Heapify and Merge

Building a Binary Heap Tree

Inserting a set of n items into an empty Binary Heap Tree is n inserts of complexity $O(\log n)$ giving a total complexity of $O(n \log n)$.

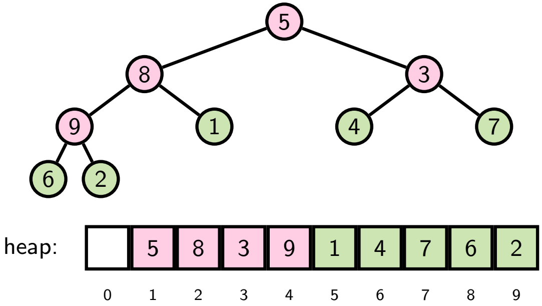
There is a more efficient way: If we have the items in an array in random order (starting at index position 1), then we already have them in Complete Binary Tree form, but not in Binary Heap Tree form. At this point, all the leaf nodes satisfy the heap tree properties, but the internal nodes do not.

We can therefore iterate over the internal nodes, starting with the last internal node and working up to the first, calling `bubbleDown` on each in turn. Each time we do, we ensure that the subtree based on that node becomes a valid Binary Heap Tree, so that in the end, the whole tree is a valid Binary Heap Tree.

Building a Binary Heap Tree

The last node is node n. The parent of the last node is node $n/2$ and that must be the last internal node in the tree, because node $n/2+1$ would have left child $2*(n/2+1) = n+2$, which doesn't exist, so node $n/2+1$ must not be an internal node.

```
public void heapify() {
    for( int i = n/2 ; i > 0 ; i— )
        bubbleDown(i)
}
```



Complexity of Heapify

Where h is the height of the heap tree, the root node needs to swap with h-many nodes, the nodes on level 1 swap with $(h-1)$-many nodes and so on. In the **worst** case, the total number of swaps is

$$\begin{aligned} C(h) &= h + 2(h-1) + 4(h-2) + \ldots + 2^{h-1}(h-(h-1)) \\ &= \sum_{i=0}^{h} 2^i(h-i) = \frac{2^h}{2^h}\sum_{i=0}^{h} 2^i(h-i) = 2^h \sum_{i=0}^{h} \frac{h-i}{2^{h-i}} \\ &= 2^h \sum_{j=0}^{h} \frac{j}{2^j} \leq 2^h \sum_{j=0}^{\infty} \frac{j}{2^j} \end{aligned}$$

and since $\sum_{j=0}^{\infty} \frac{j}{2^j} = 2$ we obtain that $C(h) = 2^h \times 2 = 2^{h+1}$.

Hence the complexity of heapify is $O(2^{h+1}) = O(2^h) = O(n)$

Merging Binary Heap Trees

Three major approaches to merge two BHTs of similar size n:

1. Insert each item of one tree into the other
   - n inserts, hence $O(n \log n)$
2. Remove the last elements of the bigger tree and insert them into the smaller until the tree made from a dummy root node and the smaller and bigger trees as its left and right child respectively, is complete. Then use the standard delete method to delete the dummy root node[1].
   - On average, about half of the leaf nodes of one tree must be inserted into the other, so $O(n)$ inserts, hence complexity is also $O(n \log n)$. However, this will be about $\frac{1}{4}$ of the number of operations of the previous method so faster.
3. Concatenate the array forms and call `heapify`
   - $O(n)$

[1]Note: more complicated with array forms than with pointer based trees