Max Heap: Introduction

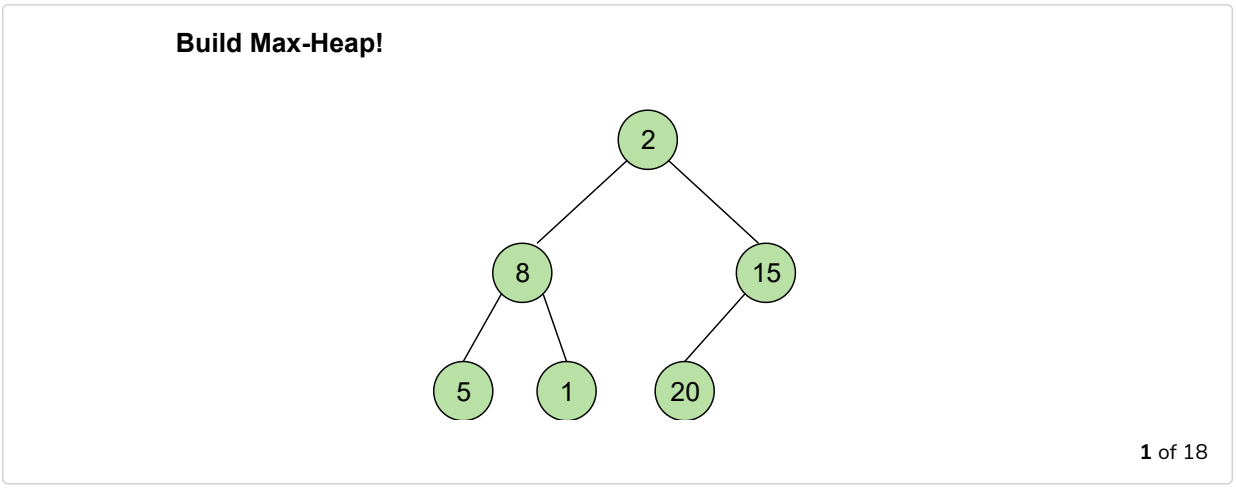
This lesson will give a brief introduction to Max-Heaps and how their elements are inserted or removed from Max-Heaps.

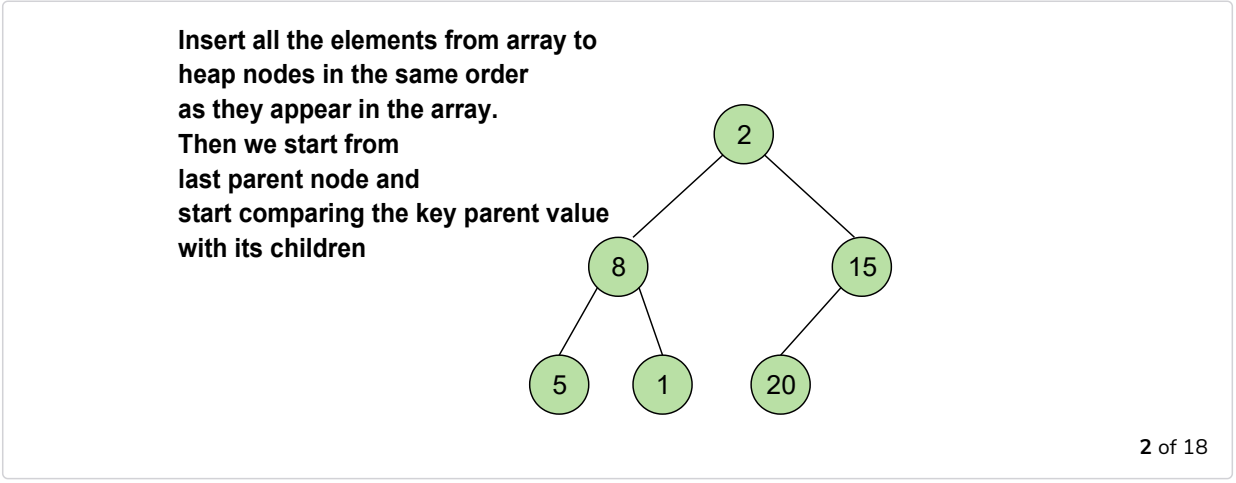
We'll cover the following

Building a Max-Heap

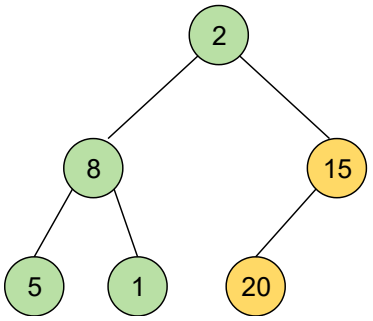
As mentioned in the previous lesson, max heaps follow the max heap property which means that the key at the parent node is always greater than the keys at the child nodes. Heaps can be implemented using lists or using node and tree classes. Although they are generally implemented using lists or arrays as that is the more space efficient approach! To build a heap, start with an empty one and successively `insert()` all the elements.

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

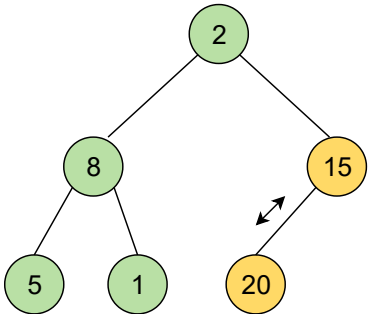


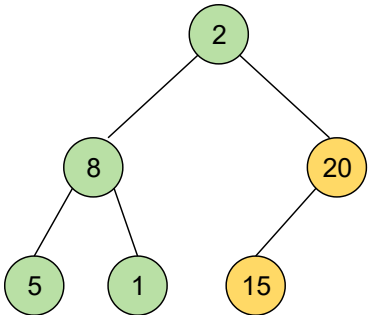


Compare 15 and 20

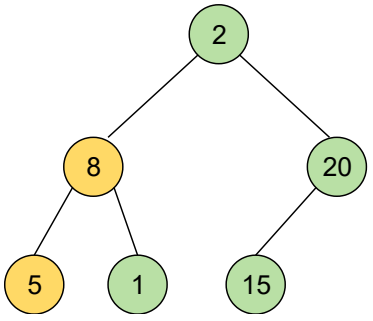


20 > 15,
so swap them

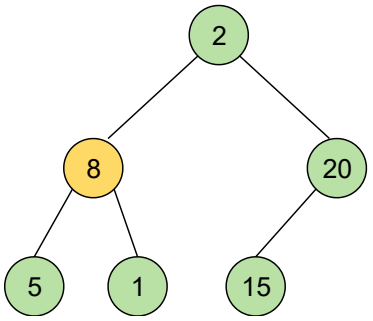




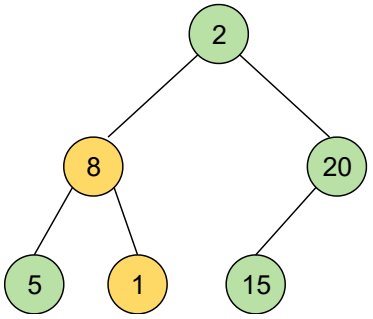
Compare 8 and 5



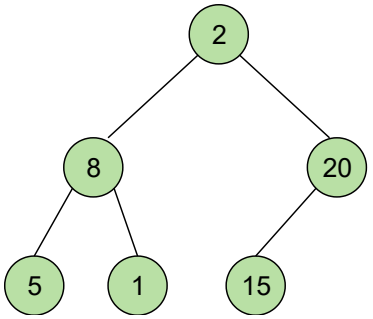
5 < 8,
so move ahead



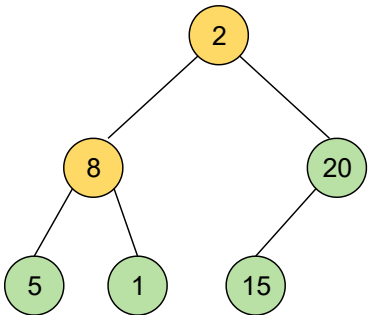
Compare 8 and 1



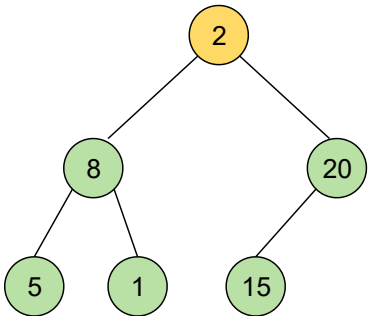
1 < 8,
so move ahead



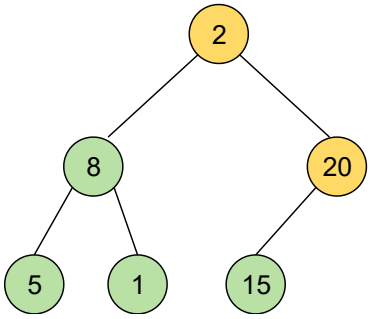
Compare 2 and 8



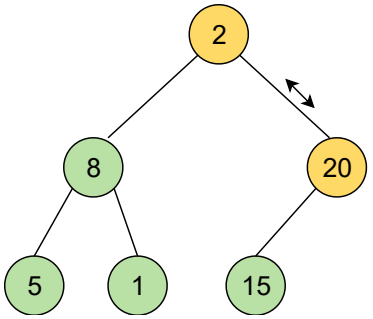
8 > 2,
check if right child of 2
is greater than 8

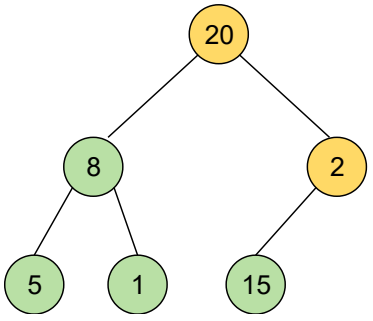


Compare 20 with 2 and 8

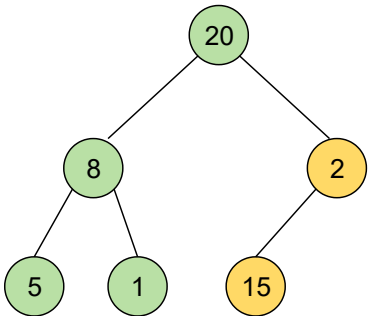


20 > 2 and 20 > 8,
so swap 2 and 20

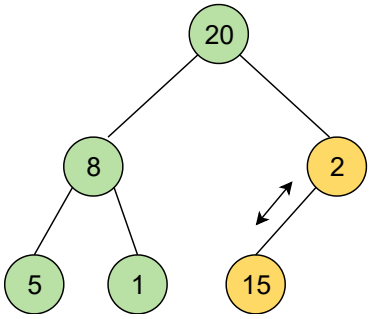


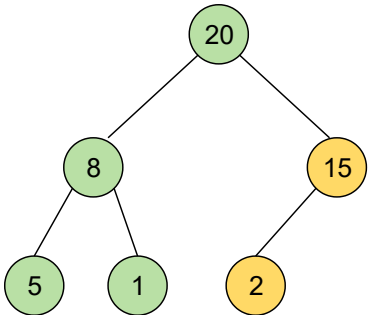


Compare 2 and 15

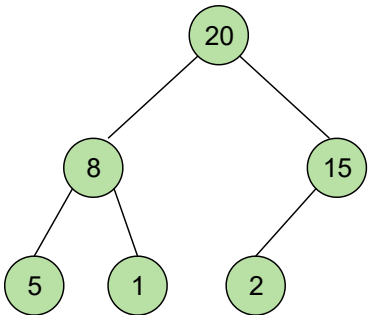


15 > 2,
so swap them





It's a Max Heap!



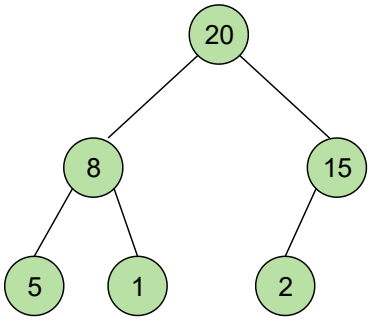
Insertion in a Max-Heap

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
- Then, restore the heap property by swapping parent and child values if the parent key is smaller than the child key. We call this ‘percolating up’.
- Continue to percolate up until the heap property is restored.

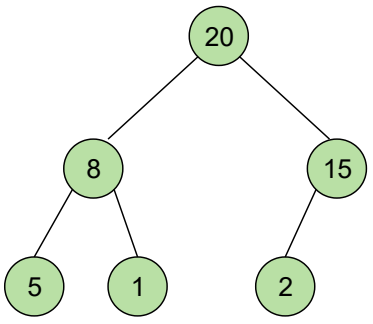
For a clearer picture, here’s a visual representation of inserting in a max heap,

Insert 30!

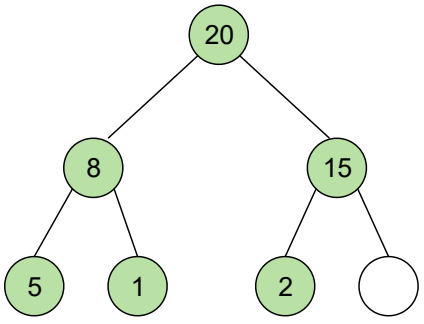


1 of 11

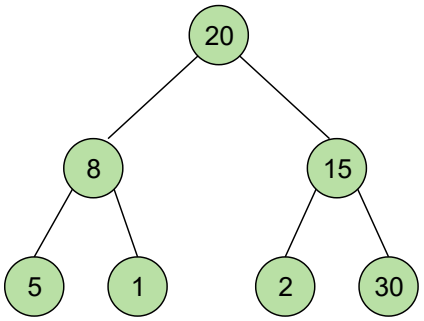
Create a new node at last level near second last child 2

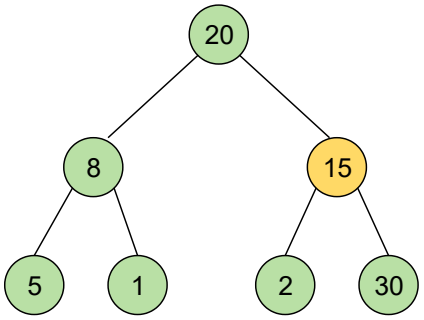


2 of 11

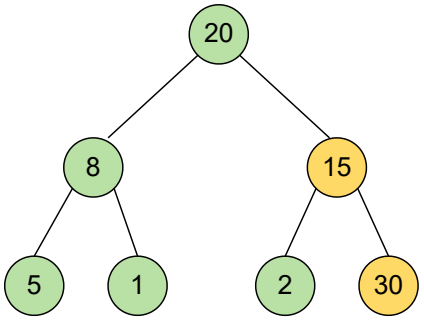


Apply max heap property till root

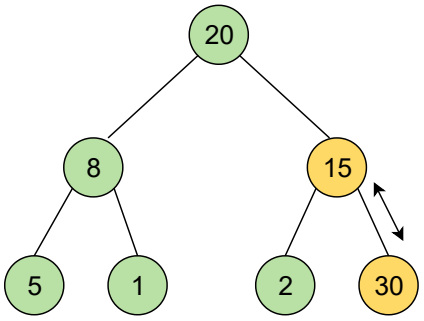




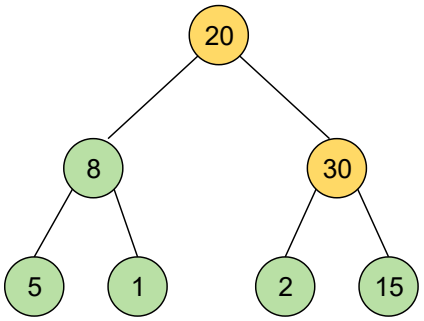
Compare 15 and 30



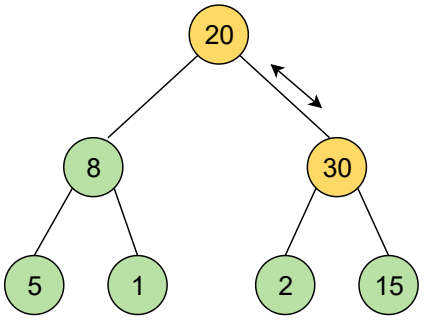
30 > 15,
so swap them

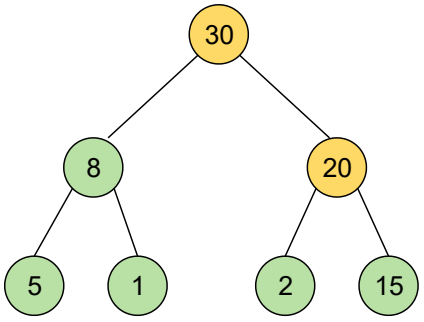


Compare 20 and 30

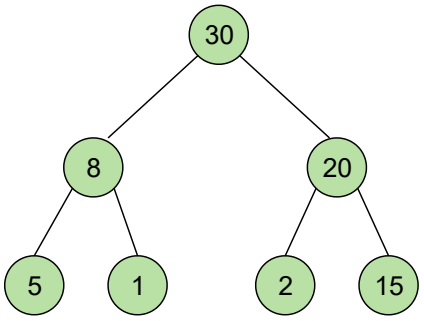


30 > 20,
so swap them





It's a Max Heap!



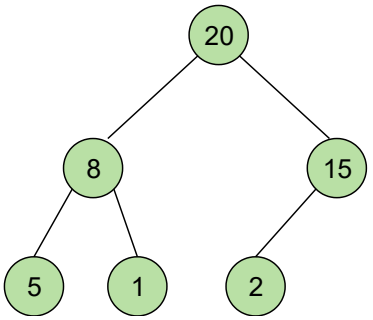
Remove Maximum in a Max Heap

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 last child node at last level to the root
- Now compare the key with its children and if the key is smaller than the key at any of the child nodes, swap values. We call this ‘max heapifying.’
- Continue to max heapify until the heap property is restored.

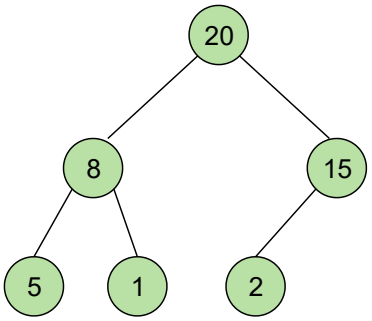
For better understanding, here’s the visual representation of what we just said:

Delete 20!



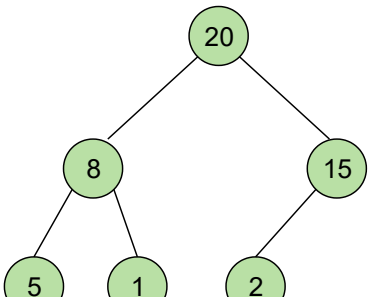
1 of 13

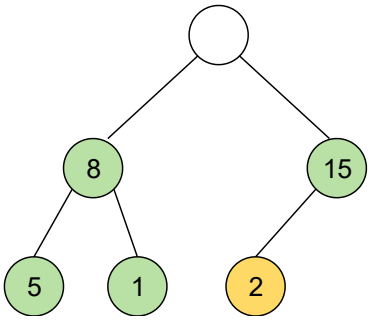
We are going to delete the root node in this case and move the element from last child node to root and then apply heap property till we reach the child node.



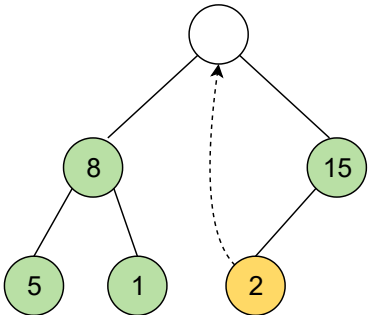
2 of 13

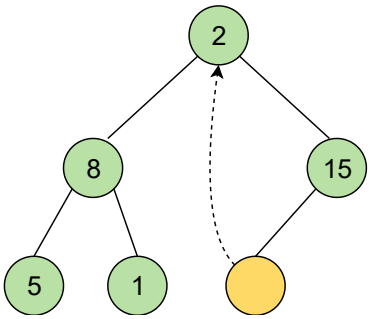
Delete root



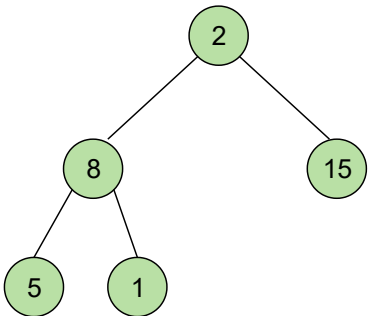


Move 2 to root

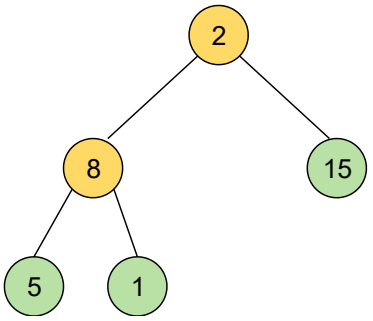




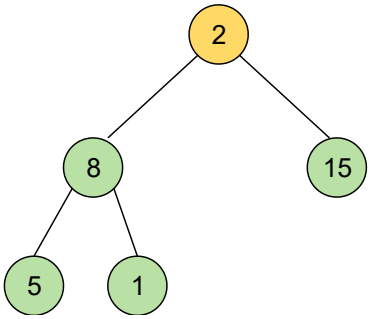
Apply Max Heap property



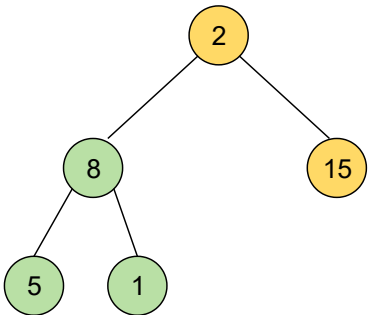
Compare 2 and 8



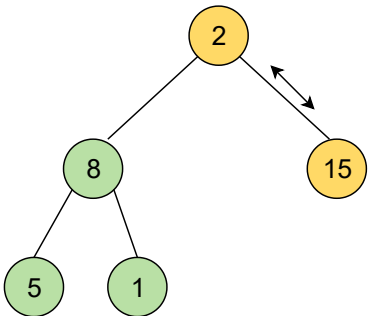
8 > 2,
so we check if right child of 2
is greater than 8

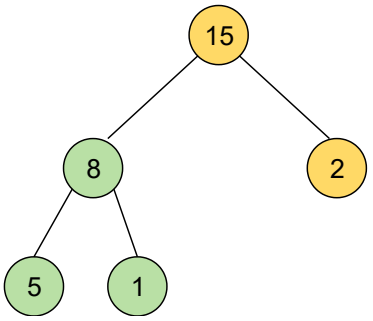


Compare 15 with 2 and 8



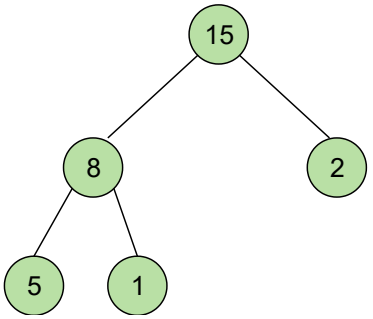
15 > 2 and 15 > 8,
so we swap 2 and 15





12 of 13

It's a Max Heap!



13 of 13

In the next chapter, we will implement a max heap in Python!

Back

What is a Heap?

Next

Max Heap (Implementation)

Mark as Completed

Report an Issue

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