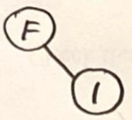
CS 202 - Homework 3

Mehmet Onur Uysal
22002609
Section 1

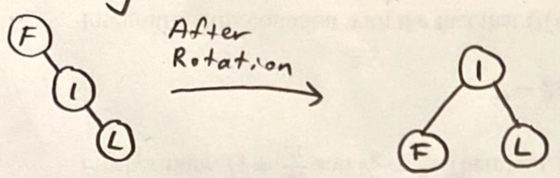
Question 1

- Inserting 'F'

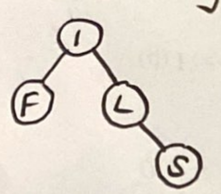
- Inserting 'I'



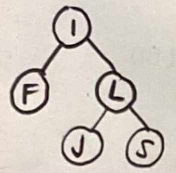
- Inserting 'L'



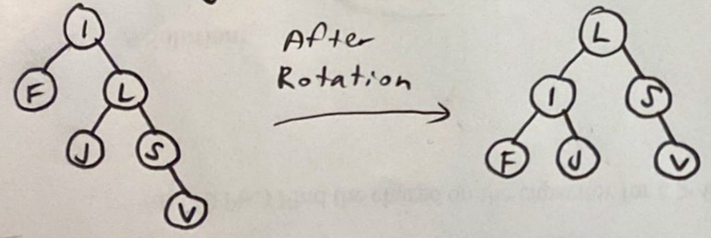
- Inserting 'S'



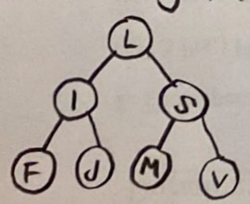
- Inserting 'J'



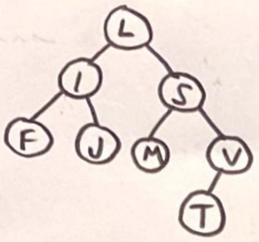
- Inserting 'V'



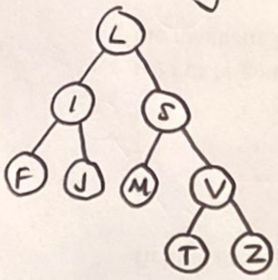
- Inserting 'M'



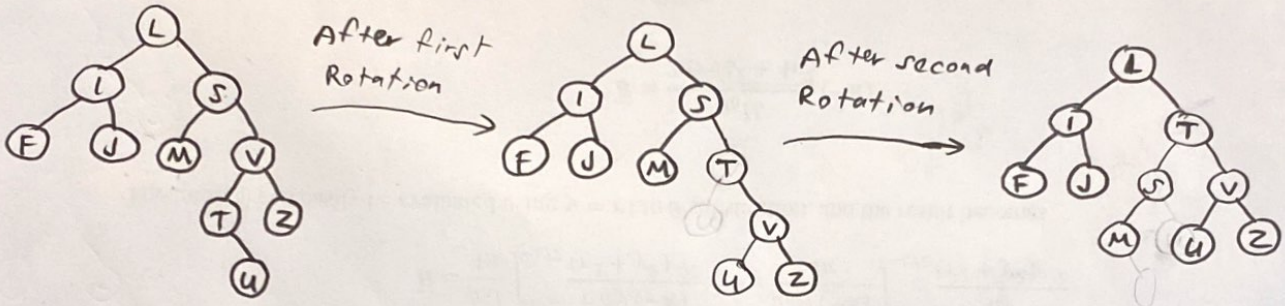
- Inserting 'T'



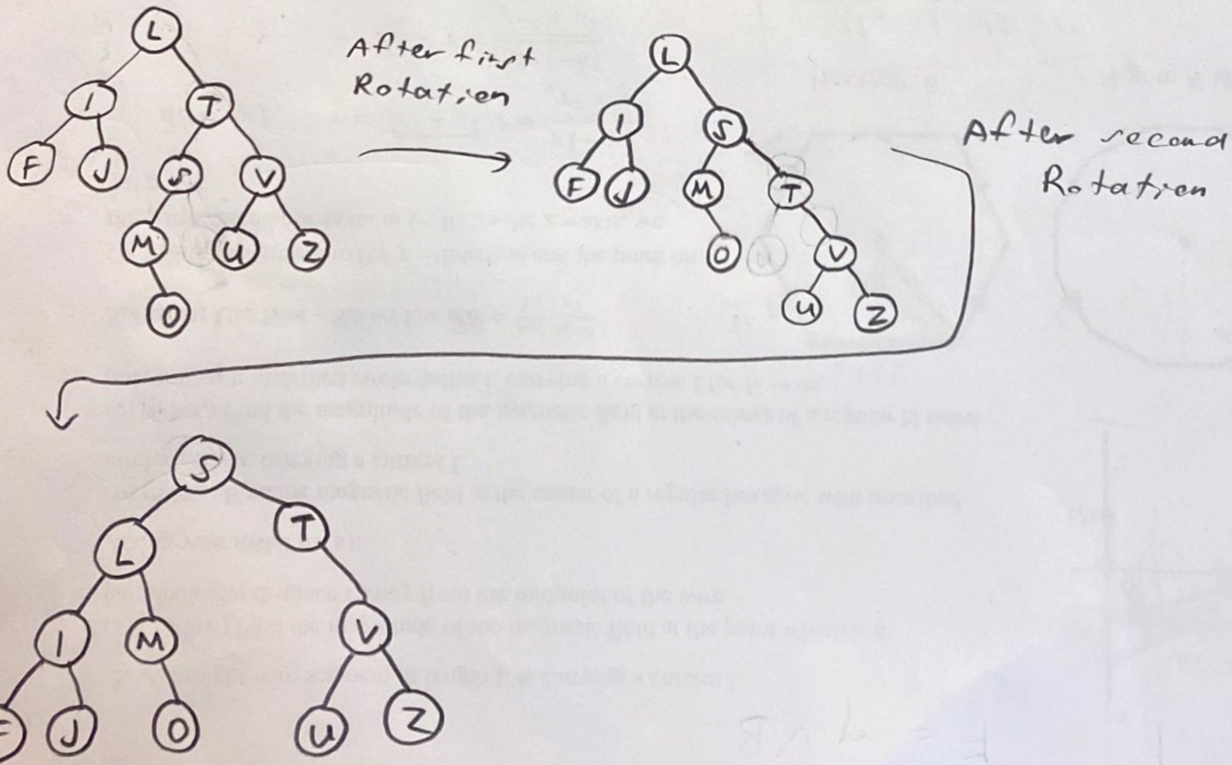
- Inserting 'Z'



- Inserting 'U'



- Inserting 'O'



Part b)

```
int computeMedian (root) {

    find the size from root size (root → size)
    if size is even
        index1 = size / 2
        index2 = index1 + 1
        find data at index1, assign to data1 } using helper
        find data at index2, assign to data2 } method
        return the average of data1 and data2
    else
        index = size/2 + 1
        return the data at index using helper method
}

int findData (root, index)
    if size of left subtree is equal to index - 1
        return root → data
    else if index is smaller or equal to size of left subtree
        return findData(root → left, index)
    else
        return findData (root → right, index)
}

// size attribute is added to tree nodes
```

ComputeMedian method gets the size of the tree from the root and decides how to compute the median. If the size is odd, median is simply the node at the middle. If the size is even median is the average of two nodes at the middle. After computing the indexes, helper method findData is called to acquire the data at necessary indexes. findData only traverses the necessary parts of the tree and since it is an AVL tree the time complexity is $O(\log N)$. Keeping the size in the nodes is a choice in order to compute the size of the tree which would decrease the efficiency.

Part c)

```
bool    check AVL (root) {
    if root is NULL
        return true
    else
        get the heights of left and right subtrees
        if absolute of their difference > 1
            return false
    else
        check balance of left subtree  }
        check balance of right subtree } recursive
        return true if they are both balanced
    else
        return false
```

This algorithm starts from the root and compares the balance of each node until it finds an imbalance. If the height attribute is kept in tree node, like checking the balance would take O(1) time, so time complexity is O(n) at worst case where all the nodes are visited and checked and O(1) at best case where the root has an imbalance. If the height is not stored at nodes computing the height will have O(n) time complexity which will make the methed run at O(n^2) time.

Question 3

If we have the same number of computers as the requests it is certain that we will get the minimum process time. So, if we start the tries from N/2 computers, we can determine in which half we should keep trying. For example if the waiting time does not exceed the given time we can try with N/4 computers and continue in the same manner recursively until we found the minimum number of computers that does not exceed the waiting time. Similarly if the waiting time exceeds the given time for N/2 computers, we can search the upper half recursively. This method will make the time complexity O(logN), which is way more efficient for very large number of requests.