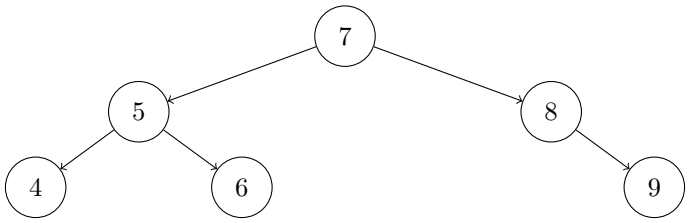
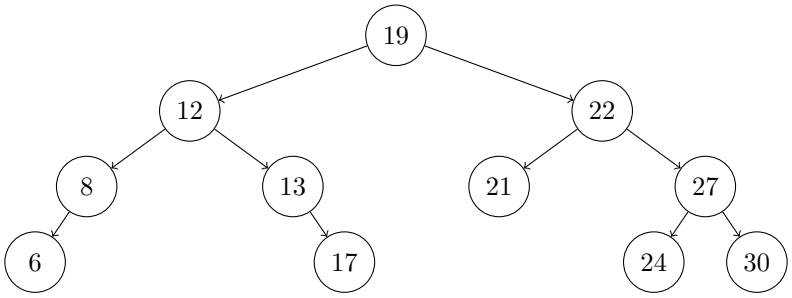
1 AVL Trees

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



x y z (8 and 9) Left rotation is performed on 5 7 replaces 5, and 4 (x) is shifted to the left of 5. 6(y) is now on the right of 5. 8 and 9 (together are z) remain on the right of 7. Now, we have a balanced AVL Tree.

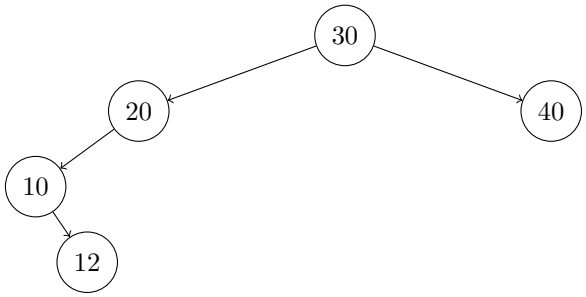
Problem 2. Show the right rotation of the subtree rooted at 27. Be sure to specify the X, Y, and Z subtrees used in the rotation.



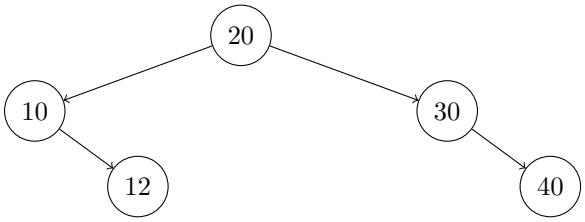
Right rotation at 27. 22 replaces 27, in which 21(x) remains on the left of 22. 24 (y) was on the right of 22, but after the right rotation, it can not remain directly right as 27 is at that position. Therefore, 24 must follow its relationship with 22 (it is greater than 22). However, since 27 is the right child of 22 (because of the right rotation), 24 must readjust to the left of 27. 30 is our z. Now, we have a balanced AVL Tree.

Problem 3. Using the appropriate AVL tree algorithm, insert the value 12 into the following tree. Show the tree before and after rebalancing.

Before



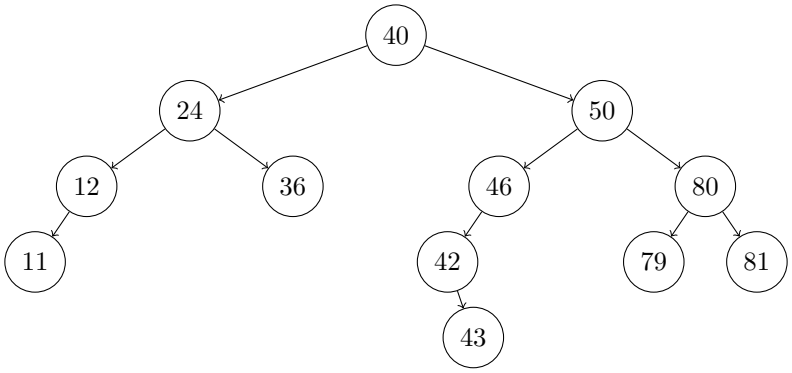
After



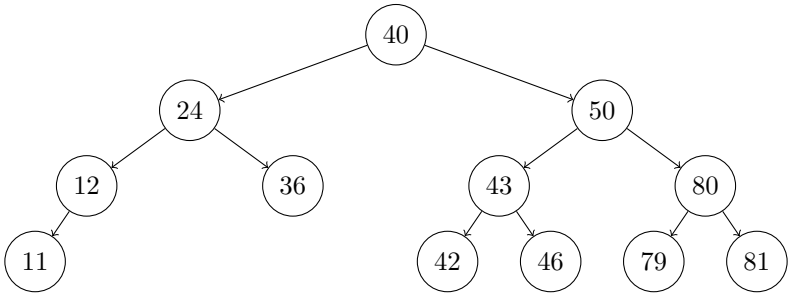
Right rotate 30 12 is first inserted on the right of 10(x), because it is a BST. y is blank. Then, we right rotated on 30, so that 20 is now the root, and 30 and 40(z) are shifted to the right of 20. Now, we have a balanced AVL Tree.

Problem 4. Using the appropriate AVL tree algorithm, remove the value 54 from the following tree. Show the tree before and after rebalancing.

Before



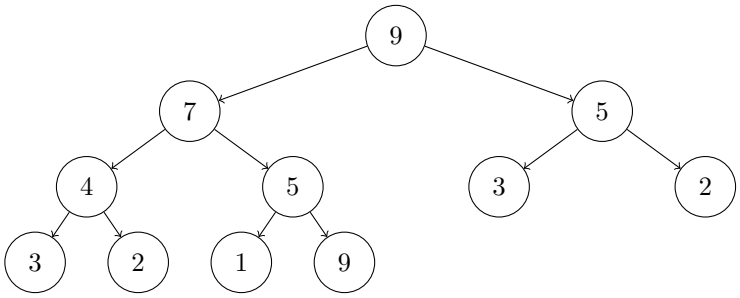
After



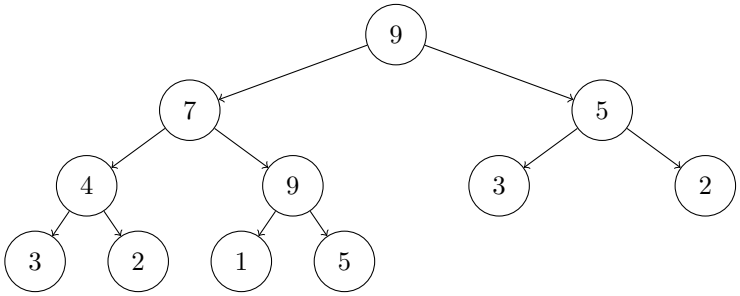
50 bubbles up to replace 54 as the predecessor. However, now 46 has one child, it has a height of 2 (42 and 43), and a height of 0 from the right side (no children). Therefore, we must balance it by performing a left rotation on 42. Then, we perform a right rotation on 43, so that 43 becomes the root of the subtree on the left of 50.

2 Heaps

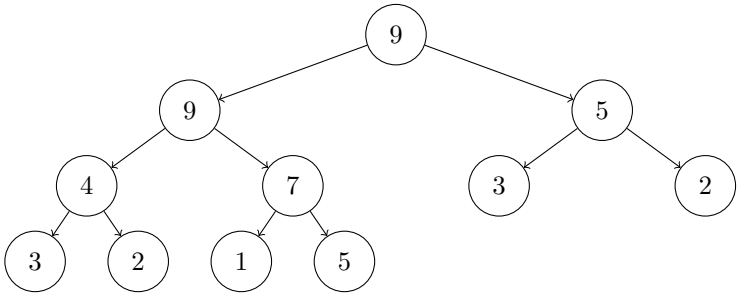
Problem 1. Show the addition of the element 9 to the max-heap below. First, show the addition of 9 to the tree; then, show each bubbling step.



9 is first added to the heap as the right-most and bottom-most child. Then it begins bubbling up.

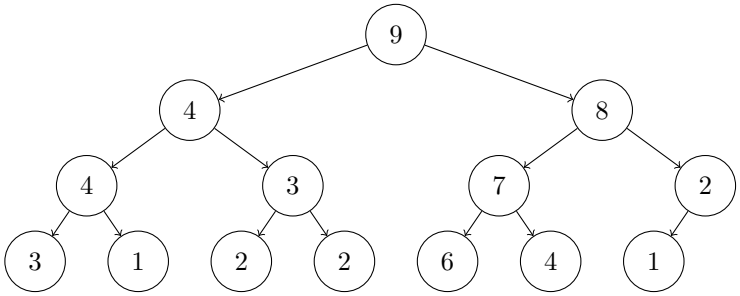


9 bubbles up to replace 5, as it is greater, and then we must check if its parent is greater. This is not true.

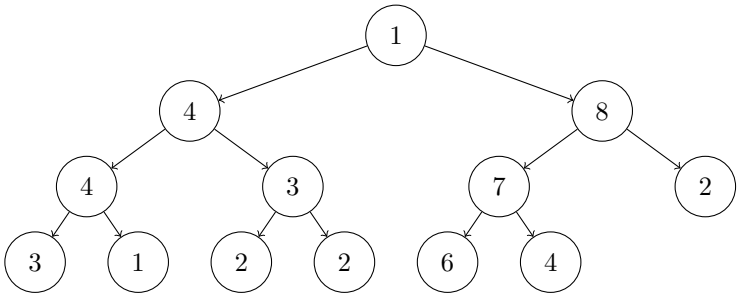


Therefore, 9 must bubble up one more time, replacing 7. Then, 9 checks if its parent is greater than or equal to itself, which is true ($9 = 9$). Now, the heap is complete!

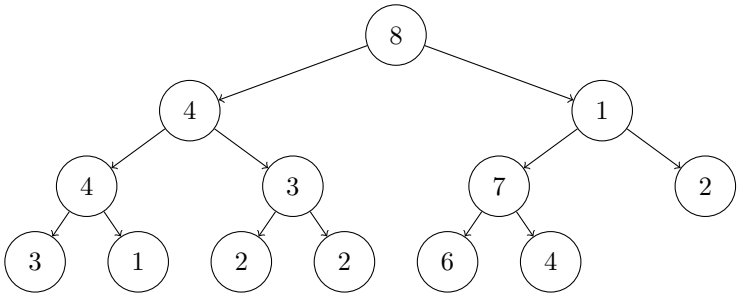
Problem 2. Show the removal of the top element of this max-heap. First, show the swap of the root node; then, show each bubbling step.



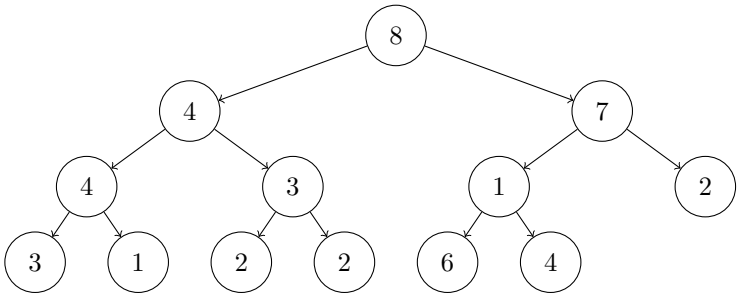
First, remove 9. This is our original heap.



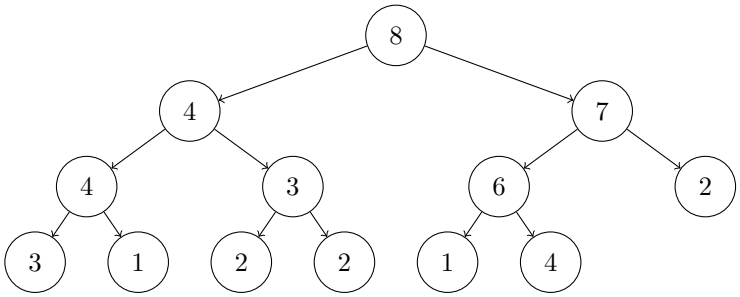
Upon removing 9, we replace it with the right-most, bottom-most value, which is 1. 1 is not greater than its children, therefore we must bubble down.



8 is the largest child of 1, therefore 8 bubbles up and 1 bubbles down. However, 1 still has children greater than it, therefore we must continue bubbling down.

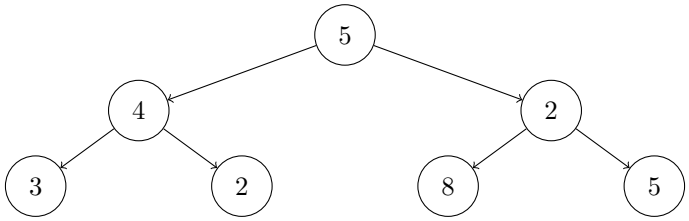


7 is the largest child of 1, therefore 7 bubbles up and 1 bubbles down. But, 1 still has children greater than it, therefore we must continue bubbling down.

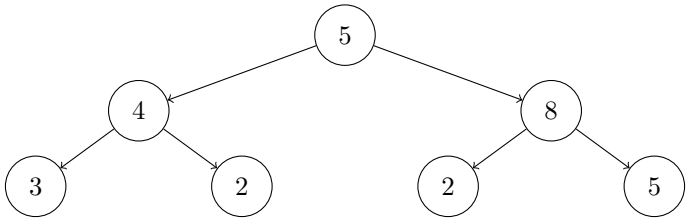


6 is the largest child of 1, therefore 6 bubbles up and 1 bubbles down. Now, we have a heap!

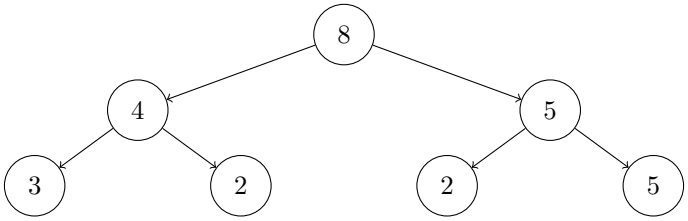
Problem 3. Consider the sequence of elements `[5,4,2,3,2,8,5]`. Using the representation discussed in class, show the tree to which this sequence corresponds. Then, show the *heapification* of this tree; that is, show how this tree is transformed into a heap. Demonstrate each bubbling step.



Starting from the root, we check that the children are less than the parents. 5 is greater than its children. Going from left to right after the root, we check this again. 4 is greater than its children. Now we check the right child of 5, which is 2. 2 is not greater than its children.



Since 2 is not greater than its children, the greatest child of 2, which is 8, bubbles up to replace it. The new subtree has a root(8) greater than its children (2,5). We go back to the root to verify once more after the swap occurs. We check if the root is greater than its children, which is false (5 ¡ 8).



Since 8 is the only greater child of 5, 8 bubbles up to replace 5 as the root. We then go back and check the whole tree once more. The tree is now a heap!