Name: ____**Sample Solution**_____

UWNetID: _____

CSE 332 Winter 2019: 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.

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

Good Luck!

VERSION I

Total: 100 points. Time: 60 minutes.

Question	Max Points	Score
1	18	Big-O
2	16	Code Analysis
3	9	Theta, etc
4	8	Heaps
5	6	AVL
6	9	Find Recurrence
7	10	Solve Recurrence
8	8	BTrees
9	5	BTrees
10	7	AVL
11	4	Heaps
Total	100	

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.

You do not need to explain your answer.

a) *Inserting an element into a **binary min heap** containing N elements. (**BEST** case)* O(1)

b) *f(N)* = $200 \log (N^2) + 5 (\log N)^2$ $O(\log^2 N)$

c) *Finding and removing the largest item in a **binary search tree** containing N elements. (**BEST** case)* O(1)

d) $T(N) = T(N - 1) + N$ $O(N^2)$

e) *f(N)* = $\log_4(N^2) + \log_4 (2^N)$ O(N)

f) *Printing all values less than a given value in an **AVL tree** containing N integers. (worst case)* O(N)

g) $T(N) = T(N/2) + 17$ $O(\log N)$

h) *Push in a **stack** containing N elements implemented using singly-linked list nodes (worst case)* O(1)

i) *Using* deleteMin*() to remove the* $\lceil \log N \rceil$ *smallest elements in a **binary min heap** containing N elements.* $O(\log^2 N)$

2. **(16 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.***

I.
```
void sledding(int n) {
  int sum = 0;
    for (int i = 1; i < Math.pow(2, n); i *= 2) {
      if (i < 10000000) {
        for (int j = 0; j < Math.pow(2, i); j++) {
          sum++;
        }
      } else {
          for (int k = 1; k < n; k *= 2) {
            sum--;
          }
      }
    }
  print("done!");
}
```

Runtime: $O(n \log n)$

II.
```
int snow(int n, int ball) {
    if (n < 200) {
       return n / 2;
    } else if (n < 3000) {
       for (int i = 0; i < n * n; i++) {
          ball++;
       }
       return ball;
    }
    ball += snow(n / 2, ball);
    return n * snow(n / 2, ball);
}
```

$O(n)$

III.
```
void snowshoe(int n, int sum) {
    int j = 0;
    while (j < n) {
       for (int i = 0; i < n; i++) {
          sum++;
       }
       for (int k = n; k > 0; k = k/2) {
          sum++;
       }
       j++;
    }
}
```

$O(n^2)$

IV.
```
void skiing(int n) {
  for (int i = 0; i < n; i = i + n/2) {
    for (int j = n; j > 0; j = j - n * 2) {
      print("downhill");
    }
  }
}
```

$O(1)$

3. (9 pts) Big-O, Big Ω, Big Θ

(3 pts each) For each of the following statement, indicate whether it is **always true**, **sometimes true**, or **never true**. You do not need to show any work or give an explanation. You should assume that the domain and co-domain of all functions in this problem are the natural numbers.

a) $f(n)$ is O ($(f(n))^2$)

Always True Sometimes True Never True

b) $f(n) + g(n)$ *is* Θ (max{ $f(n), g(n)$ })

Always True Sometimes True Never True

c) $f(n) * n$ is Θ ($(f(n))^2$)

Always True Sometimes True Never True

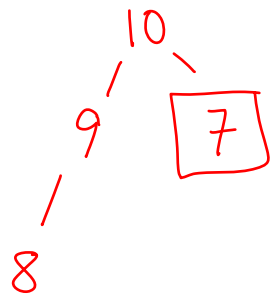
4. (8 pts) Binary MAX Heaps

Suppose you are given a binary MAX heap. This is identical to the binary min heap discussed in lecture except that the value stored at the root is the largest in the heap. For this problem we will assume no duplicate values may exist in the heap.

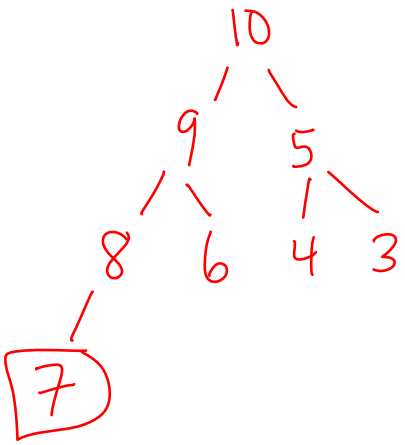
a. At what depths in the tree (distance from the root) can the fourth-largest value occur? **List all depths where the value might occur.**

1, 2, 3

b. Draw an example of a binary MAX heap tree of integers (do not draw the array) where the fourth-largest value occurs at the <u>minimal</u> possible depth. Be sure you **draw an integer for each node in the tree** and that you draw **a box around the 4th largest value.**



c. Draw an example of a binary MAX heap tree of integers (do not draw the array) where the fourth-largest value occurs at the <u>maximal</u> possible depth. Be sure you **draw an integer for each node in the tree** and that you draw **a box around the 4th largest value.**

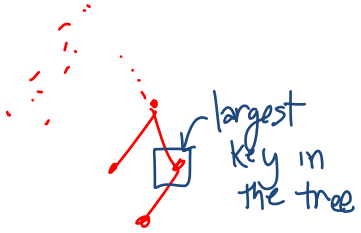


5) (6 pts) AVL Trees
You are given an AVL-Tree of height 5 and you are told what the largest key in the tree is. If you are doing a **pre-order traversal** of the tree, ***how many nodes in the tree will you visit before you encounter the largest key***? Give your answer as a ***single number*** (not a mathematical expression) but **explain your answer briefly**.

a) In the BEST case, what is the minimum number of nodes you would visit (include the largest key in your count of nodes)? **Explain your answer briefly**.

The minimum number of nodes in an AVL tree of height 5 is 20. However the right side of the tree could look like this:

$S(2) = 1 + 2 + 1 = 4$
$S(3) = 1 + 4 + 2 = 7$
$S(4) = 1 + 7 + 4 = 12$
$S(5) = 1 + 12 + 7 = 20$



Thus with a pre-order traversal you could stop as soon as you hit the largest key + not visit its left child. So a total of 19 nodes visited.

b) In the WORST case, what is the maximum number of nodes you would visit? (include the largest key in your count of nodes) **Explain your answer briefly**.

The maximum number of nodes in an AVL tree of height 5 would be in a perfect tree.

$$\sum_{i=0}^{5} 2^i = 2^6 - 1 = 64 - 1 = 63 \text{ nodes}$$

In the worst case you would need to visit all of these nodes.

6. (9 pts) Recurrences

Give a base case and a recurrence for the runtime of the following function. Use variables appropriately for constants (e.g. c_1, c_2, etc.) in your recurrence (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE** this recurrence.

```
int cake(int n) {
    if (n < 10) {
        return 1000;
    } else if (cake(n/4) < 100) {
        for (int i = 0; i < n * n; i++) {
            print i;
        }
    }
    for (int i = 0; i < log(n); i++) {
        print("Cakes have layers!");
    }
    return cake(n/3) * n + 10 * cake(n/3);
}
```

T(n) = c_0 For n < 10

T(n) = $T(\frac{n}{4}) + c_1 \cdot \log(n) + 2 \cdot T(\frac{n}{3}) + c_2$ For n >= 10

Yipee!!!! YOU DO NOT NEED TO SOLVE *this* recurrence…

7. (10 pts) Solving Recurrences

Suppose that the running time of an algorithm satisfies the recurrence relationship:

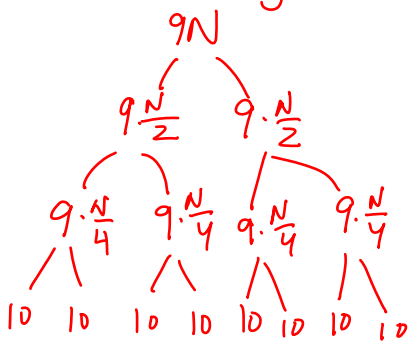
$T(1) = 10.$

and

$T(N) = 2 * T(N/2) + 9N$ for integers $N > 1$

Find the closed form for T(N). **You may assume that N is a power of 2.** Your answer should *not* be in Big-Oh notation – show the relevant ***exact*** constants and bases of logarithms in your answer (e.g. do NOT use "c_1, c_2" in your answer). You should not have any summation symbols in your answer. The list of summations on the last page of the exam may be useful. ***You must show your work to receive any credit***.

$N=8 \quad h = \log_2 N$



$$T(N) = 2 \cdot T\left(\frac{N}{2}\right) + 9 \cdot N$$

$$= 9 \cdot N \cdot \sum_{i=0}^{\log_2 N - 1} 2^i \cdot \frac{1}{2^i} + 10 \cdot N$$

$$= 9 \cdot N \cdot \sum_{i=0}^{\log_2 N - 1} 1 + 10N$$

$$= 9N \cdot \log_2 N + 10N$$

$$T(N) = 2 \cdot T\left(\frac{N}{2}\right) + 9 \cdot N$$

$$= 2\left[2 \cdot T\left(\frac{N}{4}\right) + 9 \cdot \frac{N}{2}\right] + 9N$$

$$= 2\left[2 \cdot \left[2 \cdot T\left(\frac{N}{8}\right) + 9 \cdot \frac{N}{4}\right] + 9 \cdot \frac{N}{2}\right] + 9 \cdot N$$

$$= 2^3 \cdot T\left(\frac{N}{8}\right) + 2^2 \cdot 9 \cdot \frac{N}{4} + 2 \cdot 9\frac{N}{2} + 9N$$

$$= 2^3 \cdot T\left(\frac{N}{2^3}\right) + 9N + 9 \cdot N + 9 \cdot N$$

$$= 2^3 \cdot T\left(\frac{N}{2^3}\right) + 3 \cdot 9N$$

$$= 2^k \cdot T\left(\frac{N}{2^k}\right) + k \cdot 9N$$

$$= 2^{\log_2 N} \cdot T(1) + \log_2 N \cdot 9N$$

$$= N \cdot 10 + 9N \cdot \log_2 N$$

$\frac{N}{2^k} = 1$ when

$N = 2^k$

$k = \log_2 N$

8. (8 pts total) B-Trees:
a) (6 pts) Given M=7 and L=11, what is the minimum and maximum number of **leaf nodes** in a B-Tree (as defined in lecture and in Weiss) of height 5? **Give a single number or a single number with an exponent for your answers, not a formula. For any credit, explain briefly how you got your answers.**

Minimum number of leaf nodes: 512

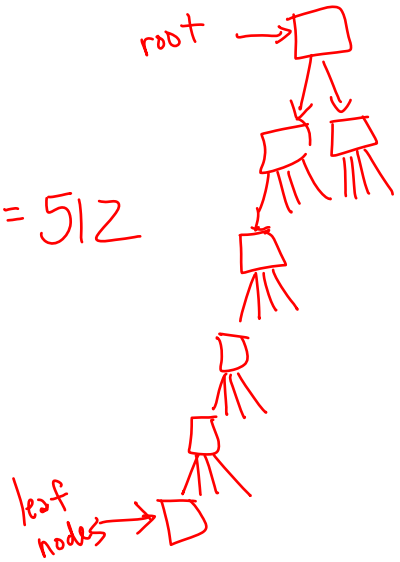
Maximum number of leaf nodes: 7^5

Explanation:

Minimum # of leaf nodes = $2 \cdot \left(\lceil \frac{M}{2} \rceil\right)^{h-1} = 2 \cdot (4)^4 = 2 \cdot 2^8 = 2^9 = 512$

min # pointers in root node

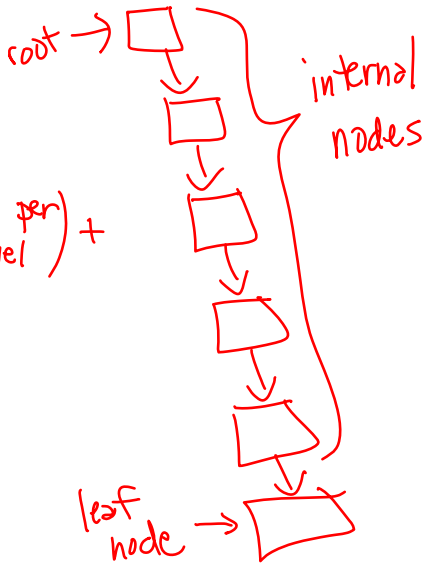
min # pointers in internal nodes



Max # of leaf nodes = $M^h = 7^5$
(Max # of pointers on each level is 7, 5 levels)

b) (2 pts) Assuming a well-constructed B-tree, with nothing cached in memory, what is the minimum number of disk blocks we would need to examine on a **find** operation **in the B-Tree described above**? **Explain your answer briefly.**

Assuming each node fits entirely on a single disk block, then we would need to examine at least 5 internal nodes (one per level) + 1 leaf node = 6 disk blocks total



9. (5 pts) B-trees - Give a tight big-O running time of the worst case of an **insert** operation on a **B-tree** (as described in lecture and in Weiss). Keep all factors of M, L, and N in your answer. Do not use any other variables in your answer. **Explain briefly how you got your answer.**

Binary Search on M ptrs at each level to get to the correct leaf node: $O(\underbrace{\log_2 M}_{\text{binarysearch}} \cdot \underbrace{\log_M N}_{\#\text{levels}})$

Binary Search + Insert (shift data) in leaf: $O(\underbrace{\log_2 L}_{\text{binary search}} + \underbrace{L}_{\text{shift data}})$

Split Leaf: $O(L)$

Split pointers in parent, all the way up to root: $O(\underbrace{M}_{\text{split pointers in interior node}} \cdot \underbrace{\log_M N}_{\#\text{levels}})$

$$O(\log_2 M \cdot \log_M N + \log_2 L + L + L + M \cdot \log_M N)$$

$$O(L + M \cdot \log_M N)$$

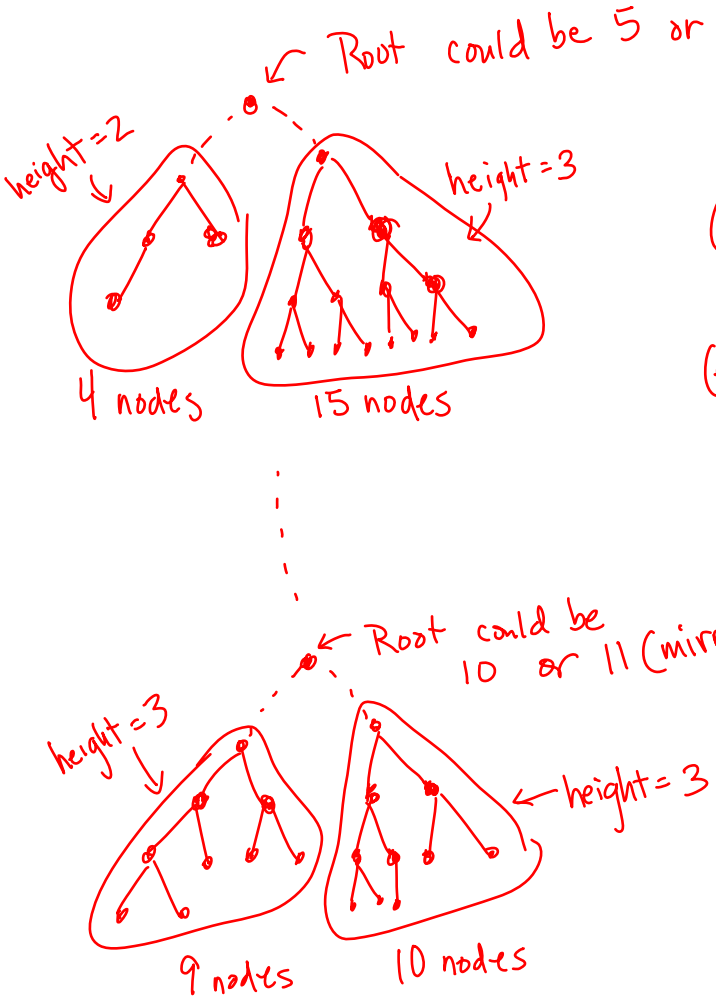
10. (7 pts) AVL Trees

We have an AVL tree that contains the integers 1, 2, 3, …20. We do not know what order the values were inserted into the tree. List all possible values that could appear as the root. **For any credit, briefly explain your answer.**

Several different AVL trees could have the values 1-20 and these values for the root. Here is one way to construct a set of such trees:

Possible Values for the root:

5, 6, ... 15, 16



Can move 1 node from right subtree into left subtree one at a time (e.g. building towards a perfect tree on the left subtree) + cover all the possible sizes of subtrees in between.

11. (4 pts) Heaps
You are given a valid array-backed D-heap that stores the smallest value in the heap at index 0. Your friend proposes an alternative implementation for deleteMin() that proceeds as follows:

1. Remove and return the element at index 0.
2. Define index 0 as "the hole".
3. Find the smallest value x out of all the immediate children of "the hole".
4. Move that value x into "the hole". Redefine "the hole" as the vacant index previously occupied by value x.
5. Repeat steps 3-4 until "the hole" has no children.

a. Given a valid D-heap, give a tight big-O running time of the worst case of this new implementation in terms of N and D. KEEP all D and N terms in your answer, do not simplify your answer by removing D terms.
Briefly explain your answer.

$O(D \cdot \log_D N)$

max # children of "the hole"

height of tree

At each level, must examine up to D children to find the smallest value, then move it up. Do this once per level, starting at the root, going down to a leaf node.

b. Would this algorithm be an acceptable implementation of deleteMin()?
Briefly explain why or why not.

No! It could leave holes in the tree - it would no longer be a complete tree.