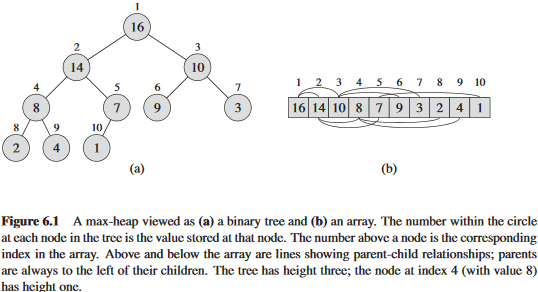
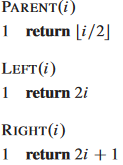
**Heapsort runtime:**

**So wtf is a heap anyway?**

A heap node is an array that, when converted to a binary tree, will be incomplete in the bottom/last row. Not all nodes in the second last row will have 2 children and are not even guaranteed to have 1. However, there must be at least one node in the bottom row, because otherwise the row wouldn’t exist, go figure.

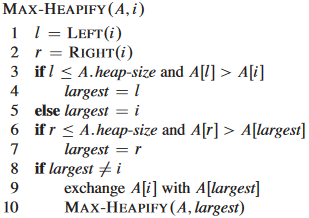
It should be noted that arrays which can be converted into full fledged binary trees can still contain heaps within them. These heaps are typically represented as being a subspace of the original array. In other words, let’s say we have an array with 15 entries which forms a full binary tree. A subarray of that array with only 12 entries would then represent a heap.

**Mathematically navigating heap arrays:**

Since computers keep track of data in a highly mathematical way, mathematical operations are needed to navigate a heap array. Assuming is the index of a node, you can then find it’s parent by dividing it by 2, find it’s leftmost child by multiplying it by 2, and find it’s rightmost child by multiplying by 2 and adding 1.

**Heap properties:**

There are two types of heap trees: Max-heap and min-heap. Max-heap are heap trees which satisfies the condition of all parent nodes being larger than or equal to their children nodes. Min-heap are the opposite, in that parents have to have smaller values than their children.

**Sorting one heap node with pseudo code:**

The function to the right sorts a single node and it’s 2 children. Starts out by defining the indexes of left and right child, then checks if left child has bigger value then the parent, while also ensuring the index of the left child isn’t bigger than the size of the entire heap array, as that would mean it does not exist as a part of the overall heap. It then does the same for the right child, except it compares that to whichever was deemed the largest in the comparison between the left and the parent. If the parent was swapped, it then recursively runs the function on the former parent to see how it holds up with any new children it might have.