Jennifer Manning

1.a. Show that the maximum number of nodes that can be stored in a BST is2^(h+1)−1.

Proof using Induction:

Base Case: h=0

N= 2^1 – 1 = 1

For a height 0, there can only be one node, the root node.

Assume: N_(n-1) = 2^(n-1+1) -1 = 2^n -1

Prove: N_n = 2^(n) -1 = 2^n *2 -1

N_n = 2^(n) + 2^(n) -1

N_n = 2*(2^(n)) -1 = 2^(n+1) = 2^(n+1) -1

b. What are the heights of your movie (hm) and actor (ha) trees? Given the heights hm and ha, how many nodes can be stored in each tree, how many nodes are actually stored in these trees?

The movie height tree, hm=16, and the actor height, ha=2. The number of nodes that ca be stored in each tree is 2^(h+1) -1. For hm, this would be2^17 -1 = 131071 movies; for ha, this would be 2^3 -1 = 7 actors. However there are only 5 actors and 39 movies stored

2. Suppose we have the following set of numbers {10, 11, 15, 19, 23, 78, 42, 56, 18, 13, 12, 38, 47}

a. It is impossible to make a perfectly balanced tree using 13 nodes, but here is a sequence for a balanced tree:

19, 12, 42, 11, 15, 38, 56, 10, 13, 18, 23, 47, 78

b. preorder traversal: 19, 12, 11, 10, 15, 13, 18, 42, 38, 23, 56, 47, 78

c.

```
        42
       /    \
     12     38
    /  \    /  \
  11   15  23  56
  /    / \      / \
10   13 18   47 78
```

3. A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a non-empty binary tree.

Let

n=number of nodes,

B=number of branches,

n0=number of nodes with no children,

n1=number of nodes with one child,

n2=number of nodes with two children.

All nodes come from a single branch except the root node, therefore:

B=n-1

B=n1 + 2*n2

n = n1 + 2*n2 +1

n = n0 + n1 +n2

n1 + 2*n2 +1 = n0 +n1 + n2

n0 = n2 +1

4. Suppose we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequences could not be in the sequence of nodes examined? Describe the logic you used to determine this in a short paragraph. (CLARIFICATION: for any given sequence (...a,b,c...) node b is the immediate child of a, and the immediate parent of c).

a. 2, 252, 401, 398, 330, 344, 397, 363
b. 924, 220, 911, 244, 898, 258, 362, 363
c. 925, 202, 911, 240, 912, 245, 363
d. 2, 399, 387, 219, 266, 382, 381, 278, 363
e. 935, 278, 347, 621, 299, 392, 358, 363

Sequence c and e are not valid. In the case of sequence c, that is because 912>911. The third node down in the tree is 911, which is too large so it branches to a smaller area of the tree, meaning that all nodes down that branch must be smaller than 911. However, two nodes later, 912 comes up which would not be placed in that branch. Sequence E has a similar problem: 299<347. The third node, 347, branches larger, however a few nodes later it branches to 299, which is a smaller branch.

5. We can sort a given set of n numbers by first building a binary search tree containing these numbers, with repeated insertions, and then printing the numbers in an in-order fashion. What are the best case and worst case big-Oh running times for this sorting algorithm?

The best case for this would be O(NlogN). This would be when there is a perfectly balanced tree so that every time another node is to be inserted, the worst running time would be O(LogN), or the height of the tree. Then for each node, there are logN nodes to iterate through, causing there to be overal N*logN iterations. For all cases, the printing would be O(N), however, that is outshadowed by the inserting which is the slower of the processes. The worst case would be O(N^2). An example of a case that would cause this situation is a linear tree, that is, one where there are either only right nodes or only left nodes. Therefore for each nodethat is to be inserted, they must iterate through n-1 nodes. This results in N*(N-1) iterations, or running time of O(N^2).