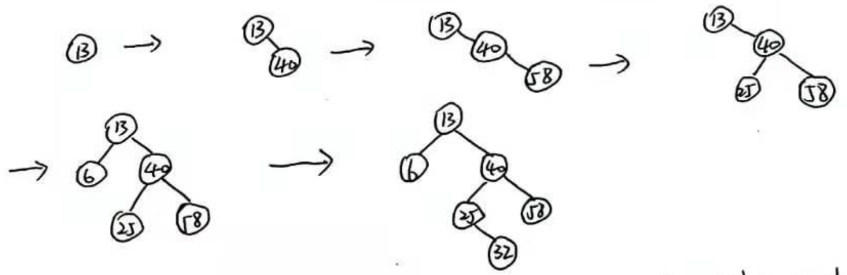
1.1

1. Binary Tree

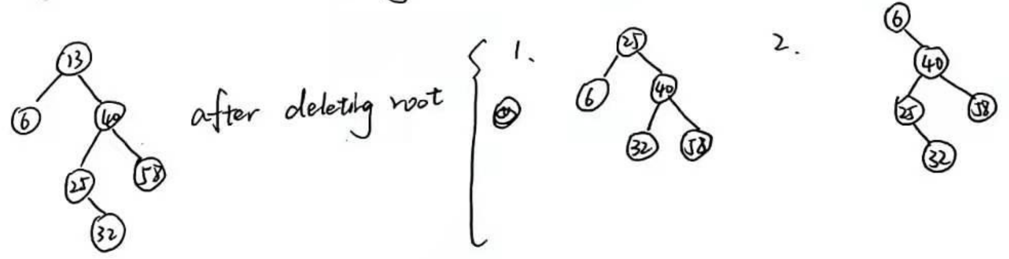
1.1 insert the {13, 40, 58, 25, 6, 32} into empty tree.



1.2 Deleting root, { ① find the minimum in right subtree
② find the maximum in left subtree.

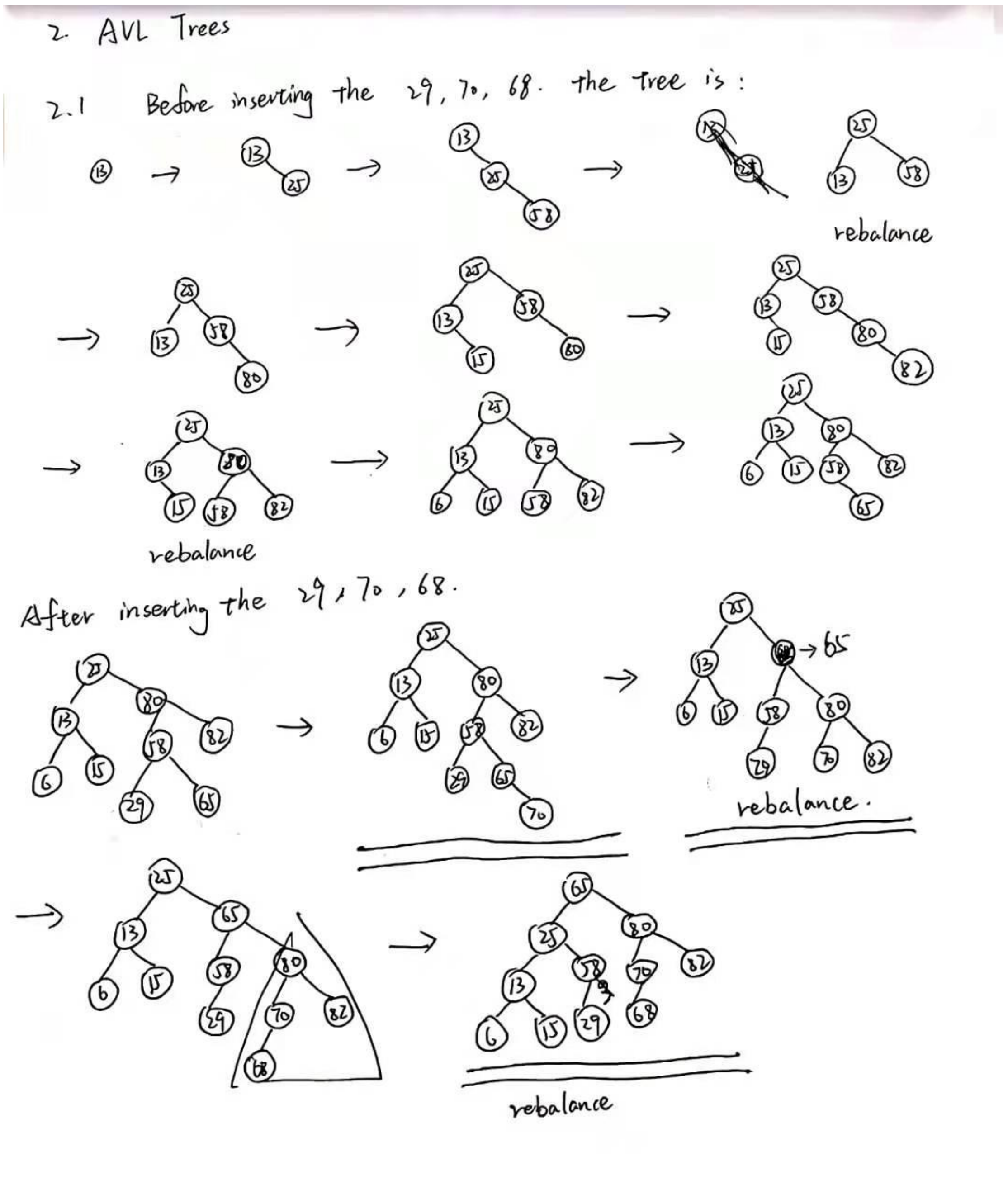
⇓

change the root by the one



1.3 The post-order Traversal should visit the left subtree firstly then the right, last ~~is~~ the root.

So the result is: 6 32 25 58 40 13 .



2. AVL Trees

2.1 Before inserting the 29, 70, 68. the tree is :

After inserting the 29, 70, 68.

2.2
The answer in the zip, the AVLSort.cpp

2.3
In the "insertNode" function, we determine whether the insert position is right or left by the node's value. After inserting some nodes, then we want to insert a new node, we go into the recursion and the time of the recursion is the depth (height) of the tree. And by the "rebalance" function, we can make sure that the height difference between the two subtrees is no more than one. So, the insert time is O(log(n)). And the "destroyTree" recursion also

free the node from the root, the times of from the top down is the height of the tree. When we assign the value in the array from the AVL tree. We use the “isOrder” function, It also acts like an insert function, recursing left and right from minimum to maximum. To conclude, the time complexity of your algorithm is O(c*nlog(n)) = O(nlog(n)), c is a constant.

3.

3.1

Because the size of each of the records is 128 bytes and the one block is 2048 bytes, the structure has 2048/128 = 16 lefts. So, the L = 16. And we know each children has the x branch if the x+1, and the 4(x+1)+8x = 2048, we can calculate the result is 170.3333, we round down the result and we can get the M = x+1 = 171