Jake Wiseberg
Writing Assignment 2

1.

The Big Omega notation defines the shortest run time of an algorithm, while Big-Theta notation accounts for the worst-case runtime. So, by defining $f(n) = O(g(n))$ and $t(n) = O(g(n))$, we are deciding that $f(n)$ and $t(n)$ are the worse runtime of the function $g(n)$. These definitions would make $f(n) = \Omega(t(n))$, true. Since both $f(n)$ and $t(n)$ are defined using the same function $g(n)$, all of their properties would hold the same, the maximum run time and the least run time would be equal, making the statement $f(n) = \Omega(t(n))$ true.

2.

```
//reverses linked list using constant memory space
  public void reverse() {
    Node curr = head;
    Node next = head.getNext();
    Node prev = null;

    //Base case, if it is the last note, set it to be the head
    if (curr.getNext() == null) {
      head = curr;
      curr.setNext(prev);
    }

    Node next1 = curr.getNext();
    curr.setNext(prev);
    reverse();
  }
```

3.

```
public static int postFixExpr(String expr) {
    char curr;
    Stack<Integer> eval = new Stack<Integer>();
    for (int i=0; i<expr.length(); i++) {
      curr = expr.charAt(i);
      if (curr == '+')
        eval.push((int)(eval.peek())+(int)(expr.charAt(i+1)));
      else if (curr == '-')
        eval.push((int)(eval.peek())-(int)(expr.charAt(i+1)));
      else if (curr == '*')
        eval.push((int)(eval.peek())*(int)(expr.charAt(i+1)));
      else if (curr == '/')
        eval.push((int)(eval.peek())/(int)(expr.charAt(i+1)));
    }
    return eval.peek();
  }
```

4.

a) A perfectly balanced binary tree could have a height of 3.

Jake Wiseberg
Writing Assignment 2

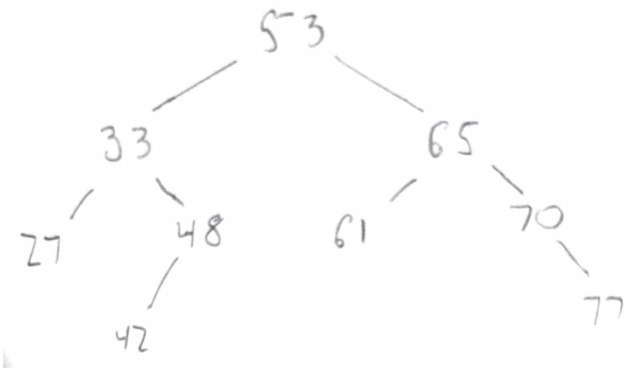
b) With a tree that has a height of 3, there would be 2^n leaf nodes, so there would be 8 leaf nodes.
c) The tree would have 7 internal nodes. The formula would be (2^n)-1.
d) It would only be possible to arrange n nodes into a perfectly balanced binary tree if there are (2*2^n)-1.

5.

```
public void insert(Node root, Node toInsert) {
  //checks to see if the toInsert value is less than the roots value
  if (toInsert.value < root.value) {
    //if value is less than and the left part isn't null, recursively call method using the left binary
search subtree
    if (root.left != null)
      insert(root.left, toInsert);
    //if value is less than and the left part is null, then insert
    else
      root.left = toInsert;
  }

  //checks to see if the toInsert value is greater than the roots value
  else if (toInsertvalue > root.value) {
    //if value is greater than and the right part isn't null, recursively call method using the right
binary search subtree
    if (root.right != null)
      insert(root.right, toInsert);
    // if the value is less than and the right part is null, then insert
    else
      node.right = toInsert;
  }

  //using an else if instead of else prevents the opportunity for doubly inserting values which
would break the binary search trees definition
}

//note, changed 'Node new' to 'Node toInsert' because new is a keyword for java and it wouldn't
compile
```

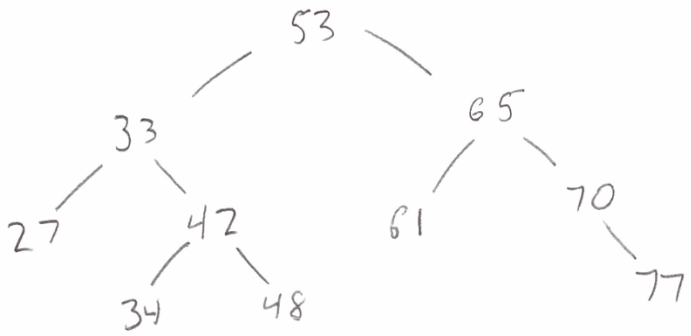
6.

a)

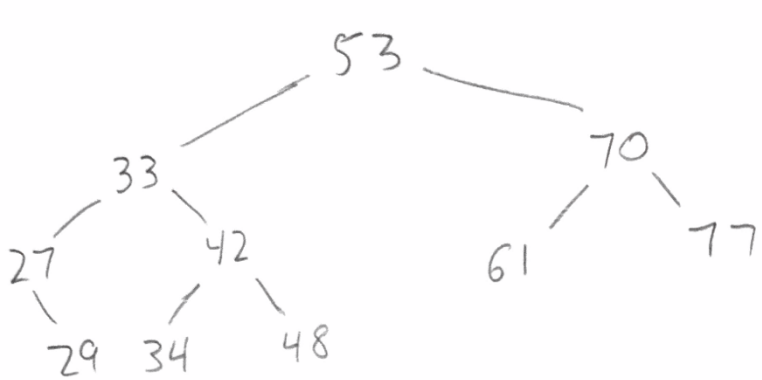


Jake Wiseberg
Writing Assignment 2

b) Inserted 34



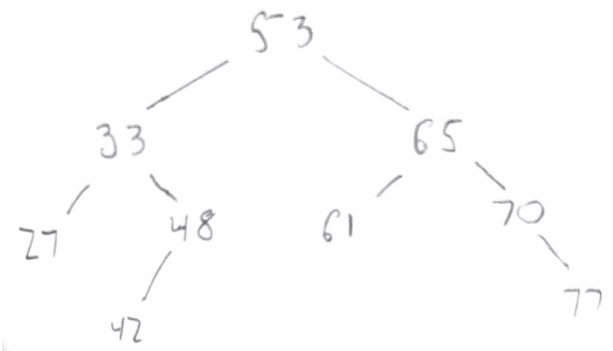
c) Deleted 65



7.

Yes, the order does matter. A simple example is if you take the original set and just insert it into an AVL tree backwards, the final trees compared are shown below.

Original Order



Backwards Order