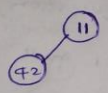
Assignment - 6:

1 ans: Nodes to insert,
42, 11, 28, 8, 13, 61, 18

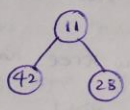
Step 1: Insert (42)

Step 2: Insert 11

Since 11 is less than 42, it can't be in the child of 42. So we perform "percolate up" and then insert it there.

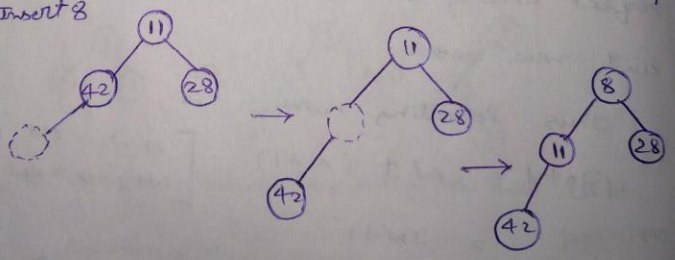


Step 3: Insert 28

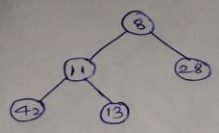


The ordering "is OK" even after inserting at the next available position. So, no need of "percolate up"

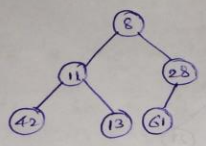
Step 4: Insert 8



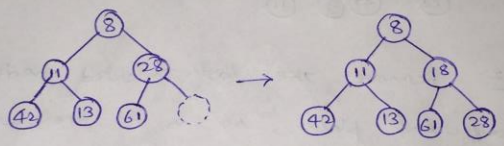
Step 5: Insert 13



Step 6: Insert 61



Step 7:



2ans: In array notation:

Parent = $i/2$

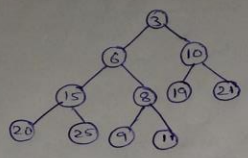
Left child = $2i$ and Right child = $2i+1$

and No node is in position 0

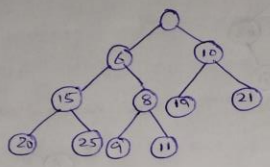
∴ the tree notation is:

	8	11	18	42	13	61	28		
0	1	2	3	4	5	6	7	8	9

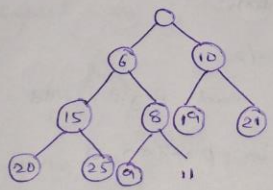
Sol: Given Binary Heap:



Step 1: Remove the root i.e 3.



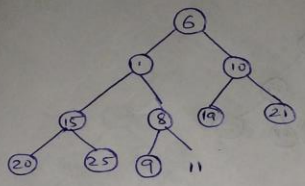
Step 2: Remove the last inserted node and store in a different place, so as to preserve the complete tree property.



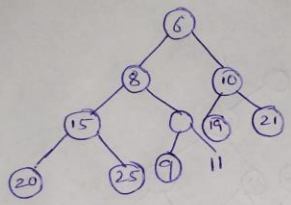
Step 3: Test the hole at the root, if element stored at a different place i.e, 11 can be inserted there. And also the order property is preserved.

Here 11 can be placed as both childs 6 & 10 are lesser than 11. So pick the least among both i.e, 6 is moved into it and hole is pushed down.

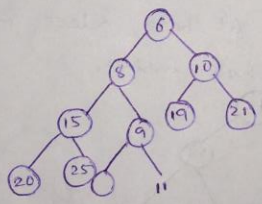
The result is,



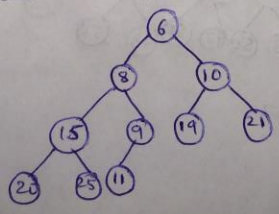
Step 4: Here 11 also doesn't fit because 8 is less than 11. So it ~~must~~ should be pushed down.



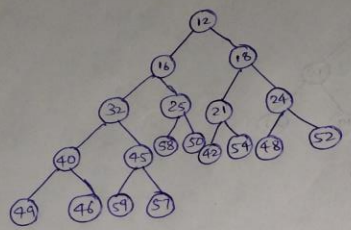
Step 5: 11 doesn't fit into hole here also. So 9 should be moved upward and hole is pushed down



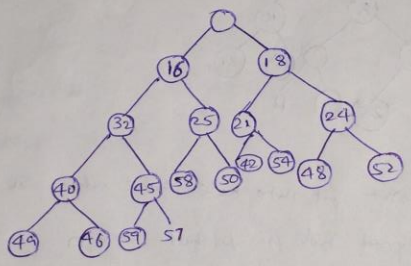
Step 6: 11 should be pushed in the hole here.



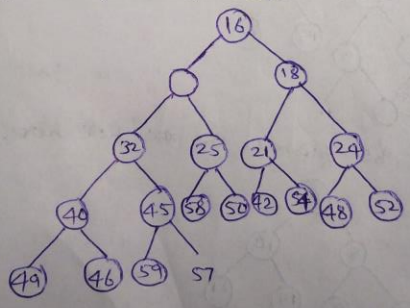
3 4 ans: Given heap,



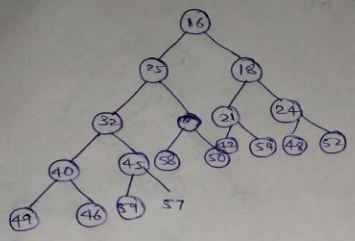
step 1: ~~remove the~~ deleteMinimum number, that ~~should~~ be the root i.e, 12 and also remove last inserted node in the ~~binary~~ heap 57 and put 57 in a temporary place.



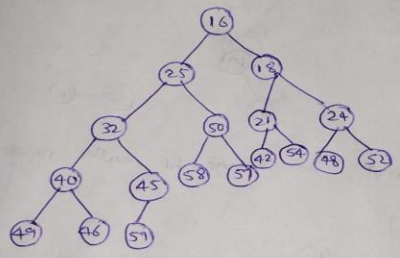
step 2: since 57 doesn't fit here, select min among children and push the hole inside.



step 3: since 57 doesn't fit here also, Push the hole inside.

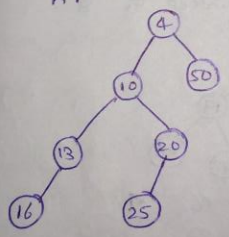


step 4: 57 doesn't fit here also, so push the hole down and place 57 in the last position.

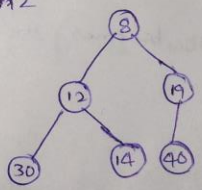


Sons: The leftist heaps to be merged,

h1



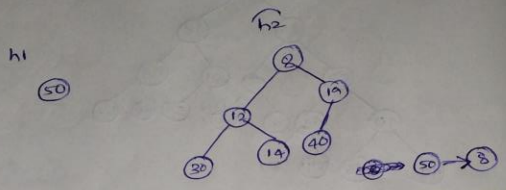
h2



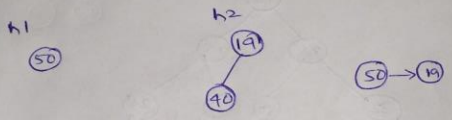
h1 has smaller root, so merge h2 with right subtree of h1

8 → 4

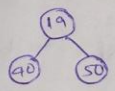
Here h1 has smaller root, so merge h2 with right subtree of h1.



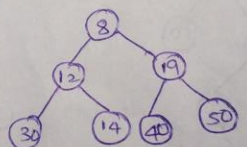
h2 has smaller root, so merge h1 with right subtree of h2



h2 has smaller root, so merge h1 with right subtree of h2.



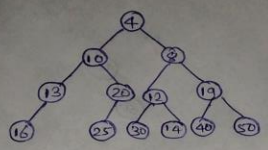
19 becomes the right subtree of 8



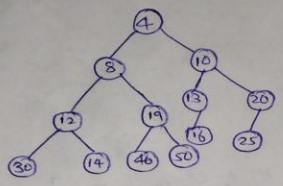
now, 8 becomes right subtree of 4

with

Now, 8 becomes the right subtree of 4

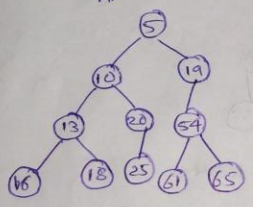


subtree swap due to leftist violation.

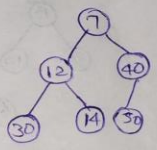


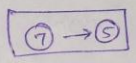
Ans: Given leftist heaps,

h1



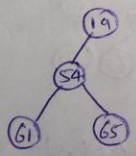
h2



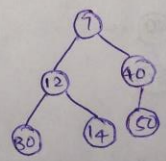


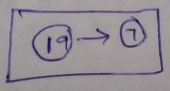
h1 has smaller root, so merge h2 with right subtree of h1

h1



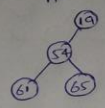
h2





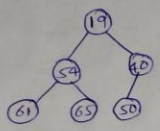
h2 has smaller root, so merge h1 with right subtree of h2

h1

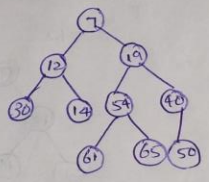


h2

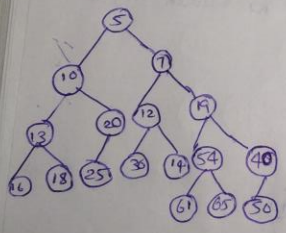
h1 has smaller root, so merge h2 with right subtree of h1



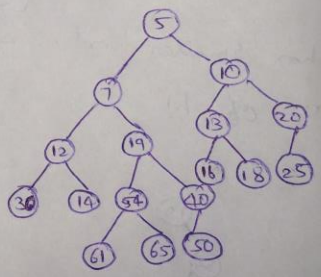
19 becomes right subtree of 7



7 becomes right subtree of 5



subtree swap



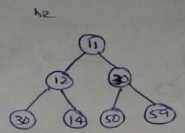
Subtree swaps due to leftist violation.

right subtree
th smaller

Ans: Given heaps,

h1

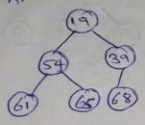
h2



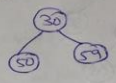
19 → 11

right subtree h2 has smaller node, so merge h1 with right subtree of h2

h1



h2

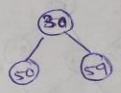


30 → 19

h1 has smaller node, so merge h2 with ~~smaller~~ right subtree of h1

h1

h2



39 → 30

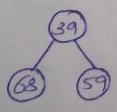
h2 has smaller node, so merge h1 with right subtree of h2

h1

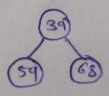
h2

59 → 39

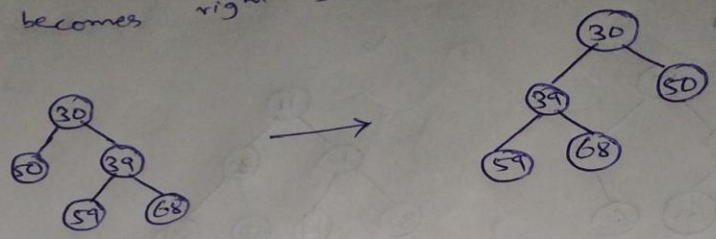
h1 has smaller node, so merge h2 with right subtree of h1



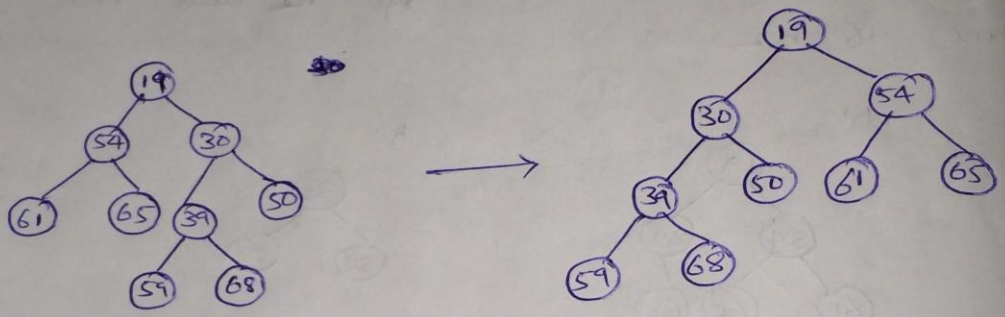
swap →



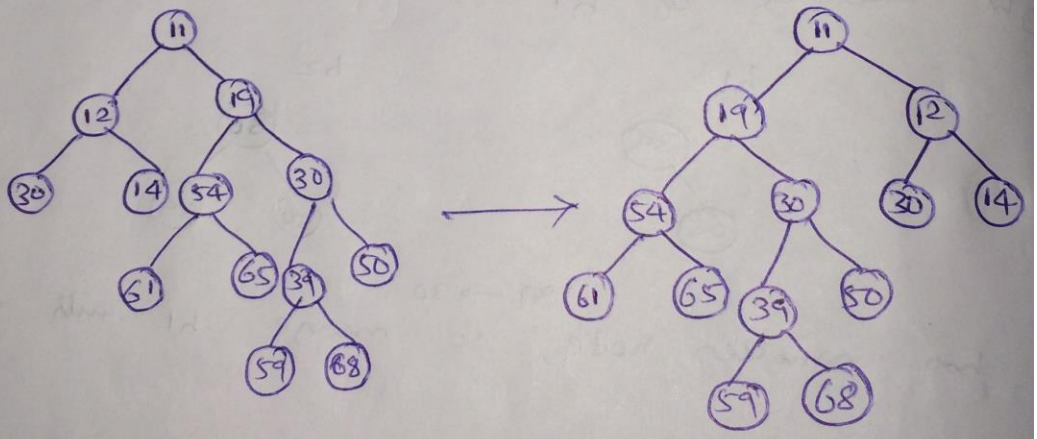
39 becomes right subtree of 30 and swap it



30 becomes right subtree of 19 & swap it

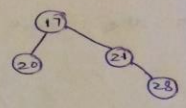


19 becomes right subtree of 11 & swap it

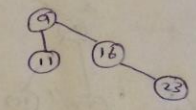


Sol: Given, binomial queues

h1 :

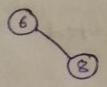


h2 :



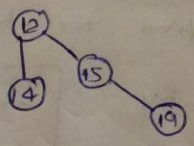
Step 1: h1 & h2 has binomial tree nodes of height 0, So we merge them to get a tree of height 1. which is put in h3.

h3 :



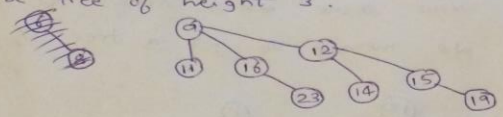
The node with larger value is put in the right subtree of smaller value, so put (8) into (6) right subtree

Step 2: binomial trees of height 1 are merged to get tree of height 2.



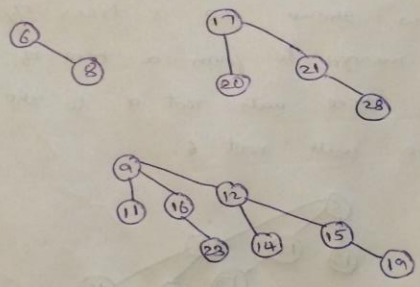
Step 3: Now there are 3 binomial trees of height 2 (one newly formed one and the from each of the h1 & h2).

we merge the tree with root 12 to the right child of the tree with root 9, to form a tree of height 3.



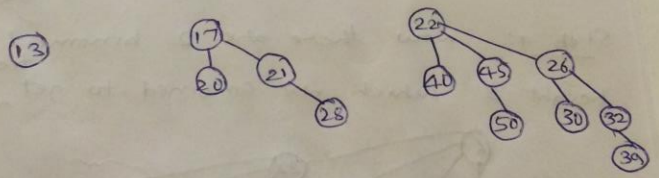
Step 4: Thus the final binomial tree,

h3:

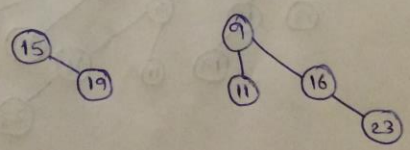


Q ans: Given Binomial Queues, 20 = 10100

h1 :



h2:

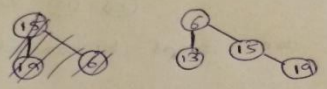


Solution:

Step 1: h_1 and h_2 has trees of height 0, so the
tree of
so we megre them to get a height 1. in
h_3:

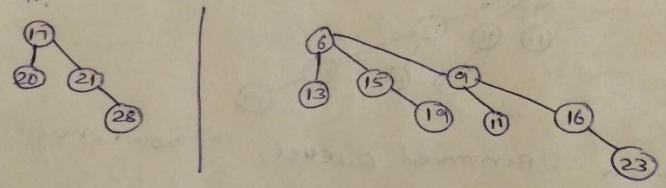
Step 2: Now there are 2 trees of height 1
which are merged to get a tree of height 2

13
1
20

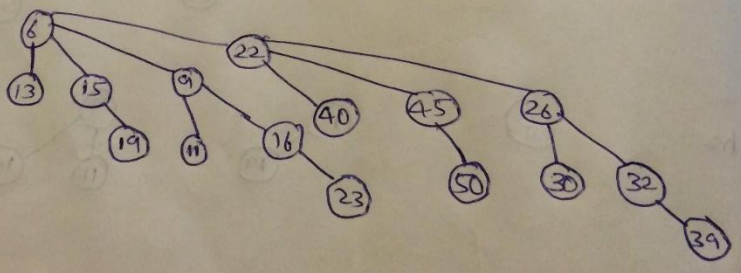


Step 3: Now there are 3 trees of height 2,
of which 2 are merged to form a tree of height 3.
So megre tree with root 9 to the right
child of tree with root 6

h_3:

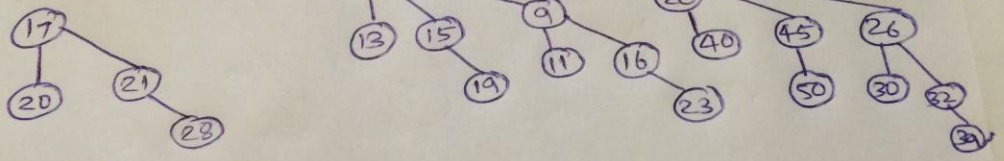


Step 4: Now there are 2 binomial trees of
tree of
height 3 which are combined to get height 4



so the Final binomial tree after merging is,

h3:

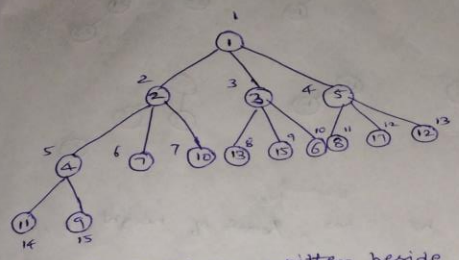


swap it

swap it

10 ans:

For the case of 3-heap shown below,



Here the numbers written beside the nodes in 02 represent the position in the array where they are inserted.

(a) so they can be inserted in array as,

	1	2	3	5	4	7	10	13	15	6	8	17	12	11	9	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

(b) The formulae to find Left, Right and middle children from any parent are.

Left child = $3i - 1$

Middle child = $3i$

Right child = $3i + 1$

(c) The formula to find the parent for any child is given by $\frac{i+1}{3}$

Here it is an integer division is used

if not it is $\left\lfloor \frac{i+1}{3} \right\rfloor$