Assignment No 12

Problem Statement:

Implement the Heap sort algorithm in Java demonstrating heap data structure with modularity of programming language

Theory:

What is a heap?

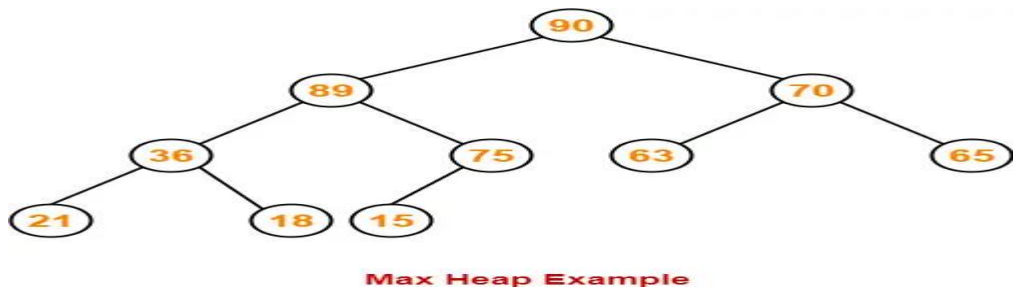
A heap is a complete binary tree, and the binary tree is a tree in which the node can have the utmost two children. A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node, should be completely filled, and all the nodes should be left-justified.

Heaps are mainly of two types — max heap and min heap:

- In a max heap, the value of a node is always >= the value of each of its children.
- In a min heap, the value of a parent is always <= the value of each of its children.

Example of max heap-

The following heap is an example of a max heap-



Max Heap Example

The Heap sort algorithm to arrange a list of elements in ascending order is performed using following steps...

- Step 1 - Construct a Binary Tree with given list of Elements.
- Step 2 - Transform the Binary Tree into Max Heap.
- Step 3 - Delete the root element from Max Heap using Heapify method.
- Step 4 - Put the deleted element into the Sorted list.
- Step 5 - Repeat the same until Max Heap becomes empty.
- Step 6 - Display the sorted list

Pseudo code for heap sort

```
Heapsort(A)
 {   BuildHeap(A)
      for i ← length(A) downto 2
     {      exchange A[1]  and A[i]
            heapsize ←heapsize -1
            Heapify(A, 1)
     }
     BuildHeap(A)
     {           heapsize ← length(A)
                 for i ←floor( length/2 ) downto 1
                        Heapify(A, i)
     }
     Heapify(A, i)
     {           le ← left(i)
                 ri ← right(i)
                 if (le<=heapsize) and (A[le]>A[i])
                              largest ← le
                 else
                              largest ← i
                 if (ri<=heapsize) and (A[ri]>A[largest])
                              largest ←ri
                 if (largest != i)
                 {      exchange A[i] and A[largest]
                           Heapify(A, largest)
                 }
}
```

Time Complexity:

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of heap sort is **O(n logn).**
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of heap sort is **O(n log n).**
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of heap sort is **O(n log n).**

Conclusion:

The time complexity of heap sort is **O(n logn)** in all three cases (best case, average case, and worst case). The height of a complete binary tree having n elements is **logn.**