CSE 5311 Notes 7a: Priority Queues

(Last updated 6/7/13 1:06 PM)

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.

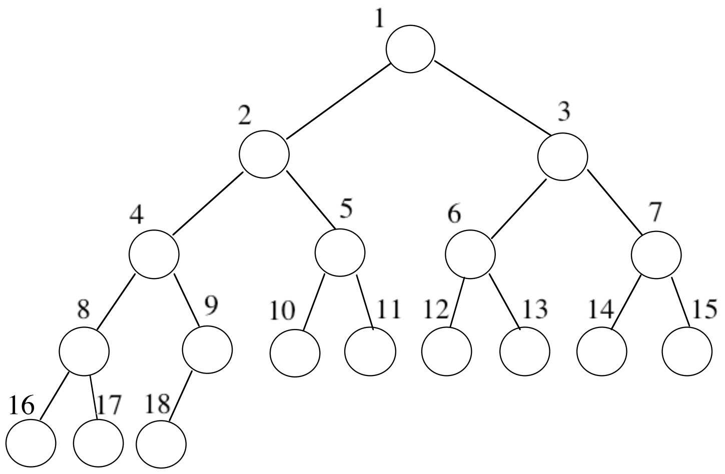
Binary heap (review)

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 $d = |E| / |V|$)

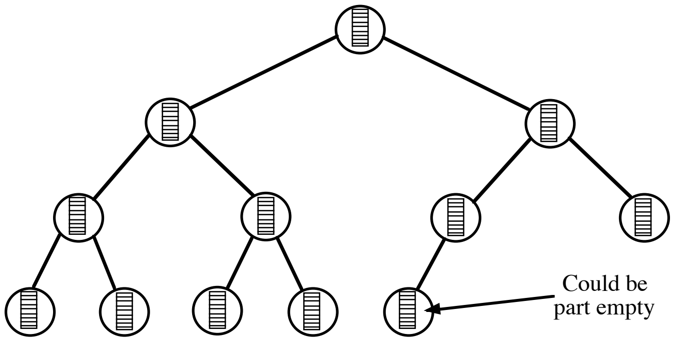
General issue - single-valued nodes vs. nodes containing table ("sack") with O(log n) values

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 *null path length*

Either subtree empty ⇒ NPL = 0

Otherwise NPL = 1 + min(left→NPL, right→NPL) (Empty tree - 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).

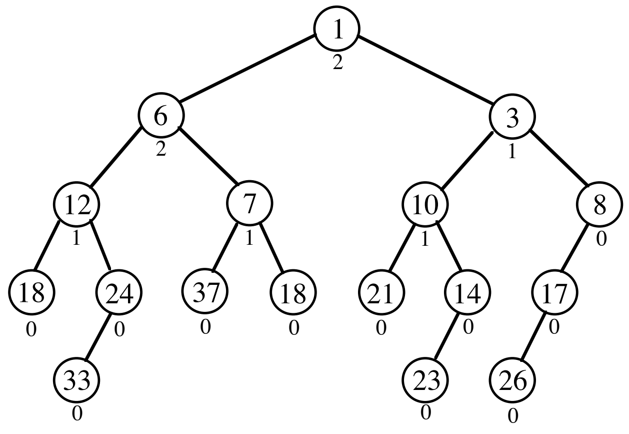
Leftist tree with r nodes on right path must have at least 2^r - 1 nodes.

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

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

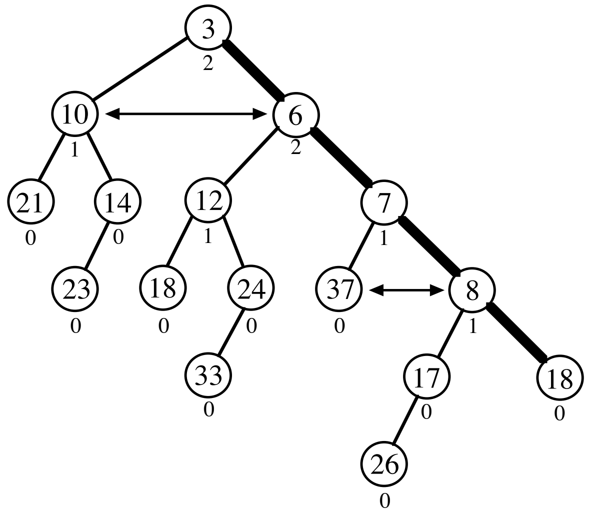
Occasionally, left and right subtrees must be swapped.

Example: EXTRACT-MIN

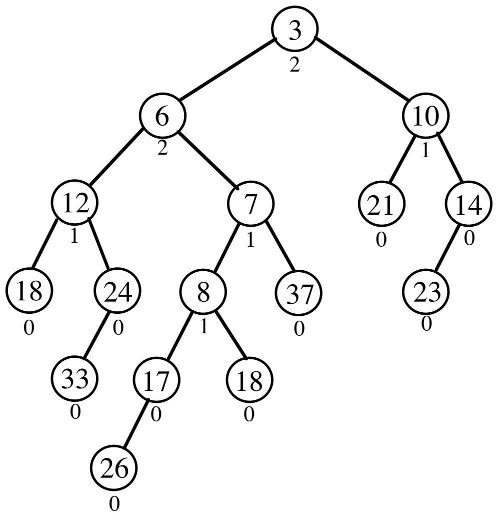


Root has item to return.

Recursively, merge right paths of two 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.

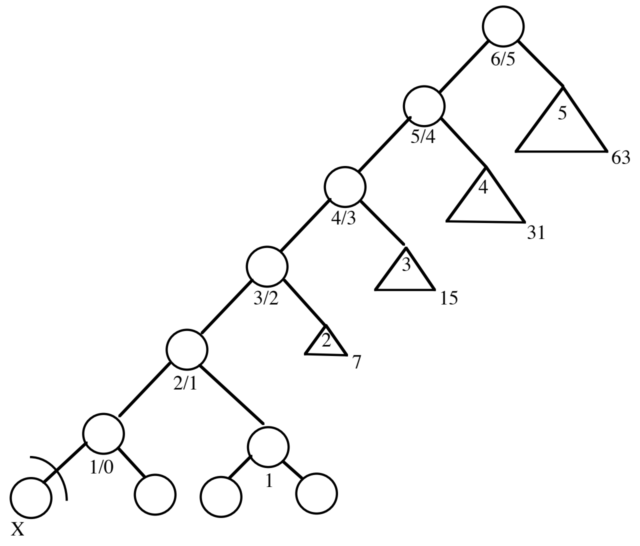
BUT, DECREASE-KEY may involve a node $\Omega(n)$ away from root, so swapping through ancestors is too slow.

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 are marked with before/after NPLs for cutting away the leftmost leaf.

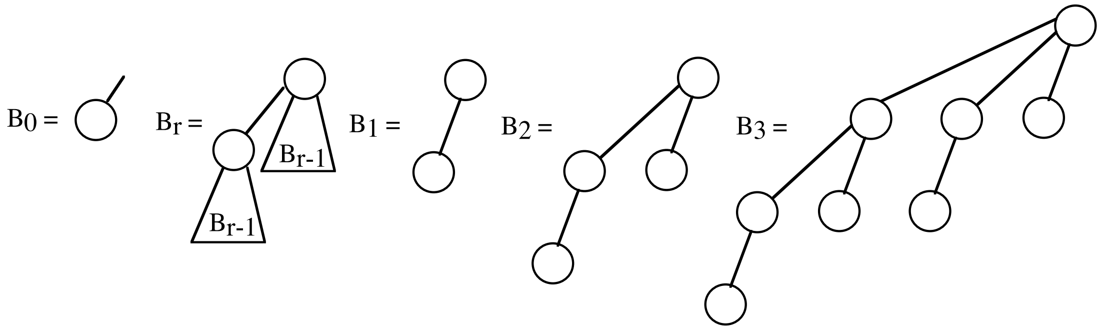


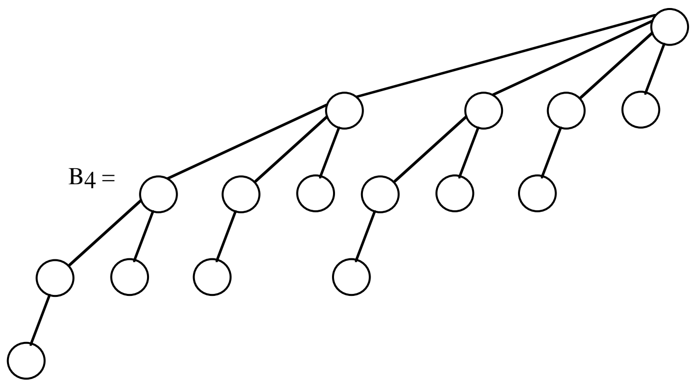
4. UNION X's subtree and modified original tree.

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.

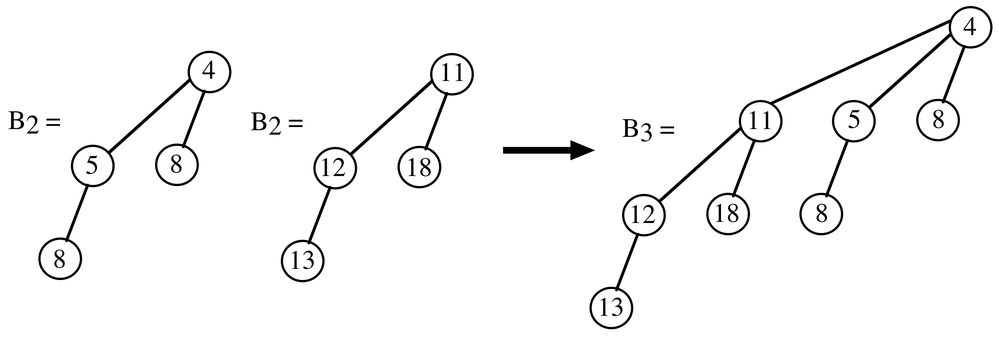
Tree roots are in a singly-linked list ordered by ascending degrees.

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

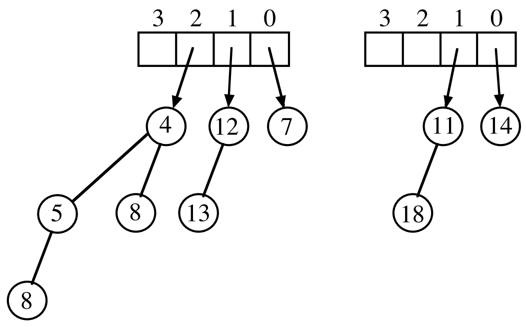
("Sack" idea can be used to reduce overhead from pointers)

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:



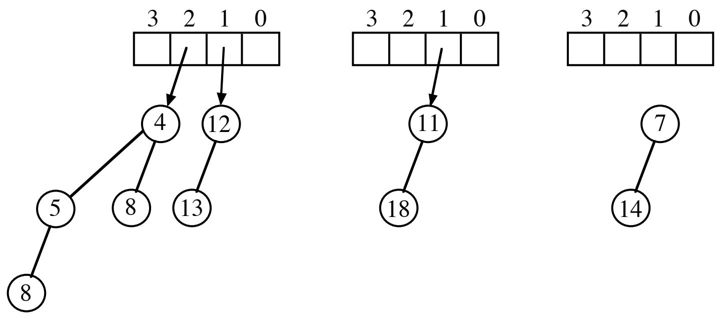
UNION of two binomial heaps



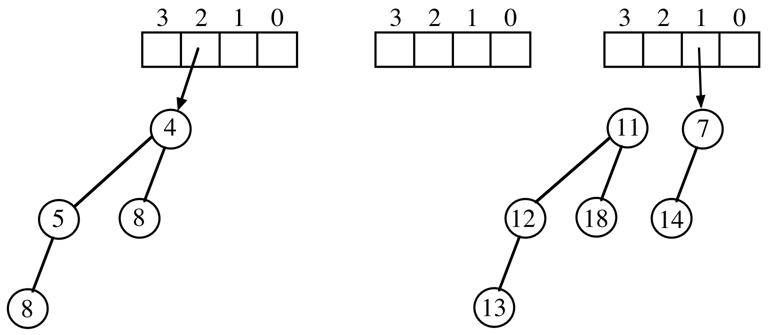
Based on binary addition:

$$0111 + 0011 = 1010$$

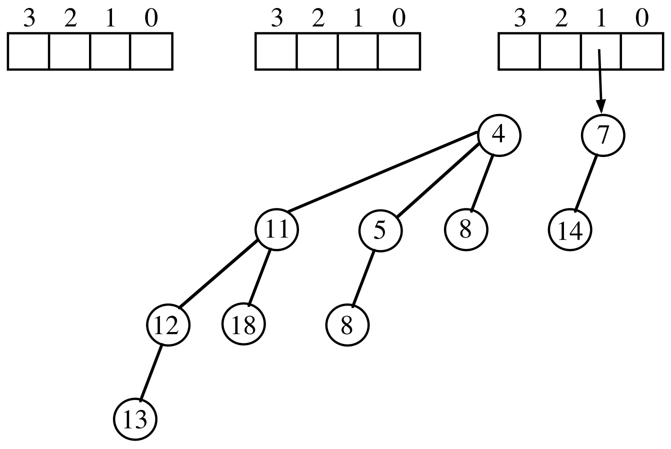
Link B_0 trees:



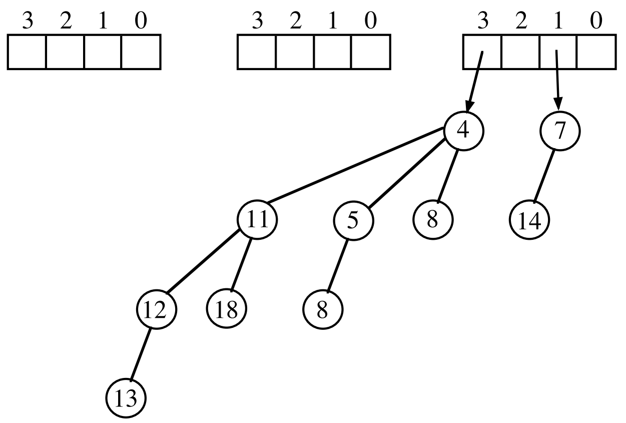
Link B_1 trees:



Link B_2 trees:



Save B_3 tree

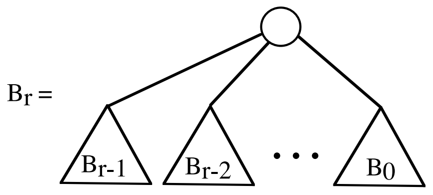


Insertion into binomial heap?

Implementing EXTRACT-MIN

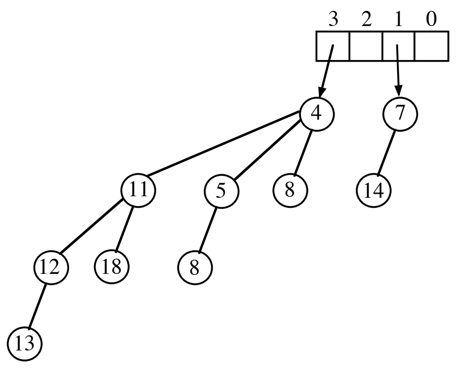
1. Scan tree roots for minimum key.

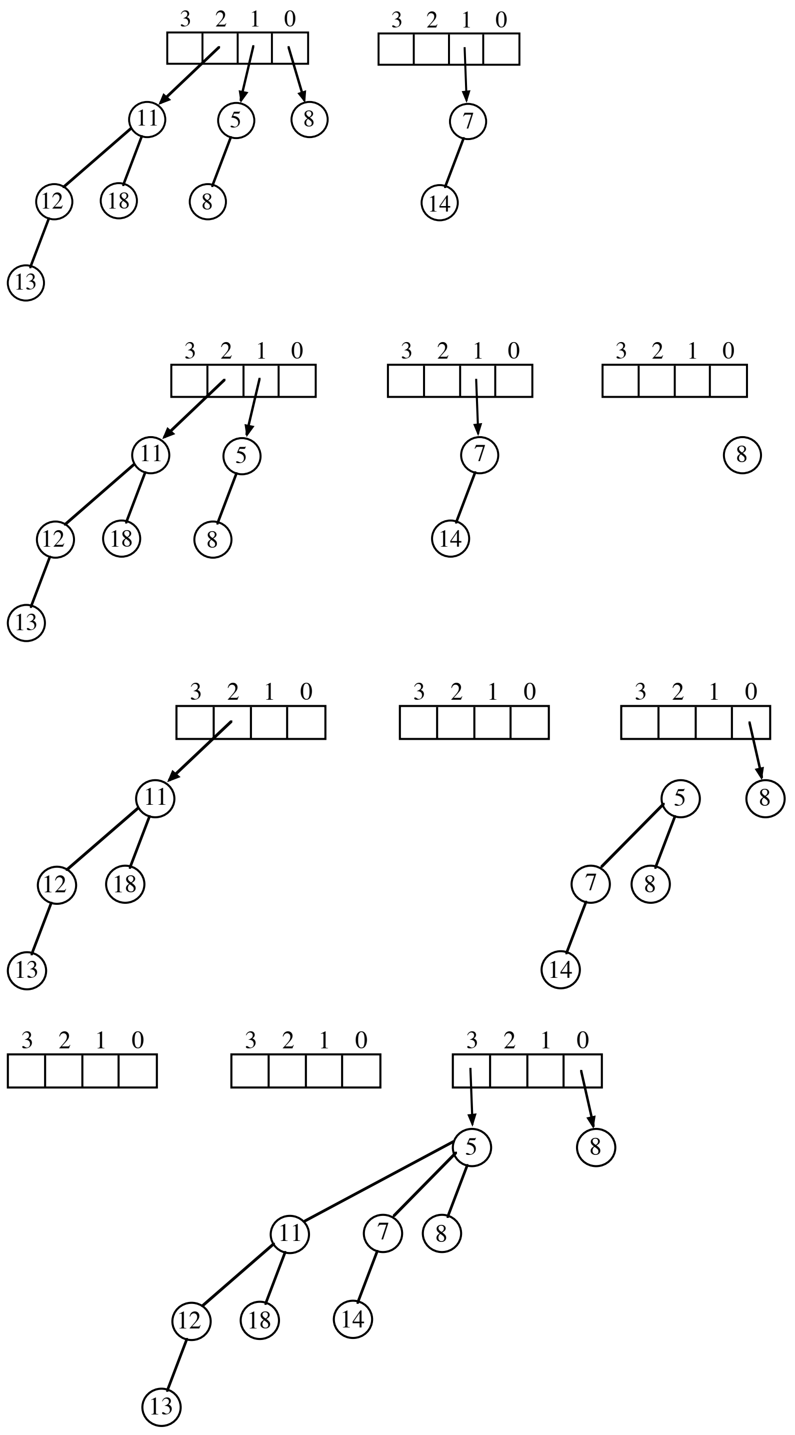
2. Decompose root of tree with minimum:



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.

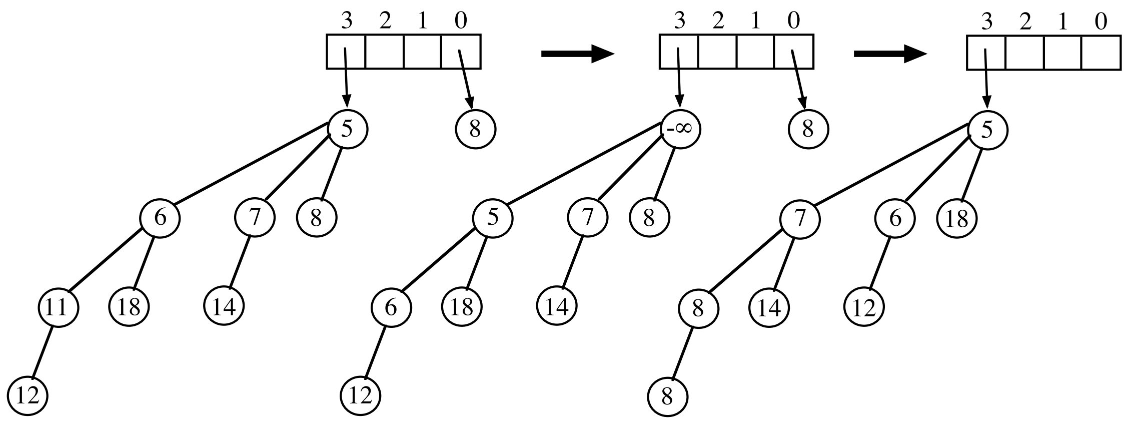
Implementing DELETE

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

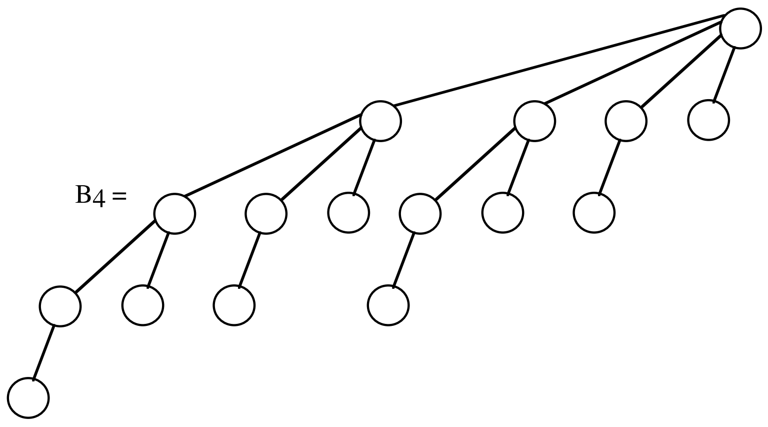
2. Use DECREASE-KEY to change priority to $-\infty$.

3. Use EXTRACT-MIN to eliminate $-\infty$.

Example: Delete 11.



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



<u>Binomial Heaps</u>	vs.	<u>Fibonacci Heaps</u>
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)

FIBONACCI HEAPS

Maintains pointer to root of tree with smallest priority.

If DELETE and DECREASE-KEY do not occur, then structure is like a binomial heap with ***multiple*** B_k trees. (Clean-up ("CONSOLIDATE") occurs for EXTRACT-MIN and DELETE)

Otherwise:

1. A (k+1)-tree will be (initially) created from two k-trees, where k is number of children ("degree") for root.

2. If a k-tree ***root*** loses a subtree, it is simply reclassified as a (k-1)-tree (degree decreases).

3. A ***non-root*** node x may lose one subtree and be "marked". If x loses another subtree, then x will be cut away from its parent.

Observation: Suppose

1. x is any node in a Fibonacci heap, and

2. c_i is the ith child attached to x (not indicated in data structure).

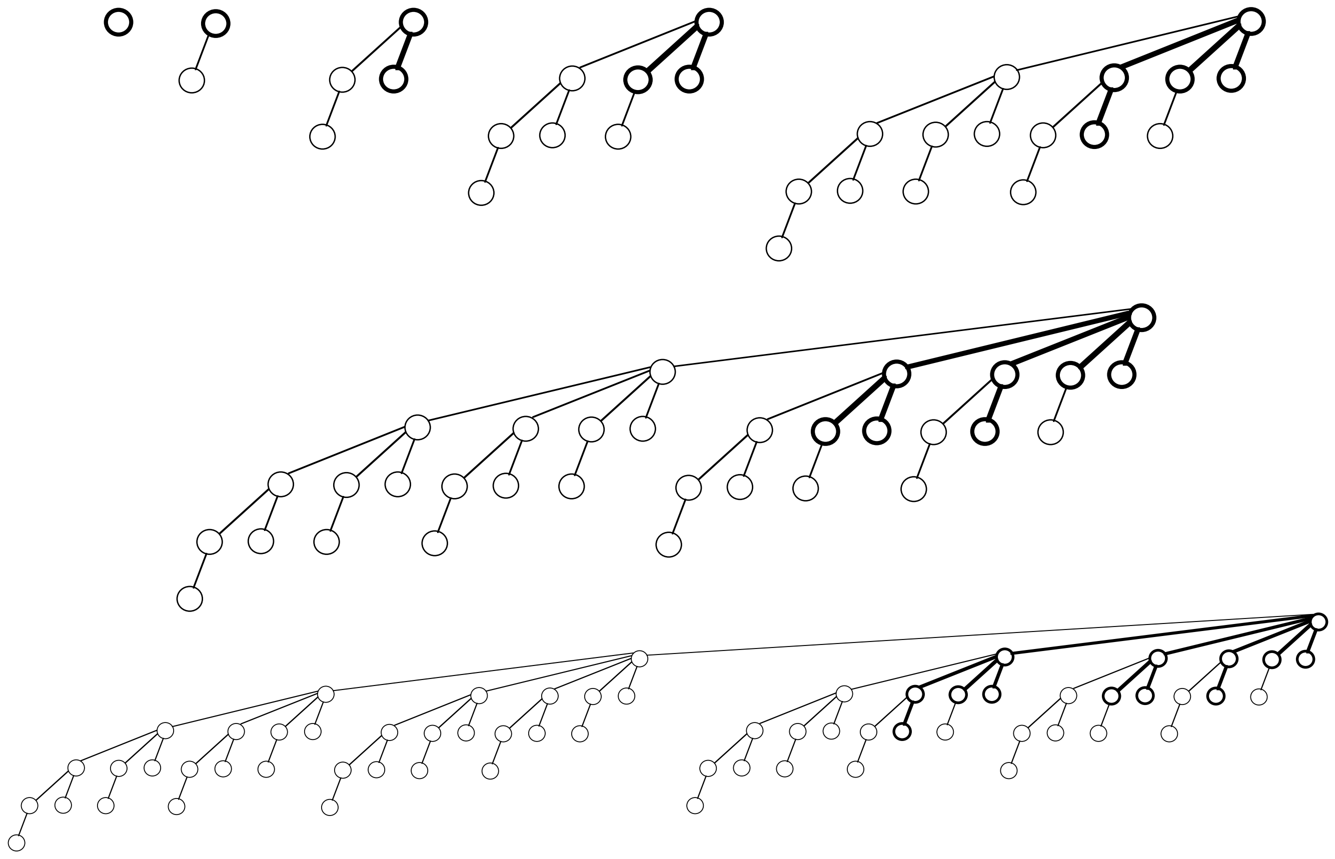
then c_i has at least i - 2 children.

Proof:

1. c_i had at least i - 1 children when attached to x.

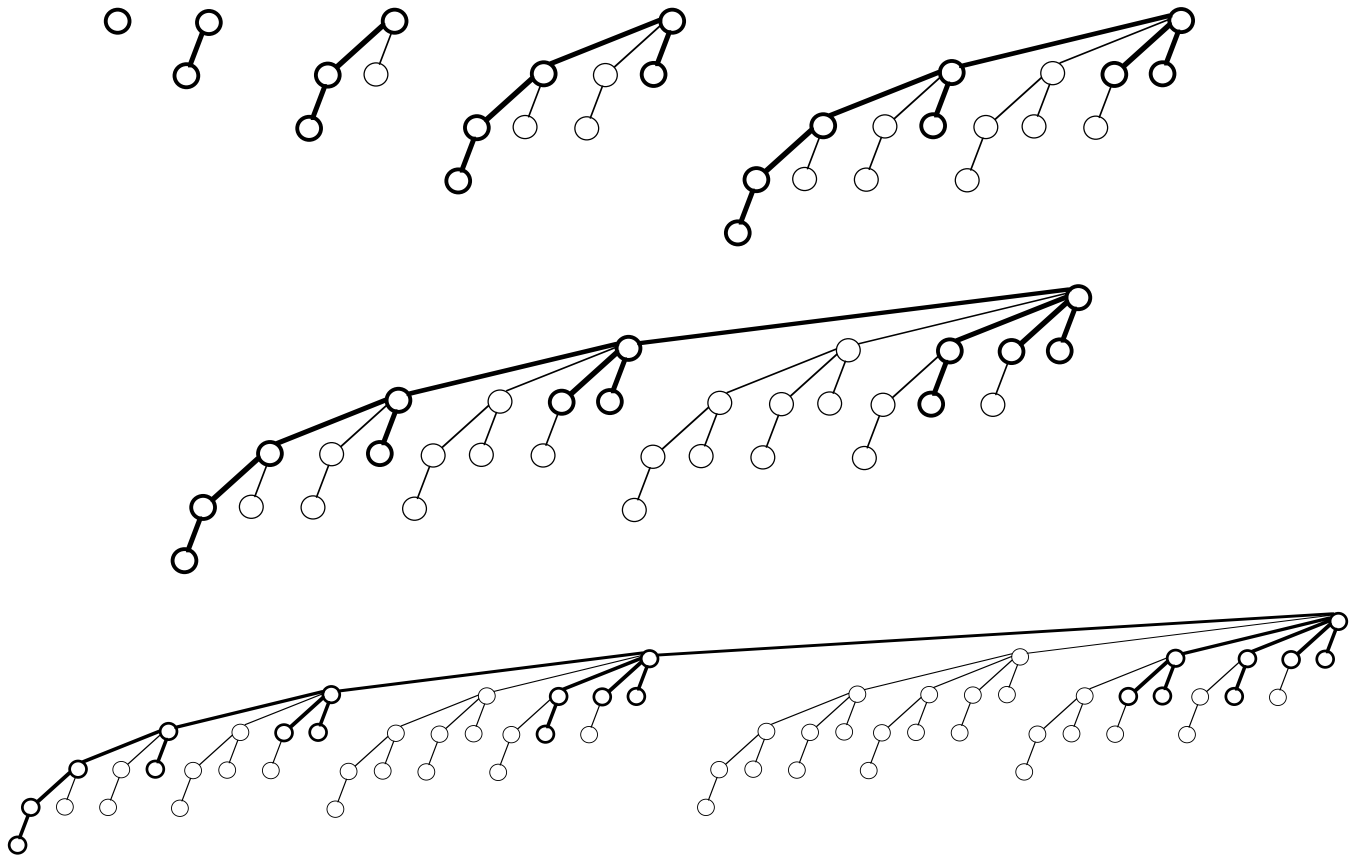
2. It could have lost 1 child (assume it is for the largest subtree).

Minimum trees for each rank by cutting away from trees built by INSERTs and an EXTRACT-MIN:



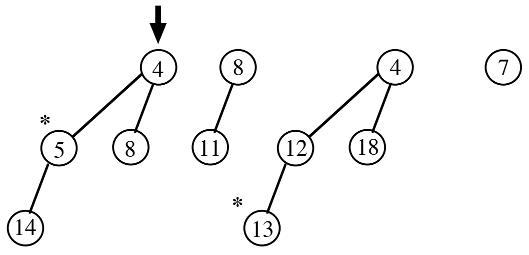
```
Most recently attached is always pruned
Original Pruned
Rank     Rank   Height Nodes
0        0      0      1
1        0      0      1
2        1      1      2
3        2      1      3
4        3      2      5
5        4      2      8
6        5      3      13
7        6      3      21
8        7      4      34
9        8      4      55
10       9      5      89
11       10     5      144
12       11     6      233
13       12     6      377
14       13     7      610
15       14     7      987
16       15     8      1597
17       16     8      2584
18       17     9      4181
19       18     9      6765
20       19     10     10946
21       20     10     17711
22       21     11     28657
23       22     11     46368
24       23     12     75025
25       24     12     121393
26       25     13     196418
27       26     13     317811
28       27     14     514229
29       28     14     832040
30       29     15     1346269
31       30     15     2178309
Ratio is 1.618034
```

Minimum trees for each height by cutting away along longest path:



```
Second most recent is sometimes pruned to preserve longest path
Original Pruned
Rank     Rank   Height Nodes
0        0      0      1
1        0      1      2
2        1      2      3
3        2      3      5
4        3      4      8
5        4      5      13
6        5      6      21
7        6      7      34
8        7      8      55
9        8      9      89
10       9      10     144
11       10     11     233
12       11     12     377
13       12     13     610
14       13     14     987
15       14     15     1597
16       15     16     2584
17       16     17     4181
18       17     18     6765
19       18     19     10946
20       19     20     17711
21       20     21     28657
22       21     22     46368
23       22     23     75025
24       23     24     121393
25       24     25     196418
26       25     26     317811
27       26     27     514229
28       27     28     832040
29       28     29     1346269
30       29     30     2178309
31       30     31     3524578
Ratio is 1.618034
```

Simple Example



$\Phi(S) = p_t \bullet \#$ of trees $+ p_m \bullet \#$ of marks

Actual cost (c_i) for each operation is stated asymptotically. Implementation-dependent constants bound the actual cost of each operation and influence the values of p_t (traditionally valued 1) and p_m (traditionally valued 2).

UNION of two Fibonacci heaps

1. Append one list of trees to another.

2. Set pointer to new minimum key.

O(1) actual and amortized. (No change in Φ.)

INSERT

1. Create single node Fibonacci heap.

2. UNION

O(1) actual and amortized. (Φ goes up by p_t.)

EXTRACT-MIN

1. Remove minimum node (a root).

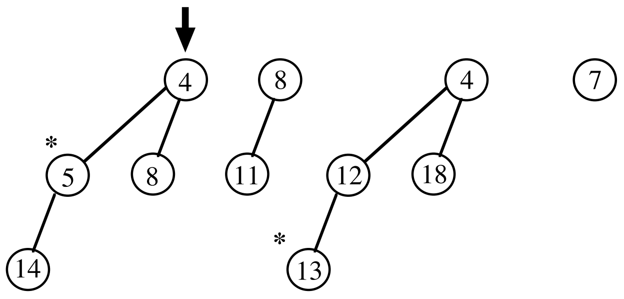
2. Append subtrees to list.

3. Much like binomial queue, use "accumulator" of pointers to CONSOLIDATE so that there is no more than one tree whose root has *k* children. (Root list is not ordered.) Initialization cost per accumulator entry is *d* (traditionally valued 1).

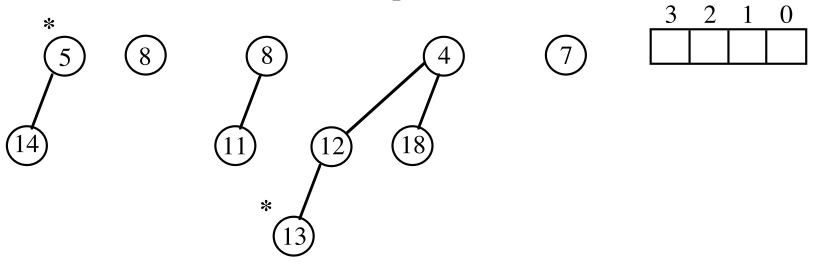
   Like binomial tree, two *k*-trees combine to give a (*k*+1)-tree. Combining cost is *e* (traditionally valued 1). Roots that become children will be set unmarked.

4. Must determine new minimum root.

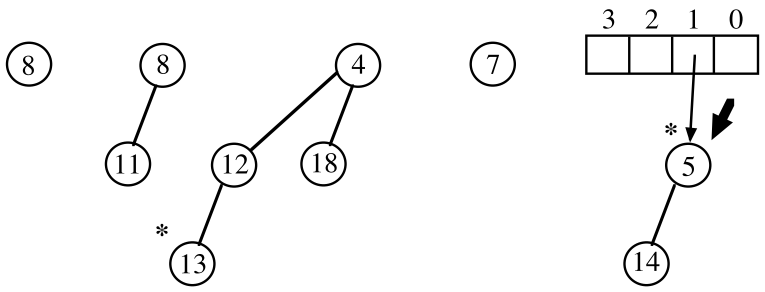
Actual cost is $d \bullet \log n + e \bullet \#$of trees = O($\log n + \#$ of trees). O($\log n$) amortized.



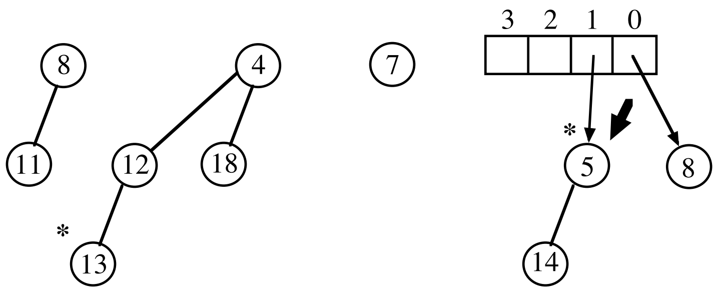
Remove minimum and decompose:



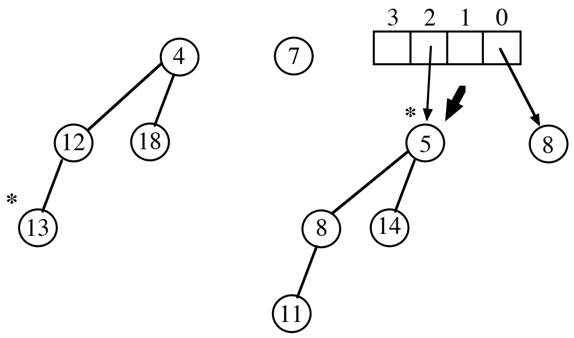
Process first tree:



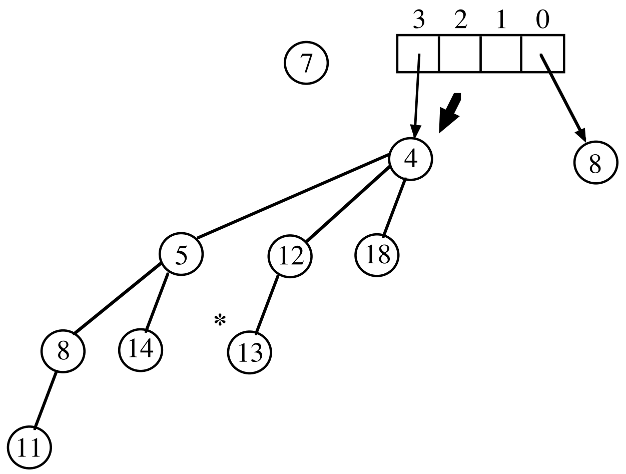
Process next tree:



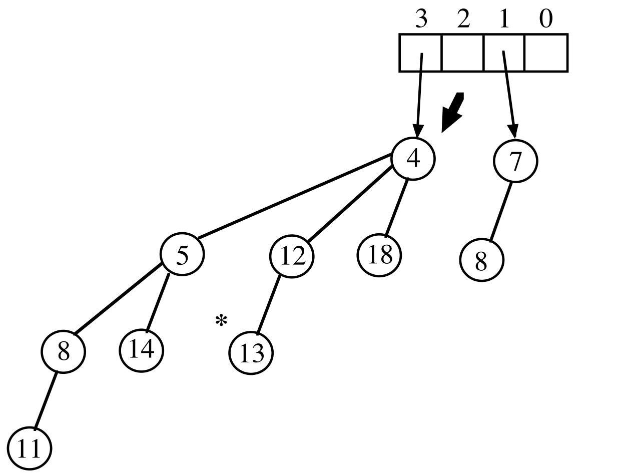
Process next tree:



Process next tree:



Final tree:



$\hat{C}_i = C_i + \Phi(S_i) - \Phi(S_{i-1})$

$C_i = d \bullet \log n + e \bullet (\text{initial \# of trees - 1 + \# of derived subtrees})$

$= \text{O(initial \# of trees} + D(n))$

$D(n) =$ max # of children for any node in structure, including roots

(worst case is one tree in structure)

$\Phi(S_{i-1}) = p_t \bullet \text{initial \# of trees} + p_m \bullet \text{ initial \# of marked nodes}$

$\Phi(S_i) \leq p_t(D(n)+1) + p_m \bullet \text{ initial \# of marked nodes}$

$(D(n)+1$ is due to nodes with 0 ... $D(n)$ children)

$\hat{C}_i \leq d \bullet \log n + e \bullet (\text{initial \# of trees - 1 + \# of derived subtrees}) + p_t(D(n)+1) - p_t \bullet \text{initial \# of trees}$

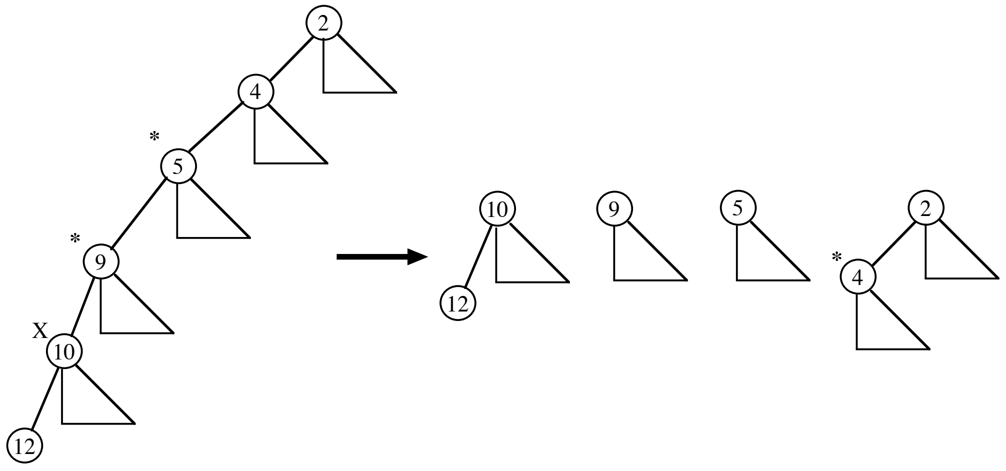
$= \text{O}(D(n))$ if $p_t \geq e$

Skim section 19.4, $D(n)$ is logarithmic in n

DECREASE-KEY and DELETE ⇒ Lose binomial heap properties

Based on *cascading cut* at X (that has a parent):

```
Clear mark on X                              // X is not necessarily marked
P := parent(X)
Break (cut) link from X to P
X := P
P := parent(X)
while P ≠ nil
        if X is marked
                Break (cut) link from X to P
                Clear mark on X              // X definitely is marked
                X := P
                P := parent(X)
        else
                Set mark on X                // X cannot be the root
                P := nil
```



DECREASE-KEY

Not based on swaps (as done for binary and binomial heaps), but similar to leftist heaps

1. Decrease key value.

2. If node has parent and key < parent's key
   Perform cascading cut at key's node

3. Check if key is lowest in structure

Actual cost is O(log *n*) based on the number of cuts. O(1) amortized

DELETE

1. DECREASE-KEY value to -∞.

2. EXTRACT-MIN

EXTRACT-MIN dominates actual cost. $O(\log n)$ amortized.

Amortized Cost of Cascading Cut

Suppose c is the number of cuts.

$c_i = fc = O(c)$ f is the actual cost of cutting a node (traditionally valued 1)

$\Phi(S_i) \le p_t \bullet (\#\text{ trees } + c) + p_m(\#\text{marked nodes} - c + 2)$

$\Phi(S_{i-1}) = p_t \bullet \#\text{ trees } + p_m \bullet \#\text{marked nodes}$

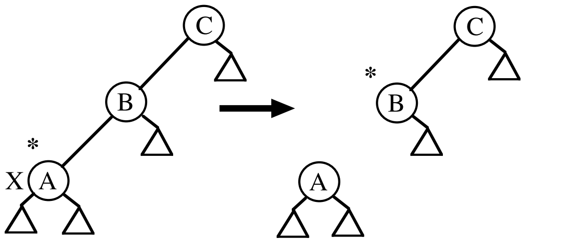
$\hat{c}_i = fc + p_t c + p_m(-c + 2) = c(f + p_t - p_m) + 2p_m = O(1)$ if $p_m \ge f + p_t$

The -c + 2 upper bound on the change in marked nodes is based on the following observations about the c cuts that occur:

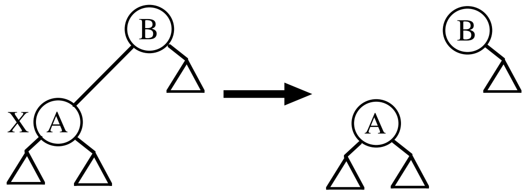
1. The first cut loses a mark only when the initial node X is marked.

2. Cuts 2 through c - 1 must lose a mark.

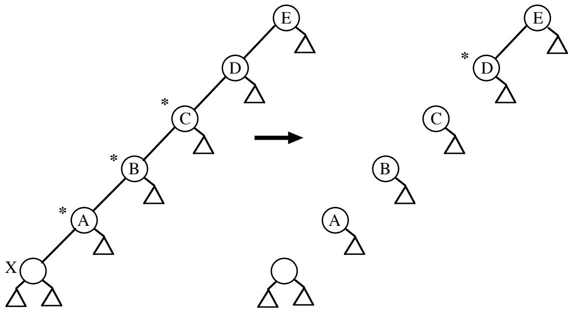
3. The last cut loses a mark only when the parent node is the root.



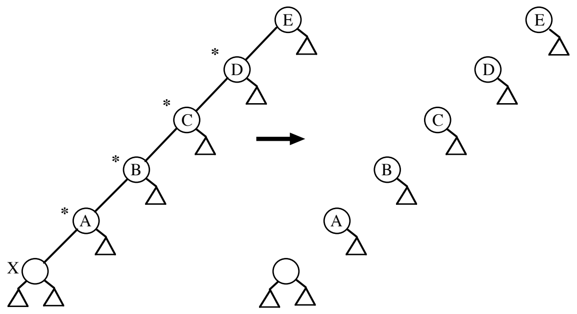
c = 1, change in marked nodes is -c + 2 ≥ 0



c = 1, change in marked nodes is -c + 2 ≥ 0



c = 4, change in marked nodes is -c + 2 = -2



c = 5, change in marked nodes is -c + 2 ≥ -4

Degenerate Fibonacci Heap

MAKE-FIB-HEAP
FIB-HEAP-INSERT(20)
FIB-HEAP-INSERT(19)
FIB-HEAP-INSERT(18)
EXTRACT-MIN
FIB-HEAP-INSERT(17)
FIB-HEAP-INSERT(16)
FIB-HEAP-INSERT(15)
EXTRACT-MIN
DELETE(17)
FIB-HEAP-INSERT(14)
FIB-HEAP-INSERT(13)
FIB-HEAP-INSERT(12)
EXTRACT-MIN
DELETE(14)
FIB-HEAP-INSERT(11)
FIB-HEAP-INSERT(10)
FIB-HEAP-INSERT(9)
EXTRACT-MIN
DELETE(11)