Removing an Element: Heapify Down

2 min

Maintaining a minimum value is no good if we can never retrieve it, so let's explore how to remove the root node.

In the diagram, you can see removing the top node itself would be messy: there would be two children orphaned! Instead, we'll swap the root node, 2, with the bottom rightmost child: 20. The bottom rightmost child is simple to remove because it has no children.

Unfortunately, we've violated the heap property. 20 is now the root node, and that's not the minimum value in the heap. We'll *heapify down* to restore the heap property.

This process is similar to heapifying up, except we have two options (5 and 10) where we can make a swap. We'll choose the **lesser of the two values** and swap 20 with 5. This is necessary for the heap property, if we had chosen to swap 20 with 10, then the minimum value would **not** be at the root. With 5 at the root, the root node is the minimum value in the heap again.

Another swap is required because 20 is greater than its children, so we swap 20 with 11.

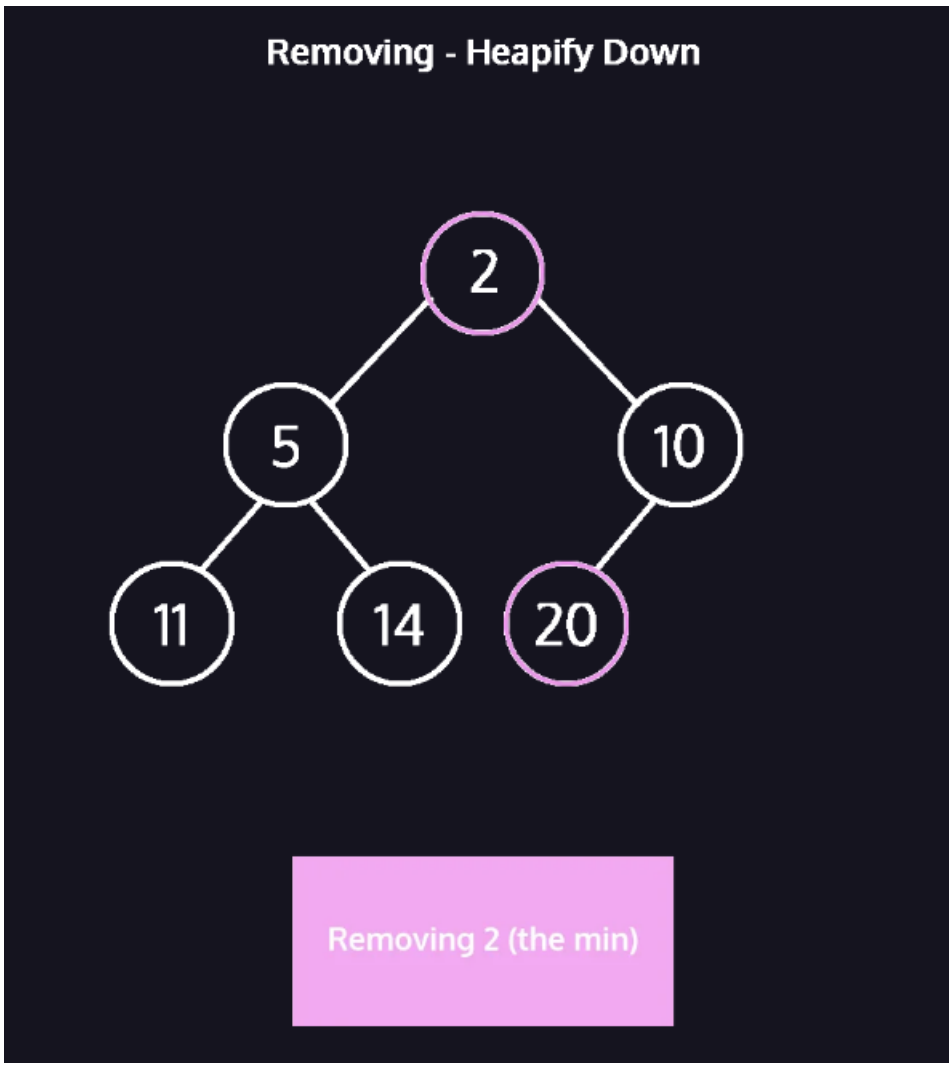
Now 20 no longer has any children (it is a child of 11), and all other nodes in the heap only have parents with smaller values.

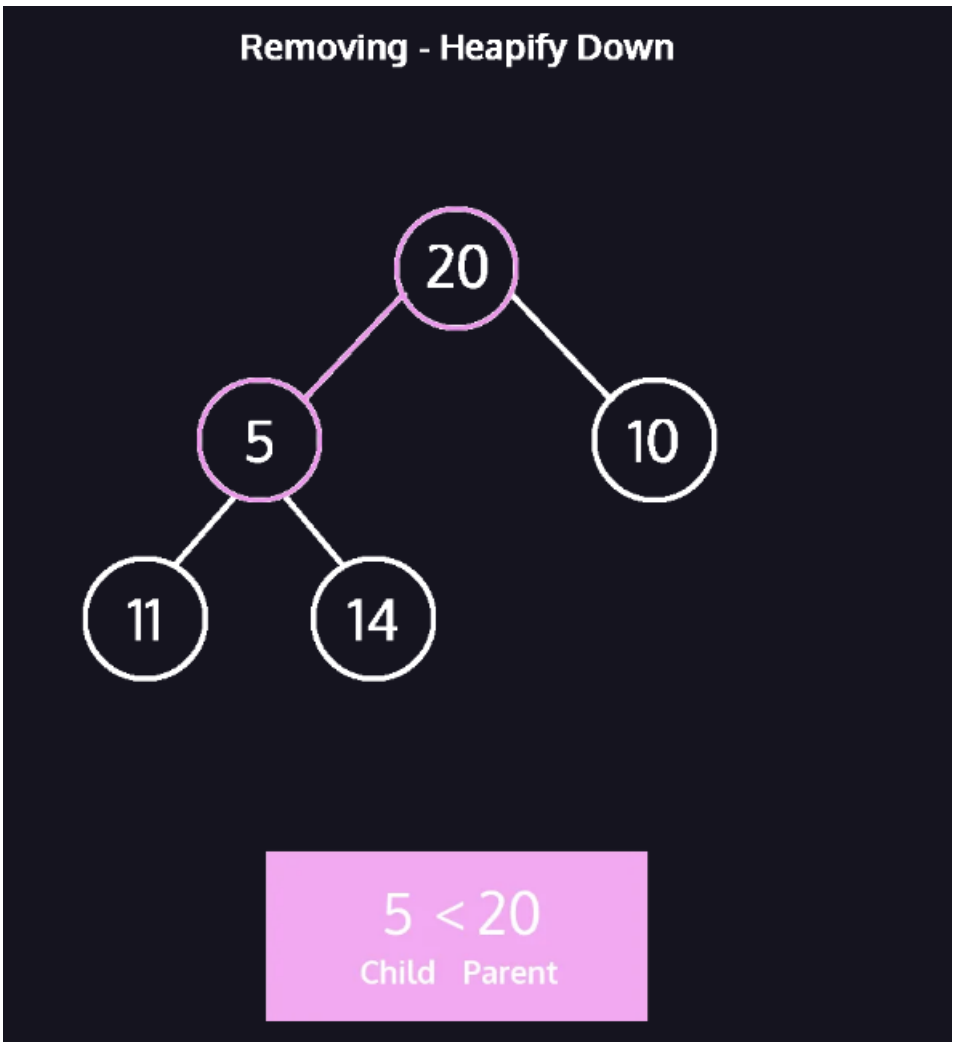
Just like that, we've retrieved the minimum value, allocated a *new* minimum, and maintained the heap property!

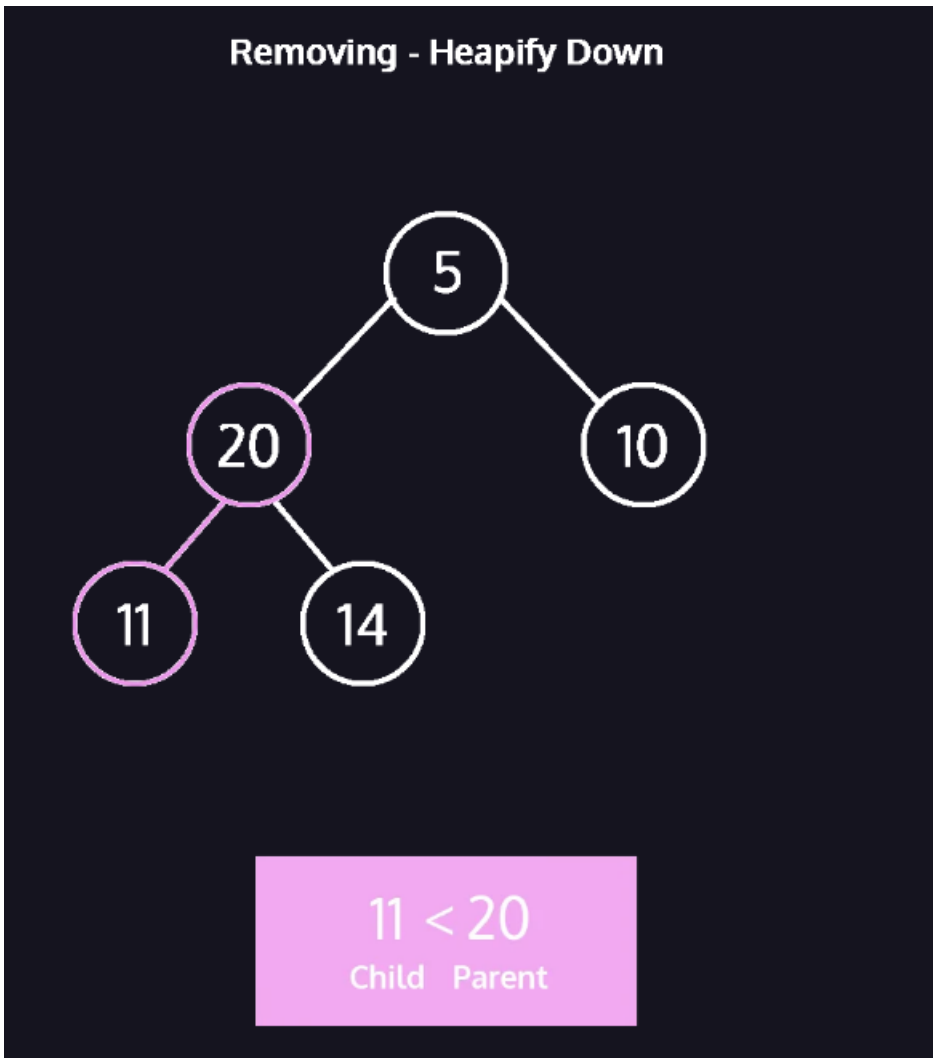
Instructions

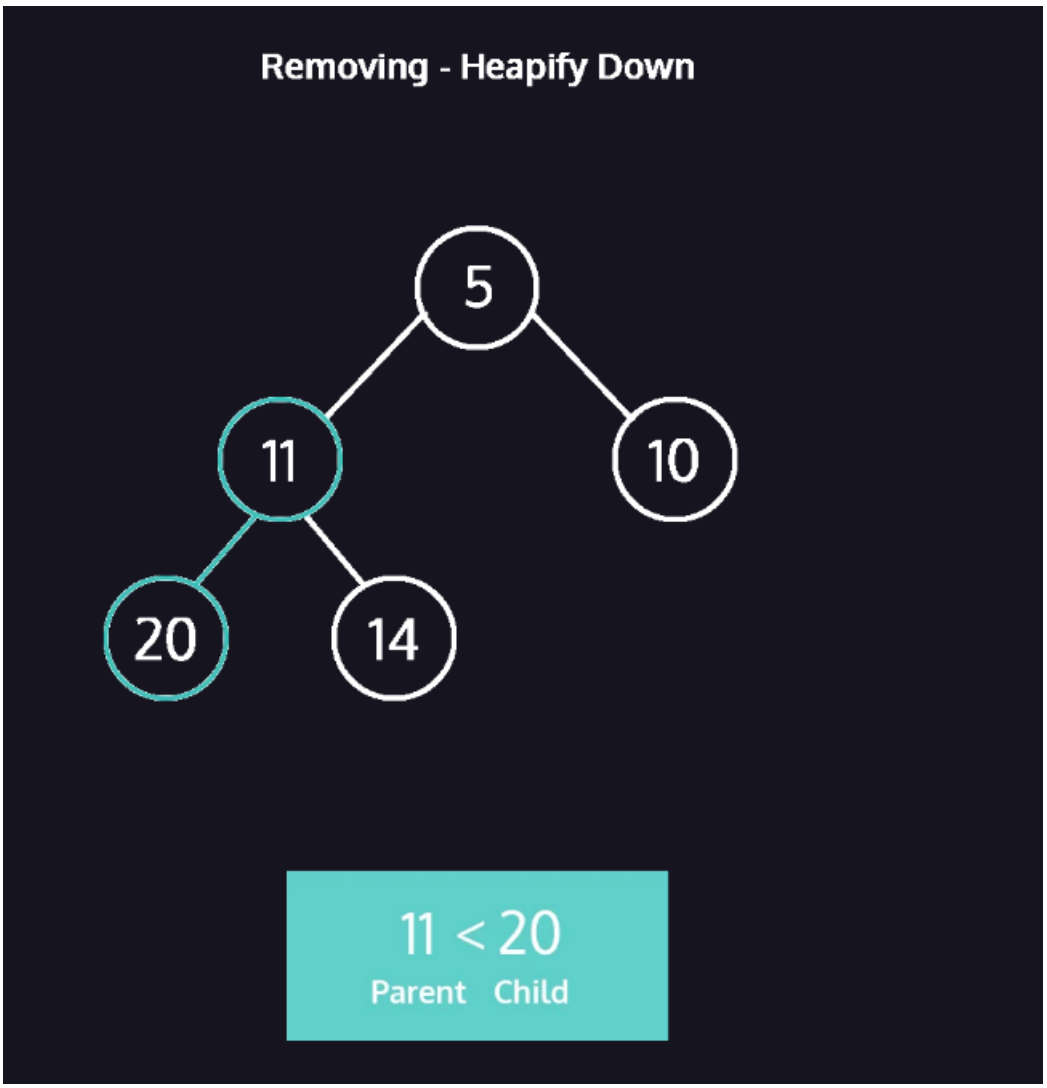
How many swaps in total did we make to retrieve the minimum value and restore the heap property?

If we have two children where we can make a swap, why do we always choose the lower of the two values?


Removing - Heapify Down
2
5
10
11
14
20
Removing 2 (the min)


Removing - Heapify Down
20
5
10
11
14
5 < 20
Child Parent


Removing - Heapify Down
5
20
10
11
14
11 < 20
Child Parent


Removing - Heapify Down
5
11
10
20
14
11 < 20
Parent Child