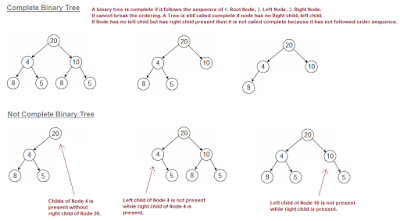
**Heap Sort.**

**Before looking into Heap Sort, let's understand what is Heap and how it helps in sorting.**

**What is Complete Binary Tree?**

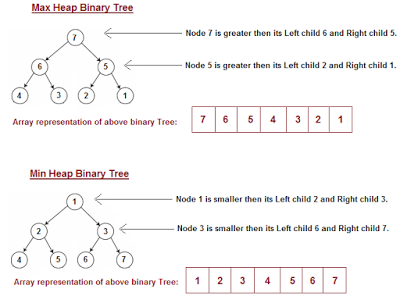
A Complete binary tree is a binary tree in which every node other than the leaves has two children. In complete binary tree at every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

**Let's understand with simple words now,**  
If a Binary Tree is filled level by level, left to right (Left child followed by Right child.) then it is called complete binary tree.   
If Right child is present without Left child then it is not complete.

[](http://3.bp.blogspot.com/-2JT-9xxZU8c/VkmFFcLNNAI/AAAAAAAAAqY/DLMfhTbRXJg/s1600/completeBinaryTree.png)

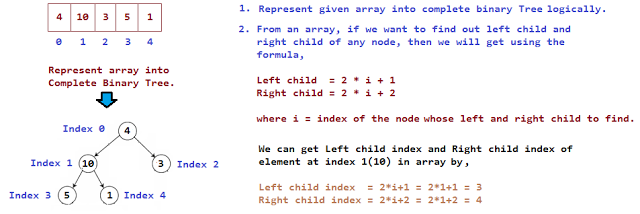
**What is Heap property in Binary Tree?**

A binary Tree is said to follow a heap property if tree is complete binary tree and every element of the tree is Larger (or Smaller) than any of its descendants if they exists.    
  
**Depending on the ordering, a heap is called a max-heap or a min-heap.**  
**In a Max-heap,** the keys of parent nodes are always greater than or  equal to those of the children.   
In max-heap, Largest element of the Tree is always at top(Root Node).  
  
**In a Min-heap,**the keys of parent nodes are less than or equal to those of the children.  
In min-heap, Smallest element of the Tree is always at top(Root Node).

[](http://2.bp.blogspot.com/-b4uSozAkmkQ/VkmLOCYek9I/AAAAAAAAAqs/0WlxmMdQP_U/s1600/Tree-Satisfying-Heap-property.png)

**Important aspects of Heap sort. (Prerequisites)**

Before going into Heapsort algorithm, Let's understand few points,  
  
If we have an array say **[4, 10, 3, 5, 1]**, then we can represent array as complete binary tree  
(start adding nodes from left to right) like shown below.

[](http://2.bp.blogspot.com/-Fi15Em4OZWc/VkmMMX_X4BI/AAAAAAAAAqw/K9134vVFWDA/s1600/heapsort.png)

**Each element has left and right child present in array except for leaf nodes, but how to find left and right child of non-leaf nodes in array.**

***We will get left and right child of non leaf elements using formula,******Left child index   = 2 \* (index of root, whose left and right child to find) + 1******Right child index = 2 \* (index of root, whose left and right child to find) + 1***

**Left child and Right child of element at index 0 (element 4) is,**  
Left child index  = 2 \* i + 1   = 2 \* 0 + 1   = 1    
Right child index = 2 \* i + 2   = 2 \* 0 + 2   = 2  
  
**Left child and Right child of element at index 1 (element 10) is,**  
Left child index  = 2 \* i + 1   = 2 \* 1 + 1   = 3    
Right child index = 2 \* i + 2   = 2 \* 1 + 2   = 4  
  
**Left child and Right child of element at index 2 (element 3) is,**  
Left child index  = 2 \* i + 1   = 2 \* 2 + 1   = 5   
(index 5 is greater than length of array, so element 3 has no left child)    
   
Right child index = 2 \* i + 2   = 2 \* 2 + 2   = 6   
(index 6 is greater than length of array, so element 3 has no right child)

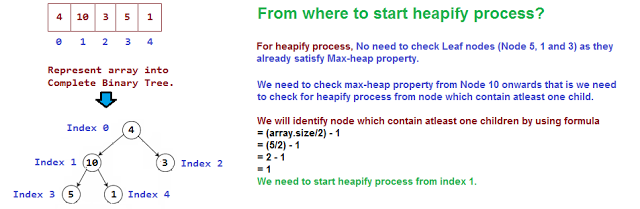
**Algorithm**

**STEP 1:**  Logically, think the given array as Complete Binary Tree,   
 **STEP 2:**For sorting the array in ascending order, check whether the tree is satisfying Max-heap   
               property at each node,   
               (For descending order, Check whether the tree is satisfying Min-heap property)   
               Here we will be sorting in Ascending order,  
 **STEP 3:** If the tree is satisfying Max-heap property, then largest item is stored at the root of the heap.   
               (At this point we have found the largest element in array, Now if we place this element at  
               the end(nth position) of the array then 1 item in array is at proper place.)  
               We will remove the largest element from the heap and put at its proper place(nth position) in  
               array.  
     
              **After removing the largest element, which element will take its place?**  
              We will put last element of the heap at the vacant place. After placing the last element at the  
              root, The new tree formed may or may not satisfy max-heap property.  
              So, If it is not satisfying max-heap property then first task is to make changes to the tree, So   
              that it satisfies max-heap property.  
              
              (**Heapify process:** The process of making changes to tree so that it satisfies max-heap   
               property is called **heapify**)  
  
              When tree satisfies max-heap property, again largest item is stored at the root of the heap.      
              We will remove the largest element from the heap and put at its proper place(n-1 position) in  
              array.  
     
              **Repeat step 3 until size of array is 1 (At this point all elements are sorted.)**

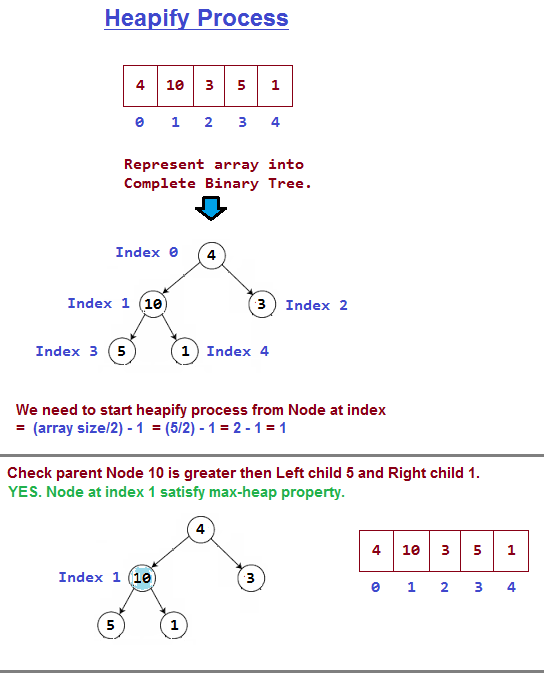
**Heapify Process with Example**

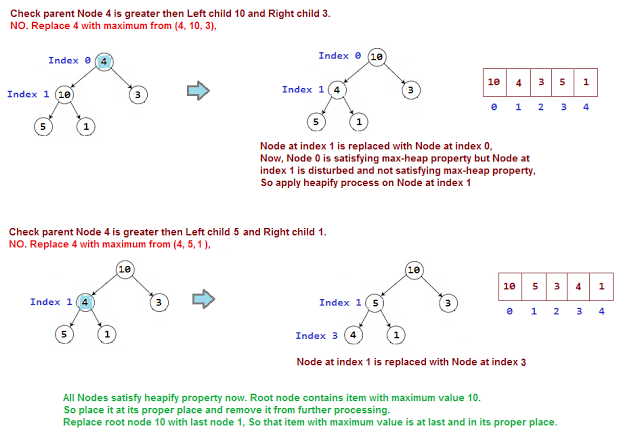
**Heapify process checks whether item at parent nodes has larger value than its left and right child.**  
  
If parent node is not largest compared to its left and right child, then it finds the largest item among parent, its left and right child and replaces largest with parent node.  
  
**It repeat the process for each node and at one point tree will start satisfying max-heap property.**  
**At this point, stop heapify process and largest element will be at root node.**  
  
We found the largest element, Remove it and put it at its proper place in array,   
Put the last element of the tree at the place we removed the node(that is at root of the tree)   
Placing last node at the root may disturbed the max-heap property of root node.   
**So again repeat the Heapify process for root node. Continue heapify process until all nodes in tree satisfy max-heap property.**  
 **Initially, From which node we will start heapify process? Do we need to check each and every node that they satisfy heap property?**  
We do not have to look into leaf nodes as they don't have children and already satisfying max-heap property.  
So, we will start looking from the node which has at least one child present.  
  
**How we will get that item in array, which has at least one child present?**

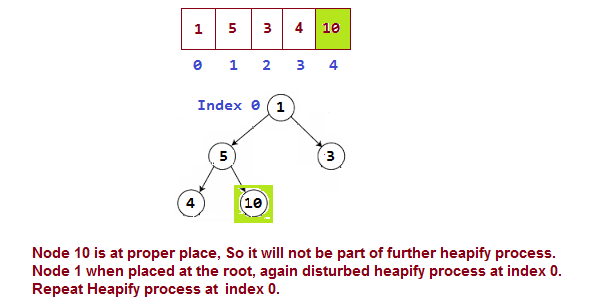
***By using the formula (array.length/2) - 1, we will be able to get the index of the item to start Heapify process.***

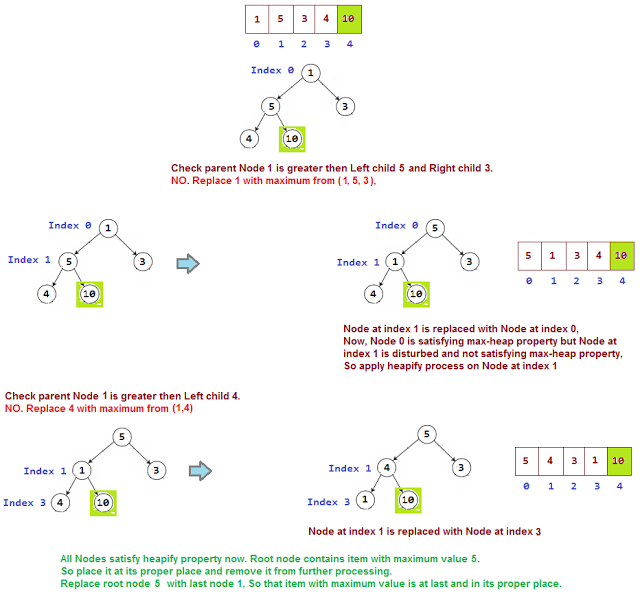
[](http://4.bp.blogspot.com/-KtBerk_cPjs/Vkmx-kmArYI/AAAAAAAAArA/11kzjNIbb8U/s1600/heapify-process-start-node.png)

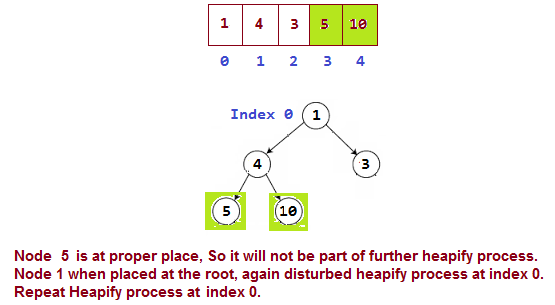
**Lets understand Heapify process with help of an example.**

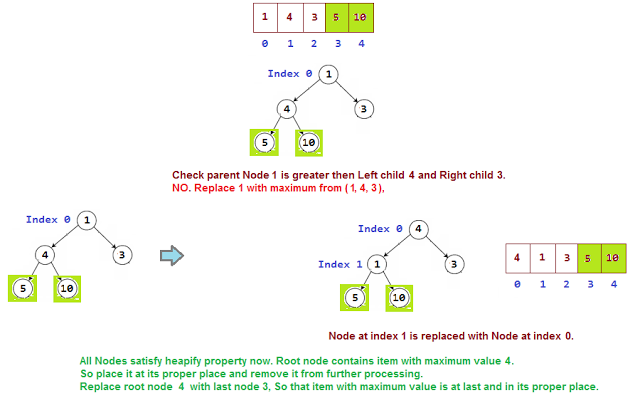
**[](http://3.bp.blogspot.com/-e_CNlxHj5Bs/Vkoe738GAjI/AAAAAAAAArg/KmIbfAE3hKw/s1600/Heapify-process-Step1-2.png)**

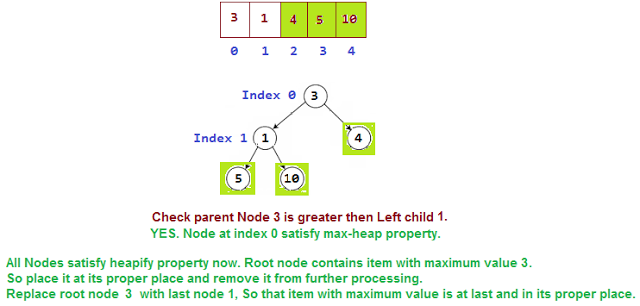
[](http://3.bp.blogspot.com/-qZtycYGmrbY/VkofGHecdII/AAAAAAAAAro/lGEdqlVd9uc/s1600/Heapify-process-Step3.png)

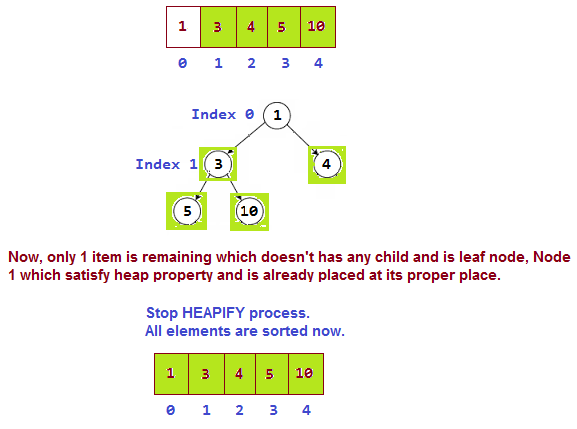
[[](http://3.bp.blogspot.com/-yHSl9-mCSrY/VkofN3NUnUI/AAAAAAAAArw/HXZvdD3qEAo/s1600/Heapify-process-Step4.png)](http://3.bp.blogspot.com/-yHSl9-mCSrY/VkofN3NUnUI/AAAAAAAAArw/HXZvdD3qEAo/s1600/Heapify-process-Step4.png)

[](http://1.bp.blogspot.com/-br1LdT46SBI/VkofV7gQnxI/AAAAAAAAAr4/jqWbKvGw1z4/s1600/Heapify-process-Step5.png)

[](http://3.bp.blogspot.com/-Tfhv3D4hNcc/VkofljCSrfI/AAAAAAAAAsA/rogcIBeV-J0/s1600/Heapify-process-Step6.png)

[](http://1.bp.blogspot.com/-A9sjW-cSW_E/VkofqTdiGrI/AAAAAAAAAsI/ROkePgWvf9s/s1600/Heapify-process-Step7.png)

[](http://2.bp.blogspot.com/-ivFdfHu3RFU/Vkofv9RSPzI/AAAAAAAAAsQ/ZKC567H4cDI/s1600/Heapify-process-Step8.png)

[](http://2.bp.blogspot.com/-qM3EncOJIvE/Vkof0yip2sI/AAAAAAAAAsY/fB-WSLOX63s/s1600/Heapify-process-Step9.png)

### Summarize Heap Sort algorithm.

**1.** We build a heap(Max or Min) from the given array elements.  
**2.** The root is the max (or min number). So extract it and put it in an array at its proper position.  
**3.** Put last element at the root of the tree and Heapify the remaining elements.  
**4.** Again extract the root and repeat heapification until there is one element in array.

### Advantage of using Heap Sort algorithm for Sorting

**1.** Heap sort has the best possible worst case running time complexity of **O(n Log n).**  
**2.** It **doesn't need any extra storage** and that makes it good for situations where array size is large.