Numbers in RED are index positions in array

* The corresponding array is shown below each max-heap visualized as a nearly complete binary tree.

Q. Illustrate stepwise execution of the Heapsort algorithm on an array A = [44, 30, 50, 22, 60, 56, 77, 55] to arrange its elements in ascending order.

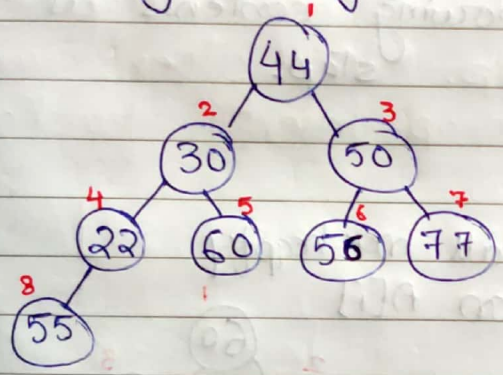
STEP I: Create max-heap.

It can be done in two ways

(1) Taking all the elements at once

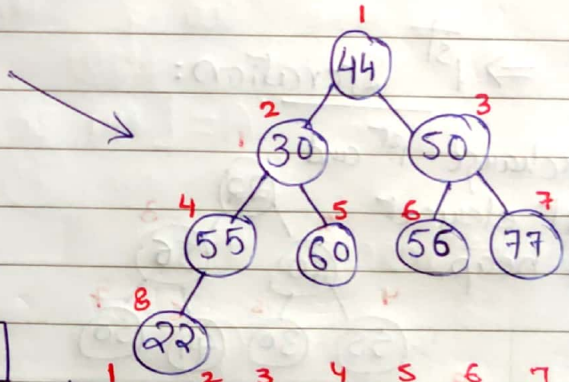
(2) Taking one element, insert in the existing max-heap.

Using way (1)



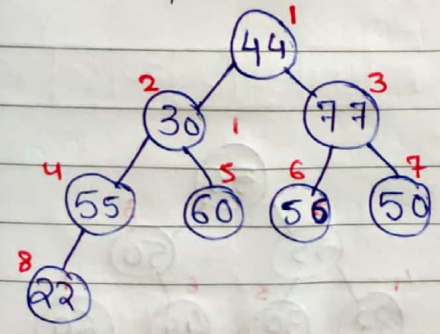
A: | 1:44 | 2:30 | 3:50 | 4:22 | 5:60 | 6:56 | 7:77 | 8:55 |

⇒ Execute max-heapify at index 4, i.e. A[4] = 22.



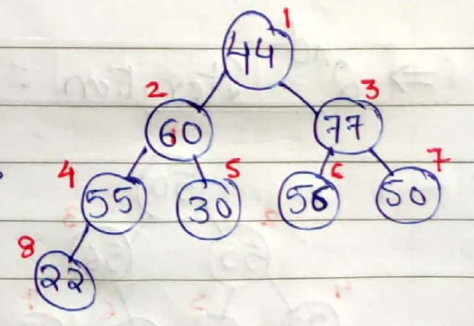
A: | 1:44 | 2:30 | 3:50 | 4:55 | 5:60 | 6:56 | 7:77 | 8:22 |

⇒ Execute max-heapify at index 3, i.e. A[3] = 50.



A: | 1:44 | 2:30 | 3:77 | 4:55 | 5:60 | 6:56 | 7:50 | 8:22 |

⇒ Execute max-heapify at index 2, i.e. A[2] = 30

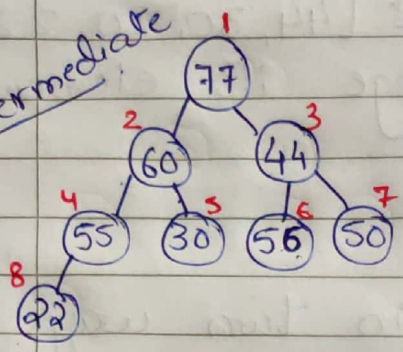


A: | 1:44 | 2:60 | 3:77 | 4:55 | 5:30 | 6:56 | 7:50 | 8:22 |

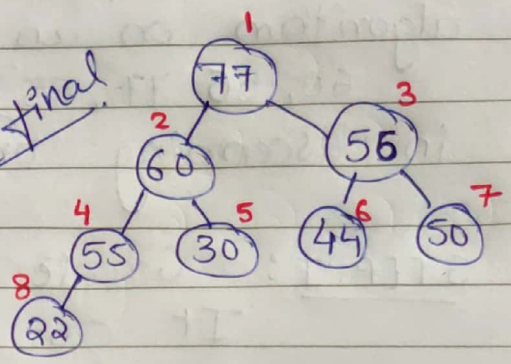
Scanned with CamScanner

⇒ Execute max-heapify at index 1, i.e. A[1] = 44.

intermediate



→ final



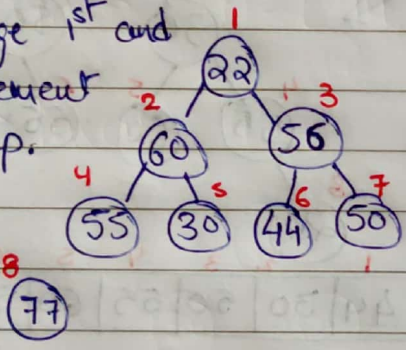
	1	2	3	4	5	6	7	8
A:	77	60	44	55	30	56	50	22

	1	2	3	4	5	6	7	8
A:	77	60	56	55	30	44	50	22

STEP II : Execute rest of the steps of heapsort algorithm showing contents at each intermediate step.

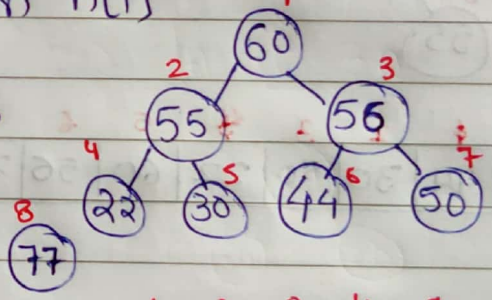
⇒ 1st Iteration:

Exchange 1st and last element in heap.



Call max-heapify on A[1]

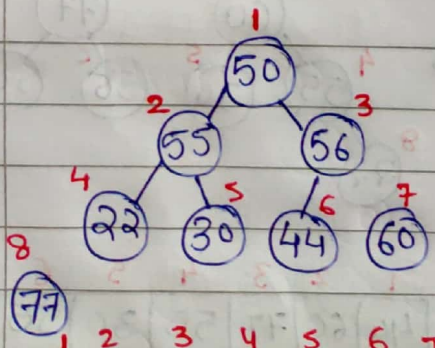
→



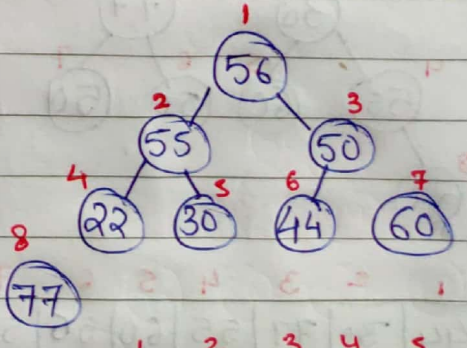
	1	2	3	4	5	6	7	8
A:	22	60	56	55	30	44	50	77

	1	2	3	4	5	6	7	8
A:	60	55	56	22	30	44	50	77

⇒ 2nd Iteration :



→

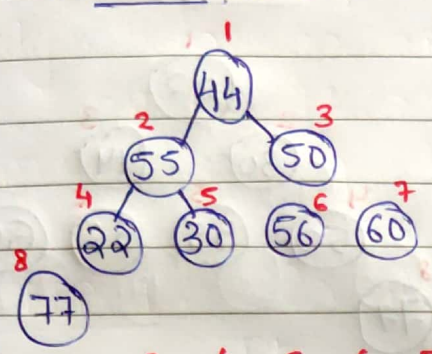


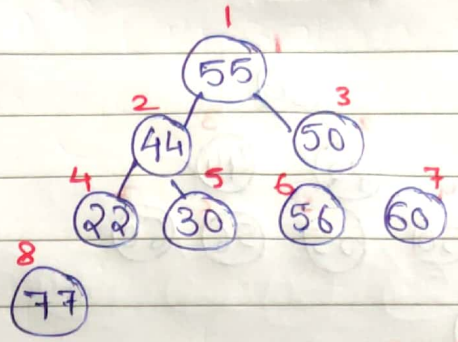
	1	2	3	4	5	6	7	8
A:	50	55	56	22	30	44	60	77

	1	2	3	4	5	6	7	8
A:	56	55	50	22	30	44	60	77

Scanned with CamScanner

⇒ 3rd Iteration

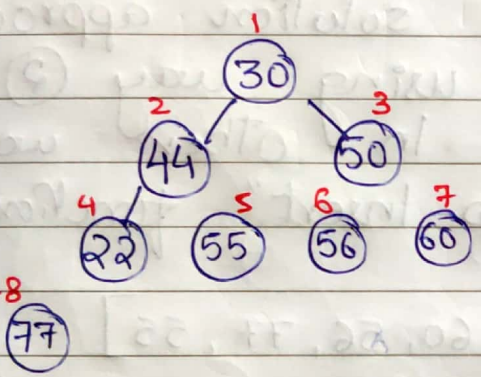


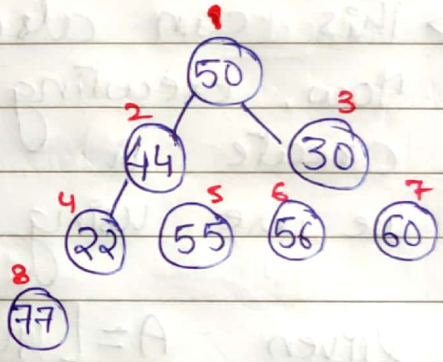


	1	2	3	4	5	6	7	8
A:	44	55	50	22	30	56	60	77

	1	2	3	4	5	6	7	8
A:	55	44	50	22	30	56	60	77

⇒ 4th Iteration

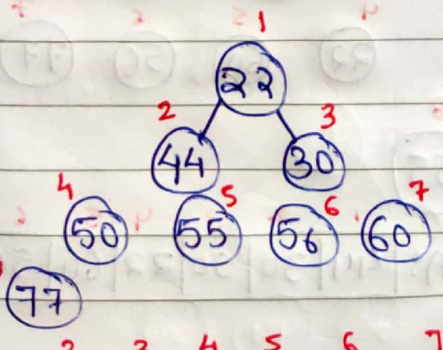


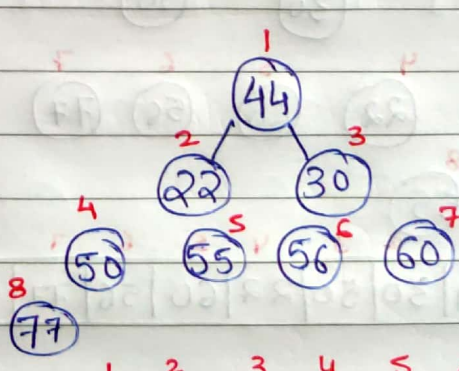


	1	2	3	4	5	6	7	8
A:	30	44	50	22	55	56	60	77

	1	2	3	4	5	6	7	8
A:	50	44	30	22	55	56	60	77

⇒ 5th Iteration



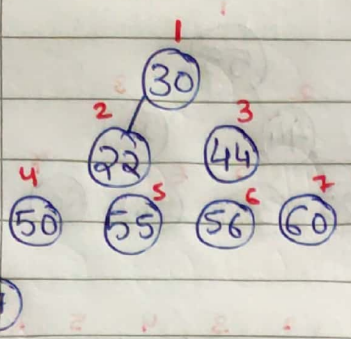


	1	2	3	4	5	6	7	8
A:	22	44	30	50	55	56	60	77

	1	2	3	4	5	6	7	8
A:	44	22	30	50	55	56	60	77

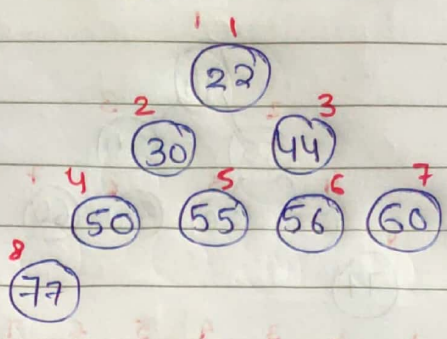
Scanned with CamScanner

⇒ 6th Iteration



	1	2	3	4	5	6	7	8
A:	30	22	44	50	55	56	60	77

⇒ 7th Iteration



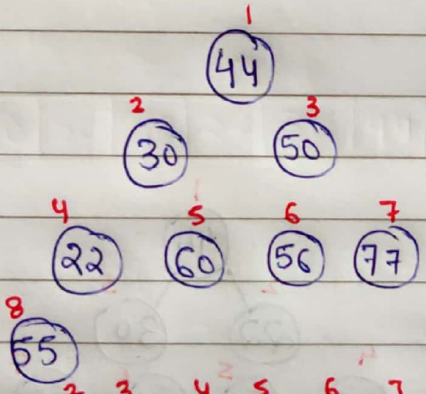
	1	2	3	4	5	6	7	8
A:	22	30	44	50	55	56	60	77

"SORTED"

⇒ This can also be the solution approach. Now, executing STEP 1 using way (2) to create a max-heap. In other words we are using "Max-heap-Insert" function.

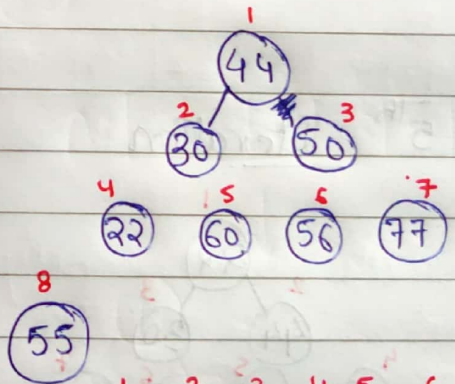
Given A = [44, 30, 50, 22, 60, 56, 77, 55]

→ Insert 44



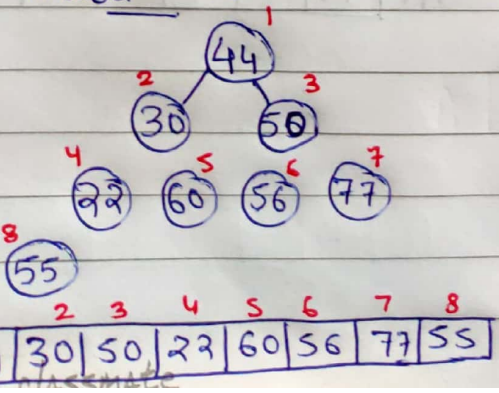
	1	2	3	4	5	6	7	8
A:	44	30	50	22	60	56	77	55

→ Insert 30



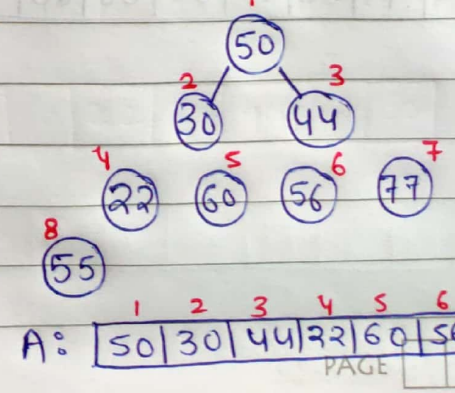
	1	2	3	4	5	6	7	8
A:	44	30	50	22	60	56	77	55

→ Insert 50



	1	2	3	4	5	6	7	8
A:	44	30	50	22	60	56	77	55

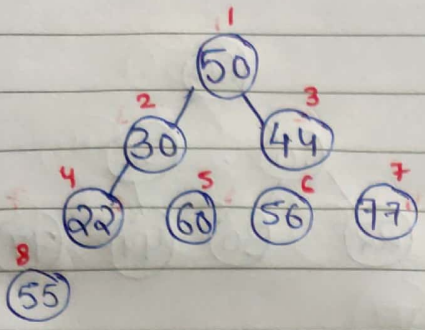
→



	1	2	3	4	5	6	7	8
A:	50	30	44	22	60	56	77	55

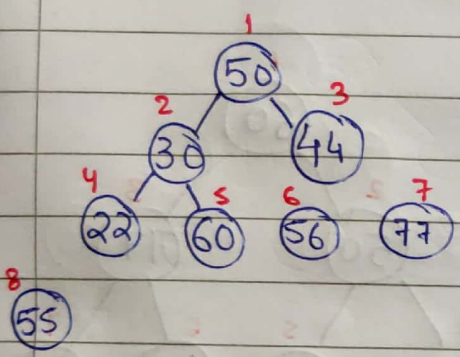
Scanned with CamScanner

→ Insert 22.

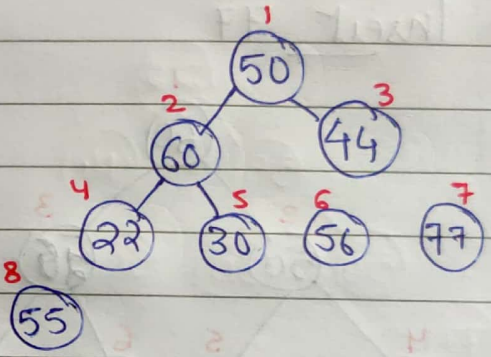


	1	2	3	4	5	6	7	8
A:	50	30	44	22	60	56	77	55

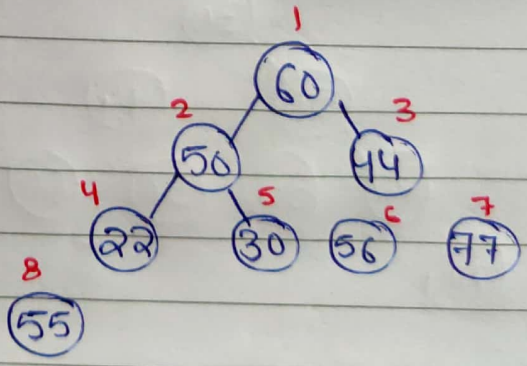
→ Insert 60



	1	2	3	4	5	6	7	8
A:	50	30	44	22	60	56	77	55



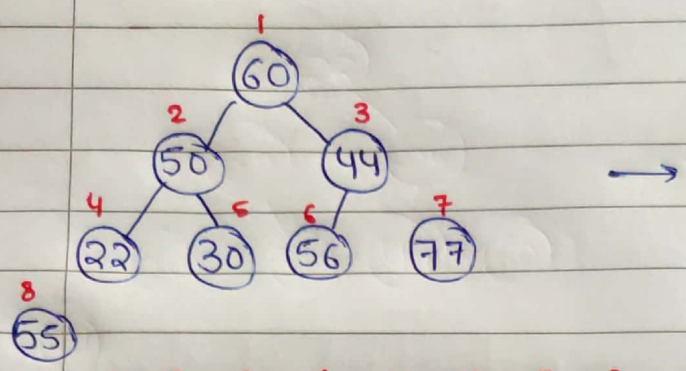
	1	2	3	4	5	6	7	8
A:	50	60	44	22	30	56	77	55

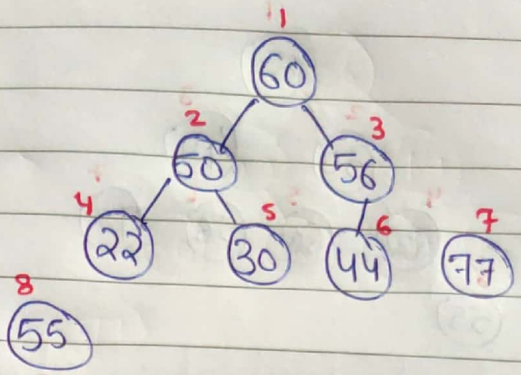


	1	2	3	4	5	6	7	8
A:	60	50	44	22	30	56	77	55

Scanned with CamScanner

→ Insert 56

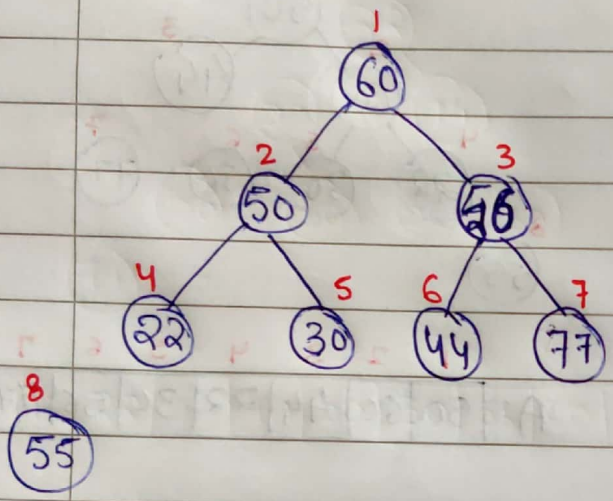


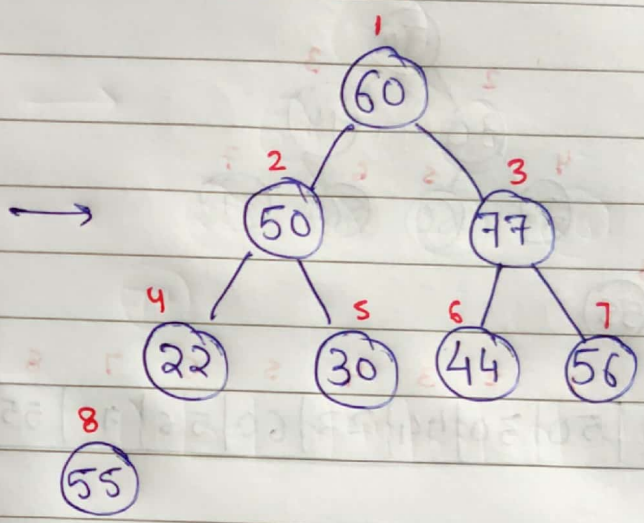


	1	2	3	4	5	6	7	8
A:	60	50	44	22	30	56	77	55

	1	2	3	4	5	6	7	8
A:	60	50	56	22	30	44	77	55

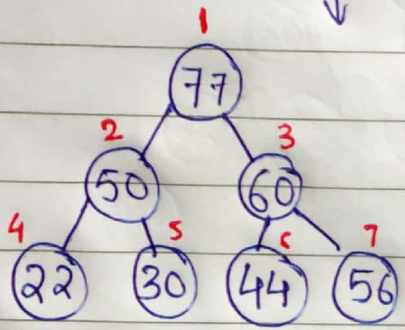
→ Insert 77





	1	2	3	4	5	6	7	8
A:	60	50	56	22	30	44	77	55

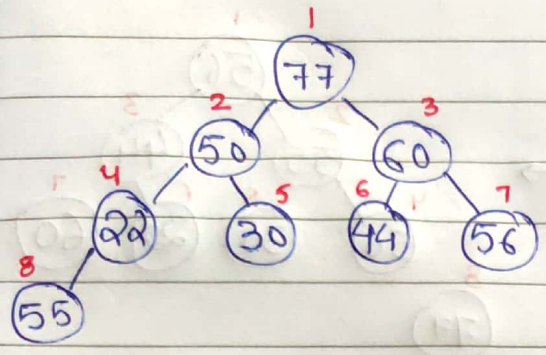
	1	2	3	4	5	6	7	8
A:	60	50	77	22	30	44	56	55

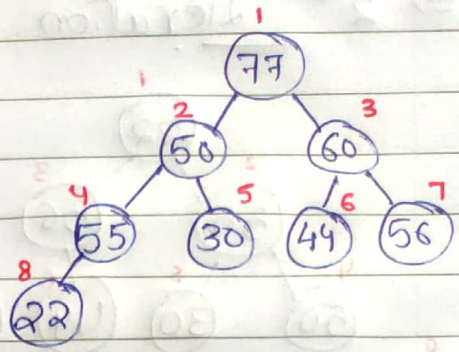


	1	2	3	4	5	6	7	8
A:	77	50	60	22	30	44	56	55

Scanned with CamScanner

→ Insert 55

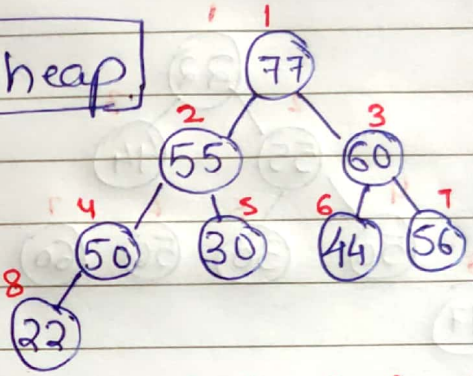




A:	1	2	3	4	5	6	7	8
	77	50	60	22	30	44	56	55

A:	1	2	3	4	5	6	7	8
	77	50	60	55	30	44	56	22

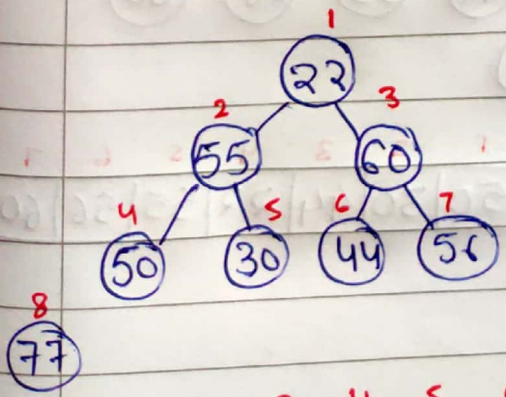
Max-heap

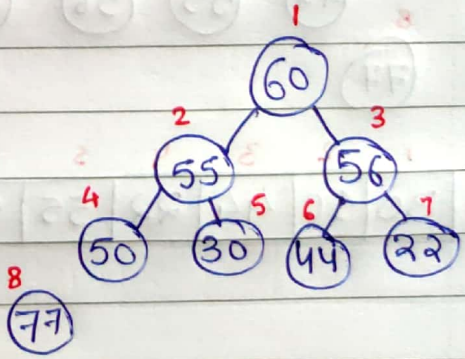


A:	1	2	3	4	5	6	7	8
	77	55	60	50	30	44	56	22

STEP II : Execute rest of the steps of heapsort algorithm showing contents at each intermediate step.

⇒ 1st Iteration



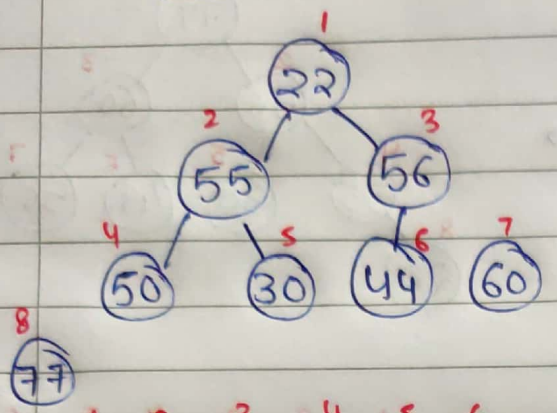


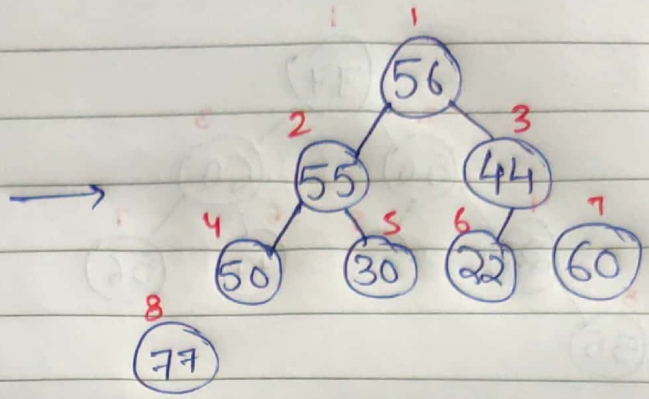
A:	1	2	3	4	5	6	7	8
	22	55	60	50	30	44	56	77

A:	1	2	3	4	5	6	7	8
	60	55	56	50	30	44	22	77

Scanned with CamScanner

⇒ 2nd Iteration

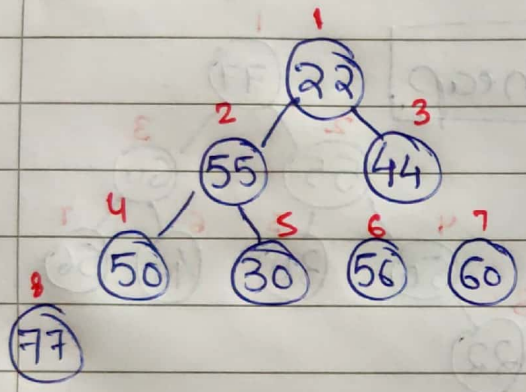


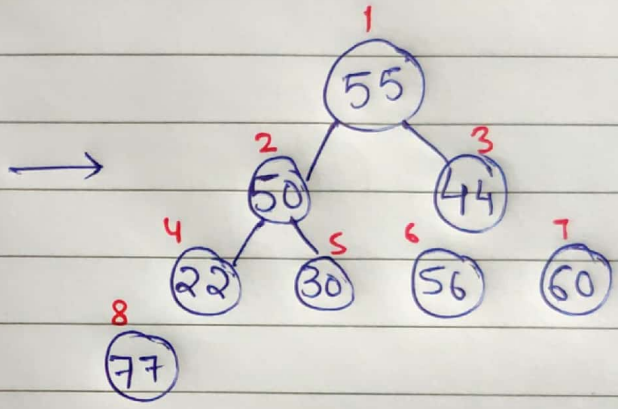


A:	1	2	3	4	5	6	7	8
	22	55	56	50	30	44	60	77

A:	1	2	3	4	5	6	7	8
	56	55	44	50	30	22	60	77

⇒ 3rd Iteration

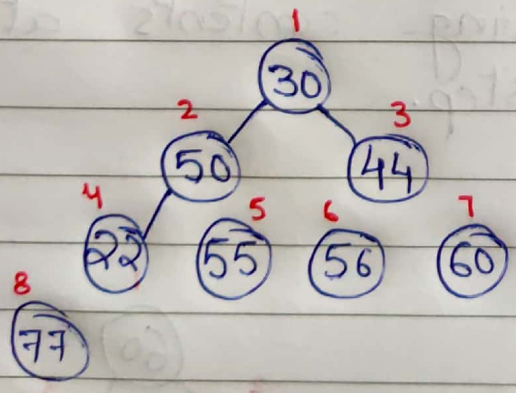


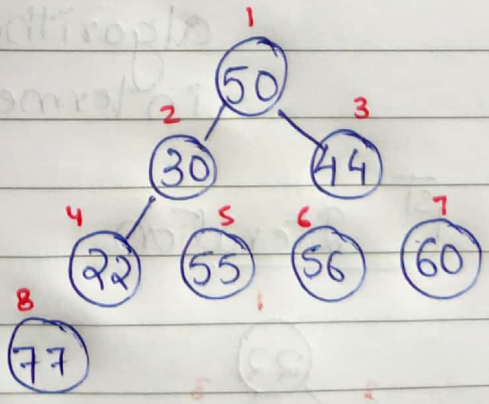


A:	1	2	3	4	5	6	7	8
	22	55	44	50	30	56	60	77

A:	1	2	3	4	5	6	7	8
	55	50	44	22	30	56	60	77

⇒ 4th Iteration



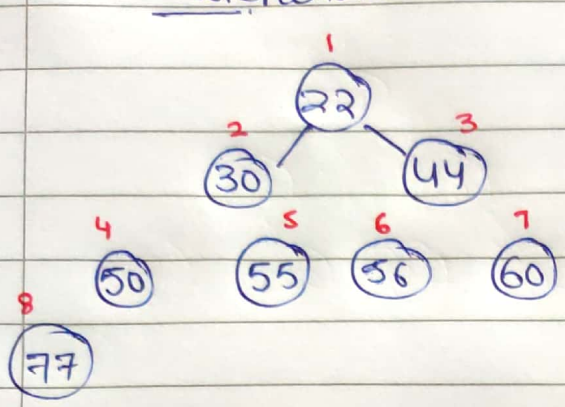


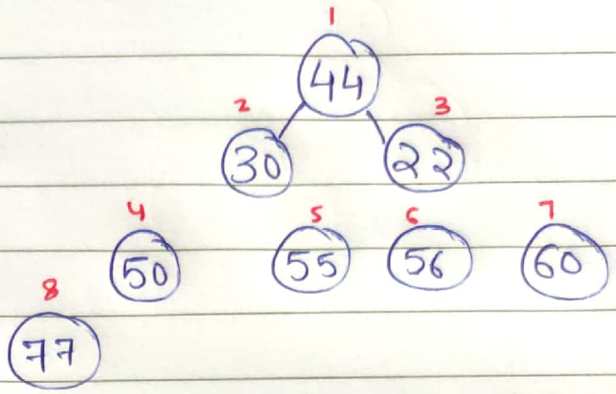
A:	1	2	3	4	5	6	7	8
	30	50	44	22	55	56	60	77

A:	1	2	3	4	5	6	7	8
	50	30	44	22	55	56	60	77

Scanned with CamScanner

⇒ 5th Iteration

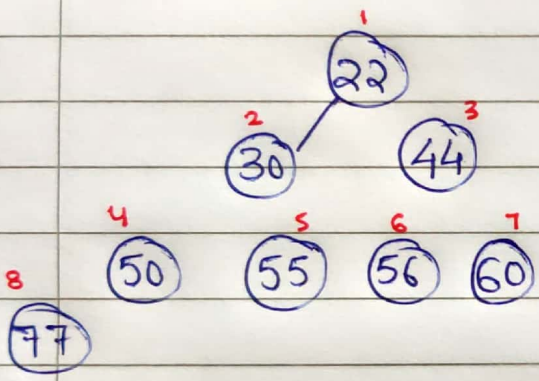


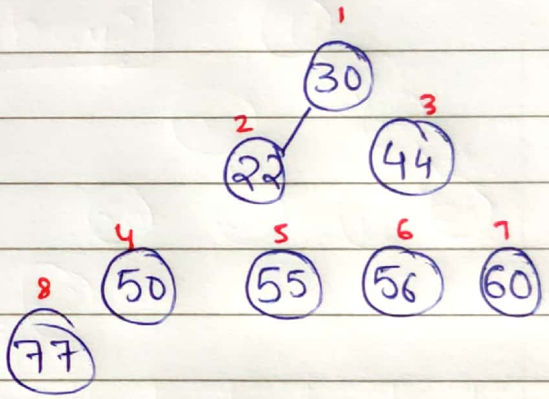


	1	2	3	4	5	6	7	8
A:	22	30	44	50	55	56	60	77

	1	2	3	4	5	6	7	8
A:	44	30	22	50	55	56	60	77

⇒ 6th Iteration

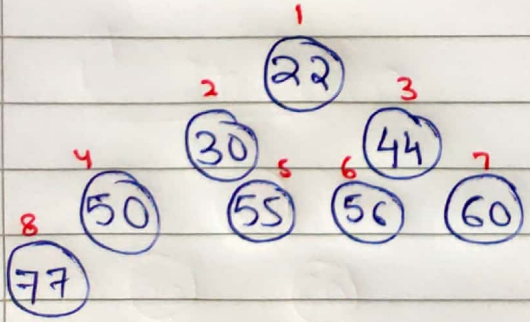




	1	2	3	4	5	6	7	8
A:	22	30	44	50	55	56	60	77

	1	2	3	4	5	6	7	8
A:	30	22	44	50	55	56	60	77

⇒ 7th Iteration



	1	2	3	4	5	6	7	8
A:	22	30	44	50	55	56	60	77

"SORTED"

Scanned with CamScanner