Q4.(i)

We will use prove by contradiction to show that if an AVL tree satisfies the median property, then it's perfectly balanced. To do this, we will show that the $h-1$ layer is fully filled.
Consider an AVL tree with depth h, since the tree satisfies median property, then for any parent node, its left subtree will have the same or one more node than that in tht right subtree.
There must be at least a leave node at depth h, taken its grandparent node at depth $h-2$ as node A. As defined for AVL tree, node A must have two children. Choose randomly another node at depth $h-2$ which has no or one children as node B, we will have two conditions:

- If the node B have only one children, taken the ansector node of node A and node B as node C, then it's clear that the difference of nodes between the two subtrees of node C is no less than 2, violating the median property.
- If the node B have no children, then the difference of nodes mentioned before will at leat be 3.

So we have proved that the $h-1$ layer of an AVL tree with median property is fully filled. Thus show that the tree is perfectly balanced.

Q4.(ii)

We define the modified *insert* operation as below:
Instead of "balance" stored in one node, we use "number_balance" to preserve the AVL tree. If the nodes in the left subtree is k more than that in the right subtree, the "number_balance" is set to k. So from this defination, the only vaild number for "number_balance" are 1 or 0.
Then we apply the previous *insert* operation as usual. If the insertion happens to be in the left subtree, we increase the "number_balance" by 1, otherwise we decrease the "number_balance" by 1. After inserting, we check "number_balance" for all the nodes.
So, there are two exceptional cases:

- If the "number_balance" is 2 after increasing, we need to perform *reorganize* operation. First, find the presuccessor of the current root node. Then, we replace the current root node with its presuccessor (Apply new *delete* operation on the presuccessor) and insert this root node to the right subtree. After this step, we can decrease the "number_balance" by 2 at the new root node and the "number_balance" becomes 0, satisfying the median property. Then, we move on checking.
- If the "number_balance" is -1 after decreasing, the procedure is similar to the first case. We just find the successor instead of presuccessor and adjust the right subtree instead of left subtree.

Q4.(iii)

We define the modified *delete* operation as below:
We apply the previous *delete* operation as usual. If the deletion happens to be in the left subtree, we decrease the "number_balance" by 1, otherwise we increase the "number_balance" by 1. After we delete and merge its subtrees, we check the "number_balance" of all nodes.
So, there are two exceptional cases as well:

- If the "number_balance" is 2 after increasing, we need to we need to perform *reorganize* operation. First, find the presuccessor of the current root node. Then, we replace the current root node with its presuccessor (Apply new *delete* operation on the presuccessor) and insert this root node to the right subtree. After this step, we can decrease the "number_balance" by 2 at the new root node and the "number_balance" becomes 0, satisfying the median property. After we delete the presuccessor, the median property in the left subtree may no longer hold. So we need to first dispel the influence of the left subtree after deleting the presuccessor by recursively applying the new *reorganize* operation. When the effect is dispelled, we move on checking.

- If the "number_balance" is -1 after decreasing, the procedure is similar to the first case. We just find the successor instead of presuccessor and adjust the right subtree instead of left subtree.

Q4.(iv)

The AVL tree mentioned in this task has height h.
For *insert* operation, the worst case happens when we need to reorganize the subtree every time we increase or decrease "number_balance". At the root node with height h, we need to apply one *reorganize* operation at height $h-1$. At the root node with height $h-1$, we need to apply one *reorganize* operation at height $h-2$. So on and so forth, until we complete insertion at leaf node.
For *delete* operation, the worst case also happens when we need to reorganize the subtree every time we increase or decrease "number_balance". At the root node with height h, we need to apply one *reorganize* operation at height $h-1$. At the root node with height $h-1$, we need to apply one *reorganize* operation at height $h-2$. So on and so forth.
Furthermore, every *reorganize* operation contains an *insert* operation and a *delete* operation at the same level. So we have following recurrence equation:

$$
\begin{aligned}
I(h) &= R(h-1) + R(h-2) + \ldots + R(1) + C_1 \\
D(h) &= R(h-1) + R(h-2) + \ldots + R(1) + C_2 \\
R(h) &= D(h) + I(h) + C_3
\end{aligned}
$$

By solving the above equations, we get:

$$R(h) = C * \sum_{i=1}^{h-1} i!$$

$$I(h) = D(h) = C * \sum_{i=1}^{h-1} \sum_{j=1}^{i} j! + C'$$

Besides, before reorganization, both *insert* and *delete* operations has complexity $O(log_2 n)$, so the worst case complexity for *insert* and *delete* is

$$O(\sum_{i=1}^{h-1} \sum_{j=1}^{i} j!)$$