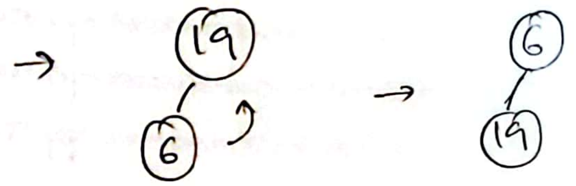
Q2.] Solution.→

Input Array → [19, 6, 8, 11, 4, 5]

We start by adding 19 as we are reading the array from left to right.

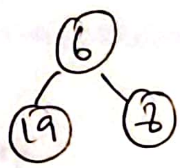
Step 1.) → 19 [Till now, 19 is the only node]

Step 2.) We add 6 to the tree. It will be added as a child to node 19 but will be bubbled in place of 19 as it is less than 19

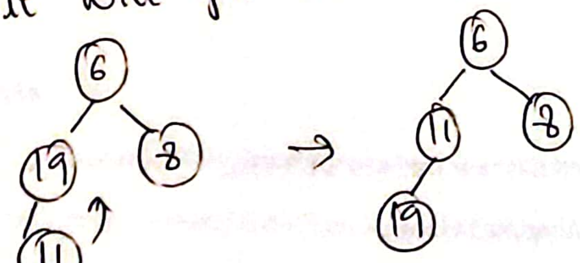


19 will be child of 6.

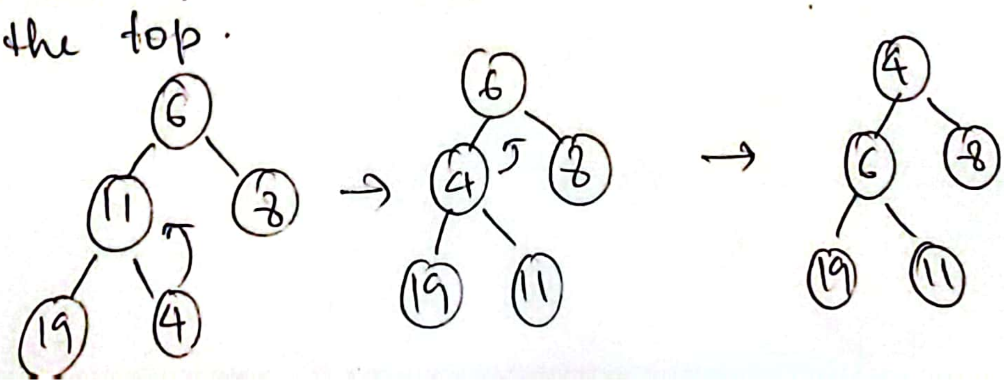
Step 3.) We add 8 to this tree. 8 will be added as a child of 6 and as 8 is larger than 6, it stays as 6's child



Step 4.) We add 11 as left child of 19 and as 11 is greater than 19, it will get bubbled to the top at 19's initial place

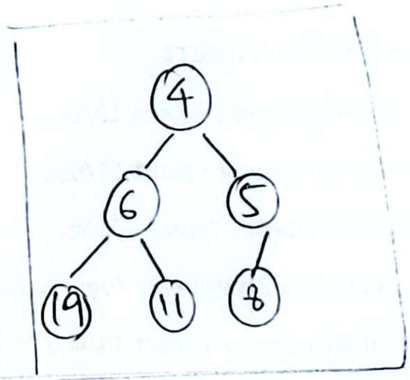


Step 5.) We add 4 as the right child of 11, but it bubbles to the top.



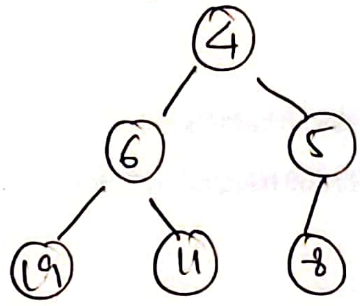
Step 6.] We add 5 as the left child of 8 node. It will be swapped with node 8 as $5 < 8$

The final min heap will be →

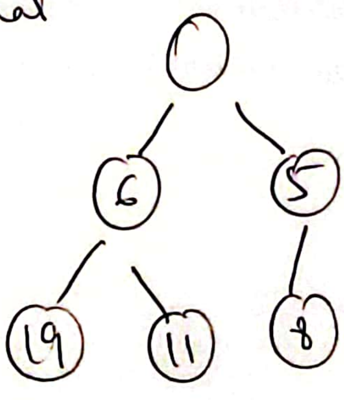


Q b.) Show a tree that can be the result of after the call to deletemin() on the above heap.

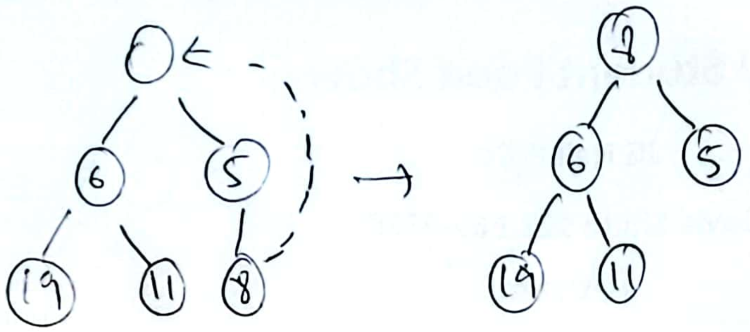
Answer - The binary tree we have here is as shown



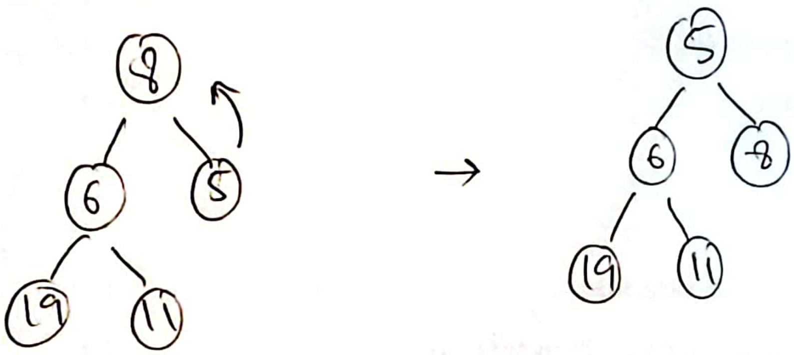
On calling deletemin(), 4 is removed and there is a vacant space at that spot.



As there is a free space at that spot, the last node which is 8 is moved there



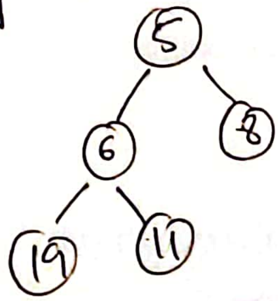
As we have to maintain min-heap property and as 8 is greater than its children, we need to swap 8 with one of its children. We compare the children & swap it with the ~~&~~ smallest child.



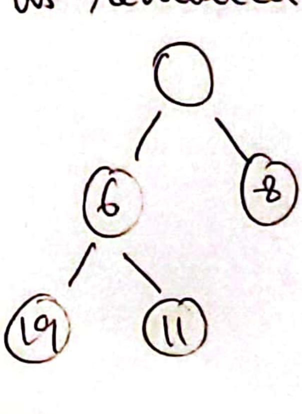
This is a tree which is a result after delete min() is called.

Q c) Show a tree after another call to deletemin()

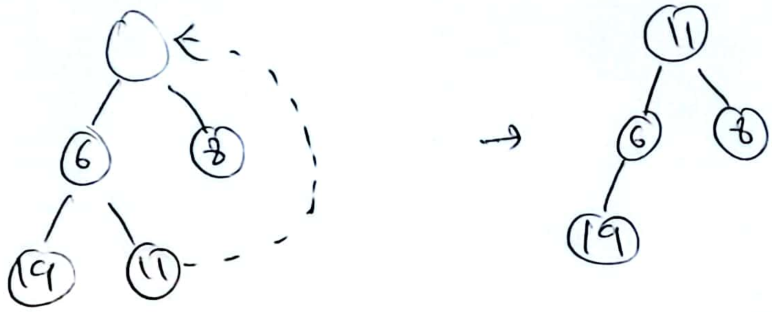
Answer - The binary tree we have now is



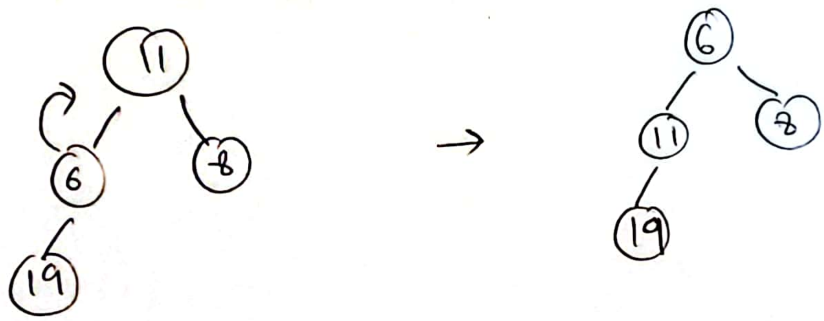
On calling deletemin(), 5 is removed and there is a vacan
space at that spot →



As there is a vacant space at that spot, the last node which is 11 is moved there



As we have do maintain min-heap property and here, 11 is greater than both of its children, we swap it with the smallest child which is 6



This is the resultant tree