CSE 5311 Notes 4a: Priority Queues

(Last updated 1/12/21 10:56 AM)

Chart on p. 506, CLRS (binary, Fibonacci heaps)

MAKE-HEAP

INSERT

MINIMUM

EXTRACT-MIN

UNION (MELD/MERGE)

DECREASE-KEY

DELETE

Applications - sorting (?), scheduling, greedy algorithms, discrete event simulation

Ordered lists - Suitable if *n* is extremely small (some simulations)

Binary trees - O(log *n*) operations, but larger constant than binary heaps. O(*n*) for UNION.

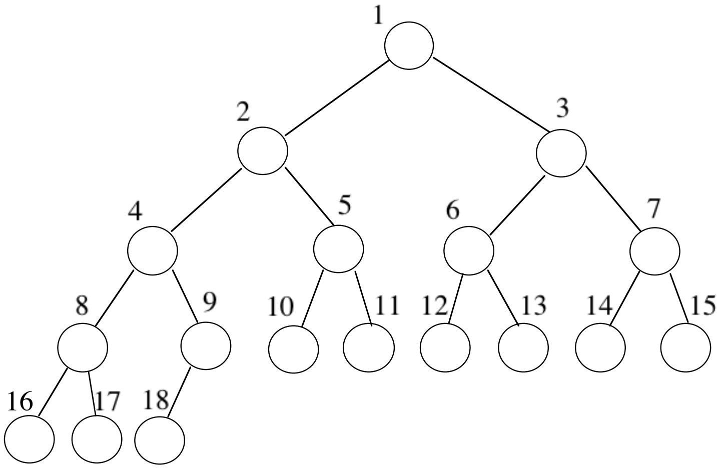
Binary heap (review)

Conceptual structure

Ordering criteria

Mapping to table

O(log *n*) operations, except UNION



d-heap

Generalizes binary heap with fan-out of *d* to get shallower structure.

Similar details as binary heap for mapping to an array.

Useful when many DECREASE-KEYs occur (example: Prim's MST, $\Theta(|E|\log|V|)$ - use $d = |E| / |V|$)

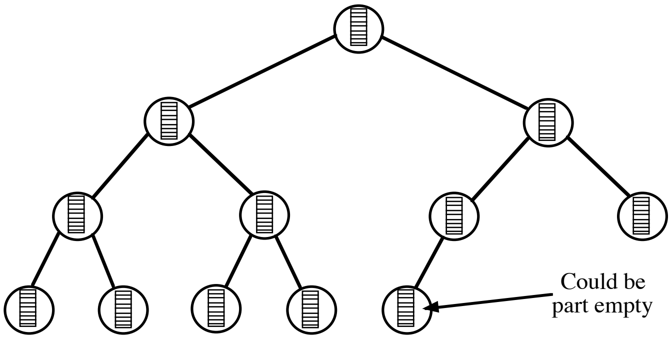
(Aside) General issue - single-valued nodes vs. nodes containing table ("sack") with O(log *n*) values

Table is in ascending priority order.

Most operations operate locally on one table.

If the first table element changes (minheap), then traditional heap processing occurs.

Tree structure must be linked, i.e. mapping nodes to table is too slow.



Moret, B.M.E., and Shapiro, H.D., "An empirical assessment of algorithms for constructing a minimal spanning tree", in *Computational Support for Discrete Mathematics*, N. Dean and G. Shannon, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science 15 (1994), 99–117.

LEFTIST HEAPS

Binary tree, heap ordered

Each node has a *null path length*

Either subtree empty ⇒ NPL = 0

Otherwise NPL = 1 + min(left→NPL, right→NPL) (Empty tree/sentinel - view NPL as -1)

Leftist property: left→NPL ≥ right→NPL at all nodes.

Leftmost path length is O(*n*).

Rightmost path length is O(log *n*).

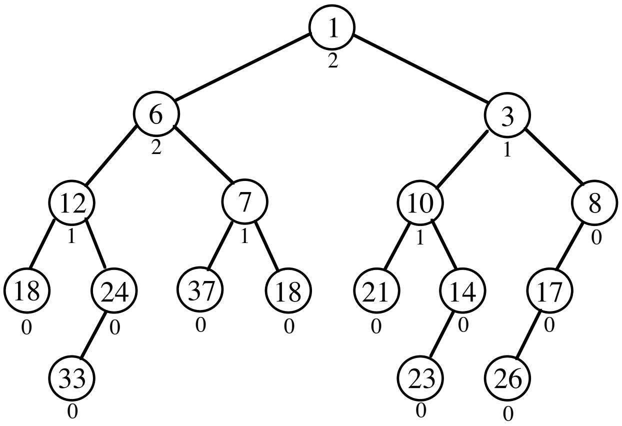
Leftist tree with *r* nodes on right path must have at least 2^r - 1 nodes.

(e.g. NPL is height of maximum embedded complete binary tree)

Operations take O(log *n*) by avoiding left paths and emphasizing right paths.

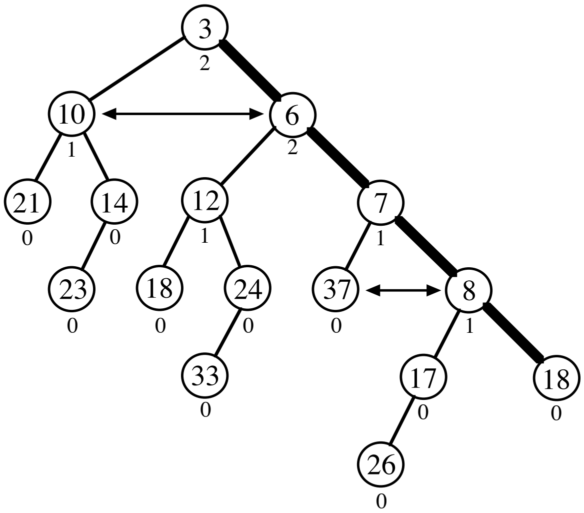
Occasionally, left and right subtrees must be swapped.

Example: EXTRACT-MIN

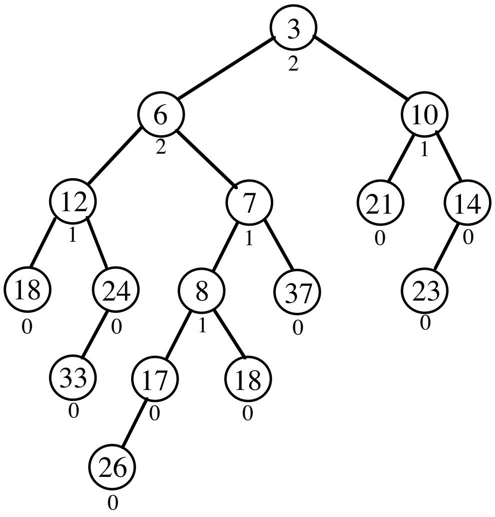


Root has item to return.

Recursively, merge right paths of two remaining subheaps (top-down) keeping same left children.



Swap subtrees, bottom-up, if necessary to restore leftist property.



UNION takes O(log n) time, so use to implement other operations.

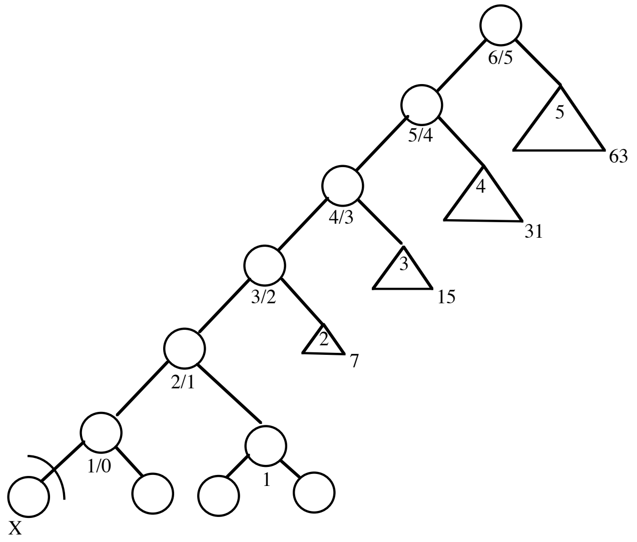
BUT, DECREASE-KEY may involve a node $\Omega(n)$ away from root, so swapping through ancestors is too slow.

1. Find node X via another data structure.

2. *Cut* X's subtree away from parent.

3. Update NPL on former ancestors of X, swapping subtrees to restore leftist property.

   Decrease in NPL continues to propagate only when ancestor NPL decreases. (Implies O(log n)) Ancestors in diagram below are marked with before/after NPLs for cutting away the leftmost leaf.

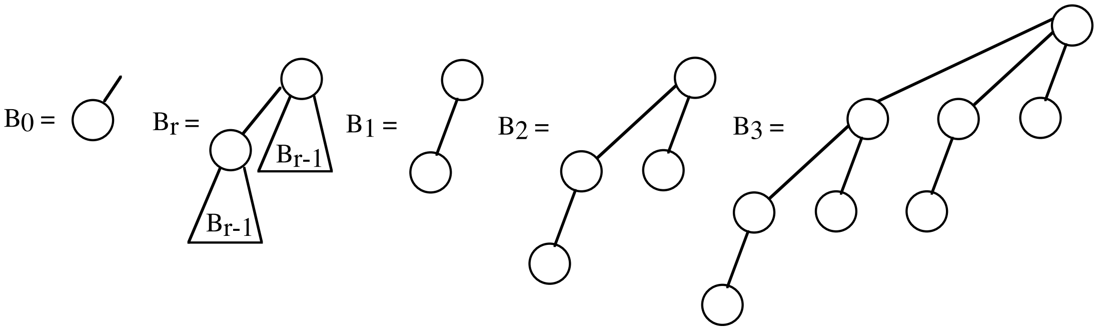


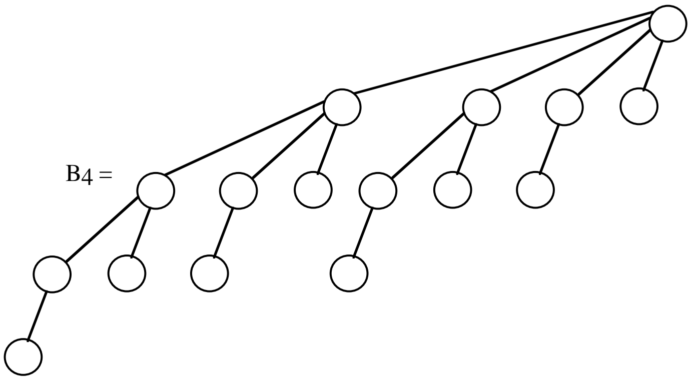
4. UNION X's subtree and modified original tree.

BINOMIAL HEAPS (See CLRS problem 19-2)

Mergeable Heap - in O(log n) time

Based on Binomial Tree (with heap ordering) - $|B_r| = 2^r$





Binomial Heap = Forest of Binomial Trees

Each node includes priority, leftmost child, right sibling, parent, and degree.

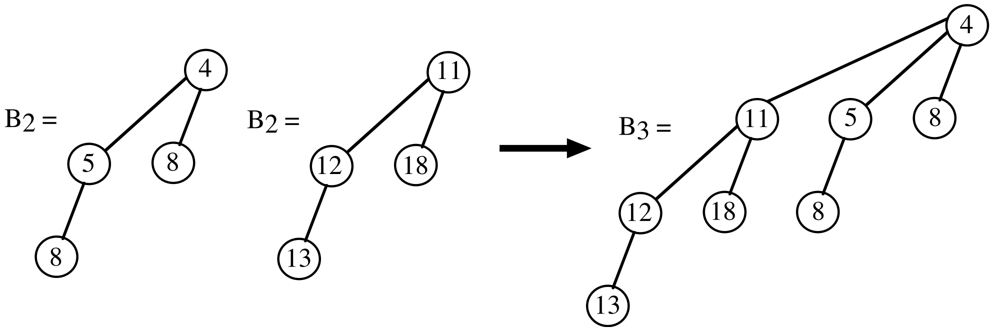
Tree roots are in a singly-linked list ordered by ascending degrees.

Children are in a singly-linked list ordered by descending degrees (could also use ascending).

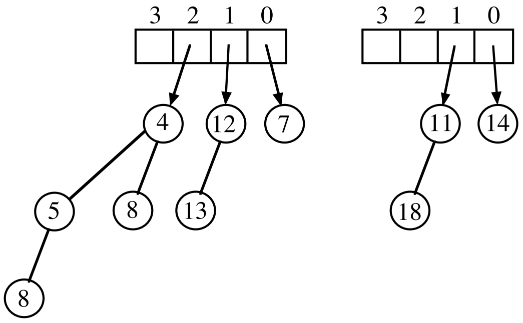
("Sack" idea can be used to reduce overhead from pointers)

Can't have two B_i trees for any $i \Rightarrow$ Use binary representation of n.

Representation is useful for combining 2 B_i trees:



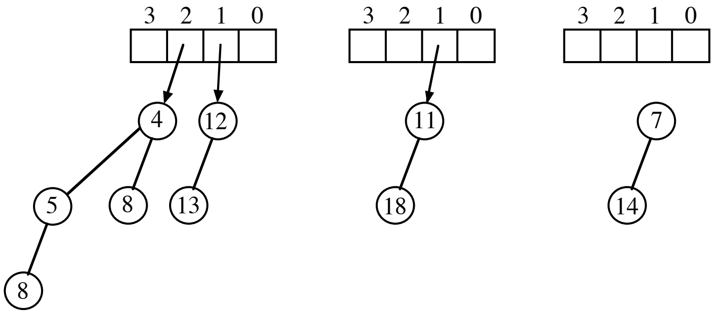
UNION of two binomial heaps



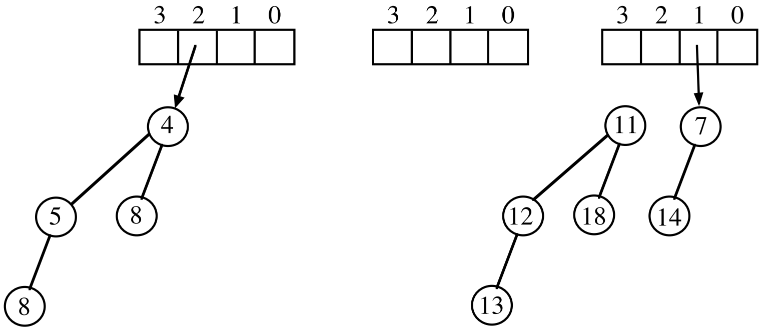
Based on binary addition:

0111 + 0011 =1010

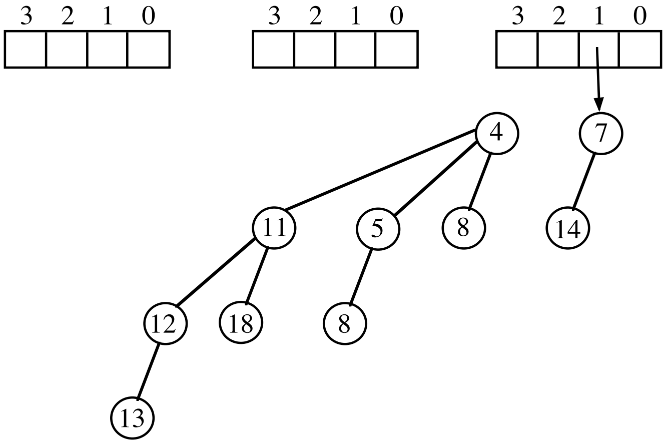
Link B_0 trees:



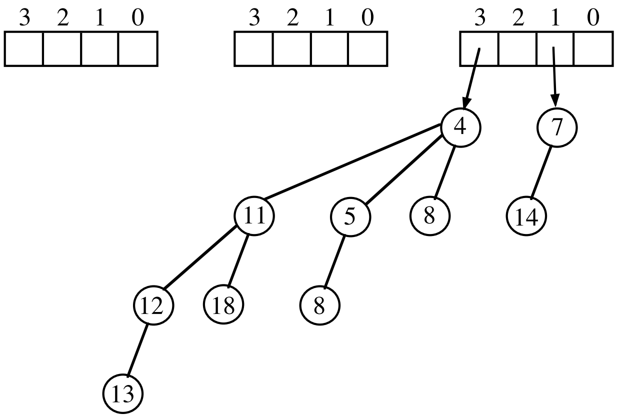
Link B_1 trees:



Link B_2 trees:



Save B_3 tree

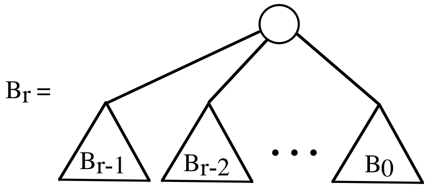


Insertion into binomial heap?

Implementing Extract-Min

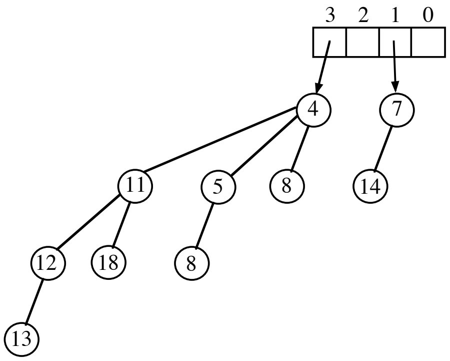
1. Scan tree roots for minimum key.

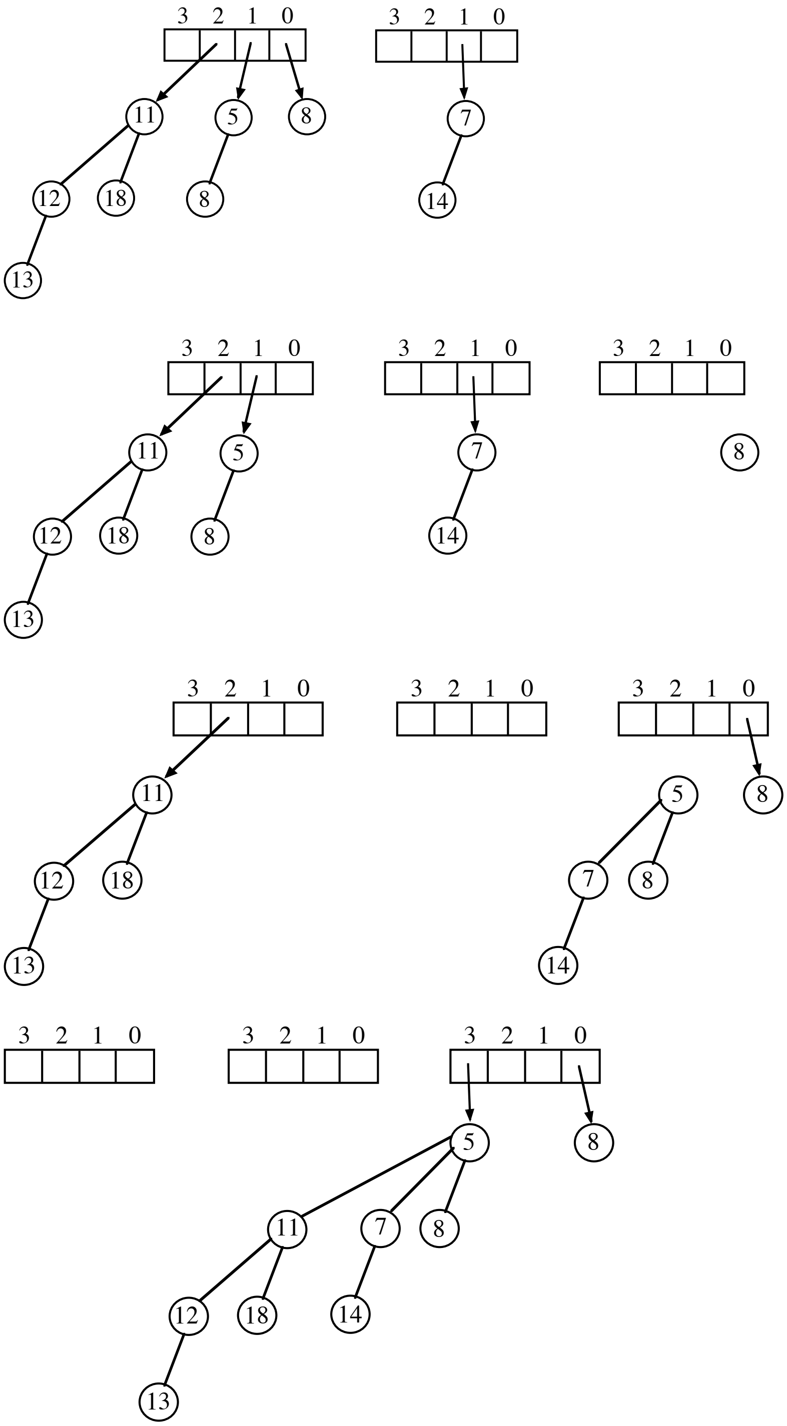
2. Decompose root of tree with minimum:



3. Treat fragments as binomial heap and Union with remainder of original heap.

Example: Returns item 4 and decomposes the B_3 tree





Implementing DECREASE-KEY

Simply do exchanges through ancestor chain until min-heap property has been restored.

Suppose 13 is decreased to 6 in the previous example.

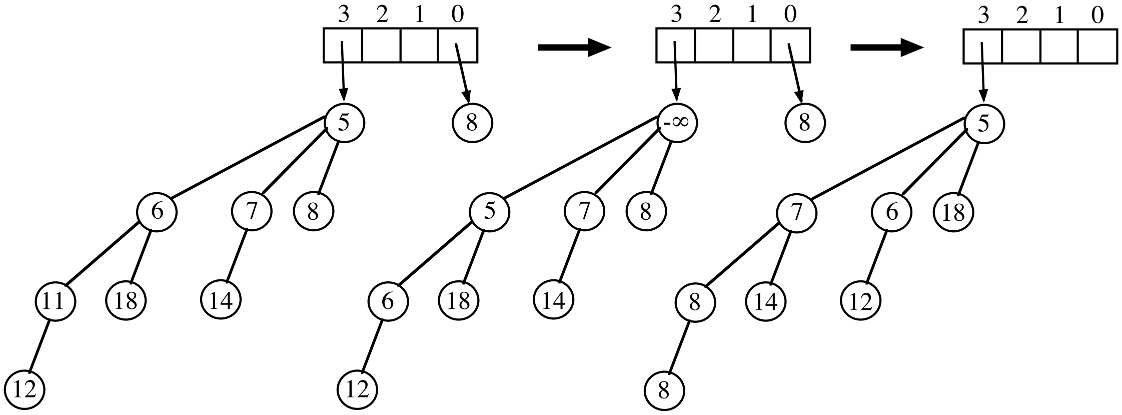
Implementing DELETE

1. Auxiliary data structure (see CSE 2320 Notes 5.D regarding dictionary) is used to find the node to delete.

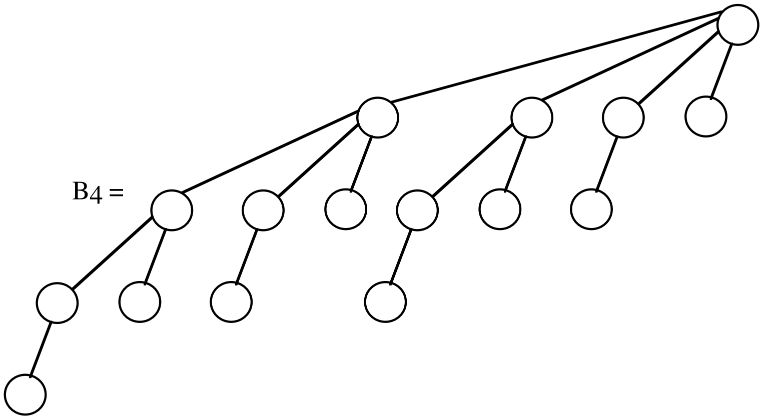
2. Use DECREASE-KEY to change priority to -∞.

3. Use EXTRACT-MIN to eliminate -∞.

Example: Delete 11.



Increase a key? What happens if obvious method is applied for key at root?



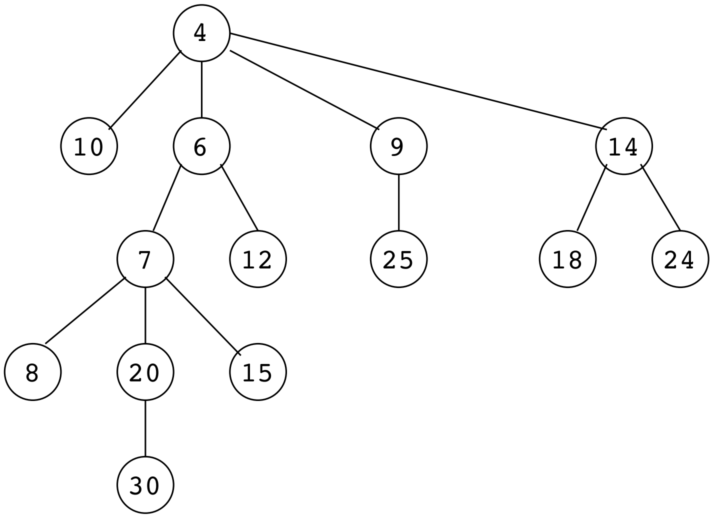
Binomial Heaps	vs.	Fibonacci Heaps
O(log *n*) actual costs		O(1) amortized, except Extract-Min and Delete (O(log *n*) amortized)
		Decrease-Key is “faster”
Strict structural properties		Flexible structural properties (Allows laziness)
Analysis is straightforward		Amortized analysis involves subtle arguments regarding constants for asymptotic notation (especially for Extract-Min and cascading cut)

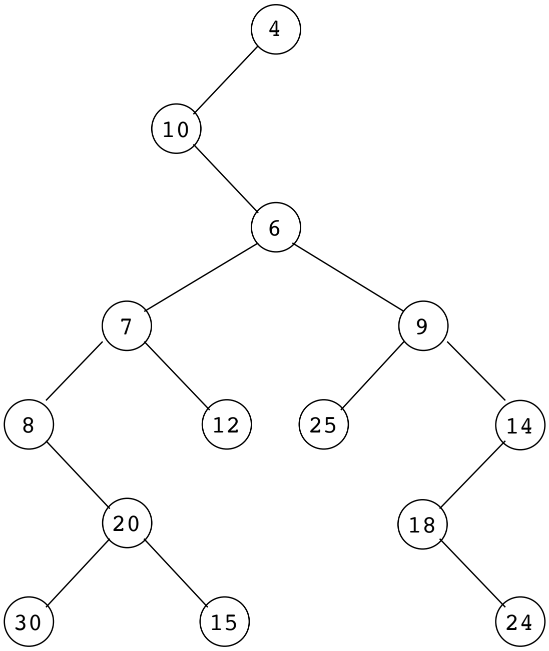
Pairing Heaps

Practical alternative to Fibonacci heaps.

Amortized costs are slightly higher than Fibonacci heaps.

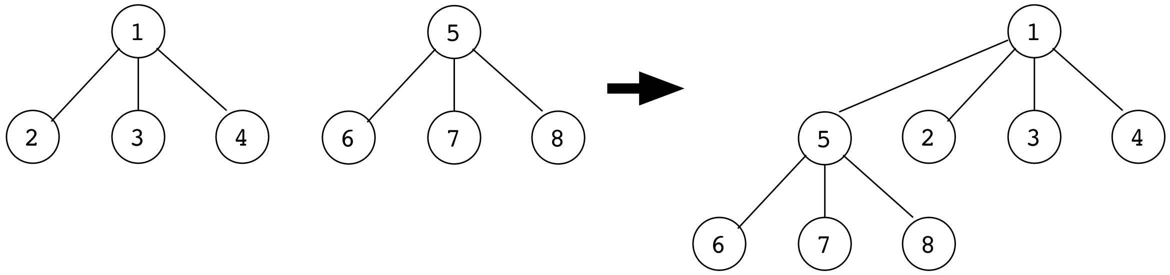
Structure is based on mapping multi-way tree to binary tree (left child, right sibling representation) that includes parent pointers.

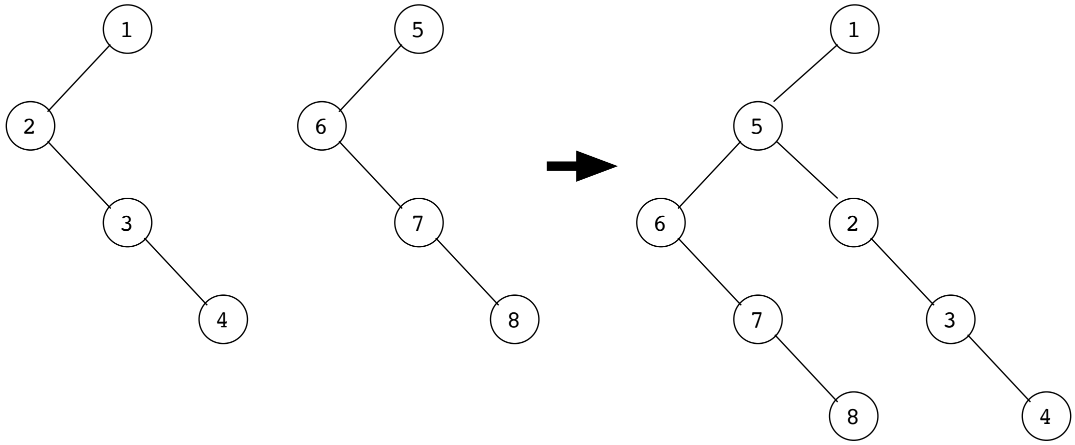




Unions are used heavily, but the structure is more flexible than a binomial heap.

UNION:





As usual, INSERT is just a UNION.

MINIMUM just requires returning the priority in the root.

EXTRACT-MIN is processed by:

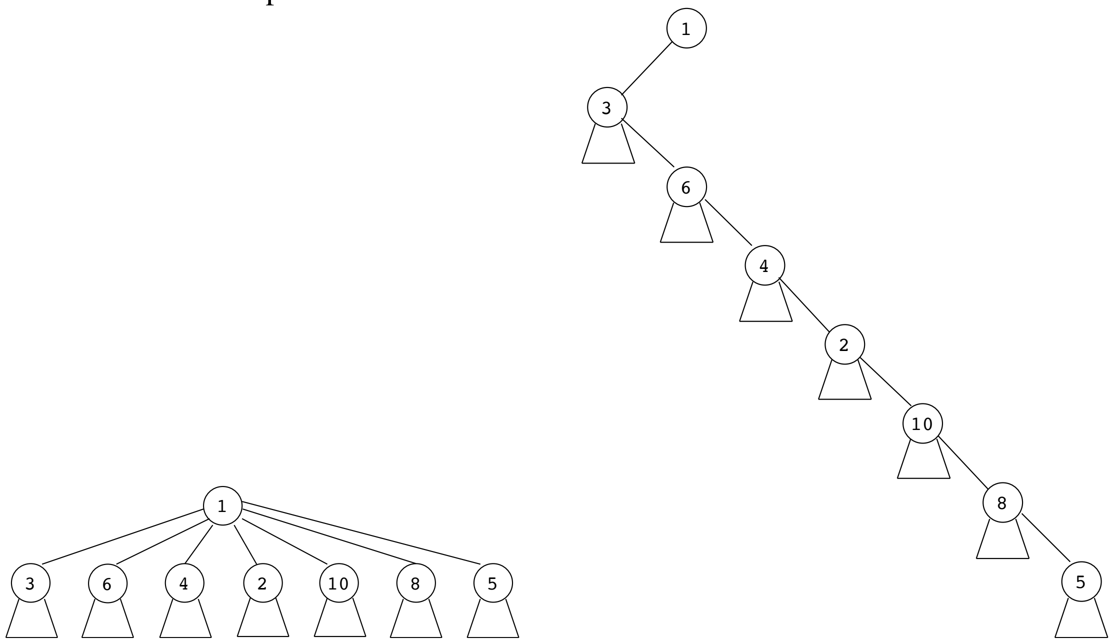
1. Remove root node. Stored priority will be returned.

2. Process rightmost path of the remaining tree using UNION on pairs going left-to-right. (If number of nodes is odd, then rightmost node is treated as a result.)

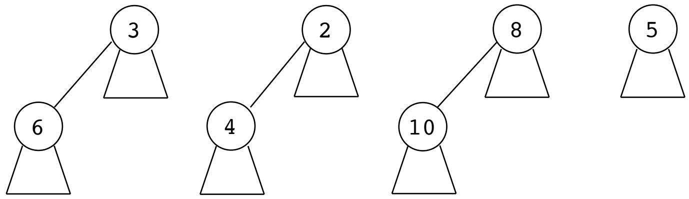
3. Process the *k* results from (2.) in right-to-left order using *k*-1 UNIONs with a result tree, i.e.

```
result := tree[$k$];
for ($i$=$k$-1; $i$>0; $i$--)
    result := UNION (result, tree[$i$])
```

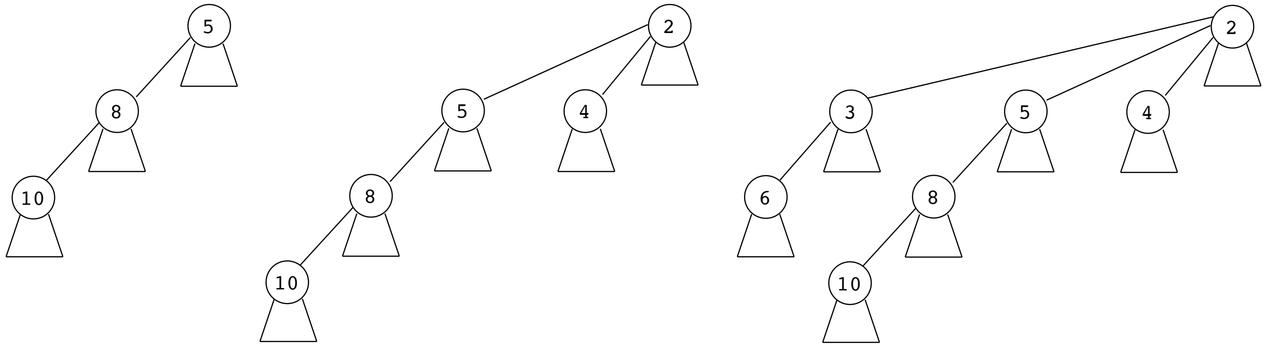
EXTRACT-MIN example



After steps (1.) and (2.):

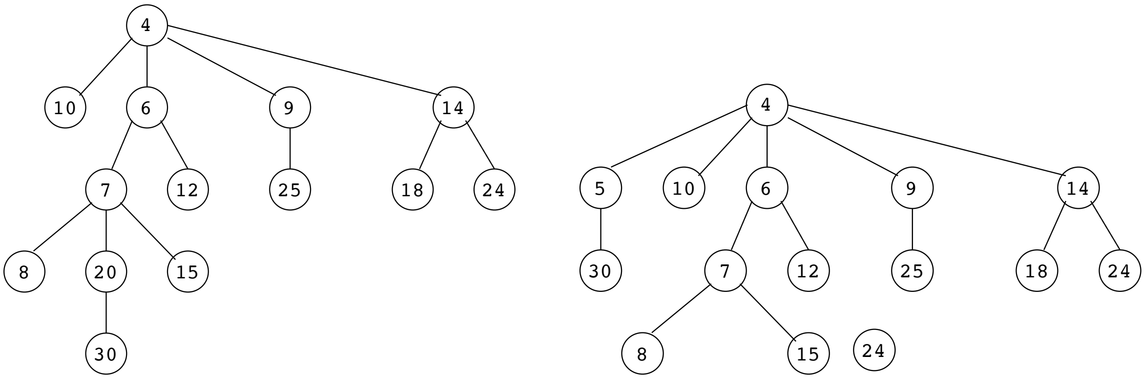


Successive result trees:



DECREASE-KEY will cut away tree (if necessary due to loss of heap ordering) and then UNION.

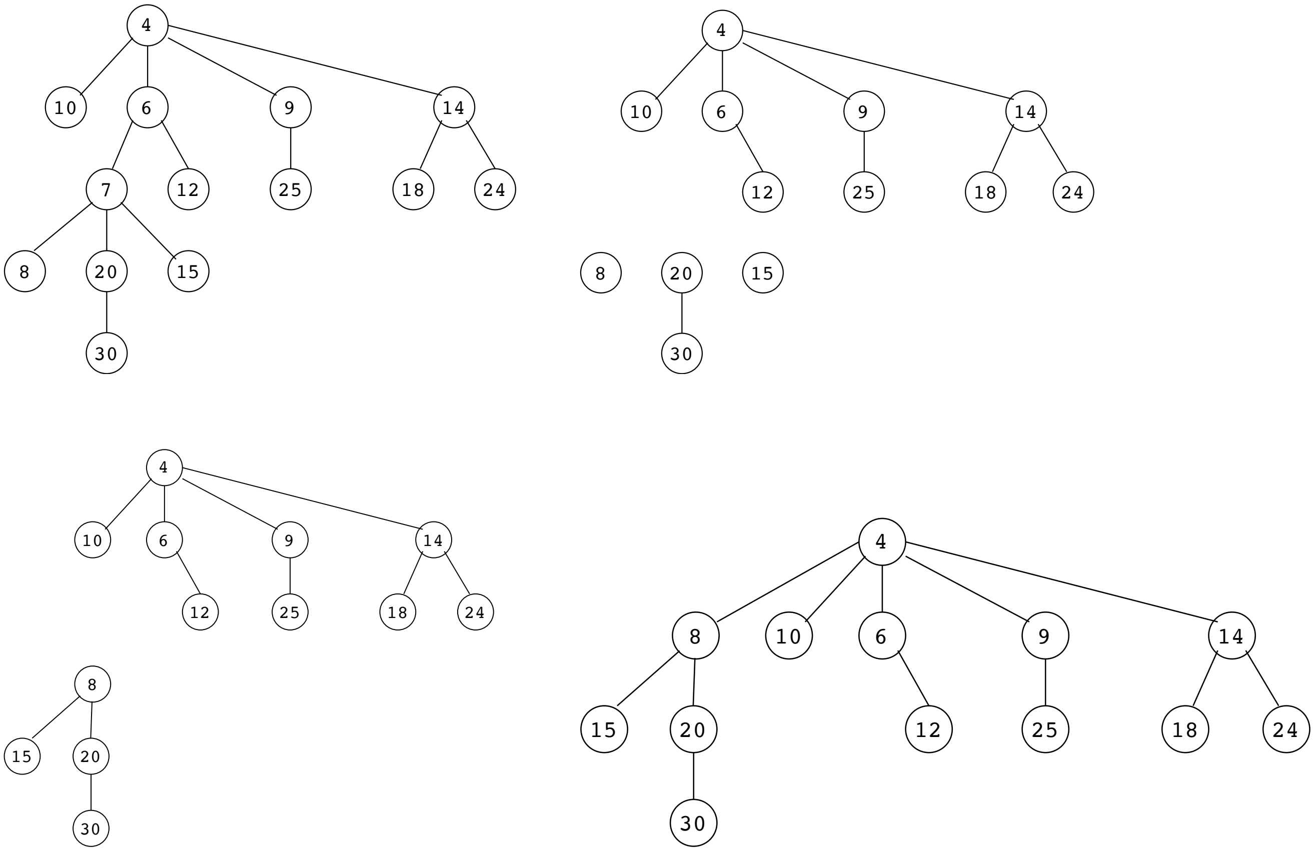
Example: Decrease 20 to 5



DELETE:

```
if item is at root, use EXTRACT-MIN
else
    cut edge to parent
    perform EXTRACT-MIN on tree of item
    UNION resulting tree with tree that was cut from
```

Example: Delete 7


4
10 6 9 14
7 12 25 18 24
8 20 15
30
4
10 6 9 14
12 25 18 24
8 20 15
30
4
10 6 9 14
12 25 18 24
8
15 20
30
4
8 10 6 9 14
15 20 12 25 18 24
30