Min Heap: Introduction

This lesson will give a brief introduction to Min-Heaps and how elements are inserted and removed from them

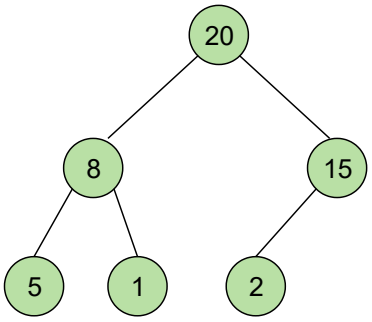
We'll cover the following

Building a Min-Heap

As mentioned in a previous lesson, Min Heaps follow the Min Heap property which means that the key at the parent node is always smaller than the keys at the child nodes. Heaps can be implemented using lists. Initially, elements are placed in nodes in the same order as they appear in the list. Then a function is called over the whole heap in a bottom-up manner that "Min Heapifies" or "percolates up" on this heap so that the heap property is restored. The "Min Heapify" function is bottom-up because it starts comparing and swapping parent-child key values from the last parent (at the $\frac{n}{2}$and index).

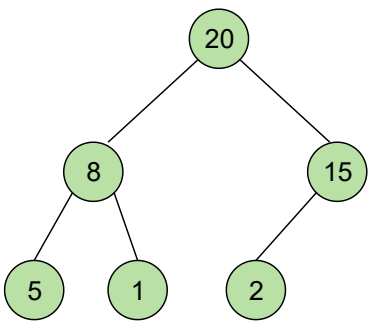
For a visual demonstration of heap creation, check out the following illustration.

Build a Min Heap!



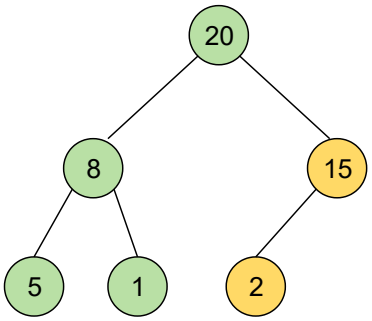
1 of 20

Insert all the elements from array to heap nodes in the same order as they appear in the array. Then we start from the last parent node and start comparing the key parent value with its children

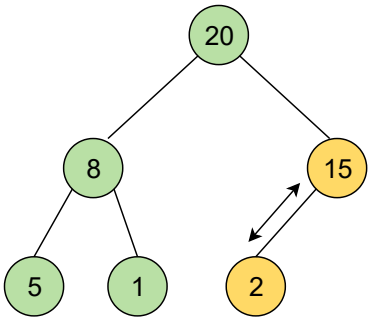


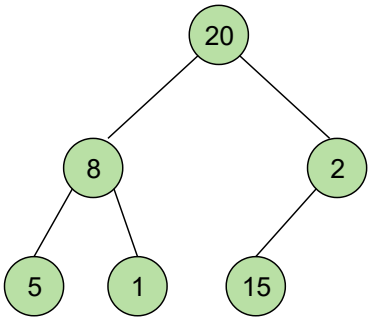
2 of 20

Compare 15 and 2

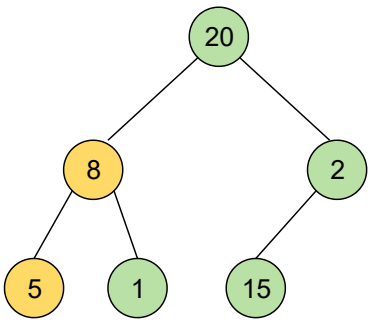


2 < 15,
so we swap them

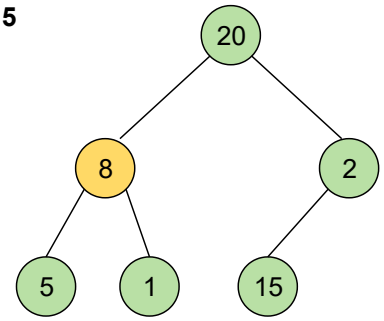




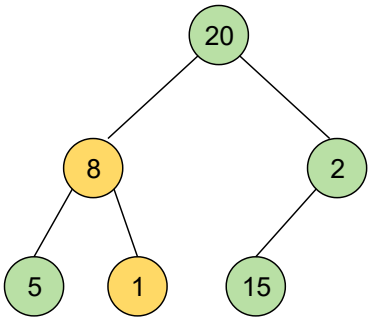
Compare 8 and 5



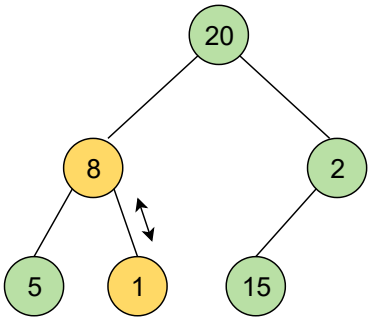
5 < 8,
check if right child
of 8 is smaller than 5

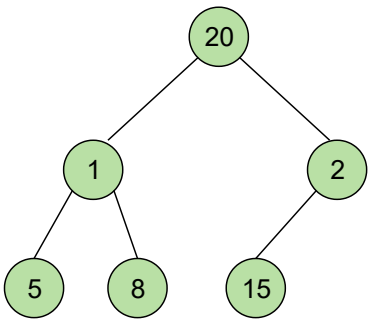


Compare 8 and 1

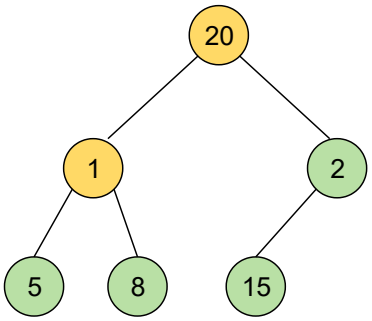


1 < 8 and 1 < 5,
so swap 8 and 1

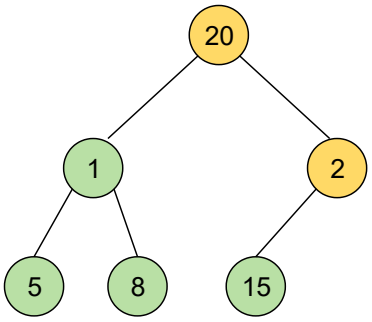




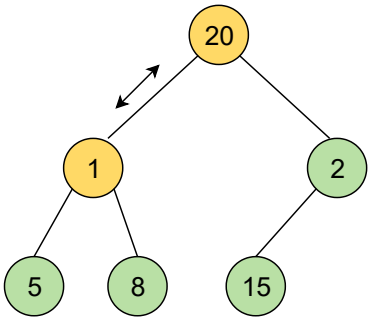
Compare 20 and 1

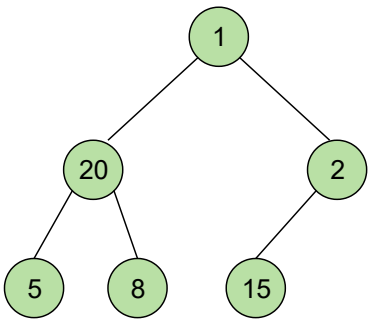


1 < 20,
check if right child of 20
is smaller than 1

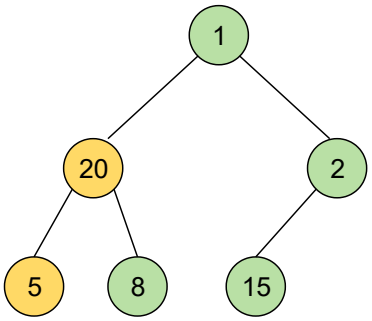


1 < 2 and 1 < 20,
so swap 1 and 20

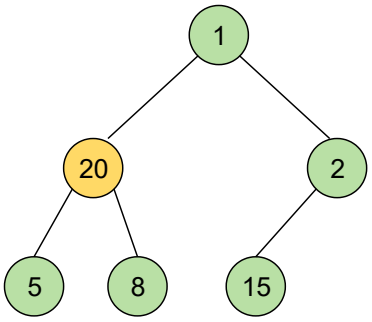




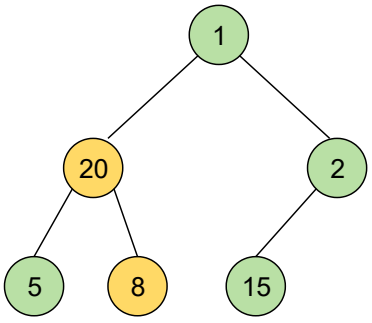
Compare 20 and 5



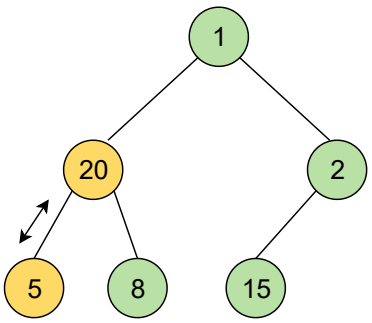
5 < 20,
check if right child of 20
is smaller than 5

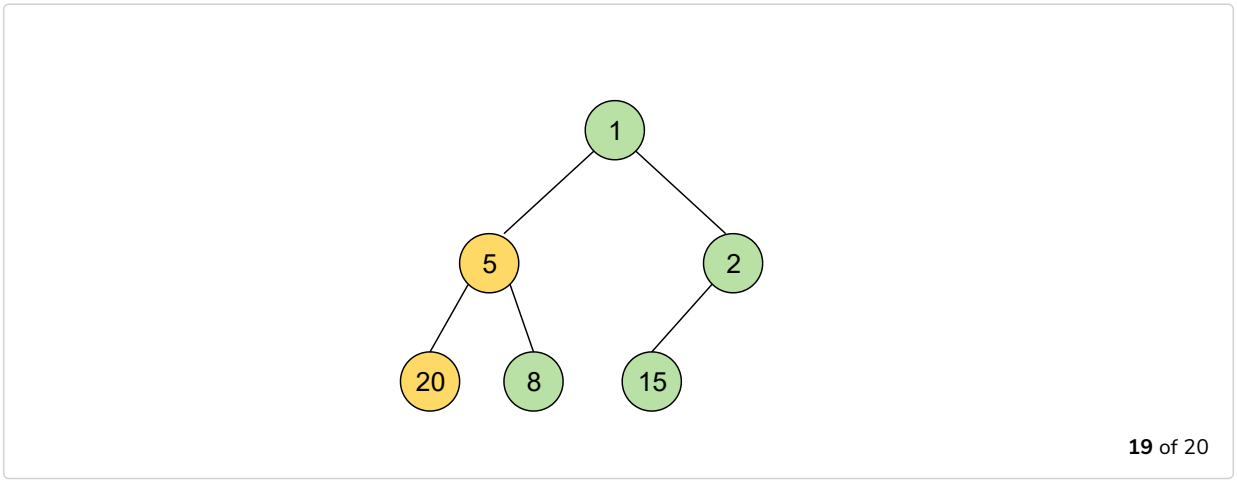


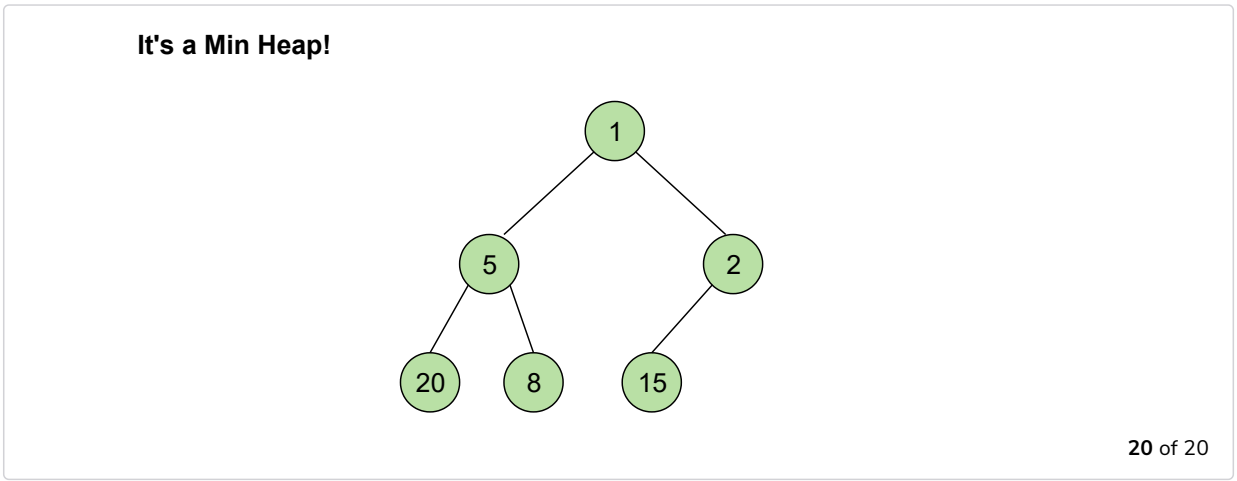
Compare 8 with 20 and 5



5 < 20 and 5 < 8,
so swap 5 and 20





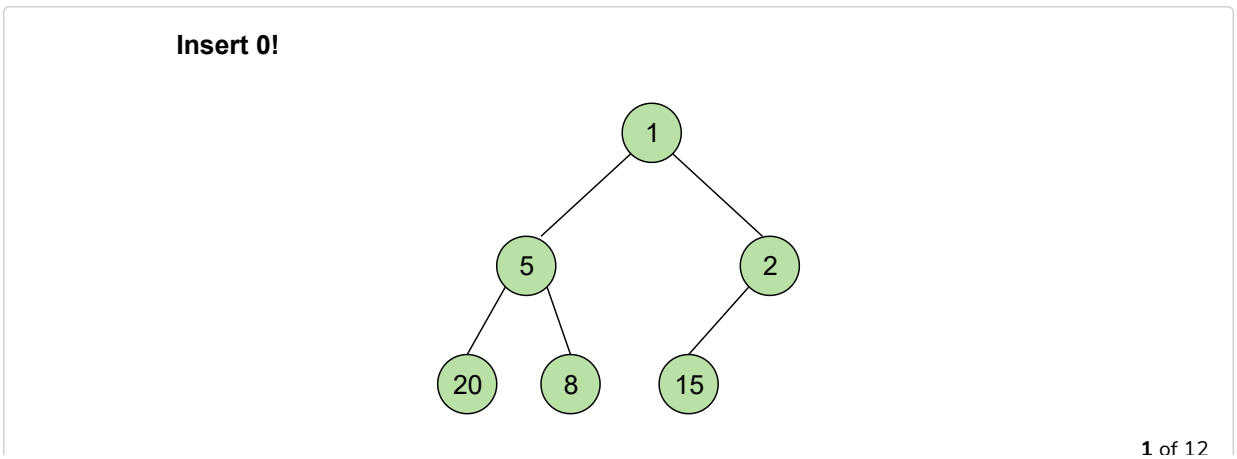


Insertion in Min Heap

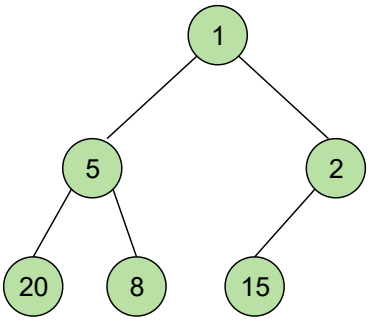
Here is a high-level description of the algorithm to insert elements into a heap and maintain the heap property:

- Create a new child node at the end of the heap
- Place the new key at that node (append it to the list or array)
- Percolate up until you reach the root node and the heap property is satisfied

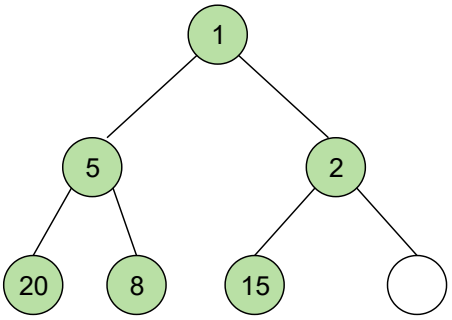
Here's a visual representation of what we just studied

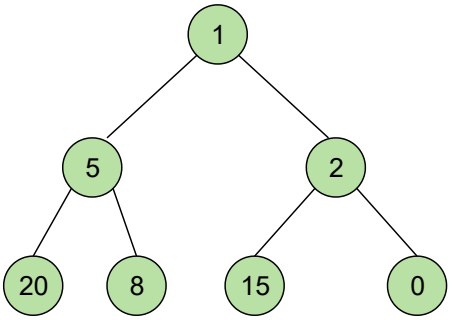


Create a new node at the last level

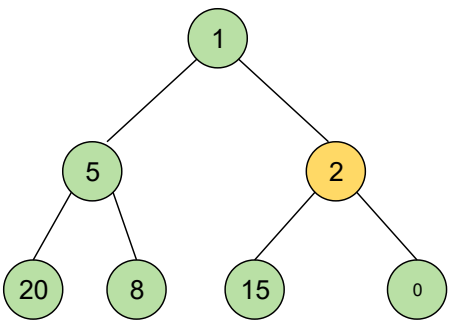


Insert 0

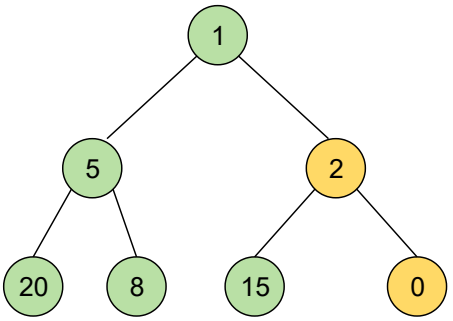




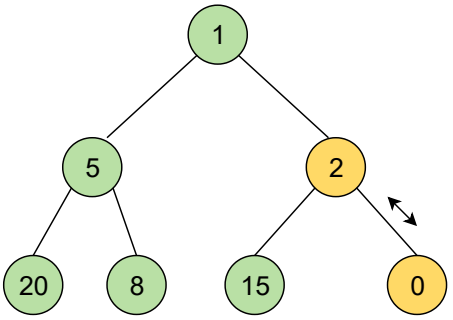
Apply Min Heap property

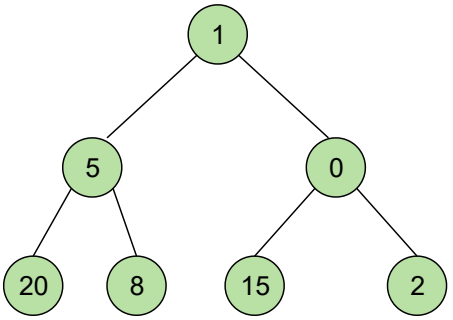


Compare 2 and 0

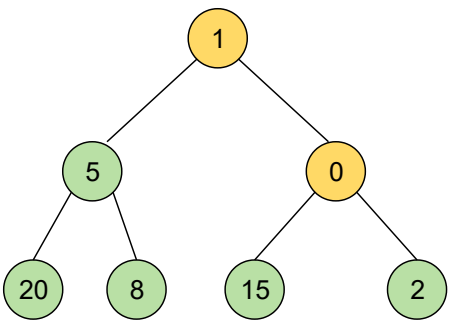


0 < 2,
so swap them

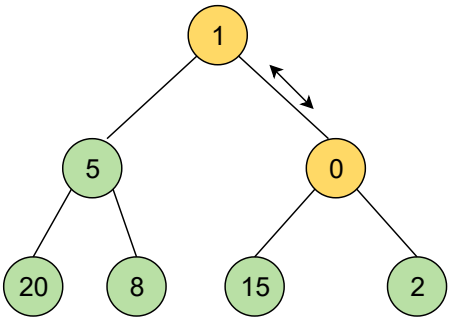




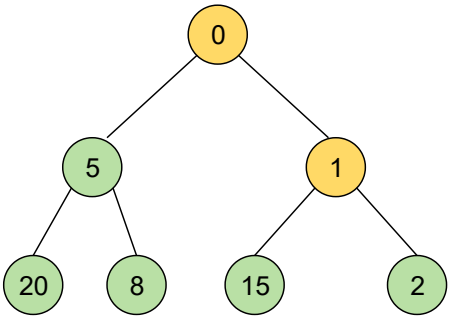
Compare 1 and 0



0 < 1,
so swap them

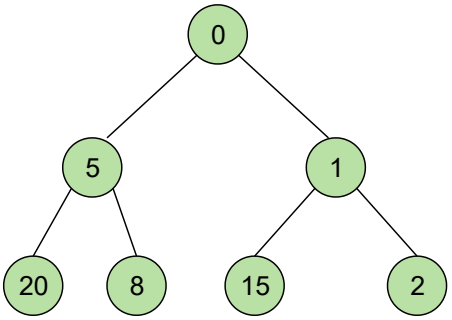


10 of 12



11 of 12

It's a Min Heap!



12 of 12

Remove Minimum in Min Heap

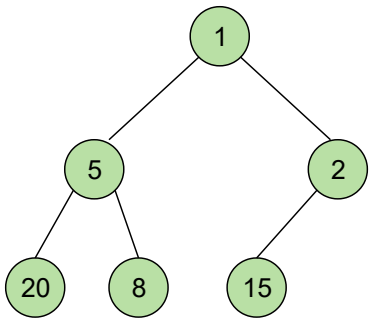
Here is the algorithm that you will follow to make sure the heap property still holds after deleting the root element

- Delete the root node
- Move the key of the last child node to root
- Perculate down: if the key is larger than the key at any of the child nodes, swap values

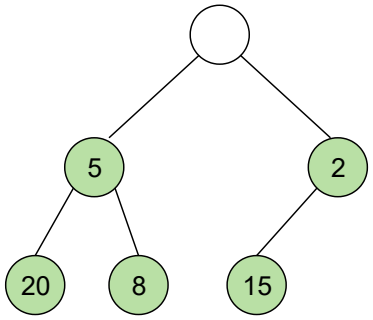
- Repeat until you reach the last node

Here's a visual representation of the remove maximum algorithm

Delete Root!

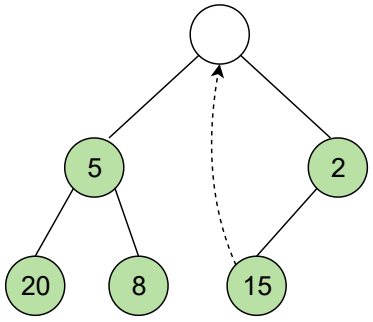


1 of 10



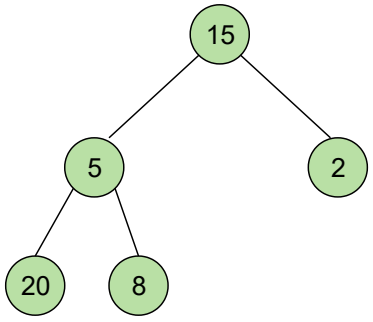
2 of 10

Move key of last child at the last level to the root

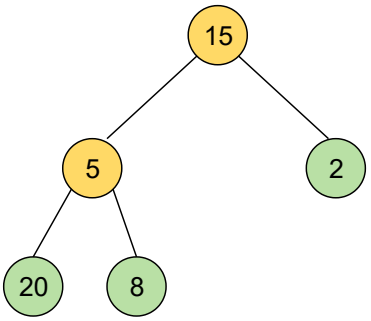


3 of 10

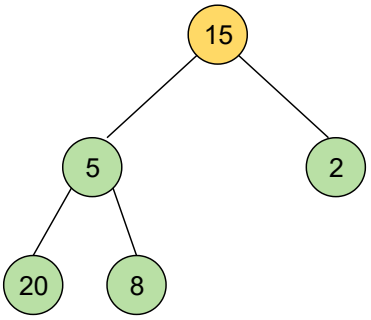
Apply Min Heap property



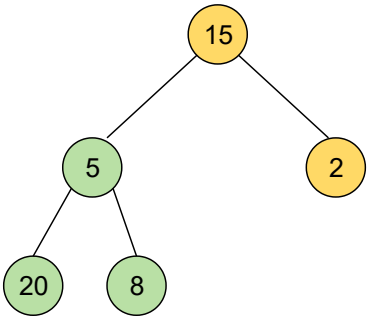
Compare 15 and 5



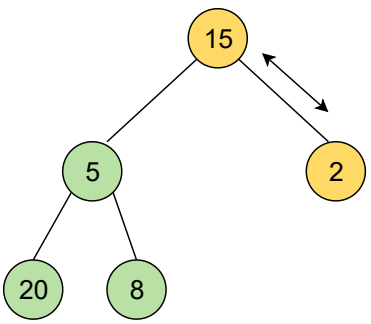
5 < 15,
check if the right child of 15
is smaller than 5

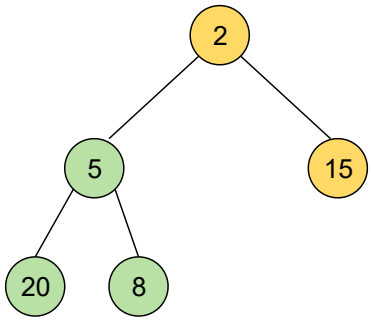


Compare 2 with 15 and 5



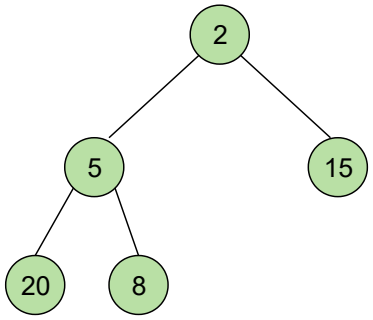
2 < 15 and 2 < 5,
so swap 2 and 15





9 of 10

It's a Min Heap!



10 of 10

Now that we have already implemented a Max Heap in the previous lesson, implementing a Min Heap would be a piece of cake! Let's look at its implementation in the next lesson.

Back

Max Heap (Implementation)

Next

Min Heap (Implementation)

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/min-heap-introduction__introduction-to-heap__data-structures-for-coding-interviews-in-python)