PICT, PUNE

# DSAL Assignment 9

Name : Bhavana Bafna
Batch : E 09
Class : SE 09
Roll no. : 23107

**Title** : Heap Sort

**Aim** : Implement Heap Sort to given set of values using max or min heap.

**Theory** :

1) Heap data structure concept, definition:

- Heap is a specialised tree based data structure which is almost complete tree that satisfies heap property.
- The heap is one with 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, heap is not a sorted structure, it can be regarded as being partial ordered.
- A heap is a useful data structure when it is necessary to repeatedly remove the object with highest (or lowest) priority.
- A common implementation of a heap is the binary heap, in which tree is a binary tree.

2) Properties of binary heap data structure:

- It is a complete tree, i.e., all levels are completely filled except possibly the last level and the last level has all keys as left possible. This property of binary heap makes them suitable to be stored as an array.
- A binary heap is either minimum heap or Maximum 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 index of node.

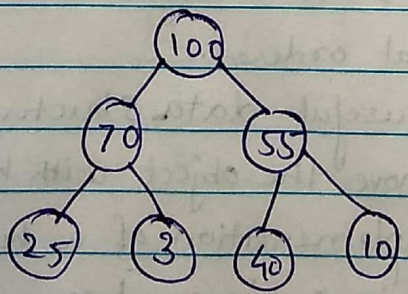
  arr[(2*i)+1] : Gives left child of node.
  arr[(2*i)+2] : Gives right child of node.

## 3) Types of heap:

### i) 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 a binary tree.

Eg:

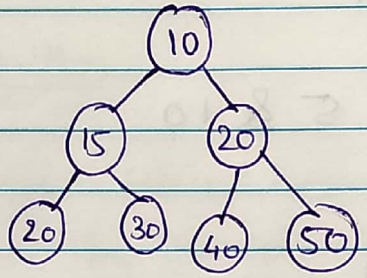



Max Heap

PICT, PUNE

## 2) Min Heap:

The key present at root node must be minimum among the keys present at all its children. The same property must be recursively true for all subtrees in the binary tree.

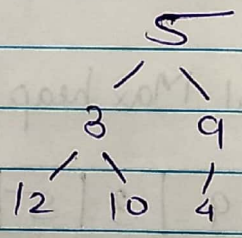



## 4) How to construct a heap from scratch explaination with diagram:

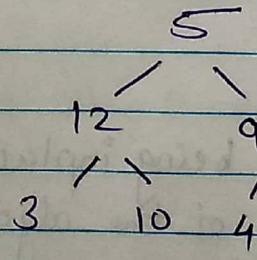
Eg: Consider an array

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

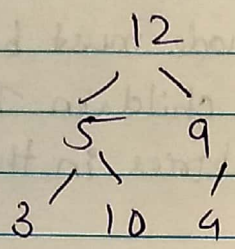
Complete binary tree:



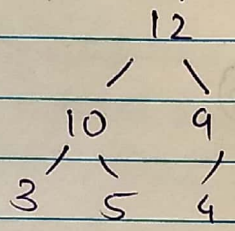

i) Heapify 3 : Swap 3 and 12



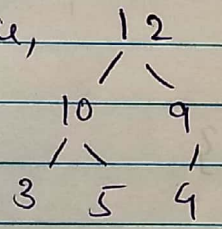
ii) Heapify 12 : Swap 12 & 5




iii) Heapify 5 : Swap 5 & 10




Hence,




This is the final Max heap for given array :

| 12 | 10 | 9 | 3 | 5 | 4 |
|---|---|---|---|---|---|

5) Application of heap data structure :

1) Heap Sort :

One of the best sorting method being inplace and with no quadratic worst case scenario of the algorithms.

2) Selection Algorithms:

Finding the minimum, maximum or both the minimum & maximum. Find 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, runtime will be reduced by the polynomial order.
Eg: Prim's MST and Dijkstra's shortest path algorithm.

7) Algorithms:

A) For Heap Sort:
For sorting in increasing order, max heap is used, for sorting in deacreasing 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 sorted at root of the heap. Replace it with last time of heap followed by reducing the size of the heap by 1. Finally heapify the root of tree.
3) Repeat step 2 while size of heap is greater than 1

PICT, PUNE

## Important algorithms:

1) Procedure Heapsort :

```
// arr is the array of element with size n
// 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] and arr[i]
      heapify (arr, i, 0)
End.
```

2) Procedure Heapify :

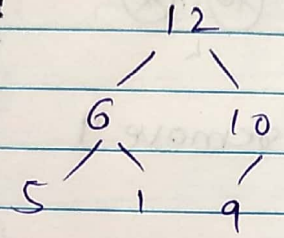
```
//to heapify a subtree rooted with node i
largest ← i
left ← 2*i + 1
right ← 2*i + 2
// if left child is larger than root
If left < n and arr[left] > arr[largest]
  largest ← left
// if right child is larger than root
If right < n and arr[right] > arr[largest]
  largest ← right.
// If largest is not root
If largest ≠ i
   swap arr [i] and arr [largest]
   heapify (arr, n, largest)
END
```

PICT, PUNE

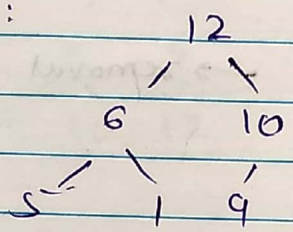
Example:

Let arr = {1, 12, 9, 5, 6, 10}

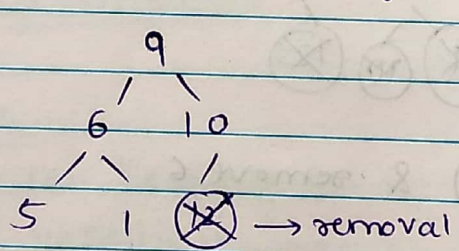
a) After heapify :



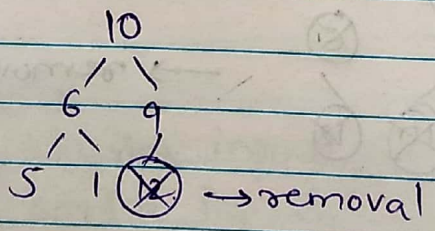

HeapSort :



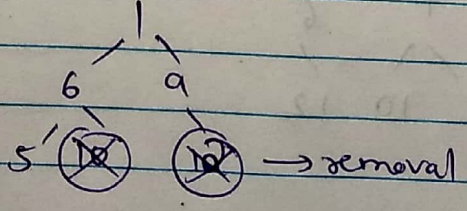

Swap (12, 9) & remove 12 from tree



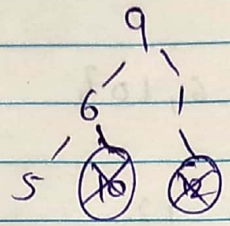

b) Heapify :



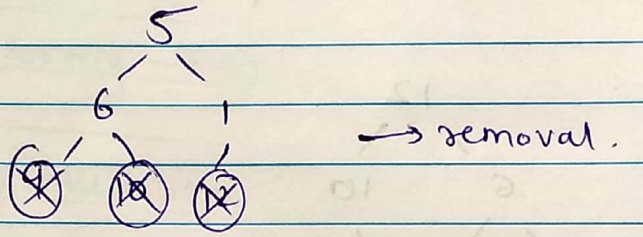

Swap (10, 1) and remove 10.



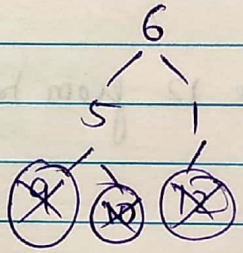
c) Heapify:



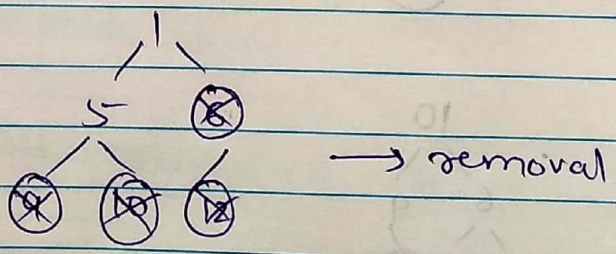

swap (9,5) & remove 9



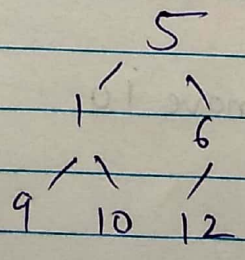

d) Heapify:




swap (6, 1) & remove 6

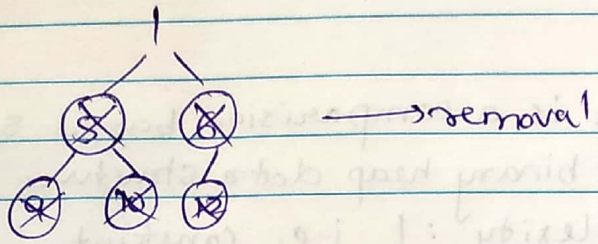



e) Heapify:

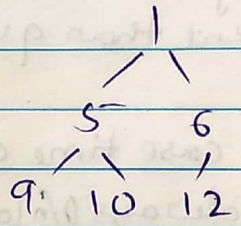


PICT, PUNE

Swap (5, 1) and remove 5




∴ Final tree :




∴ After sorting → array = {1, 5, 6, 9, 10, 12}

8) Test cases / Validations:

a) Validations:
1) Data can be integers
2) Limit Validations

b) Test cases:
1) Sorted input
2) Completely unsorted input
3) Partially sorted input.

All test cases are implemented and attributed in output.

11) Conclusion:

- Heap sort is a comparision based sorting technique based on binary heap data structure.
- Space complexity : 1 i.e., constant
  Time complexity : O (nlogn) in all the three cases.
- Heap sort is more efficient than quick sort and merge sort.
- For quick sort, worst case time complexity is $O(n^2)$ and that for heap sort is always O(nlogn).