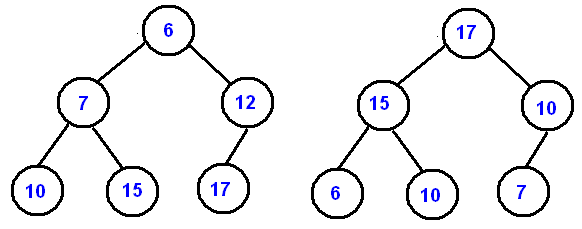
**Data Structure Notes**

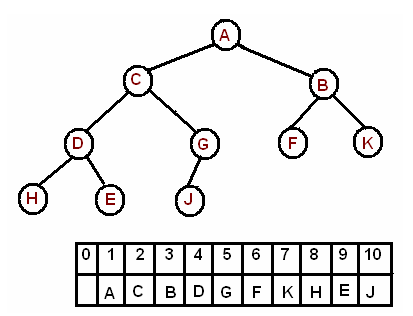
**Binary (MIN/MAX) Heaps:**

* **Introduction:**
  + A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:
    - The *min-heap* property: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
    - The *max-heap* property: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.



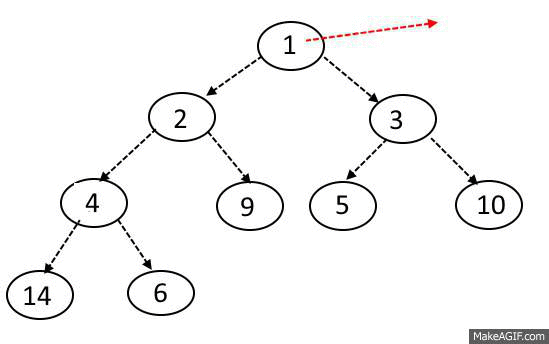
**Figure 1: Example of a Min Binary Heap (left) and a Max Binary Heap (right)**

* **Array Implementation:**
  + We can uniquely represent a complete binary tree with an array. The level order traversal of the tree is stored in the array.
  + Some implementations include leaving the first element of the array open and storing the root element as the second item. Others have the root value as the first element of the array. Either way we can visualize the storing of a binary tree in the examples below.



**Figure 2: Level-Order Binary Tree Array Storing**

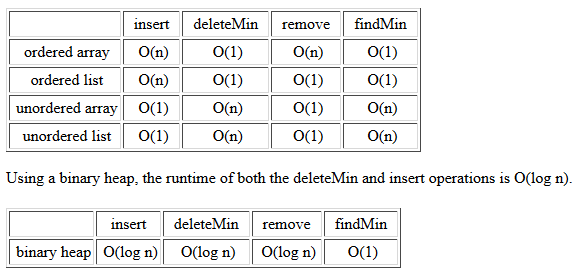
* + Index from 1:
    - Arr[i/2]: returns the **parent** node of index i.
    - Arr[(2\*i)]: returns the **left** child node of index i.
    - Arr[(2\*i) + 1]: returns the **right** child node of index i.
  + Index from 0:
    - Arr[(i – 1)/2]: returns the **parent** node of index i.
    - Arr[(2\*i) + 1]: returns the **left** child node of index i.
    - Arr[(2\*i) + 2]: returns the **right** child node of index i.
* **Insert:**
  + When we insert a new value into the heap, we put that value at the end of the array. Then we compare that value with its parent, if it is larger (for max heap) or smaller (for min heap) than the parent’s value we swap the nodes. If not, we leave it.
* **Extract Minimum:**
  + To Extract/Delete the minimum value from the heap we swap the min with the last element of the array and discard the last element. We have to make sure that we restore the property of the heap, which is known as heapify. We do this with the sink-down method (5).
    - If the parent node is larger than either of the child nodes. Compare the child nodes, which every of the two nodes are smaller swap with the parent node.
    - Proceed down the tree until the parent node is smaller than the child node.

**[](Delete-OR-Extract-Min-from-Heap.gif)**

**GIF: Press ctr + click to see the animation.**

* **Delete A Node:**
  + Find the index for the element to be deleted.
  + Take out the last element from the last level from the heap and replace the index with this element.
  + Perform the Sink-Dow method.
* **Comparative Time and Space Complexity:**





* **Applications:**
  + Priority Queue
  + Heap Sort
  + Graph Algorithms: The priority queues are especially used in Graph Algorithms like Dijkstra’s Shortest Pat and Prim’s Minimum Spanning Tree.
  + K’th Largest Element in an array
  + Sort an almost sorted array.
  + Merge K sorted arrays.
* **References:**

1. [**https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html**](https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html)
2. [**https://www.geeksforgeeks.org/binary-heap/**](https://www.geeksforgeeks.org/binary-heap/)
3. [**https://www.youtube.com/watch?v=WCm3TqScBM8**](https://www.youtube.com/watch?v=WCm3TqScBM8)
4. [**https://algorithms.tutorialhorizon.com/binary-min-max-heap/**](https://algorithms.tutorialhorizon.com/binary-min-max-heap/)