1. For the Binary Search Tree, shown in Figure 1, remove the node with value 60, insert a node with value 17, and show the resulting BST.

![BST Diagram]

Figure 1. BST

2. Transform the Binary Search Tree, shown in Figure 1, to the AVL tree. After each step of your transformation, draw the resulting tree. How many balancing transformations do you need?

3. Given the AVL tree, which you obtained in Problem 2, remove the node with value 60, insert a node with value 17, and show the resulting AVL tree.
4. For the following **Binary Search Tree**, list the nodes as they are visited in a **pre-order, in-order, and post-order** traversal.

5. **AVLTree Implementation** -- Complete the function below to perform a left rotation in an AVL Tree. Assume you have a function `_setHeight()` that takes an AVLNode as a parameter, and sets its height appropriately.

```c
struct AVLNode *rotateLeft(struct AVLNode *cur) {
}
```
6. Write a recursive function `sumTreeNodeHelper(struct BNode *node, double *sum)` that sums the elements of a binary search tree starting with the smallest element, and adding elements in order up to the largest element and prints the running sum as each new value (i.e., `node->val`) is added to the sum (We assume double values are stored in the tree). You will not use an iterator.

```c
void sumTreeNodeHelper(struct BNode *node, double *sum) {
    printf("Running sum = %f\n", *sum);
}

void sumTree(struct BSTree *tree, double *sum) {
    sumTreeNodeHelper(tree->root, sum);
}
```

/* Example use */
struct BSTree myTree;
int sum = 0;
/* Initialize and Add elements to the tree */
addTree(&myTree, 5);
addTree(&myTree, 1);
addTree(&myTree, 10);

sumTree(&myTree, &sum);
printf("Final sum = %fn", sum);
...

The output of the above should be:

Running sum = 1
Running sum = 6
Running sum = 16
Final sum = 16
7. Write a function that takes as input a BST, and prints out node values in the breadth-first order. The breadth-first order traversal of the BST in Figure 1 (see problem 1), produces: 33, 20, 60, 10, 25, 100, 15...

```c
void PrintBreadthFirstBST(struct BSTree *tree)
{
}
```

8. Add the following numbers, in the order given, into an empty AVL Tree: {3, 15, 2, 99, 54, 34, 5}, and draw the resulting AVL tree.

9. Add the following numbers to an empty SkipList in the given order: {3, 9, 23, 45, 6, 15, 1, 34, 4}, using the following sequence of coin flips: {T, H, T, H, H, T, H, H, T, T, H, T, H, H, T, H, T, T, H}(you may not use all of the flips). Draw the resulting SkipList.

10. What is the difference between _setCapacity function for the Dynamic Array implementation of a Deque, and _setCapacity function for the Dynamic Array implementation of a Stack?
11. Write the pseudo code for `contains` and `remove` functions of a SkipList:

```c
skipListContains (struct skipList *slst, TYPE val) {
    current = topmost sentinel;
    while (
    current = slide right (   )
    if (  )
        return true;
    current = current down
} return false;
}
```

```c
skipListRemove (struct skipList *slst, TYPE val) {
    current = sentinel;
    while (  ) {
    current = slide right (   )
    if (  ){
        remove
    if (  ){
        reduce the element count
    }
    current = current down
}
}
12. Write the pseudo code for the function addSkipList:

```c
addSkipList ( ) {
}
```

13. The smallest element in a binary search tree is always found as the leftmost child of the root. Write the function `getFirstBST` to return this value, and the function `removeFirstBST` to modify the tree so as to remove this value.

14. Write the function `int equalsBST(struct BSTree *tree1, struct BSTree *tree2)` that returns `true` if two input trees are equal, i.e., if they have the same elements, or `false`, otherwise. What is the complexity of your operation?