BST AVL Struct

```
template <class T>
class TreeNode {
private:
    T _item;
    TreeNode<T>* _left;
    TreeNode<T>* _right;
    //Could include
    TreeNode<T>* _parent;
    int _height;
public:
    TreeNode(T x) {
     _left = _right = NULL;
     _item = x;
     _height = 0; };
     friend BinarySearchTree<T>;
};
```

_item = value
_height = height of the node (will have to calculate)
_left , _right = default set to NULL in public constructor.

Search Max,Min

```
template <class T>
T BinarySearchTree<T>::search-
Max() {
     TreeNode<T>* current =
_root;
     while (current->_right) {
           current = current->-
_right;
      }
      return current->_item;
}
template <class T>
T BinarySearchTree<T>::search-
Min() {
      TreeNode<T>* current =
_root;
```

Search Max,Min (cont)

```
      while (current->_left) {
      current = current->_left;
      }
      return current->_item;
}
```

Successor

```
template <class T>
T BinarySearchTree<T>::succes-
sor(T x){
     TreeNode<T>* root = _root;
     T succ = NULL;
     while (root != NULL) {
       if (x < root->_item){
           succ = root->_item;
           root = root->_left;
        }
        else{
           root = root->_right;
        }
    }
    return succ;
}
```

exist() // iterative

```
//Iterative
template <class T>
bool BinarySearchTree<T>::-
exist(T x) {
        TreeNode<T>* nd = _root;
        while (nd != NULL) {
                if (nd->_item ==
x) return true;
                if (nd->_item >
x) nd = nd->_left;
                else nd = nd-
>_right;
```

exist() // iterative (cont)

```
        }
        return false;
}
```

exit() // Recursive

```
template <class T>
bool BinarySearchTree<T>::exi-
st(TreeNode<T> Node,T x) {
         // RECURSIVE
         exists ( _root , value
x )
         if (_root == null)
return false;
         if (_root->_item == x)
return true;
         if (_root->_item > x)
            return exists (
_root->left, x);
         else exists ( _root--
>right, x );
}
```

LR RL Rotation

```
//LR Rotation
template <class T>
TreeNode<T> BinarySearchTree<T-
>::_leftrightRotation(TreeNode-
<T> node)
{
node->_left = _rightRotation(-
node->_left);
node = _leftRotation(node);
return node;
}
//RL Rotation
template <class T>
TreeNode<T> BinarySearchTree<T-
>::_rightleftRotation(TreeNode-
<T> node) {
node->_right = _leftRotation(n-
ode->_right);
```

LR RL Rotation (cont)

```
node = _rightRotation(node);
return node;
}
```

PreOrderPrint

```
template <class T>
void BinarySearchTree<T>::_pr-
eOrderPrint(TreeNode<T>* node) {
      if (!node) return;
      cout << node->_item << "
";
      _preOrderPrint(node->-
_left);
      _preOrderPrint(node->_ri-
ght);
}
```

InOrderPrint

```
template <class T>
void BinarySearchTree<T> ::
_inOrderPrint(TreeNode<T>* node)
{
      if (!node) return;
      _inOrderPrint(node->_-
left);
      cout << node->_item << "
";
      _inOrderPrint(node->_-
right);
}
```

PostOrderPrint

```
template <class T>
void BinarySearchTree<T>::
_postOrderPrint(TreeNode<T>*
node) {
     if (!node) return;
     _postOrderPrint(node->_l-
eft);
     _postOrderPrint(node->_r-
ight);
     cout << node->_item << " ";
}
```

Height

```
template <class T>
int BinarySearchTree<T>::height-
(TreeNode<T>* node) {
return node == NULL ? -1 :
(node->_height);
}
template <class T>
int BinarySearchTree<T>::_calhe-
ight(TreeNode<T>* node) {
   int leftHeight = -1;
   int rightHeight = -1;
   if (node != NULL) {
     if (node->_left != NULL) {
        leftHeight = _calheigh-
t(node->_left);
     }
     if (node->_right != NULL) {
      rightHeight = _calheigh-
t(node->_right);
     }
   }
   return max(leftHeight,
rightHeight) + 1;
}
//Height from TA
template <class T>
void TreeNode<T>::rectifyH-
eight()
{
 int left = _left ? _left->_h-
eight : -1;
 int right = _right ? _right->_-
height : -1;
//Height Value
 _height = (left > right ? left
: right) + 1;
}
```

Insert

```
//Task 1 and 6
template <class T>
TreeNode<T> BinarySearchTree<T-
>::_insert(TreeNode<T> current,
T x) {
    if (current->_item > x) {
      if (current->_left)
        current->_left = _inser-
t(current->_left, x);
      else {
        current->_left = new
TreeNode<T>(x);
        _size++;
       }
     }
    else if (x > current->_item)
{
      if (current->_right)
         current->_right =
_insert(current->_right, x);
      else{
         current->_right = new
TreeNode<T>(x);
         _size++;
       }
     }
     else
//item already exists, dont need
do anything
      return current;
```

Code From TA// Rotation

```
//-1 if no children, st bf will
work
int left = current->_left ?
current->_left->_height : -1;
int right = current->_right ?
current->_right->_height : -1;
current->_height = (left > right
? left : right) + 1;
```

Code From TA// Rotation (cont)

```
  //GET DIFF IN height
  if (abs(left - right) > 1){

if (left > right){
       int LLH = current->_left->_left ? current->_left->_left->_height : 0;

       int LRH = current->_left->_right ? current->_left->_right->_height : 0;
    if (LLH < LRH){
    current->_left = _leftRotation(current->_left);
    }
    current = _rightRotation(current);
}
else {
    int RLH = current->_right->_left ? current->_right->_left->_height : 0;
    int RRH = current->_right->_right ? current->_right->_right->_height : 0;
    if (RRH < RLH){
    current->_right = _rightRotation(current->_right);
    }
    current = _leftRotation(current);
 }
}
//go through tree and rect height
   current->rectifyHeight();
   return current;
}
```

LL RR Rotation

```
//LL Rotation
template <class T>
TreeNode<T> BinarySearchTree<T>::_leftRotation(TreeNode<T> node)
{
TreeNode<T>* nd;
nd = node->_left;
node->_left = nd->_right;
nd->_right = node;
nd->_height = calheight(nd);
node->_height = calheight(node);
return nd;

/*
*
*Codes from tutorialshorizon
*copyrights to - https://algorithms.tutorialhorizon.com/avl-tree-insertion/
*
//*/
//TreeNode<T>* nd = node->_right;
//TreeNode<T>* T2 = nd->_left;
//nd->_left = node;
//node->_right = T2;
//nd->_height = calheight(nd);
//node->_height = calheight(node);
//return nd;
}
//RR Rotation
template <class T>
TreeNode<T> BinarySearchTree<T>::_rightRotation(TreeNode<T> node)
```

LL RR Rotation (cont)

```
{
TreeNode<T>* nd;
nd = node->_right;
node->_right = nd->_left;
nd->_left = node;
nd->_height = calheight(nd);
node->_height = calheight(node);
return nd;
/*
*
*Codes from tutorialshorizon
*copyrights to - https://algorithms.tutorialhorizon.com/avl-tree-insertion/
*
*
//
TreeNode<T>* nd = node->_left;
TreeNode<T>* T2 = nd->_right;
nd->_right = node;
node->_left = T2;
nd->_height = calheight(nd);
node->_height = calheight(node);
return nd;*/
}
```