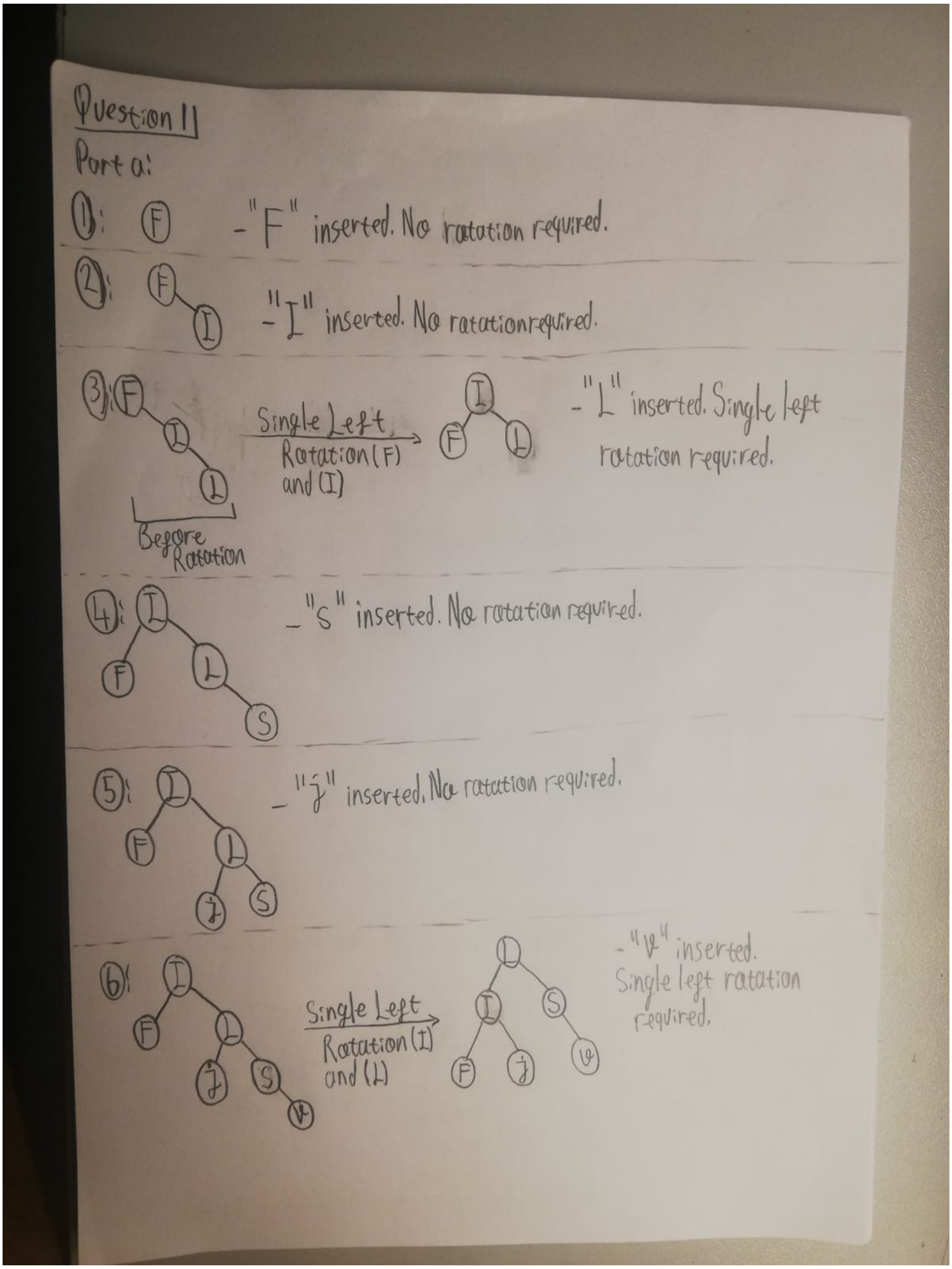
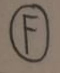
Name: Erkin Aydın Id: 22002956 Section: 2



**Question 1|**

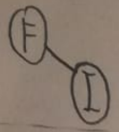
**Part a:**

1:



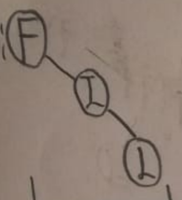
-"F" inserted. No rotation required.

2:



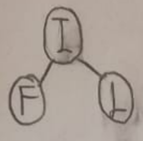
-"I" inserted. No rotation required.

3:



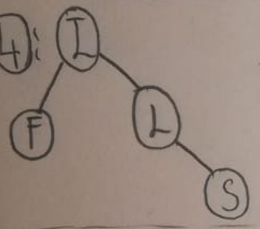
Before Rotation

$\xrightarrow{\text{Single Left}}$ Rotation(F) and (I)



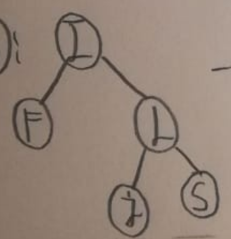
-"L" inserted. Single left rotation required.

4:



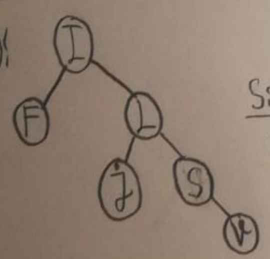
-"S" inserted. No rotation required.

5:

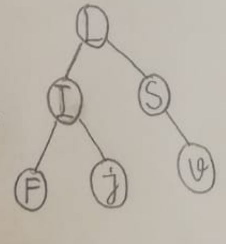


-"J" inserted. No rotation required.

6:

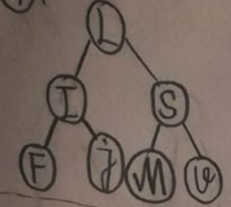


$\xrightarrow{\text{Single Left}}$ Rotation(I) and (L)



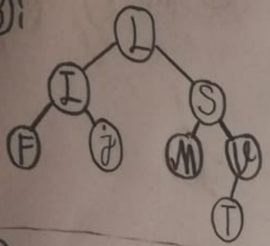
-"V" inserted. Single left rotation required.

7:



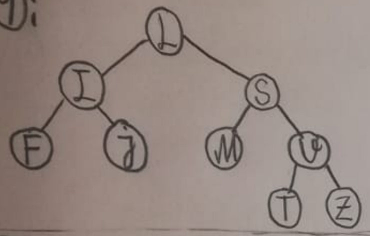
- "M" inserted. No rotation required.

8:



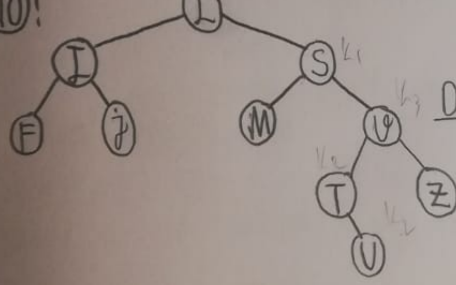
- "T" inserted. No rotation required.

9:



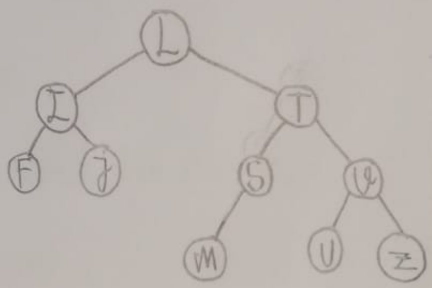
- "Z" inserted. No rotation required.

10:



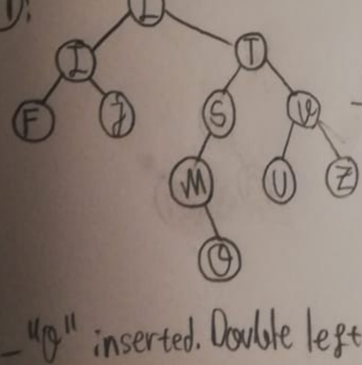
$\xrightarrow[\text{Rotation}]{\text{Double Right-Left}}$

- Right on (T) and (V)
- Left on (T) and (S)



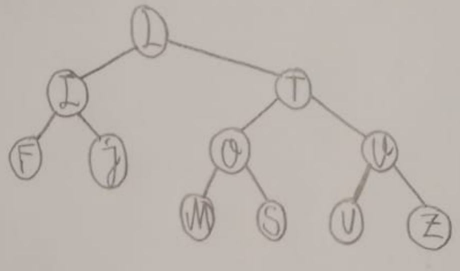
- "U" inserted. Double right-left rotation required.

11:



$\xrightarrow[\text{Rotation}]{\text{Double Left-Right}}$

- Left on (M) and (Q)
- Right on (Q) and (S)



- "Q" inserted. Double left-right rotation required.

part 6:

```
int computeMedian(Node* rootPtr)
    if rootPtr == NULL
        return -1
    endif
    int indexCount:=0
    inorderRecursive( rootPtr, inorderArr, indexCount)
    if rootPtr->size % 2 == 0
        return (inorderArr[rootPtr->size/2] + inorderArr[rootPtr->size/2 -1])/2
    else
        return inorderArr[rootPtr->size/2]
    endif
end computeMedian
void inorderRecursive( Node* rootPtr, int* arr, int& index)
    if rootPtr == NULL
        return
    endif
    inorderRecursive(rootPtr->leftChildPtr, arr, index)
    arr[index] := rootPtr->nodeValue
    index := index+1
    inorderRecursive( rootPtr->rightChildPtr, arr, index)
end inorderRecursive
```

Change in the node structure: "size" property added. It holds the size of the tree where the node is the root of that tree.

Time Complexity: O(logn), where n is the size of the AVL tree

Algorithm Logic: If the AVL tree is empty, then we return -1; if not, we first inorder traverse the tree and store the values of visited nodes in an array. Then, if the number of values is even we return the average of two of the values in the middle; if it is odd we return the middle value.

Part c:

```
bool checkAVL(Node* rootPtr){
    if rootPtr == NULL
        return true
    endif
    int leftHeight = 0
    int rightHeight = 0
    if rootPtr->leftChildPtr != NULL
        leftHeight = findHeight(rootPtr->leftChildPtr)
    endif
    if rootPtr->rightChildPtr != NULL
        rightHeight = findHeight(rootPtr->rightChildPtr)
    endif
    if (rightHeight - leftHeight) >= 2 OR (rightHeight - leftHeight) <= -2
        return false
    else
        return checkAVL(rootPtr->leftChildPtr) AND checkAVL(rootPtr->rightChildPtr)
    endif
end checkAVL
int findHeight(Node* rootPtr)
    if rootPtr == NULL
        return 0
    endif
    int leftHeight = findHeight(rootPtr->leftChildPtr)
    int rightHeight = findHeight(rootPtr->rightChildPtr)
    if (leftHeight > rightHeight
        return leftHeight +1
    else
        return rightHeight +1
    endif
end findHeight
```

Time Complexity: O(nlogn)
Algorithm Logic: If the BST is empty, then it is an AVL tree for sure. If not, we calculate the heights of the left subtree and right subtree. If they differ more then 1, then the BST is not AVL tree. If they differ 1 or 0, then we check whether both left and right subtrees are AVL trees or not.

Question 3|

It wouldn't be a good idea to start from 1 computer and increase the computer number in this case. Since we have N computers, at max as it says "potential", it reminds me of searching in a sorted data structure. Assume that we need exactly N computers, if we start with 1 computer, then we should also check 2, 3, 4, ..., N-1 computers in the old approach. If we know the max number N, then we can technically mimick binary search by starting N/2 computers, and then continue binary searching with respect to the average wait time of N/2 simulation.