CSE 332 Summer 2025 Exam 1

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! :-).**

Q1: Short-answer questions

- 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. A binary min heap is a type of binary search tree.

○ True

○ False

b. A heap can contain duplicate priorities.

○ True

○ False

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

$O(\boxed{\quad\quad})$

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

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

$O(\boxed{\quad\quad})$

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

$$f(N) = (2^N + N)^5$$

$O(\boxed{\quad\quad})$

Q1: (continued)

(Same instructions as on previous page)

f. For any constants a and b > 1, $\log_a(n) \in \Theta(\log_b(n))$.

◯ True

◯ False

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

O( )

h.Give the **worst-case** runtime for finding the smallest key in a separate chaining hash table.

O( )

i. Give the **worst-case** runtime for pushing 2^N elements into an initially empty ArrayStack with initial capacity of 1.

O( )

j. Give the minimum number of elements in a binary max heap of height 3.
(Remember: A single node is a tree of height 0.)

k. Give the exact number of leaves in a complete binary tree of n nodes.

Q2: Code Analysis

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)
```
public static void mystery1(int[] arr) {
    if (arr.length < 10000) {
        int cnt = 0;
        for(int i = 0; i < arr.length; i++) {
            for(int j = 0; j < arr.length; j++) {
                if (arr[i] < arr[j]) {
                    cnt++;
                }
            }
            arr[i] = cnt;
        }
    } else {
        int mid = arr.length / 2;
        int[] newArr = new int[mid];
        for (int i = 0; i < mid; i++) {
            newArr[i] = arr[i];
        }
        mystery1(newArr);
        for (int i = 0; i < mid; i++) {
            arr[i] = newArr[i];
        }
    }
}
```

O($\boxed{}$)

Q2: (continued)

b)
```
void mystery2(int n) {
    int sum = 0;
    for (int i = 1; i < Math.pow(4, n); i *= 2) {
        for (int j = 1; j < n; j++) {
            if (i < n) {
                sum += i + j;
            }
        }
    }
}
```

O(   )

c)
```
int mystery3(int n) {
     while (n > 0) {
          if (n % 5 == 0) {
               return n;
          }
          n--;
     }
     return 0;
}
```

O(   )

d)
```
int mystery4(int n) {
     int count = 0;
     for (int i = 0; i < n; i++) {
          for (int j = 0; j < i; j += 5) {
               count++;
          }
     }
     return count;
 }
```

O(   )

Q3: O, Ω, and Θ

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 $\Theta(n^2 + n)$ is ________ $\Theta(n^2)$.

○ Always ○ Never ○ Sometimes

b) A function that is $O(n\log(n))$ is ________ $O(n)$.

○ Always ○ Never ○ Sometimes

c) A function that is $O(\log^{30} n)$ is ________ $O(n)$.

○ Always ○ Never ○ Sometimes

d) A function that is $\Omega(n^2)$ is ____ $O(n^{1.5})$.

○ Always ○ Never ○ Sometimes

e) A function that is $\Theta(2^n)$ is ________ $\Omega(n^{2025})$.

○ Always ○ Never ○ Sometimes

f) if $f(n) \in O(g(n))$ and $f(n) \in \Omega(h(n))$, then $h(n)$ is ________ $O(g(n))$.

○ Always ○ Never ○ Sometimes

g) if $f(n) \in O(g(n))$, then $2^{f(n)}$ is ________ $O(2^{g(n)})$.

○ Always ○ Never ○ Sometimes

h) $f(n) + g(n)$ is ________ $\Theta(\max\{f(n), g(n)\})$.

○ Always ○ Never ○ Sometimes

Q4: Write a Recurrence

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 f(int n) {
     if (n < 100) {
         return 42;
     }

     int m = 10 * n;
     return f(n/2) + f(m/100) + f(1);
}
```

$T(n) =$ ____ For $n < 100$

$T(n) =$ ____ For $n \geq 100$

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

Q5: Tree Method

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

$$T(0) = 1$$

$$T(N) = 2T(N - 2) + 2^N \quad \text{for integers N > 0}$$

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 even.

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), and 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.

$\Theta(\boxed{})$

Q6: Heaps

.

Here's an array representation of a 0-indexed binary heap, named `heap` (the priority of each element is the same as its value, just like in lecture):

24	26	53	84	50	63	78	99	96	71	67		

a) This is a ____________ (choose one: max/min) heap.

b) Draw the visual representation of the given heap.

c) Suppose we execute the code: `heap.insert(x)`, where `x` is some integer (`x` is both the value and the priority, just like in lecture). For each assertion below, give a value of `x` such that the assertion is true after the code is executed, or explain why no such `x` exists (1-2 sentences max). Consider each assertion independently; use the same initial heap for each one.

i) Some node with value "63" has a child with value "71"

ii) No node with value "78" has a parent with value "53"

iii) Every node with value "50" has exactly one child

iv) No leaf has a value of "63"

Q7: AVL

a) Start from an empty AVL tree, and do the insertions of elements in the following order:

50, 40, 30, 35, 34, 60, 70, 80

Box the final tree.

b) To your tree in part (a), add in the value 90. Box the final tree.

c) The insertion from part (b) required: (SELECT ONE)

◯ One single rotation

◯ One double rotation

◯ One single rotation AND one double rotation

◯ No rotations

Q8: Hashing

a) For the following hash table, **insert** the following elements in this order: **23, 5, 3, 66, 43, 19, 79**, using the **Linear Probing** technique. Use the hash function **h(x) = x%10**. In the table below, the first column holds the indices and the second holds the values. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining values.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

b) Consider an initially empty quadratic-probing hash table with table size *p* and hash function h(x) = 0. What is the best-case runtime of inserting *n* elements into the table? Give a simplified tight big-oh bound in terms of *n*, and assume that *p* is prime and *n/p* < 0.5.

$O(\boxed{})$

c) Suppose we insert *n* elements into an initially empty separate chaining hash table with initial table size 12 and hash function h(x) = 4x. In the updated hash table, let *p* be the table size and let *L* be the length (i.e. number of elements) of the longest chain. What is the minimum possible value of *L*? Give an exact answer in terms of *n* and *p*. Assume that *n* is a multiple of *p*, and that resizing doubles the table size.

$\boxed{}$

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. $\log_b(a) = \frac{\log_d(a)}{\log_d(b)}$
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!