1)

for an AVL tree satisfies the median property and it has odd number of nodes, then l(v) will be the median of the set. And the number of nodes in the left subtree must be equal to the right subtree. And we notice that the two subtrees rooted at v also satisfies the median property, so the symmetric structure is about the root v.Then the AVL property of root node v is satisfied. Induction shows that all AVL trees which satisfies the median property with odd number of nodes are perfectly balanced. For an AVL tree satisfies the median property and it has odd number of nodes, then the l(v) must be the high median of the set.

2)

For insert(x) operation, we developed a recursive algorithm.

• If the root of the AVL tree is NULL, we just create a new node which contains the value x and set the root pointer to this node.

• If the root is not NULL, then we perform an AVL insert operation. A new node containing x is created and is set as child of that leaf node according to the comparison result between x and the value of that leaf node.

• Then we need to adjust the tree to satisfy both the AVL property and the median property of the new tree.

– If the tree rooted at v has odd number of nodes before insertion:

∗ If the value of the new node is greater than the root value, we need to find the smallest element that is greater than v.value. Take the node out of the tree and place it to the root v. Then perform insert(v.value) to the subtree rooted at root.left.

∗ Else, nothing to do with the current tree rooted at v.

– If the tree rooted at v has even number of nodes before insertion:

∗ If the value of the new node is greater than the root value, nothing to do with the current tree rooted at v.

∗ Else, we need to find the largest element that is smaller than v.value. Take the node out of the tree and place it to the root v. Then perform insert(v.value) to the subtree rooted at root.right.

3)

Perform the AVL remove operation, the rest operations are the same as the insert operation. And the entire implementation is shown in the code.

4)

We consider the worst cases, every node requires 2log2 height (height is the height of the subtree rooted on the node). Assume n is the 2 to the power of m, h(2n)=h(n)+2n. so h( 2 to the power of m+1)=h( 2 to the power of m)+ 2 to the power of m+1. By using the characterist polynomial equation we can get h(n) ∈O（n|n= 2 to the power of m）

So h(n) ∈ O(n)