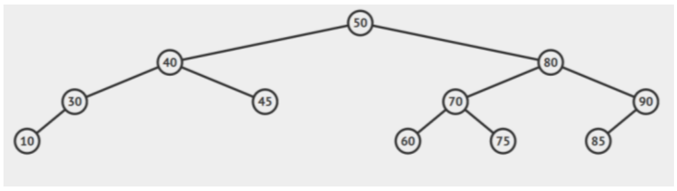
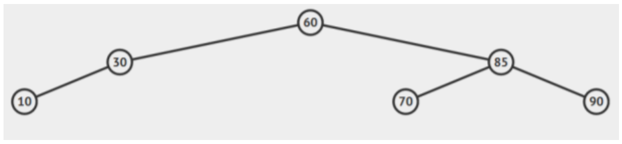
Question – 1:

(a) The preorder, inorder and postorder traversals of the given binary tree are as follows:
- Preorder: 1, 2, 4, 7, 8, 5, 9, 10, 12, 13, 3, 6, 11
- Inorder: 7, 4, 8, 2, 9, 5, 12, 10, 13, 1, 3, 6, 11
- Postorder: 7, 8, 4, 9, 12, 13, 10, 5, 2, 11, 6, 3, 1

(b) After inserting the given values into a Binary Search Tree we have the below BST:



After deleting 45, 40, 80, 75, 50 in respective order the BST looks as follows:



Question-3:

In the addNgram method that I wrote, the function calls a helper function which runs in the following fashion: it recursively traverses the tree to try and find the item with the given ngram string value. If it does not find it where it needs to be found, it creates a new node with the given ngram value and inserts it in the corresponding place. If such a node already exists in the tree, it increments the count value of that node. By this algorithm, since we are traversing all the nodes, if the tree were balanced the time complexity would be Θ(n) however, in the worst-case scenario, the tree would be unbalanced, thus requiring us to do the process in linear time. Thus, the worst-case runtime of the addNgram method I wrote is Θ(n), where n is the number of nodes in the tree.

In the printNgramFrequencies method that I wrote, similar to the addNgram method, I call a helper function which operates as follows: starting from the root of the tree, it recursively visits all nodes in inorder traversal, which requires for the method to visit all nodes of the tree regardless of it being balanced or not. Thus, including the worst-case scenario, in all cases this method will run in Θ(n) time where n is the number of nodes is the tree.