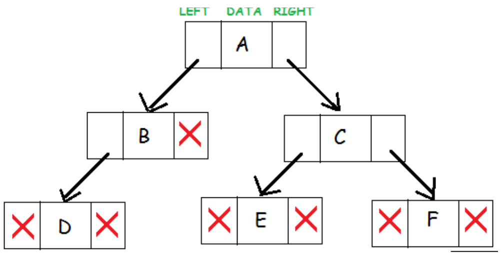
NOTES

A **binary tree** is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.

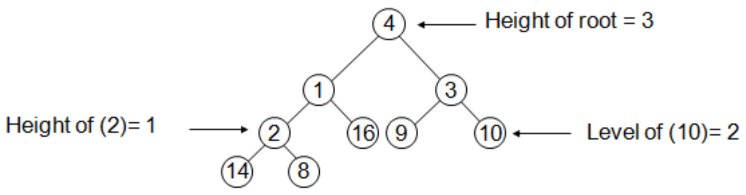
Each node contains three components:

1. Pointer to left subtree
2. Data element
3. Pointer to right subtree



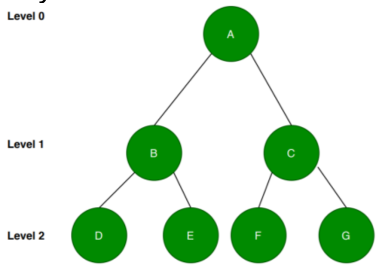
The topmost node in the tree is called the root. An empty tree is represented by None.

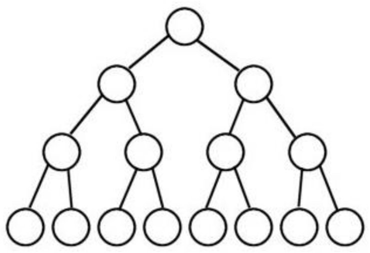
- **Root:** Topmost node in a tree.
- **Parent:** Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent.
- **Child:** A node directly connected to another node when moving away from the root.
- **Leaf/External node:** Node with no children.
- **Internal node:** Node with at least one children.
- **Height of a node:** The number of edges on the longest simple path from the node down to a leaf
- **Level of a node:** The length of a path from the root to the node
- **Height of tree:** Height of root node



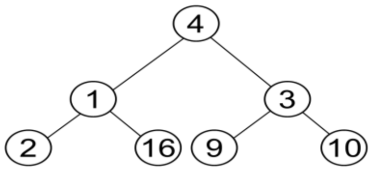
- There are at most 2^L nodes at level (or depth) L of a binary tree
- A binary Tree with height d has at most $2^{d+1} - 1$ nodes.
- A binary tree with n node has height at log n

Full Binary Tree: A Binary Tree is a full binary tree if every node has 0 or 2 children. We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children.

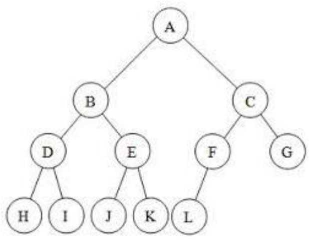




Complete binary tree: A Binary Tree is a complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible



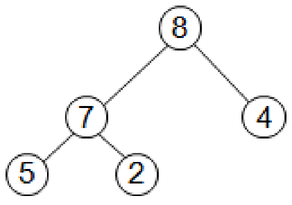
Complete binary tree



Heap

A heap is a nearly complete binary tree with the following two properties:

1. Structural property: All levels are full, except possibly the last one, which is filled from left to right
2. Order (heap) property: for any node x, Parent(x) ≥ x means the root is the maximum element of the heap.



Heap

A binary heap or simply a heap is a complete binary tree where the items or nodes are stored in a way such that the root node is greater than its two child nodes. This is also called **max heap**.

The items in the binary heap can also be stored as **min-heap** wherein the root node is smaller than its two child nodes.

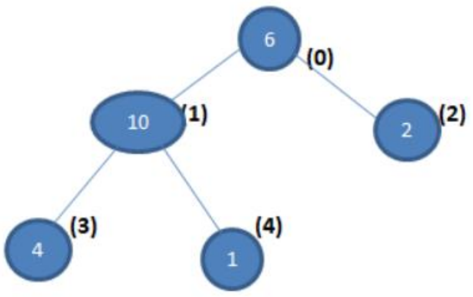
While representing a heap as an list, assuming the index starts at 0, the root element is stored at 0. In general, if a parent node is at the position i, then the left child node is at the position (2* i + 1) and the right node is at (2*i +2).

General algorithm for Heap Sort

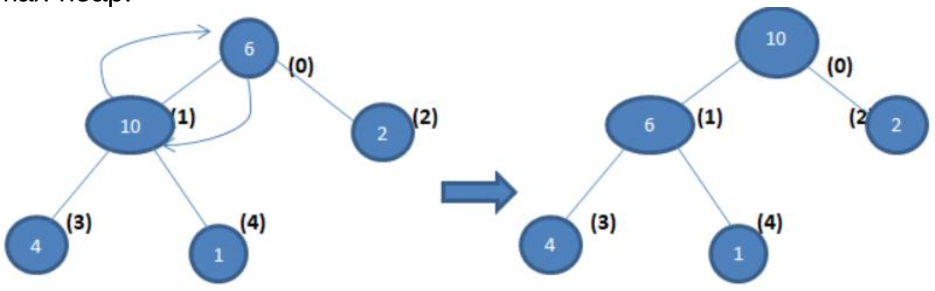
1. Build a max heap from the given data such that the root is the highest element of the heap.
2. Remove the root i.e. the highest element from the heap and replace or swap it with the last element of the heap.
3. Then adjust the max heap, so as to not to violate the max heap properties (heapify).
4. The above step reduces the heap size by 1.
5. Repeat the above three steps until the heap size is reduced to 1.

Creating Heap from Dataset

Say we have a data set 6, 10, 2, 4, 1



In the above tree representation, the numbers in the brackets represent the respective positions in the array. In order to construct a max heap of the above representation, we need to full fill the heap condition that the parent node should be greater than its child nodes. In other words, we need to "heapify" the tree so as to convert it to max-heap.



Program to check of the given list is heap or not

```
def isHeap(arr, i, n):
    if i > int((n - 2) / 2):
        return True
    left = arr[i] >= arr[2 * i + 1] and isHeap(arr, 2 * i + 1, n)
    right = (2*i + 2 == n) or arr[i] >= arr[2 * i + 2] and isHeap(arr, 2 * i + 2, n))
    return left and right

arr = [90, 15, 10, 7, 12, 2, 7, 3]
n = len(arr) - 1

if isHeap(arr, 0, n):
        print("Yes")
else:
        print("No")
```