

Insert an Element into a Max Heap

Now we will move on to another tree data structure, the binary heap. A binary heap is a partially ordered binary tree which satisfies the heap property. The heap property specifies a relationship between parent and child nodes. You may have a max heap, in which all parent nodes are greater than or equal to their child nodes, or a min heap, in which the reverse is true. Binary heaps are also complete binary trees. This means that all levels of the tree are fully filled and if the last level is partially filled it is filled from left to right.

While binary heaps may be implemented as tree structures with nodes that contain left and right references, the partial ordering according to the heap property allows us to represent the heap with an array. The parent-children relationship is what we're interested in and with simple arithmetic we can compute the children of any parent and the parent of any child node.

For instance, consider this array representation of a binary min heap:

```
[ 6, 22, 30, 37, 63, 48, 42, 76 ]
```

The root node is the first element, 6. Its children are 22 and 30. If we look at the relationship between the array indices of these values, for index i the children are $2 * i + 1$ and $2 * i + 2$. Similarly, the element at index 0 is the parent of these two children at indices 1 and 2. More generally, we can find the parent of a node at any index with the following: $\text{Math.floor}((i - 1) / 2)$. These patterns will hold true as the binary tree grows to any size. Finally, we can make a slight adjustment to make this arithmetic even easier by skipping the first element in the array. Doing this creates the following relationship for any element at a given index i :

Example array representation:

```
[ null, 6, 22, 30, 37, 63, 48, 42, 76 ]
```

An element's left child: $i * 2$

An element's right child: $i * 2 + 1$





An element's parent: $\text{Math.floor}(i / 2)$

Once you wrap your head around the math, using an array representation is very useful because node locations can be quickly determined with this arithmetic and memory usage is diminished because you don't need to maintain references to child nodes.

Instructions: Here we will create a max heap. Start by just creating an `insert` method which adds elements to our heap. During insertion, it is important to always maintain the heap property. For a max heap this means the root element should always have the greatest value in the tree and all parent nodes should be greater than their children. For an array implementation of a heap, this is typically accomplished in three steps:

1. Add the new element to the end of the array.
2. If the element is larger than its parent, switch them.
3. Continue switching until the new element is either smaller than its parent or you reach the root of the tree.

Finally, add a `print` method which returns an array of all the items that have been added to the heap

	The <code>MaxHeap</code> data structure should exist.
	<code>MaxHeap</code> should have a method called <code>insert</code> .
	<code>MaxHeap</code> should have a method called <code>print</code> .
	The <code>insert</code> method should add elements according to the max heap property.