Rank-Pairing Heaps

Robert Tarjan, Princeton University & HP Labs

Joint work with Bernhard Haeupler and
Siddhartha Sen, ESA 2009

Heap (Priority Queue) Problem

Maintain a set (heap) of items, each with a real-valued key, under the operations

Find minimum: find the item of minimum key in a heap

Insert: add a new item to a heap

Delete minimum : remove the item of minimum key

Meld: Combine two item-disjoint heaps into one

Decrease key: subtract a given positive amount from the key of a given item in a known heap

Goal: O(log *n*) for delete min, O(1) for others

Related Work

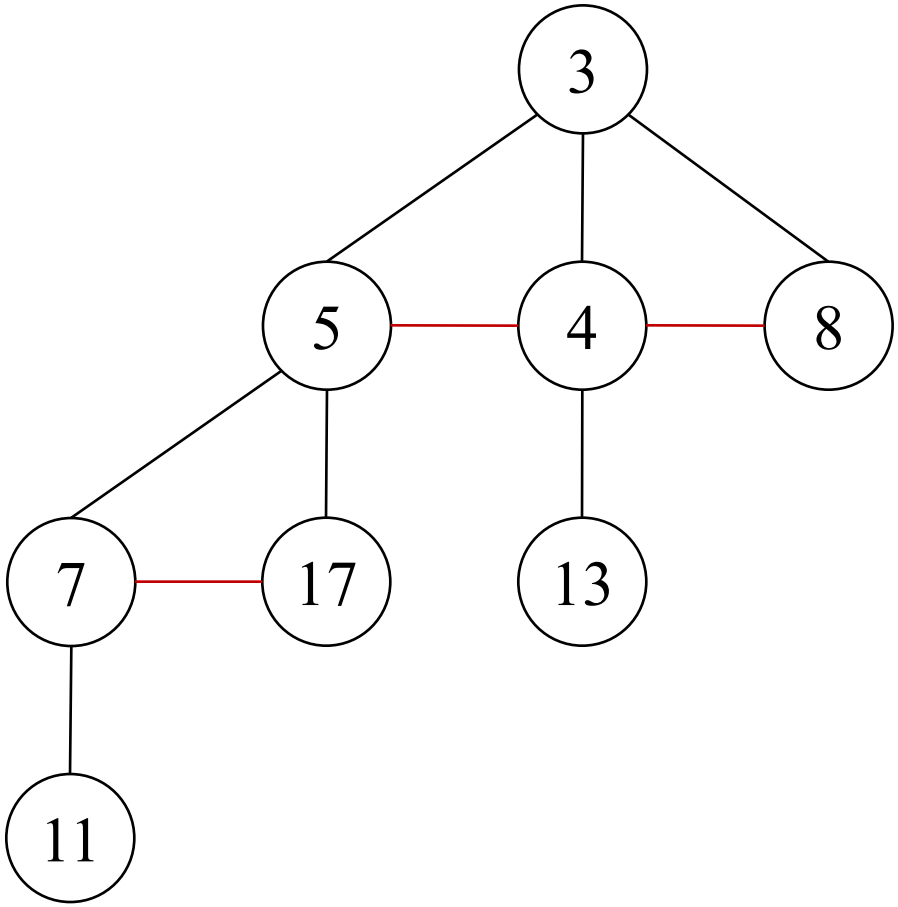
Fibonacci heaps achieve the desired bounds (Fredman & Tarjan, 1984); so do

- Peterson's heaps (1987)
- Høyer's heaps (1995)
- Brodal's heaps (1996), worst-case
- Thin heaps (Kaplan & Tarjan, 2008)
- Violation heaps (Elmasry, 2008)
- Quake heaps (Chan, 2009)

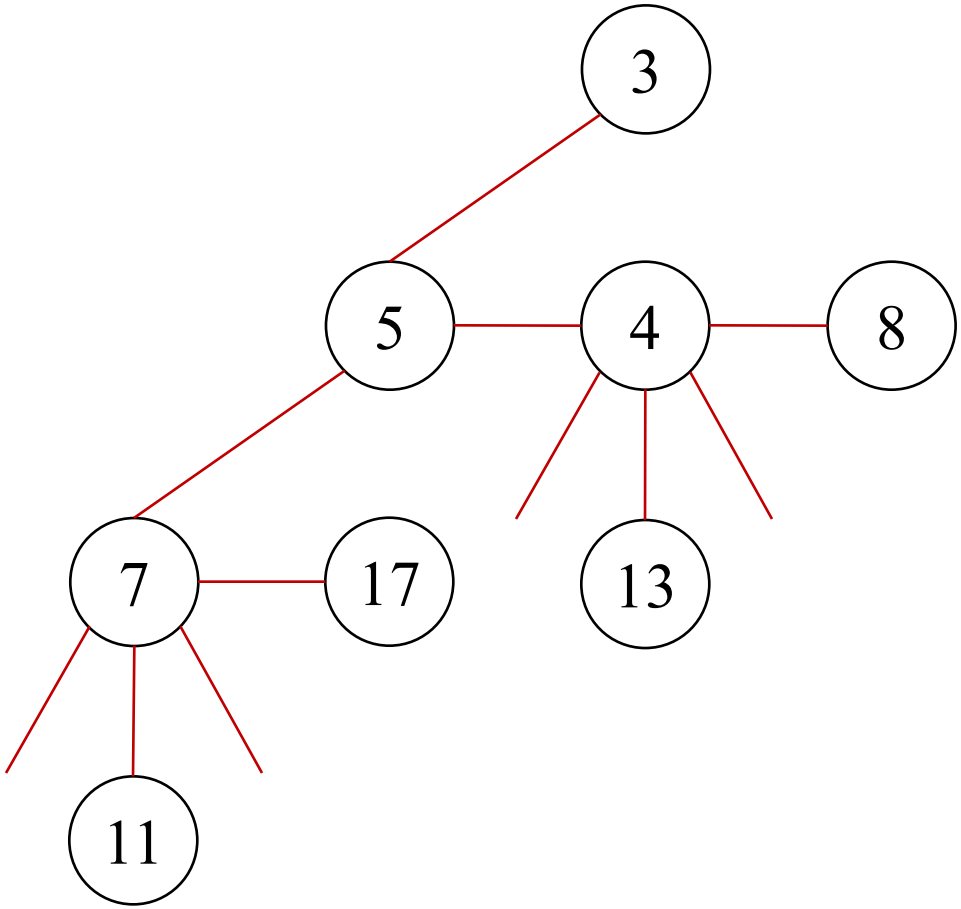
Not pairing heaps:

$\Omega(\log\log n)$ time per key decrease, but good in practice

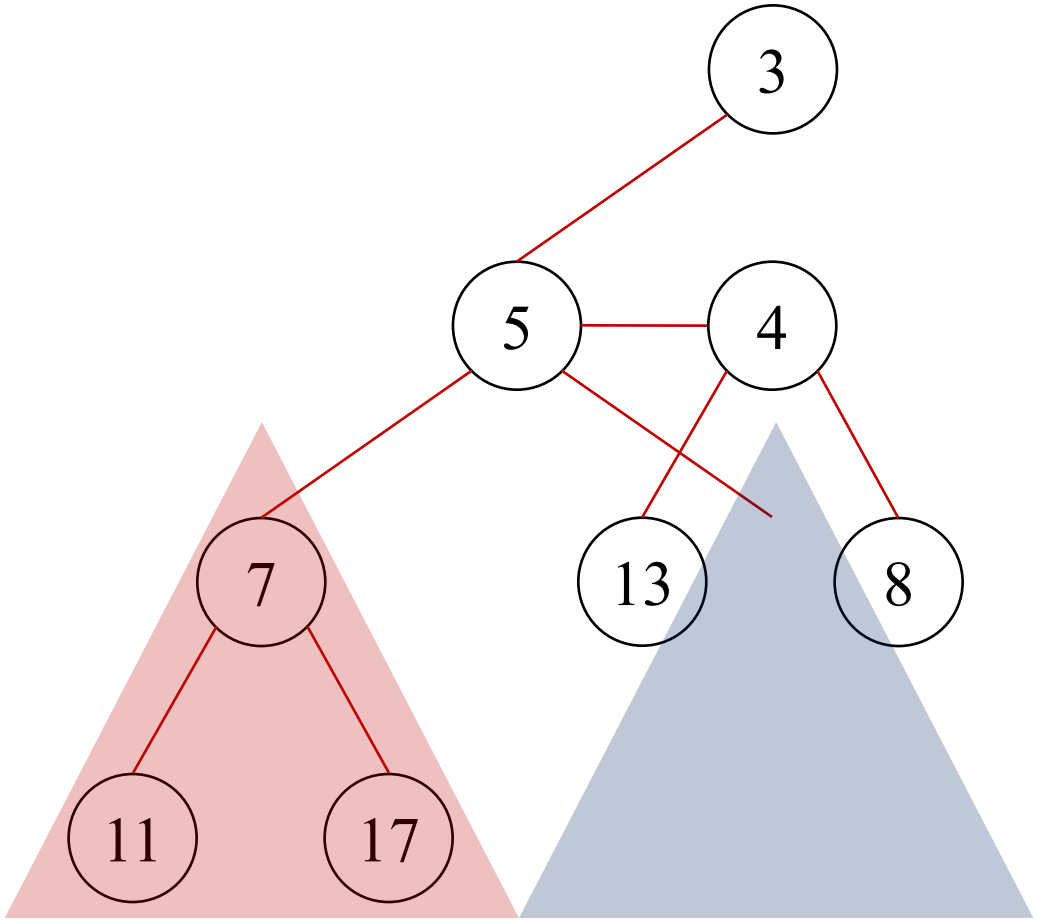
Heap-ordered model



Half-ordered model



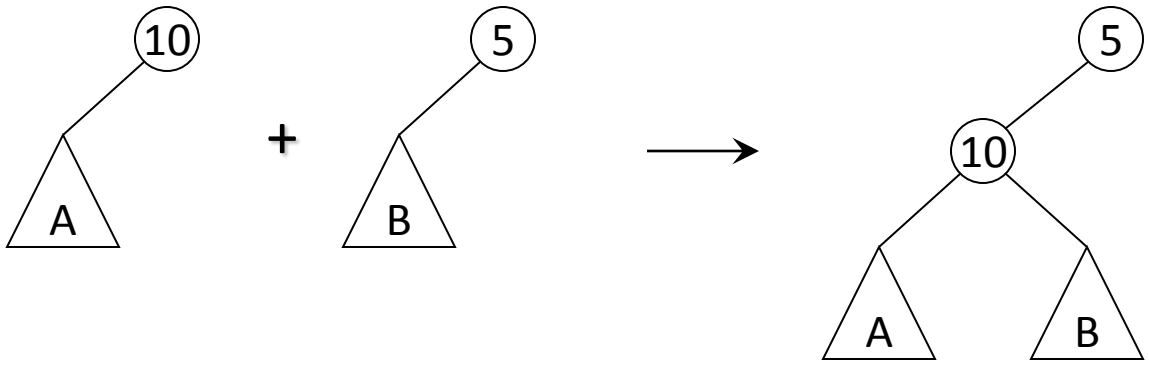
Half-ordered model



Half-ordered tree: binary tree, one item per node, each item less than all items in left subtree

Half tree: half-ordered binary tree with no right subtree

Link two half trees:



O(1) time, preserves half order

Heap: a set (circular singly-linked list) of half trees, with minimum root first on list

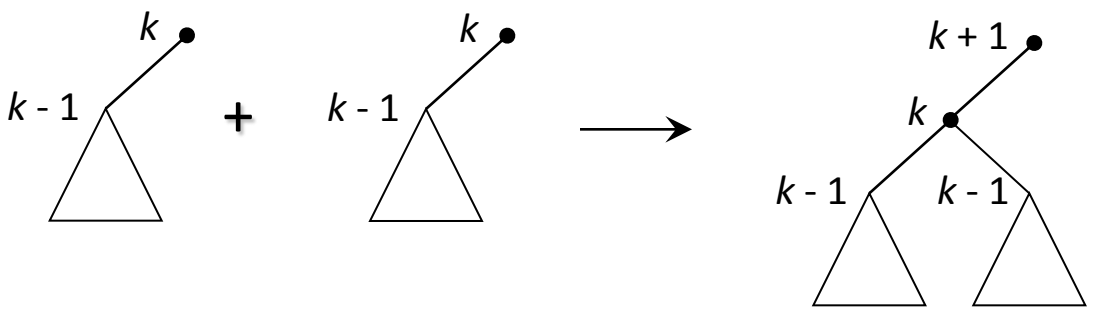
Find min: return minimum

Insert: Form a new one-node tree, combine with current set of half trees, update the minimum

Meld: Combine sets of half trees, update the minimum

Delete min: Remove minimum root (forming new half trees); Repeatedly link half trees, form a set of the remaining trees

How to link? Use ranks: leaves have rank zero, only link trees whose roots have equal rank, increase winner's rank by one:

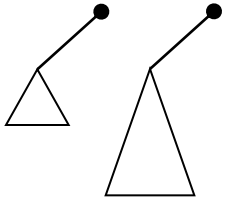


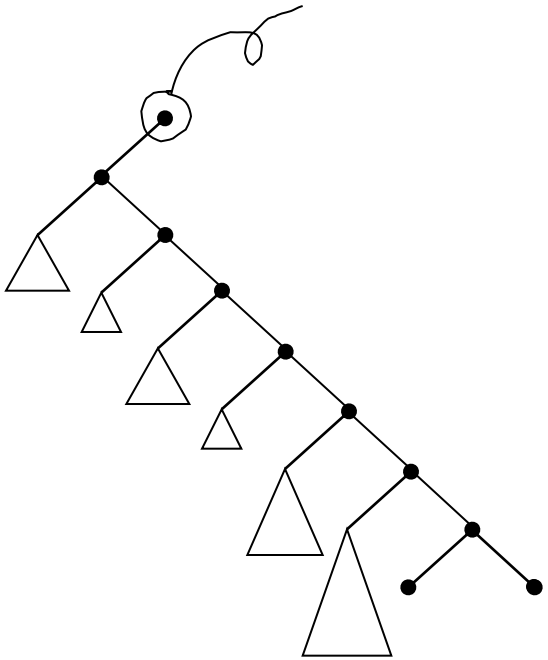
Vuillemin's binomial queues!

All rank differences are 1: a half tree of rank k is a perfect binary tree plus a root:

2^k nodes, rank = lg n

Delete min

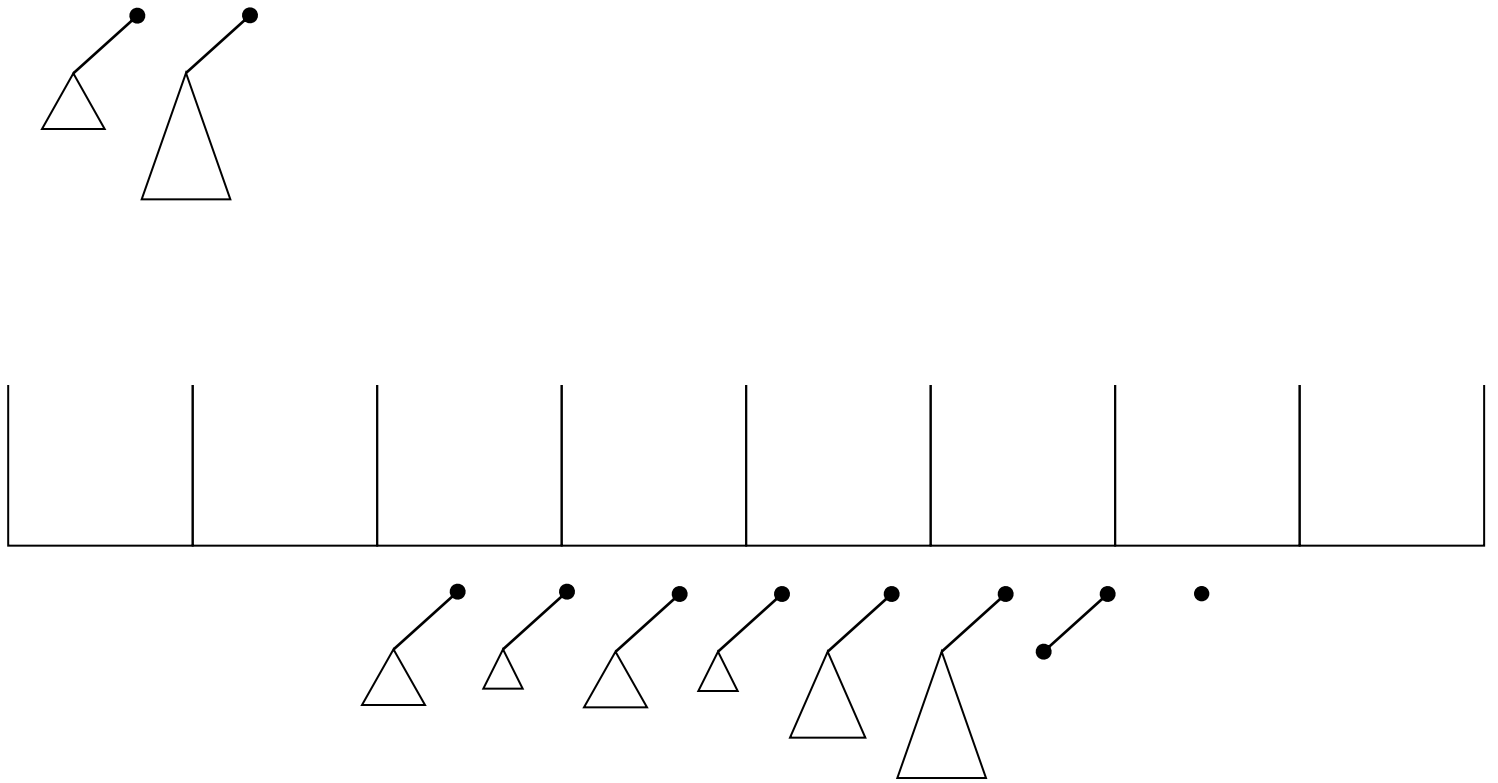




Each node on right path becomes a new root

Delete min

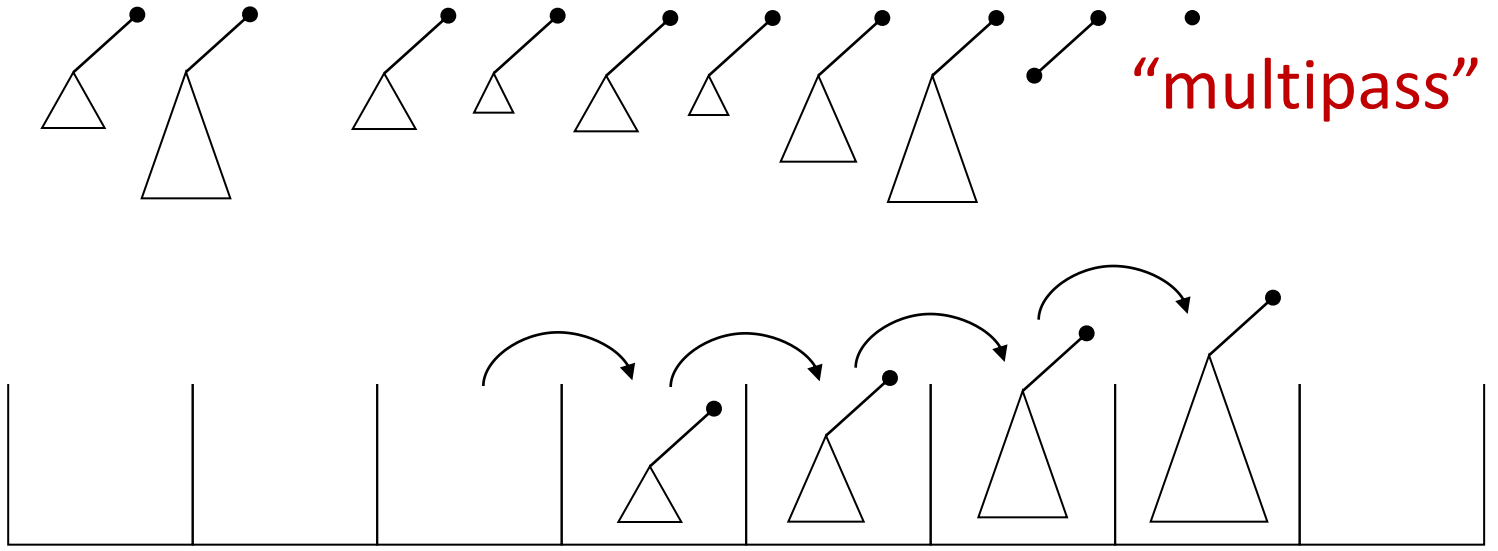
Delete min



Link half trees of equal rank

Array of buckets, at most one per rank

Delete min

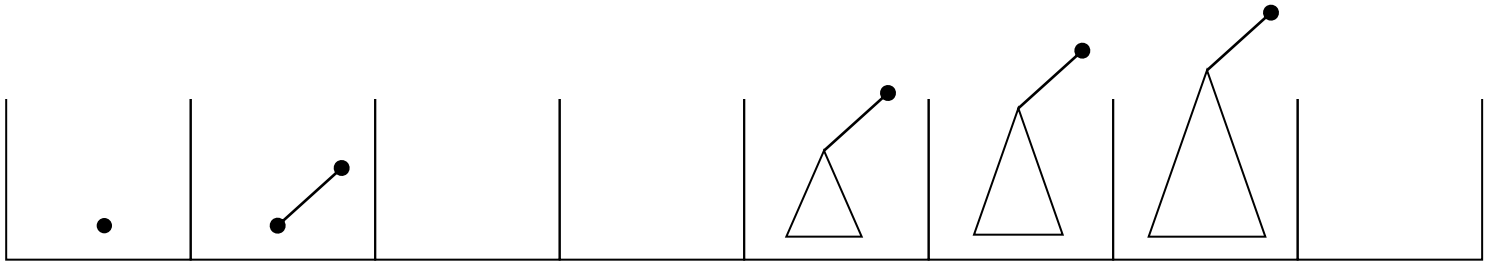


Link half trees of equal rank

Form new set of half trees

Array of buckets, at most one per rank

Delete min

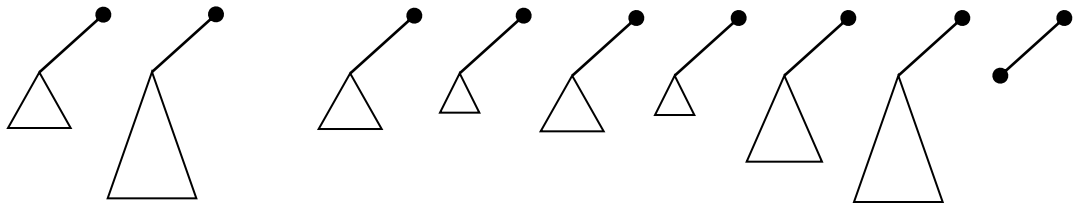


Find min in O(log n) time

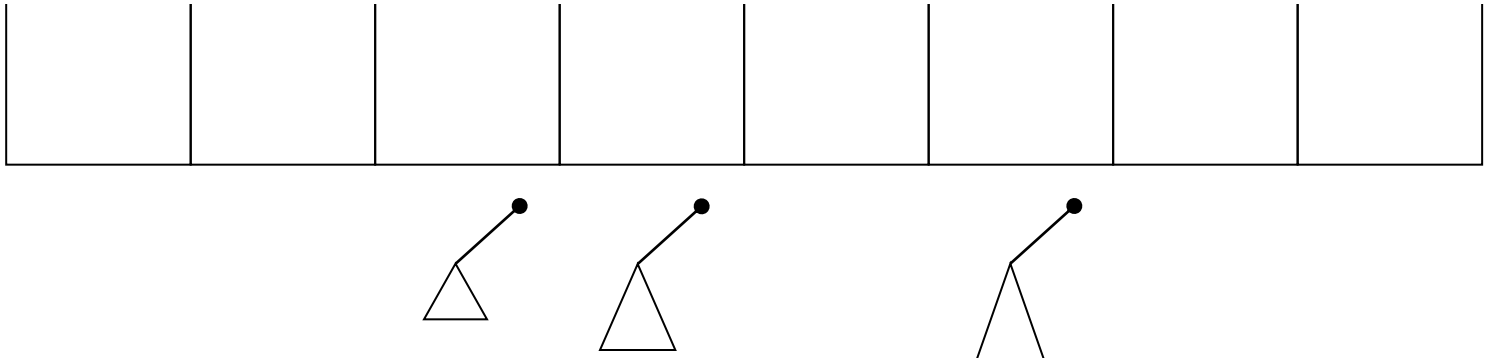
Form new set of heaps

From min in O(log n) time

Delete min: lazier linking



"one-pass"

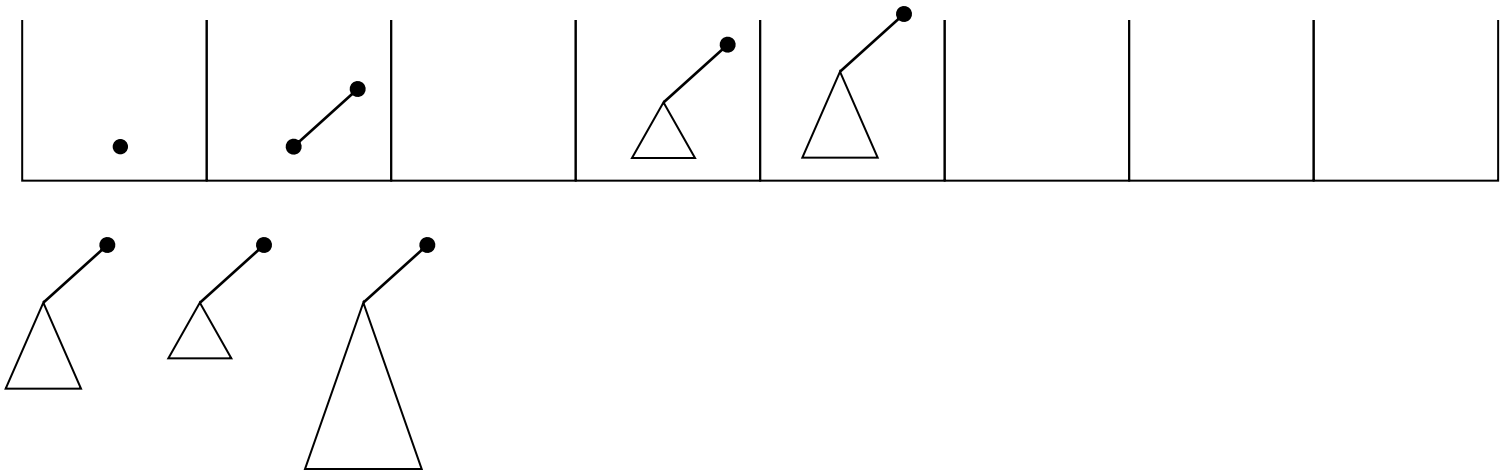


Find minimum in O(log *n*)

Keep track of minimum during links

of trees in O(1) time

Delete min: lazier linking



Form new set of half trees

Amortized Analysis of Lazy Binomial Queues

Φ = #trees

Link: O(1) time, $\Delta\Phi = -1$, amortized time = 0

Insert: O(1) time, $\Delta\Phi = 1$

Meld: O(1) time, $\Delta\Phi = 0$

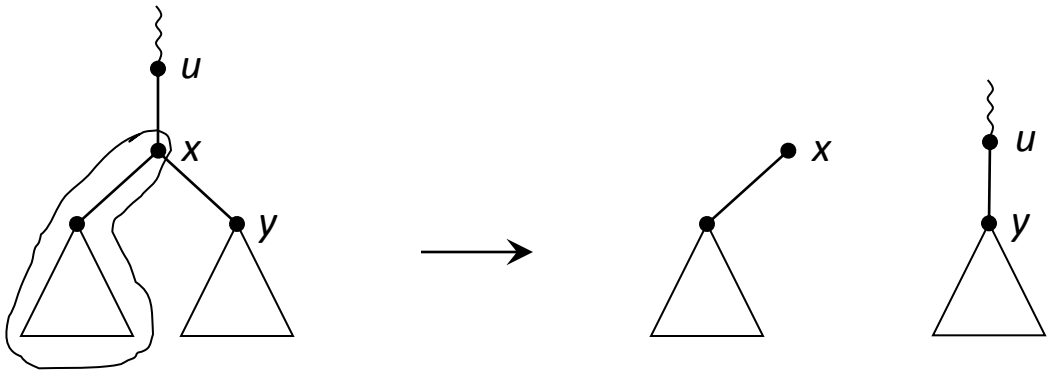
Delete min: if *k* trees after root removal, time is O(k), potential decreases by k/2 – O(log *n*)

$\Rightarrow$ O(log *n*) amortized time

Decrease key?

Application: Dijkstra's shortest path algorithm, others

Method: To decrease key of *x*, detach its half tree, restructure if necessary



(If *x* is the right child of u, no easy way to tell if half order is violated)

How to maintain structure?

All previous methods, starting with Fibonacci heaps, change ranks and restructure

Some, like Quake heaps (Chan, 2009) and Relaxed heaps (Driscoll et al., 1988), do not restructure during key decrease, but this just postpones restructuring

But all that is needed is rank changes:

Trees can have arbitrary structure!

Goal is SIMPLICITY

Rank-Pairing Heaps = rp-heaps

Node Ranks

Each node has a non-negative integer rank

Convention: missing nodes have rank -1
(leaves have rank 0)

rank difference of a child =
rank of parent - rank of child

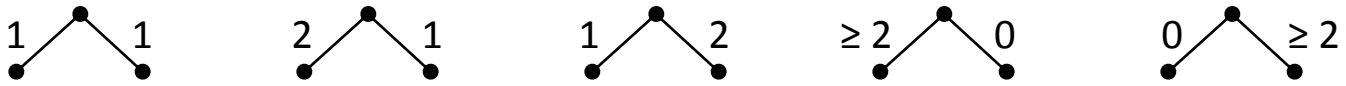
i-child: node of rank difference *i*
i,*j*-node: children have rank differences *i* and *j*

Convention: the child of a root is a 1-child

Rank Rules

Easy-to-analyze version (type 2):

- All rank differences are non-negative
- If rank difference exceeds 2, sibling has rank difference 0
- If rank difference is 0, sibling has rank difference at least 2



Simpler but harder-to-analyze version (type 1):

- If rank difference exceeds 1, sibling has rank difference 0
- If rank difference is 0, sibling has rank difference ≥ 1

Tree Size (type 2)

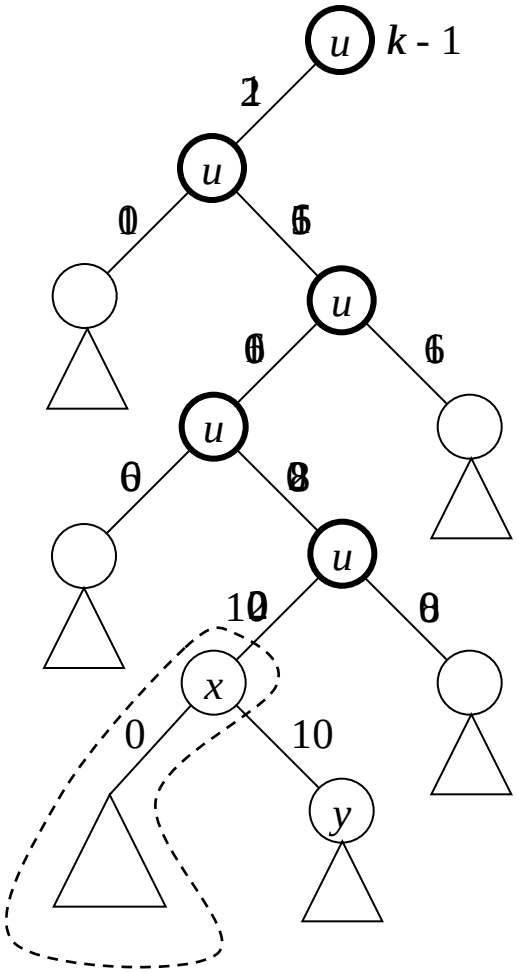
If n_k is minimum number of descendants of a node of rank k,

$n_0 = 1$, $n_k = n_{k-1} + n_{k-2}$: Fibonacci numbers

$n_k \geq \phi^k$, $\phi = \frac{1+\sqrt{5}}{2}$

$k \leq \log_\phi n$

Decrease key



propagation of rank decreases from sibling's subtree

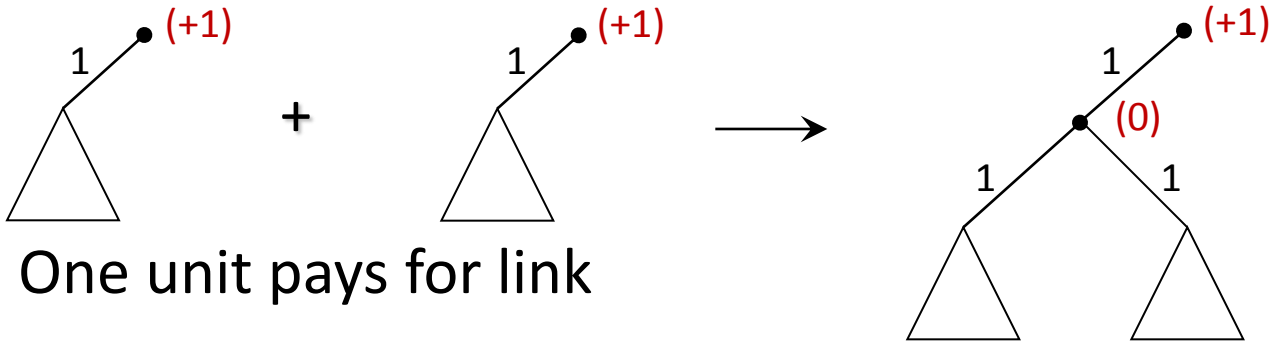
Amortized Analysis

Potential of node = sum of rank differences of children - 1

+1 if root (= 1)

-1 if 1,1-node (= 0)

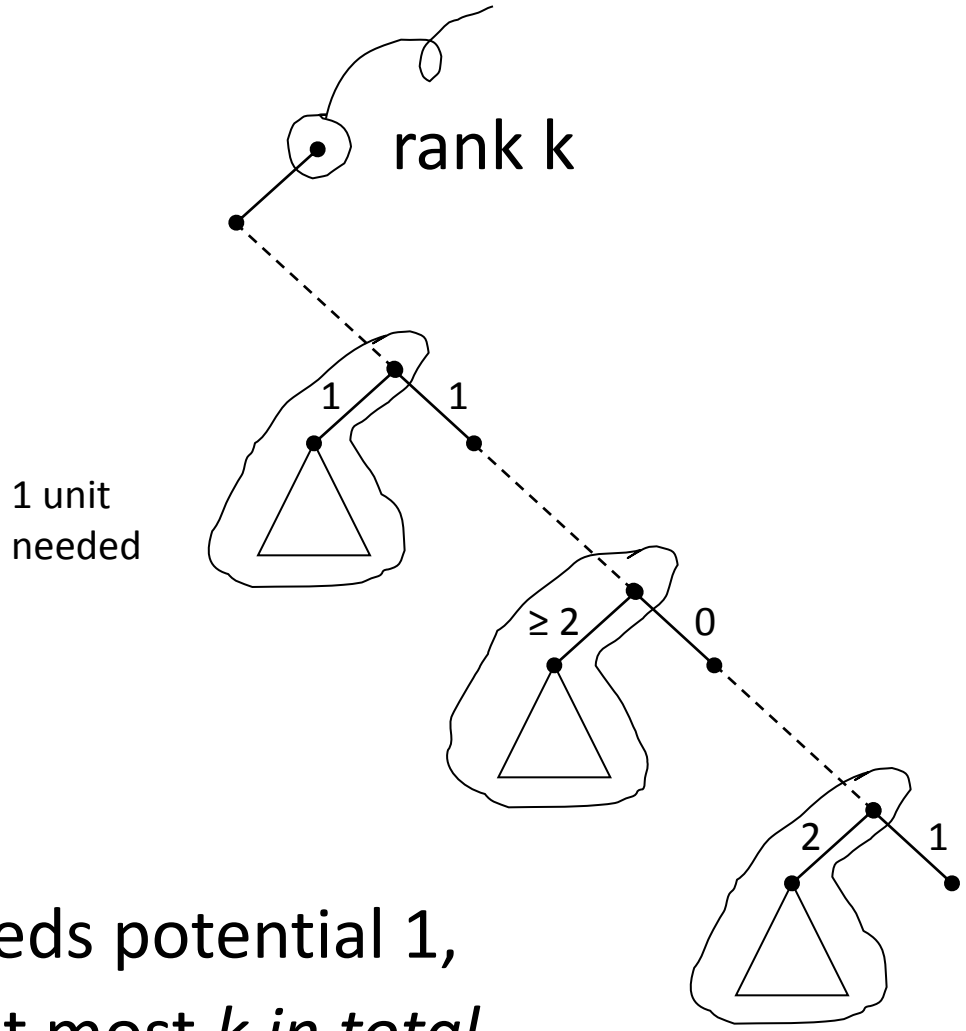
Link is free:



One unit pays for link

Insert needs 1 unit, meld none

Delete min



Each 1,1 needs potential 1,

adding at most *k in total.*

Delete min takes O(log *n*) amortized time

Decrease Key

Successive rank decreases are non-increasing

At most two 1,1's occur on path of rank decreases –

1,1 becomes 0, *j* : prev decrease >1, next decrease = 1

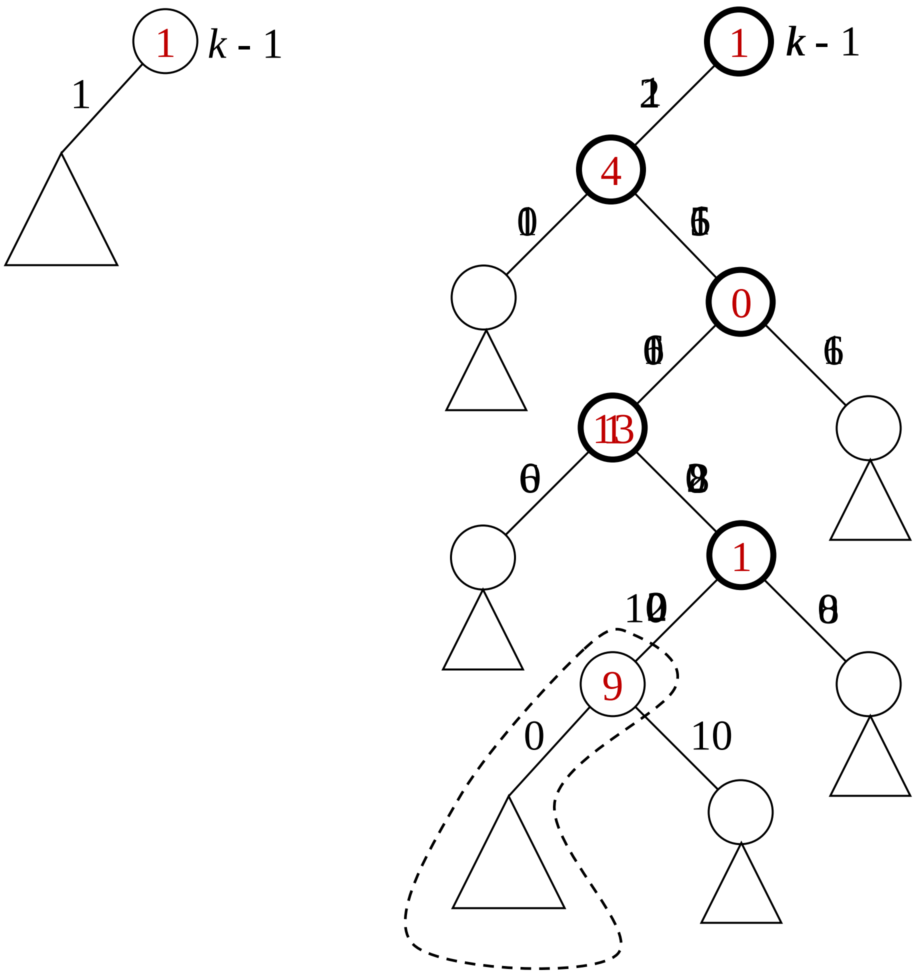
1,1 becomes 1,2 : terminal

Give each 1,1 one extra unit of potential

Each rank decrease releases a unit to pay for decrease:

rank diffs of both children decrease by *k*, rank diff of parent increases by *k*

Decrease key



Detach half tree

Restore rank

Type-1 rp-heaps

Max $k \leq \lg n$

Analysis requires a more elaborate potential based on rank differences of children and grandchildren

Same bounds as type-2 rp-heaps, provided we preferentially link half trees from disassembly