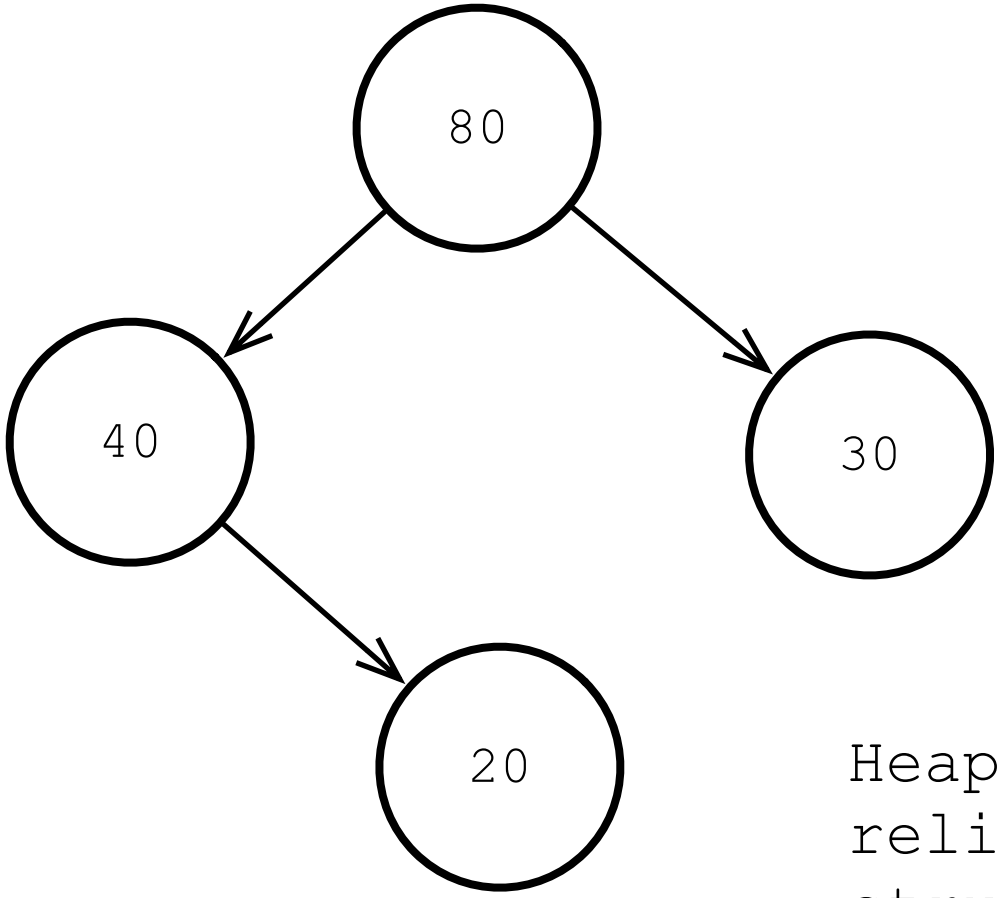
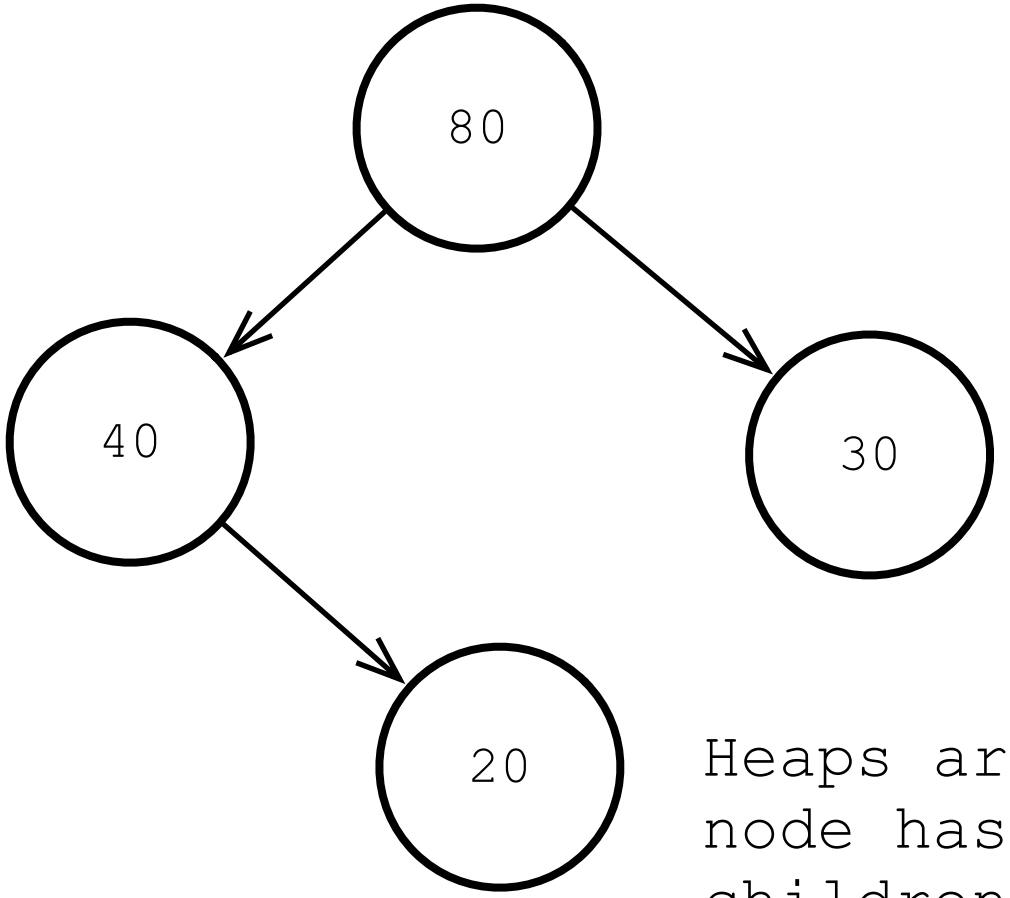


Heap Sort is an algo that relies on a specific data structure called a *heap*.

Heaps can be top-heavy, called a max heap, or bottom-heavy, called a min heap.

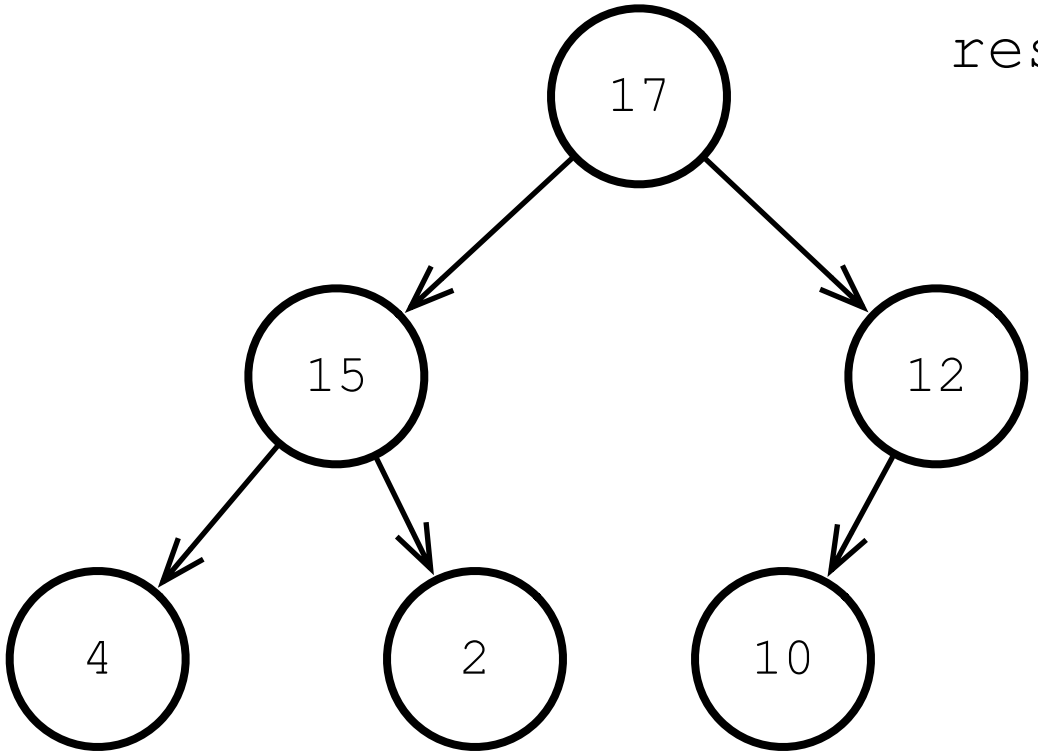


Heaps are dbinary trees: each node has zero, one, or two children. In a max heap a parent node is at least as large as its children. No distinction is made between left and right branches. Min heaps have smaller values at the top instead.

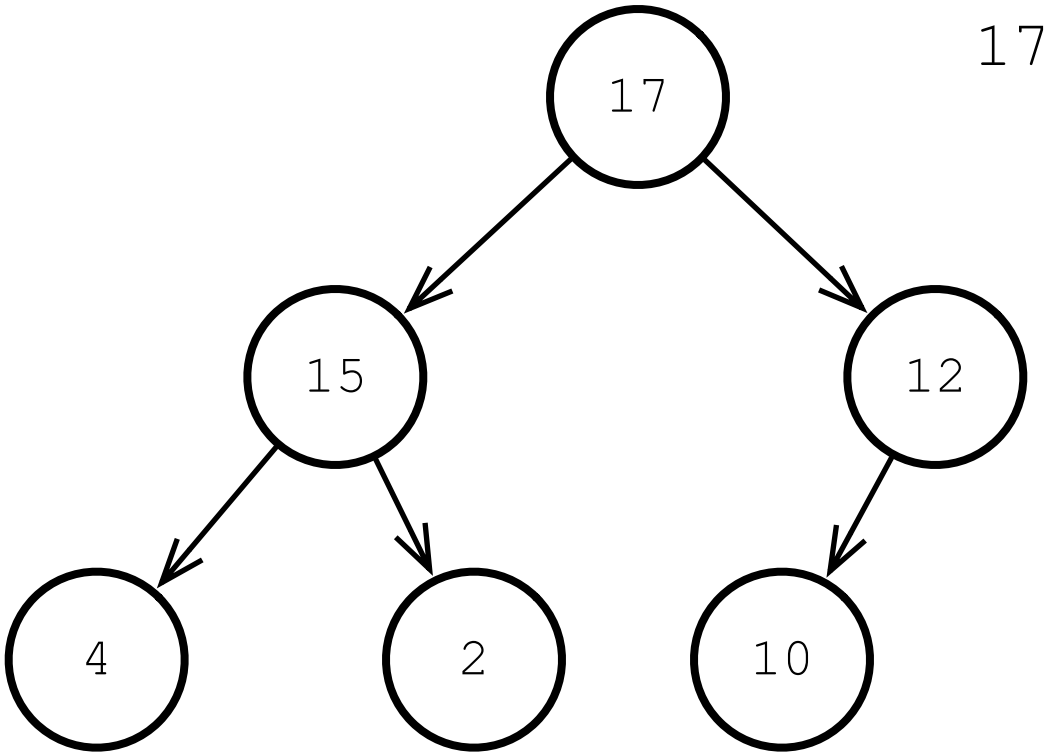
The first step in using heapsort is to build a heap out of your data. E.g. given the list:

[10, 15, 2, 4, 17, 12]

This is one possible resulting heap.



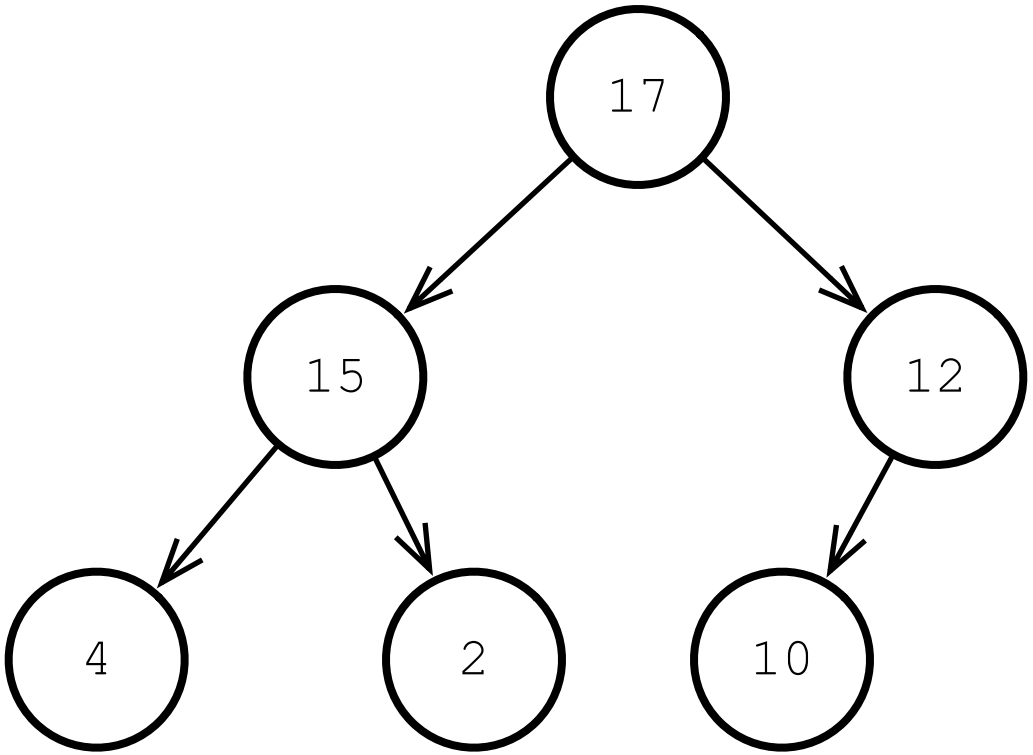
When we have a heap like this, we say that the *order* of the heap is like this: first node, then second row (left to right), then third row (left to right), and so on.



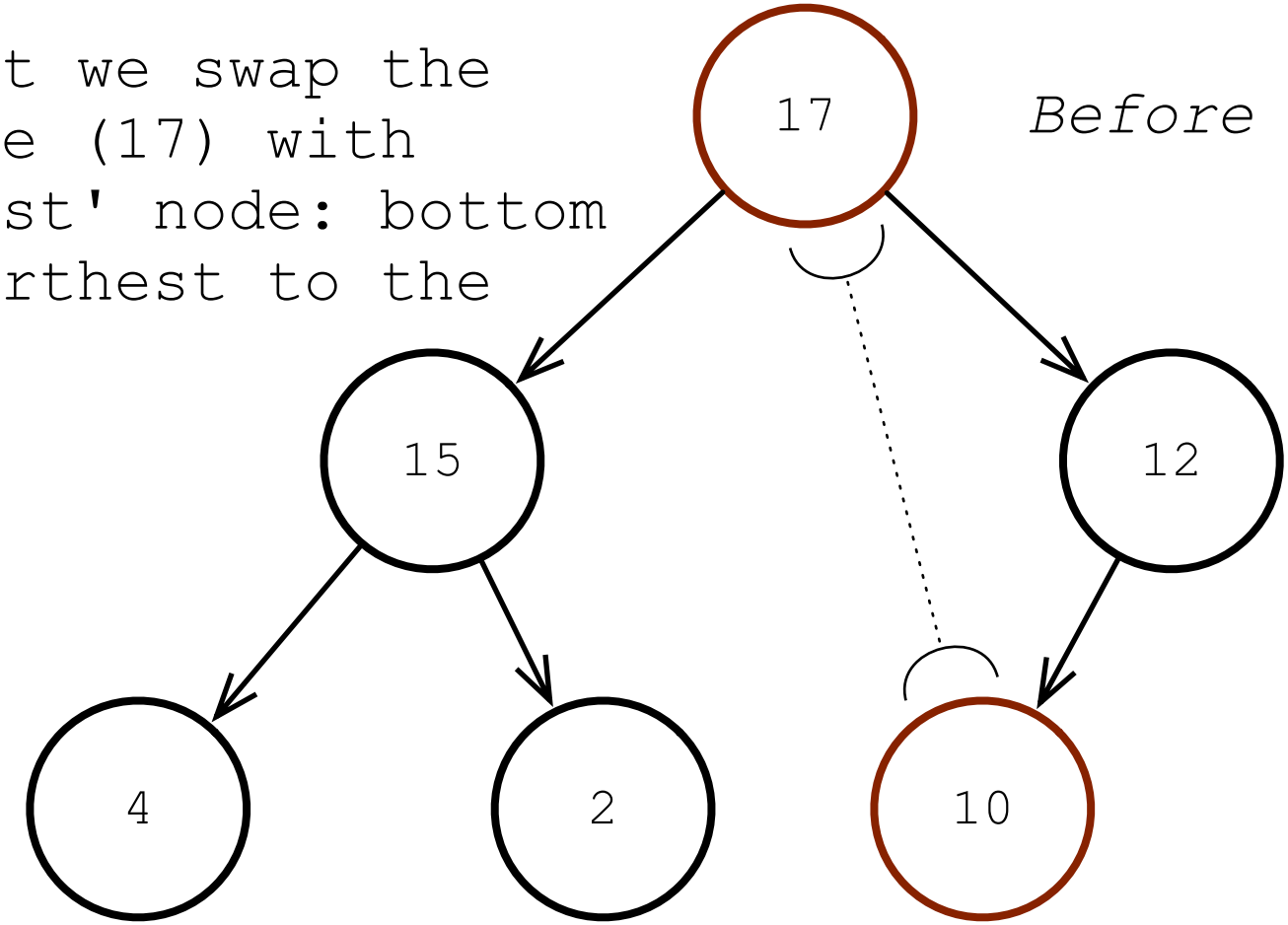
Heap order:
17, 15, 12, 4, 2, 10

The algorithm now is pretty straight forward.

1. Swap the top of the heap with the last element.
2. Remove the last element, add it to sorted list.
3. Repair the heap: large numbers above small ones.
4. Repeat until heap has nothing in it.



At start we swap the top node (17) with the 'last' node: bottom row, farthest to the right.

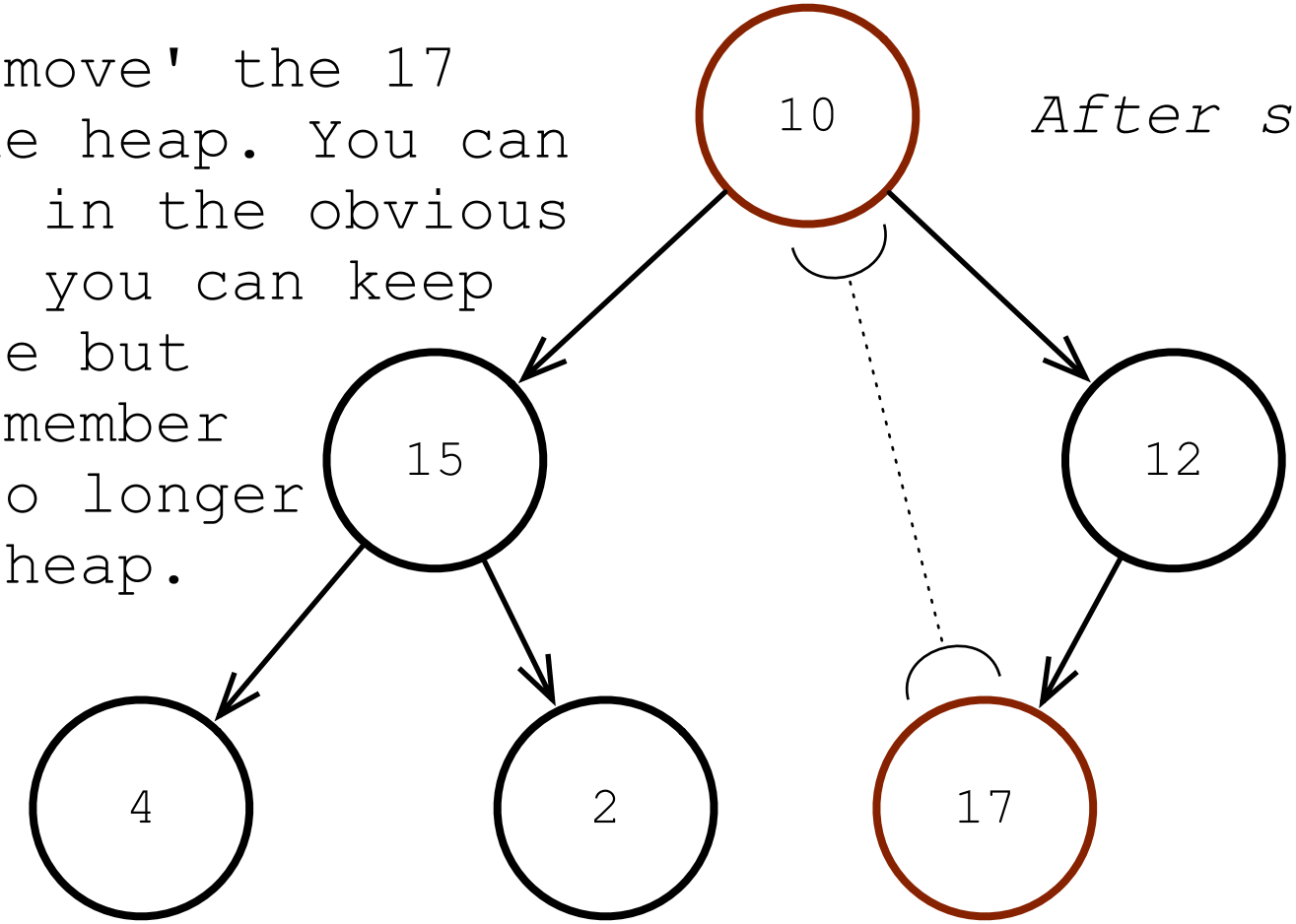


Before swapping

Return list:

(empty)

Now 'remove' the 17 from the heap. You can do this in the obvious way, or you can keep it there but just remember it is no longer in the heap.

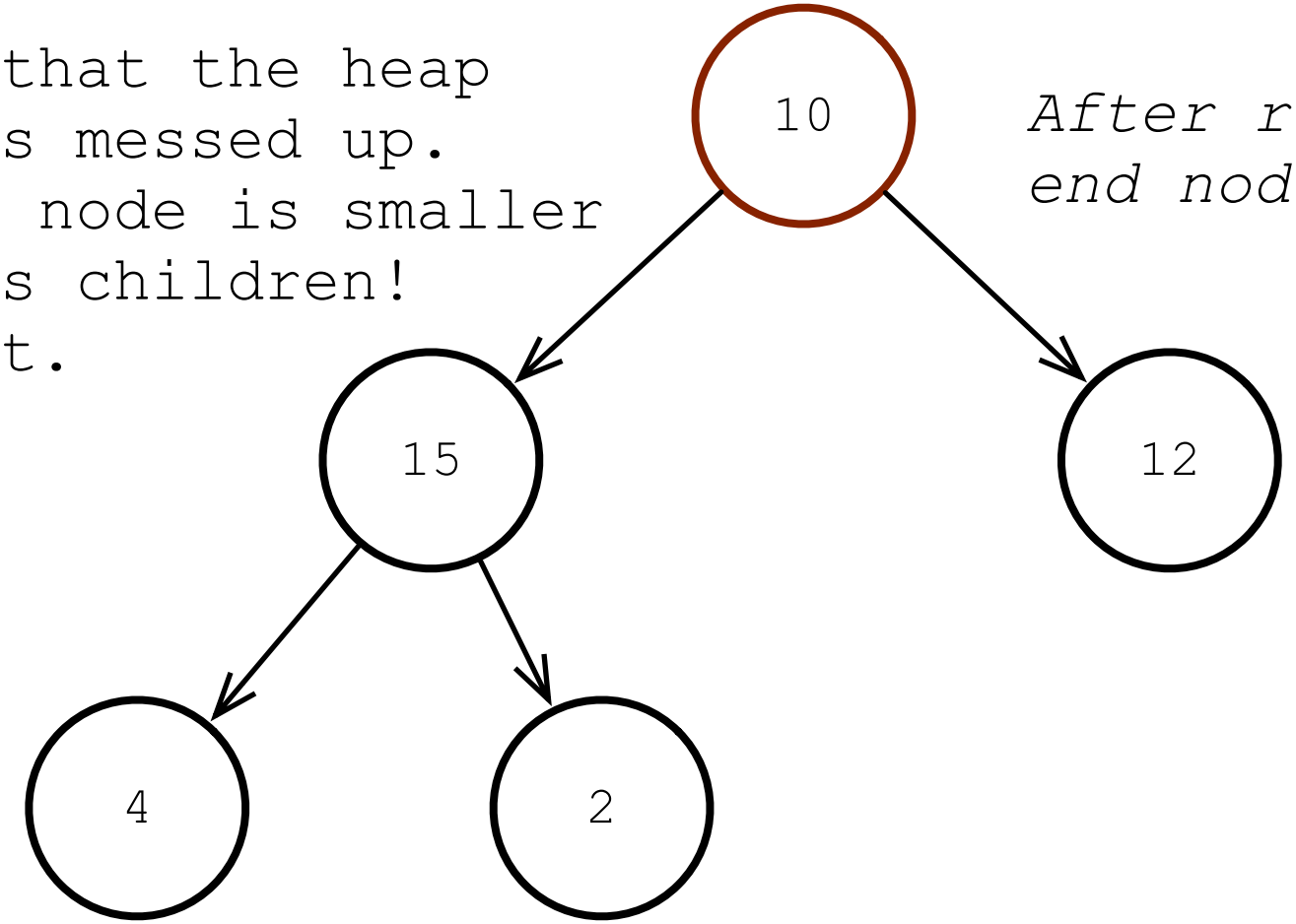


After swapping

Return list:

(empty)

Notice that the heap order is messed up. The top node is smaller than its children! Fix that.

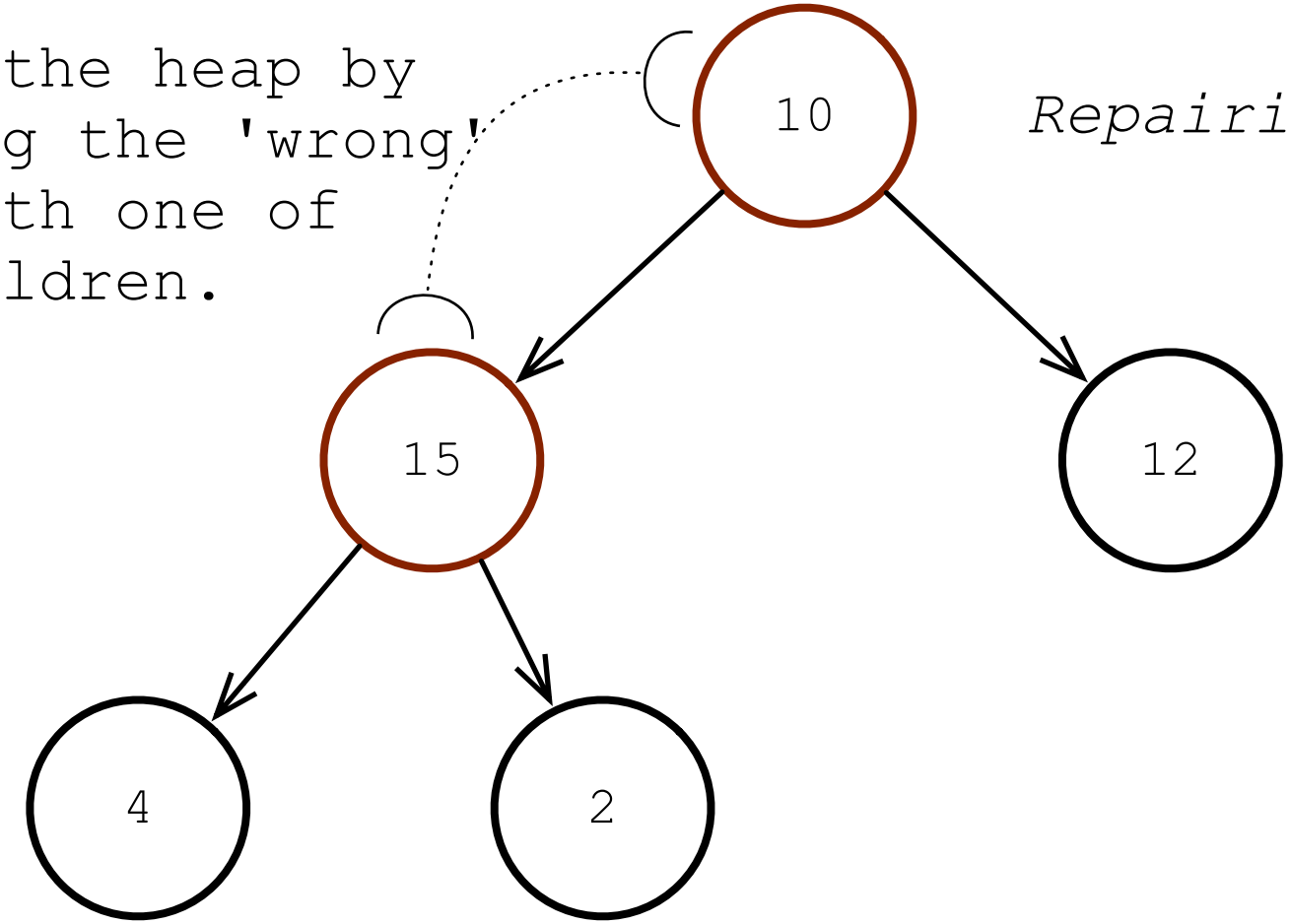


After removing end node.

Return list:

17

We fix the heap by swapping the 'wrong' node with one of the children.

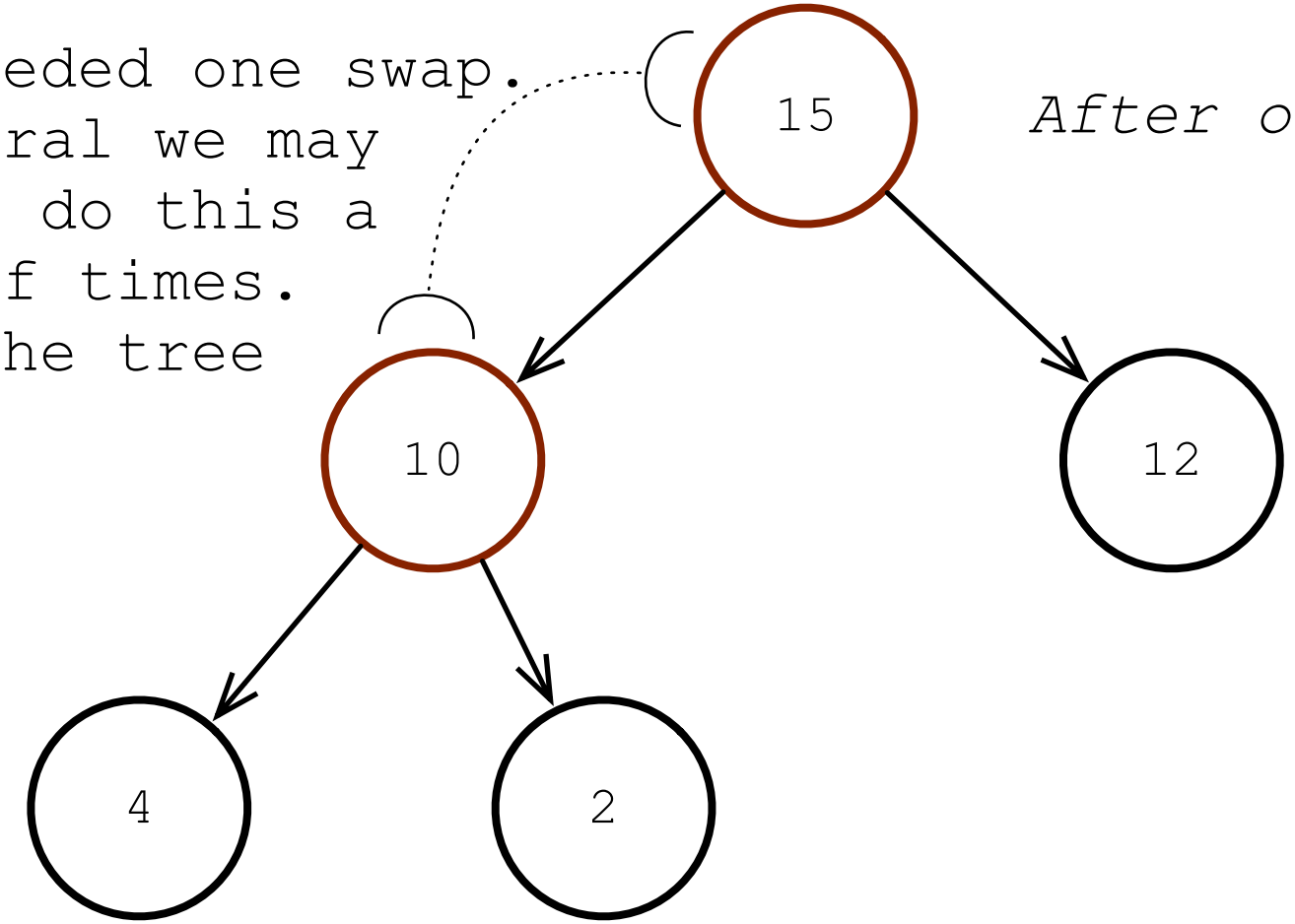


Repairing heap.

Return list:

17

Only needed one swap.
In general we may need to do this a bunch of times.
Up to the tree height.

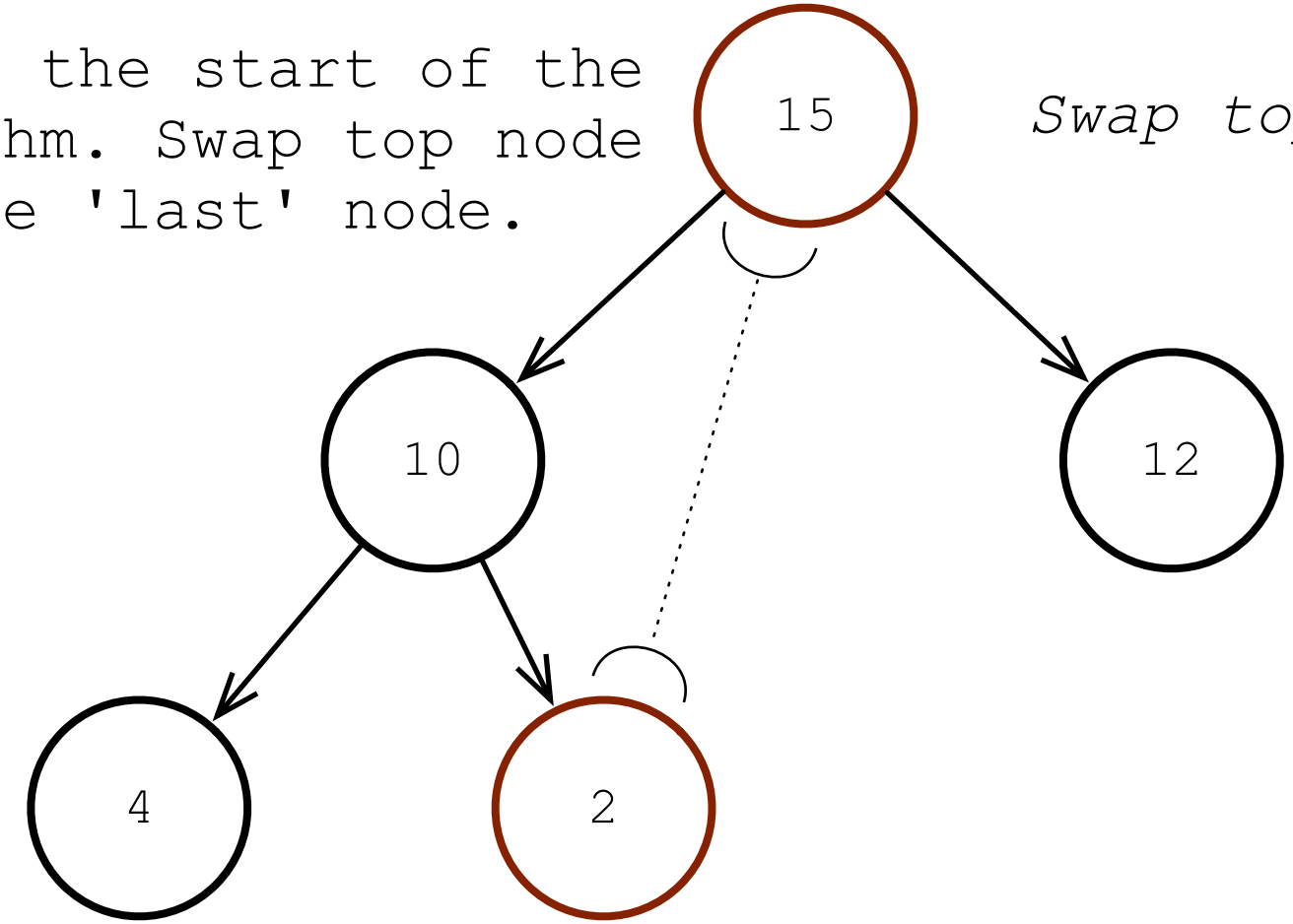


After one swap.

Return list:

17

Back to the start of the algorithm. Swap top node with the 'last' node.

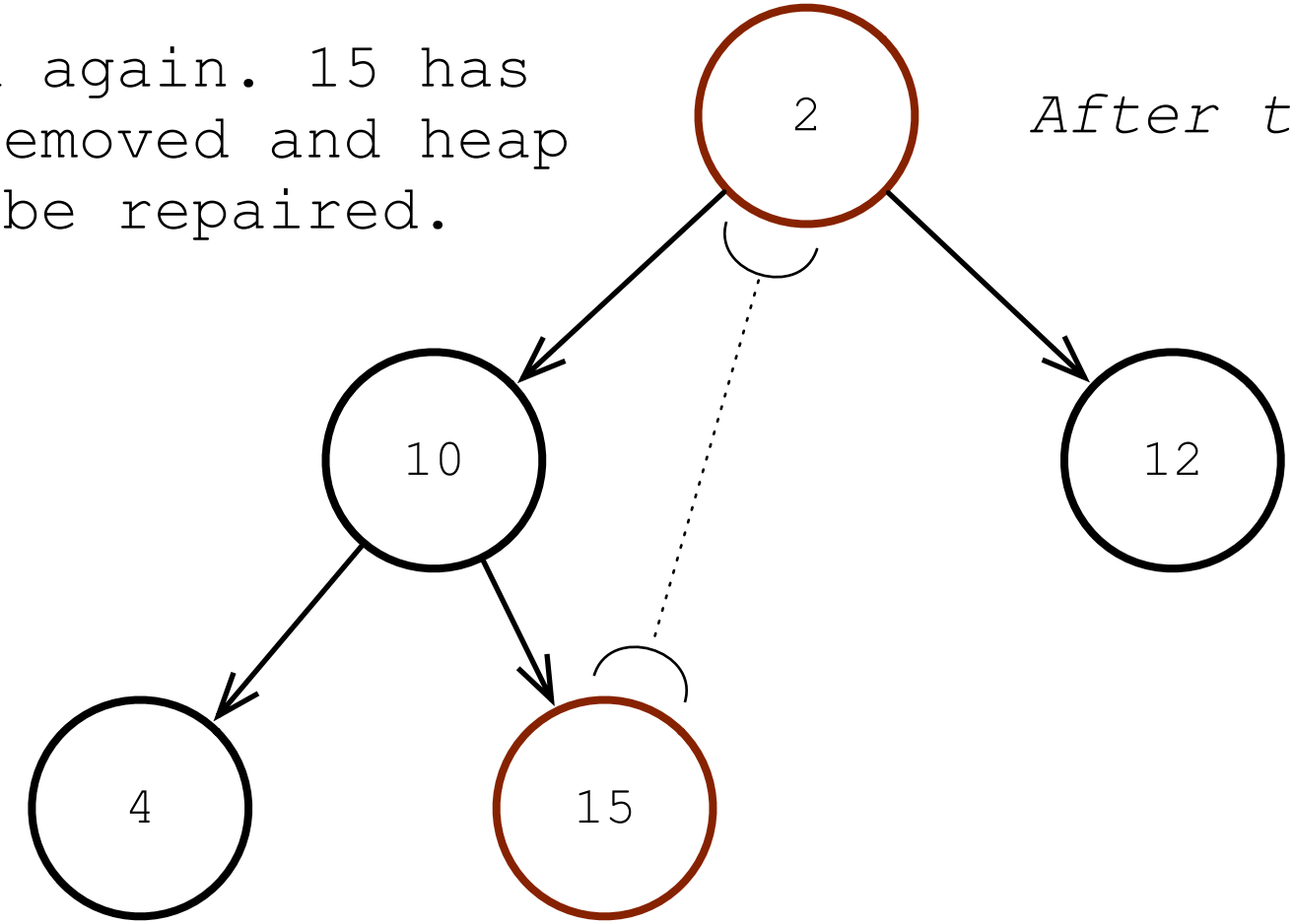


Swap top with end

Return list:

17

Swapped again. 15 has to be removed and heap has to be repaired.

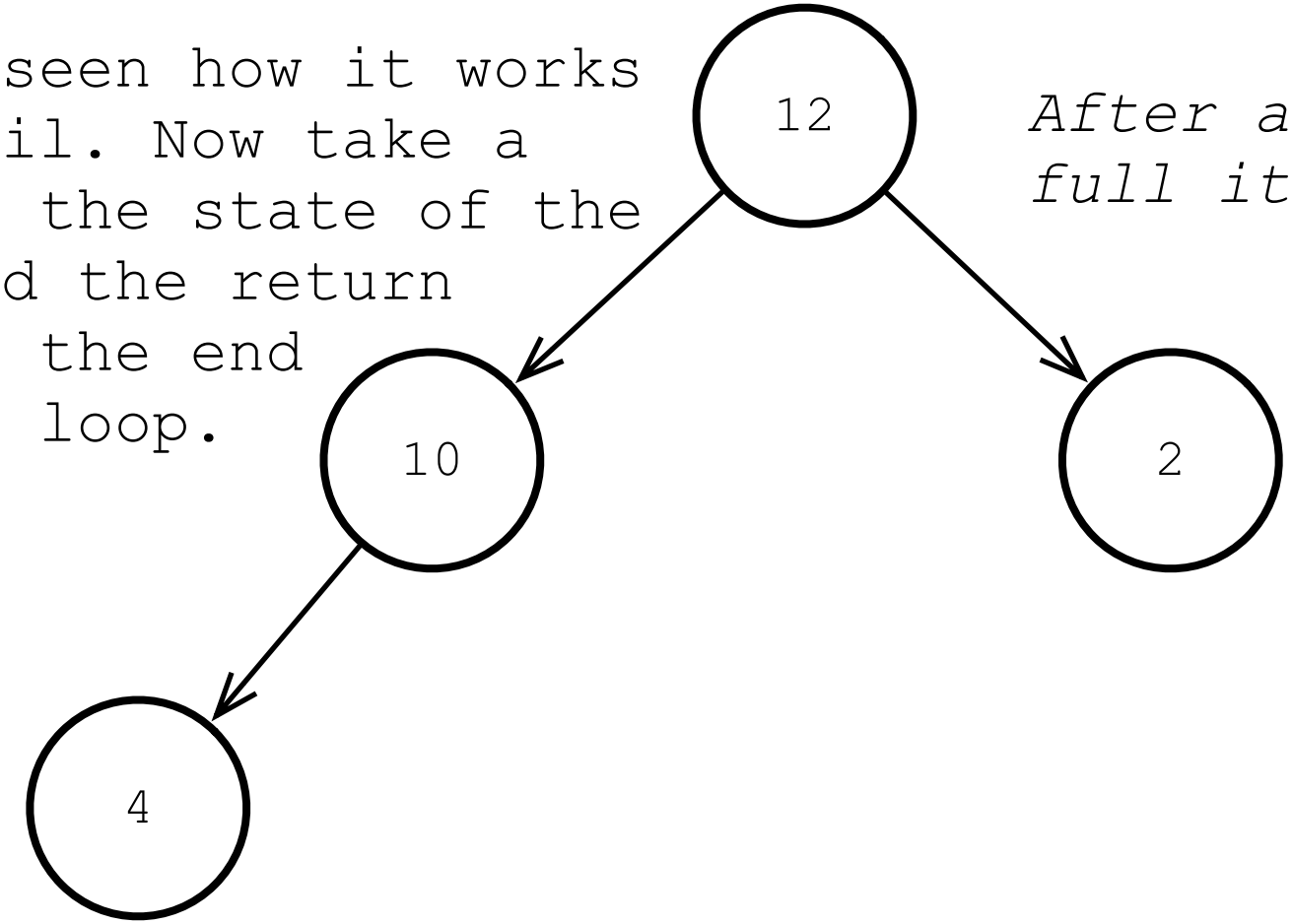


After top swap.

Return list:

17

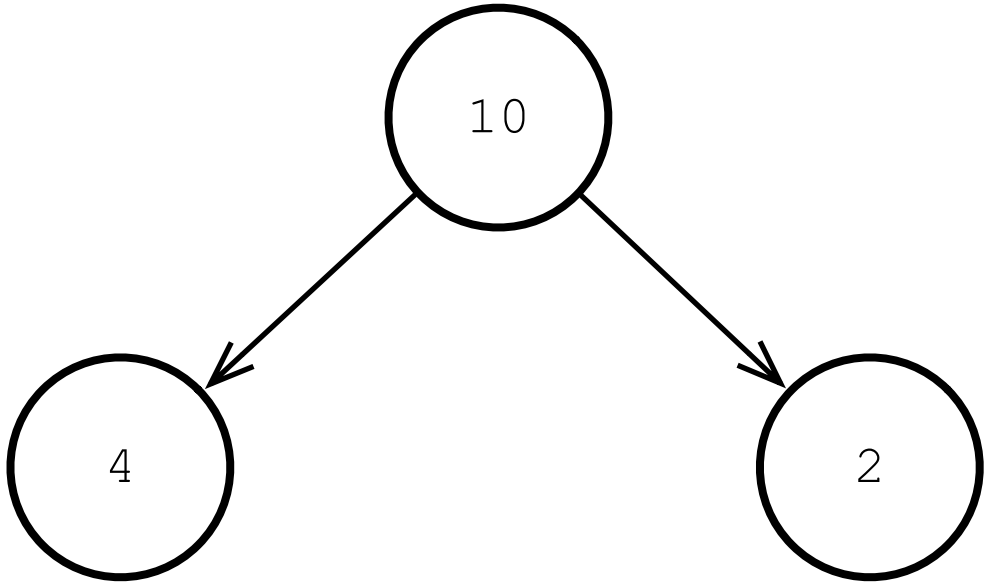
You've seen how it works in detail. Now take a look at the state of the heap and the return list at the end of each loop.



After another full iteration.

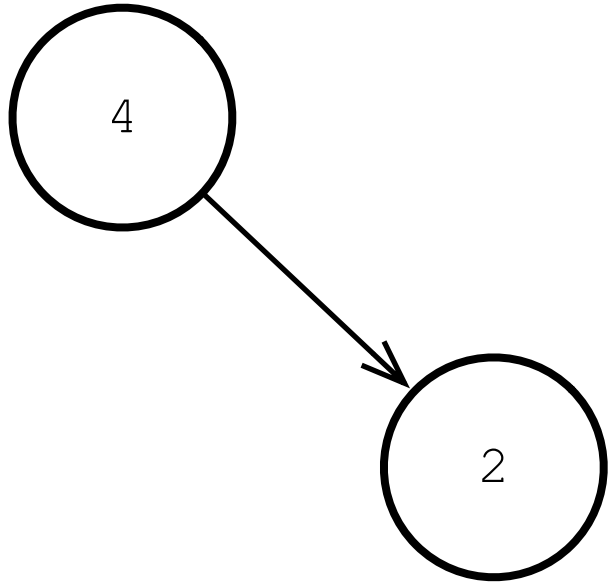
Return list:

17, 15



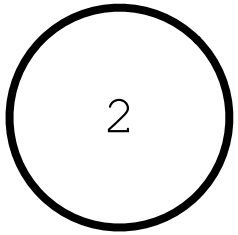
Return list:

17, 15, 12



Return list:

17, 15, 12, 10



Return list:

17, 15, 12, 10, 4

Finally at the end. Notice that the return list is actually in the reverse order from what we expect. We could fix this in a couple ways. We could use a *min heap* instead of a *max heap.*

Return list:

17, 15, 12, 10, 4, 2

Or, we could use a trick with the tree structure. If we swap with the last node and then *keep it around*, but just make it off limits to the rest of the algorithm, the resulting heap structure is in proper sort order.

For added brain melt, you can implement this without even using a binary tree. You can use an array instead. I'll just leave it at that :)