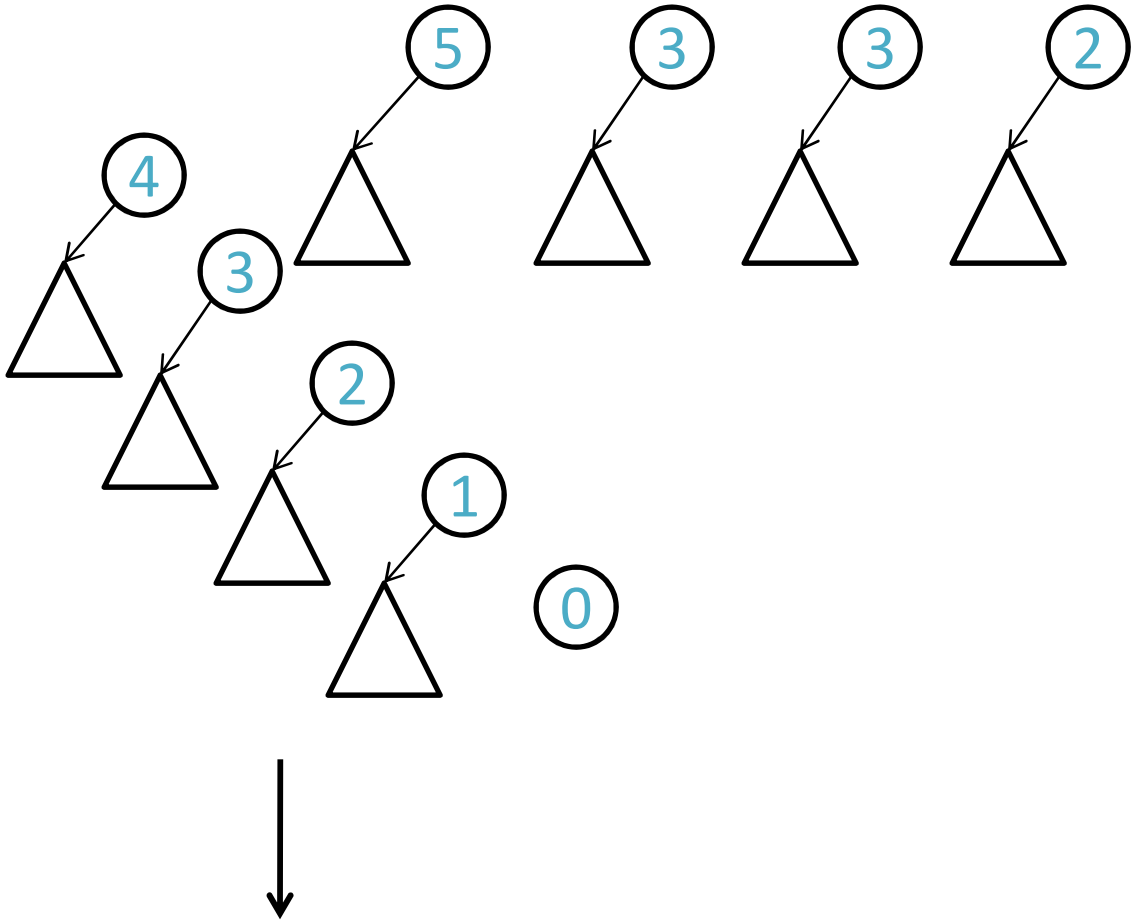
*delete-min*: Delete min-root. Cut edges along right path down from new root to give new half-trees. Link roots of equal rank until no two roots have equal rank. Form list of remaining half trees.

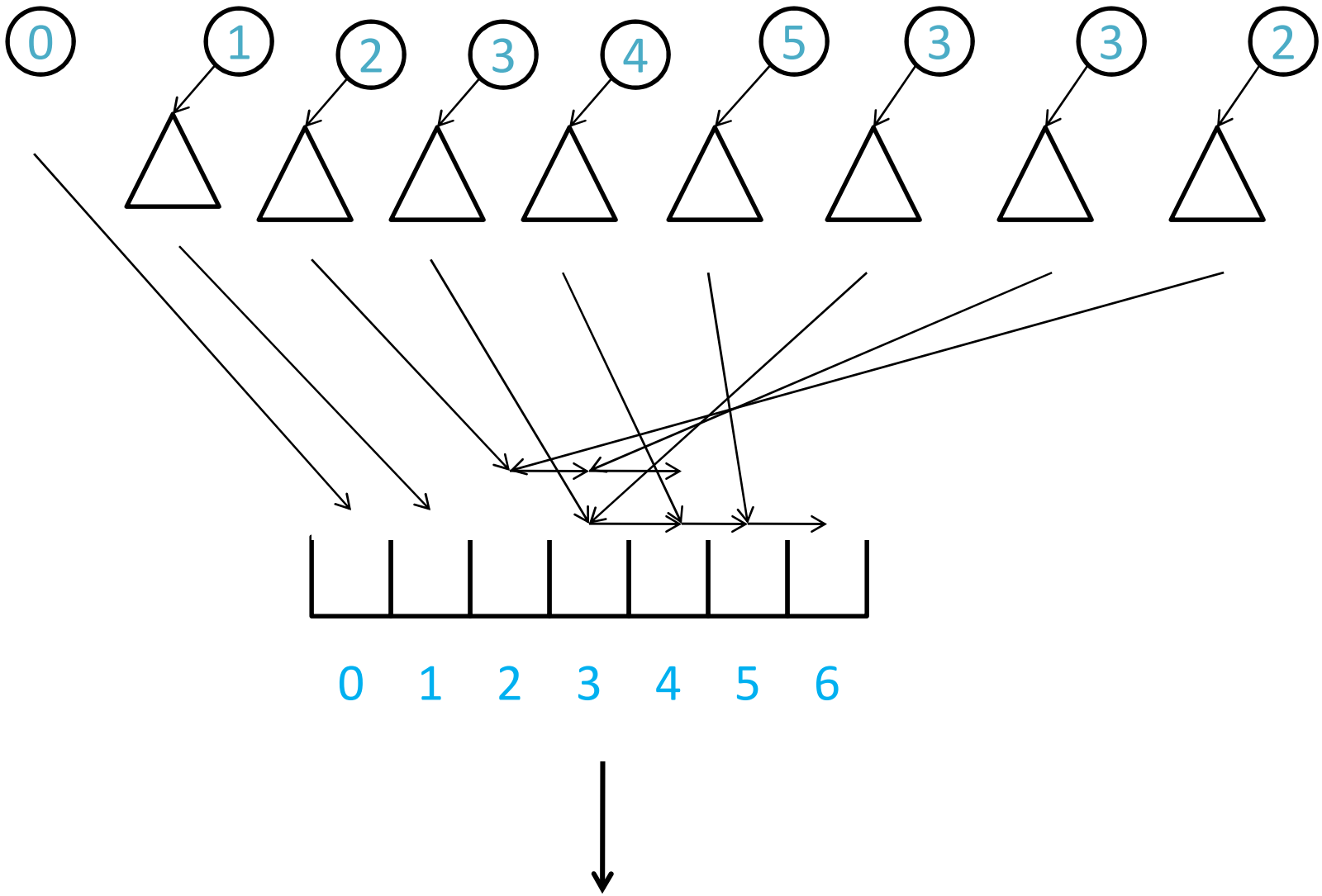
To do links, use array of buckets, one per rank. Add each tree to its bucket, link with tree in bucket if there is one, add tree formed by link to next bucket.

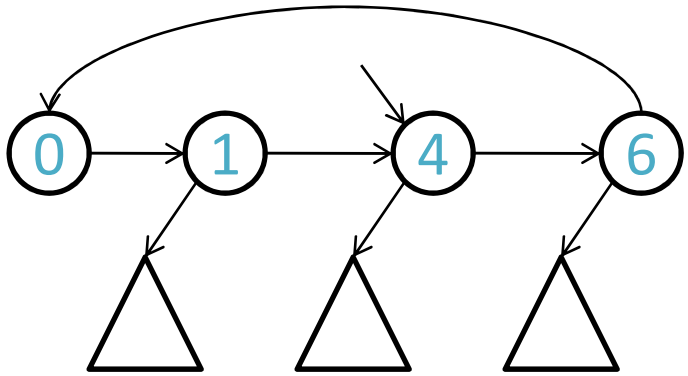
*delete-min*: numbers are keys



*delete-min*: numbers are ranks




0
1
2
3
4
5
3
3
2
0 1 2 3 4 5 6




Each link takes O(1) time, forming final tree takes O(max rank) time → *delete-min* takes O(max rank + #links) time

Amortize links: let $\Phi$ = # half trees

One unit of time per link , $\Delta\Phi = -1$

→ link takes 0 amortized time

*make heap*, *find-min*, *meld* do not change Φ,

*insert* increases Φ by 1 → each takes O(1) amortized time

*delete-min* takes O(#new half trees + max rank) amortized time

**How many new half trees?**

**What is max rank**?

All rank differences are 1, all leaves have rank 0

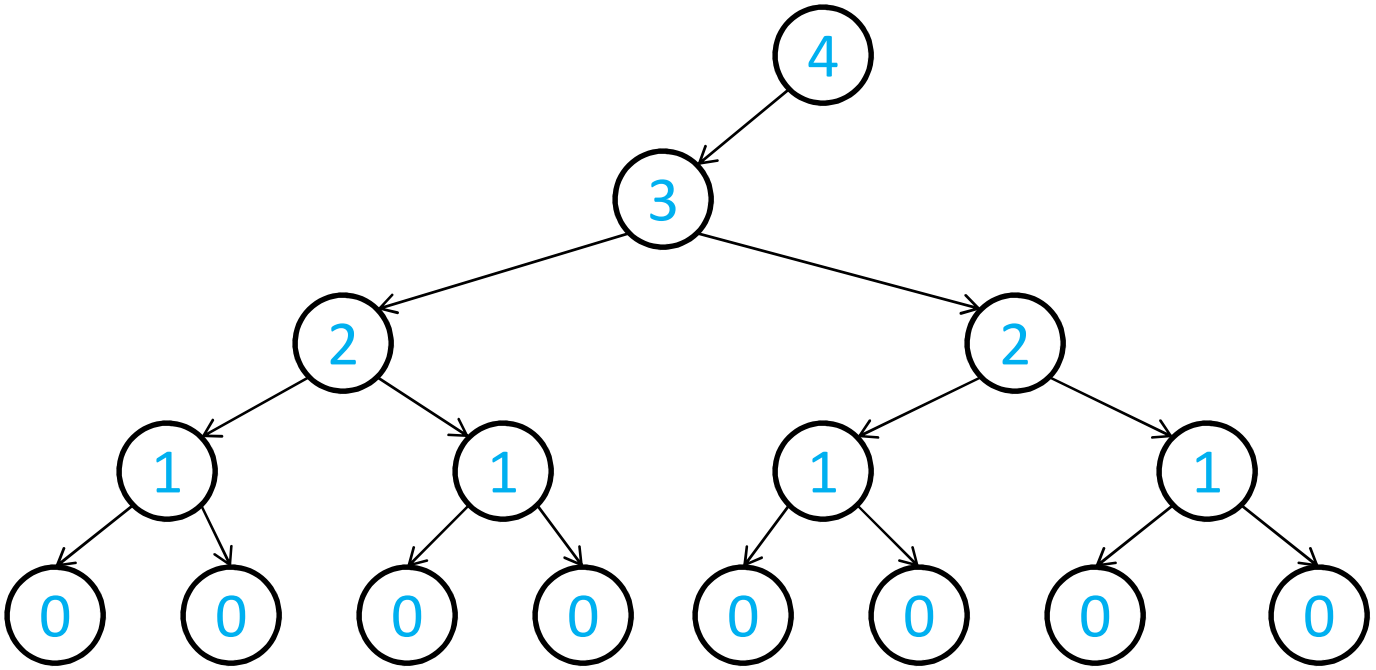
→ all half trees are perfect: a half tree of rank $r$ contains exactly $2^r$ nodes

→ #new half trees, max rank ≤ lg$n$

→ *delete-min* takes O(lg$n$) amortized time

Data structure: ***lazy binomial queue***.

# A perfect half tree of rank 4



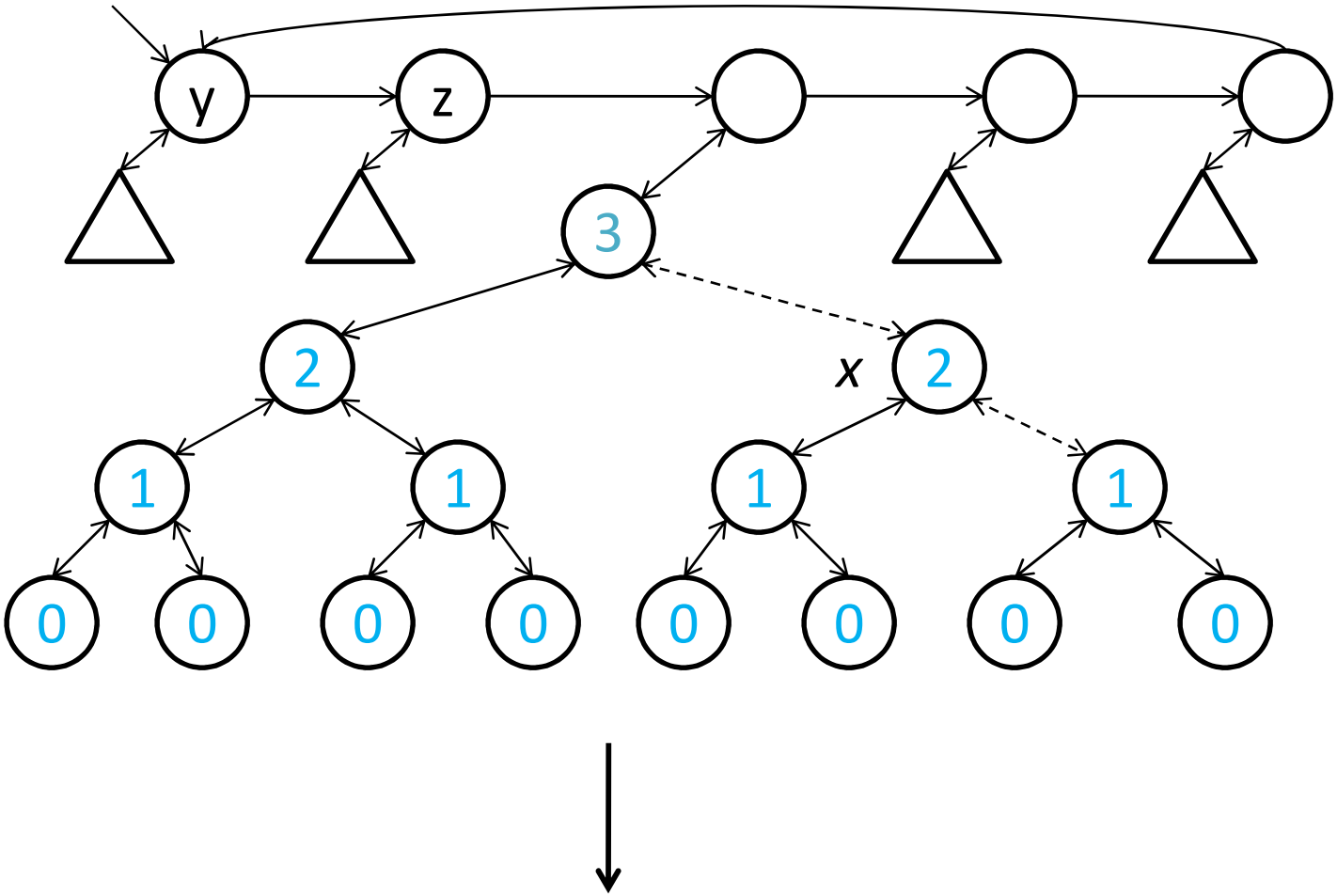
# What about *decrease-key*?

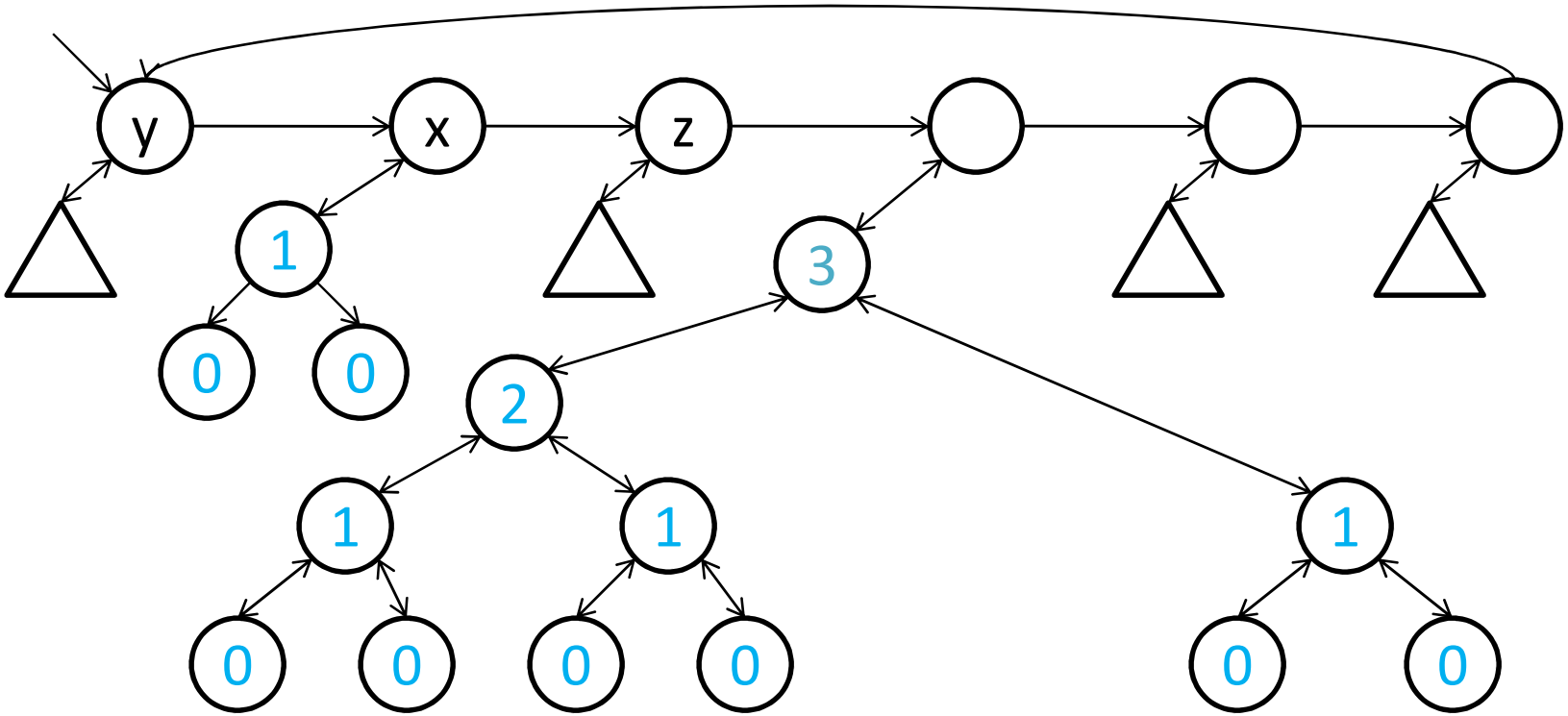
Add parent pointers to data structure

*decrease key*(*x*, *k*, *H*): Remove *x* and its left subtree (becomes a new half tree). Replace *x* by its right child. Change key of *x* to *k*. Add *x* to the list of half tree roots. Update the min-root.

But: rank differences are no longer all 1, half trees are no longer perfect, max rank can grow beyond O(lg*n*)

*decrease-key*(*x*, *k*, *H*): numbers are ranks




y
x
z
1
3
0
0
2
1
1
1
0
0
0
0
0
0

Solution: maintain an invariant on rank differences. Allowed node types are 1,1 (perfect); 1,2; and 0,*j* for any $j > 1$.

At end of *decrease-key* of *x*, set $r(x) = r(left(x)) + 1$; start from old parent *y* of *x* and walk up path toward root, decreasing ranks to restore the invariant:

**while** $p(y) \neq$ null **and** $r(y)$ too big **do**

{let $y$ be $i,j$ with $i \leq j$;

**if** $j - i > 1$ **then** $r(y) \leftarrow r(y) - i$

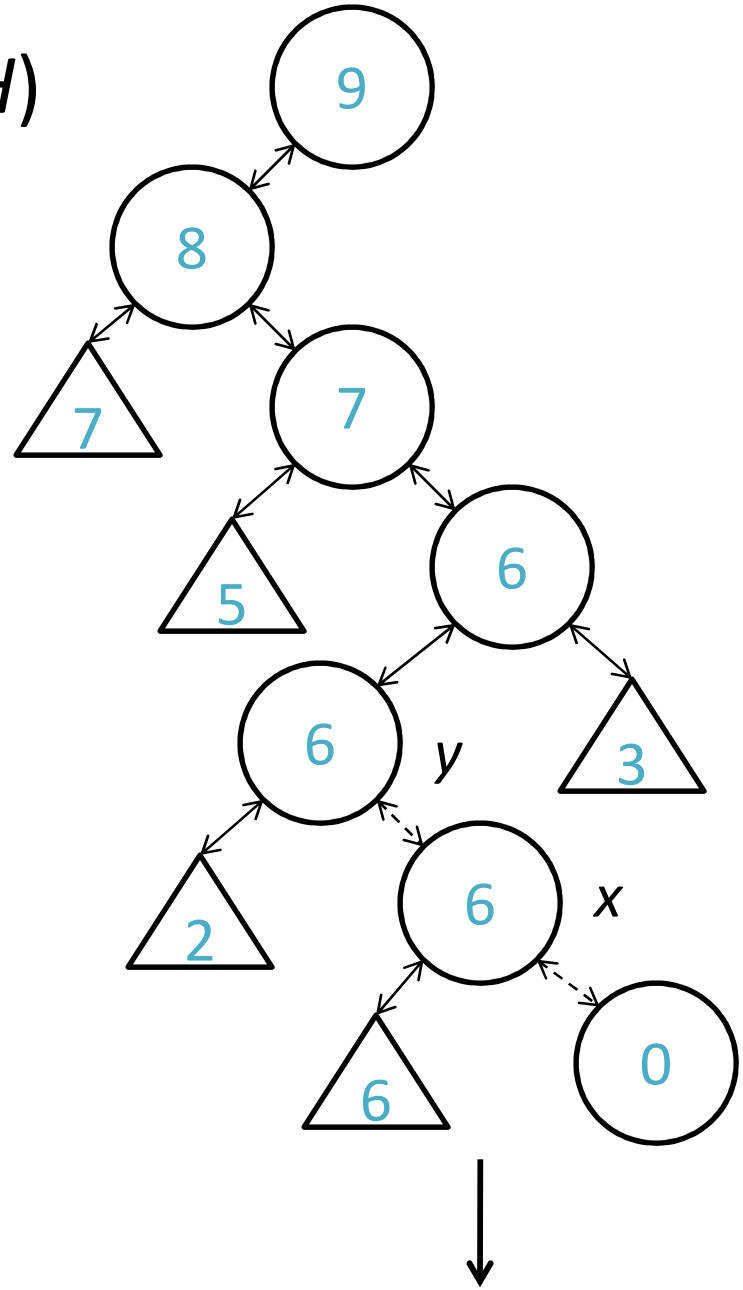
**else** $r(y) \leftarrow r(y) - i + 1$;

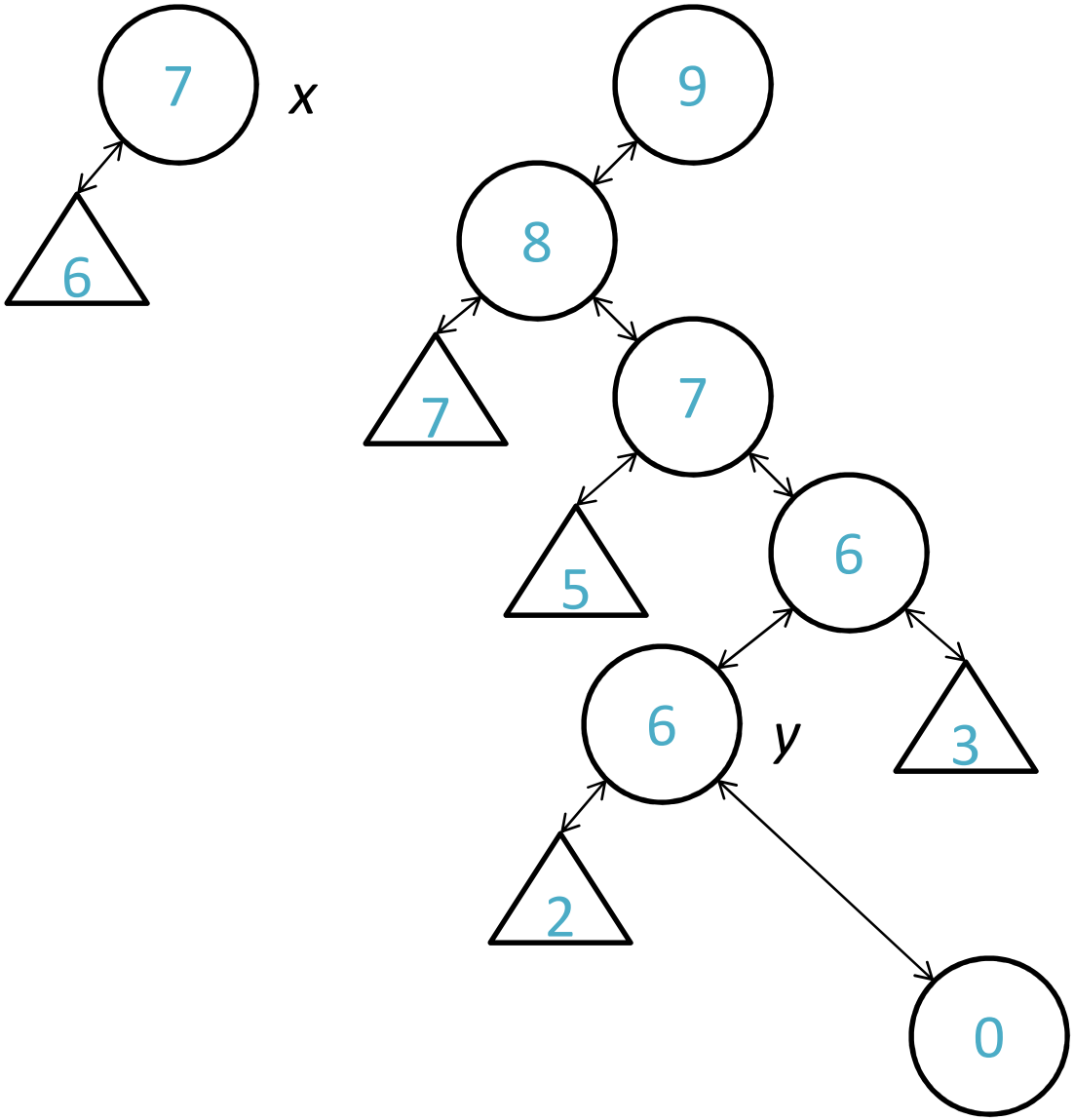
$y \leftarrow p(y)$};

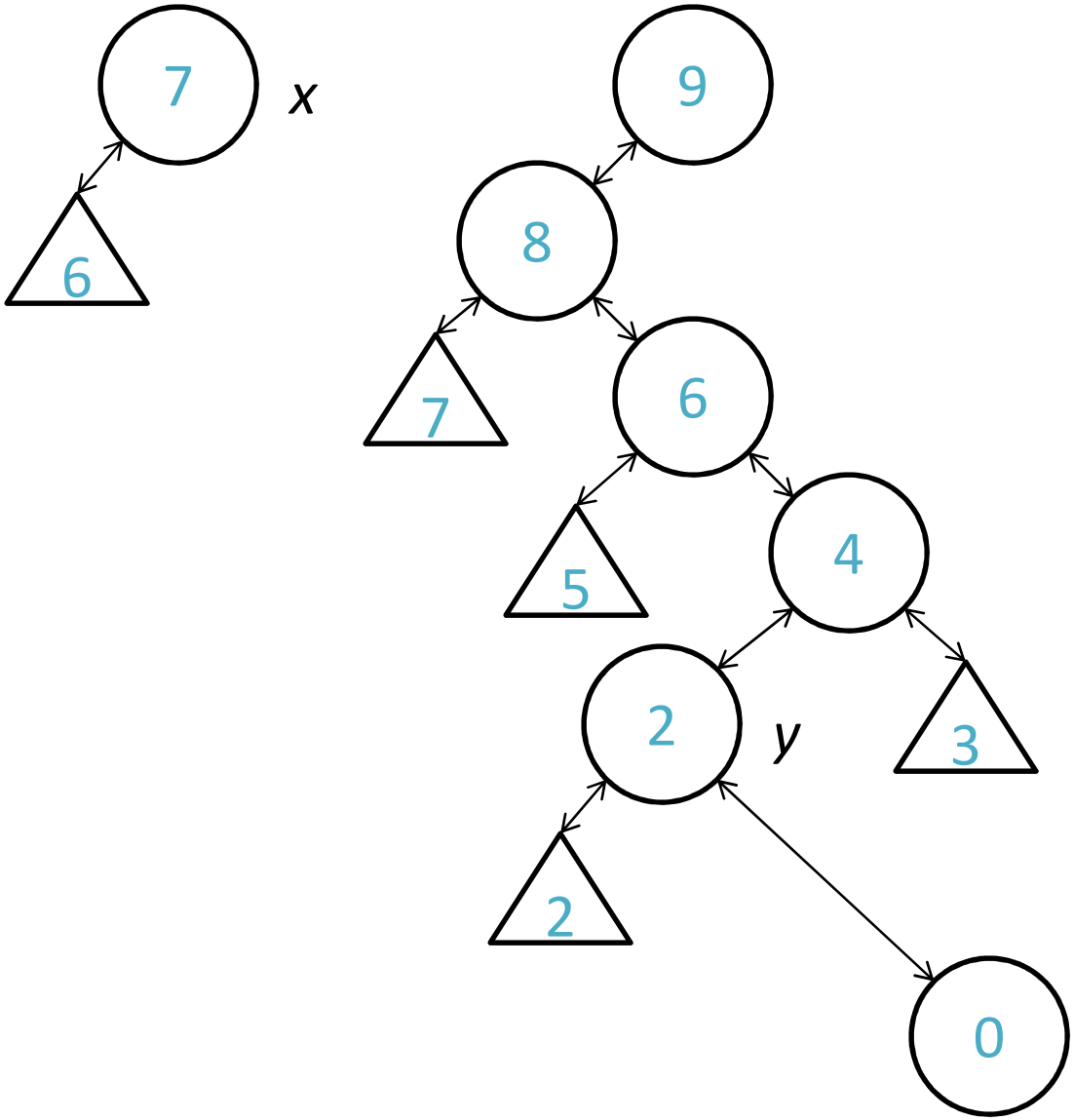
**if** $p(y) =$ null **then** $r(y) = r(left(y)) + 1$

In successive steps, $r(y)$ decreases by a non-increasing amount

*decrease-key*(*x*, *k*, *H*)




7
x
6
9
8
7
7
5
6
6
y
3
2
0

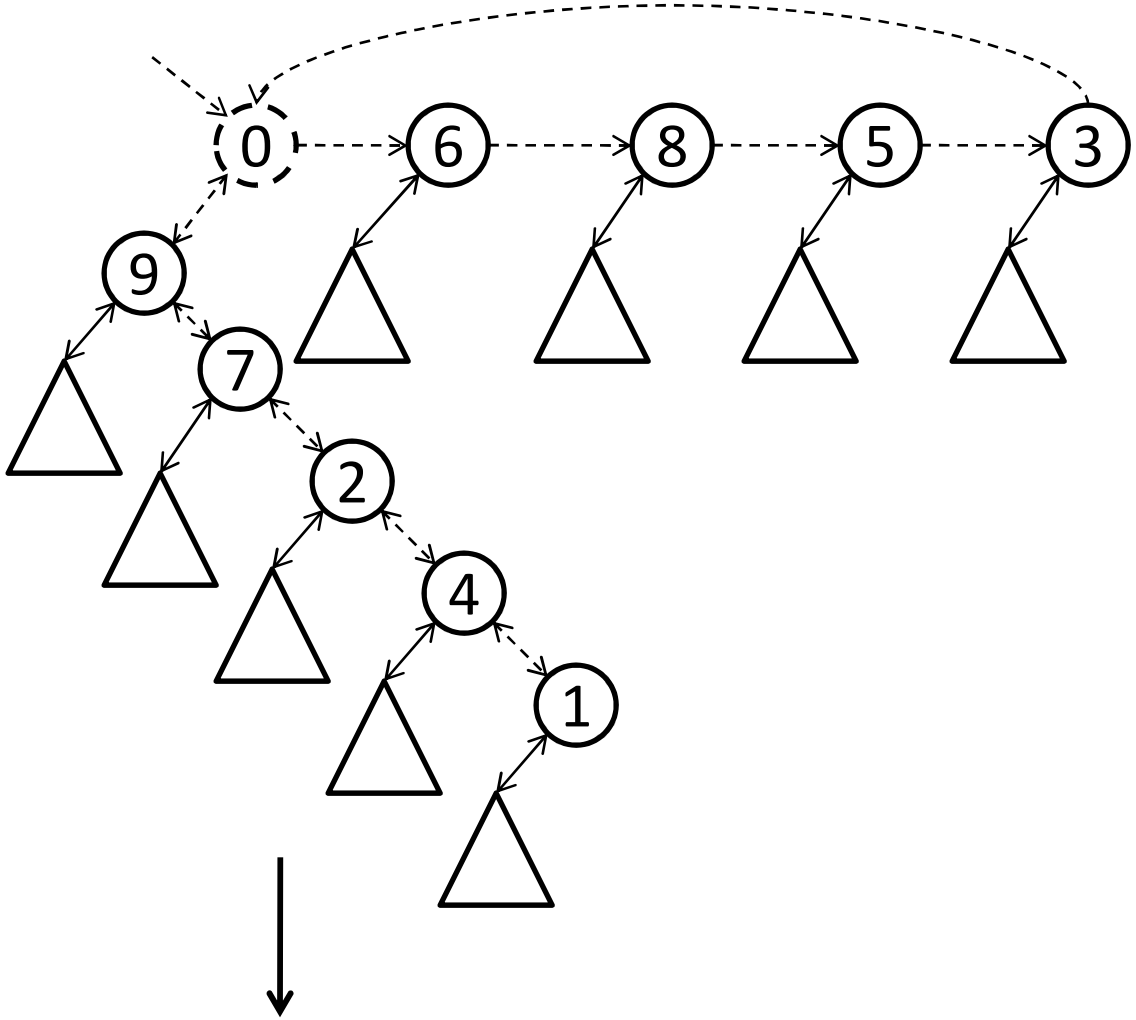

7
x
6
9
8
7
6
5
4
2
y
3
2
0

*delete*(*x*, *H*): *decrease-key*(*x*, –∞, *H*);

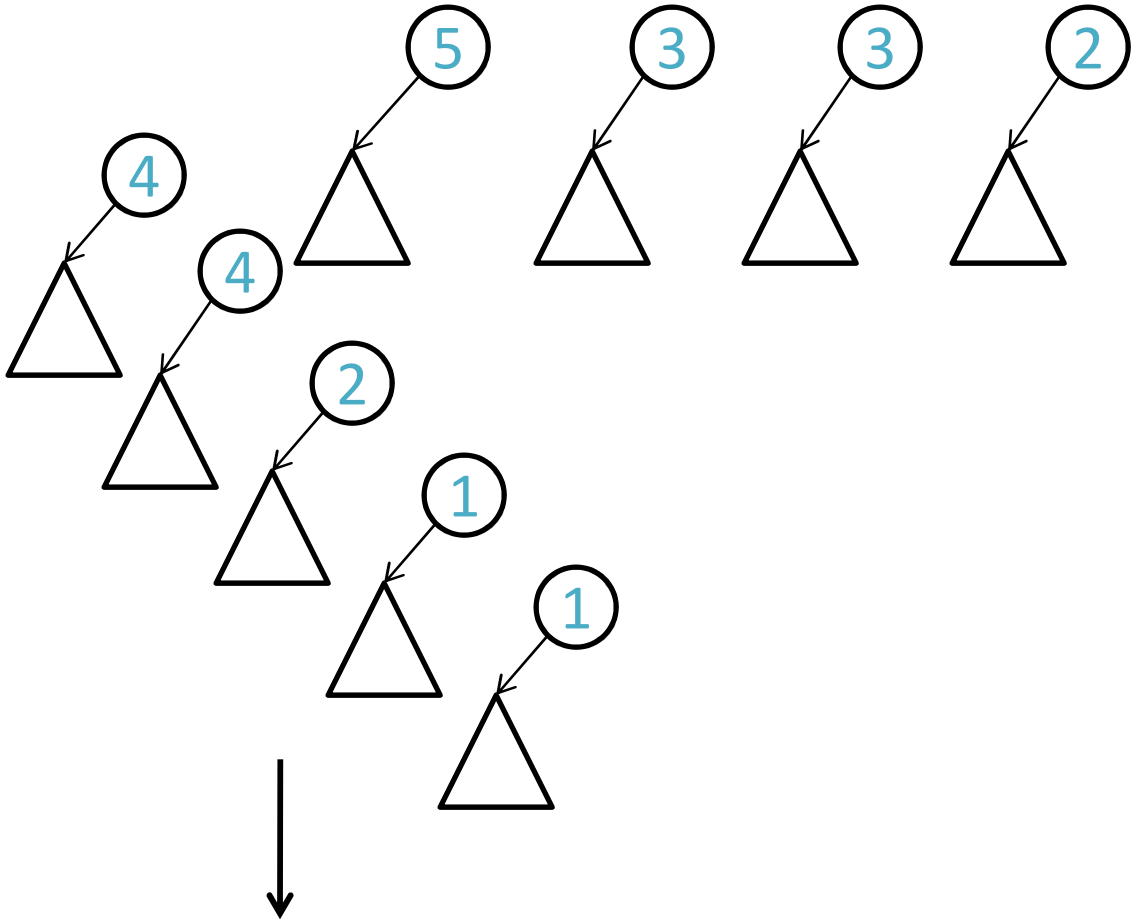
*delete-min*(*H*)

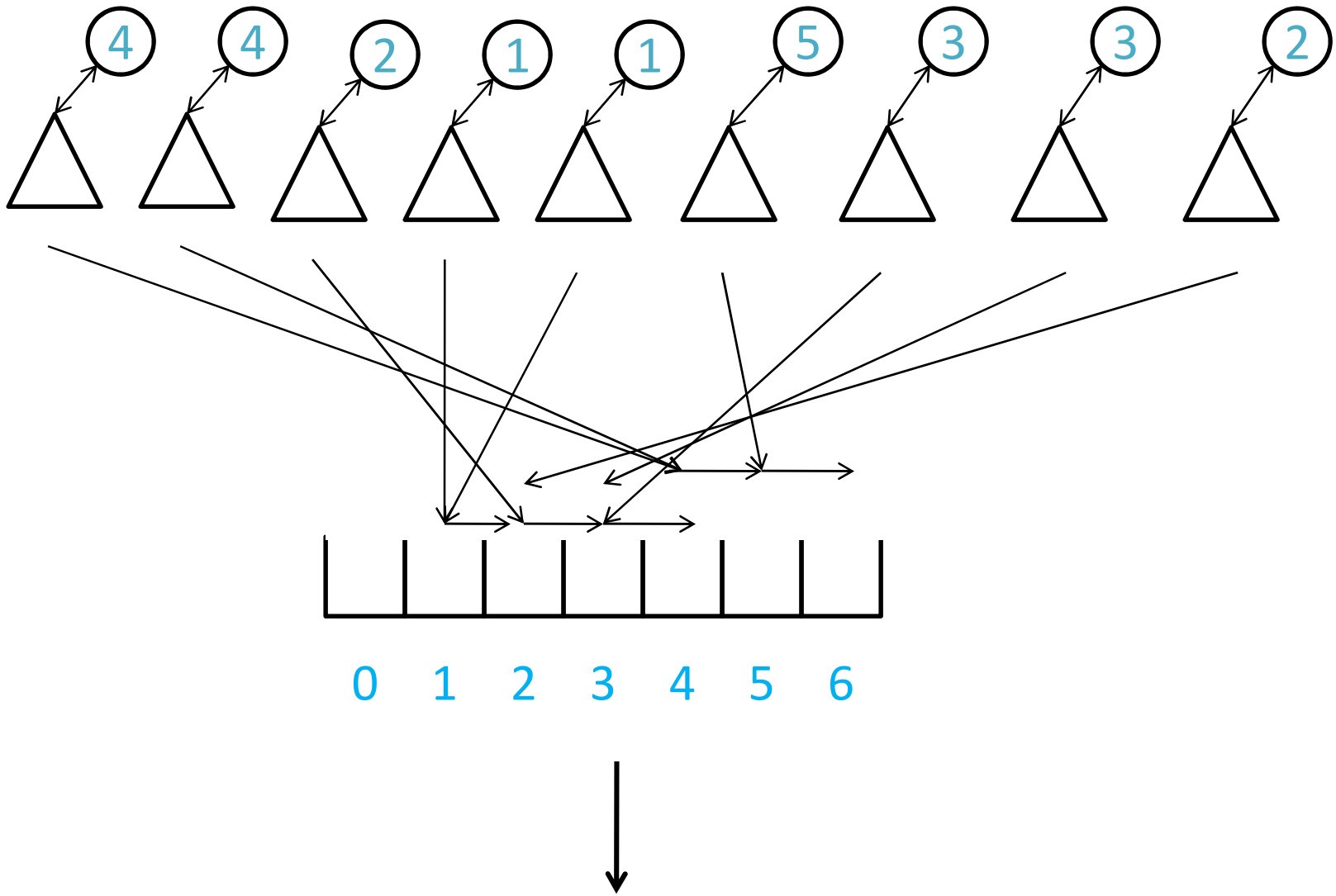
*delete-min*: Delete min-root. Cut edges along right path down from new root to give new half-trees. Set rank of each new root = 1 + rank of left child. Link roots of equal rank until no two roots have equal rank. Form list of remaining half trees.

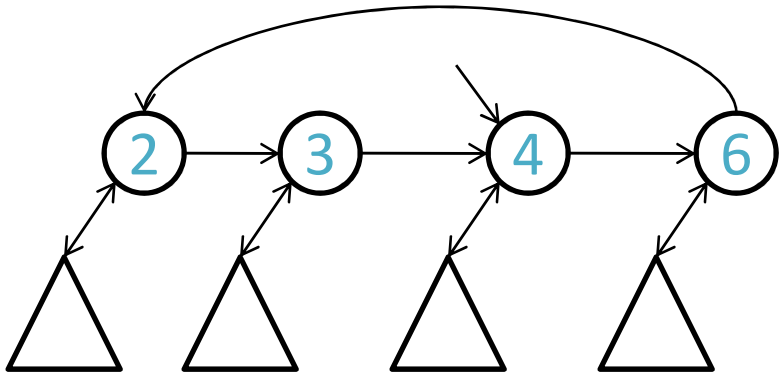
# Delete-min: numbers are keys



# Delete-min: numbers are ranks




4
4
2
1
1
5
3
3
2
0
1
2
3
4
5
6




Each link takes O(1) time, forming final tree takes O(max rank) time → *delete-min* takes O(max rank + #links) time

Data structure: (***type-2***) ***rank-pairing heap***

Invariant on rank differences

→ max rank is O(lg*n*)

Amortize rank decrease steps as well as links

→ amortized time of *decrease-key* is O(1),

other amortized time bounds the same

# Half tree height bound

Fibonacci numbers

$F_0 = 0, F_1 = 1, F_k = F_{k-1} + F_{k-2}$ for $k > 1$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,…

Golden ratio $\varphi = (1 + \sqrt{5})/2$

$\varphi^2 = \varphi + 1$ $\varphi^k \leq F_{k+2}$

For any non-root $x$, $s(x) + 1 \geq F_{r(x)+3}$

For any root x, $s(x) \geq F_{r(x)+2}$

**Proof** for non-roots by induction on $h(x)$:

$s(\text{null}) + 1 = 0 + 1 = F_2$

$s(\text{leaf}) + 1 = 1 + 1 \geq F_3$

$h(x) > 0$: $s(x) + 1 = s(left(x)) + 1 + s(right(x)) + 1 \geq$

$F_{r(x)+2} + F_{r(x)+1} = F_{r(x)+3}$ if $x$ is 1,1 or 1,2;

immediate if $x$ is 0,$j$

$n \geq F_{r(x)+2} \geq \varphi^{r(x)} \rightarrow$

$\boldsymbol{r(x) \leq \lg_{\varphi} n < 1.44043 \lg n}$

How to amortize rank decrease steps and links?

Such a step must decrease Φ

$\Phi(x) = i + j + c$ if $x$ is $i,j$ (c any constant)

→ when r(x) decreases by δ, $\Phi(x)$ decreases by 2δ, $\Phi(p(x))$ increases by δ, net change is –δ

A link converts a root to a 1,1

→ $\Phi(\text{root}) > \Phi(x)$ if $x$ is 1,1

Choosing c = –2, Φ(root) = 1 gives Φ(*x*) = 0 if x 1,1

Almost works!

Problem: 0,2 nodes also have Φ = 0, but such nodes need Φ = 1 when they become roots. Need extra potential.

Only one 1,1 node can decrease in rank during a decrease-key: choose c = –1, give 1,1 nodes potential 0 instead of 1.

$\Phi(x) = r(x) - r(\mathit{left}(x))$ if root

$= 0$ if 1,1

$= 2r(x) - r(\mathit{left}(x)) - r(\mathit{right}(x)) - 1$ otherwise

*make-heap* ($\Delta\Phi = 0$), *find-min* ($\Delta\Phi = 0$), *insert* ($\Delta\Phi = 1$), *meld* ($\Delta\Phi = 0$) take O(1) amortized time

*delete-min*: Each new root needs one unit of Φ, which it has unless it is a 1,1 node. Only $\lg_\varphi n$ 1,1 nodes on a path (at most one per rank)

→ creation of new half trees increases Φ by at most $\lg_\varphi n$

Each link takes 0 amortized time

→ delete-min takes O(lg*n*) amortized time

*decrease-key*: Creating new subtree takes O(1) amortized time ($\Delta\Phi \leq 2$). Each rank decrease step reduces $\Phi$ unless it is of a 1,1 node. Only one 1,1 node can decrease in rank: can only decrease by 1, previous decrease must be by at least 2

→ *decrease-key* takes O(1) amortized time

# Variants

Can strengthen or weaken rank invariant:

Stronger: nodes are 1,1 or 0,*j* for $j > 0$

***type-1 rank-pairing heap***

Weaker: nodes are 1,1; 1,2; 1,3; or 0,*j* for $j > 2$

***type-3 rank-pairing heap***

**Type 1**: smaller height bound but more complicated amortized analysis, worse constant factors, seems to require a special linking order in *delete-min*

**Type 2**: the sweet spot

**Type 3**: simpler amortized analysis (Φ(non-root) = sum of rank differences of children – 2) but bigger height bound

# Why care about fast decrease-key?

Dijkstra's shortest path algorithm: coming up!