4.

* Merge Sort *

Trace for: 8 3 2 9 7 1 5 4

8 3 2 9 7 1 5 4

8 3 2 9   7 1 5 4

8 3   2 9   7 1   5 4

8   3   2   9   7   1   5   4

3 8   2 9   1 7   4, 5

2 3 8 9   1 4 5 7

1 2 3 4 5 7 8 9

DATE : / /

Refer handout for algorithm.

Efficiency Analysis:

$$T(n) = \begin{cases} 0 & n=1 \\ T(n/2) + T(n/2) + c(n) & \text{otherwise} \end{cases}$$

↳ divide & merge one instance of solution.

$T(n) = 2T(n/2) + c(n)$

$n = 2^k$

$T(2^k) = 2T(2^{k-1}) + c.2^k \rightarrow (1)$

put $k = k-1$ in (1)

$T(2^{k-1}) = 2T(2^{k-2}) + c.2^{k-1} \rightarrow (2)$

substitute (2) in (1)

$T(2^k) = 2[T(2^{k-2}) + c.2^{k-1}] + 2^k c$

$= 2^2T(2^{k-2}) + c.2^k + c.2^k$

$= 2^2T(2^{k-2}) + 2c.2^k$

$= 2^3T(2^{k-3}) + 3.c2^k$

$= \dots$

$= 2^kT(2^{k-k}) + k.c.2^k$

$= 2^k.0 + k.c.2^k$

$= c.k.2^k$

$= n.\log_2 n$

$\in \Theta(n\log_2 n)$

$2^k = n$

$k\log_2 2 = \log_2 n$

$k = \log_2 n$

The shortcoming of merge sort is the linear amount of extra storage the algorithm requires.

* Quick Sort *

Trace for:

0	1	2	3	4	5	6	7
$\boxed{5}$	3	1	9	8	2	4	7
			i			j	
$\boxed{5}$	3	1	4	8	2	9	7
				i	j		
$\boxed{5}$	3	1	4	2	8	9	7
				j	i		
2	3	1	4	$\boxed{5}$	8	9	7

$\boxed{2}$ 3 1 4

i j

~~r /3/ $\boxed{2}$ / 4~~

$\boxed{2}$ 1 3 4

j i

1 $\boxed{2}$ 3 4

1

$\boxed{3}$ 4

j i

$\boxed{3}$

4

PAGE NO. :
DATE : / /

0	1	2	3	4	5	6	7
					$\boxed{8}$	9	7
						i	j
					$\boxed{8}$	7	9
						j	i
					7	$\boxed{8}$	9
					7		
							9

Trace for:

0	1	2	3	4	5	6	7
$\boxed{42}$	37	11	98	36	72	65	10
			i				j
$\boxed{42}$	37	11	10	36	72	65	98
				j	i		
36	37	11	10	$\boxed{42}$	72	65	98
$\boxed{36}$	37	11	10				
	i		j				
$\boxed{36}$	10	11	37				
		j	i				
11	10	$\boxed{36}$	37				
$\boxed{11}$	10 i						
	j						
10	$\boxed{11}$						
10							

DATE : / /

0 1 2 3 4 5 6 7

37

$\boxed{72}$ 65 98

j i

65 $\boxed{72}$ 98

65

98

Efficiency Analysis:

Best Case: When input is divided into two equal halves in each iteration.

$$T(n) = \begin{cases} 0 & n = 1 \\ T(n/2) + T(n/2) + C(n) & \text{otherwise} \end{cases}$$

Time for partition.

$T(n) = 2T(n/2) + c.n$ ~~→ (1)~~

$n = 2^k$

$T(2^k) = 2T(2^{k-1}) + c.2^k \rightarrow (1)$

put $k = k-1$ in (1)

$T(2^{k-1}) = 2T(2^{k-2}) + c.2^{k-1} \rightarrow (2)$

put (2) in (1)

$T(2^k) = 2[2T(2^{k-2}) + c.2^{k-1}] + c.2^{k-1}$

PAGE NO. :
DATE : / /

$= 2^2 T(2^{k-2}) + c.2^k + c.2^k$

$= 2^2 T(2^{k-2}) + 2.c.2^k$

$= 2^3 T(2^{k-3}) + 3.c.2^k$

$= \dots$

$= 0 + k.c.2^k$

$= n \log_2 n$

$\in \Omega(n \log_2 n)$

Worst Case: One partition is empty & other with remaining elements. Happens when input is in ascending/descending order.

Eg:-

$\boxed{5}$ 10 15 10
j i

$\boxed{5}$ 10 15 10

← 0 items n-1 items →

$$T(n) = \begin{cases} 0 & n=1 \\ T(0) + T(n-1) + cn & \text{otherwise.} \end{cases}$$

$T(n) = T(n-1) + cn \rightarrow (1)$

put $n = n-1$

$T(n-1) = T(n-2) + c(n-1) \rightarrow (2)$

put (2) in (1)

$T(n) = T(n-2) + c(n-1) + c(n)$

PAGE NO. :
DATE : / /

$$= T(n-3) + c(n-2) + c(n-1) + c(n)$$

$$= \dots$$

$$= T(n-(n-1)) + c(n-(n-1)) + \dots \; c(n-1) + c(n)$$

$$= c(1) + c(2) + \dots \; c(n-1) + c(n)$$

$$= c\left(\frac{n(n+1)}{2}\right)$$

$$= c\left(\frac{n^2}{2} + \frac{n}{2}\right)$$

$$\in O(n^2)$$

Average case: Partition split can happen in each position s $(0 \leq s \leq n-1)$ with same probability $1/n$

$$T(n) = \begin{cases} 0 & 0 \\ 0 & 1 \\ \frac{1}{n}\sum_{s=0}^{n-1}\left[T(s) + T(n-1-s) + (n+1)\right] & \text{otherwise} \end{cases}$$

↳ no of comparisons

On solving, we obtain

$$T(n) \approx 1.38\, n \log_2 n$$

$$\in \Theta(n \log_2 n)$$

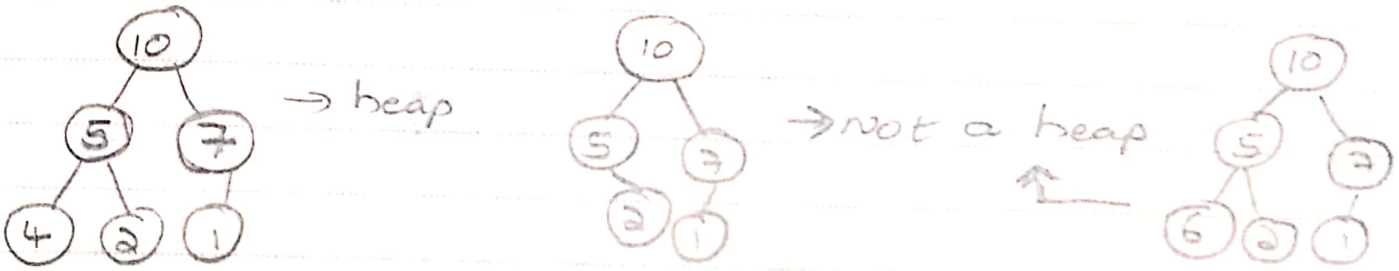
PAGE NO. :
DATE : / /

* Heap Sort *

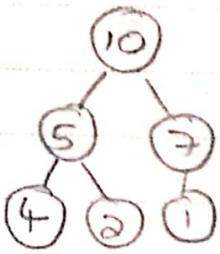
Heap is a binary tree with following conditions:

i) Tree's shape requirement - essentially complete or almost complete

ii) Parental dominance requirement - Parent is greater than children for max heap.



Array Representation:



0	1	2	3	4	5	6
	10	5	7	4	2	1

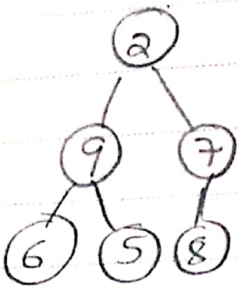
- children of node i will be in
  - $2i$
  - $2i+1$

- parent of a key in position i will be in $\lfloor i/2 \rfloor$

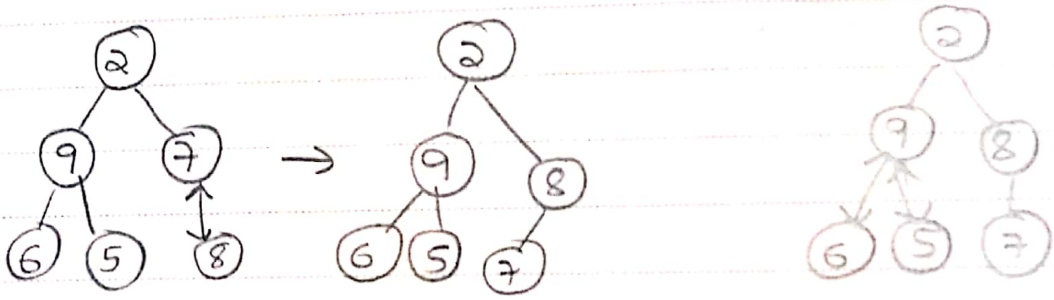
Scanned by CamScanner

Bottom Up Heap Construction:

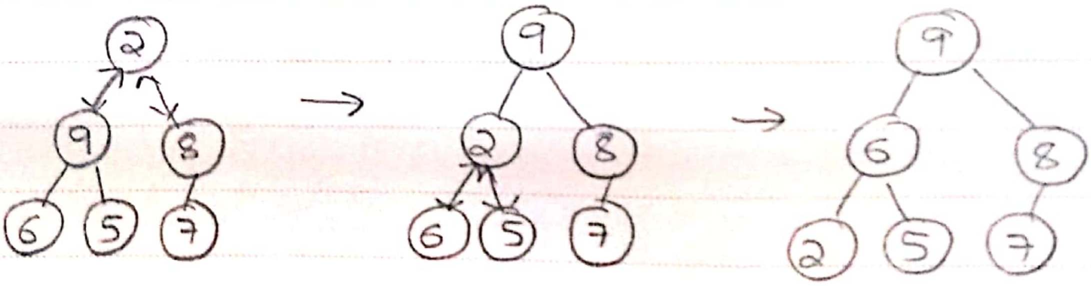
2 9 7 6 5 8



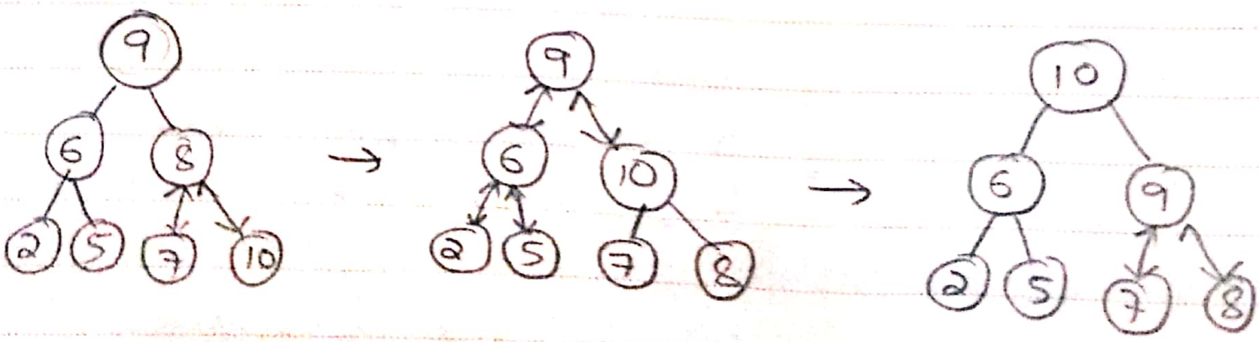
- Level order Tree
- Then heapify.



→ No swap.

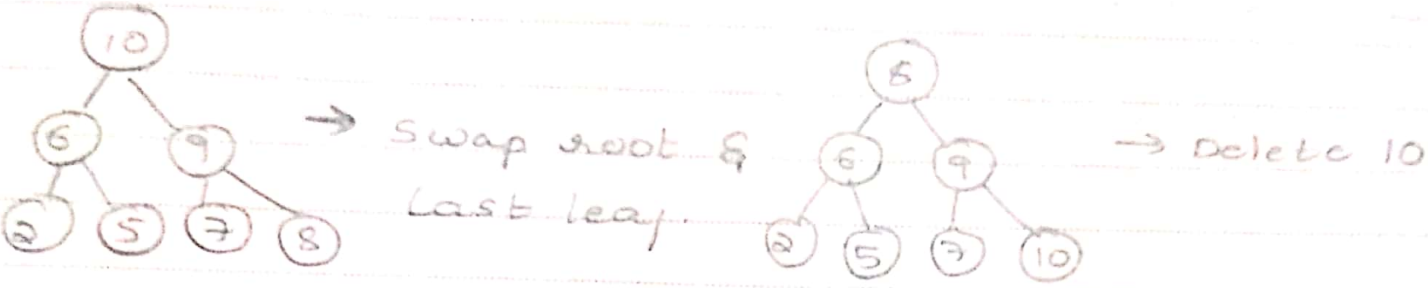


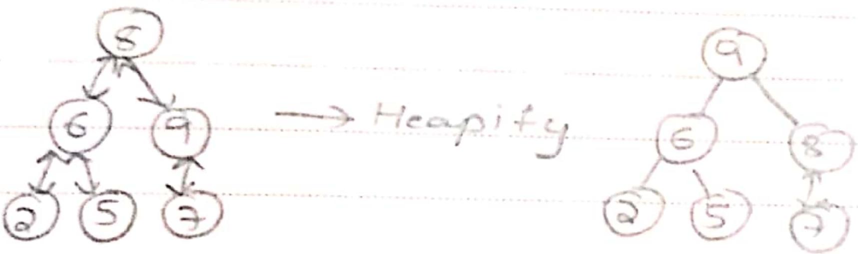
Insert 10 into the heap:



No change

Delete from a heap:-

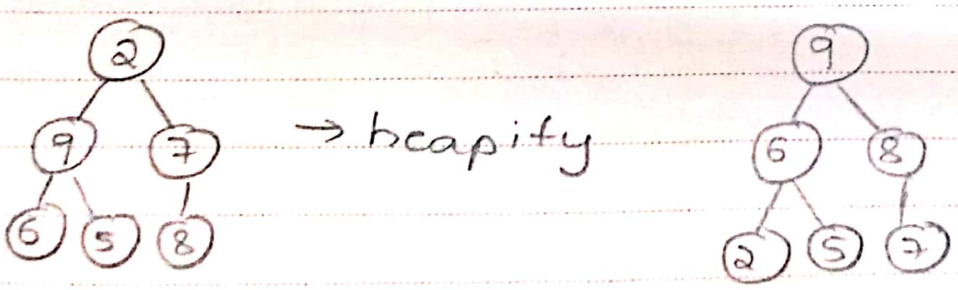




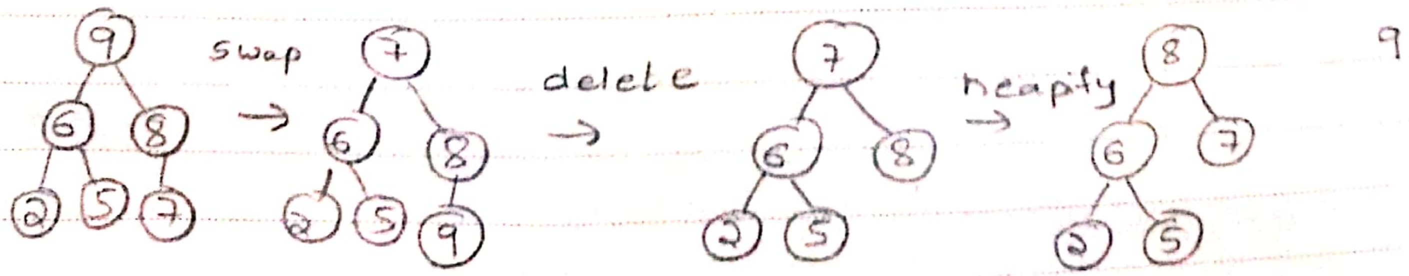
Heap Sort:

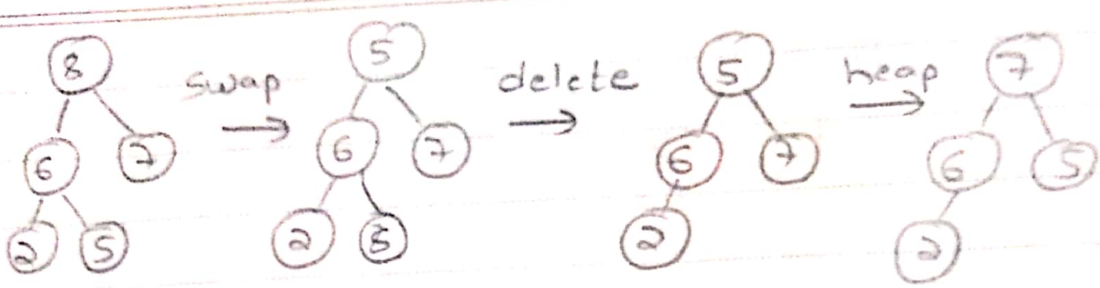
- Construct heap for the given array
- Apply root deletion operation n-1 times

Example:

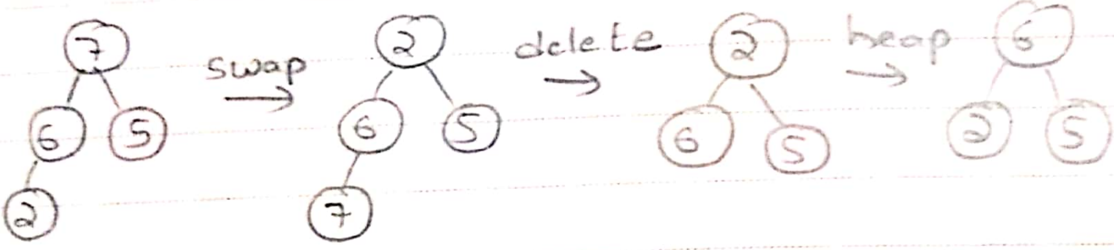


Now apply root deletions on heapified tree.

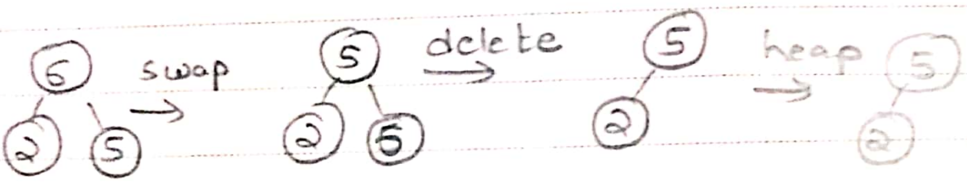




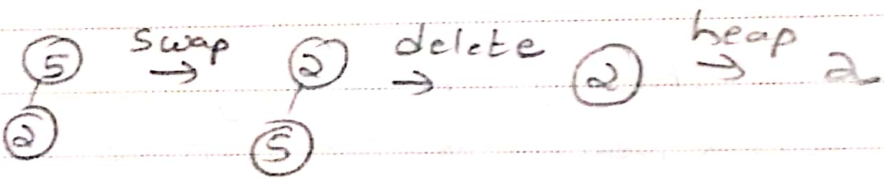
8 9



7 8 9



6 7 8 9



5 6 7 8 9

2 5 6 7 8 9