CSE 332 25wi Sample Midterm

Name: _____**Sample Solution**_____

Email address (UWNetID): _____

Instructions:

- The allotted time is 50 minutes.
- Please do not turn the page until the staff says to do so.
- This is a closed-book and closed-notes exam. You are NOT permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- When provided, write your answers in the box or on the line provided.
- Unless otherwise noted, every time we ask for an O, Ω, or Θ bound, it must be simplified and tight.
- Unless otherwise noted, assume all data structures are implemented as described in lecture.
- For answers that involve bubbling in a ◯ or ☐, make sure to fill in the shape completely.
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.
- A formula sheet has been included at the end of the exam.

Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- **Relax and take a few deep breaths. You've got this! :-).**

Total: ?? points

Q1: Short-answer questions (20 pts)

- For questions asking you about runtime, give a simplified, tight Big-O bound. This means that, for example, $O(5n^2 + 7n + 3)$ (not simplified) or $O(2^{n!})$ (not tight enough) are unlikely to get points. Unless otherwise specified, all logs are base 2.
- For questions with a mathematical answer, you may leave you answer as an unsimplified formula (e.g. $7 \cdot 10^3$).

We will only grade what is in the provided answer box.

a. Give the **worst-case** runtime for inserting a value into a binary search tree containing N elements.

$O(\boxed{N})$

b. Give a simplified, tight big-O bound for: T(N) = T(N/2) + 16, given T(1) = 1

$O(\boxed{log_2 N})$

c. Give a simplified, tight big-O bound for: $f(N) = log^2(N) + 20log(N^2)$

$O(\boxed{log^2 N})$

d. Give the **worst-case** runtime for inserting a value into an AVL tree containing N elements, **where the value being inserted is smaller than the current smallest value in the tree**.

$O(\boxed{log_2 N})$

e. Give a simplified, tight big-O bound for: $f(N) = Nlog\,log(8^N) + N(log(N))(log(N))$

$O(\boxed{Nlog^2 N})$

Q1: (continued)

(Same instructions as on previous page)

f. Give the **worst-case** runtime for removing the next element from a CircularArrayFIFOQueue (from P1) containing N elements.

O(1)

g.Give the **worst-case** runtime for deleteMin() from a binary min heap **containing** 2^N **elements.**

O(N)

h. Suppose a 5-ary heap is stored in an array with the root in cell 0. Give the formula for computing the index of the cell containing the third (aka middle) child of the node in cell x. Give your answer in terms of x.

5x + 3

i. Give the **worst-case** runtime for pop() in a stack containing N elements implemented using an array .

O(1)

j. Give the **smallest value for c** that could be used **with** $n_0 = 1$ in a proof to show that:

$3n^2 + 5$ is O(n^2)

Given $n_0 = 1$

c = 8

Q2: Code Analysis (16 pts)

Describe the worst-case running time for the following pseudocode functions in Big-O notation in terms of the variable n. Your answer **MUST** be tight and simplified. **You do not have to show work or justify your answers for this problem.**

a)
```
void summer(int n) {
    for (int i = 0; i*i < n; i++) {
        for (int j = n; j >= 1; j = j/3) {
            System.out.println("Enjoy the sunshine!");
        }
    }
}
```

$O(\sqrt{n}\log_3 n)$

b)
```
void happy(int n) {
    if (n % 3 == 0) {
        for (int i = 456; i >= 0; i -= 5) {
            for (int j = i; j >= 0; j -= 2) {
                System.out.println("Yipee!");
            }
        }
    } else {
        int p = 0;
        for (int i = 1; i < n; i *= 2) {
            p += 1;
        }
        for (int j = 1; j < p; j *= 2) {
            System.out.println("Smile!");
        }
    }
}
```

$O(\log_2 n)$

Q2: (continued)

c)
```
AVLTree sunny(int n, BinaryMinHeap input) {
      // The size of input is n
      AVLTree output = new AVLTree();
      for (int i = 0; i < n; i++) {
            int val = input.deleteMin();
            output.insert(val);
      }
      return output;
 }
```

O($n \log n$)

d)
```
void funny(int n) {
      for (int i = 0; i < n; i++) {
            for (int j = i; j >= 0; j--) {
                  int z = 3 * n;
                  for (int k = 0; k < z; k += 3) {
                        System.out.println("Ha ha ha!");
                  }
            }
      }
 }
```

O(n^3)

Q3: O, Ω, and Θ (9 pts)

For each of the following statements, indicate whether it is always true, sometimes true, or never true. You do not need to include an explanation. Assume that the domain and codomain of all functions in this problem are natural numbers(1, 2, 3 ...).

a) If $f_a(n)$ is the worst case runtime for insert in a Binary Search Tree and $g_a(n)$ is the worst case runtime for insert in an AVL tree, then $f_a(n)\ is\ \Omega\left(g_a(n)\right)$ (Big-Omega)

- [x] Always
- [ ] Never
- [ ] Sometimes

b) $f_b(n)\ is\ O\left(\log\left((f_b(n))^2\right)\right)$

- [ ] Always
- [ ] Never
- [x] Sometimes

c) If $f_c(n)$ is the worst case runtime of merging three binary min heaps, each with n elements, and $g_c(n)$ is the worst case runtime of inserting $3n$ elements into a binary min heap, then $f_c(n)\ is\ O\left(g_c(n)\right)$

- [x] Always
- [ ] Never
- [ ] Sometimes

Q4: Write a Recurrence (8 pts)

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 MI6(int n) {
      if (n <= 7) {
            for (int i = 0; i < n; i++) {
                  System.out.println("The name's Bond, James Bond");
            }
            return n + 1;
      } else {
            if (n % 1 == 1) {
                  return MI6(n + 1);
            } else {
                  int x = n;
                  while (x != 0) {
                        System.out.println("Double O-" + x);
                        x -= 1;
                  }
                  for (int i = 1; i < n; i *= 3) {
                        System.out.println("Shaken not stirred");
                  }
                  return MI6(n / 2) + MI6(n - 3) + (n * n);
            }
      }
}
```

$T(n) =$ c_0 For $n \leq 7$

$T(n) =$ $T(n/2) + T(n-3) + c_1 n + c_2 log_3 n + c_3$ For $n > 7$

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

Q5: Solve a Recurrence with the Tree Method (10 pts)

Suppose that the running time of an algorithm is expressed by the recurrence relation:

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

$$T(1) = 1$$

For the following questions, use the tree method to solve the recurrence relation. We have broken up the process into subquestions to guide you through your answer. You may assume that n is always a power of 2.

1) Sketch the tree in space below. Include at least the first 3 levels of the tree (i.e. the root, its children, and its grandchildren), make clear what the input size is for each recursive call as well as the work per call.

...

2) Indicate exactly the total amount of work done at level i of the tree (define the root to be level 0). Include all constants and non-dominant terms.

$(\frac{3}{4})^i n^2$

3) Indicate the level of the tree in which the base cases occur.

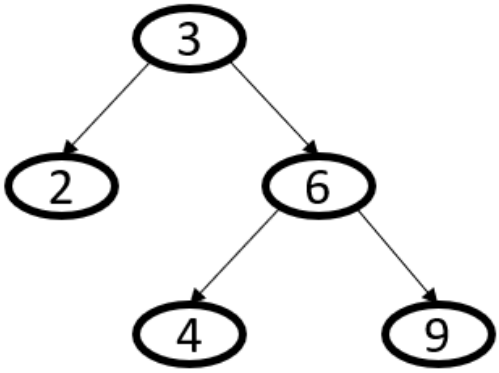
$\log_2(n)$

4) Give a simplified Θ bound on the solution. When simplified, n should not appear in any exponents.

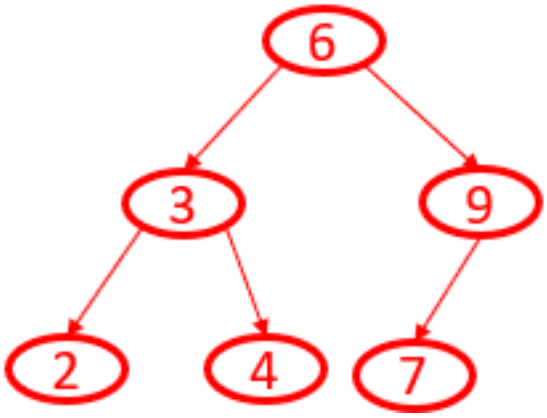
$\Theta\left(n^2 \right)$

Q6: AVL (7 pts)

a) (3 pts) Draw the AVL tree that results after inserting 7 into this AVL tree. Be sure to draw your final tree in the box below AND **indicate the total number of single and double rotations required for this insertion.**



Final Tree:



Total # of Single Rotations required:

1

Total # of Double Rotations required:

0

After inserting 7, a single rotation about 3 is all that is required

Q6: (continued)

b) (4 pts) Give the **minimum** and **maximum** **height** for an AVL tree containing 13 nodes.

Minimum **Height**: ___3____ Maximum **Height**: ___4_____

How many more **nodes** are needed to increase the **minimum** height by 1? ___3_____

How many more **nodes** are needed to increase the **maximum** height by 1? ____7_____

Max height: With 13 nodes, we can build an AVL tree with at most height 4. We would need another 7 nodes to be able to increase the height to 5.

Min Height: With 13 nodes, we could fit them into an AVL tree of height 3 (7 nodes will create a perfect tree of height 2, we could add the remaining 6 nodes on the next level). We would need at least 16 nodes total to force us to height 4.

Q7: Heaps (7 pts)

a. (4 pts) Given a **binary <u>max</u> heap** represented by this array [10, 9, 8, 7, 6, 5, 4, 2, 1], show the resulting heap (in array representation) after inserting the following elements in this order: {11, 33}

Show the final array contents (11 values total, starting from index 0):

0	1	2	3	4	5	6	7	8	9	10
33	**11**	**8**	**7**	**10**	**5**	**4**	**2**	**1**	**6**	**9**

b. (3 pts) Given the following values: {10, 2, 9, 11, 3, 99, 37, 1}, give an order of insertions into a **binary <u>max</u> heap** that results in the smallest number of `percolateUp` operations.

Order of insertion (8 values total):

__99_, __37__, ___11__, __10__, __9___, __3___, __2___, __1____