Name: _____

UW NetID: _____

CSE 332 Summer 2018: 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.

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

Good Luck!

Total: 110 points. Time: 60 minutes.

Question	**Max Points**	**Score**
1	20	
2	12	
3	16	
4	10	
5	11	
6	8	
7	6	
8	15	
9	12	
Total	110	

1 Big-O

[20 points, 2 points each.] Give a **simplified**, tight big-O bound for each of these situations. Tight means that, in particular, while $O(2^{n!})$ will be technically correct for every problem, it will not earn you any points. For any array-based implementation, assume that the array still has empty space (and so doesn't need to be resized).

1. The best case running time for finding the minimum value in an AVL tree with n elements. 1. _____

2. The best case running time for finding the minimum value in a BST with n elements 2. _____

3. $f(n) = \log_3(2^n)$ 3. _____

4. $T(n) = \begin{cases} T(n/2) + 7 & \text{if } n \geq 4 \\ 5 & \text{otherwise} \end{cases}$ 4. _____

5. $h(n) = \sqrt{n} + \log(n)$ 5. _____

6. The worst case running time to insert into a sorted linked list with n elements. 6. _____

7. Calculating the height of every node in an AVL tree with n elements (from scratch). 7. _____

8. Inserting a new element into a minheap that has a worse (i.e. larger) priority than the n elements already in the heap. 8. _____

9. The maximum number of rotations that can occur on a single insertion into an AVL tree with n elements. 9. _____

10. $g(n) = \log^2(n^3) + \log(\log(2^n))$ 10. _____

2 Run Time Analysis

[12 points. 3 points each] For each of the following code snippets, give a big-Θ bound on the running time. Make sure your bound is as simple as possible.

```
int blah(int n){
    for(int i = n^2; i > 0; i=i/2){
          print "i";
    }
}
```

Runtime: Θ(_____)

```
int foo(int n){
      if(n < 20){
            for(int i = 0; i < n^3; i++){
                  print "hi"
            }
      }
      else{
           for(int i = 0; i < n^2; i++){
                  print "bye"
           }
      }
      return n/2;
}
```

Runtime: Θ(_____)

```
void baz(int n){
     for(int i = 0; i < n^2; i++){
           for(int j = i; j < n^3; j++){
              print "hello"
           }
     }
}
```

Runtime: Θ(_____)

```
int bar(int n){
      for(int i = 0; i < n^3; i++){
          if(i % 3 == 0){
                 break;
          }
          else{
               print ":D"
          }
     }
}
```

Runtime: $\Theta($_____$)$

3 O, Ω, Θ

[16 points. 4 points each] For each of the following, circle ALL the correct descriptions of the function (if any).

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

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

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

4. 2^n is
   a. $O(2^{2n})$ b. $\Omega(2^{2n})$ c. $\Theta(2^{2n})$ d. $\Omega(3^n)$

4 AVL tree insertions

[10 points.] Insert the following elements in order into an empty AVL tree.

$$1, 7, 6, 8, 9, 0, 4, 2, 5, 3$$

Circle your final tree. We strongly recommend showing intermediate trees, so we can give partial credit if you make a mistake.

5 New Operations and New Heights

1. (6 points) You just finished writing a library implementation of a binary min-heap for your new programming language, when you get a feature request – your users keep getting confused on whether low priority numbers means "urgent" or "not urgent." You decide to write a new operation

   > `reverse`:
   > given a binary min-heap stored as an array (with root at index 0)
   > convert it to a binary max-heap stored in the same array (with root at index 0).

   Recall that a max-heap is the same as a min-heap except every element has a larger priority than its children (instead of a smaller priority).

   Your first idea for `reverse` is just to reverse the order of the elements in the array i.e.

   ```
   for(int i = 0; i < size/2; i++){
         temp = arr[i]
         arr[i] = arr[size-i-1]
         arr[size-i-1] = temp
   }
   ```

   where `size` is the number of elements in the heap, and the heap is stored in `arr`.

   Does this algorithm work? If it does, informally argue why (formal proof not required). If it doesn't, provide a counter-example (with explanation of why it is a counter-example).

2. (3 points) What is the minimum number of nodes in an AVL tree of height 4? justify your answer.

3. (2 points) What is the minimum number of nodes in a BST of height 4?

6 Proof

[8 points] Write a formal proof that $5n^2 + 3\log(n) - 7$ is $O(n^2)$. Please include both your scratch work and the final proof.

Scratch work:

Proof:

7 Writing A Recurrence

[6 points] Write a recurrence describing the 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.

You do not need to solve for a closed form of THIS recurrence

```
int foobar(int n){
     if(n <= 15){
          return 2n + 8
     }
     else{
          for(int i = 0; i < n; i++){
               for(int j = 0; j < i; j++){
                    print i
               }
          }
          return foobar(n/2) * foobar(n-3) + n
     }
}
```

$$T(n) = \begin{cases} \\ \\ \\ \end{cases}$$

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

8 Solving A Recurrence

[15 points] In this problem you will find a closed form of the following recurrence, and check your answer with the Master Theorem.

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

1. Solve the recurrence. Your final answer should not have any recursion or summations, but need not "look nice." You should not simplify to a big-Θ answer in this part, keep all the exact constants in your answer. You may use either unrolling or the tree method. In either case, you must show your work for **ANY** credit. The list of steps for the tree method and the list of summations on the last page may come in handy.

2. Use the Master Theorem (see the last page) to get a big-Θ bound on the recurrence.

9 Data Structure Analysis

[12 points] Dumbledore needs more help. Every night Professor McGonagall and Professor Snape each submit a list of all disciplinary action they took in the last day. Each of them submits their list as a separate minheap to Dumbledore, with priority representing the severity of the incident (more severe incidents have lower-number priorities). Professor McGonagall's list consistently has n elements, while Snape's has 2^n elements.

Dumbledore would like to examine the $\sqrt{n}$ most severe incidents (regardless of whether they were in Snape's heap or McGonagall's). Note that Dumbledore only wants to look at $\sqrt{n}$ indicdents, fewer elements than are in either of the heaps.

Design an algorithm to get the $\sqrt{n}$ minimum priority elements appearing in either `S_Heap` or `M_Heap`. Section 4's handout contained one possible algorithm, based on merging the two heaps. Submitting that algorithm will earn you some credit, but for full credit you will need to provide a more efficient algorithm.

You know that `S_Heap` has size 2^n, `M_Heap` has size n, and they both implement standard heap operations. You may assume the priorities of all elements are distinct.

1. Write pseudocode for your algorithm here (if in doubt about what pseudocode should look like, emulate the code snippets in problems 2 and 5):

Continued on the next page.

2. What is the best case situation and running time for your algorithm? (your running time bound should be a simplified $O()$)

3. What is the worst case situation and running time for your algorithm? (your running time bound should be a simplified $O()$)

Extra page for scratch work.

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 $\Theta\left(n^{\log_b(a)}\right)$