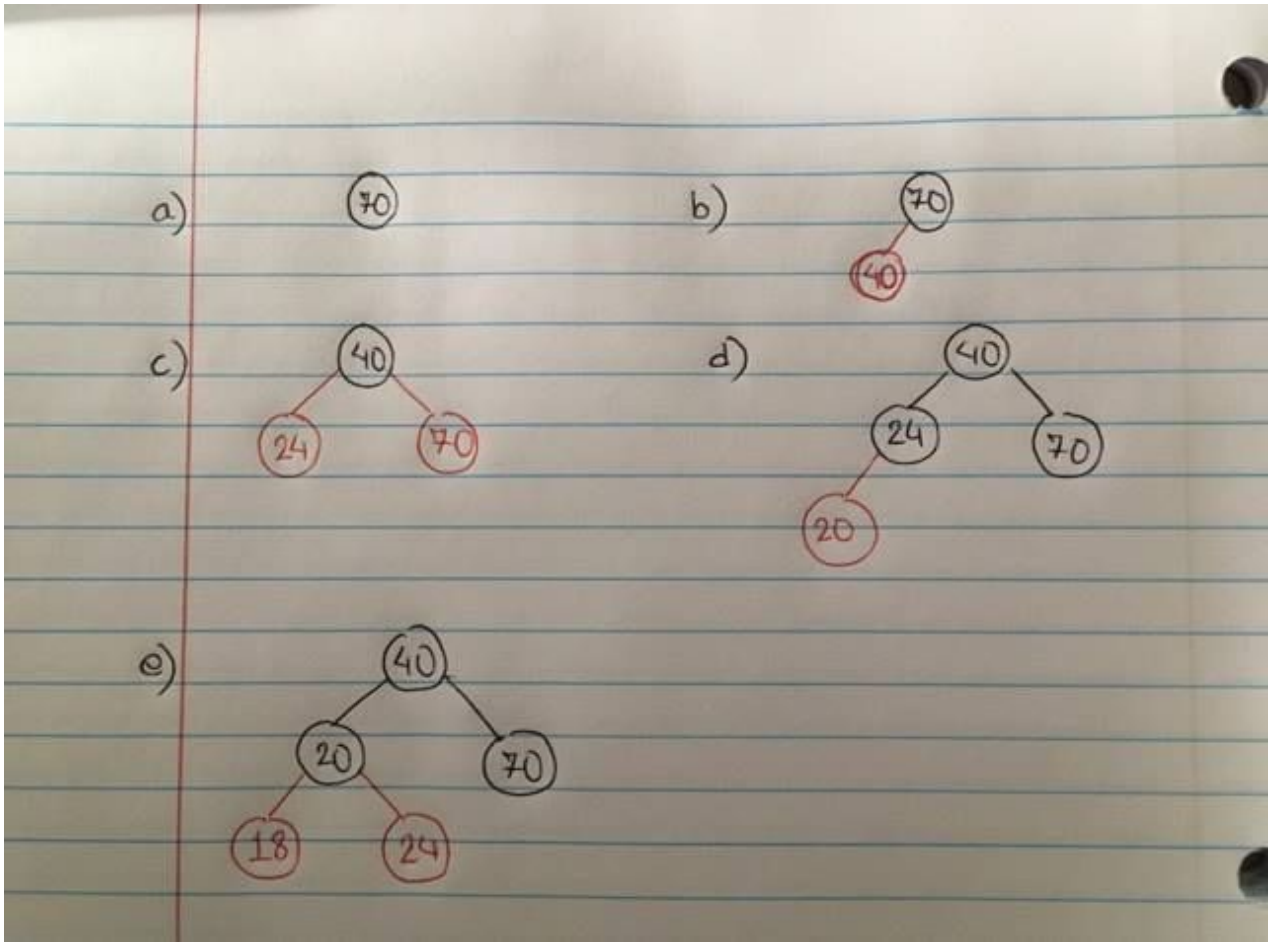
Ashutosh Rai
CS 335: Advanced Data Structure

Lab 4

#1a

Red-Black Tree is a Binary Search Tree where the nodes are colored red or black. Additionally it has other properties such as that the root node is always black, the children of red node are black and all nodes with zero or one children have the same number of black ancestors. Every Red-Black Tree has an equivalent (2,4) Tree and can be visualized as one. The height of a Red-Black Tree is O(log n).

#1b



#1c

Red-Black Trees and (2,4) Trees are secretly the same thing because every Red-Black Tree has an equivalent (2,4) Tree. In a Red-Black Tree, every red node can be merged into its parent, with the entry from the red node being stored in the parent and the children of the red node become the ordered children of the parent.Conversely, every (2,4) tree can be converted to a Red-Black Tree as well.

#2

With the current test case I have, I first generate an array that has 800 integers (0 through 799). Then, to make it more interesting, I shuffle the elements of the array. I implemented a Red-Black Tree and a standard Binary Search Tree as the two types of trees. I timed the time it takes for both of the trees to insert the 800 items, look up 500 items and delete all the 800 items. The results were as expected, with the Red-Black Tree being much faster than the standard Binary Search Tree in terms of insertion and look-up times. However, the Binary Search Tree was much faster than Red-Black Tree in terms of deletion, which I am guessing might be as the BST does not have any rebalancing operations to do after each deletion unlike the Red-Black Tree. I would definitely choose the Red-Black Tree compared to the Binary Search Tree as the speed of the insertion and looking-up were significantly faster. The results do back up the theoretical differences. Because of the rebalancing that occurs in the Red-Black Tree, we would expect the insertion and looking-up to be much faster than the BST that lacks rebalancing techniques.

#3

In general, AVL trees maintains a more rigidly balanced tree. This means that the height of the tree generally will be shorter in an AVL tree than a Red-Black Tree, causing the lookup time to be faster in the AVL tree. However, due to more rotation operations, AVL Trees are slower for insertion and deletion. So when the task is insertion/deletion intense, Red-Black Trees are the better choice as they are faster than AVL trees for insertion and deletion. In comparison to Splay trees, for a random input, Splay Trees are generally slower than Red-Black Trees, making Red-Black Trees a better choice of data structure unless memory is critical, or there are some nodes we look up more often than others.

In terms of code implementation and maintenance, the rotations for AVL Trees are harder to implement and debug compared to the Red-Black Tree. This is also an important thing to consider while choosing data structures. Hence, even though both the AVL Tree and Red-Black Tree are commonly used self-balancing Binary Search Trees, Red-Black Tree might be preferred for the reasons of faster insertion/deletion and code implementation/maintainability.