Problem Set 7, Part I

Problem 1: Choosing an appropriate representation

1-1) *ArrayList or LLList?*

Explanation: LLList, because the volume changes and only the first option will be shown, so the dummy head node can be used as the place of an insertion, and as the orders are delivered you can delete from the front. It'll end up acting a lot like a stack FILO, since orders that were made recently are shown at the front (top) of the list.

1-2) *ArrayList or LLList?*

explanation:ArrayList, because the number of workshops if fairly consistent, so there won't really need to be a case where the number of workshop exceeded a predetermined maximum number of workshops– which can be made arbitrarily large in comparison to the number of workshops. This implementation will allow O(1) access to any given workshop, and the implementation could also use the Queue interface as well. The Queue ADT is used because the appearance of a workshop will be in relation to when it was uploaded–chronologically, so the Queue will preserve the order in which they were uploaded and hence how they are displayed.

1-3) *ArrayList or LLList?*

Explanation: A LLList would work best if additions and deletions were necessary, but a ArrayList would be the best, because the course size is fixed and random access will allow for rapid traversal of the array for when a certain course's information needs to be updated. The Id of the course would be used as the position.

Problem 2: Switching from one list implementation to another

2-1) O(n), the for-loop is O(n), and the getItem() and addItem() are inside the scope of the for-loop and each are O(1) and O(1) respectively. The O(n) term is bigger so the resulting expression is O(n) * O(1) which is O(n).The addItem is O(1), because were are moving backwards and adding to front, so we don't have to traverse the array n times to get to where we want to add the node.The getItem() method is being used on an array, so its random access and therefore O(1).

2-2) O(n^2), the for-loop is O(n), and the getItem() and addItem() are inside the scope of the for-loop and each are O(1) and O(n) respectively. The O(n) term is bigger so the resulting expression is O(n) * O(n) which is O(n^2). The addItem method is O(n), because were are moving forward, so when we want to add a mode, we have to traverse the entire array inorder to add a node. The getItem() method is being used on an array, so its random access and therefore O(1).This implementation is worse than the first implementation.

2-3)

```
public static ArrayList convert_LtoA_v1(LLList list){
        ArrayList converted = new ArrayList(LLList.length());

        ListIterator iter = list.iterator();
        int i = 0;
        while(iter.hasNext()){
                Object itemAt = iter.next();
                converted[i] = itemAt.data;
                i++;
        }
} (needs testing)
```

2-4) It's O(n^2), because I need to set the length of the ArrayList using the length() method which has to traverse the entire LLList and the while loop runs O(n) times.

Problem 3: Working with stacks and queues
3-1)
```
public static void doubleAllStack(Stack<Object> stack, Object item) {
        //stack to queue then to stack
            LLStack<Object> stonks = new LLStack<Object>();

            while(!stack.isEmpty()){
                if(stack.peek() == item){
                    stonks.push(stack.pop());
                    stonks.push(item);
                }
                stonks.push(stack.pop());
            }
            //LLStack<Object> panik = new LLStack<Object>();


            while(!stonks.isEmpty()){
                stack.push(stonks.pop());
            }

    }






}
```

3-2)
```
 public static void doubleAllQueue(Queue<Object> queue, Object item) {

        LLQueue<Object> british = new LLQueue<Object>;

        while(!queue.isEmpty()){
            if(queue.peek() == item){
                british.insert(queue.remove());
                british.insert(item);
            }
            british.insert(queue.remove());
        }

        while(!british.isEmpty()){
            queue.insert(british.remove());
        }
    }
```

Problem 4: Binary tree basics

4-1) height is 3

4-2) There are 4 leaf nodes

4-3) preorder: 21,18,7,25,19,27,30,26,35

4-4) postOrder: 7,19,25,18,26,35,30,27

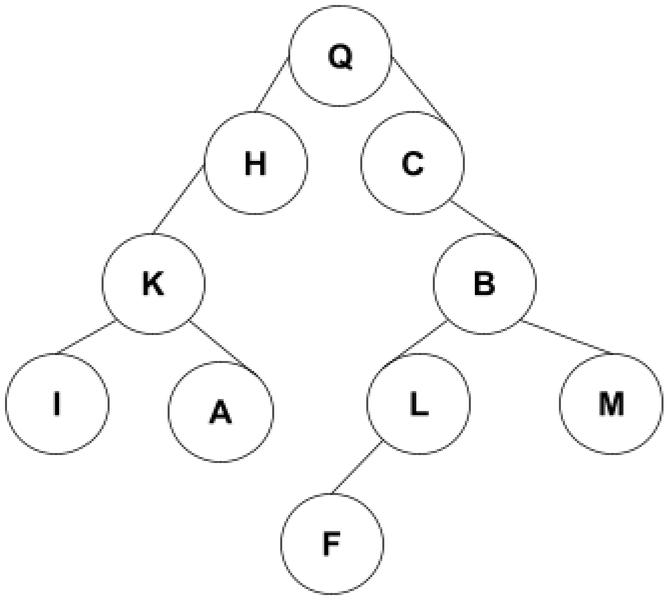
4-5) level-order: 21,18,27,7,25,30,19,26,35

4-6) NO, all children to the left of a parent are less than the parent, and all children to the right of the parent are greater than the parent, hence it is a binary search tree, but 25 is out of place, so its not.

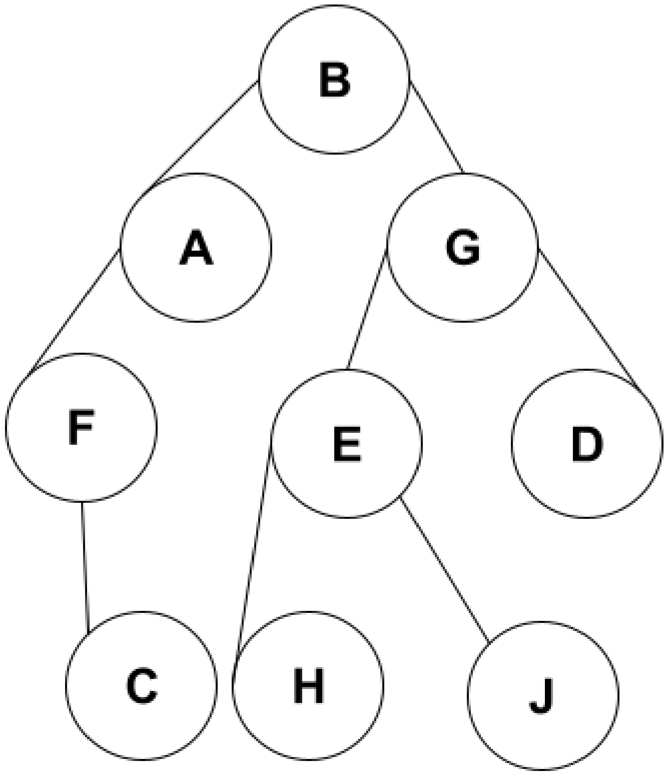
4-7) Balanced, because the height of the is only 1 minus the height of the children ahead of it in the other branches, so it’s not unbalanced according to height.

Problem 5: Tree traversal puzzles

5-1)

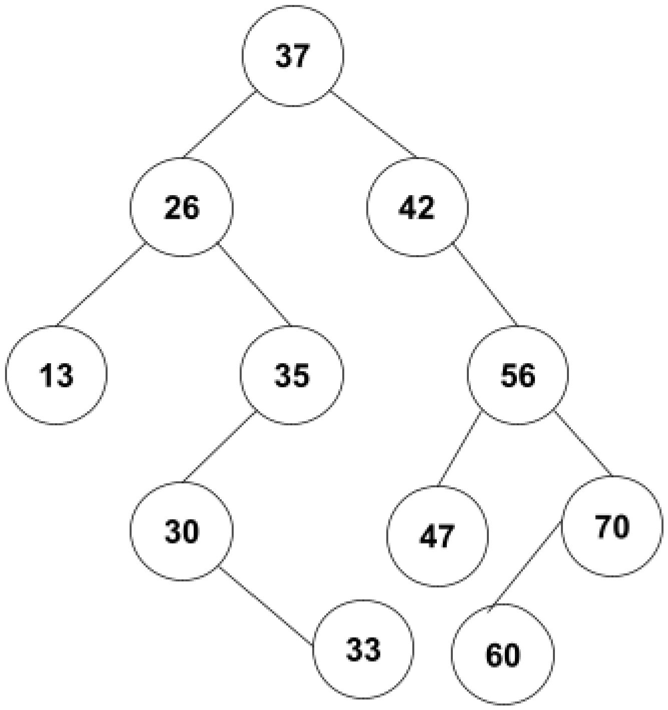


5-2)

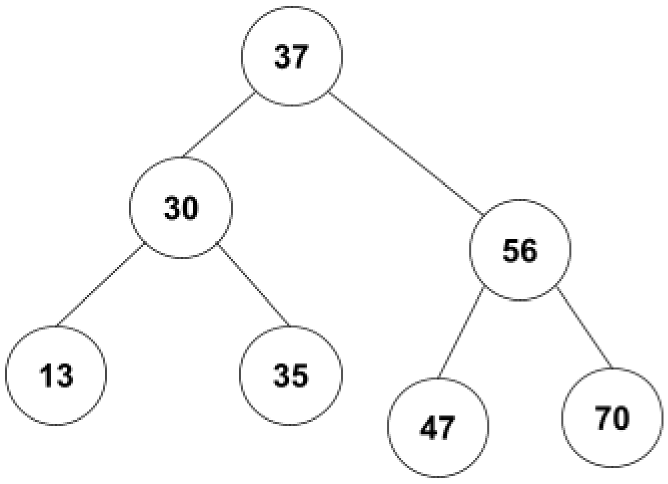


Problem 6: Binary search trees

6-1)



6-2)


37
30
56
13
35
47
70