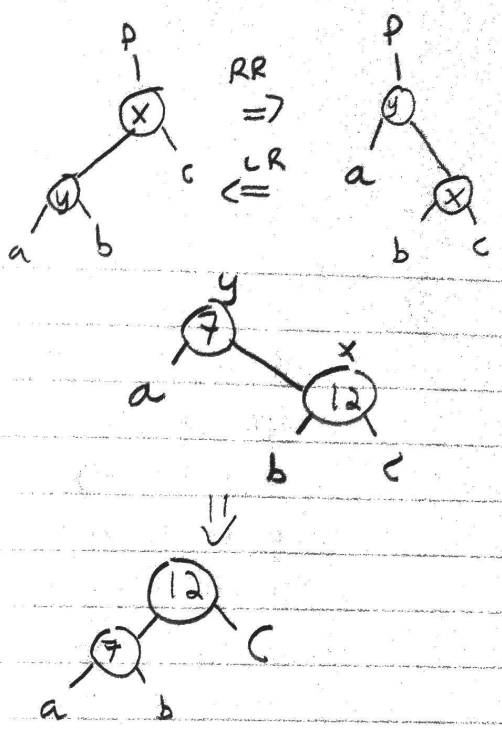
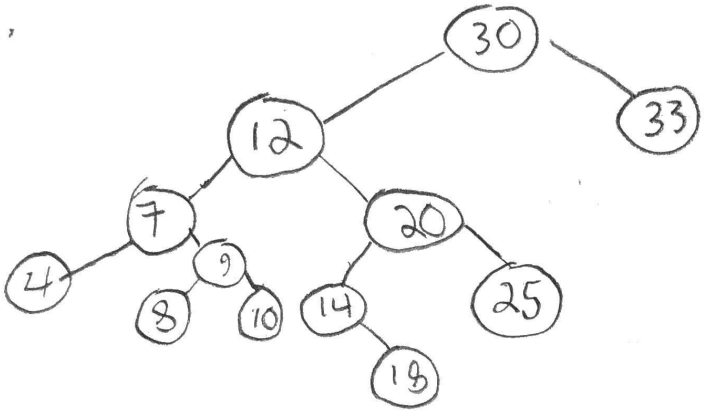
Robert Frenken

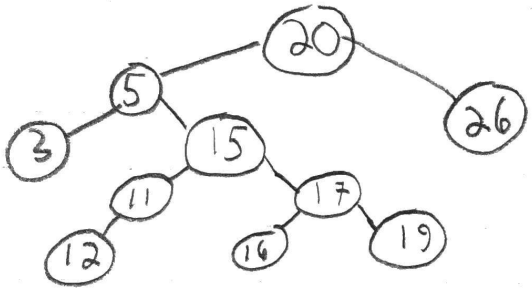
Homework 9

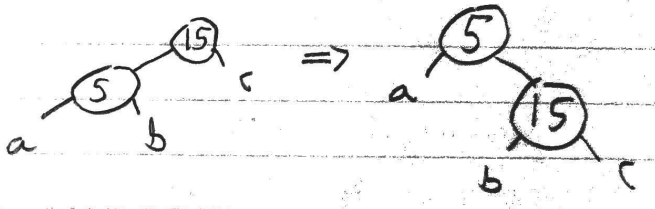


1.

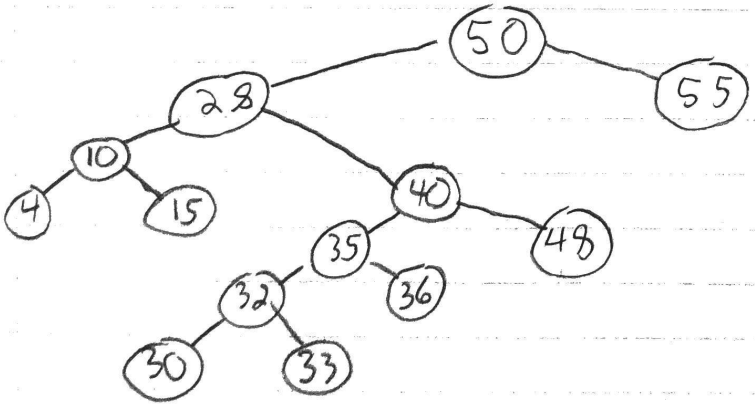


2.





3. Remove 25, replace w/ 28
Transplant 28 w/ 32



4.

a)
```
function InsertAndUpdateSum(T, z)
TreeInsert(T,z);
z.sum ← z;
y ← z.parent;
while (y ≠ NIL) do
    y.sum ← y.sum + z.sum
    y ← y.parent
end
```

Assuming z is an int, record that value in the sum field, then work way up the tree updating all the parents sum fields

b) The precondition for this algorithm is that the previous sum fields are correct, and the insertion is correct. Given this, the sum of z can be found, as that's the attribute of the element, and the parent sums are updated as the algorithm iteratively goes through and adds the value of z on top of the current sum, with the last update being at the root

c) The first part of the function is constant, and the while loop is contingent on the height of the tree, so the running time is $\in \Theta(h)$

5. a) function RightRotateSum(T, x)

```
y ← x.left;
b ← y.right;
Transplant(T, x, y);
x.left ← b
if (b ≠ NIL) then b.parent ← x;
y.right ← x
x.parent ← y
x.sum ← x.sum - y.sum + b.sum;
y.sum ← y.sum + 1 + x.right.sum;
```

end

The modification is to make the rotation then update the sum fields for both x and y

b) This algorithm correctly rotates x and y, and updates their child and parent nodes. Then it updates the sum of x, by subtracting y's sum field, and adding b.sum as b is still a child of x. It then updates y by adding the right child sum of x, or < from the slides.

c) The worst case is that the rotations have transplants up to the root for a runtime of $\in \Theta(h)$

6. a)

```
function TreeSumLE(x, K)
 SumLE <- 0
 V <- x
 while (V ≠ NIL) do
   if (V.key ≤ K) then
     if (V.left ≠ NIL) then  SumLE <- SumLE + V.left.sum
     V <- V.right
   else
     V <- V.left
   end
 end
 return(SumLE)
```

b) This function goes down from the root, and will sum the left side of the root if the key is less than K, then preforming the same check on the right child, or going to the left child. This ensures that it will capture all values less than K

c) The worst case is that it traverses all the way down the longest path, or $\in \Theta(h)$