CS 202

Homework 3

Sec 03

Kutay Tire

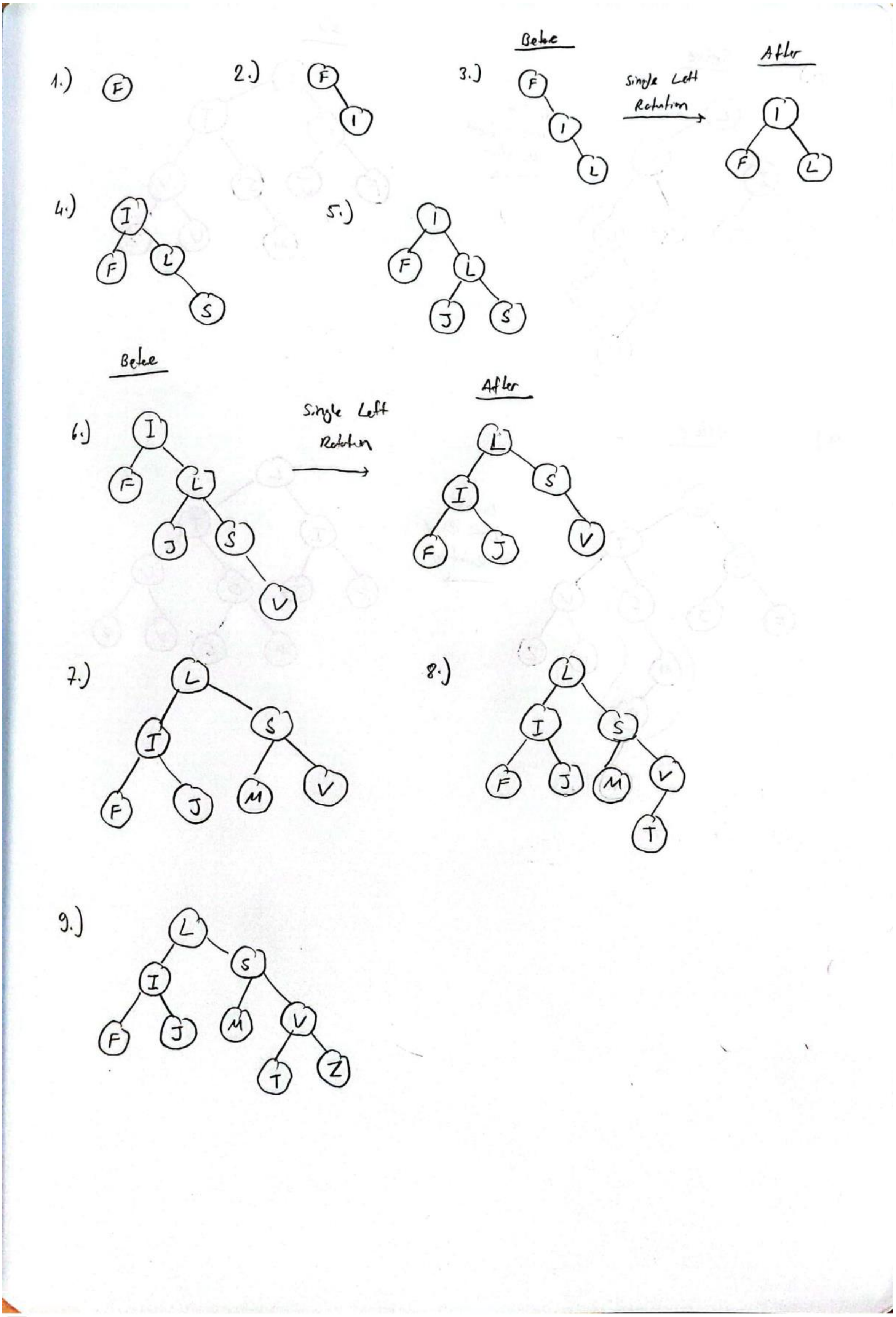
22001787

19.04.2022

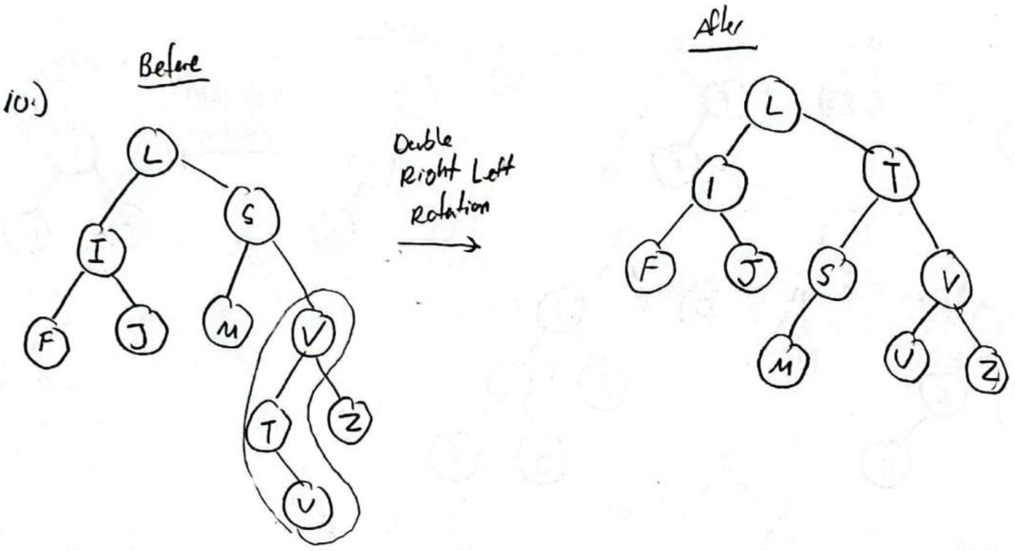
TA: Salman Mohammad

Instructor: Ertuğrul Kartal Tabak

Question 1.)


1.)
F
2.)
F
I
3.)
Before
F
I
L
Single Left Rotation
After
I
F
L
4.)
I
F
L
S
5.)
I
F
L
J
S
Before
6.)
I
F
L
J
S
V
Single Left Rotation
After
L
I
S
F
J
V
7.)
L
I
S
F
J
M
V
8.)
L
I
S
F
J
M
V
T
9.)
L
I
S
F
J
M
V
T
Z

10.)

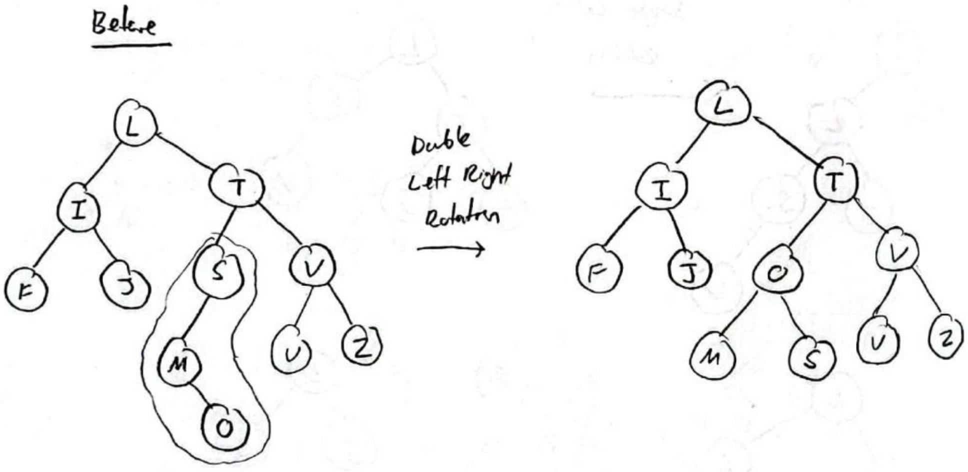


Before

Duble Right Left Rotation

Afler

11.)



Belore

Duble Left Right Rotatrn

b.) For this solution, each node holds a variable called "size" that holds the size of its subtree. Also, a helper method is used.

```
int computeMedian( Node *root) {
        if( root == NULL)
              return -1
        otherwise
              int count = 0
              int median = 0
              int med1 = 0
              int med2 = 0
              if( root → size is odd)
                     findMedian(root , count, (root→size)/2, median)
              otherwise
                     findMedian( root, count, (root→size)/2 -1, med1)
                     count = 0
                     findMedian( root, count, (root→size)/2, med2)
                     median = (med1 + med2)/2
        return median
}
void findMedian( Node *root, int & count, int target, int& median) {
      if ( count >= target || root == NULL)
          return
      findMedian( root→left, count, target , median)
      if ( count == target)
            median = root→item
      count = count +1
      if ( count > target)
          return
      findMedian ( root→right, count, target, median)
}
```

the algorithm uses the fact that AVL trees are infact binary search trees. As a result, the inorder traverse of the tree gives the elements in ascending order. For this, a target value is used to indicate the stopping index. When the total number of nodes is odd, it is enough to find only the value in the target index as it is the median. However, for even values, the list is traversed twice to get the average. For instance, for an AVL tree consisting of 2, 4, 6, 8; median is calculated as 5. To prevent unnecessary traversals, the algorithm returns when count exceeds target

When n is odd, the tree is traversed $n/2$ times. However, when n is even, it is traversed n times as $n/2 + n/2 = n$. Still, both indicate that the algorithm is linear with having a growth rate of $\underline{O(n)}$.

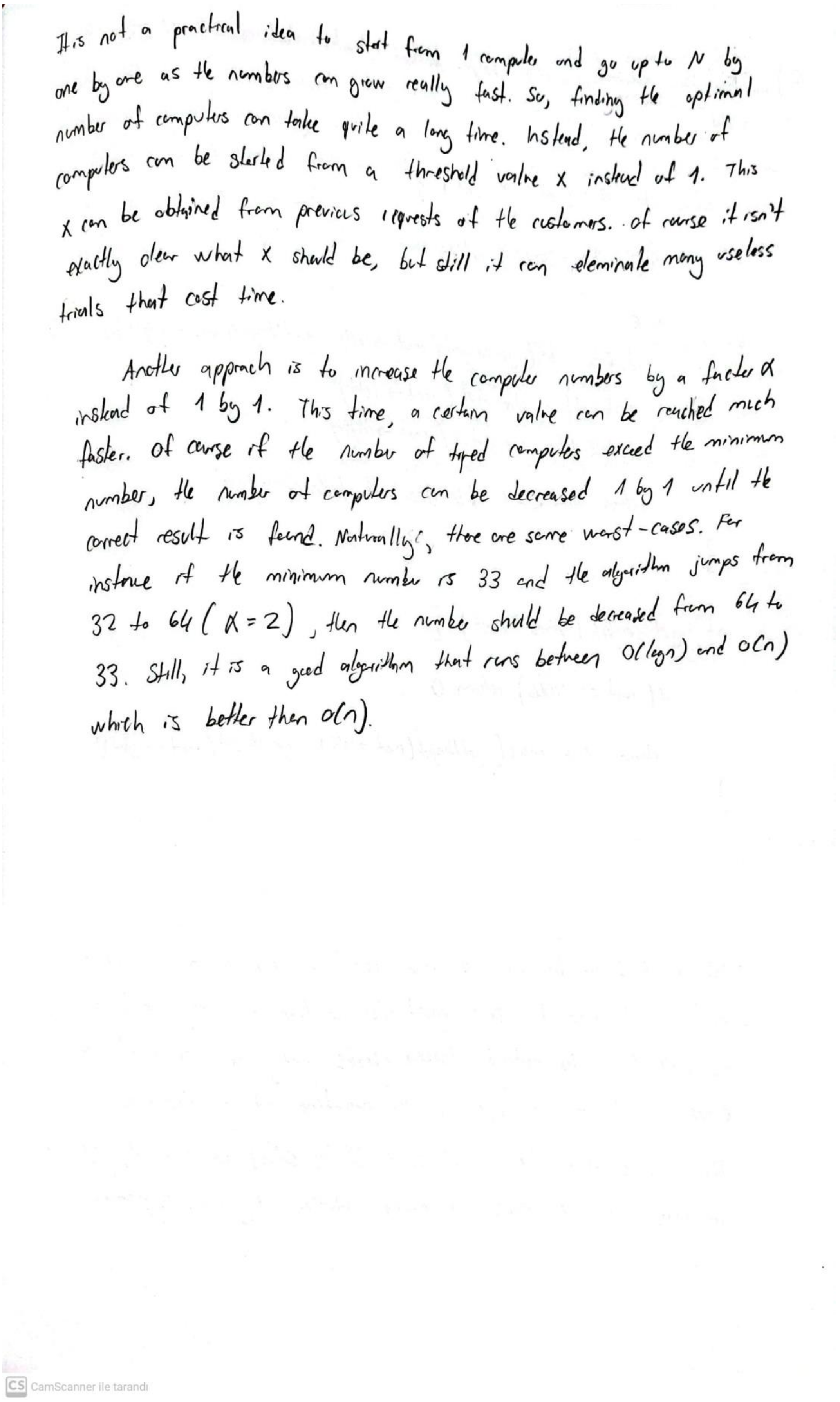
c.) For this solution, another helper method that calculates the height is used.

```
bool checkAVL ( Node * root) {
    if( root == NULL) return true
    else
        diff = abs( getHeight ( root → Left) - getHeight(root → right))
        checkLeft = checkAVL ( root → Left)
        checkRight = checkAVL( root → right)
        return diff <= 1 && checkLeft && checkRight
}

int getHeight ( Node * root) {
    if( root == NULL) return 0
    return 1 + max( getHeight (root → left), getHeight ( root → right))
}
```

For a BST to be an AVL tree, height difference between its children must be at most 1. This must also be true for every node of the AVL tree. The method therefore checks every node recursively and checks itself at the beginning. The run-time of the algorithm is $O(n^2)$ where n is the size of BST. It is $O(n^2)$ because the algorithm needs to find the height of every node by calling getHeight (Node * root). This method takes $O(n)$. Using it for every node makes the algorithm $O(n^2)$ by simply multiplying n with n.

Question 3.)



It is not a practical idea to start from 1 computer and go up to N by one by one as the numbers can grow really fast. So, finding the optimal number of computers can take quite a long time. Instead, the number of computers can be started from a threshold value x instead of 1. This x can be obtained from previous requests of the customers. Of course it isn't exactly clear what x should be, but still it can eleminate many useless trials that cost time.

Another approach is to increase the computer numbers by a factor α instead of 1 by 1. This time, a certain value can be reached much faster. Of course if the number of tried computers exceed the minimum number, the number of computers can be decreased 1 by 1 until the correct result is found. Naturally, there are some worst-cases. For instance if the minimum number is 33 and the algorithm jumps from 32 to 64 ($\alpha = 2$), then the number should be decreased from 64 to 33. Still, it is a good algorithm that runs between $O(\log n)$ and $O(n)$ which is better then $O(n)$.