CSE 332 Winter 2025 Midterm
Version A

Name: _____

Email address (UWNetID): _____

Instructions:

- The allotted time is 60 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: 81 points

Q1: Short-answer questions (20 pts)

- Unless otherwise noted, assume all data structures are implemented as described in lecture.
- 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 an inorder traversal of a binary search tree containing N elements.

$O(\square)$

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

$O(\square)$

c. Give a simplified, tight big-O bound for:

$f(N) = N^2 + \log(N) + 2^{\log(N)}$

$O(\square)$

d. Give the **worst-case** runtime for creating a binary min heap out of an array containing N elements.

$O(\square)$

e. Give a simplified, tight big-O bound for:

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

$O(\square)$

Q1: (continued)

(Same instructions as on previous page)

f. Give the **worst-case** runtime for finding an element in an AVL tree containing N elements.

O( )

g.Give the **worst-case** runtime for inserting a value that is smaller than all the elements currently present into a binary max heap containing N elements. Assume that there is enough space in the array for the insertion.

O( )

h. Give the **worst-case** runtime for pushing **N^2** items onto a stack implemented using linked list nodes.

O( )

i. Give the minimum number of elements in a binary min heap of height 5.
(Remember: A single node is a tree of height 0.)

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

$2n^3 + 4$ is O(n^3)

Given $n_0 = 1$

c =

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 sunny(int n) {
    int count = 0;
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n*n; j+= n) {
            count++;
        }
        for (int k = 1; k < n*n*n; k = 2*k) {
            count++;
        }
    }
}
```

O( )

b)
```
public static void snowy(int n) {
    MyQueue<Integer> q = new LinkedQueue<>();
    q.enqueue(n);
    while (!q.isEmpty()) {
        int val = q.dequeue();
        if (val > 1) {
            q.enqueue(val / 2);
            q.enqueue(val / 2);
        }
    }
}
```

O( )

Q2: (continued)

c)
```
int swim(int n) {
       int bar = 0;
       for (int i = 0; i < n; i += 2) {
           if (i % 2 == 1) {
               for (int j = 0; j < n * n * n; j++) {
                   bar++;
               }
           } else {
               for (int j = 0; j < i; j++) {
                   bar++;
               }
           }
       }
       return bar;
}
```

O( )

d)
```
int ski(int n) {
     for (int i = 0; i < n * n; i++) {
          if (i % 10 == 0) {
               for (int j = 0; j < i; j++) {
                    sum++;
               }
          }
     }
     return sum;
 }
```

O( )

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) A function that is O(n) is _____ $O(n^2)$.

○ Always ○ Never ○ Sometimes

b) A function that is $\Theta(n)$ is _____ $O(n^2)$

○ Always ○ Never ○ Sometimes

c) A function that is O(n) is _____ O(log n).

○ Always ○ Never ○ Sometimes

d) A function that is O(n) is _____ $\Theta(n^2)$

○ Always ○ Never ○ Sometimes

e) A function that is $\Omega(\log(n^{1/\log n}))$ is _____ $\Omega(n^{2/3})$.

○ Always ○ Never ○ Sometimes

f) A function that is Ω(log n) is _____ Ω(n)

○ Always ○ Never ○ Sometimes

g) A function that is O(n) is _____ $\Omega(n^2)$.

○ Always ○ Never ○ Sometimes

h) If f(n) is $\Theta(g(n))$ and g(n) is Ω(h(n)), then f(n) is Ω(h(n)).

○ Always ○ Never ○ Sometimes

i) If f(n) is both O(g(n)) and Ω(h(n)), then **g(n)** is $\Theta(h(n))$.

○ Always ○ Never ○ Sometimes

Q4: Write a Recurrence (5 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.

```
public static int fun(int n) {
    if (n < 5) {
        return n;
    } else if (fun(n / 2) < 10) {
        int a = fun(n - 3);
        int b = fun(n / 2);
        return b;
    } else {
        int j = fun(n - 7);
        for (int i = 0; i < n * n; i += 4) {
            j += i;
        }
        return j;
    }
}
```

$T(n) =$ _____ For $n < 5$

$T(n) =$ _____ For $n > 5$

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

Q5: Tree Method (8 pts)

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

$$T(1) = 1$$

$$T(N) = 2T(\frac{N}{3}) + N^2 \qquad \text{for integers N > 1}$$

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 3. **Be sure to include bases in logs in your answers if any.**

a) Sketch the tree in the 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.**

b) 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.

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

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

Θ( )

Q6: Heaps (11 pts)

.

Here's an array presentation of a 0-indexed binary heap:

23	25	55	81	49	64	79	98	95	70	68		

a) (1 pt) This is a _____ (choose one: max/min) heap.
b) (2 pts) Draw the visual representation of the given heap.

c) (2 pts each) For each statement below, if there's an **insertion** that can achieve the goal, list the element to insert; otherwise, justify why it cannot be achieved (1-2 sentences max). Each statement is independent; you are allowed to insert one element for each statement.

i) 70 is a child of 64

ii) 79 is no longer a child of 55

iii) 49 only has one child

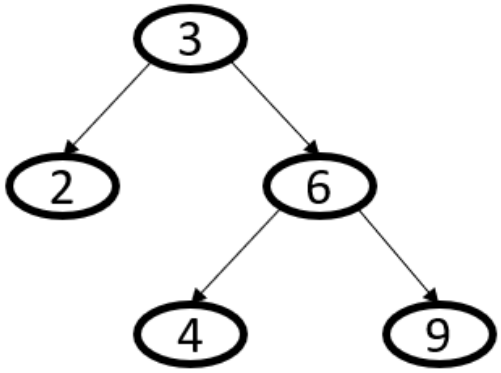
iv) No percolation is done for the insertion

Q7: AVL (12 pts)

a) (3 pts) Draw the AVL tree that results after **inserting 5** 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.**

b) (1 pts) This insertion required: (SELECT ONE)

- ◯ One single rotation
- ◯ One double rotation
- ◯ One single rotation AND One double rotation
- ◯ More than one single rotation and one double rotation



Final Tree:

Q7: (continued)

Jolie discovered that today is National Cream Cheese Brownie day and wants to organize 21 brownie recipes (numbered 1, 2, 3, …, 20, 21) in an AVL tree. (Remember: A single node is a tree of height 0.)

c) (2 pts) What is the **minimum** height of the tree? _____

d) (2 pts) What is the minimum number of recipes would we need to remove (not which ones) to **decrease the minimum height** of the tree? _____

e) (2 pts) How many more recipes would we need to add to **increase the minimum height** of the tree? _____

f) (2 pts) How many more recipes would we need to add to **increase the maximum height** of the tree? _____

Summations

1. $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ for $|x| < 1$
2. $\sum_{i=1}^{n} cf(i) = c\sum_{i=1}^{n} f(i)$
3. $\sum_{i=0}^{n-1} 1 = \sum_{i=1}^{n} 1 = n$
4. $\sum_{i=0}^{n} i = 0 + \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
5. $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
6. $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
7. $\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}$
8. $\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$

Logs:

1. $a^{\log_b(c)} = c^{\log_b(a)}$
2. $\log_b(a) = \dfrac{\log_d(a)}{\log_d(b)}$
3. $\log_b(b) = 1$
4. $\log_b(1) = 0$
5. $b^{\log_b(n)} = n$
6. $\log_b(n \cdot m) = \log_b(n) + \log_b(m)$
7. $\log_b(\frac{n}{m}) = \log_b(n) - \log_b(m)$
8. $\log_b(n^k) = k \cdot \log_b(n)$

This is a blank page! Enjoy!