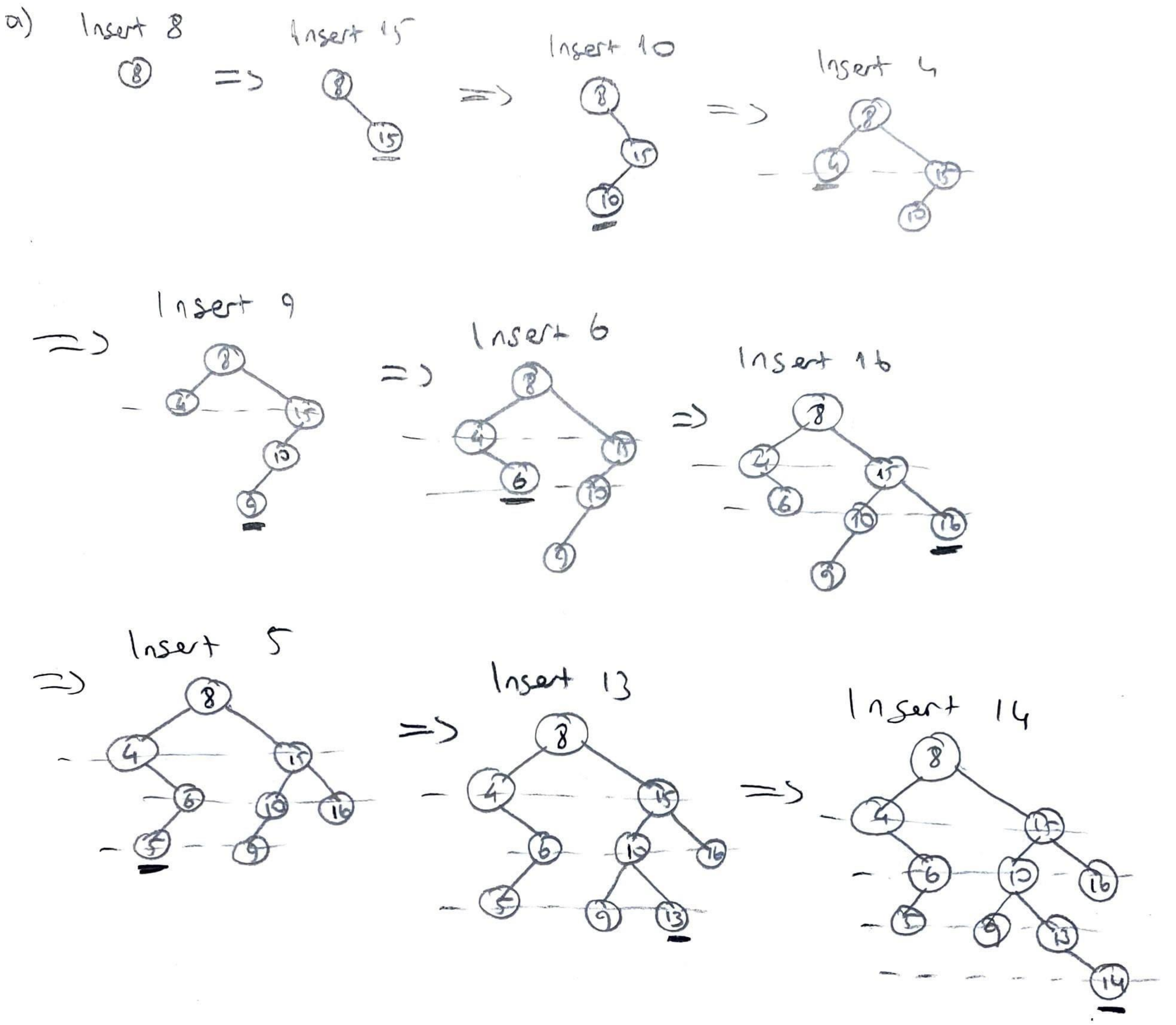
Utku Boran Torun, CS202-1, 21901898

Question 1

Question-1 Utku Boran Torun

a)



b)

Preorder: 8-4-6-5-15-10-9-13-14-16

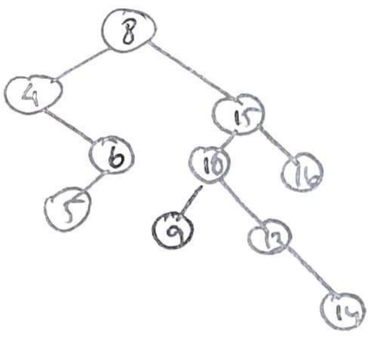
Inorder: 4-5-6-8-9-10-13-14-15-16

Postorder: 5-6-4-9-14-13-10-16-15-8

c)

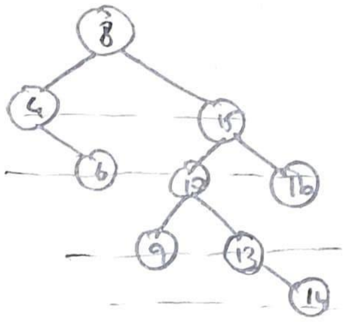
Utku Baran Torun

Initially



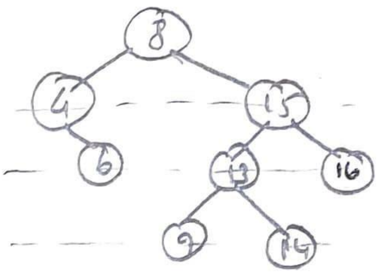
⇒

Delete 5



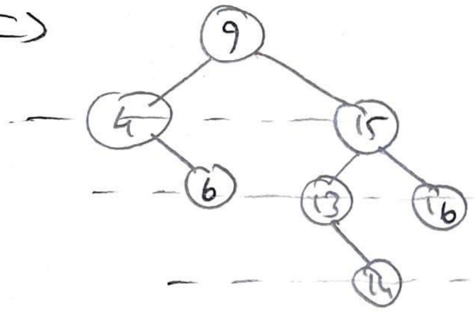
⇒

Delete 10



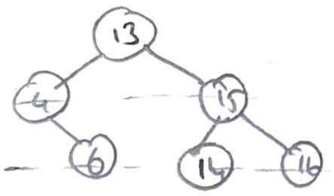
Delete 8

⇒



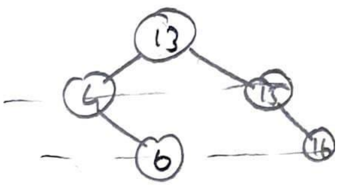
⇒

Delete 9



⇒

Delete 14



Question 3

Insert Item: For insertItem implementation, I used one helper method called inserter in order to have access to root pointer and put parameter of BSTNode*&. This function recursively calls itself and puts the item in proper place by comparing item of every node of the tree. If it is less than that node, it goes to left and if it is greater than that node, it goes to right.

Average case for inserting item to the balanced or complete binary search tree. To compute the time complexity of average case, first we should write the reccurence relation.

T(N) = T(N/2) + O(1)

T(1) = O(1) -> Base case

If we solve the reccurence relation we will see that T(N) = O(logN).

Worst case is inserting to end of the binary search tree. To compute the time complexity for worst case, first we should write the reccurence relation.

T(N) = T(N) + 2 x O(1)

T(1) = O(1) -> Base case

If we solve the reccurence relation we will see that T(N) = O(N)

Delete Item: For deleteItem implementation, I used three different helper methods. One is to find which node should be deleted by giving the key. The other one is to delete the node that we found in previous function and delete it. However, if we cannot find directly the node we want we use processLeftMost in order to reach left most of right subtree. In addition, we do these functions with all recursion.

Average case for delete item is deleting item from the balanced or complete binar search tree. To compute the time complexity of average case, first we should write the reccurence relation.

T(N) = T(N/2) + O(1)

T(1) = O(1) -> Base case

If we solve the reccurence relation we will see that T(N) = O(logN)

Worst case for delete item is deleting a first node of a binary search tree which is at its maximum height. To compute the time complexity for worst case, first we should write the reccurence relation.

T(N) = T(N-1) + 2O(1)

T(1) = O(1) -> Base case

If we solve the reccurence relation we will see that T(N) = O(N)

Retrieve Item: For retrieveItem function I used one additional helper function in order to reach the node parameter and called this function in retrieveItem function with root. This function helps with retrieving item without going unneccessary nodes.

Average case for retrieve item is retrieving item from balanced or complete binary search tree. To compute the time complexity of average case, first we should write the reccurence relation.

T(N) = T(N/2) + 3O(1)

T(1) = O(1)

If we solve the reccurence relation we will see that T(N) = O(logN)

Worst case for retrieve item is retrieving last node of a binary search tree which is at its maximum height. To compute the time complexity for worst case, first we should write the reccurence relation.

T(N) = T(N-1) + 2O(1)

T(1) = O(1) -> Base case

If we solve the reccurence relation we will see that T(N) = O(N)

Inorder Traversal: For inorderTraversal implementation one additional helper function is used. In order to, have access to array that we should return and have access to how many nodes we have in our tree. This function added these nodes' items by going left subtree first, then root and finally right subtree.

Average case's time complexity and worst case's time complexity for inorder traversal is same. They are all order of N^2.

T(N) = T(N – 1) + O(N) + O(1)

If we solve the relation we will see that T(N) = O(N^2)

Has Sequence: For hasSequence function I used just one additional helper function which is retrieve item to reach whether item is in tree or not. This function takes an array and its size as input and checks whether that sequence is in our tree or not. Hence, its average and worst cases' time complexity is same with retrieve item.

Count Nodes Below Level: For countNodesBelowlevel function I used two additional helper function. One is to access the root and other one is to compute how many nodes we have under that root. Then, I called both of them in countNodesBelow function.

Average case's time complexity and worst case's time complexity are sam efor countNodesBelow and they are equal to order of N.

T(N) = T(N – 1) + T(N – 1) + O(1)

If we solve the relation we will see that T(N) = O(N)

Merge: For merge function I used just one additional helper function to have access to number of nodes via using pass by reference. That one additional function uses countNumberOfNodes and inorderTraversal. countNumberOfNodes' time complexity is O(N) and inorderTraversal's time complexity is O(N^2). Hence, inorderTraversal dominates countNumberOfNodes.

Average case and worst case is same for merge function and it is O(N^2), because of inorderTraversal function.