Insertion

Best Case: $\Omega(1)$ when the tree is empty

Average Case: $T(n) \in \Omega(1) \land T(n) \in O(\log_2 n)) \rightarrow \boldsymbol{\Theta}(\log_2 n)$

Worst Case: $O(\log_2 n)$ since the tree maintains a logarithmic height

```
private AVLNode<K,V> put(K key, V value, AVLNode<K,V> root)

{
    if (root == null) {return new AVLNode<K,V>(key, value);}
    //Key is greater than the current node's, need to insert in the right subtree
    if (key.compareTo(root.getKey()) >= 0) {root.setRight(put(key, value, root.getRight()));}
    //Key is less than the current node's, need to insert in the left subtree
    else if (key.compareTo(root.getKey()) < 0) {root.setLeft(put(key, value, root.getLeft()));}
    //Adjust ancestor height after insertion and rebalance if needed
   return enforceAVL(root); O(1) * logn = O(logn) operation
}
```

Rotations

Best Case: $\Omega(\mathbf{1})$

Average Case: T(n) ∈ Ω(1) ∧ T(n) ∈ O(1)) → **Θ**(1)

Worst Case: $O(1)$

```
public AVLNode<K,V> leftRotate(AVLNode<K,V> x)
{
    AVLNode<K,V> y = x.getRight(); O(1) Operation
    AVLNode<K,V> T2 = y.getLeft(); O(1) Operation

    y.setLeft(x); O(1) Operation
    x.setRight(T2); O(1) Operation

    //Recompute the heights of the Subtrees
    x.setHeight(computeHeight(x)); O(1) Operation
    y.setHeight(computeHeight(y)); O(1) Operation

    return y; O(1) Operation
}


public AVLNode<K,V> rightRotate(AVLNode<K,V> y)
{
    AVLNode<K,V> x = y.getLeft(); O(1) Operation
    AVLNode<K,V> T2 = x.getRight(); O(1) Operation

    x.setRight(y); O(1) Operation
    y.setLeft(T2); O(1) Operation

    //Recompute the heights of the Subtrees
    y.setHeight(computeHeight(y)); O(1) Operation
    x.setHeight(computeHeight(x)); O(1) Operation

    return x; O(1) Operation
}
```

Remove Minimum

Best Case: $\Omega(\mathbf{1})$ when there is no left subtree.

Average Case: Since $\Omega(1) \supset \Omega(\log_2 n)$, $(T(n) \in \Omega(\log_2 n) \wedge T(n) \in O(\log_2 n)) \rightarrow \mathbf{\Theta}(\log_2 n)$

Worst Case: $O(\log_2 n)$ since the tree maintains a logarithmic height

```
private AVLNode<K,V> removeMinimum(AVLNode<K,V> root)
{
    //There is a leaf node yet to be deleted
    if (root.getLeft() != null)
    {
        root.setLeft(removeMinimum(root.getLeft()));
    }
    //The leaf node is the current node, return the right child
    else
    {
        //Stores the value associated with the minimum key in the subtree
        min = root.getValue();
        return root.getRight();
    }
    return enforceAVL(root); O(1) * logn = O(logn) operation
}
```

Rebalancing a Subtree

Best Case: $\Omega(\mathbf{1})$

Average Case: Average Case: $T(n) \in \Omega(1) \wedge T(n) \in O(1)) \rightarrow \mathbf{\Theta}(1)$

Worst Case: $O(1)$

```
private AVLNode<K,V> enforceAVL(AVLNode<K,V> root)
   {

       root.setHeight(computeHeight(root)); O(1) Operation
       int balanceFactor = balanceFactor(root); O(1) Operation

       //The tree is left leaning
       if (balanceFactor > 1)
       {
           //The left subtree is left leaning, perform a right rotation to rebalance.
           if (balanceFactor(root.getLeft()) >= 0)
           {
               return rightRotate(root); O(1) Operation
           }
           //The left subtree is right leaning, perform a left-right rotation to rebalance.
           else
           {
               root.setLeft(leftRotate(root.getLeft())); O(1) Operation
               return rightRotate(root); O(1) Operation
           }
       }
       //The tree is right leaning
       else if (balanceFactor < -1)
       {
           //The right subtree is right leaning, perform a left rotation to rebalance.
           if (balanceFactor(root.getRight()) <= 0)
           {
               return leftRotate(root); O(1) Operation
           }
           //The left subtree is left leaning, perform a right-left rotation to rebalance.
           else
           {
               root.setRight(rightRotate(root.getRight())); O(1) Operation
               return leftRotate(root); O(1) Operation
           }
       }
       return root; O(1) Operation
   }
```