DSA A2 Q4 Explanation
Arush Nandankar 2022101116

First we scan “n” which denotes the number of nodes and then declare an array lvlOrder and scan the elements in them.

The we create the Root Node (T) and enqueue it in a Queue Q and the pass it the function superInsert which has arguments T, lvlOrder array, n (number of elements), Q.

Then we start a while loop till we reached the end of the lvlOrder array (ie till all elements are inserted). Then we start inserting each element in the following manner :-

1. If the element can be inserted left of the node in the front of the Q we insert it to the left of it and then enqueue the node in the queue Q. Then we increment the index for the lvlOrder array.
2. If the element can be inserted right of the node in the front of the Q we insert it to the right of it and then enqueue the node in the queue Q and dequeue the Q because the node in the front can no longer have any nodes inserted left/right of it because top-down order has been provided as input. Then we increment the index for the lvlOrder array.
3. If the element cannot be inserted to left/right of the node in the front of the queue we dequeue the node from Q.

The time complexity of the function superInsert is O(n) because each node will be enqueued once in the Q and also dequeued after either It has had a no children or it has had a right child or if it has had both right and left children.

To figure that if a node can go under a node I have first modified the struct node to include a greaterThan and lessThan value which denotes the range of keys which can go under it. When a node has be added to the left of the parent node the greaterThan value is retained from the parent and the lessThan value becomes key of the parent. Similarly when the node has be inserted to the right.

The time complexity of the function canItgoUnderIt is O(1).

Then the function superInorder is called which uses ignorer travels to travel the BST in ascending order and instead of printing the node updates it’s value to value+carrySum and the updates the carrySum to to the new value of the node. carrySum is a long long int declared in main and passed by reference to the superInorder function.
The time complexity of the function superInorder is O(n) as every node is visited once.

The the function LvlOrderWithQueue is called which performs level order traversals using queues as discussed in lectures and tutorials in addition to it, it also adds the value of the node to the totalSum variable which is passed by reference to the function to calculate sum of all node in the BST.
The time complexity of the function LvlOrderWithQueue is O(n) as every node is visited once.