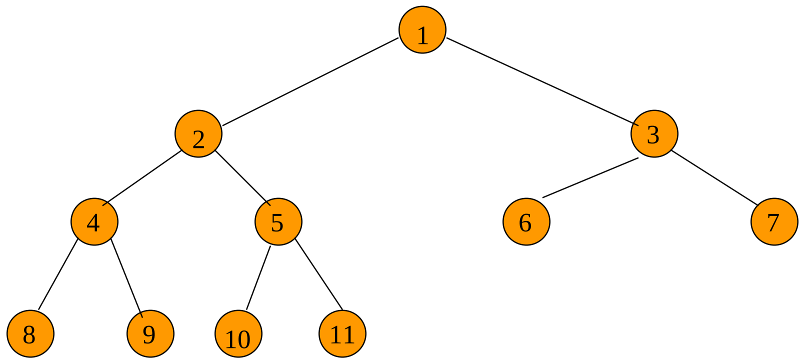
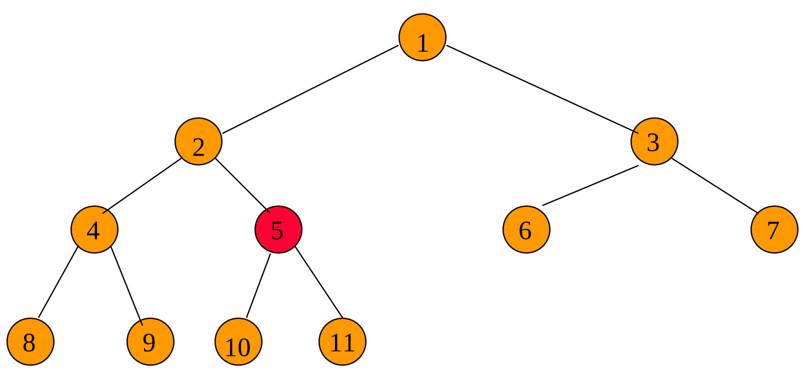
Initializing A Max Heap



input array = [-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

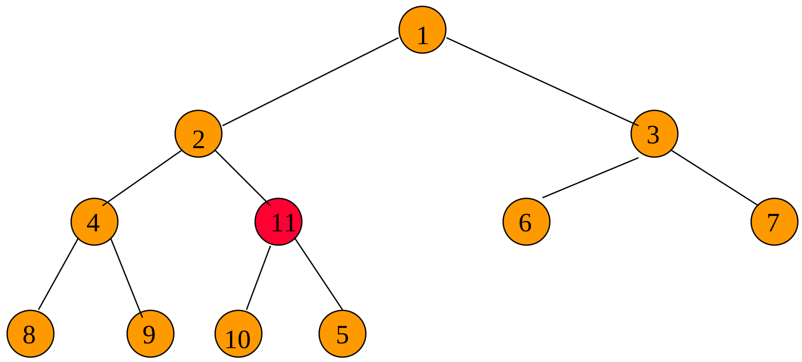
Initializing A Max Heap



Start at rightmost array position that has a child.

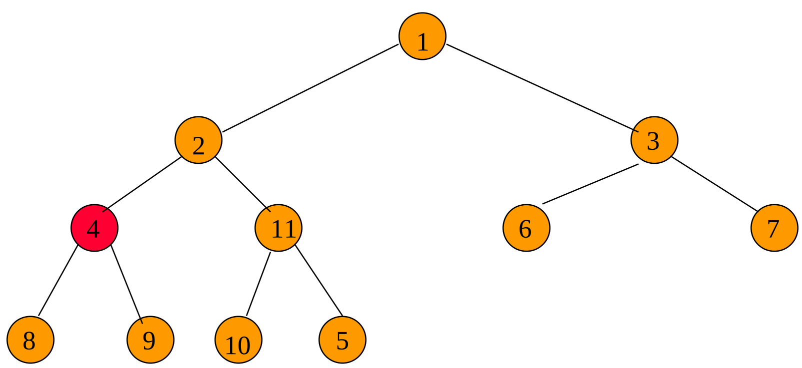
Index is n/2.

Initializing A Max Heap

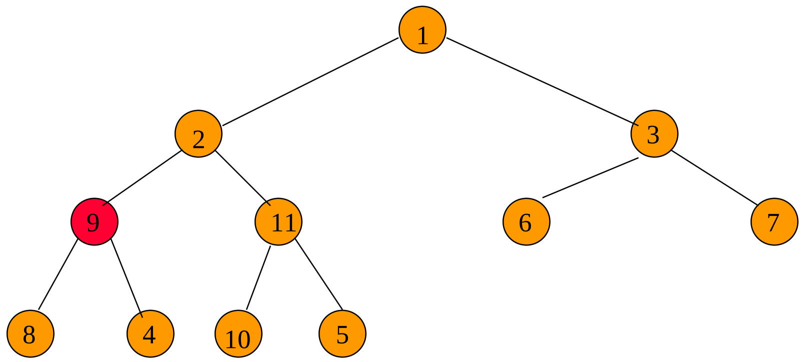


Move to next lower array position.

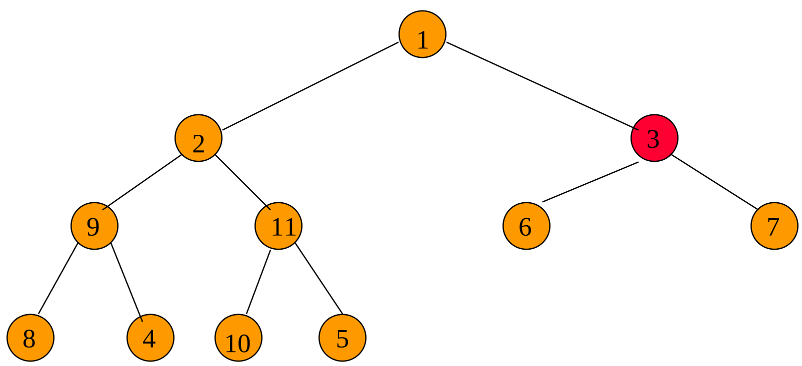
Initializing A Max Heap



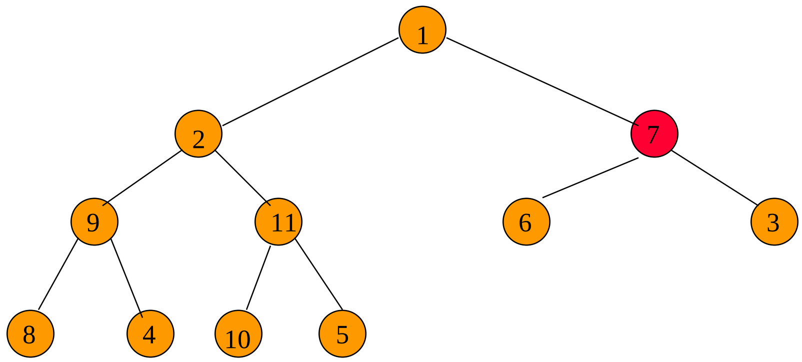
Initializing A Max Heap



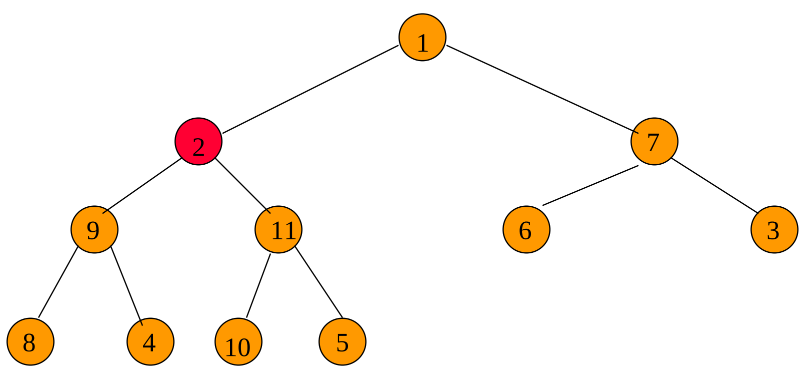
Initializing A Max Heap



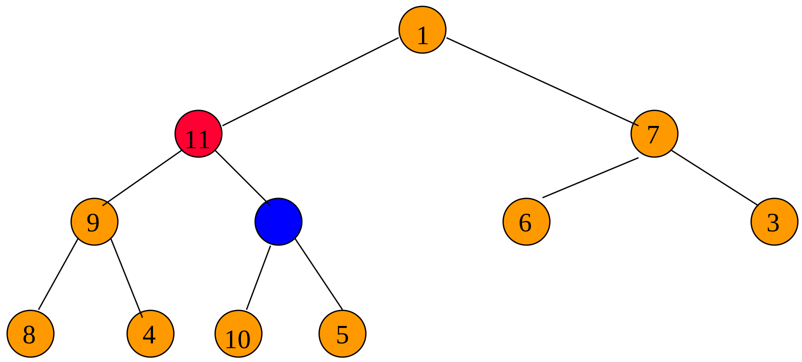
Initializing A Max Heap



Initializing A Max Heap

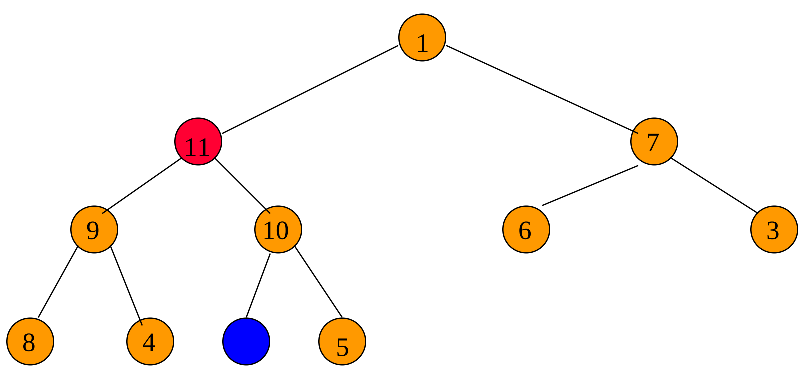


Initializing A Max Heap



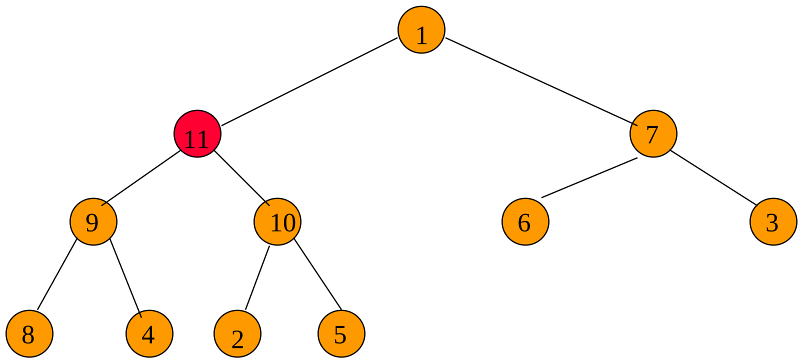
Find a home for 2.

Initializing A Max Heap



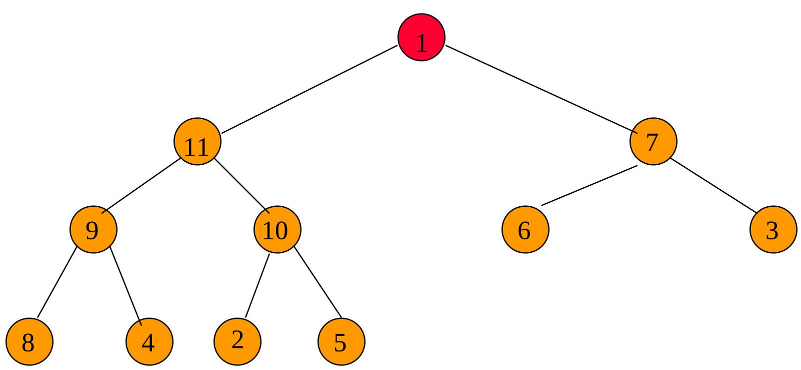
Find a home for 2.

Initializing A Max Heap



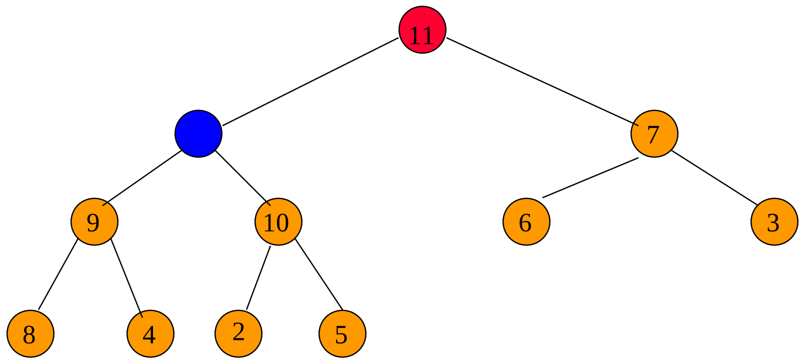
Done, move to next lower array position.

Initializing A Max Heap



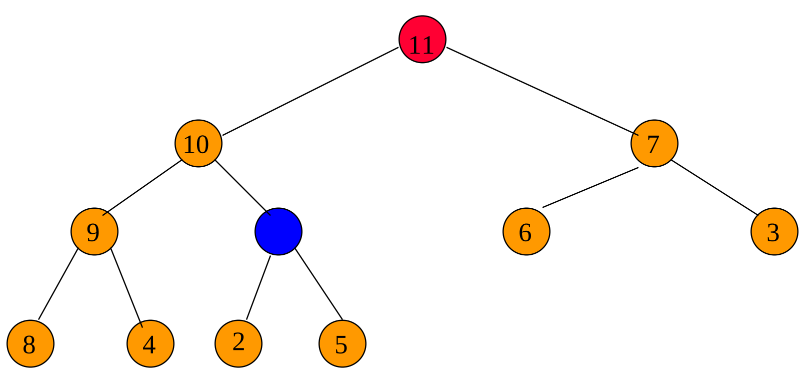
Find home for 1.

Initializing A Max Heap



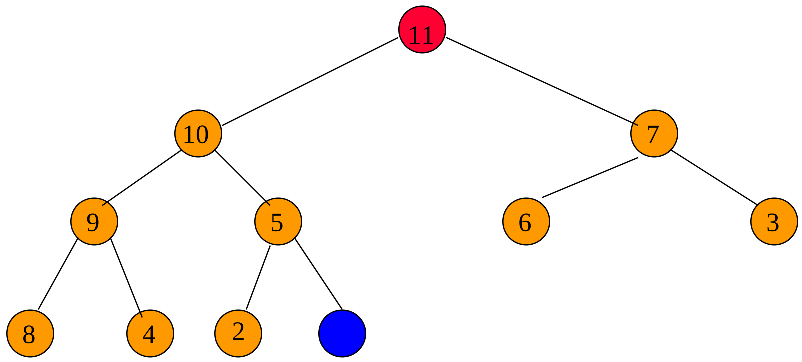
Find home for 1.

Initializing A Max Heap



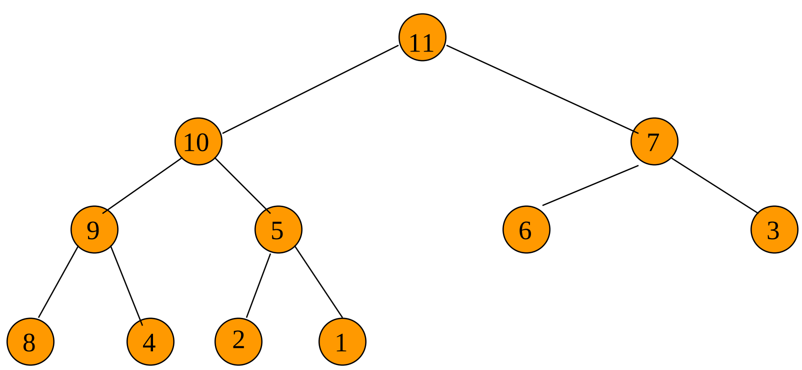
Find home for 1.

Initializing A Max Heap



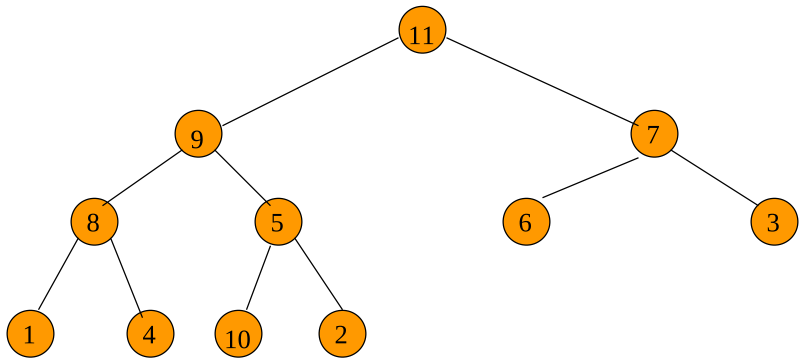
Find home for 1.

Initializing A Max Heap



Done.

Time Complexity



Height of heap = h.

Number of subtrees with root at level j is <= 2^{j-1}.

Time for each subtree is O(h-j+1).

Complexity

Time for level j subtrees is <= $2^{j-1}(h-j+1) = t(j)$.

Total time is t(1) + t(2) + … + t(h-1) = O(n).

Leftist Trees

Linked binary tree.

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

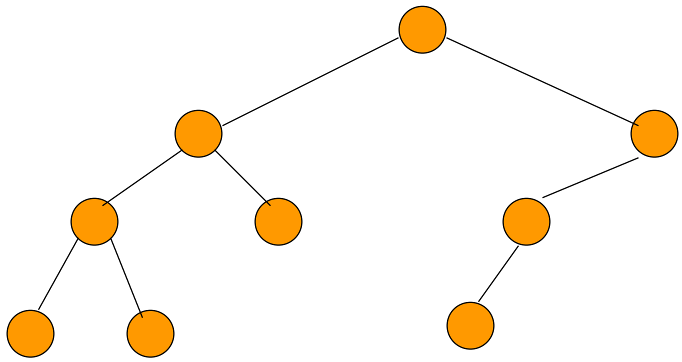
Can meld two leftist tree priority queues in O(log n) time.

Extended Binary Trees

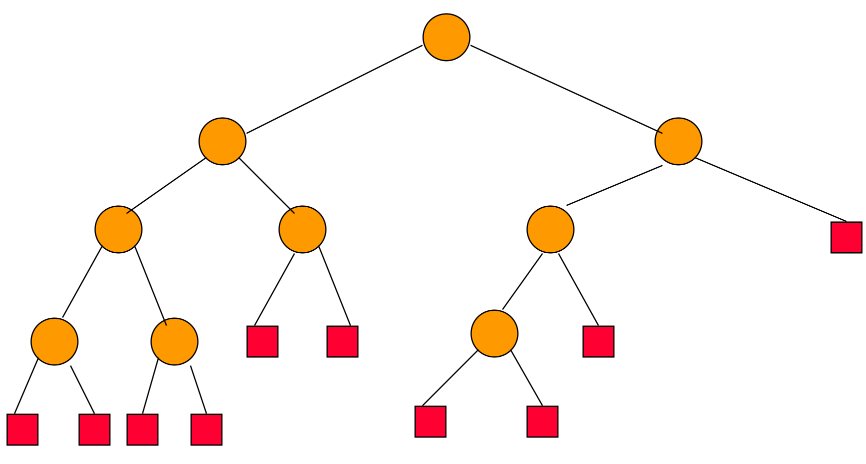
Start with any binary tree and add an external node wherever there is an empty subtree.

Result is an extended binary tree.

A Binary Tree



An Extended Binary Tree

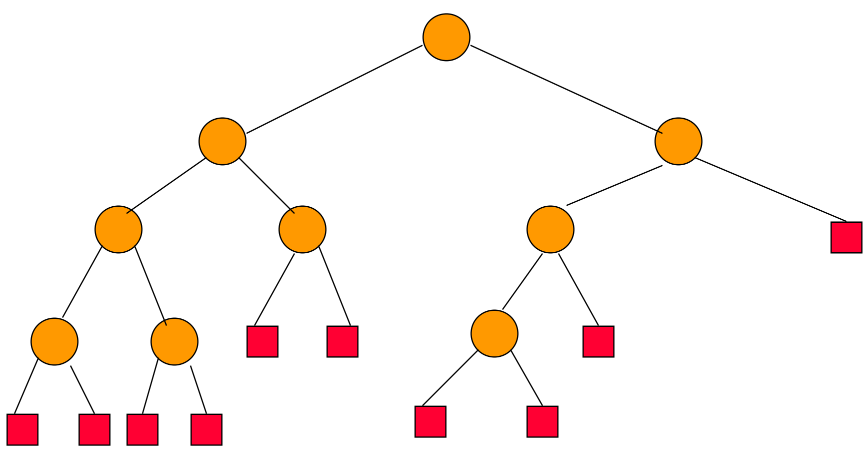


number of external nodes is n+1

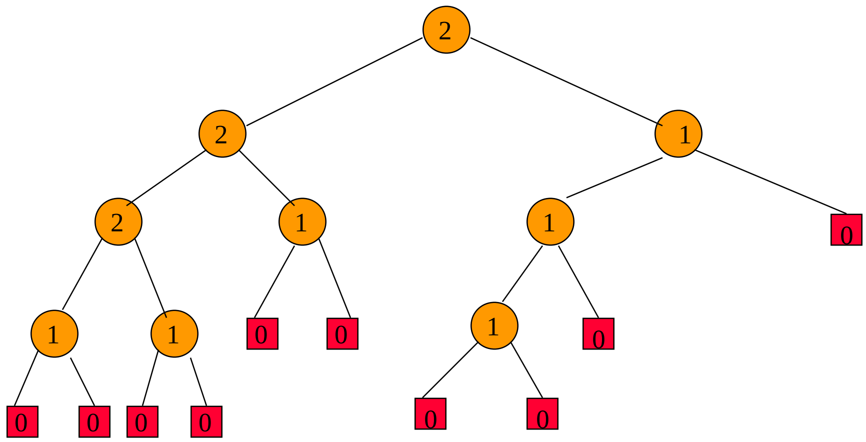
The Function s()

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.

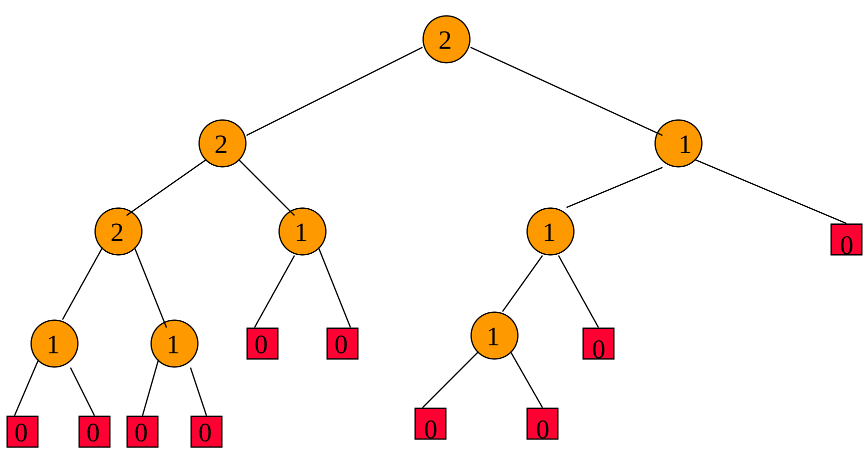
Otherwise,

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))

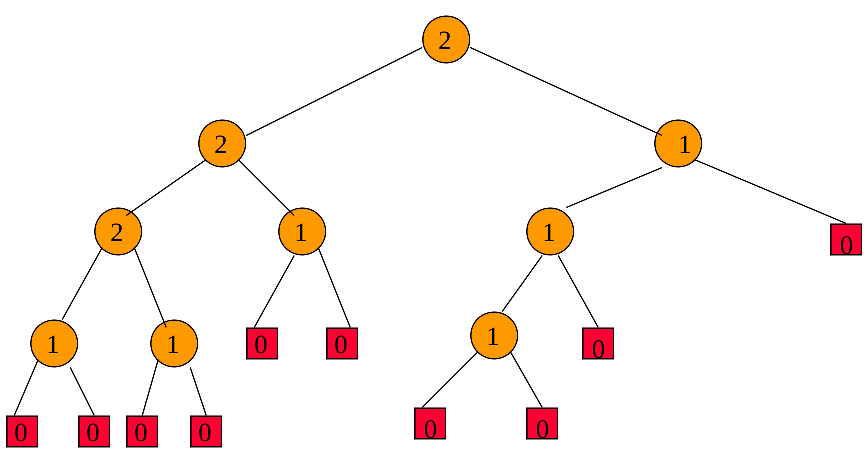
A Leftist Tree



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

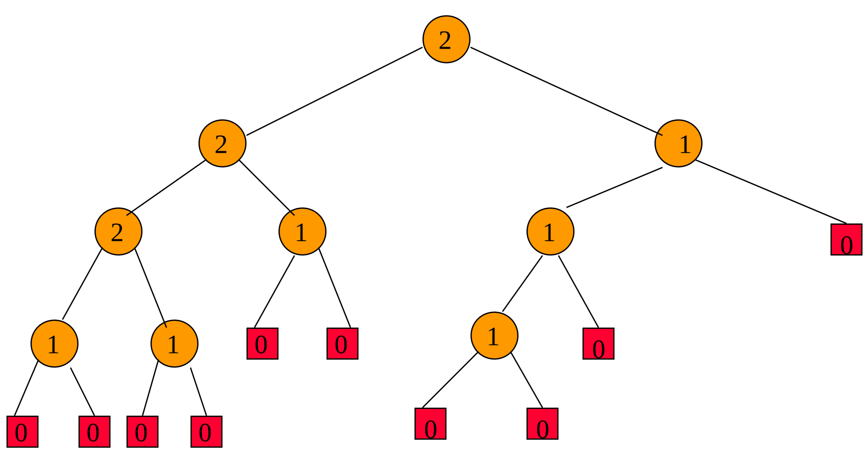
The number of internal nodes is at least

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

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

So, s(root) <= log(n+1)

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 nodes in a leftist tree.

Follows from Properties 1 and 2.

Leftist Trees As Priority Queues

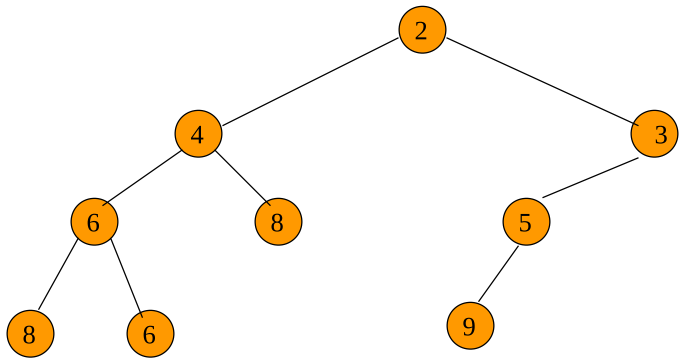
Min leftist tree … leftist tree that is a min tree.

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()

remove()

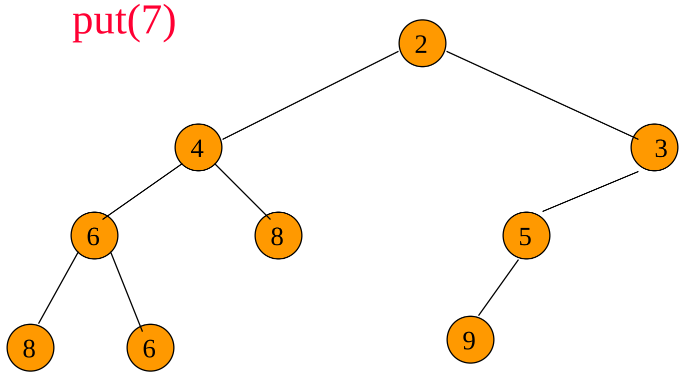
meld()

initialize()

put() and remove() use meld().

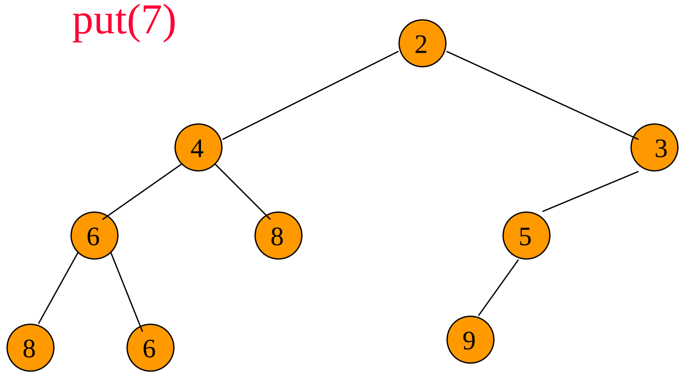
Put Operation

put(7)



Put Operation

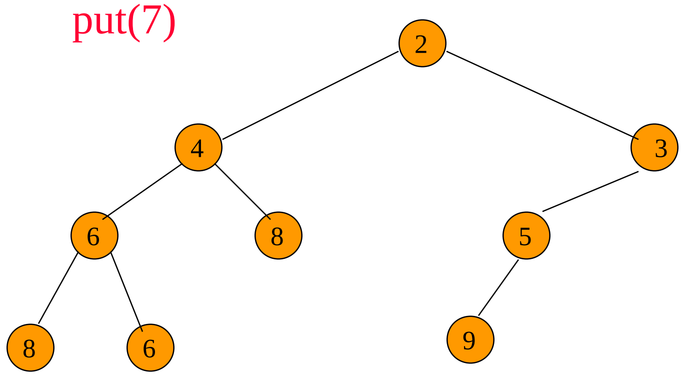
put(7)



Create a single node min leftist tree.

Put Operation

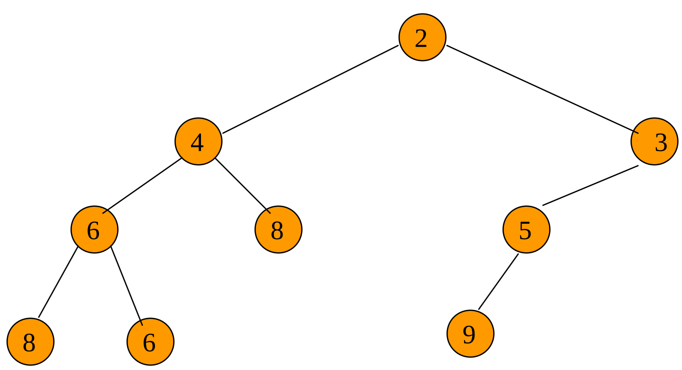
put(7)



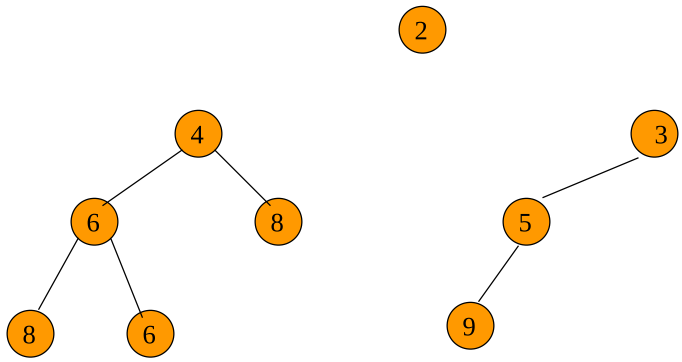
Create a single node min leftist tree.

Meld the two min leftist trees.

Remove Min

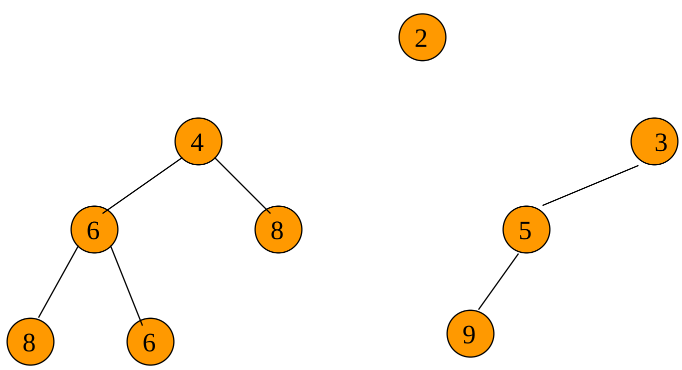


Remove Min



Remove the root.

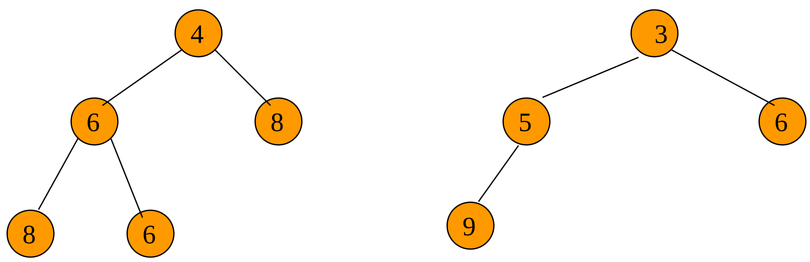
Remove Min



Remove the root.

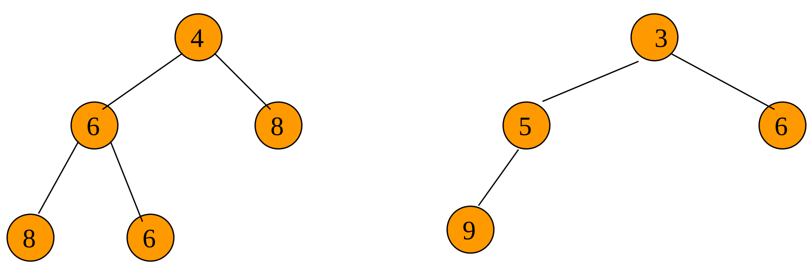
Meld the two subtrees.

Meld Two Min Leftist Trees



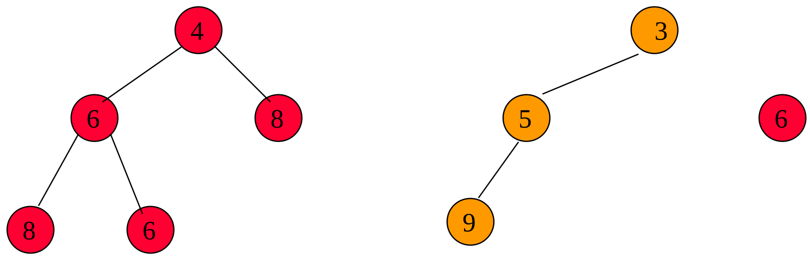
Traverse only the rightmost paths so as to get logarithmic performance.

Meld Two Min Leftist Trees



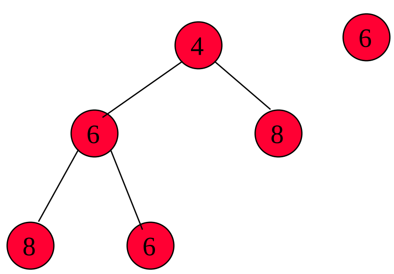
Meld right subtree of tree with smaller root and all of other tree.

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.

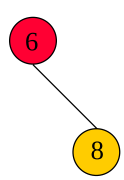
Meld Two Min Leftist Trees

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

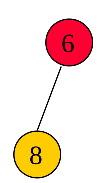
Right subtree of 6 is empty. So, result of melding right subtree of tree with smaller root and other tree is the other tree.

Meld Two Min Leftist Trees

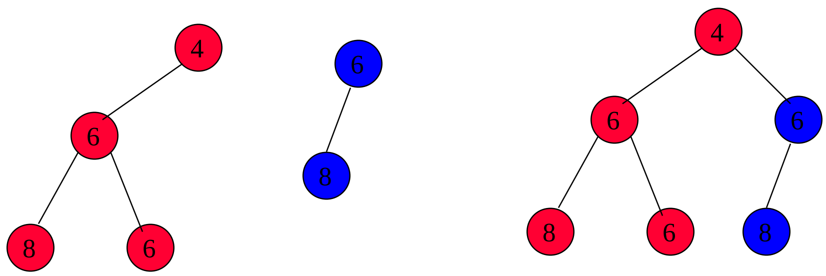
Make melded subtree right subtree of smaller root.



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



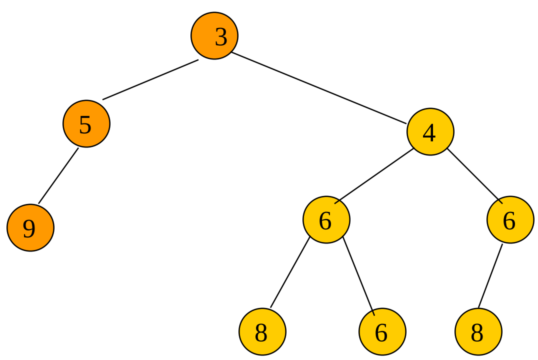
Meld Two Min Leftist Trees



Make melded subtree right subtree of smaller root.

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

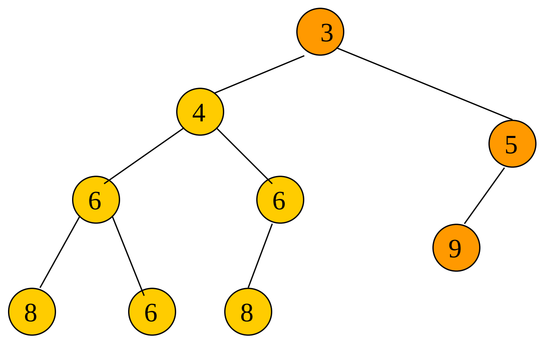
Meld Two Min Leftist Trees



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