What is application of heap sort?

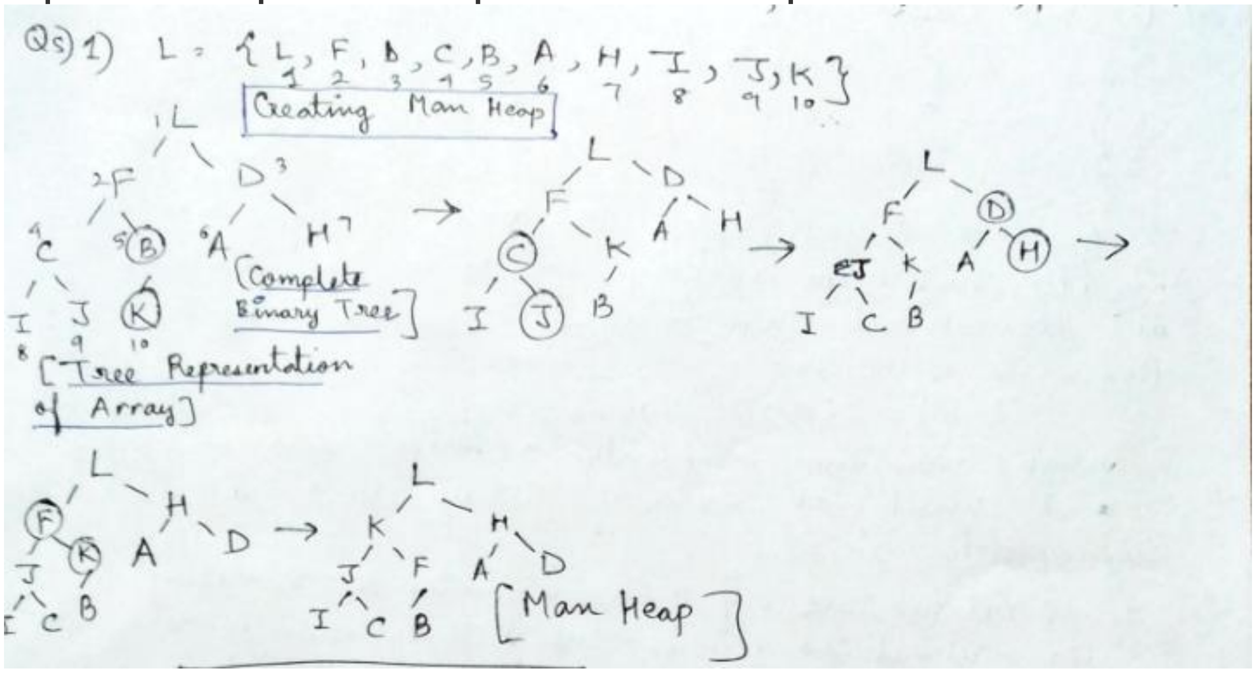
Advantages: Optimized performance, efficiency, and accuracy are a few of the best qualities of this algorithm. The algorithm is also highly consistent with very low memory usage. No extra memory space is required to work, unlike the Merge Sort or recursive Quick Sort.

Applications of Heap Sort

- Implementation of priority queues
- Security systems
- Embedded systems (for example, Linux Kernel)
- Graph Algorithms: The heaps are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.

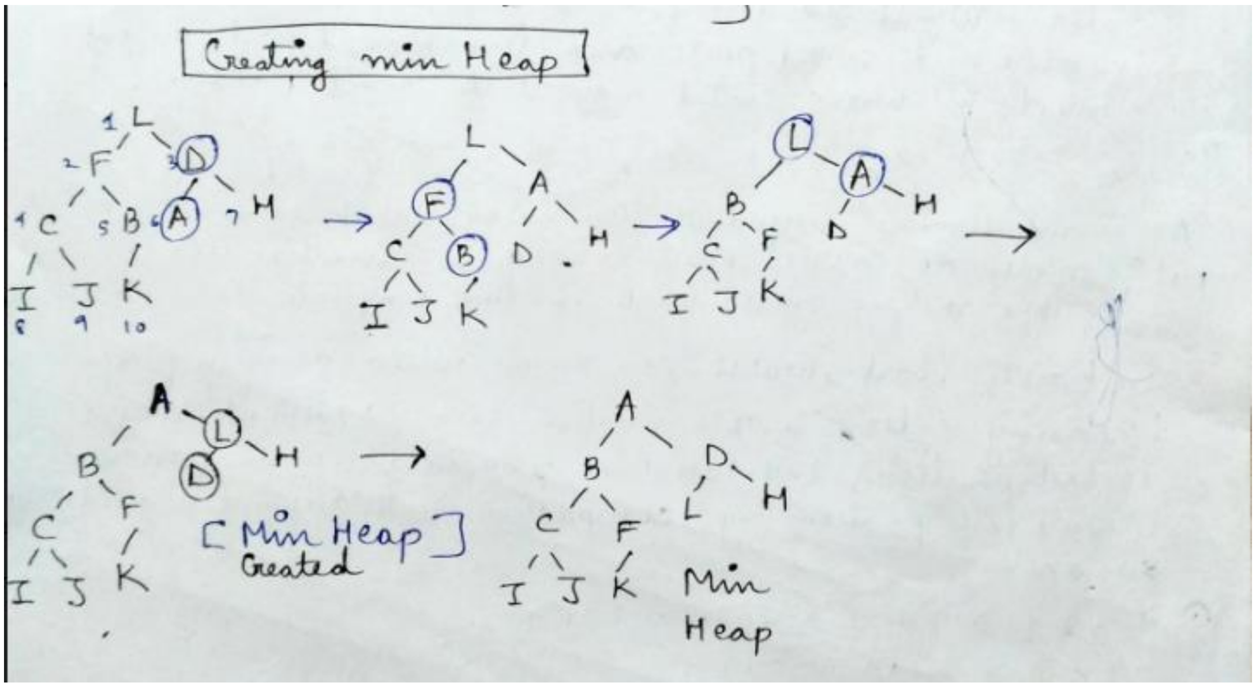
Because algorithms like **merge sort** and **quicksort** are better in practice, heap sort has limited usage. Heaps are extensively used for problems like getting the largest or smallest elements in an array, sorting an almost sorted array, etc.

Explain Max Heap and Min Heap with suitable examples.



Qs) 1) L = {L, F, D, C, B, A, H, I, J, K}
Creating Max Heap
Complete Binary Tree
Tree Representation of Array
Max Heap

This is Max heap creation.



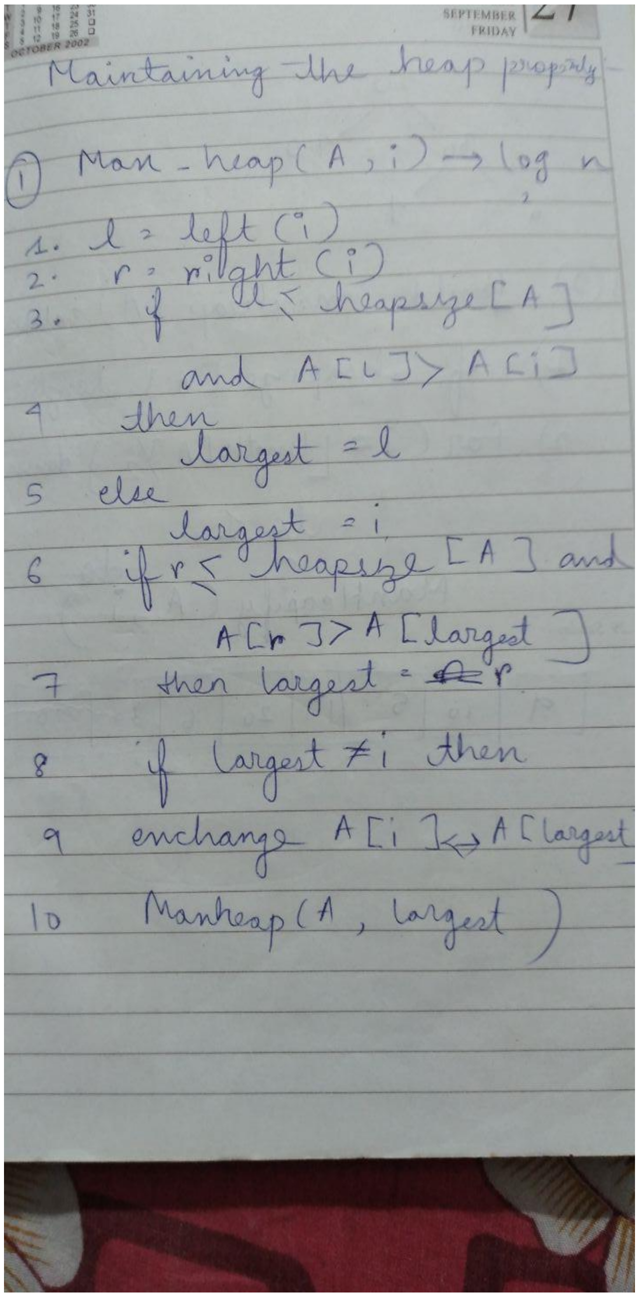
This is Min Heap Creation.

Difference between Min Heap and Max Heap

	Min Heap	Max Heap
1.	In a Min-Heap the key present at the root node must be less than or equal to among the keys present at all of its children.	In a Max-Heap the key present at the root node must be greater than or equal to among the keys present at all of its children.
2.	In a Min-Heap the minimum key element present at the root.	In a Max-Heap the maximum key element present at the root.
3.	A Min-Heap uses the ascending priority.	A Max-Heap uses the descending priority.
4.	In the construction of a Min-Heap, the smallest element has priority.	In the construction of a Max-Heap, the largest element has priority.
5.	In a Min-Heap, the smallest element is the first to be popped from the heap.	In a Max-Heap, the largest element is the first to be popped from the heap.

How is Max Heap represented?
A-Max Heap is a Complete Binary Tree. A-Max heap is typically represented as an array. The root element will be at Arr[0]. Below table shows indexes of other nodes for the **ith** node, i.e., Arr[i]:
Arr[(i-1)/2] Returns the parent node.
*Arr[(2*i)+1] Returns the left child node.*
*Arr[(2*i)+2] Returns the right child node.*

Heap sort Algo:-



OCTOBER 2002

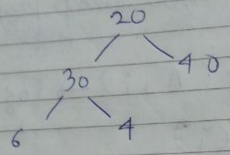
SEPTEMBER
FRIDAY

Maintaining the heap property

```
(1) Max-heap(A, i) → log₂ n

1.  l = left(i)
2.      r = right(i)
3.        if   l ≤ heapsize[A]
               and A[l] > A[i]
4       then
               largest = l
5     else
               largest = i
6     if r ≤ heapsize[A] and
                A[r] > A[largest]
7          then largest = ~~A~~ r
8       if largest ≠ i then
9     enchange A[i] ↔ A[largest]
10       Maxheap(A, largest)
```

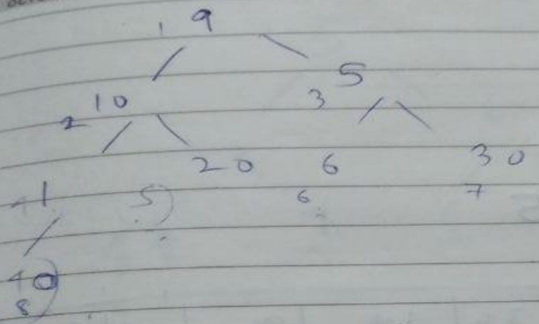
1	8	15	22	29	S
2	9	16	23	30	M
3	10	17	24		T
4	11	18	25		W
5	12	19	26		T
6	13	20	27		F
7	14	21	28		S



② Build maxheap (A) → nlogn

```
1) ~~if~~ heapsize [A] = length[A]
2) for ( i = ⌊length[A]/2⌋ down
        to 1
            do
     MaxHeapify ( A , i )
   )
```

9	10	5	1	20	6	30	40



```
Heap Sort (A) →
i)   Build MaxHeap (A)
ii)  for i = length(A) down to
                2
            do
iii) exchange A[1] ↔ A[i]
iv)  heapsize[A] = heapsize[A]
                     - 1
v)   Maxheapify (A, 1)
```