HEAP SORT

INTRODUCTION

In heap sort, the given elements are first arranged in a heap. Then the elements are removed one by one. After every deletion, the elements are reheapified. If the heap in question is a max-heap, then we get the elements in the descending order. In the case of a min-heap, the output is an ascending sequence.

The following algorithm presents the formal procedure of heapsort.

Algorithm 6.3 Heapsort

Input: A list of elements
Output: Sorted elements
Strategy: Discussed above
Heapsort (List elements) returns sorted_list

```
      {
      heap h=heapify (elements);
//the elements are heapified and inserted into a heap namely h
      i=0;
      while(i!=n)
            {
            x=delete(
//the delete function removes the element at the root of the heap and inserts
it into x
            insert(sorted_list, x);
//the element x is inserted into sorted_list
            }
       }
```

Complexity: As explained earlier, the algorithm requires heapify. The worst-case complexity of heapify is $(\log(n))$; therefore, the complexity of the algorithm is $O(n \log n)$.

Analysis of Heap sort Algorithm:

- In-place sorting algorithm – memory efficient

- Time complexity – O (n log (n))

- 1st Step- Build heap, O (n) time complexity

- 2nd Step – perform n delete Max operations, each with O (log (n)) time complexity

Total time complexity = O (n log (n))

- Fast sorting algorithm, memory efficient, especially for very large values of n.

- **Slower of the O (n log(n)) sorting algorithms**

The binary heap data structure is an array that can be viewed as a complete binary tree. Each node of the binary tree corresponds to an element of the array. The array is completely filled on all levels except possibly lowest.

An Array A that presents a heap with two attribute:

- Length [A]: the number of elements in the array.
- heap- size [A] : the number of elements in the heap stored with array A.
- Length [A] ≥ heap-size [A].

To convert the heap tree into a heap array as shown in the figure below , the root of the tree A[0] and given index i of a node , the indices of its parent , left child and right child can be computed as :

- A[0] is the root of the tree
- The PARENT (i) is at [(i-1)/2] if i≠ 0 .
- The left child LEFT (i) is at [2i+1]
- The right child RIGHT(i) is at [2i+2]

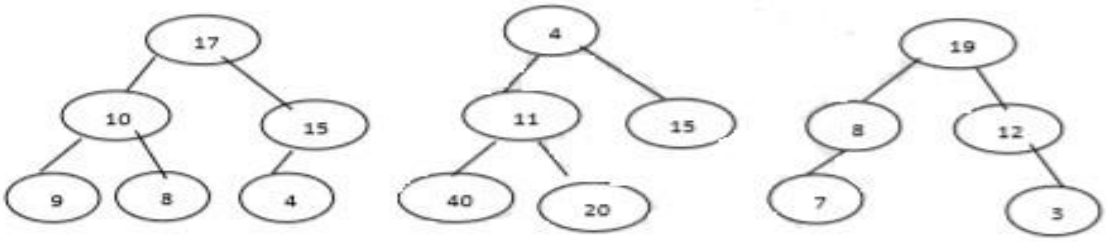
Complete Binary Trees , and Heap

Array representation:

0	1	2	3	4	5	6	7	8	9
50	24	30	20	21	18	3	12	5	6

Heap Sort is the one of the most efficient comparison-based algorithms, is a sorting algorithm that works by first organizing the data to be sorted into a binary heap data structure (uses a heap as its data structure).

Example: which of the following trees is max-heap, min-heap ,non heap



Max-heap min-heap non-heap

Step by step Example:

The heap sort algorithm consists of procedure :

BuildHeap (A)

Build a heap out of array A , the procedure BuildHeap goes through the remaining nodes of the tree and runs MaxHeapify procedure on each one.

MaxHeapify(A,i) or MinHeapify(A,i)

Make the sub tree of A starting in node i fulfill the heap property (MaxHeapify picks the largest child and compare it to the parent . If parent is larger than MaxHeapify quits , otherwise it swaps the parent with the largest child . So that the parent is now becomes larger than its children).

HeapSort(A)

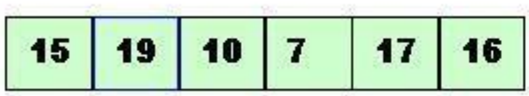
Make A heap , then take out the root , repeat until the array is sorted.

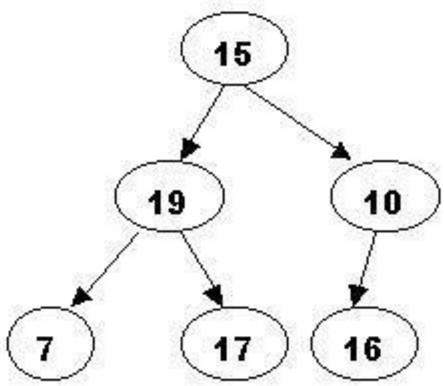
Example:

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

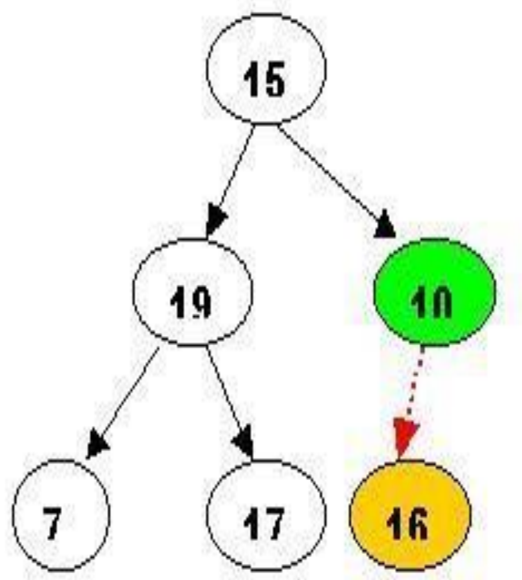
A. Building the heap tree

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

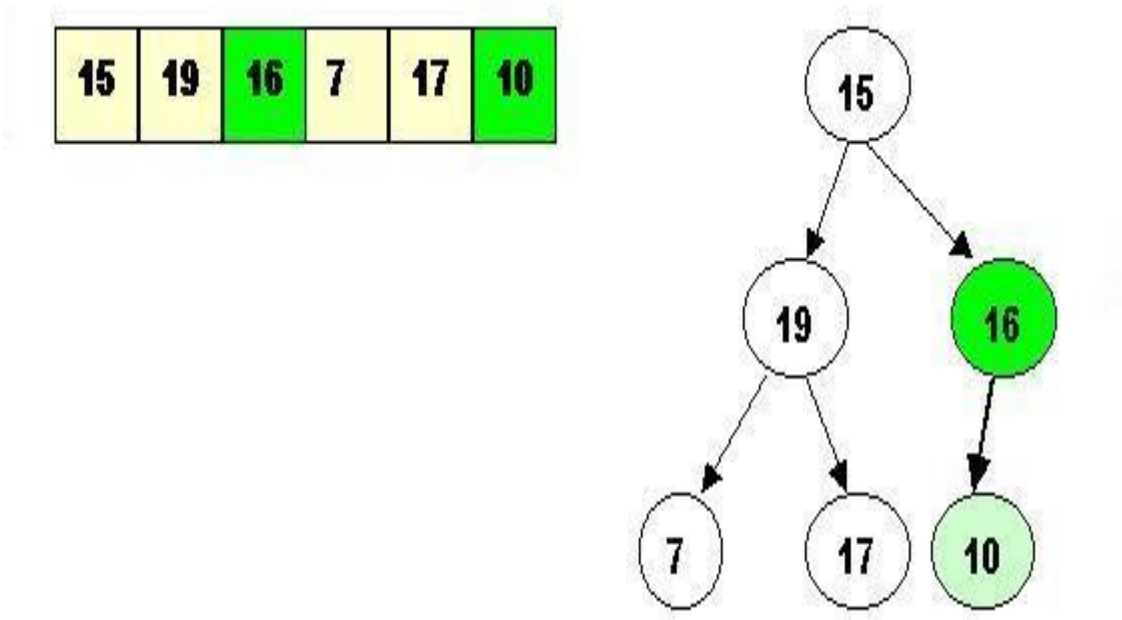




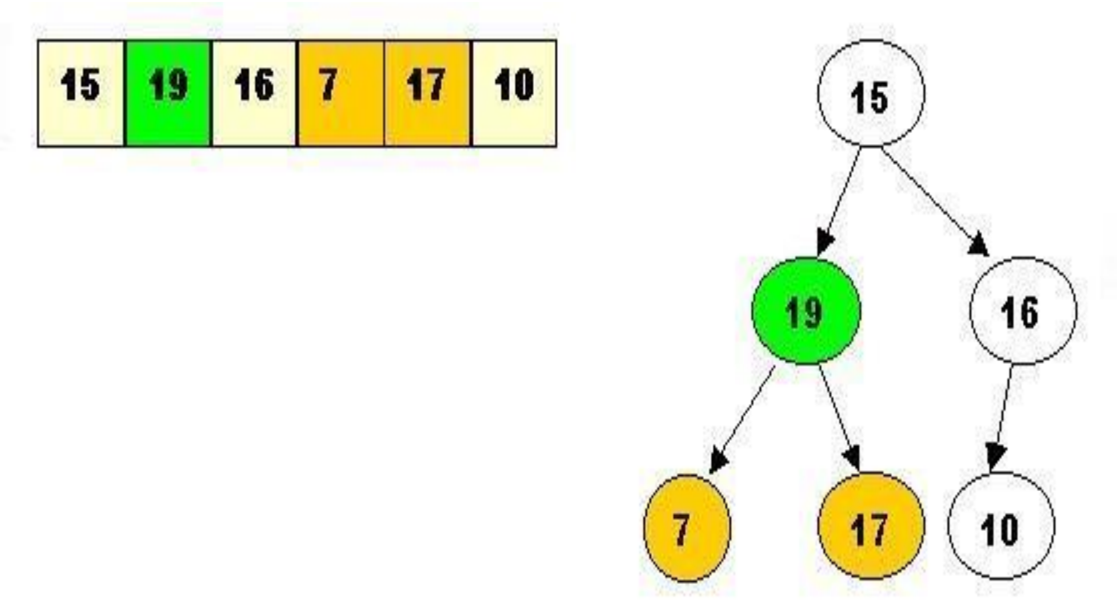
Start with the rightmost node at height 1-the node at position 3=size/2. It has one grater child and has to be percolated down:



After processing array [3] the situation is:

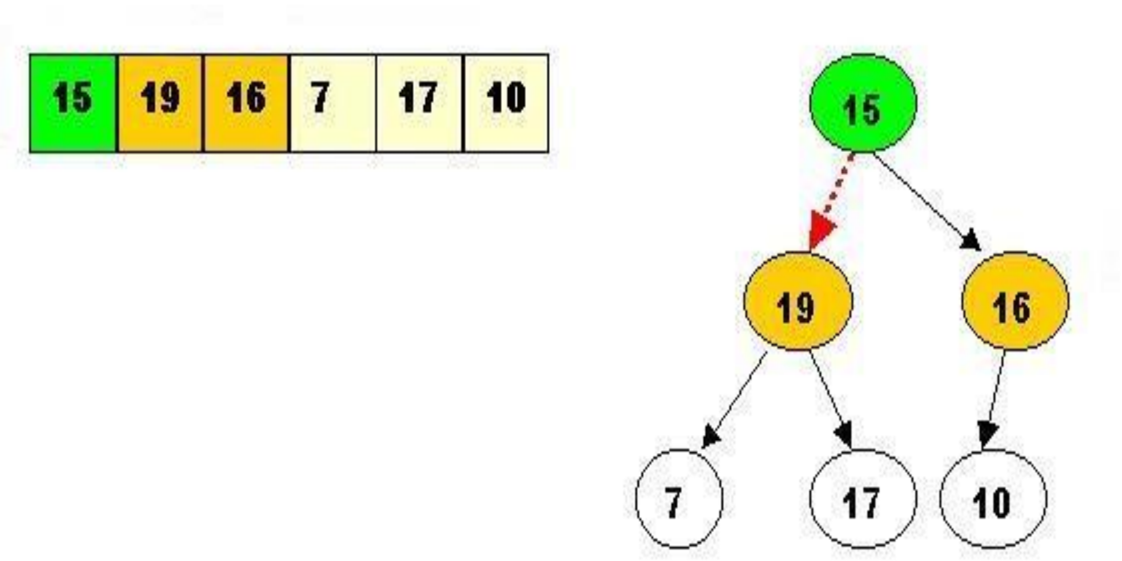


Next comes array [2]. Its children are smaller, so no percolation is needed.

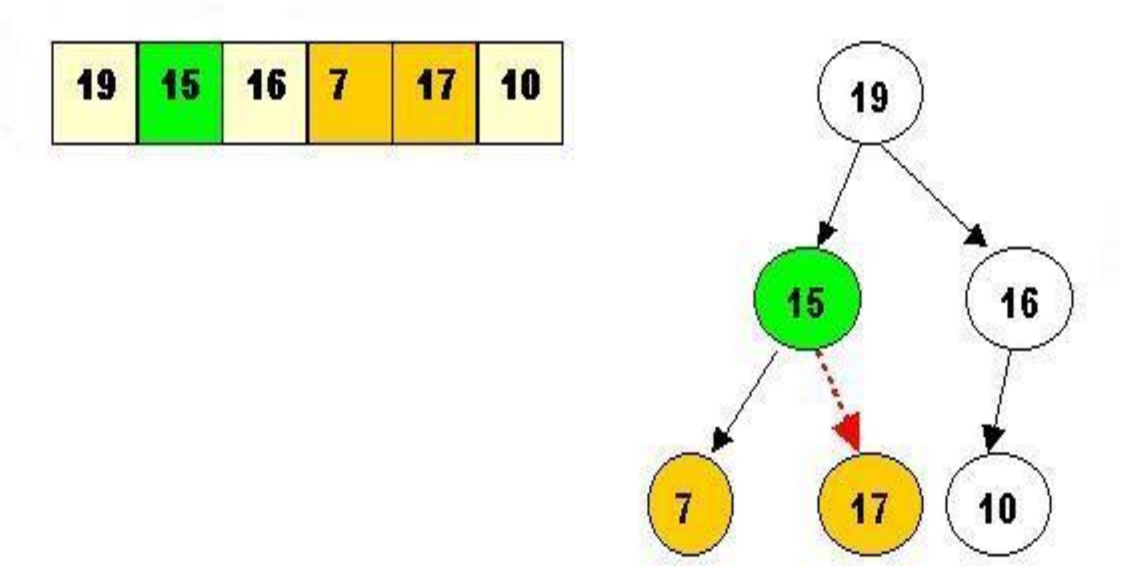


The last node to be processed is array [1]. Its left child is the grater of the children.

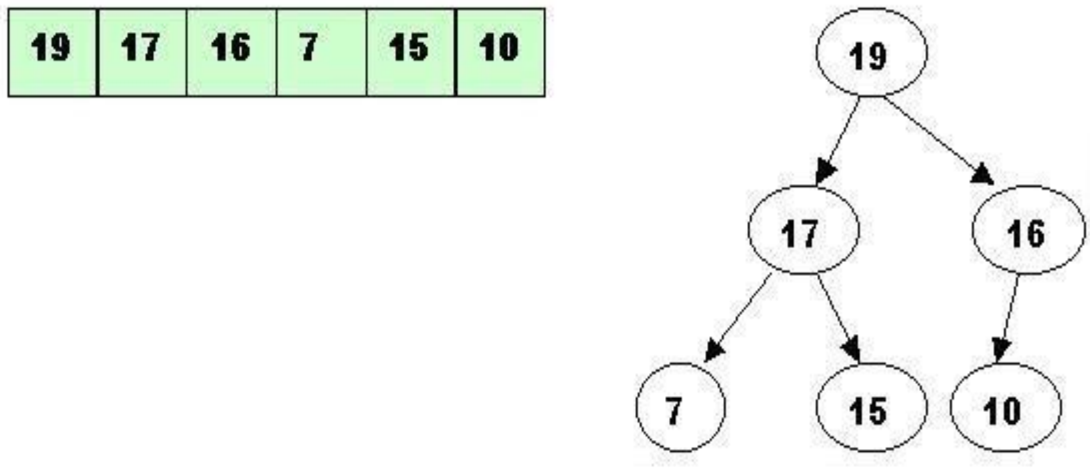
The item at array [1] has to be percolated down to the left, swapped array [2].



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].

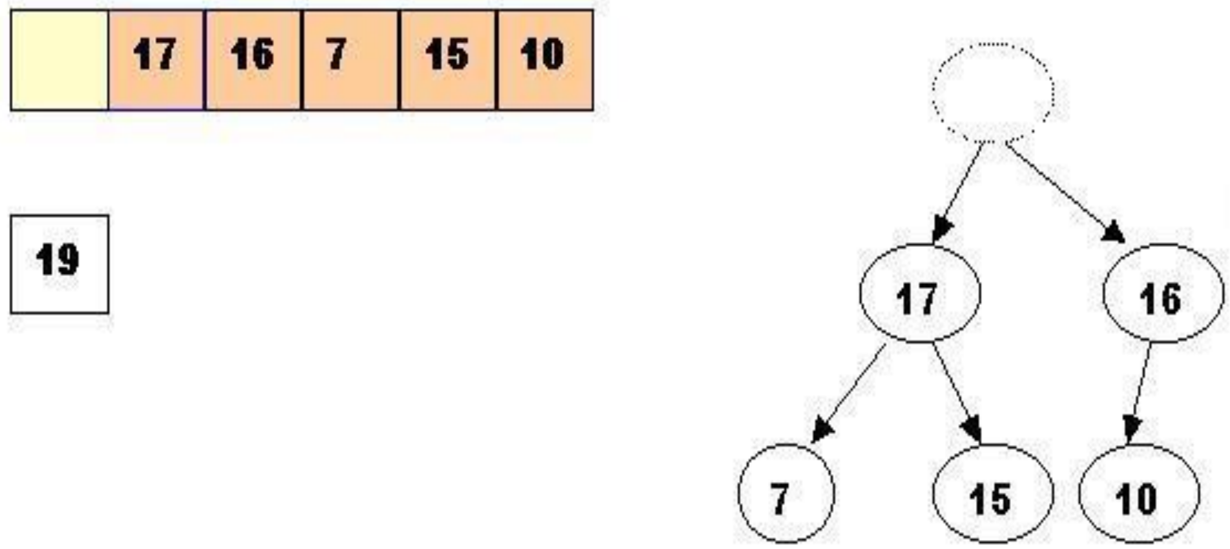


Now the tree is ordered, and the binary heap is built.

B. Sorting - performing deleteMax operations:

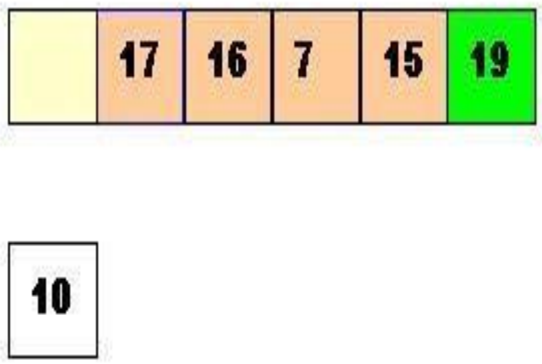
1. Delete the top element 19.

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

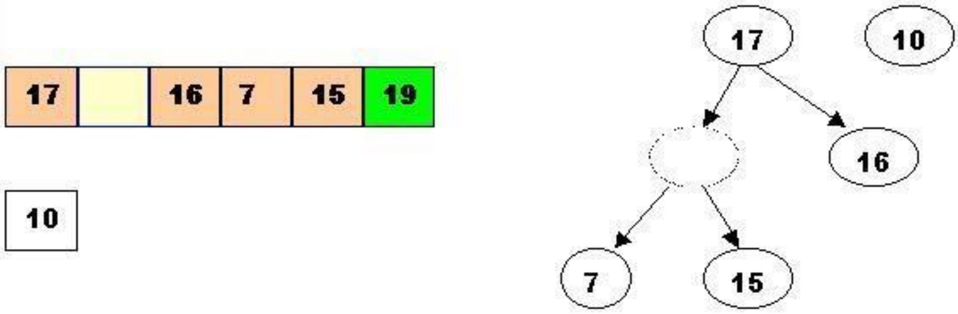


1.2. Swap 19 with the last element of the heap.

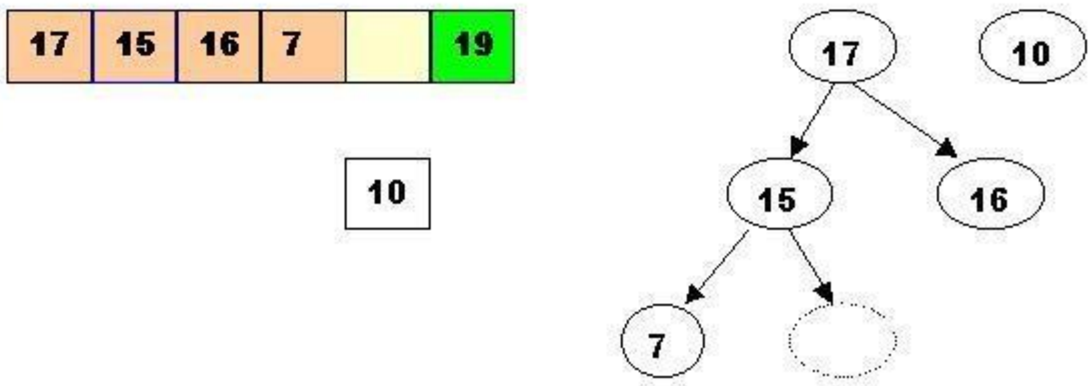
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a cell from the sorted array



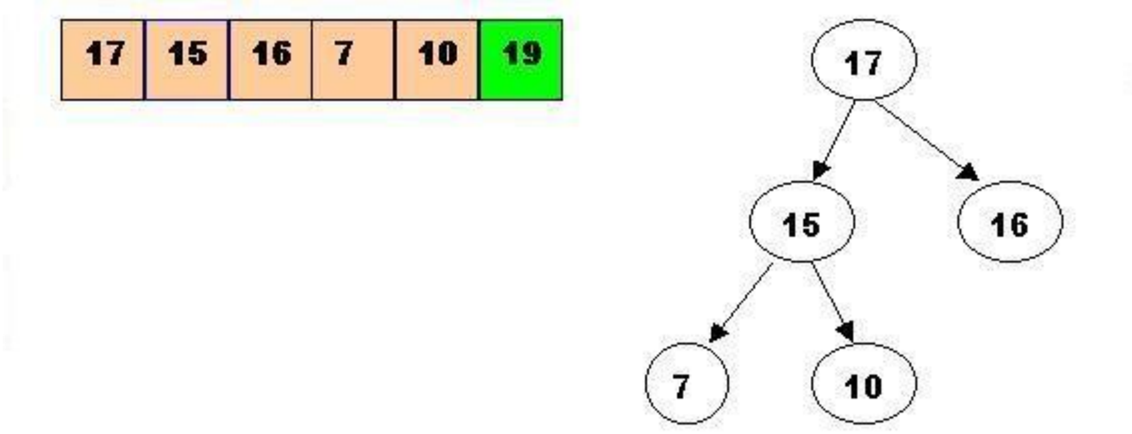
1.3 Percolate down the hole



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

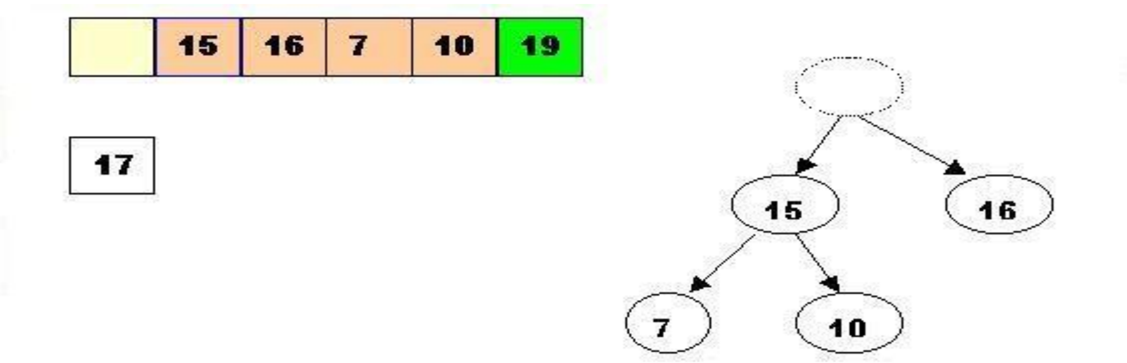


Now 10 can be inserted in the hole



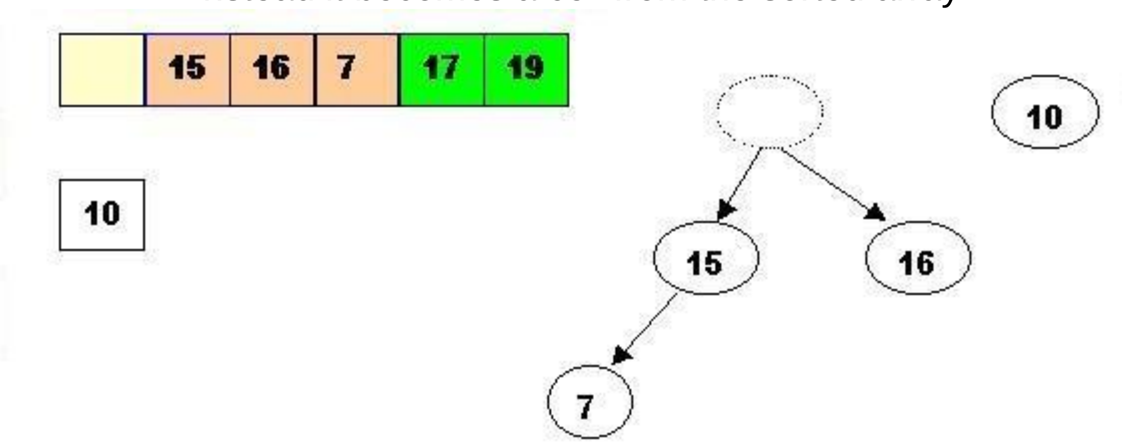
2. DeleteMax the top element 17

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

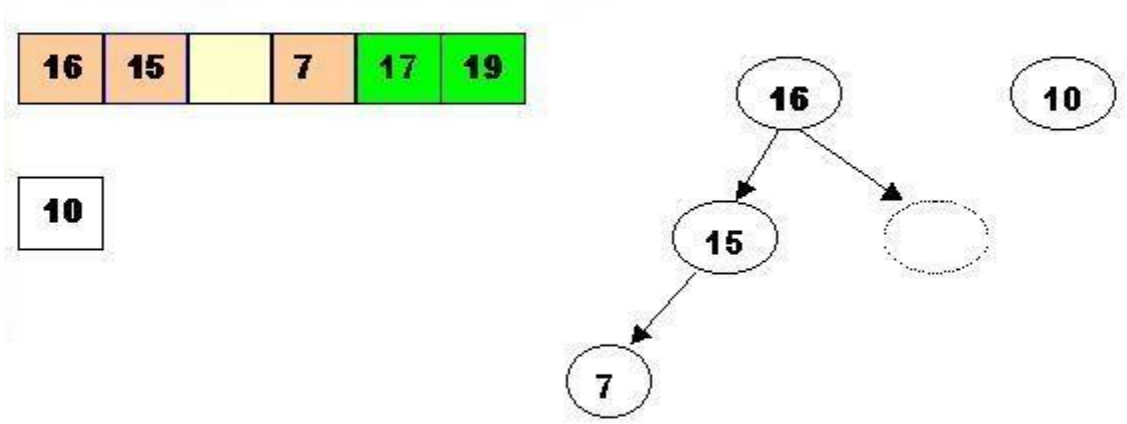


2.2. Swap 17 with the last element of the heap.

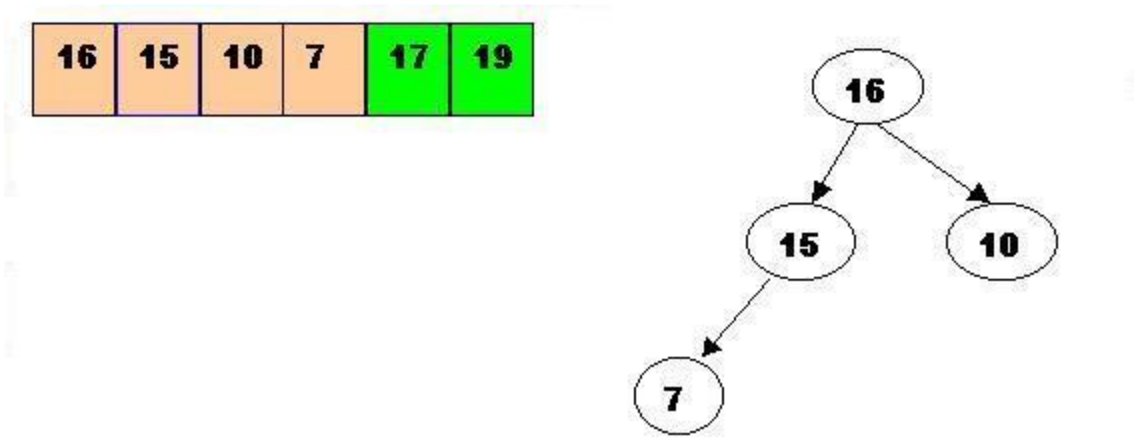
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a cell from the sorted array



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

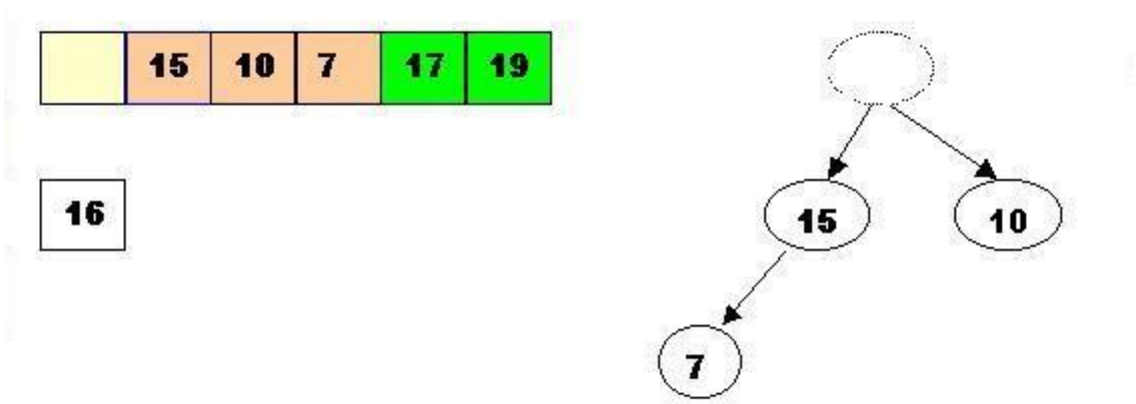


2.4 Insert 10 in the hole



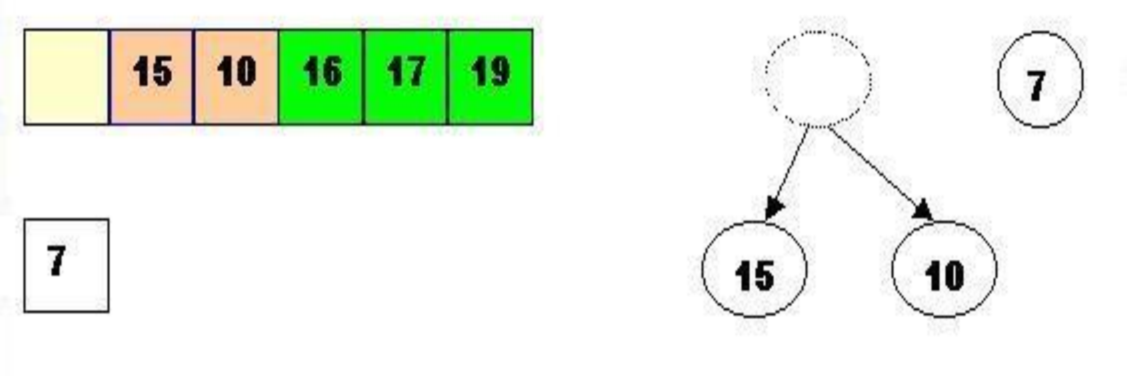
3. DeleteMax 16

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

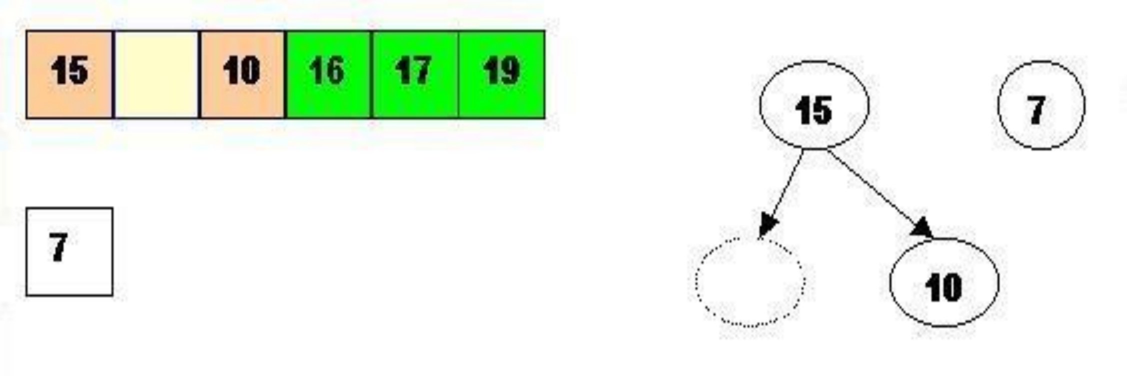


3.2. Swap 16 with the last element of the heap.

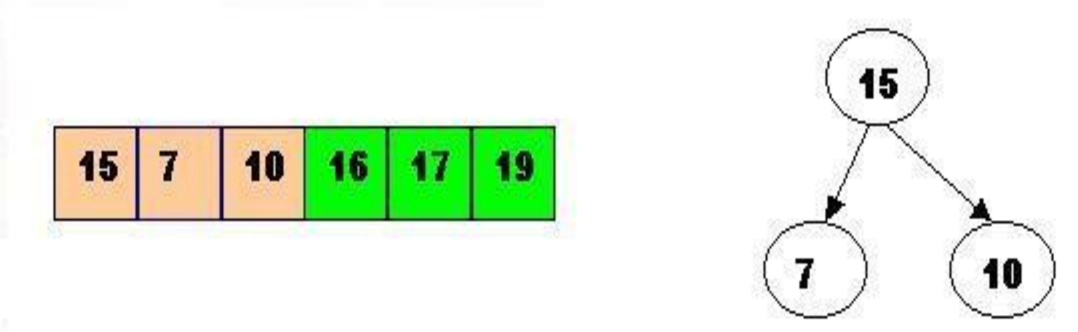
As 7 will be adjusted in the heap, its cell will no longer be a part of the heap.
Instead it becomes a cell from the sorted array



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



3.4. Insert 7 in the hole



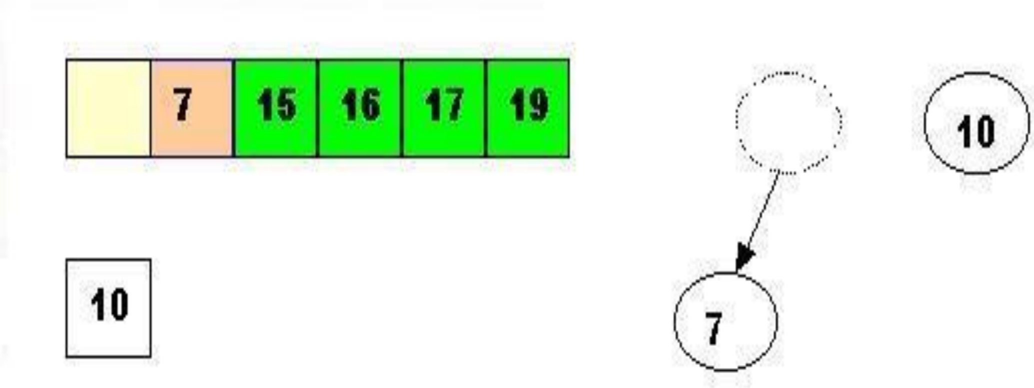
4. DeleteMax the top element 15

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

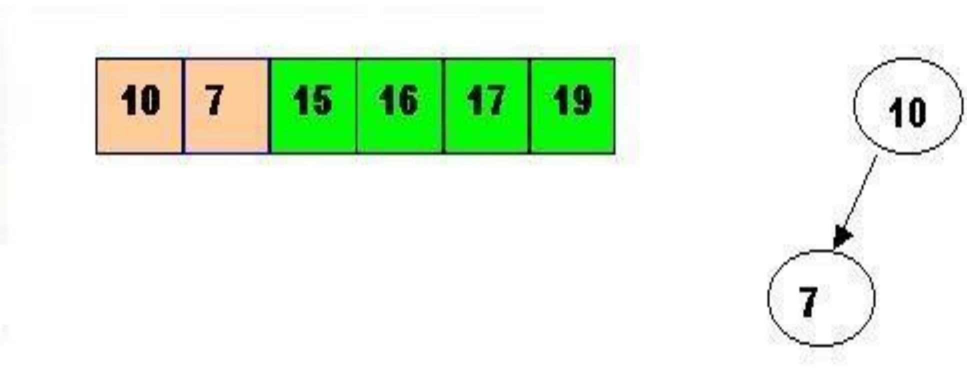
4.2. **Swap 15 with the last element of the heap.**

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.