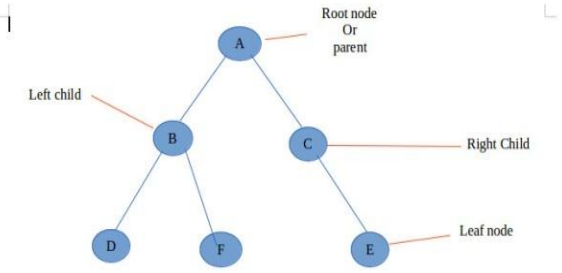
Introduction to tree data structures

A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree is a structure consisting of one node called the root or parent and zero or one or more children.



1- Introductio to sorting:

The term sorting came into picture, as humans realised the importance of searching quickly.
There are so many things in our real life that we need to search for, like a particular record in database, roll numbers in merit list, a particular telephone number in telephone directory, a particular page in a book etc. All this would have been a mess if the data was kept unordered and unsorted, but fortunately the concept of sorting came into existence, making it easier for everyone to arrange data in an order, hence making it easier to search.

Sorting arranges data in a sequence which makes searching easier.

Since the beginning of the programming age, computer scientists have been working on solving the problem of sorting by coming up with various different algorithms to sort data.

The two main criteria as to judge which algorithm is better than the other have been:

1. Time taken to sort the given data.
2. Memory Space required to do so.

2-Different Sorting Algorithms: There are many different techniques available for sorting, differentiated by their efficiency and space requirements. Following are some sorting techniques:

1-Heap Sort 2-Bubble Sort 3-Insertion Sort 4-Selection Sort 5-Quick Sort 6- Merge Sort

2.1Heap Sort Algorithm

Heap Sort is one of the best sorting methods. Heap sort involves building a **Heap** data structure from the given array and then utilizing the Heap to sort the array.

What is a Heap?

Heap is a special tree-based data structure that satisfies the following special heap properties:

1-Shape property represents all the nodes or levels of the tree are fully filled. Heap data structure is a complete binary tree.

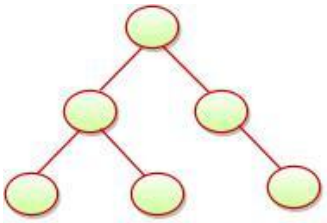


Fig. Complete-Binary Tree

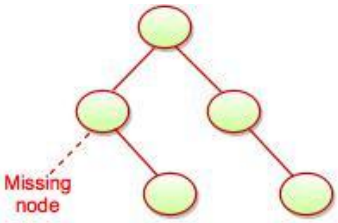


Fig. In-Complete-Binary Tree

2. Heap property is a binary tree with special characteristics. It can be classified into two types:

I. Max Heap[Descending heap]: If the parent nodes are greater than their child nodes, it is called a **Max-Heap.**

II. Min Heap[Ascending heap]: If the parent nodes are smaller than their child nodes, it is called a **Min-Heap.**

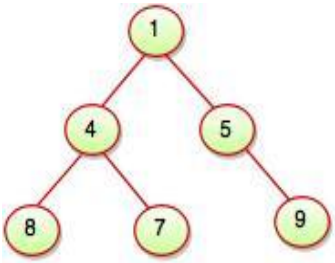


Fig. Min Heap

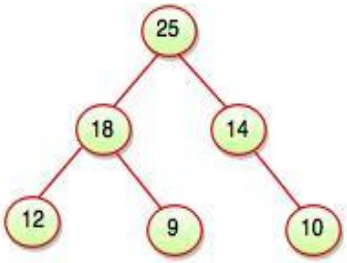


Fig. Max Heap

2.2How Heap Sort Works?

Heap sort algorithm is divided into two basic parts:

- Creating a Heap of the unsorted list/array.
- Then a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal.

2.3Complexity Analysis of Heap Sort

*Average Time Complexity: **O (n*log n)** ***Space Complexity: **O (1)**

- Heap sort is not a Stable sort, and requires a constant space for sorting a list.
- Heap Sort is very fast and is widely used for sorting.

2.4 Pseudocode of Heap Sort

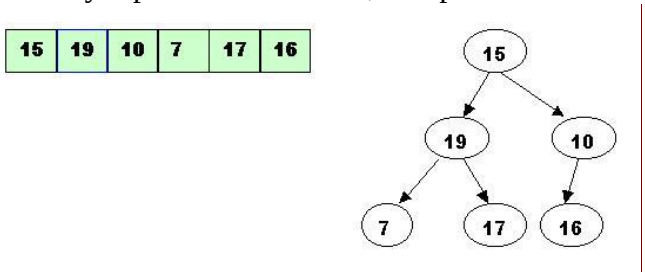
```
MaxHeapify(A, i)
   l = left(i)
   r = right(i)
   if l <= heap-size[A] and A[l] > A[i]
      then largest = l
      else largest = i
   if r <= heap-size[A] and A[r] > A[largest]
      then largest = r
   if largest != i
      then swap A[i] with A[largest]
      MaxHeapify(A, largest)
end func
BuildMaxHeap(A)
   heap-size[A] = length[A]
   for i = |length[A]/2| downto 1
      do MaxHeapify(A, i)
end func
HeapSort(A)
   BuildMaxHeap(A)
   for i = length[A] downto 2
      do swap A[1] with A[i]
         heap-size[A] = heap-size[A] – 1
         MaxHeapify(A, 1)
end func
```

2.5An Example of Heap sort: Given an array of 6 elements: 15, 19, 10, 7, 17, 16, sort it in ascending order using heap sort.

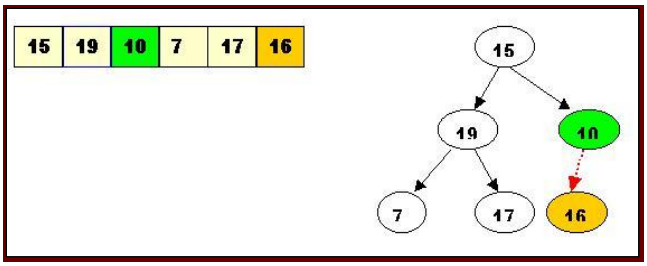
Steps: Here is the array: 15, 19, 10, 7, 17, 16

A. Building the heap tree

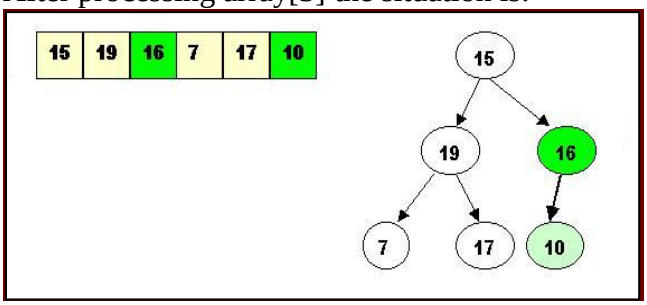
The array represented as a tree, complete but not ordered:



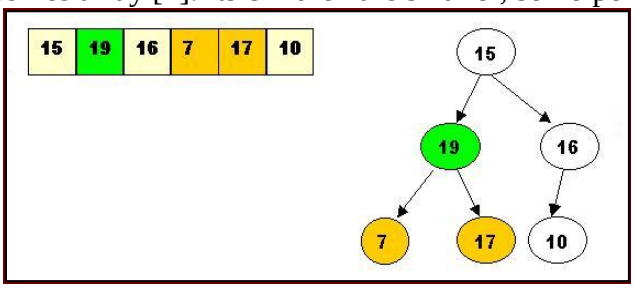
Start with the rightmost node at height 1 - the node at position 3 = Size/2.
It has one greater child and has to be percolated down:



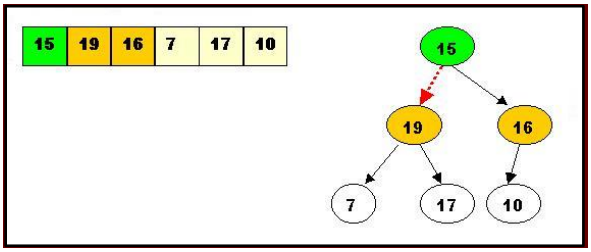
After processing array[3] the situation is:



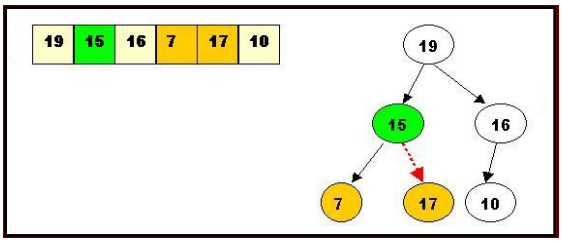
Next comes array [2]. Its children are smaller, so no percolation is needed.



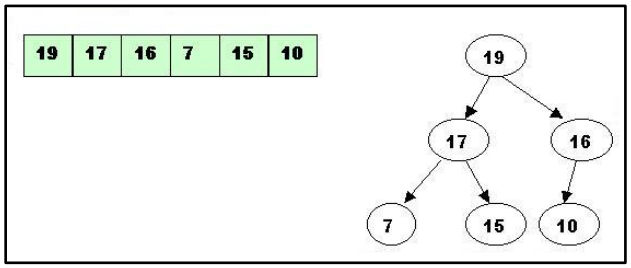
The last node to be processed is array [1]. Its left child is the greater of the children.
The item at array [1] has to be percolated down to the left, swapped with array [2].



As a result the situation is:



The children of array[2] are greater, and item 15 has to be moved down further, swapped with array[5].

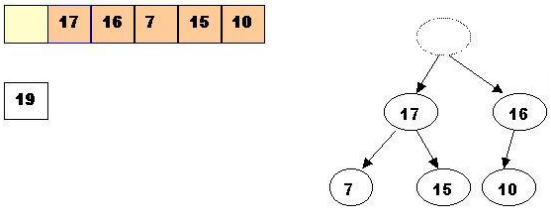


Now the tree is ordered, and the binary heap is built.

B. Sorting - performing deleteMax operations:

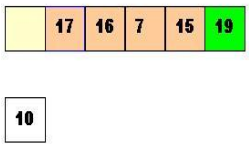
1. Delete the top element 19.

1.1. Store 19 in a temporary place. A hole is created at the top

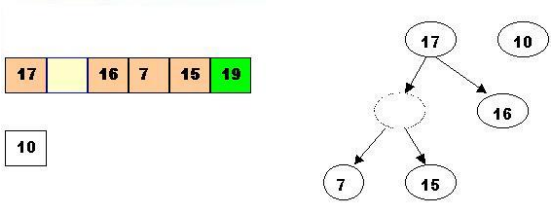


1.2. Swap 19 with the last element of the heap.

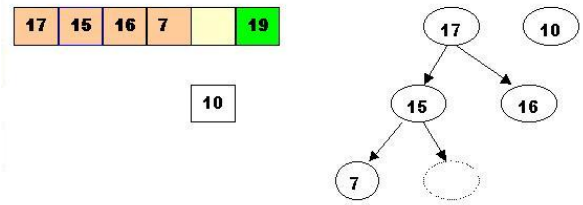
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a cell from the sorted array



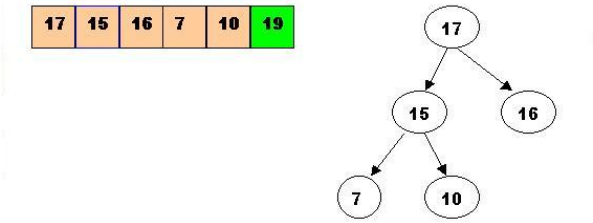
1.3. Percolate down the hole



1.4. Percolate once more (10 is less than 15, so it cannot be inserted in the previous hole)

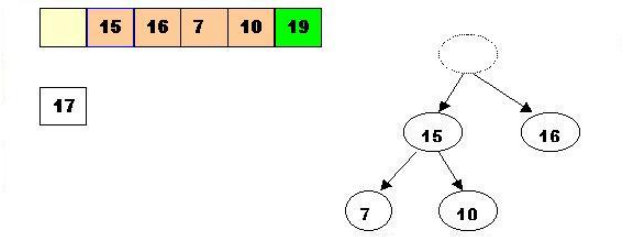


Now 10 can be inserted in the hole



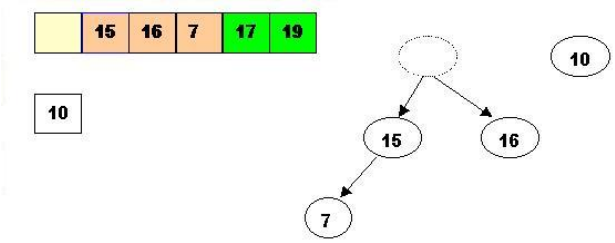
2. DeleteMax the top element 17

2.1. Store 17 in a temporary place. A hole is created at the top

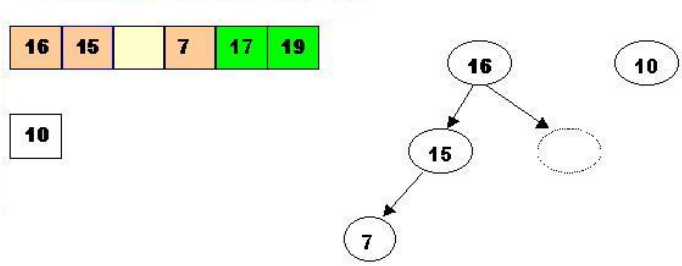


2.2. Swap 17 with the last element of the heap.

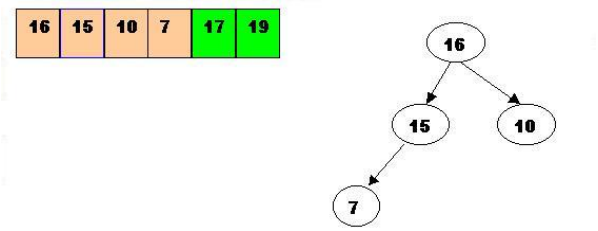
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a cell from the sorted array



2.3. The element 10 is less than the children of the hole, and we percolate the hole down:

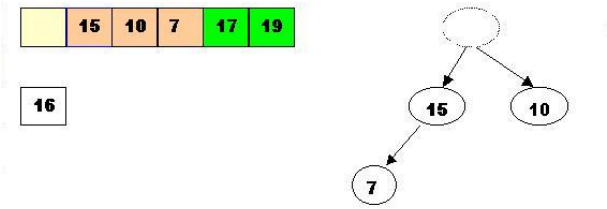


2.4. Insert 10 in the hole



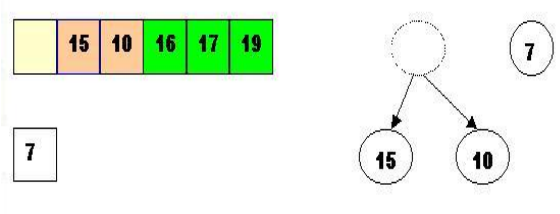
3. DeleteMax 16

3.1. Store 16 in a temporary place. A hole is created at the top

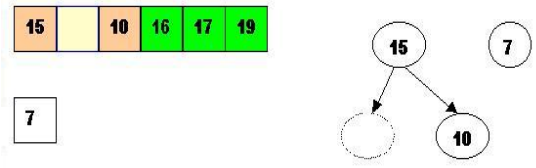


3.2. Swap 16 with the last element of the heap.

As 7 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a cell from the sorted array



3.3. Percolate the hole down (7 cannot be inserted there - it is less than the children of the hole)



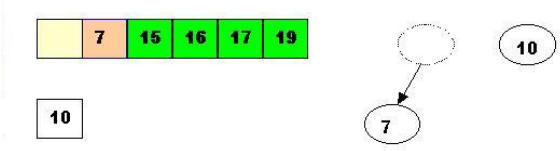
3.4. Insert 7 in the hole

4. DeleteMax the top element 15

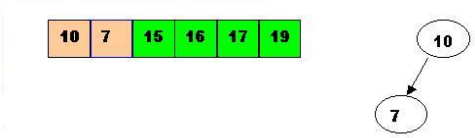
4.1. Store 15 in a temporary location. A hole is created.

4.2. Swap 15 with the last element of the heap.

As 10 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a position from the sorted array

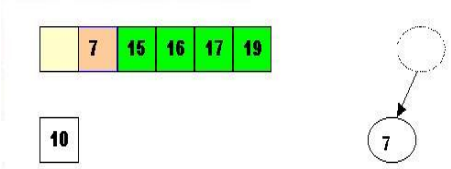


4.3. Store 10 in the hole (10 is greater than the children of the hole)



5. DeleteMax the top element 10.

5.1. Remove 10 from the heap and store it into a temporary location.



5.2. Swap 10 with the last element of the heap.

As 7 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a cell from the sorted array

5.3. Store 7 in the hole (as the only remaining element in the heap

7 is the last element from the heap, so now the array is sorted