Rajdhani
DATE / /

Roll No. 23355

Title : Assignment 9 : Heap Sort

Aim : To implement a heap sort

Problem statement : Implement heap sort to sort given set of values using max or min heap.

Theory :

- Heap data structure :
- Heap is a specialized tree based data structure which is an almost complete tree that satisfies heap property.
- The heap is one maximally efficient implementation of an abstract data type called a priority queue.
- In heap, the highest (or lowest) priority element is always stored at the root.
- However, a heap is not a sorted structure, it can be regarded as being partially ordered.
- A heap is a useful data structure when it is necessary to repeatedly remove the object with the highest (or lowest) priority.
- A common implementation of a heap is the binary heap, in which the tree is binary tree

Properties of binary heap data structure :

1) It's a complete tree i.e. All levels are completely filled except possibly the last level and the last level has all keys as left as possible).
This property of binary heap makes them suitable to be stored in an array

2) A binary heap is either min heap or max heap.

- Binary heap representation :
- A binary heap is a complete binary tree. A binary heap is typically represented as an array.

The root element will be at arr[0].
Let 'i' be the index of node

arr [(2*i)+1] => Gives left child of node
arr [(2*i)+2] => Gives right child of node.

- Types of heap

1) Max heap :
The key present at root node must be greater among the keys present at all of its children.
The same property must be recursively true for all subtrees in binary tree.

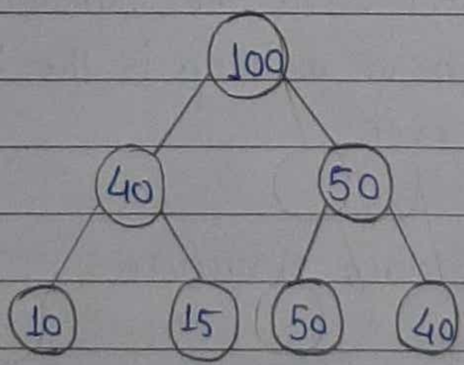


fig. Max heap

2) <u>Min heap</u> :
Key present at root node must be minimum among the keys present at all of its children.
The same property must be recursively true for all subtrees in that binary tree.

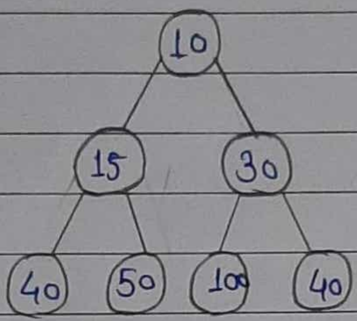
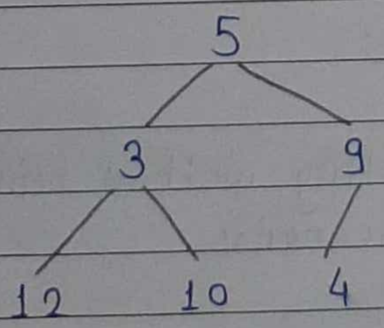


fig. ~~Hea~~ Min heap

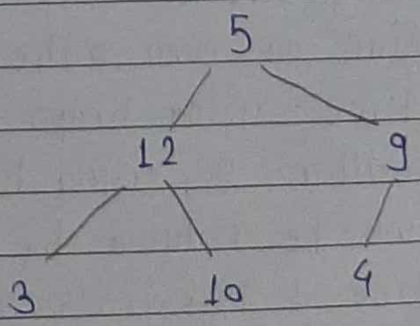
• How to construct a heap from scratch :

Array = {5, 3, 9, 12, 10, 4}

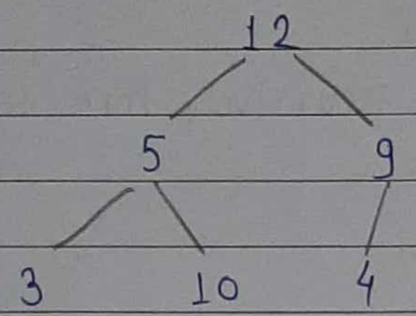
complete binary tree =



i) Heapify 3 : Swap 3 & 12



Heapify 12: swap 12 & 5



Heapify 5: swap 5 & 10

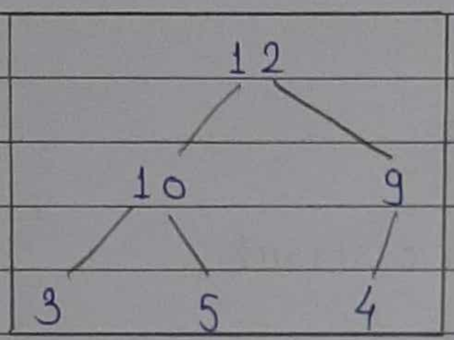


fig. This is final max heap for given array.

A						
	12	10	9	3	5	4

- Applications of heap data structure:

1) Heap sort: One of the best sorting method being inplace and with no quadratic worst case scenarios.

2) Selection algorithms: Finding the min, max both the min and max, median or even the k'th largest element can be done in linear time using heaps.

3) Graph algorithm: By using heaps as internal traversal data structure, run time will be reduced by polynomial order example: prim's minimum spanning tree & Dijkstra's shortest path problem.

Algorithm ~~for~~ heap Sort :

For sorting in increasing order, max heap is used.
For sorting in decreasing order, min heap is used.

Steps for sorting in increasing order:

1. Build a max heap from input data.
2. At this point, the largest item is stored at root of the heap. Replace it with the last item of heap followed by reducing the size of heap by 1. Finally heapify the root of tree.
3. Repeate ~~this~~ step 2 while size of heap is greater than 1.

- **Procedure Heapsort**

```
    // arr is the array of element with size n.
    ~~i = n/2~~ // build heap
    for i <- n/2 to i >= 0
        heapify (arr, n, i)
    // one by one extract an element from heap
    for i <- n-1 to i > 0
        swap arr[0] & arr[i]
        heapify (arr, i, 0)
End
```

- **Procedure Heapify**

```
    // to heapify a subtree rooted with node i.
    largest <- i
    left <- 2*i+1
    right <- 2*i+2
```

// if left child is larger than root
If left < n && arr[left] > arr[largest]
    largest ← left
// if right child is larger than root
If right < n && arr[right] > arr[largest]
    largest ← right

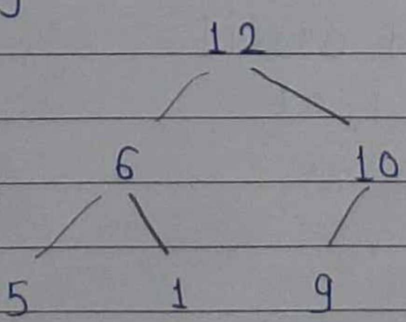
// if largest is not root
If largest ≠ i
    swap arr[i] & arr[largest]
    heapify (arr, n, largest)

END

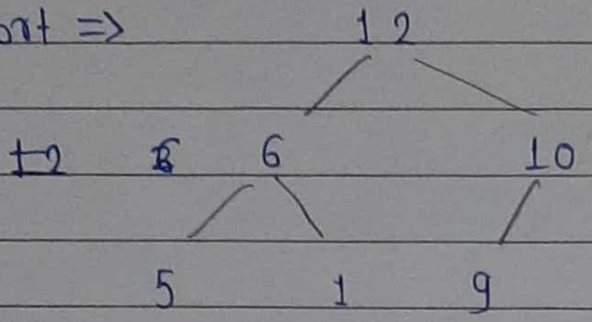
Example:

lets arr = {1, 12, 9, 5, 6, 10}
n = 6

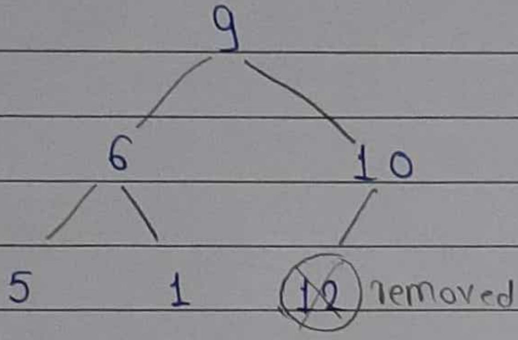
After heapify =>



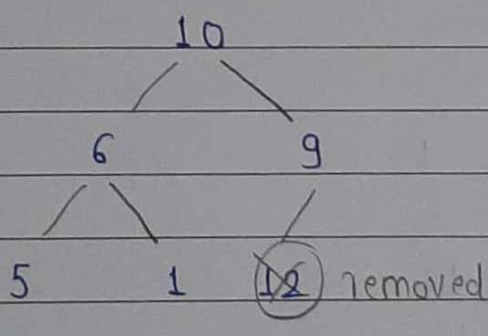
Heap Sort =>



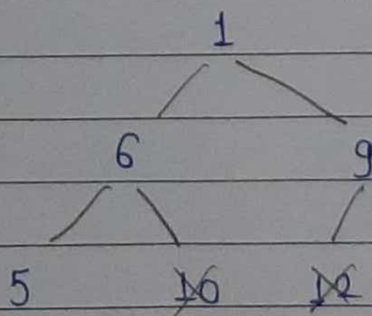
Swap (12, 9) & remove 12 from tree



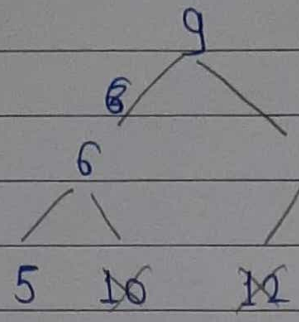
& heapify =>



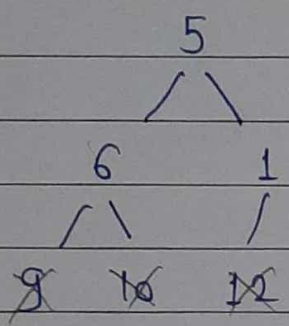
Swap (10, 1) & remove 10



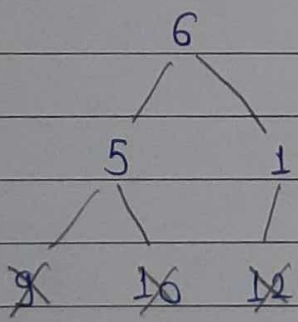
heapify =>



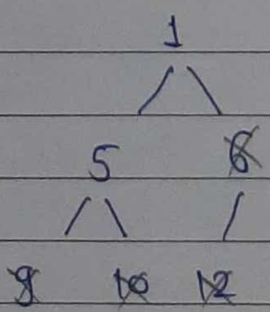
Swap (9,5) & remove 9.



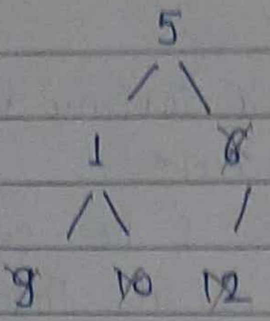
heapify =>



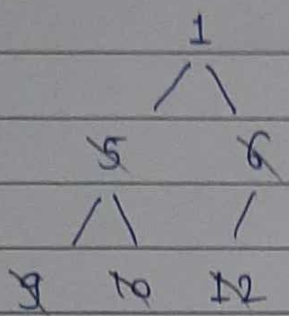
Swap (6, 1) & remove 6



heapify =>



Swap (5,1) & remove 5



∴ After sorting => Array={1, 5, 6, 9, 10, 12}

- validations:

1) limit validations
2) Data can be integer

Test cases:

1) Sorted input
2) completely unsorted input
3) Partially ~~un~~sorted input

All test cases are implemented & attached in output.

Rajdhani
DATE / /

Conclusion:
Heap sort is a comparison based sorting technique based on binary heap data structure.
space complexity of heap sort is 1 i.e. constant
Time complexity is $O\,n(\log n)$ is all three cases.
Heap sort is more efficient than quick sort & merge sort.
As for quick sort, worst case complexity is $O(n^2)$ & that for heap sort it is alway $O\,n(\log n)$.