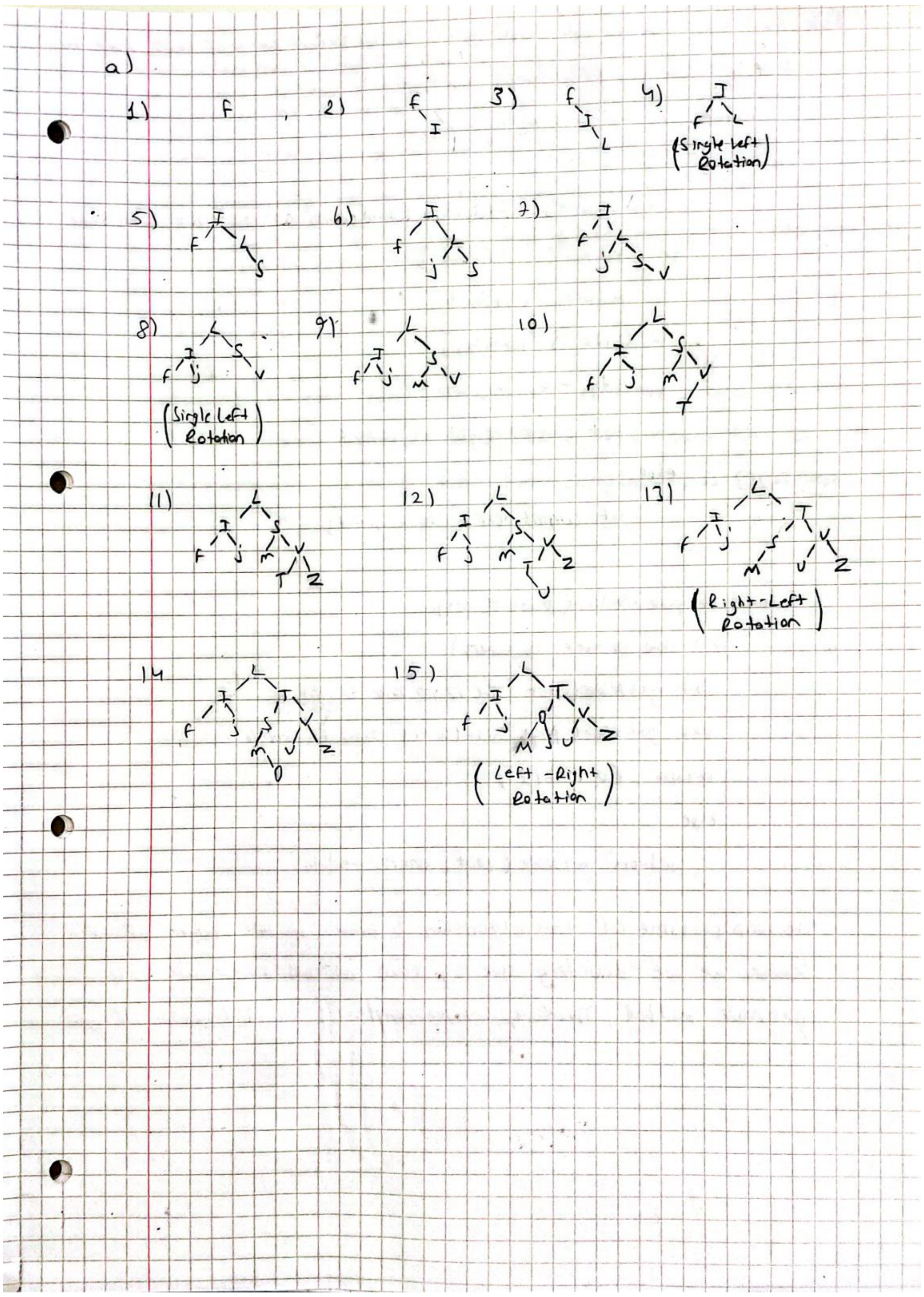
Gökberk Altıparmak

21901798

CS202-03

a)



b)

Node structure has one more variable “childCount” to keep track of the number of child Node has.

```
Node getNode(Node*& root, int idx)
1) set root to Node* current
2 for(;;)
2.1) set current’s left subtree’s childCount + 1 to int leftSize
2.2) if (idx == leftSize)
2.2.1) return current
2.3) if (idx > leftSize)
2.3.1) idx -= leftSize + 1
                              2.3.2) set current->right to current
                    2.4) else set current->left to current
```

```
Double computeMedian(Node *& root)
    1)  If (size of tree is even)
        1.1) set getNode(root, size/2)->data to int a
        1.2) set getNode(root, size/2-1)->data to int b
        1.3) return (a+b)/2
    2) else return getNode(root, size/2)->data
```

The running time is linear depending on the height of tree because we are dividing size by two at each time we call getNode method. Therefore, time complexity is O(height) = O(logn)

c)

```
Bool checkAVL(Node* root)
1) set height of root->left to int leftHeight
2) set height of root->right to int rightHeight
3) if (checkAVL(root->left) and checkAVL(root->right) and abs(rightHeigh-leftHeight < 2))
3.1) return true
4) return false
```

Height of tree can be found in n time and checkAVL method traverse right and left subtree separately thus time complexity of this algorithm is O(nlogn)

3)

For better solution, we may set the initial computer number according to the given number of requests so that we can find an optimal number to calculate average. It is like the similar procedure for quick sort algorithm.