# Worksheet 31 - Group 1

## Worksheet Group 1 Members

Marc Clinedinst: clinedim@onid.oregonstate.edu
Kelby Faessler: faesslek@onid.oregonstate.edu
James Fitzwater: fitzwatj@onid.oregonstate.edu
Tom Gariepy: gariepyt@onid.oregonstate.edu
Sean Reilly: reillys@onid.oregonstate.edu
Joseph Struth: struthj@onid.oregonstate.edu

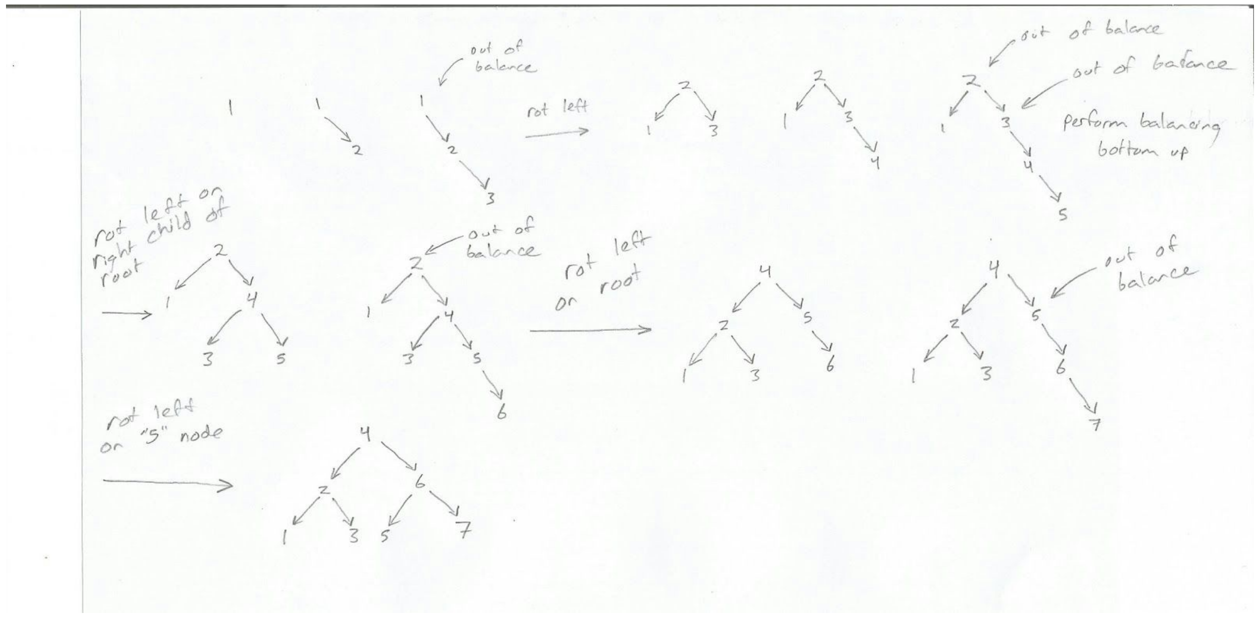
## Collaborators

Marc, Kelby, James, Tom, Sean, Joseph

## Worksheet 31: AVL Trees

In this assignment, we implement three different functions related to AVL trees. More specifically, we implement the `_rotateLeft`, `_rotateRight`, and `_removeNode` functions. These functions are described in detail below and are accompanied by comments where needed.

Insert the values 1 to 7 into an empty AVL tree and show the resulting tree after each step. Remember that rebalancing is performed bottom up after a new value has been inserted, and only if the difference in heights of the child trees are more than one.



```
/*
```

```
This function takes a pointer to an AVLnode as a parameter. It then
rotates the subtree rooted by the passed node to the left, and
returns the resulting tree.
*/
struct AVLnode* _rotateLeft(struct AVLnode* current) {
   assert(current != 0);
   if(current->right != 0) {
      struct AVLnode* tmp = current->right->left;
      struct AVLnode* newRoot = current->right
      newRoot->left = current;
      current->right = tmp;
      _setHeight(current);
      _setHeight(newRoot);
   }
   return current;
}


struct AVLnode * _rotateRight (struct AVLnode * current) {
   assert(current != 0);
   if (current->left != 0) {
      struct AVLnode* tmp = current->left->right;
      struct AVLnode* newRoot = current->left;
      newRoot->right = current;
      current->left = tmp;
      _setHeight(current);
      _setHeight(newRoot);
   }
   return newRoot;
}



struct AVLNode *_removeNode(struct AVLNode *cur, TYPE val) {
     if (cur->value == val) {
          if (cur->right == 0) {
               struct AVLNode *temp = cur->left;
               free(cur);
               return temp;
          } else {
                cur->value = _leftMost(cur->right);
                cur->right =_removeLeftmost(cur->right);
            }
        } else if (val < cur->value) {
            cur->left = _removeNode(cur->left, value);
        } else {
            cur->right = _removeNode(cur->right, value);
        }
        _balance(cur);
        return cur;
}
```

**Piazza Discussion**

https://piazza.com/class/ib2kus4hsie528?cid=209