## Leftist Trees

Linked binary tree.

Can do everything a heap can do and in the same asymptotic complexity.

- insert
- remove min (or max)
- initialize

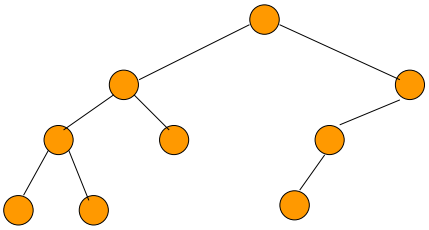
Can meld two leftist tree priority queues in O(log n) time.

## Extended Binary Trees

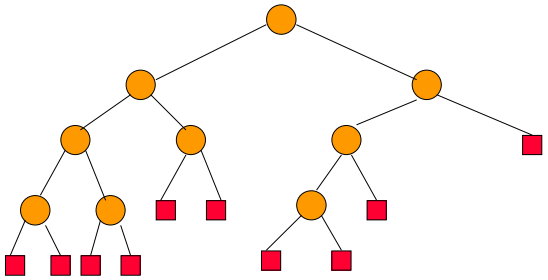
Start with any binary tree and add an external node wherever there is an empty subtree.

Result is an extended binary tree.

## A Binary Tree



## An Extended Binary Tree

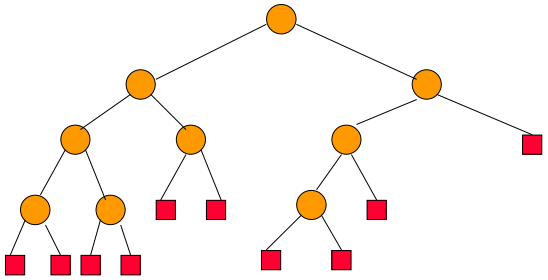


number of external nodes is n+1

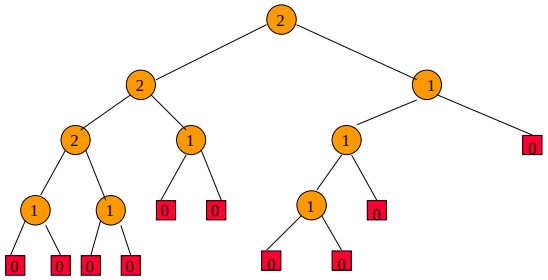
## The Function s()

For any node x in an extended binary tree, let s(x) be the length of a shortest path from x to an external node in the subtree rooted at x.

## s() Values Example



## s() Values Example




## Properties Of s()

If x is an external node, then s(x) = 0.

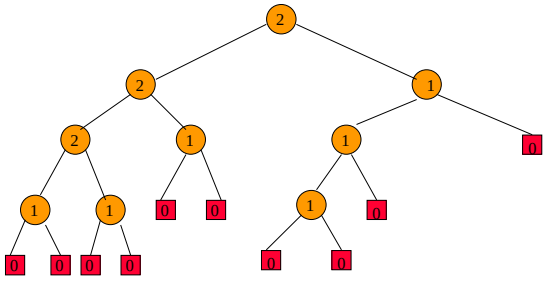
Otherwise,

s(x) = min {s(leftChild(x)),
s(rightChild(x))} + 1

## Height Biased Leftist Trees

A binary tree is a (height biased) leftist tree iff for every internal node x, s(leftChild(x)) >= s(rightChild(x))

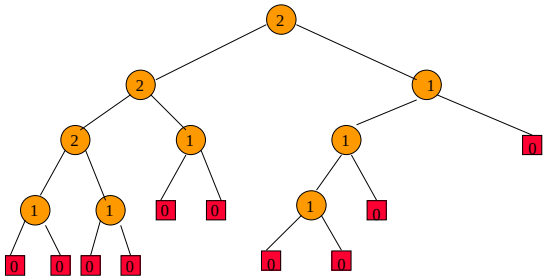
## A Leftist Tree




## Leftist Trees – Property 1

In a leftist tree, the rightmost path is a shortest root to external node path and the length of this path is s(root).

## A Leftist Tree




Length of rightmost path is 2.

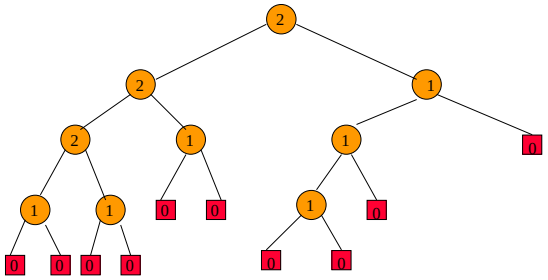
## Leftist Trees—Property 2

The number of internal nodes is at least

$2^{s(root)} - 1$

Because levels 1 through s(root) have no external nodes.

## A Leftist Tree




Levels 1 and 2 have no external nodes.

## Leftist Trees—Property 3

Length of rightmost path is O(log n), where n is the number of (internal) nodes in a leftist tree.

Property 2 =>

- $n >= 2^{s(root)} - 1 \Rightarrow s(root) <= \log_2(n+1)$

Property 1 => length of rightmost path is s(root).

## Leftist Trees As Priority Queues

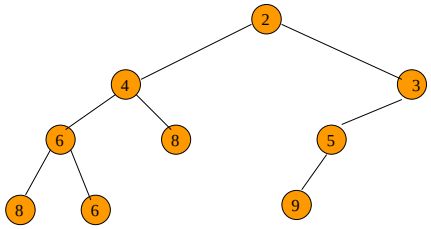
Min leftist tree … leftist tree that is a min tree.

Used as a min priority queue.

Max leftist tree … leftist tree that is a max tree.

Used as a max priority queue.

## A Min Leftist Tree




## Some Min Leftist Tree Operations

put

removeMin()

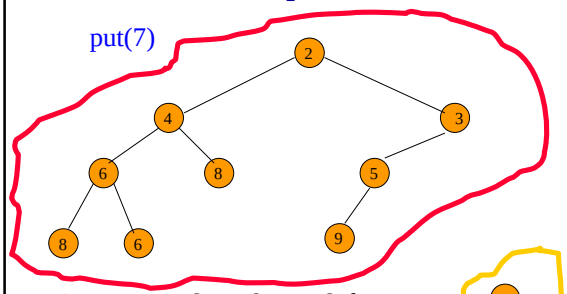
meld()

initialize()

put() and removeMin() use meld().

## Put Operation

put(7)

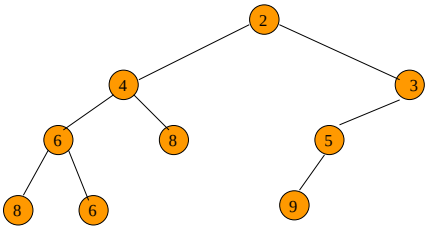




Create a single node min leftist tree.

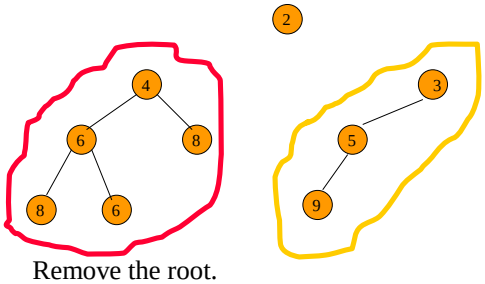
Meld the two min leftist trees.

## Remove Min




Remove the root.

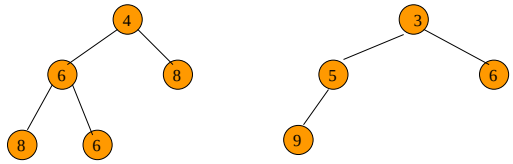
## Remove Min




Remove the root.

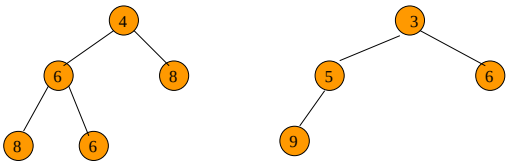
Meld the two subtrees.

## Meld Two Min Leftist Trees



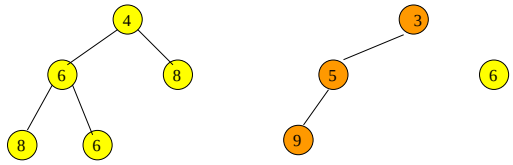

Traverse only the rightmost paths so as to get logarithmic performance.

## Meld Two Min Leftist Trees



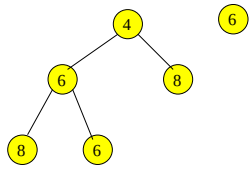

Meld right subtree of tree with smaller root and all of other tree.

## Meld Two Min Leftist Trees




Meld right subtree of tree with smaller root and all of other tree.

## Meld Two Min Leftist Trees




Meld right subtree of tree with smaller root and all of other tree.

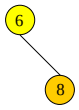
## Meld Two Min Leftist Trees


Meld right subtree of tree with smaller root and all of other tree.

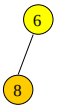
Right subtree of 6 is empty. So, result of melding right subtree of tree with smaller root and other tree is the other tree.

## Meld Two Min Leftist Trees

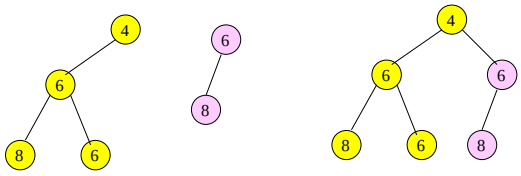

Make melded subtree right subtree of smaller root.




Swap left and right subtree if s(left) < s(right).



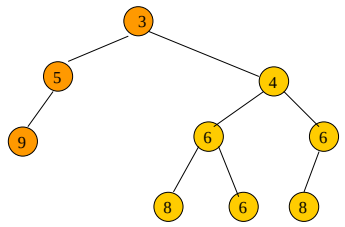

## Meld Two Min Leftist Trees




Make melded subtree right subtree of smaller root.

Swap left and right subtree if s(left) < s(right).

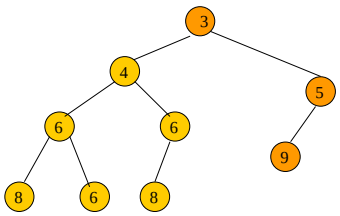
## Meld Two Min Leftist Trees




Make melded subtree right subtree of smaller root.

Swap left and right subtree if s(left) < s(right).

## Meld Two Min Leftist Trees

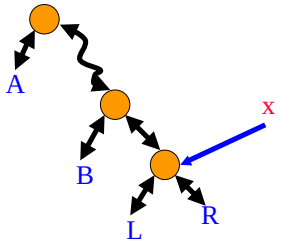



## Initializing In O(n) Time

- Create n single-node min leftist trees and place them in a FIFO queue.
- Repeatedly remove two min leftist trees from the FIFO queue, meld them, and put the resulting min leftist tree into the FIFO queue.
- The process terminates when only 1 min leftist tree remains in the FIFO queue.
- Analysis is the same as for heap initialization.

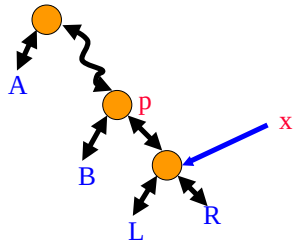
## Arbitrary Remove

Remove element in node pointed at by x.




x = root => remove min.

## Arbitrary Remove, x != root




Make L right subtree of p.

Adjust s and leftist property on path from p to root.

Meld with R.

## Skew Heap

- Similar to leftist tree
- No s() values stored
- Swap left and right subtrees of all nodes on rightmost path rather than just when s(l(x)) < s(r(x))
- Amortized complexity of each operation is O(log n)