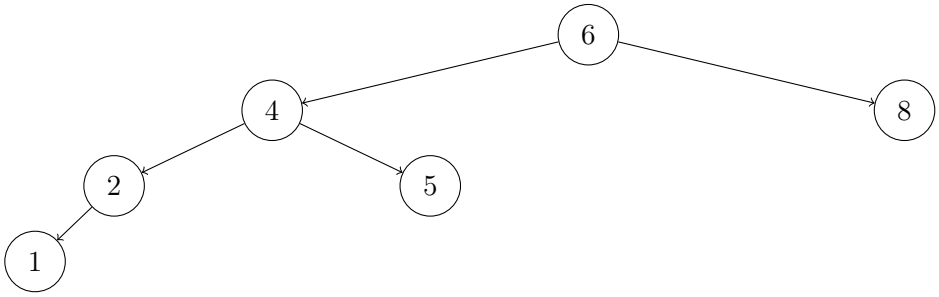
Part II: Written Lab

AVL Tree Questions

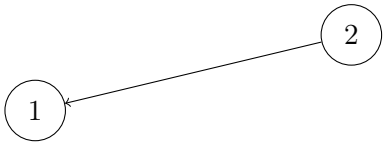
1. Perform a right rotation on the root of the following tree. Be sure to specify the X, Y, and Z subtrees used in the rotation.



Solution:
Here, we are performing right rotation on the root, so center of rotation is:

After rotation the new root will be:

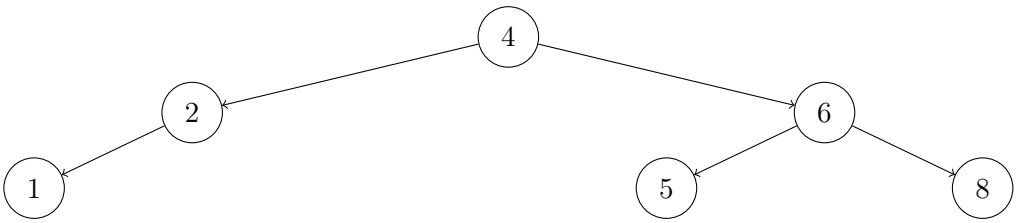
For this rotation, X subtree is:



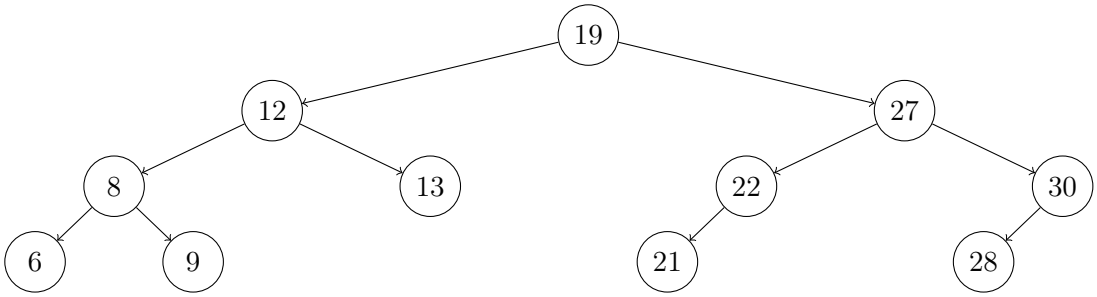
Y subtree is:

Z subtree is:

Now, after performing the right rotation, the new root will have left pointer to X subtree and the right pointer to the center of rotation node, and the Y subtree will now be pointed by left pointer of center of rotation. And the rotated tree will be as follows:



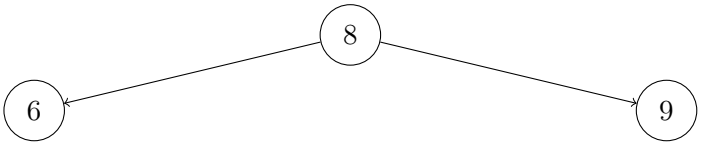
2. Show the left rotation of the subtree rooted at 12. Be sure to specify the X, Y, and Z subtrees used in the rotation.



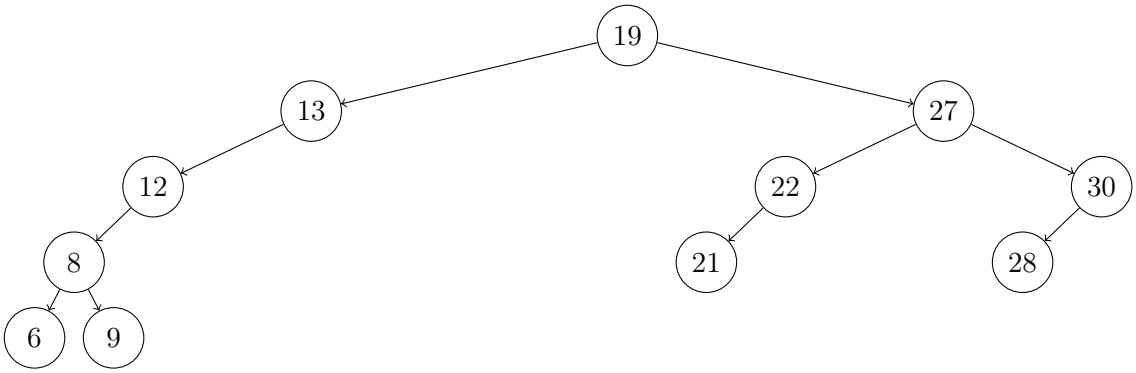
Solution:
Here, we are performing left rotation of the subtree rooted at 12, so center of rotation is:

After rotation the new root of the subtree (i.e. left child of node with key 19) will be

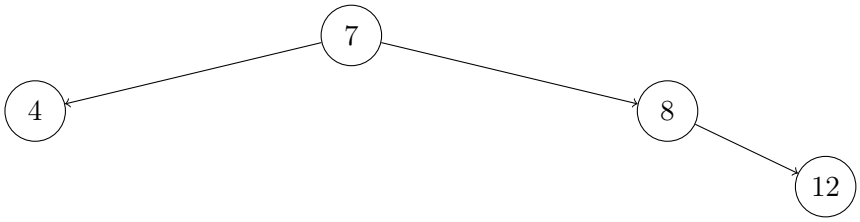
For this rotation, X subtree is:



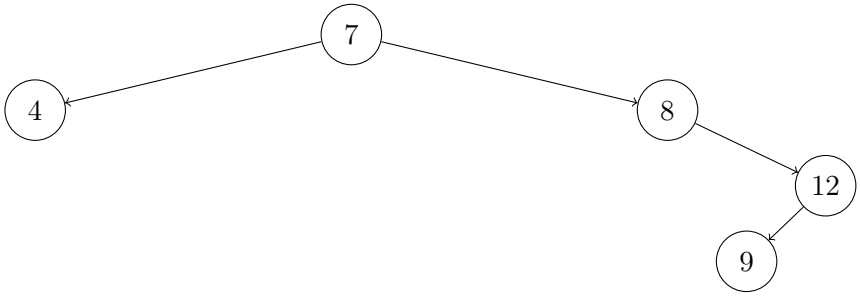
Y subtree is empty (as node 13's left pointer is a null pointer)
Similarly, Z subtree is also empty (as node 13's right pointer is a null pointer)
Now, after performing the left rotation of that the subtree, the new root of the subtree (left child of node with key 19) will have left pointer to the center of rotation node and the center of rotation node will have the right pointer to Y subtree. And the rotated tree will be as follows:



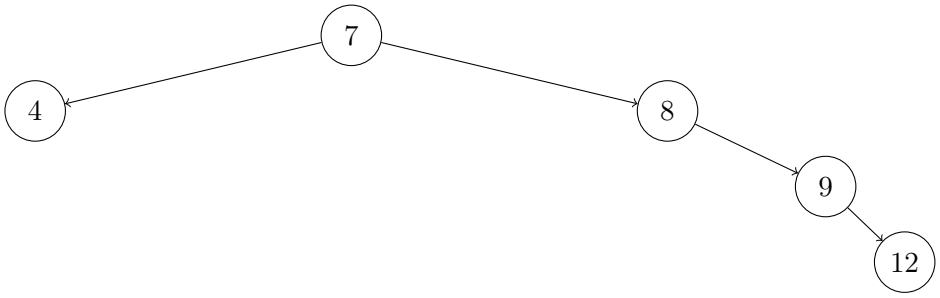
3. Using the appropriate AVL tree algorithm, insert the value 9 into the following tree. Show the tree before and after rebalancing.



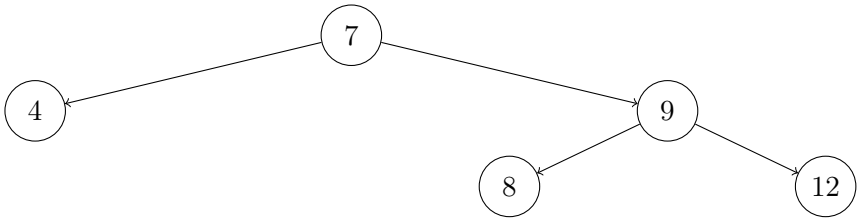
Solution:
Here 9 is less than 12, so when we insert 9 it will be added as the left child of the node with value 12. The tree right after the insert will look like this before re-balancing:



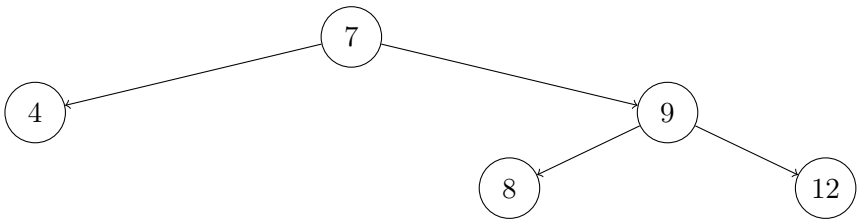
After insertion, the height of left-subtree at node 8 is -1 while the height of right-subtree at node 8 is 1 so the difference in height is greater than 1 i.e. at node 8 the subtree is right heavy. So we need to rebalance the tree at node 8. More specifically, node 8 it's right-left heavy condition. So, first we will do a right rotation at node 12 and the tree after this step will look like:



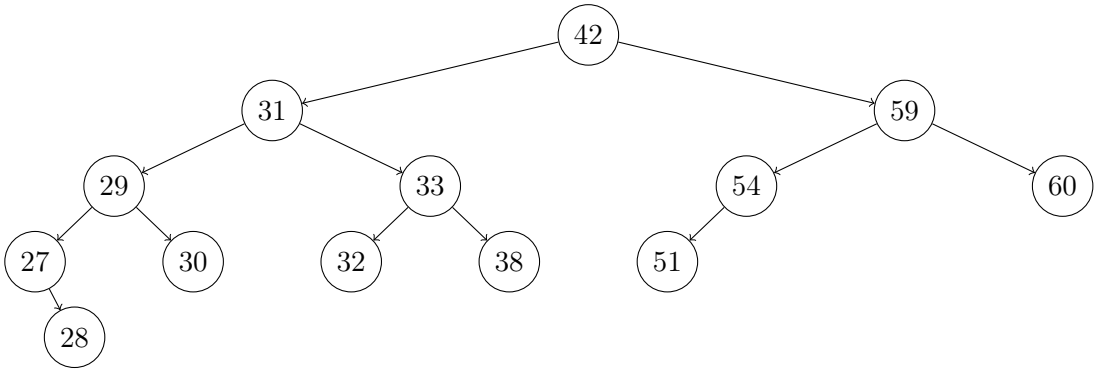
Now, at node 8 the subtree is right-right heavy so we will do a left rotation at node 8 which will rebalance the tree as follows:



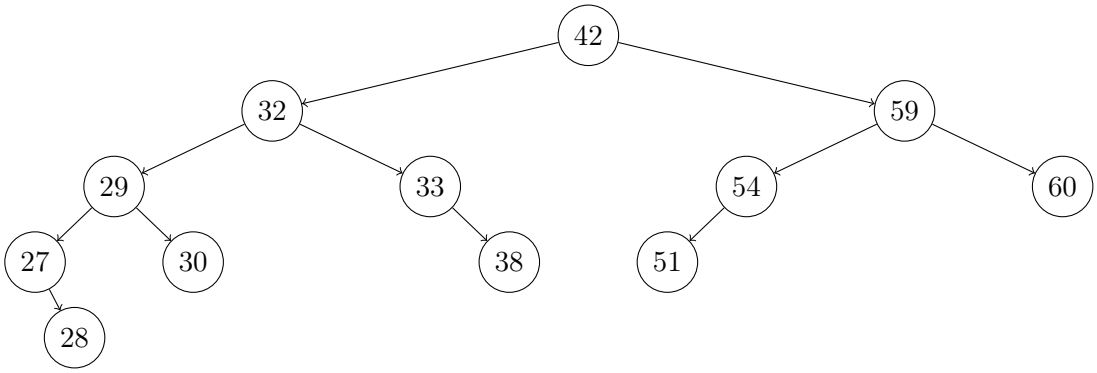
Now, as we move up we will be at the root of the BST and at the root the AVL invariant holds so we have obtained our final balanced AVL tree which is as follows:



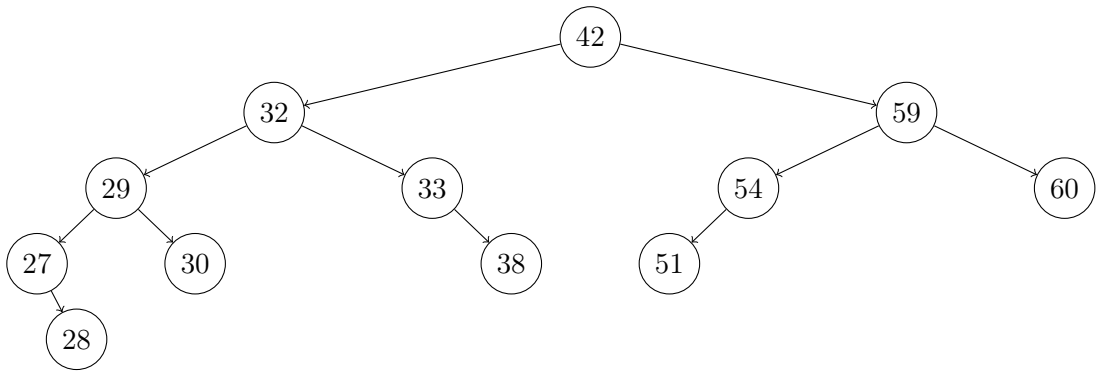
4. Using the appropriate AVL tree algorithm, remove the value 31 from the following tree. Show the tree before and after rebalancing.



Solution:
First we will change the value of current node 31 with the minimum key from the right-subtree and remove the node with minimum key from the right subtree. After changing the value and removing the node with minimun key from right subtree our BST will be as follows:

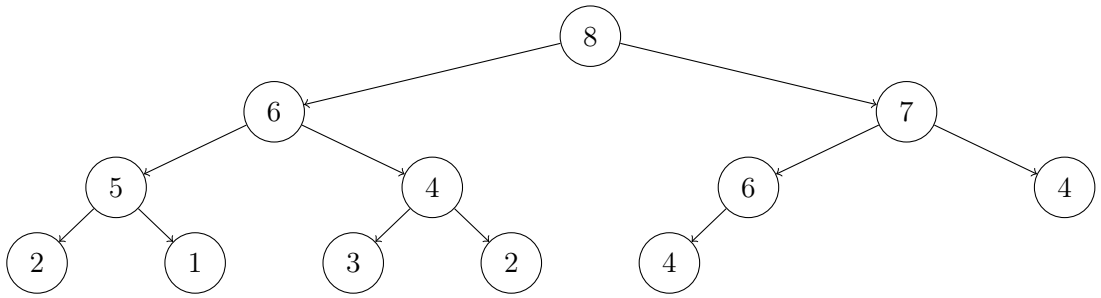


Now we will move from node 33 upwards to check the AVL invariant and rebalance if needed.
At node 33, height of left subtree is -1 and height of right subtree is 0 so AVL invariant holds.
Moving up, at node 32: height of left subtree is 2 and height of right subtree is 1 so AVL invariant holds.
Moving up further, we are at root of the BST i.e. node 42 where height of left subtree is 3 and height of right subtree is 2 so the AVL invariant holds here as well.
Hence our BST is now an AVL Tree and holds the AVL invariant.
Therefore,
there will not be any rebalancing after the removal and our final tree will look like:

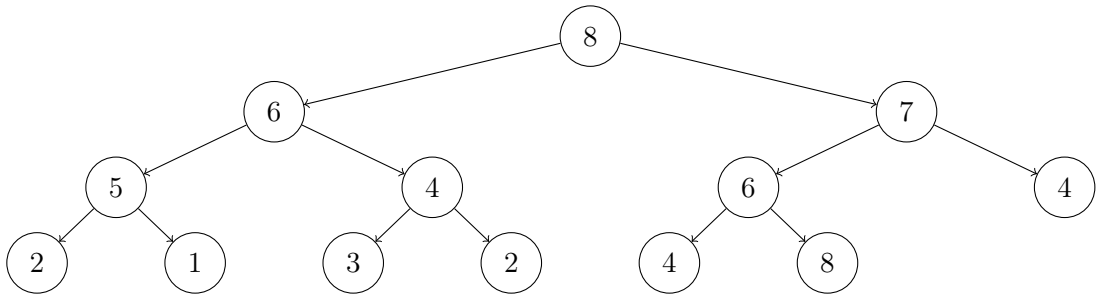


Heap Questions

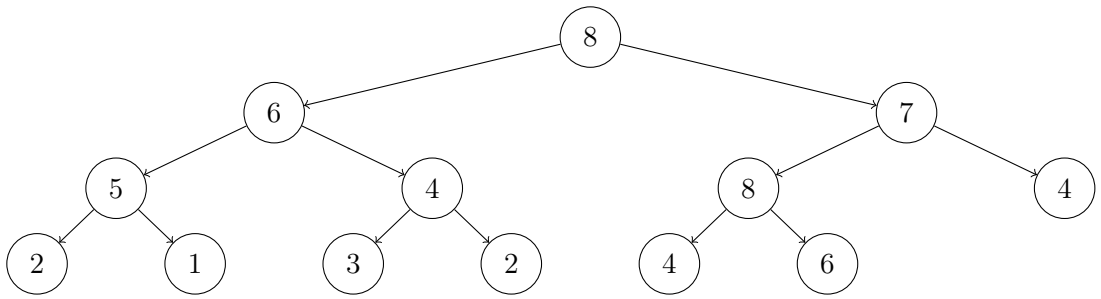
1. Show the addition of the element 8 to the max-heap below. First, show the addition of 8 to the tree; then, show each bubbling step.



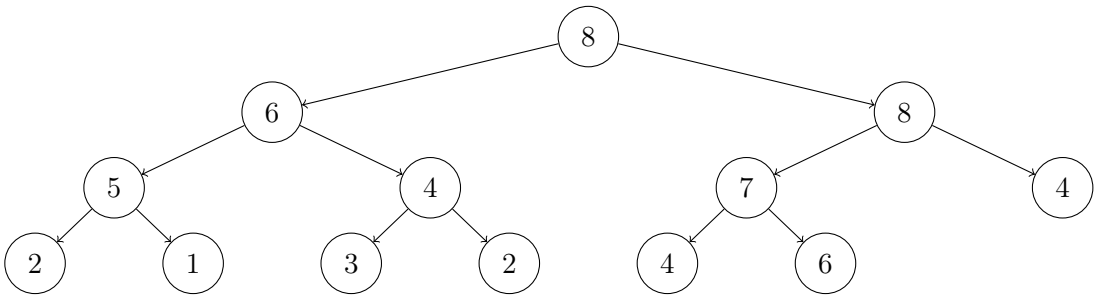
Solution:
A max-heap is a complete binary tree so its last level is always packed to the left. Following this the addition of 8 will happen by creating a new node 8 as a right child of node 6 in the right subtree of root node. Just after addition the max-heap will be as below:



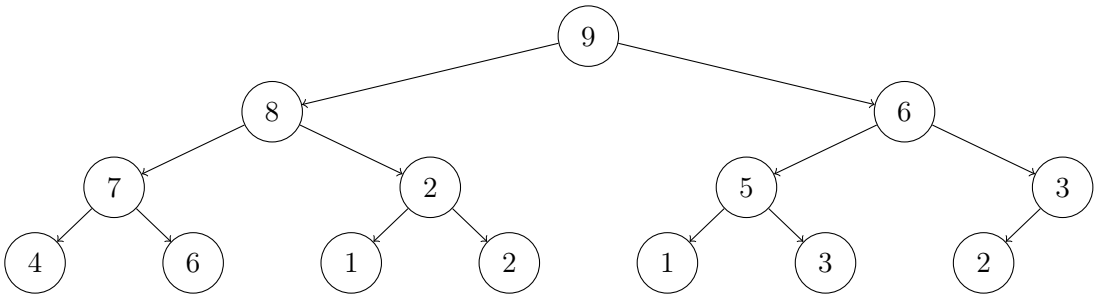
Now, max-heap has a property that every node has a priority $\geq$ its children's priority. Now to make this property to hold after we add a new element, we will be bubbling up from that node unitl we restore this property of the max-heap. Firstly, as the parent node 6 is less than node 8, we will swap 8 and 6 s.t. 6 will now be the right child of 8.



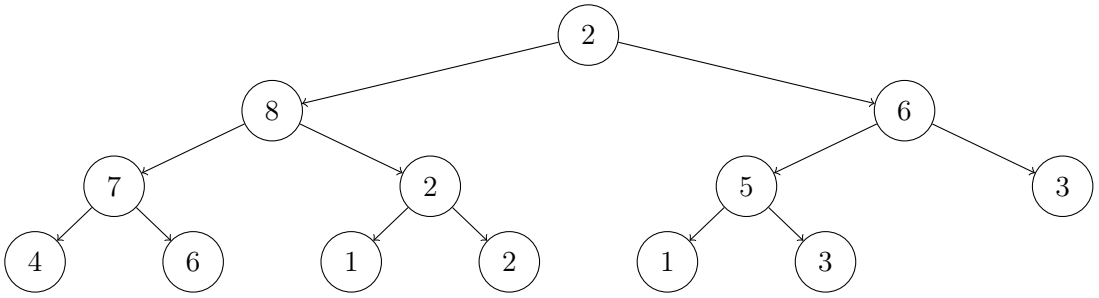
Now, the parent node of 8 (i.e 7) is less than 8 so we will swap node 7 and node 8. Finally the parent node of 8 (i.e. root node 8) has the same priority as 8 so the property of max-heap has been restored. Hence, the final max-heap after addition of the element 8 and bubbling up will be:



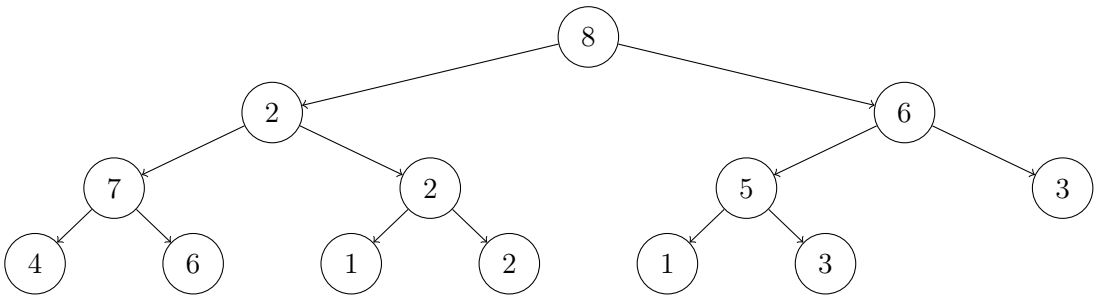
2. Show the removal of the top element of this max-heap. First, show the swap of the root node; then, show each bubbling step.



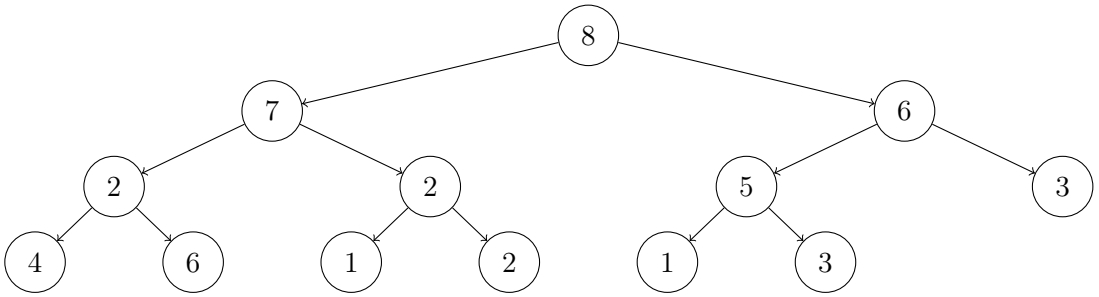
Solution:
To ensure the max-heap after is still a complete binary tree, we will swap the root(top element) with last element and delete the root(from last) so that previous last in now the root i.e. we will swap 9 with 2 (7th node from left in the 3rd level or, root->right->right->left) and delete 9 so that 2 is now the new root. The max-heap after the removal of 9 will be:



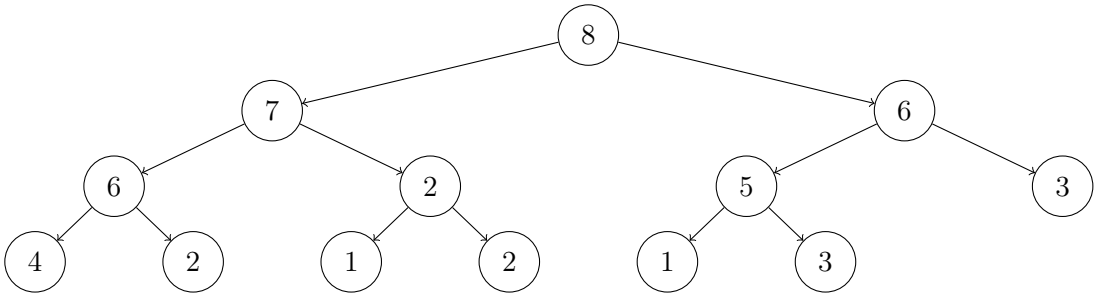
We know, max-heap has a property that every node has a priority $\geq$ its children's priority. Now to make this property to hold after we remove the top element, we will be bubbling down from the root until we restore this property of the max-heap. Firstly, $2 < 8$ and 8 is the greatest child of 2 so we will swap element 2 with element 8 so that 8 is now the new root of the binary tree.



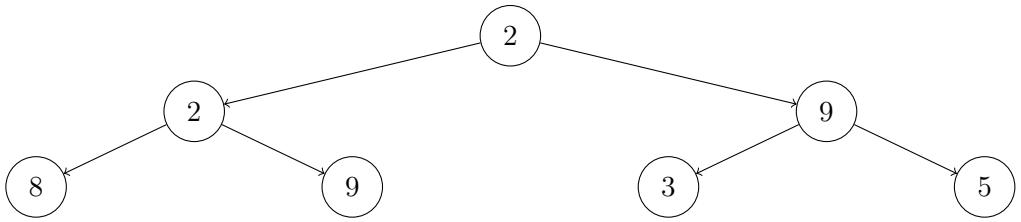
Similarly, $2 < 7$ and 7 is the greatest child of 2 so we will swap element 7 with element 2.



Again, $2 < 6$ and 6 is the greatest child of 2 so we will swap element 6 and element 2. At this point we have reached a leaf of the tree so now the property of the max-heap has been restored. Hence, the final max-heap after removal of the top element and bubbling down will be:

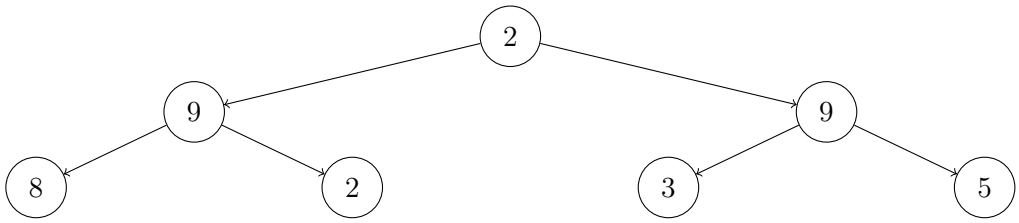


3. Consider the sequence of elements $[2, 2, 9, 8, 9, 3, 5]$. Using the representation discussed in class, show the tree to which this sequence corresponds. Then, show the *heapification* of this tree; that is, show how this tree is transformed into a heap. Demonstrate each bubbling step.
Solution:
As we know the elements are ordered in Level Order Traversal, the corresponding complete tree represented by $[2, 2, 9, 8, 9, 3, 5]$ is:

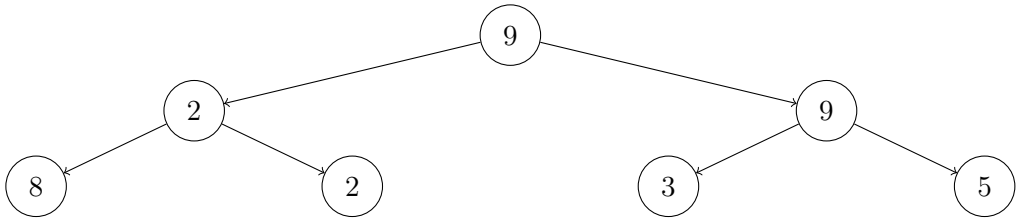


In *heapification* using the heapify algorithm, we will start with the last index to index 0 in the array list and start bubbling down in each index. First we will start with index 6 which is a leaf so we will move to

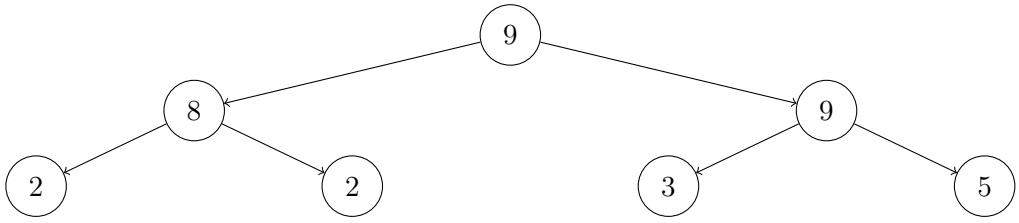
next index, similarly index 5, index 4, index 3 are all leaf so bubble down in these indices, the tree will not be effected.
Now, we are in index 2 where the element in index 2 (i.e. 9) is greater than its left and right child, so the max-heap invariant holds, so the bubble down at this index will not change anything.
After this, we will move to index 1, this index has element 2 which is less than both left & right child so we will swap it with the greatest child(here it's the right child). Now the corresponding sequence will update to $[2, 9, 9, 8, 2, 3, 5]$ and the tree will now be:



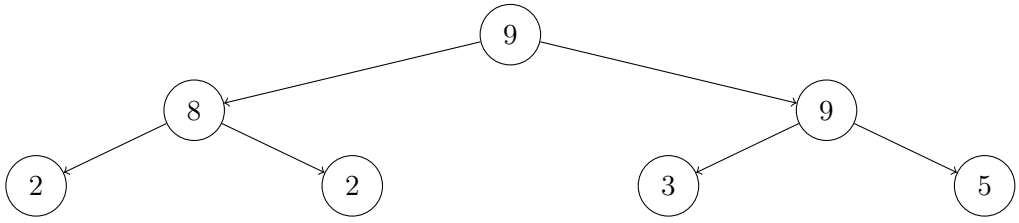
After swapping 2 now becomes the leaf so bubbling down at 2 will not change anything.
Finally we will reach the root of the tree i.e. index 0 which has element 2. Bubbling down at 2, 2 is less than both left and right child so we will swap 2 with the greatest child(here both child are equal so we will choose left child), at this step the corresponding sequence will update to $[9, 2, 9, 8, 2, 3, 5]$ and tree will now be:



Now as 2 still has a left child we will bubble down at index 1 (i.e. element 2). Here 2 is less than the left child so we will swap 2 with its left child (i.e. 8). At this step the corresponding sequence will update to $[9, 8, 9, 2, 2, 3, 5]$ and tree will now be:



At this step we have restored the max-heap invariant in the entire tree and *heapification* is complete.
Hence, finally the max-heap in array list form will be $[9, 8, 9, 2, 2, 3, 5]$ and the max-heap tree will be:



End of Lab 7 Writings