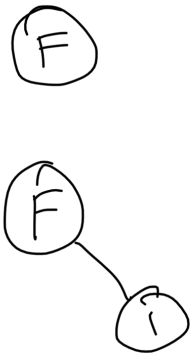
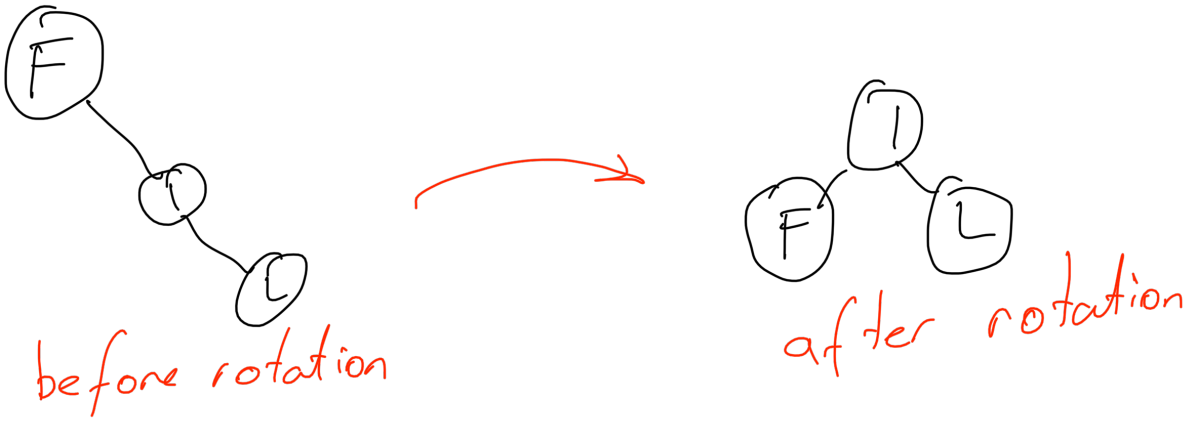
Alp Tugrul Agcali
21801799 – Section 3
CS 202 - HW 3

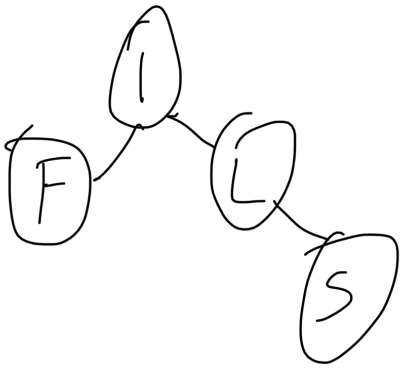
Q1. a

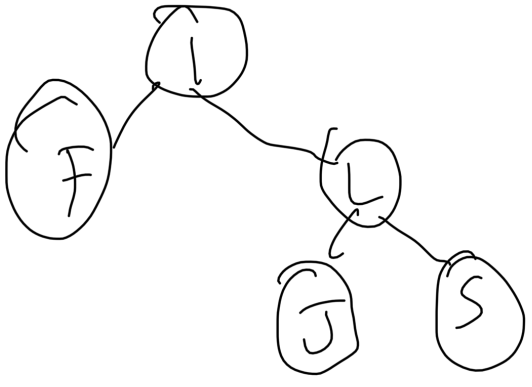


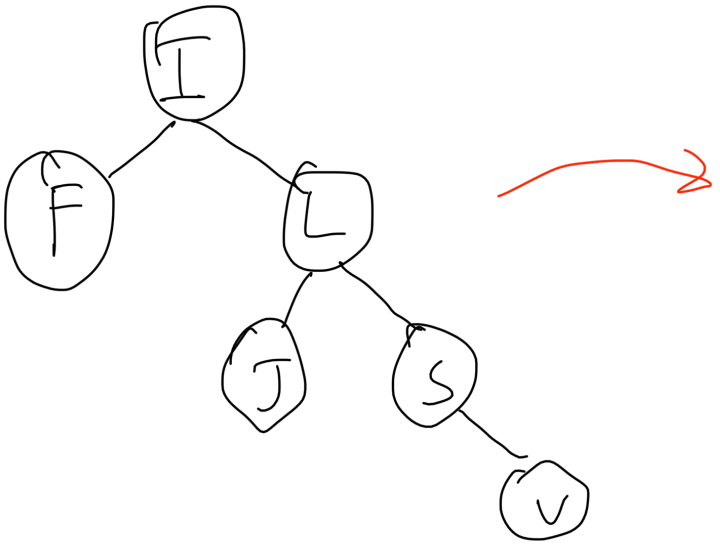


before rotation

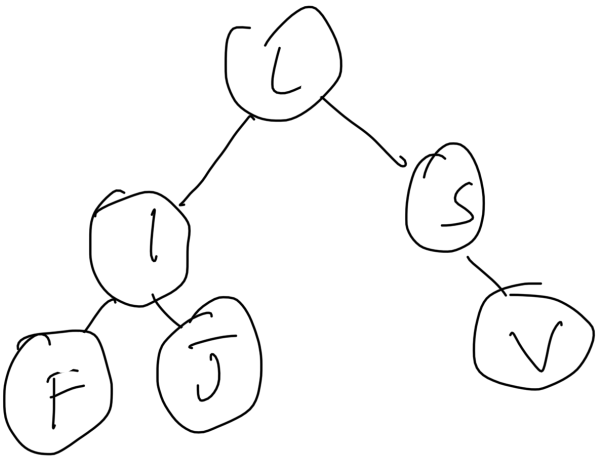
after rotation



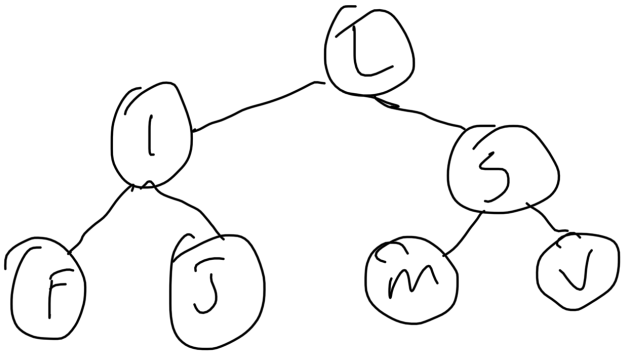


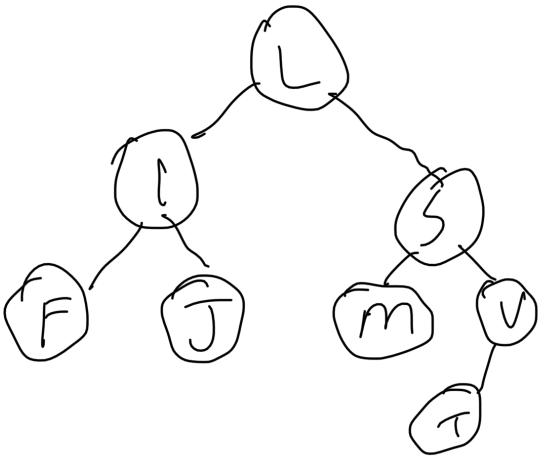


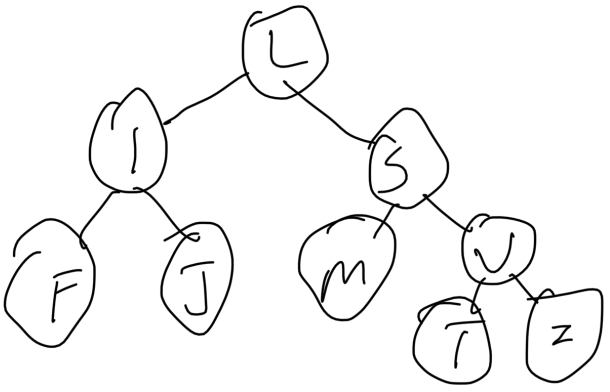
Before rotation

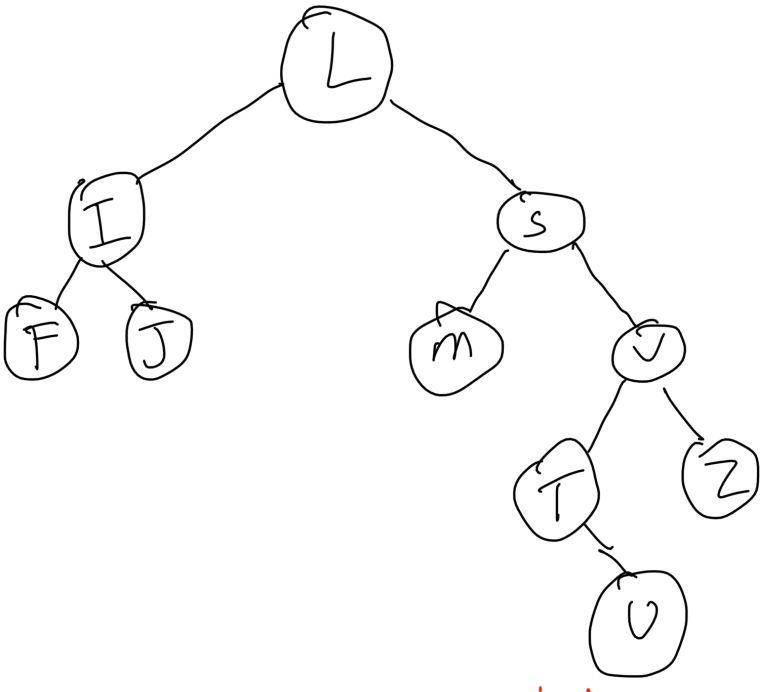


after rotation

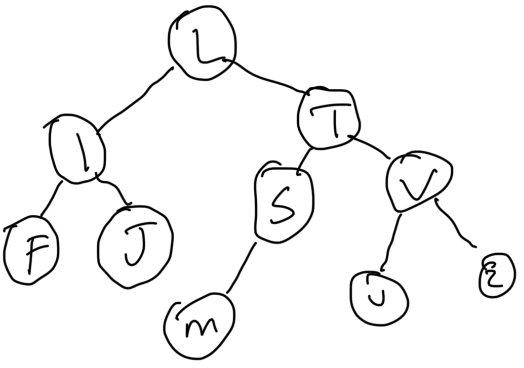




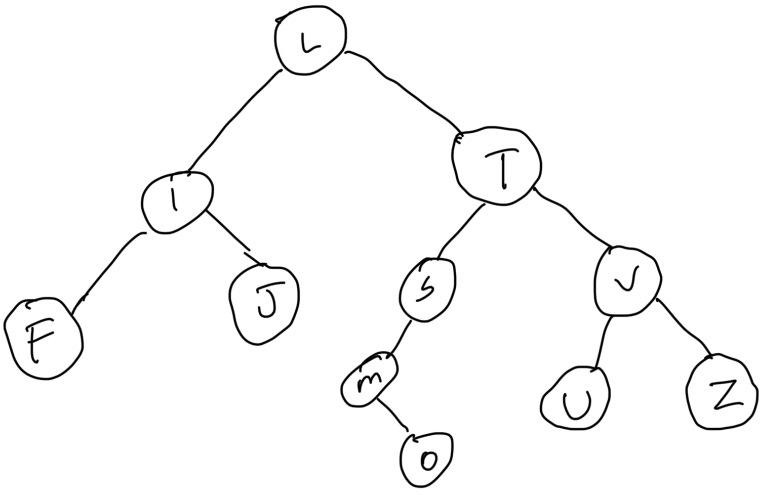




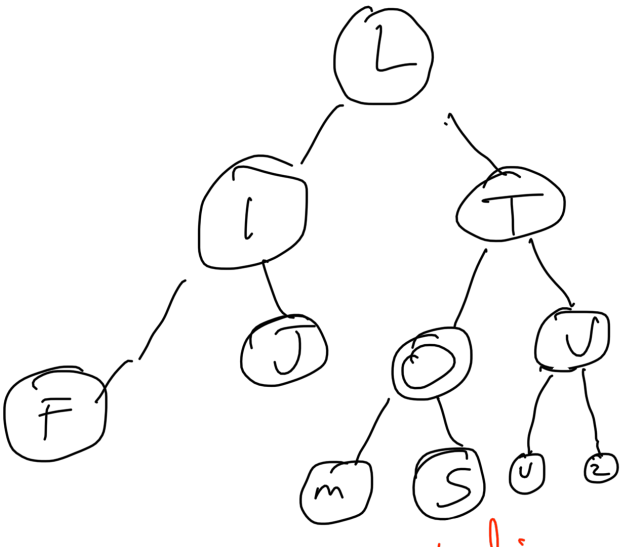
before rotation



after rotation



before rotation



after rotation

Q1. b

```
double computeMedian( Node* root ){
    int size1 = root->size;
    int middle1 = 0;
    int middle2 = 0;
    int count = 0;
    findMedian( root, size1, middle1, middle2, count );
    if ( size1 % 2 == 0 ){
        double result = (middle1 + middle2) / 2;
        return result;
    }
    else {
        double result = middle2;
        return result;
    }
}
void findMedian ( Node* root, int size, int& mid1
                  int& mid2, int& count ){
    if ( root == NULL ){
        return;
    }
    else {
        findMedian( root->leftchild, size, mid1, mid2, count );
        count ++;
        if ( count == (size/2))
            mid1 = root->data;
        else if (count == (Size/2)+1)
            mid2 = root->data;
```

findMedian(root->rightchild, size, mid1, mid2, count);

}

}

* In part b of Q1, we should add thi size of tree (or subtree) in node structure
* Time complexity is same with traversal. O(n)

Q1.c

```
int max(int a, int b) {
    if (a >= b)
        return a;
    else
        return b;
}

int height(node* root) {
    if (root == NULL)
        return 0;
    else return (1 + max(height(root->leftchild), height(root->rightchild)));
}

bool checkAVL(Node* root) {
    if (root == NULL)
        return 1;
    else {
        int left = height(root->leftchild);
        int right = height(root->rightchild);
        int difference = left - right;
```

```
if ( difference >= -1 && difference <= 1 &&
        checkAVL(root->leftchild) &&
        checkAVL(root->rightchild) ) {

    return 1;
}
else return 0;
}
```

* time complexity of height function is O(n).
So we have, n for calculating height and because of the recursive function is traversal n for recursive.
n * n = $O(n^2)$

Q3.

If we have many many more requests and because of that we will need much more computer, counting computers one by one is ineffective. Because, if we need N computers, with this method we simulate N times. Instead of that, if we have N potential computer, first we try N computer, then N/2 then N/4 or 3N/4 and it goes until we find the computer amount whose simulation time under and closest to the maximum time. With this method we only simulate $\log_2 N$ times.