What is a Heap?

A brief introduction to Heaps and their uses. We will also discuss the Heap property and how a heap is implemented.

We'll cover the following

Introduction

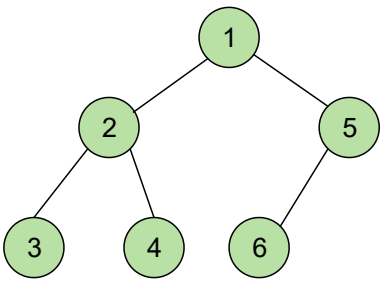
Heaps are advanced data structures that are useful in applications such as sorting and implementing priority queues. They are regular binary trees with two special properties

Heaps must be Complete Binary Trees

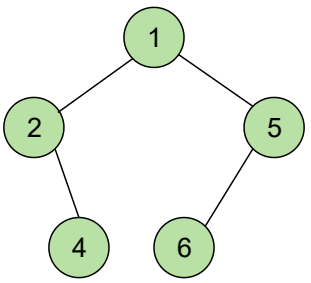
As discussed in the chapter on trees (https://www.educative.io/collection/page/5642554087309312/5634727314718720/5750116644290560), a Complete Binary tree is a tree where each node has at most two children and the nodes at all levels are full, except for the leaf nodes which *can* be empty.

Some Complete Binary Tree Properties:

1. All leaves are either at depth d or depth $d-1$.
2. The leaves at depth d are to the left of the leaves at depth $d-1$
3. There is at most one node with just one child
4. If the singular child exists, it is the left child of its parent
5. If the singular child exists, it is the right most leaf at depth d.



Complete Binary Tree



Incomplete Binary Tree

The nodes must be ordered according to the Heap Order Property

The heap order property is different for the two heap structures that we are going to study in this chapter:

- Min Heap
- Max Heap

Min Heaps are built based upon the *Min Heap property* and Max Heaps are built based upon *Max Heap Property*. Let's see how they are different.

Max Heap Property:

All the parent node keys must be greater than or equal to their child node keys in max-heaps. So the root node will always contain the largest element in the Heap. If Node A has a child node B, then,

$$key(A) >= key(B)$$

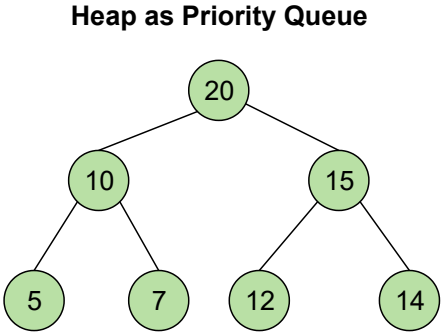
Min Heap Property:

In Min-Heaps, all the parent node keys are less than or equal to their child node keys. So the root node, in this case, will always contain the smallest element present in the Heap. If Node A has a child node B, then:

$$key(A) <= key(B)$$

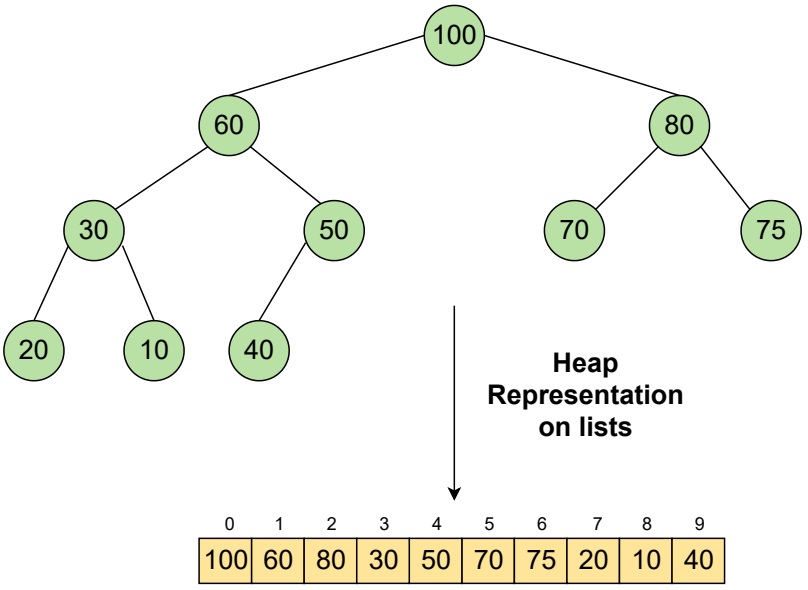
Where are Heaps Used?

The primary purpose of heaps is to return the smallest or largest element. This is because the time complexity of getting the minimum/maximum value from a min/max heap is *O(1)*, i.e., constant time complexity. This way, algorithms that require retrieving the maximum/minimum value can be optimized. Heaps are also used to design *Priority Queues*. Some of the famous algorithms which are implemented using Heaps are Prim's Algorithm (https://en.wikipedia.org/wiki/Prim%27s_algorithm), Dijkstra's algorithm (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) and the famous Heap Sort algorithm (https://en.wikipedia.org/wiki/Heapsort#Algorithm) which is entirely based on the Heap data structure.



Heap Representation in Lists

Heaps can be implemented using arrays or lists in python. The node values are stored such that all the parent nodes occur in the first half of the list (where $index \leq floor(\frac{n-1}{2})$ where n is the last index) and the leaves exist in the rest. So the last parent will be at the $floor(\frac{n-1}{2})$ index. The left child of the node at the kth index will be at the $2k+1$ index and the right child will be at $2k+2$. To put it simply, the index of each node is how much you'd count if you started from 0 at the root and went left to right level wise in a tree. See the figure below to see how nodes are mapped to a list:



As you can see, all the parent nodes are present in the first half of the list and the last parent appears at the floor(n/2th) position. In this case, 'n' is the last or largest index so

$$n = 9$$

$$floor((9-1)/2) = floor(8/2) = floor(4) = 4$$

So the last parent is at the 4th index, the key of which is 50. The children nodes appear on the second half. The following two properties also hold:

$$LeftChild = 2k + 1$$

$$RightChild = 2k + 2$$

Some common misconceptions

Heaps are sometimes called *Binary Heaps* because they are in fact binary trees. Also, the Heap data structure is not the same as heap memory. Furthermore, it is commonly believed that the elements of Heaps are sorted. They are not at all sorted, in fact, the only key condition that a Heap follows is that the largest or smallest element is always placed at the top (parent node) depending on what type of Heap we are using (Min/Max).

In the next chapter, we'll study Max Heaps in detail!