Name: Sample Solution

UWNetID: ______________________

Lecture Section: A

CSE 332 Winter 2015: Midterm Exam

(closed book, closed notes, no calculators)

Instructions: Read the directions for each question carefully before answering. We will give partial credit based on the work you **write down**, so show your work! Use only the data structures and algorithms we have discussed in class so far.

You are forbidden to communicate with anyone about the exam until after 1:30pm.

Note: For questions where you are drawing pictures, please circle your final answer.

Good Luck!

Total: 100 points. Time: 50 minutes.

Question	Max Points	Score
1	18	
2	8	
3	10	
4	8	
5	10	
6	9	
7	11	
8	16	
9	10	
Total	100	

Name: ____________________________________

1. (18 pts) Big-Oh

(2 pts each) For each of the operations/functions given below, indicate the tightest bound possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **Your answer should be as "tight" and "simple" as possible.** For questions that ask about running time of operations, assume that the most efficient implementation is used. For array-based structures, assume that the underlying array is large enough. For questions about hash tables, assume that no values have been deleted (lazily or otherwise).

You do not need to explain your answer.

a) *Pop in a* ***stack*** *containing N elements implemented using an array (worst case)* ___O(1)___

b) *Find in an* ***open addressing hash table*** *containing N elements where linear probing is used to resolve collisions (worst case). Tablesize =* N^2 ___O(N)___

c) *Merging two* ***binary min heaps*** *containing N elements each. (worst case)* ___O(N)___

d) *Determining what the 10 largest items are in an* ***open addressing hash table*** *containing N elements where quadratic probing is used to resolve collision (worst case). Tablesize = 2*N.* ___O(N)___

e) $T(N) = T(N/2) + 100$ ___O(log N)___

f) $f(N) = \log \log (N + N) + \log^2 N$ ___$O(\log^2 N)$___

g) *Insert in a* ***separate chaining hash table*** *containing N elements where each bucket points to a* sorted *linked list (worst case) Tablesize =* N^2 ___O(N)___

h) $f(N) = N \log^2 N + N^2 \log N$ ___$O(N^2 \log N)$___

i) decreaseKey *(k, amount) on a* ***binary min heap*** *containing N elements. Assume you have a reference to the key k that should be decreased. (worst case)* ___O(log N)___

Name: ______________________________

2. (8 pts) Big-Oh and Run Time Analysis: Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable n. Your answer should be as "tight" and "simple" as possible. ***Showing your work is not required***

Runtime:

```
I.  void happy (int n, int sum) {
        int k = 1;
        while (k < n) {
           for (int i = 0; i < k; i++) {
              sum++;
           }
           k++;
        }
        for (int j = n; j > 0; j--) {
              sum++;
        }
    }
```

$O(n^2)$

```
II.  int smiley (int n) {
        if (n < 5)
           return n * n;
        else {
           for (int i = 0; i < 10,000; i++) {
             print i
           }
           return smiley (n / 2);
        }
     }
```

$O(\log n)$

```
III.  void sunny (int n, int sum) {
        for (int i = 1; i < n * n; i++) {
           if (sum > 10) {
              for (int j = 0; j < n; j++) {
                 sum++;
              }
           } else {
              sum++;
           }
        }
     }
```

$O(n^3)$

```
IV.  void funny (int n, int sum) {
        for (int i = 0; i < n * n; i++) {
           if (i % 10 == 0) {
              for (int j = 0; j < i; j++) {
                 sum++
              }
           }
        }
     }
```

$O(n^4)$

Name: ____

3. (10 pts) Big-O, Big Ω, Big Θ

(2 pts each) For parts (a) – (e) circle **ALL** of the items that are TRUE. You do not need to show any work or give an explanation.

a) $N^2 + N^2 \log N$ is:

$\Omega(N^2)$ $O(N^2)$ $\Theta(N^2)$ None of these

b) $N \log N + \log(\log N) + 300$ is:

$\Omega(\log N)$ $O(\log N)$ $\Theta(\log N)$ None of these

c) $N^2 \log N + N^4$ is:

$\Omega(N^3)$ $O(N^5)$ $\Theta(N^4)$ None of these

d) $N \log^2 N + N^2 \log N$ is:

$\Omega(N^2)$ $O(N^2)$ $\Theta(N^2 \log N)$ None of these

e) If f(n) is O(g(n)) and g(n) is O(h(n)), which of the following is correct? (**circle all that are true**)

i. f(n) * g(n) is O(f(n) * g(n))

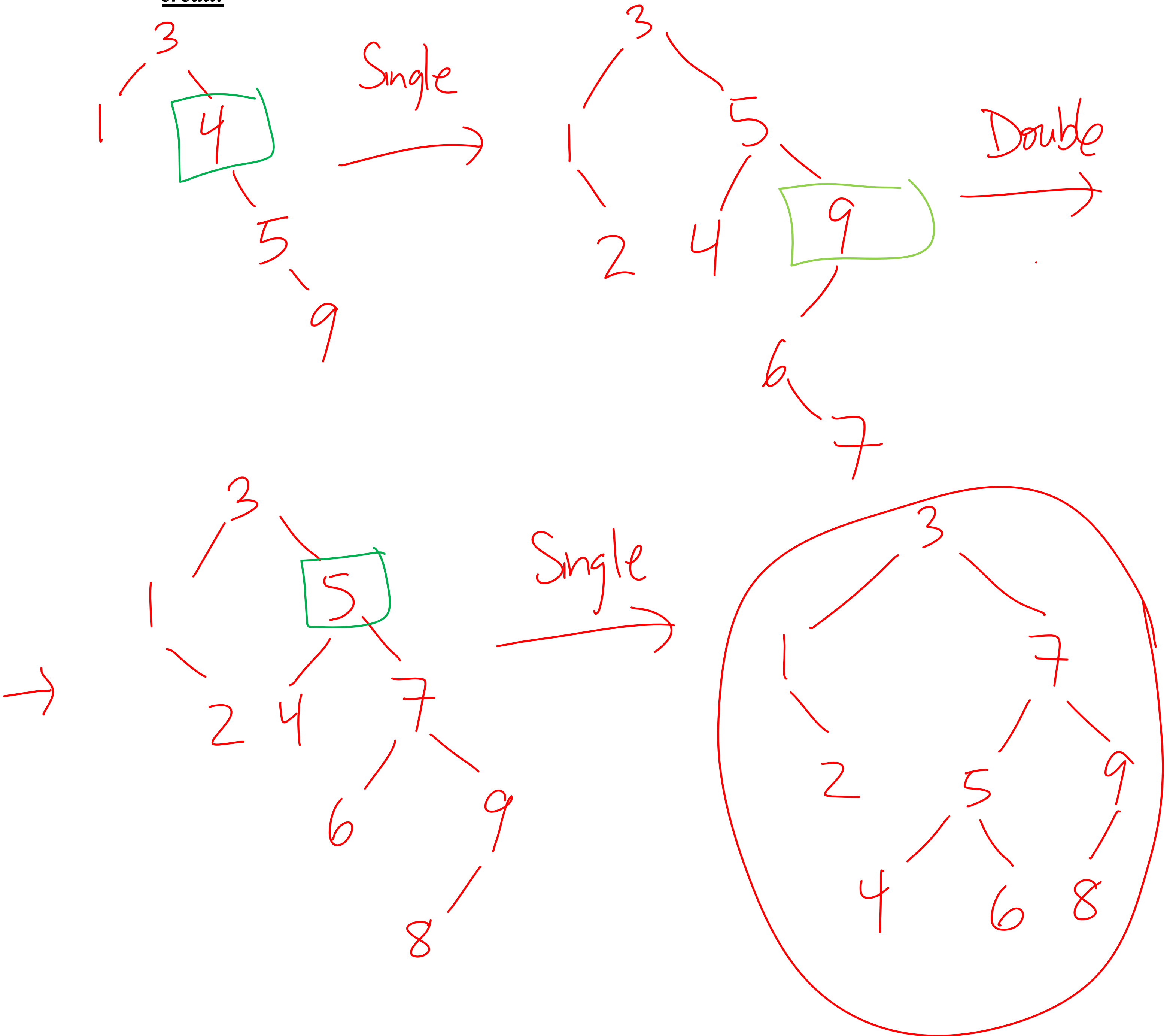
ii. f(n) + g(n) is O(min(g(n), h(n)))

iii. f(n) is O(h(n))

iv. h(n) is O(f(n))

v. none of the above

4. (8 pts) Draw the AVL tree that results from <u>inserting the keys 3, 1, 4, 5, 9, 2, 6, 7, 8 in that order</u> into an <u>initially empty AVL tree</u>. You are only required to show the final tree, although if you draw intermediate trees, ***<u>please circle your final result for ANY credit.</u>***



Name: ____________________________________

5. (10 pts) Data Structure Choice

Select the best (in terms of run-time) data structure to use in the following scenarios and **give a one sentence description of why you chose it**.

Choose from the following data structures (each of which may be used more than once):

Move-to-front List (as in project 2), Stack, Queue, AVL Tree, Heap, Hash Table, B-tree

a) Selecting the order of messages to send in a network when some messages require a faster delivery than others. New messages are always coming in.

Heap – Messages have a notion of priority.

b) Tracking the orders placed at a counter in a sandwich shop.

Queue – customer orders should be handled in a FIFO manner.

c) Tracking the roster of Seahawks players. Individual player records are occasionally requested by the player's name. Requesting the entire list of players sorted by player names is the **most** common operation. New players are added to the roster fairly often.

AVL tree was expected answer because returning a list of players in sorted order is best supported using an inorder traversal in O(N). Some folks seemed to think that B-tree would also be appropriate. We did not intend the number of players on the Seahawks roster to be so big as to require using lots of memory but we accepted B-trees as well. We did not talk about it but B-trees can also support listing all elements in sorted order efficiently O(N).

d) A company has a huge amount of data stored on external servers. They have even more data to add and will be performing many insert operations, which they want to be fast.

B-tree – Best for inserting data when it must be stored on external disks.

e) A company has a large amount of data that is *not comparable*. They want to use a data structure that gives them the fastest possible find operation. **Should they use an AVL tree or a Move-to-front List?**

Move-to-front – You cannot use AVL if data is not comparable.

Name: ______________________________

6. (9 pts) Hashing

Assuming inputs are all 3-digit numbers and d0 is the digit in the rightmost position. (e.g. the three digit number 456 has these digits: d2=4, d1=5, d0=6), you are given 3 possible hash functions:

h1(x) = d2 + d1 + d0
h2(x) = d2 + d1
h3(x) = d2

a) For a table of size 10 and the following 6 inputs, rank the 3 hash functions in order from best (least collisions) to worst (most collisions). Assume **separate chaining** and for the purposes of this question, take "number of collisions" to mean "number of items not in a bucket by themselves".

160

610

345

254

532

449

Best: h3   2nd Best: h1   Worst: h2

b) Give the **load factor** if each of the hash functions above is used on the inputs given above on a hash table of size 10:

h1: 3/5   h2: 3/5   h3: 3/5

c) Rank the hash functions again by the number of expected collisions for a **random** set of inputs:

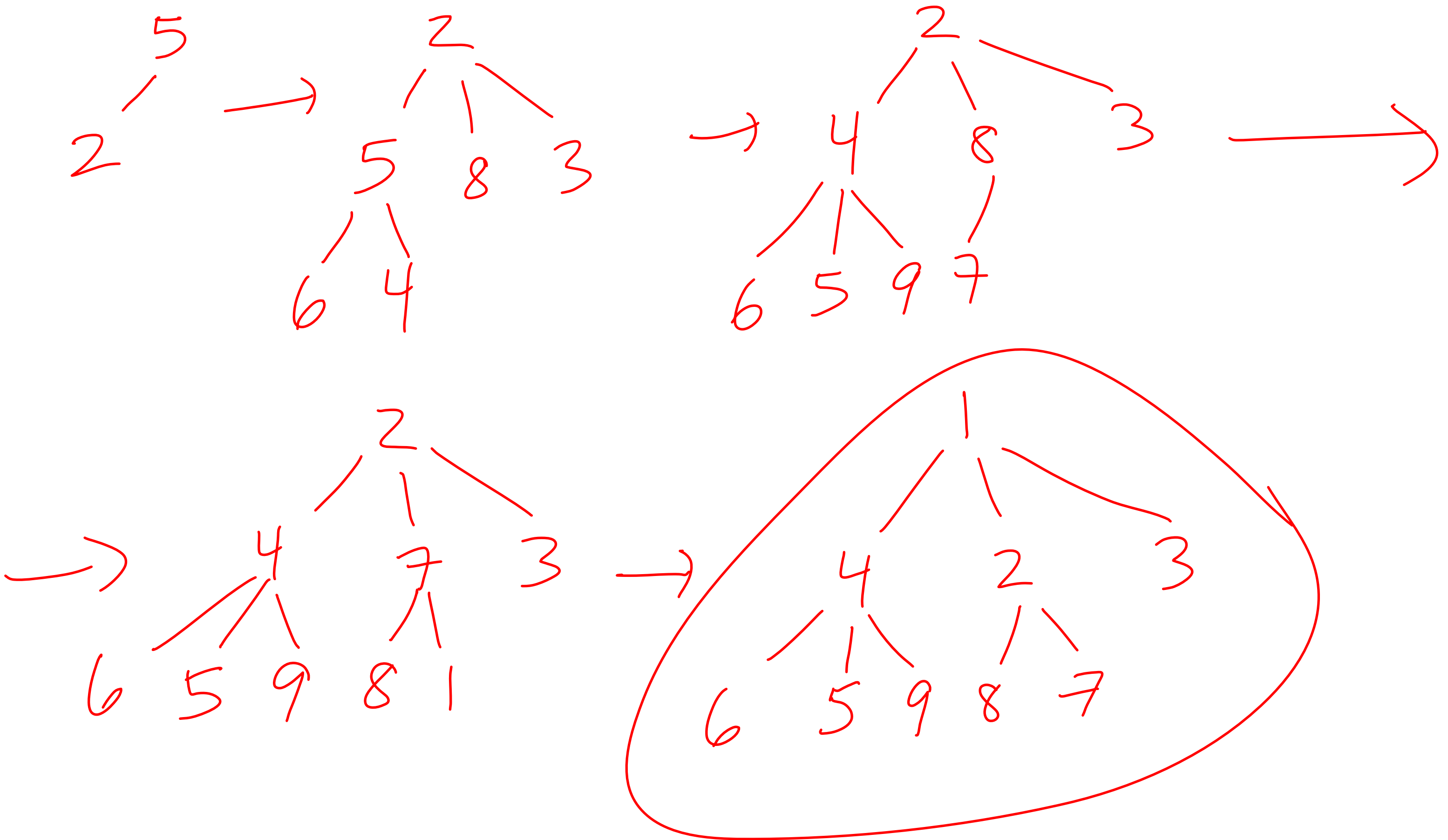
Best: h1   2nd Best: h2   Worst: h3

(If Tablesize = 10, then
Best = h3, 2nd Best = h2, Worst = h1)

Name: ____________________________________

7. (11 pts) Min Heaps -

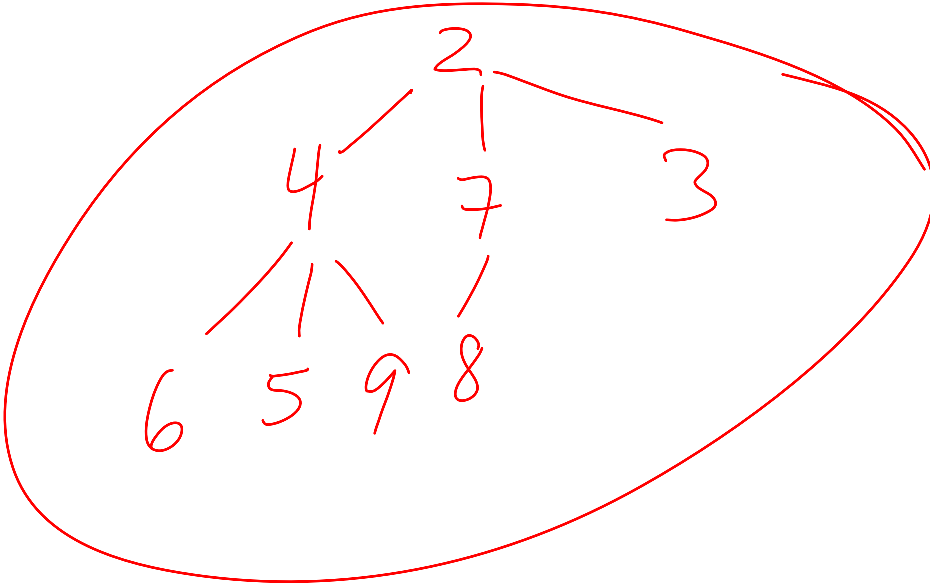
a) (6 pts) As discussed on homework 2, a three-heap with n elements can be stored in an array A, where A[0] contains the root of the tree. Draw the **three**-heap that results from inserting 5, 2, 8, 3, 6, 4, 9, 7, 1 in that order into an **initially empty three-heap**. You do not need to show the array representation of the heap. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***



Name: ______________________________

7. Min Heaps (continued)

b) (2 pts) Draw the result of doing 1 deletemin on the heap you created in part a. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***



c) (3 pts) **Assuming that elements are placed in the array starting at location A[0]**, give expressions to calculate the left, middle, and right children of the element stored in A[i]:

Left child: $(3*i) + 1$

Middle child: $(3*i) + 2$

Right child: $(3*i) + 3$

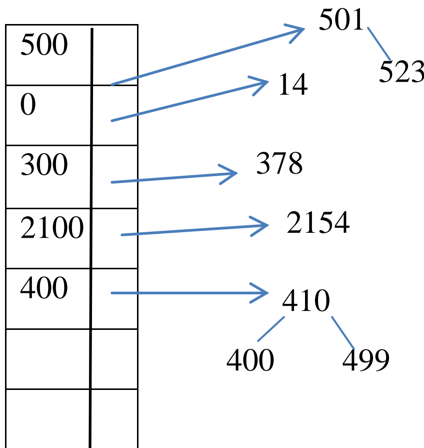
Name: ______________________________________

8. (16 points) Data Structure Analysis

Your co-worker has created a new data structure for storing positive integers and needs your help evaluating it. The structure consists of an unsorted array, where each location in the array points to an AVL tree containing at most 100 integers, inserted as follows.

Integers will be placed in a bucket with other integers containing the same digits in the "hundred and above" positions (i.e. ignoring digits in the 10's place and the 1's place). When an integer is inserted, first it is **divided** by 100 (not mod!) and then an appropriate array location is searched for. If an appropriate array location exists, the integer is inserted into the AVL tree linked there. If an appropriate array location does not yet exist then one is added at the end of the list and the integer is inserted as the first value in the AVL tree linked there.

Below is an example of a structure that would be created if the integers: 501, 14, 378, 2154, 410, 523, 400, and 499 had been inserted in that order. In this example, the first bucket would hold integers in the range 500-599, the second in the range 0-99, the third in the range 300-399, the fourth in the range 2100-2199, and the fifth in range 400-499.



Each location in the array will contain two fields: an integer representing the smallest value that could possibly be stored in this AVL tree and a pointer to an AVL tree. You may assume that no duplicate values will ever be inserted and that the original array will always be large enough.

a) What would be the worst case big-O run-time of a **find** operation on this data structure? **Explain your answer briefly.**

O(N) – Worst case is each integer is in its own bucket (so N buckets), and the bucket you need is the last one in the array (this is basically an unsorted array). This scenario has just one element per bucket, but even a maximally full AVL tree has find of O(log 100), not O(log N) as many people said, so overall O(N+ log 100) → O(N)

b) What would be the worst case big-O run-time of an **insert** operation on this data structure? **Explain your answer briefly.**

O(N) – Same as above to find the place to put the integer. If no appropriate bucket is found, constant time to create new bucket at end and add item into the AVL tree.

8. Data Structure Analysis (continued)
Your co-worker suggests a couple of modifications to the original data structure:

Option 1: Instead of an unsorted array, use **a move-to-front list** created with linked list nodes (as in Project 2).

c) Worst case big-O run-time of a **find** operation? **Explain your answer briefly.**

O(N) – Still an unsorted list. Worst case is every integer in its own bucket and you are looking for the least-recently used bucket or one that is not present do must traverse the whole list containing N buckets. Find on these AVL trees is O(1).

d) Worst case big-O run-time of an **insert** operation? **Explain your answer briefly.**

O(N) – Still an unsorted list. Worst case is every integer in its own bucket and you are looking for the least-recently used bucket or one that is not present. Have to first find the appropriate bucket or examine them all to determine you need to create a new one. If needed, you should create the new bucket at the front of the list, but you must search the entire list, O(N), to determine that you need to create it. Insert into an AVL tree containing a max of 100 integers is O(1).

Option 2: Instead of an unsorted array, use **a sorted array.**

e) Worst case big-O run-time of a **find** operation? **Explain your answer briefly.**

O(log N) – Worst case is every integer in its own bucket, so N buckets. Can do a binary search on sorted array to find correct bucket. Then search on these AVL trees is constant time.

f) Worst case big-O run-time of an **insert** operation? **Explain your answer briefly.**

O(N) – Worst case is every integer in its own bucket, so N buckets. Can do a binary search on sorted array to find correct bucket. In worst case the bucket we want is not present and we need to insert it at the front and shift all buckets down O(N). Then insert into these AVL trees is constant time.

You propose that your co-worker *forget this idea completely* and just use a **separate chaining hash table** of size N, where each bucket points to an **AVL tree**.

g) Worst case big-O run-time of a **find** operation? **Explain your answer briefly.**

O(log N) – Worst case is every integer in the same hash table bucket. Finding correct hash table bucket is O(1). Then search on an AVL tree containing N items is O(log N).

h) Worst case big-O run-time of an **insert** operation? **Explain your answer briefly.**

O(log N) – Worst case is every integer in the same hash table bucket. Finding correct hash table bucket is O(1). Then insert on an AVL tree containing N items is O(log N).

Name: ______________________________________

9. (8 pts) B-trees

a) (2pts) In the B-Tree shown below, please **write in the appropriate values for the interior nodes**.

b) (2 pts) Based on the picture below, what are the values for M and L?

M = 3 L = 3

c) (4 pts) Starting with the B-tree shown below, insert **61**. Draw and circle the resulting tree (*including values for interior nodes*) below. Use the method for insertion described in lecture and used on homework.