Insertion algorithm:

- TreeInsert Description: The insertion algorithm first calls LocateParent function that goes through the BST and returns the parent of the node to insert. It then sets the nodeToInsert's parent to point to that parent node returned. If that node is NULL meaning that parent doesn't exist, it means the tree is empty so the nodeToInsert is set to be the root. Otherwise, it compares the x values of the nodeToInsert and its parent node to figure out which side to insert the nodeToInsert. If nodeToInsert's x value is smaller than the parent node's x value, it gets inserted on the left side otherwise it gets inserted to the right.

- TreeInsert Pseudo-Code:

function TreeInsert(T, n)

p = LocateParent(T, n);

n->parent= y;

if (!y) then T->root = z;

else if (n->x < p->x) then p->left = n;

else p->right = n;

end function

- LocateParent Description: The LocateParent first takes the root of the tree and uses a while loop to keep going down a tree. It'll either go left or right depending on the values of the NodeToInsert and the currentNode it's pointing at. It keeps going until the currentNode is NULL and that returns the parent of that currentNode. This is the parent of the NodeToInsert. If it never went into the while loop, and since the parentNode was set to NULL, it'll return NULL and that means the tree doesn't exist.

- LocateParent Pseudo-Code:

function LocateParent(T, n)

p = NIL;

x = T->root;

while (x) do

p = x;

if (n->x < x->x) then x = x->left;

otherwise x = x->right;

end while

return p;

end function

Range sum algorithm:

- sumYinRangeX Description: The algorithm first uses a loop to make sure the current node's x value is within the range. So, if the x value is lower than the min value, it goes right and if the x value is larger than the max, it goes left. It keeps doing this until either the node is null, or its x value is within the range. Next, if the node isn't null, that means it has a x value within the range, so we do a leftSum that sums all the y values of the nodes' x values greater than the min value in the left subtree of the current node and rightSum that sums all the y values of the nodes' x values smaller than the max value in the right subtree. We add leftSum and rightSum and add the current node's y value since that's in the range too.

- sumYinRangeX Pseudo-Code:

function sumYinRangeX(root, min, max)

traversalNode = root;

while (traversalNode) and (traversalNode->x not in [min. max]) do

if (traversalNode->x <= min) then traversalNode = traversalNode->right;

else if (traversalNode->x >= max) then traversalNode = traversalNode->left;

end while

if (traversalNode) then

leftSum = TreeSumGe(traversalNode->left, min);

rightSum = TreeSumLe(traversalNode->right, max);

return traversalNode->y + leftSum + rightSum;

else return 0

end function

- TreeSumGe Description: The algorithm starts from the root and that node is called the traversalNode. If the traversalNode not null, it checks if its x value is greater than the key. If it is, it adds the y value of the traversalNode to the sum and if traversalNode has a right subtree, then it adds the sum of that right sub tree to the sum. It then goes left to check if there's any node in the left subtree that has a x value greater than the key. If it doesn't, it goes into the right subtree. It keeps repeating this until the traversalNode is null and finally it returns the total sum.

- TreeSumGe Pseudo-Code:

function TreeSumGe(root, x)

sum = 0;

traversalNode = root;

while (traversalNode)

if (traversalNode->x >= x) then

sum += traversalNode->y;

if (traversalNode->right) then sum += traversalNode->right->sum;

traversalNode = traversalNode->left;

else

traversalNode = traversalNode->right;

end if

end

return sum;

end function

- TreeSumLe Description: The algorithm starts from the root and that node is called the traversalNode. If the traversalNode not null, it checks if its x value is lesser than the key. If it is, it adds the y value of the traversalNode to the sum and if traversalNode has a left subtree, then it adds the sum of that left sub tree to the sum. It then goes right to check if there's any node in the right subtree that has a x value smaller than the key. If it doesn't, it goes into the left subtree. It keeps repeating this until the traversalNode is null and finally it returns the total sum.

- TreeSumLe Pseudo-Code:

function TreeSumLe(root, x)

sum = 0;

traversalNode = root;

while (traversalNode)

if (traversalNode->x <= x) then

sum += traversalNode->y;

if (traversalNode->left) then sum += traversalNode->left->sum;

traversalNode = traversalNode->right;

else

traversalNode = traversalNode->left;

end if

end

return sum;

end function