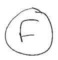
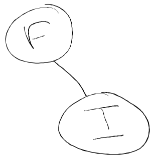
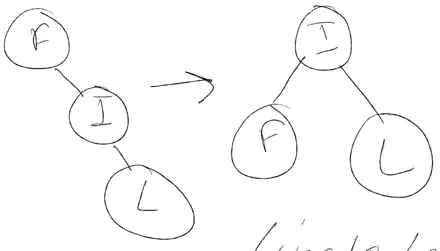
Insert 'F'



Insert I

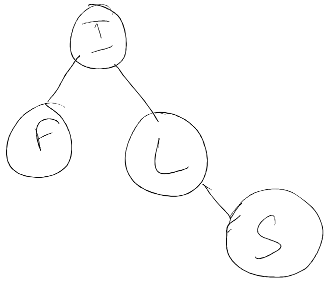


Insert 'L'

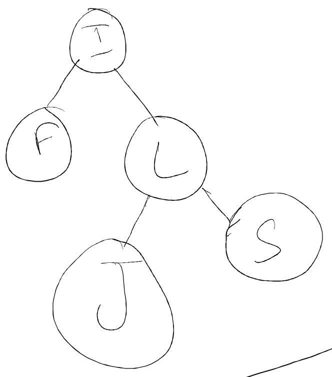


Single Left Rotation

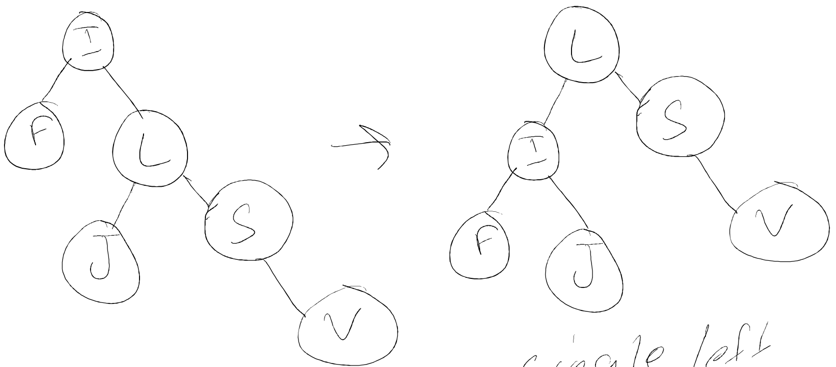
Insert 'S'



Insert 'J'

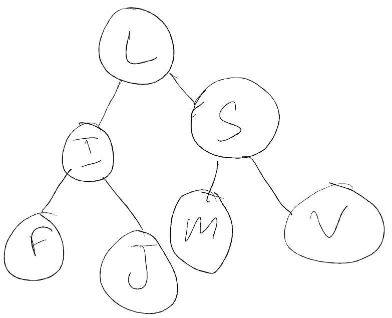


Insert 'V'

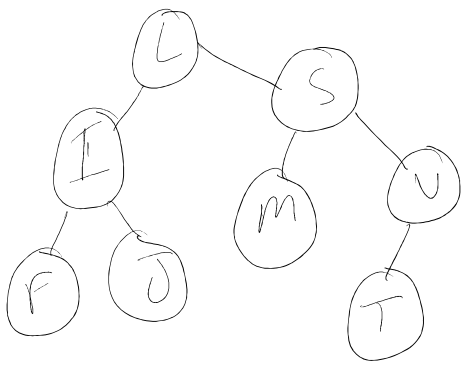


single left rotation at 'I'

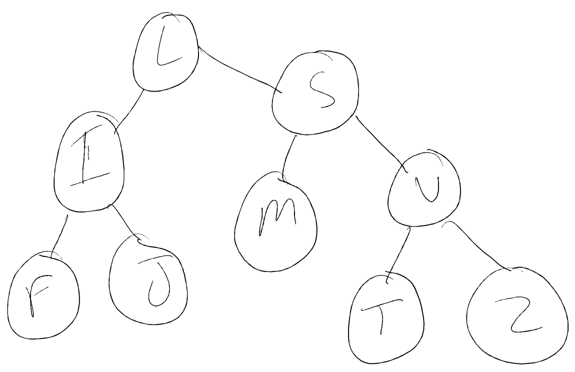
Insert 'M'



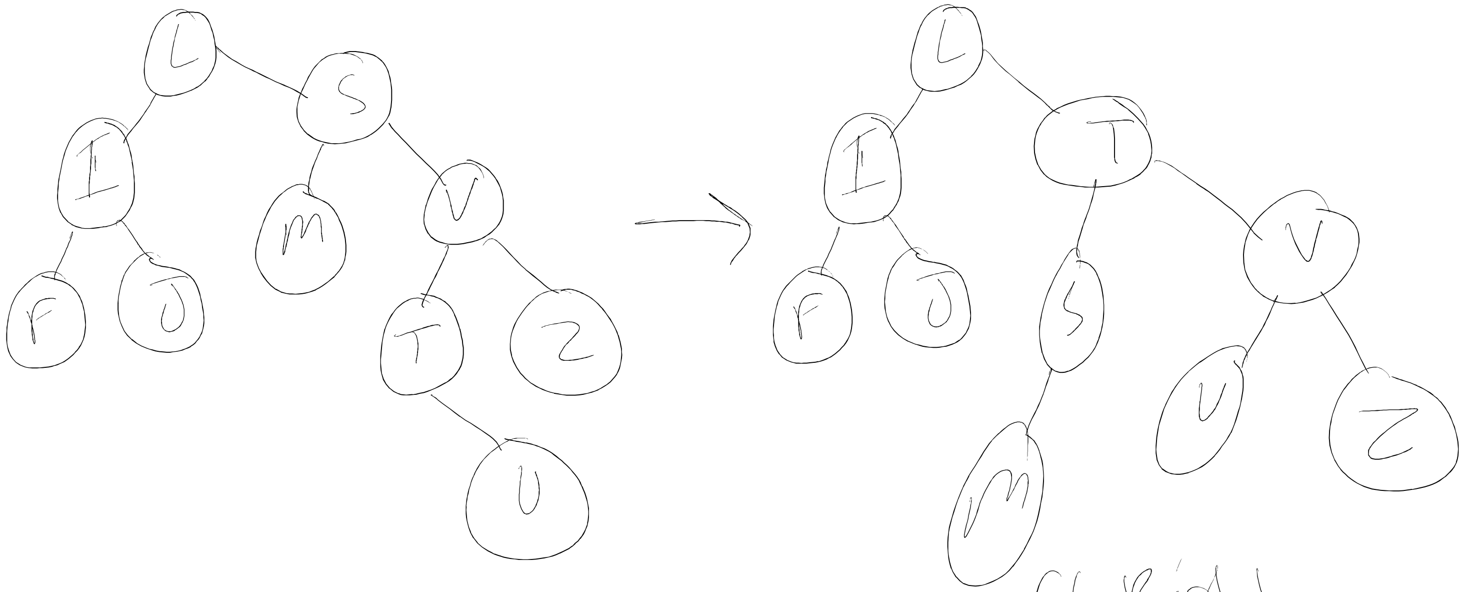
Insert 'T'



Insert 'Z'

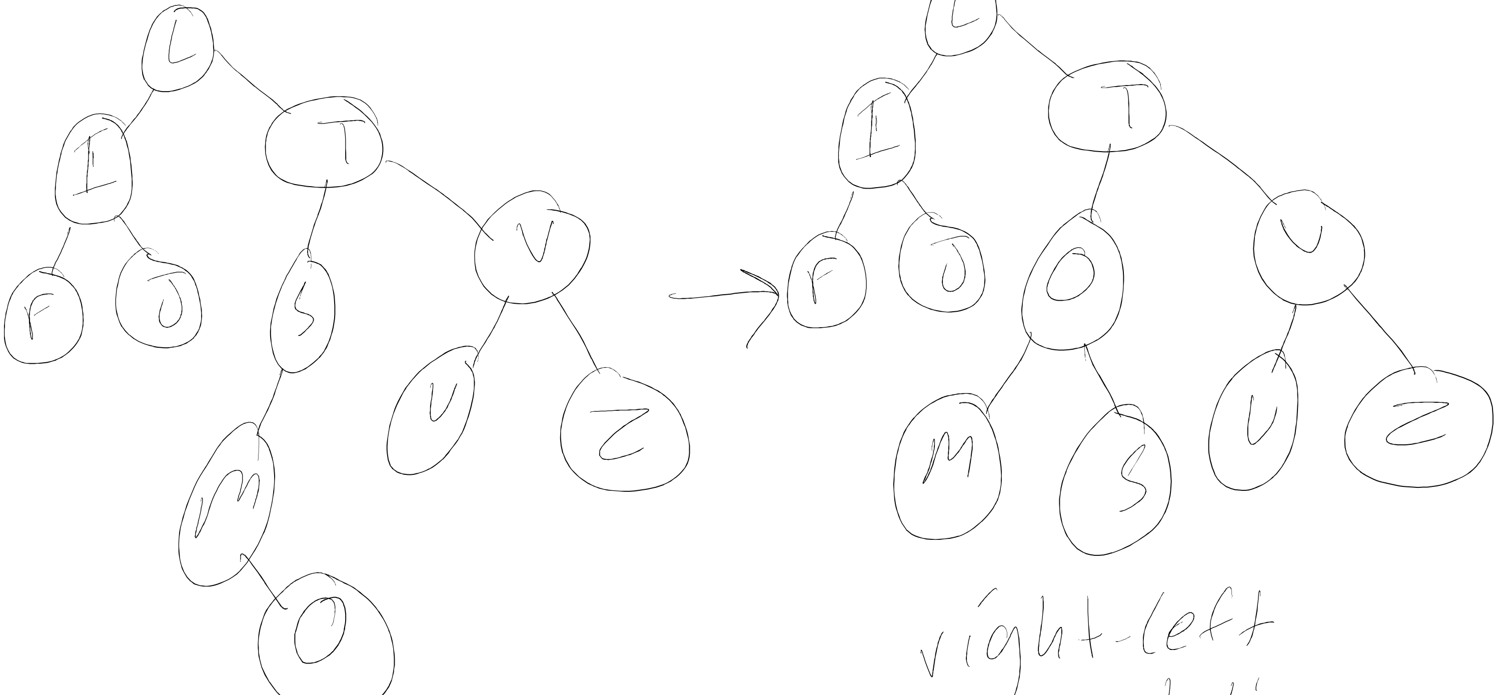


Insert 'U'



Left-Right rotation

Insert 'O'



right-left rotation

Q1b

Q1b)

```
double computeMedian(Treenode *rootptr){
        int n; //nodecount
        bool isodd;
        int nMedian, nMedian2;
        int nthval, nthval2;
    funct inorder (rootptr, incrementCount(n) .)   //incrementCount increases node count on
                                                   //each visit
        isodd = checkOdd(nodecount)

        if (isodd){
              nMedian = (n-1)/2

        else {nMedian = (n/2)
              nMedian2 = (n/2)-1}

          if (isodd){

        funct inorder (rootptr, getNodeValat(nmedian, nthval))  //get nodeValat has local node count
                                                                // that increments on each visit
         return nthval                                          //if nodecount == n, return the value at that
          }                                                     //point to nthval

         else {

          funct inorder (rootptr, getNodeValat(nmedian, nthval))
         funct inorder (rootptr, getNodeValat(nmedian2, nthval2))

      return (nthval + nthval2)/2
        }
```

The time complexity is $O(\log n)$. The logic of algorithm is to do an inorder travesel and get total number of nodes. If total number of notes is odd, the median element is at $\frac{n}{2}$ location (when going through AVL tree in inorder traverse) Likewise, if its even, the median is value of

$$\frac{(n/2) + ((n/2)-1)}{2}$$

(The treenode structure and methods refrenced here are from slides such as inorder function)

Q1c

```
int CheckAVL (Treenode* rootptr, bool& isbalanced = true) {        // in main function we would return
    return 0 if rootptr is null                                    // isbalanced

    int lh = CheckAVL (root->left, isbalanced)
    int rh = CheckAVL (root->right, isbalanced)
     if lh-rh is greater than 1, or less than -1 isbalanced = false

     return (max(lh,rh)+1) }
```

The timecomplexity is $O(n^2)$. The alogorithm is based on recursive calls. It takes root pointer till it reaches the bottom nodes. if the node is a leaf, it returns 0. for all else, it returns the max of height of their subtrees plus 2 for the node itself. if at any point the left and right height of nodes are greeted than 1 or less than -1, it makes boolean isbalanced false thus indicting it is not avl tree.

Q3

A better strategy to find optimum number of computers would be to start from a big number of computers and keep halving number of computers till waiting time is greater than required. The take that as minimum and keep increasing till we get satisfying waiting time for a number of computers