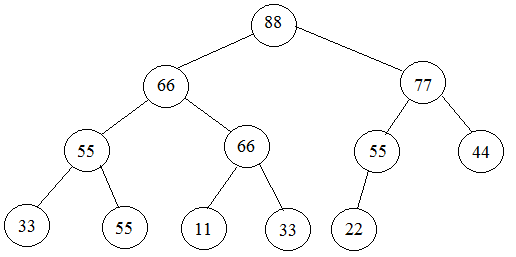
**Advanced Sorting Algorithms**

**Heap Sort**

A heap is a binary tree that has two properties:

* The value of each node is greater than or equal to the values stored in each of its children.
* The tree is perfectly balanced, and the leaves in the last level are all in the leftmost positions.

A property of the heap data structure is that the largest element is always at the root of the tree. A common way of implementing a heap is to use an array. A binary tree is said to have the heap property if the elements along any path from root to leaf are non-increasing. A heap is a complete binary tree that has the heap property.

**Example: a heap**

This binary tree shown here has the heap property. It has six root to leaf paths, one for each leaf: [88-66-55-33], [88-66-55-55], [88-66-66-11], [88-66-66-33], [88-77-55-22] and [88-77-44]. Each one is non-increasing (i.e., x>=y if y follows x in the sequence).

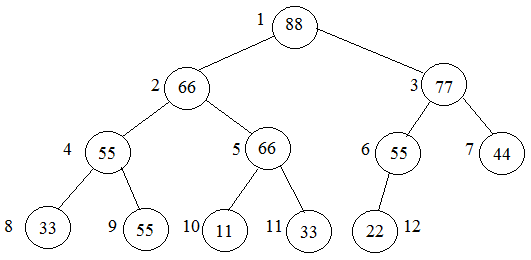
Note that the heap property is consistent with ordinary family trees of people: each child is younger than his/her parent.

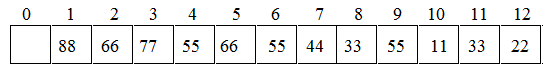
In a heap, every element is the maximum value of all the elements in its subtree. This is because every element y in the subtree rooted at a given element x is in a path from x to a leaf. That path can be extended back up to the root of the whole tree. In that path, y follows x. so by the heap property, x >= y.

In a heap, the largest element is at the root, and the smallest element is at some leaf. The word “heap” is also frequently used to describe the collection of unallocated bytes in memory. Sometimes it is also called free store.

By definition a heap is a complete binary tree. This means that a heap can be stored naturally in an array.

**A heap stored in an array**

Here is the heap from the above example. It is stored in an array of 13 elements: note that the heap property is easy to see in the tree structure, but not clear at all in the linearized array.

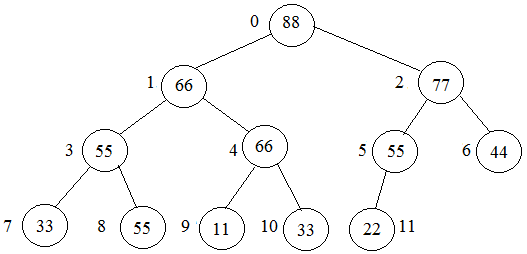


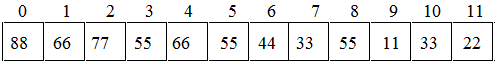
**The heap sort**

A heap is by definition, partially sorted, because each linear string from root to leaf is sorted. This leads to an efficient general sorting algorithm called the **Heap Sort.** As with all sorting algorithms, it applies to an array (or vector). But the underlying heap structure which the array represents is used to define this algorithm.

Like the merge and quick sort, the heap sort uses an auxiliary function which is called the sort() function. And also like the merge sort and quick sort, the heap sort has complexity function of O(nlogn). But unlike the merge sort and the quick sort, the heap sort is not recursive.

The natural mapping of a complete binary tree into an array maps the n elements of the tree into array elements a[1] througha[n]. but to be consistent with all our other sorting algorithms, the n elements should be stored in positions a[0:n-1] in the array. So for this reason we modify the natural mapping so that the correspondence looks like this.

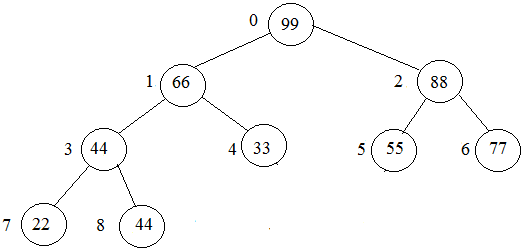


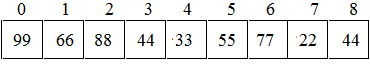


The only consequences of this modifications are that now:

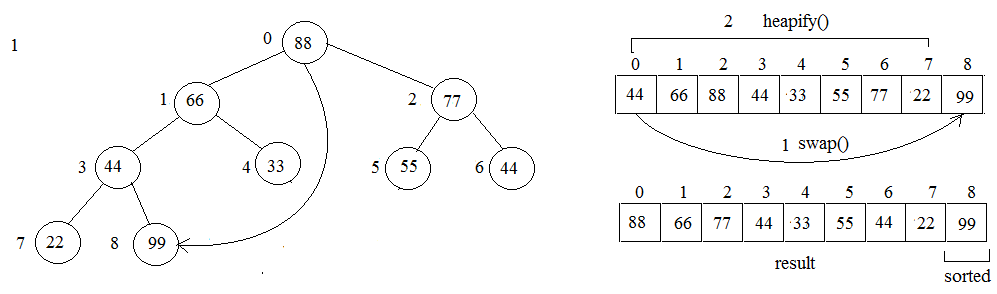
* The children of element k are elements 2k+1 and 2k+2
* The parent of element k is element (k-1)/2

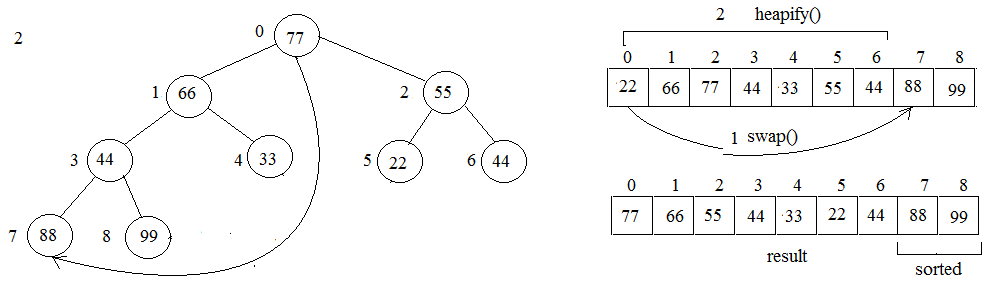
The sort() function first converts the array so that its underlying complete binary tree is transformed into a heap. This is done by applying the heapify() function to each nontrivial subtree. The nontrivial subtrees(i.e., those having more than one element) are the subtrees that are rooted above the leaf level. In the array, the leaves are stored at positions a[n/2] through a[n]. So the first for loop in the sort() function applies the heapify() function to elements a[n/2 – 1] back through a[0] (which is the root of the underlying tree). The result is an array whose corresponding tree has the heap property:

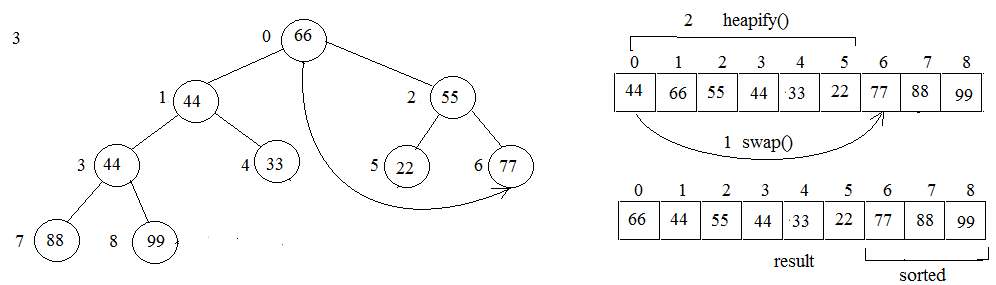


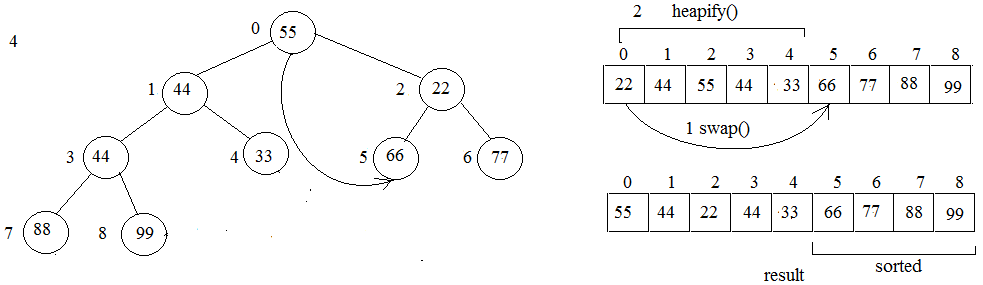


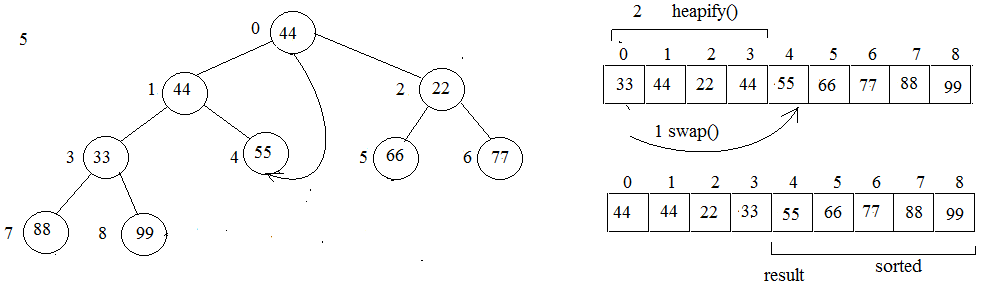
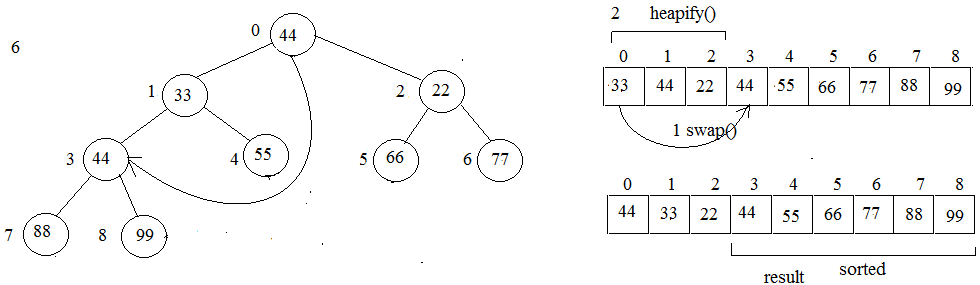
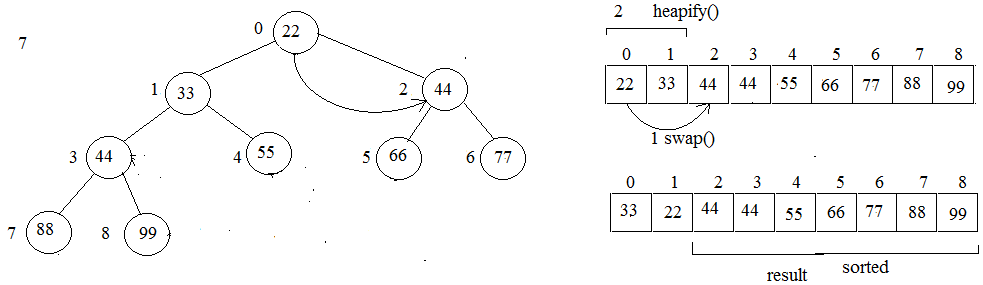
Now the main (second) for loop progresses through n-1 iterations. Each iteration does two things: it swaps the root element with element a[i], and then it applies the heapify() function to the subtree of elements a[0:i-1]. That subtree consists of the part of the array that is still unsorted. Before the swap() executes on each iteration, the subarray a[0:i] has the heap property, a[i] is the largest element in that subarray. That means that the swap() puts element a[i] in its correct position.

The first seven iterations of the main for loop have the effect shown by the seven pictures on the next two pages. The array (and its corresponding imaginary binary tree) is portioned into two parts: the first part is the subarraya[0:i-1] that has the heap property, and the second part is the remaining a[i:n-1] whose elements are in their correct positions. The second part is shaded in each of the seven pictures below. Each iteration of the main for loop decrements the size of the first part and increments the size of the second part. So when the loop has finished, the first part is empty and the second (sorted) part constitutes the entire array. This analysis verifies that the Heap Sort works:







finally after swaping 22 and 33, the list will be sorted.

The complete heap sort code using c++ is shown below

The heap sort complete code in C++

