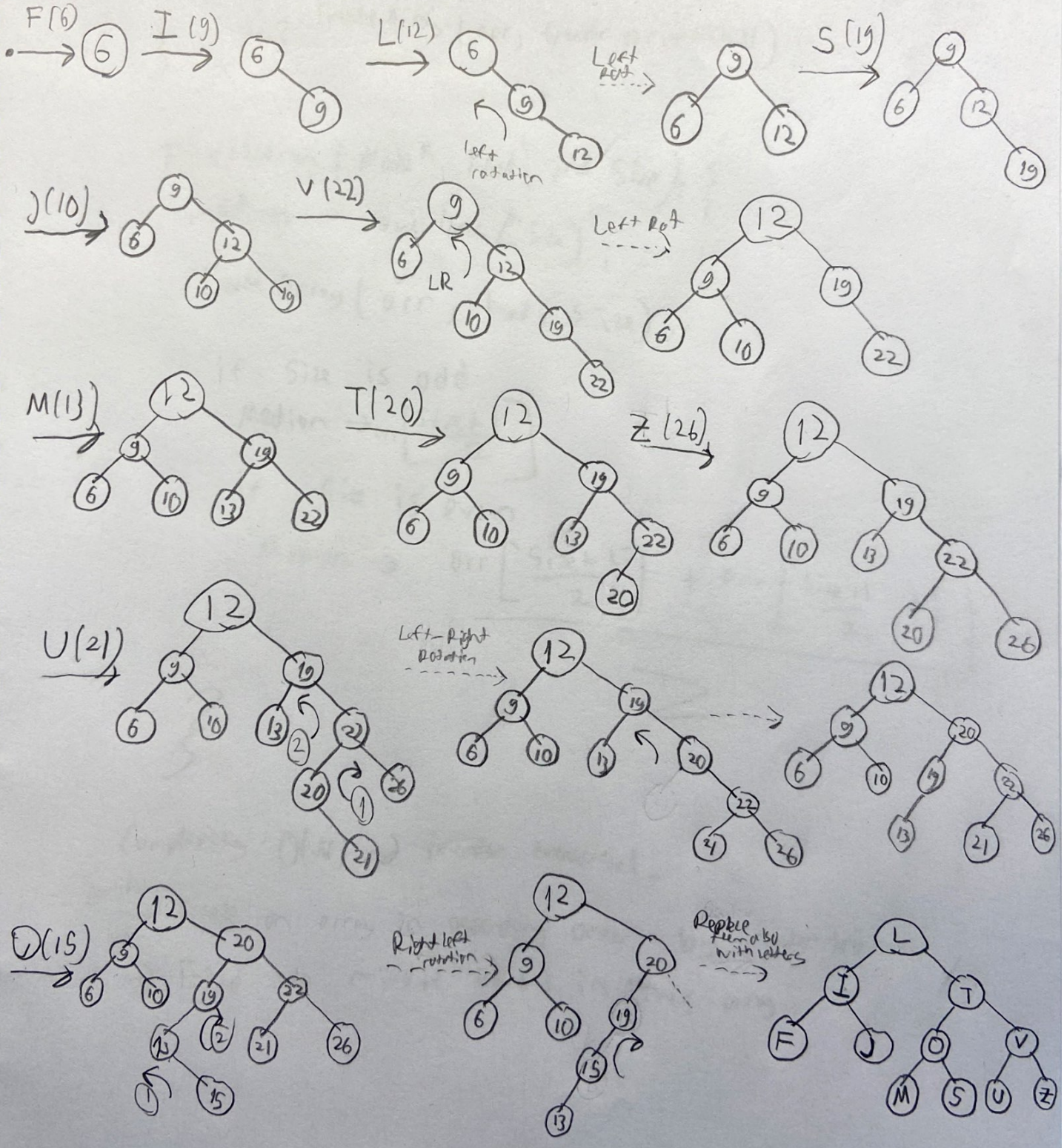
Q1) F, I, L, S, J, V, M, T, Z, U, O

q) F I L S J V M T Z U O
6 9 12 19 10 22 13 20 26 21 15

I will use numbers then replace them with letters, just to avoid any confusion.



b)

```
// uses inorder traversal to store elements in ascending order
CreateArray (int *arr, Node* treePtr, int size)
    if (treePtr is not Null) {
        CreateArray (arr, treePtr->leftChild, size);
        StoreElement (arr, treePtr, size); // Stores the element to array
        CreateArray (arr, treePtr->rightChild);
    }
}

FindMedian (Node* head, int size) {
    int* arr = new int [size];
    CreateArray (arr, head, size);
    if size is odd
        Median -> arr[(size+1)/2]
    if size is even
        Median -> (arr[(size+1)/2] + arr[(size+1)/2 + 1]) / 2
}
```

Complexity $O(N)$ → inorder traversal.

Algorithm
→ Create an array in ascending order by inorder traversal.
→ Find the middle element in this array

C)

```
bool checkAVL(Node* head) {
    if (head is null)
        return true;

    int leftHeight = head->left.Height();
    int rightHeight = head->right.Height();

    // if both subtrees are AVL and |Left height - Right height| <= 1
    // then the tree is AVL
    if (checkAVL(head->left) && checkAVL(head->right) &&
        |leftHeight - rightHeight| <= 1)
        return true;

    return false;
}

int height()
    // if node is null height is 0
    // else check height of left and right subtrees,
    // take the bigger one add 1 to it
```

// Algorithm checking if given tree is AVL

1. Check if left subtree is AVL
2. // // right // // - -
3. Check if |leftHeight - rightHeight| <= 1

if 1 2 3 are true return true

Algorithm Analysis

1. Checking the height of subtrees → O(N) → We visit each node and call the height function recursively
2. Checking if left-right trees are avl or not → O(N) → we check each node once.

↑
O(N)

(Q3) A better Strategy could be using an algorithm like binary search, we could take an expected number of computers, then look at the result.

if computer number is not enough we can double the computer number,

if we are too faster than the avg waiting time we could take the arithmetic average of the previous computer count and current and re-run again.

i.E

expected Number of Computers 5, Wanted time (50ms)

70 ms ↑ Simulate (5 computers) → Simulate (10) computers ↑ 25 ms

→ Simulate (7 computers) ↑ 49 ms ✓

This way we can approximate the running complexity O(log n) instead of O(N)