Name: _____

UW NetID: _____

CSE 332 Summer 2019: Midterm Exam

(closed book, closed notes, no calculators)

Instructions: Read the directions for each question carefully. We can only give partial credit based on the work you write down, so show your work. If you are running out of time, take note of which questions are weighted more and budget your time accordingly.

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

Good Luck!

Total: 100 points. Time: 60 minutes.

Question	**Max Points**
1	18
2	10
3	8
4	12
5	8
6	14
7	12
8	10
9	8
Total	100

1 Big-Θ

[18 points total] Give a simplified Θ bound for each of these situations. For any array-based implementations assume the array is not full and still has empty space.

1. Worst-case time to delete an element in a sorted linked-list of size n

   1. $\Theta(n)$

2. Best-case time to find an element in a sorted array of size n

   2. $\Theta(1)$

3. Worst-case inserting a value into an AVL Tree of size n

   3. $\Theta(\log(n))$

4. Worst-case time to find a value in a Separate Chaining Hash Table using an AVL tree in each bucket

   4. $\Theta(\log(n))$

5. Memory required to store n numbers in an array-based implementation of a minheap

   5. $\Theta(n)$

6. $T(n) = \begin{cases} T(n-1)+n & \text{if } n \geq 1 \\ 7 & \text{otherwise} \end{cases}$

   6. $\Theta(n^2)$

7. $f(n) = 7\log(\log(n)) + 2\log^2(n)$

   7. $\Theta(\log^2(n))$

8. $T(n) = \begin{cases} 3T(n/3)+n & \text{if } n \geq 1 \\ 1 & \text{otherwise} \end{cases}$

   8. $\Theta(n\log(n))$

9. $f(n) = 1000n + 2^n + n^2$

   9. $\Theta(2^n)$

2 Run Time Analysis

[10 points total] Describe the best-case and worst-case running times for each of the following pseudocode functions in big-Oh notation in terms of the arbitrarily large variable n. Your answer should be as tight and simplified as possible. You can assume no execution will throw run-time exceptions, so any lookups to arr will return valid values.

```
int bar(int n, int[] arr) {
    int tot = 100;
    if (n > 1000) {
        for (int j = n; j > 0; j--) {
            tot += j \% 4;
        }
        for (int i = 0; i < n*n*n; i++) {
            tot += i;
            if (tot == arr[i]) {
                return i;
            }
        }
    } else {
        return 5;
    }
}
```

Worst-Case Runtime: $O(\overline{O(n^3)})$
Best-Case Runtime: $O(\underline{O(n)})$

```
void baz(int n, int[] arr) {
    for (int i = 1; i < n; i *= 2) {
        for (int j = 0; j < arr[i] % n; j++) {
            print(n)
        }
    }
}
```

Worst-Case Runtime: $\Theta(\overline{O(n\log(n))})$
Best-Case Runtime: $\Theta(\underline{O(\log(n))})$

3 O, Ω, Θ

[8 points total] For each of the following, circle ALL the correct descriptions of the function (if any).

1. $n^{3/2}$ is
   a. $O(n^2)$ b. $O(n \log n)$ c. $\Omega(n)$ d. $\Theta(n^2)$

2. 2^n is
   a. $\Theta(3^n)$ b. $O(3^n)$ c. $O(n^2)$ d. $\Omega(1)$

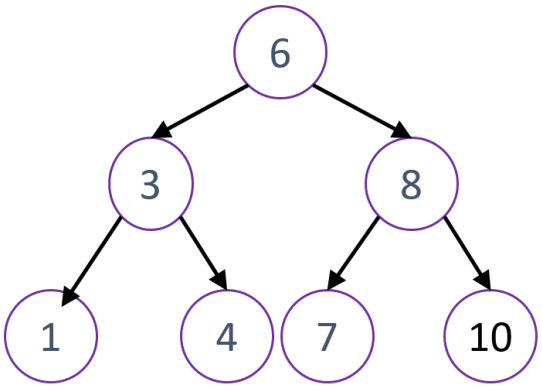
3. $\log_3(n)$ is
   a. $\Omega(\log(n))$ b. $O(\log(n))$ c. $\Theta(\log(n))$ d. $O(\log^2(n))$

4. $n \log(n)$ is
   a. $\Theta(n)$ b. $\Omega(n \log(\log(n)))$ c. $\Omega(n\sqrt{n})$ d. $O(n^2)$

4 AVL tree insertions

[12 points total] Insert the following items, in order, into an empty AVL tree. Circle your final tree. We strongly recommend showing intermediate trees, so we can better award partial credit if you make a mistake.

$$1, 8, 3, 4, 6, 7, 10$$



Imagine a new form of AVL insert: $insert(key, value, node)$ that takes, as a third parameter, a pointer to a node in the AVL tree. If the passed key already exists in the tree, the node parameter will be a pointer to the node with the old value. If the key is new, the pointer will be to a leaf node directly above where the new node should be placed in the tree. From this point, the rest of a normal AVL insert operation is performed. Assume you can traverse up the tree. What is the worst-case Θ asymptotic behavior for this new insert function? Briefly justify your answer.

Once the location where the insert into an AVL tree takes place is found, the addition of the new node is a constant-time operation. From there however the algorithm must travel back up the tree to find the nearest node of imbalance and perform a rotation. This search can travel the height of the tree again so the operation is, like the original insert, still a $\Theta(\log(n))$ operation.

5 Writing A Recurrence

[8 points total] Write a recurrence relation for the worst-case running time of the recursive code below. Use constants like c_1 and c_2 when describing the number of operations. You do not have to give an exact count. Note: You do not have to solve this relation, only put it in T(n) terms.

```
int recurr(int n){
    if (n > 7) {
        int sum = recurr(n-1);
        for (int j = 0; j < n*n; j++) {
            sum += j;
        }
        return recurr(n/2) * sum;
    } else {
        for (int i = 0; i < n; i++) {
            print(i);
        }
        return 0;
    }
}
```

$$T(n) = \begin{cases} c_1 & \text{if } n \leq 7 \\ T(n-1) + T(n/2) + c_2 n^2 & \text{otherwise} \end{cases}$$

Don't find a closed form of *this* recurrence.

6 Solving A Recurrence

[14 points total] Solve the recurrence relation below using the tree method. Give a tight Θ-bound that does not include any summations or recursion. Steps for the tree method are listed on the last page of this test, as well as several potentially useful summations. Feel free to check your answer using the master theorem, but you **must** calculate the closed form yourself first. You will not receive **ANY** credit if you do not show your work.

$$T(n) = \begin{cases} 4T(n/2) + 3n & \text{if } n > 1 \\ 5 & \text{otherwise} \end{cases}$$

Actual student work can vary here and does not have to conform so strictly to the tree method steps below.

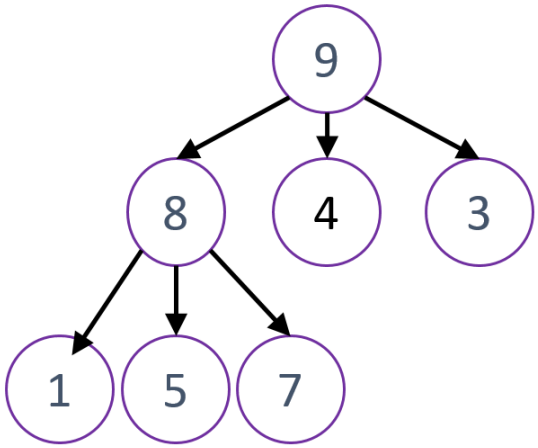
0. Draw a few levels of the tree. Tree will be a 4-ary tree
1. Let the root node be at level 0. Give a formula for the size of the input at level i. $\frac{n}{2^i}$
2. What is the number of nodes at level i? 4^i
3. What is the work done at the i^{th} recursive level? $3 \cdot 2^i n$
4. What is the last level of the tree? $\log n$
5. What is the work done at the base case? $5 \cdot 4^{\log n} = 5n^2$
6. Write an expression for the total work done. $\sum_{i=0}^{\log(n)-1} 3 \cdot 2^i n + 5n^2$
7. Simplify until you have a "closed form" (i.e. no summations or recursion). $\sum_{i=0}^{\log n - 1} 3 \cdot 2^i n + n^2 = n \sum_{i=0}^{\log n - 1} 2^i + 5n^2 = 3n(2^{\log n} - 1) + 5n^2 = \Theta(n^2)$

7 Heaps

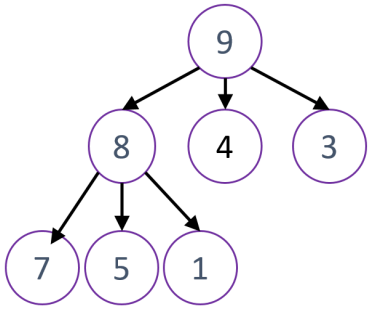
[12 points total]

1. Insert these items, in order, into a **max 3-heap** in tree form, and circle the final tree structure. Showing your intermediate steps will greatly increase your chance of partial credit..

$$1, 5, 4, 3, 7, 9, 8$$



2. Create a **max 3-heap** in tree form using Floyd's buildHeap and the same input data from part 1 (re-printed below). Circle the final tree structure.



3. What is the benefit to using Floyd's buildHeap versus creating a heap through successive insert calls? Compare the two methods using Θ bounds.

   Floyd's buildHeap guarantees a $\Theta(n)$ run time even in the worst case while calling insert n times is $\Theta(n \log n)$.

8 Heaps 2

[10 points total] Rob intends to propose a new data structure he calls a Perfect Heap. This is a binary min heap as discussed in class, with the added stipulation that the heap shape must be perfect. When insert is called and the total number of values in the structure will not result in a perfect tree, the new value is instead stored in an unsorted array. In the case the latest insert does allow the heap to take a perfect shape, the new value and all values stored in the unsorted array are added to the heap as leaf nodes, percolateDown is then called on the nodes in the heap from right to left, and the unsorted array is emptied.

1. What can the minimum length of the associated unsorted array be if the heap part of the Perfect Heap is height h?
   $2^{h+1} - 1$ One less than the size of the next level in the heap.

2. What is the best-case amortized behavior for the insert function if the heap part of the Perfect Heap is size n? Give as close of a O asymptotic bound as you can and briefly explain why.
   The best-case is n insertions that take $O(n)$ time total, which becomes $O(1)$ amortized for a single operation.

3. What is the worst-case behavior for the insert function if the heap part of the Perfect Heap is size n? Give as close of a O asymptotic bound as you can and briefly explain why.
   $O(n)$ This action is just buildHeap for a size 2n+1 heap. A constant factor does not change the asymptotic behavior.

9 Separate Chaining Hashing

[8 points total]

1. Insert the following (key,value) pairs into the hash table below. Use the provided indexing function. For collisions, use separate chaining with linked lists.

index = key % tableLength

Pairs: (2,'a'), (3,'b'), (8,'c'), (13,'d'), (30,'e'), (22,'f'), (3,'g')

0	(30,'e')
1	
2	(2,'a') → (22,'f')
3	(3,'g') → (13,'d')
4	
5	
6	
7	
8	(8,'c')
9	

2. After completing all inserts, what is the load factor λ for the hash table above?
   $\frac{6}{10}$

3. If you were to re-hash the table above, assuming new keys would need to be inserted after and you had to increase the table size to a new value, what value would you suggest and why?
   I would set the table size to a prime number to reduce the likelihood of collisions. 11 is the next largest prime after the original size of 10, though if you don't know how many more values are coming then a larger number might be better, such as 17 or 23.

Some Useful Facts

When we're using the tree method to solve a recurrence, we usually use the following steps:

0. Draw a few levels of the tree.
1. Let the root node be at level 0. Give a formula for the size of the input at level i.
2. What is the number of nodes at level i?
3. What is the work done at the i^{th} recursive level?
4. What is the last level of the tree?
5. What is the work done at the base case?
6. Write an expression for the total work done.
7. Simplify until you have a "closed form" (i.e. no summations or recursion).

Geometric series identities:

$$\sum_{i=0}^{k} c^i = \frac{c^{k+1}-1}{c-1} \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c} \text{ if } |c| < 1$$

Common Summations:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad \sum_{i=0}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

Log identities:

$$a^{\log_b(c)} = c^{\log_b(a)} \qquad \log_b(a) = \frac{\log_d(a)}{\log_d(b)}$$

Master Theorem:
Given a recurrence of the following form:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{ some constant} \\ aT(n/b) + n^c & \text{otherwise} \end{cases}$$

with a, b, c are constants.
If $\log_b(a) < c$ then $T(n)$ is $\Theta(n^c)$
If $\log_b(a) = c$ then $T(n)$ is $\Theta(n^c \log n)$
If $\log_b(a) > c$ then $T(n)$ is $\Theta\left(n^{\log_b(a)}\right)$