CSE 332 Spring 2024: Midterm Exam

Name: _____Solution_____

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 noted, assume all data structures are implemented as described in lecture.
- For answers that involve bubbling in a ◯, 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 the exams; edges often get cropped.
- 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! :-).**

Question #	Topic	Points	Page #
Q1	**Short Answer**	**16**	**1-2**
Q2	**Code Analysis**	**12**	**3-4**
Q3	**O, Ω, and Θ**	**9**	**5**
Q4	**Stacks, Queues, BSTs, and Tries**	**10**	**6-7**
Q5	**Write a Recurrence**	**8**	**8**
Q6	**Solve a Recurrence**	**10**	**9**
Q7	**Heaps & AVL Trees**	**12**	**10-11**
Q8	**AVL Trees**	**9**	**12**
Q9	**B Trees**	**9**	**13**
Extra Credit		**(1)**	**14**
Total		**95**	

Q1: Short Answer (16 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 a simplified, tight big-O bound for $T(N) = T(\frac{n}{2}) + N$, with $T(1) = 1$.

O(***N*** **)**

b. The key that we will percolate down from the root in the following binary max heap, once we call `deleteMax()`.

42	7	18	5	3	14	15	4	1	6		

6

c. The number of nodes in a perfect binary tree of height h.

$2^{h+1} - 1$

d. Combining 2 binary min heaps, each containing N elements, into one binary min heap (worst case).

O(***N*** **)**

Q1: (continued)

(Same instructions as on previous page)

e. Suppose a 3-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 1st child of the node in cell x. Give your answer in terms of x.

3x + 1

f. Given M = 4 and L = 11, what is the maximum number of key-value pairs in a B-tree with height 5? You may leave your answer as an unsimplified formula.

$11*4^5$

g. Give a simplified, tight big-O bound for $f(N) = N \log N^{1/2} + \frac{1}{3}N^{1/2}$

O(***N log N*** **)**

h. Printing all values less than a given value in an AVL tree containing N integers. (worst case).

O(***N*** **)**

Q2: Code Analysis (12 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 (ie **Θ**) and simplified. **You do not have to show work or justify your answers for this problem.**

a) Assume **`str`**'s length is **`n`**, and functions **`charAt`** and **`length`** are constant-time operations.

```
boolean isPalindrome(String str) {
    int left = 0;
    int right = str.length() - 1;
    while (left < right) {
        if (str.charAt(left) != str.charAt(right)) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
```

O(***N*** **)**

We iterate through half of the string, moving inward by 1 each iteration

b)

```
int timeToCatchUp(int n) {
    if (n <= 0) return -1;
    int A = 332;
    int B = 0;
    int time = 0;
    while (A > B) {
        A += n;
        B += 2 * n;
        time += 1;
    }
    return time;
}
```

If n is large, the loop will "catch B up" in one iteration

O(***1*** **)**

Q2: (continued)

c) Assume the size of Stack A is **M** and the size of Stack B is **N**.

```
void processStack(Stack<Integer> A, Stack<Integer> B) {
    if (A.isEmpty() || B.isEmpty()) return;

    int x = A.pop();
    int y = B.pop();

    if (x <= y) {
        A.push(x);
    } else {
        B.push(y);
    }

    processStack(A, B);
}
```

O(***M + N*** **)**

We pop off the smallest element on the top of the stacks each time, so at most we recurse a number of times equal to the number of elements in both stacks.

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 $f(n)$ is in $O(g(n))$ and $g(n)$ is in $O(f(n))$, then $f(n)$ is in $\Theta(g(n))$.

● **Always True** ○ Sometimes True ○ Never True

$f(n) \leq g(n)$ and $g(n) \leq f(n)$ means $f(n) = g(n)$ which is Θ

b) $f(n) * g(n)$ is in $O\left(max\left(f(n)^2, g(n)^2\right)\right)$.

● **Always True** ○ Sometimes True ○ Never True

It is always true that $f(n)$ is in $O(f(n)^2)$
If $f(n)^2 > g(n)^2$, then $f(n) > g(n)$ and $f(n)^2 > f(n)*g(n)$
If $g(n)^2 > f(n)^2$, then $g(n) > f(n)$ and $g(n)^2 > f(n)*g(n)$

c) $f(n)$ is in $\Omega\left(log(f(n)^2)\right)$.

● **Always True** ○ Sometimes True ○ Never True

$\log(f(n)^2)$ simplifies to $2 \log(f(n))$ which is $\Omega(\log(fn))$
The log of a function is always smaller than or equal to the function itself, for functions with natural number domain/codomain.

Q4: Stacks, Queues, BSTs, and Tries (10 pts)

a) (6 pts) You are developing an application to handle book checkouts and returns at a library. Below, we list a few operations the library should support. For each operation, fill in the circle corresponding to the ***most efficient*** data structure that would be best suited.

1) **Manage the hold line:** If a book is unavailable, users are added to a hold line. Users are removed from the hold-line on a first-come first-serve basis.

◯ ArrayStack ●**LinkedListQueue** ◯BST ◯Trie

2) **Search for books by title:** Allow for fast lookup of books by their full title for the catalog.

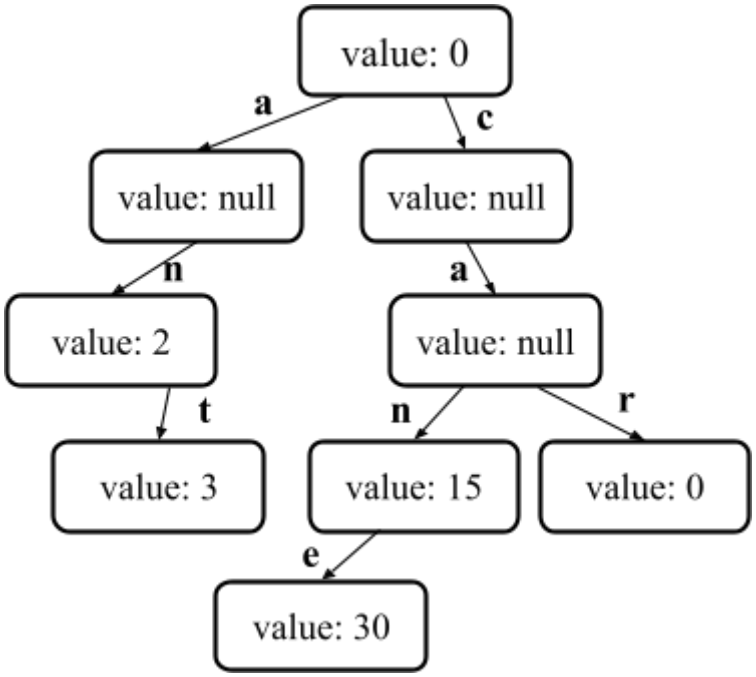
◯ArrayStack ◯LinkedListQueue ◯BST ●**Trie**

3) **Inventory tracking:** Able to generate a sorted inventory of books by the book's ID at any given time for inventory management.

◯ArrayStack ◯LinkedListQueue ●**BST** ◯Trie

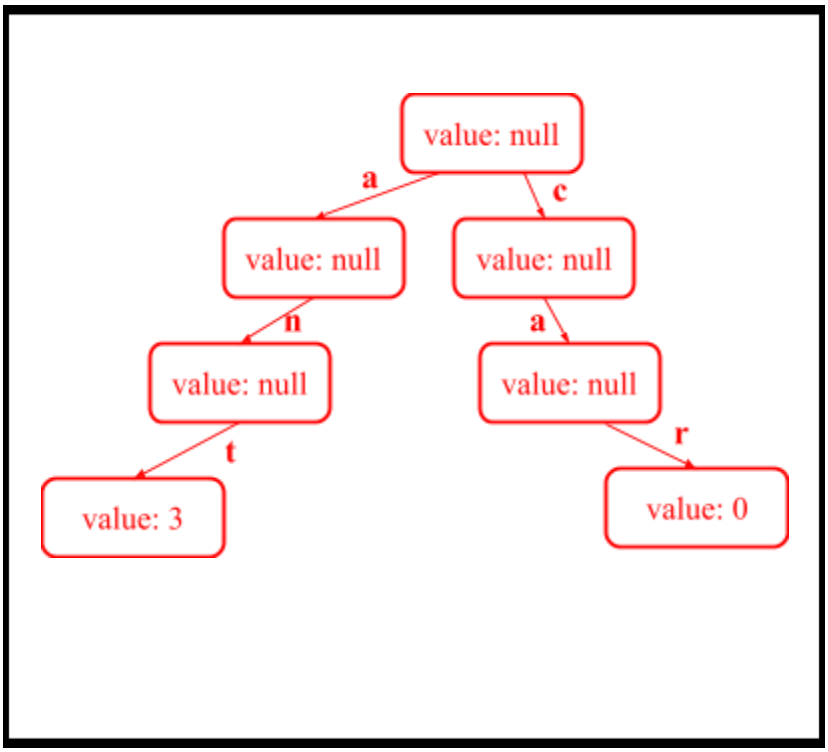
Q4: (continued)

b) (4 pts) Given the trie below, draw the final trie after completing the following deletions. As in Project 1, `delete(k)` should delete `k` and its value from the trie as well **as all of the nodes that become unnecessary**.




```
delete("can")
delete("an")
delete("cse")
delete("cane")
delete("")
```

Draw the final trie below. **Please draw it in the box.**



Q5: 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 cherryBlossom(int n) {
    if(n < 3) {
        for (int i = 0; i < 5; i++) {
            for (int j = i; j < 12; j++) {
c0              print("good luck on the exam!")
            }
        }
        return 6;
c1  } else {
        int flowers = 0;
T(n/3)  if (cherryBlossom(n / 3) < 5){
            flowers++;
        }
c2n^2   for (int i = 0; i < n * n; i++) {
            print("spring");
        }
2T(n-5) return 4 * cherryBlossom(n - 5) +
                2 * cherryBlossom(n - 5);
    }
}
```

$T(n) =$ | c_0 | for $n < 3$

$T(n) =$ | $c_1 + c_2n^2 + T(n/3) + 2T(n-5)$ | otherwise

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

Q6: Solve a Recurrence (10 pts)

Suppose that the running time of an algorithm satisfies the recurrence given below. Find the closed form for T(N) and its corresponding **Θ** bound. We have broken up the process into subquestions to guide you through your answer.

You may assume that N is a power of 3. You may find the list of summations and logarithm identities on the last page of the exam to be useful. You may use either the unrolling method or the tree method, whichever you are most comfortable with.

$$T(n) = 3T(n/3) + 6n$$

$$T(1) = 3$$

a) Show at least three expansions of the recurrence using either the tree or unrolling method:

Unrolling Method:

$T(n) = 3(3T(n/3^2) + 6(n/3)) + 6n$

$= 3^2T(n/3^2) + 6n*2$

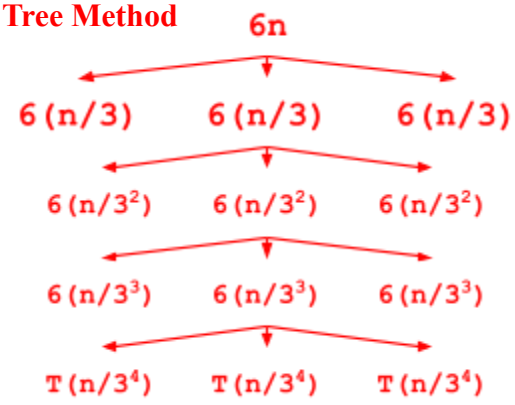
$= 3^2(3T(n/3^3) + 6(n/3^2)) + 6n*2$

$= 3^3T(n/3^3) + 6n*3$

$= 3^3(3T(n/3^4) + 6(n/3^3)) + 6n*3$

$= 3^4T(n/3^4) + 6n*4$

Tree Method



b) Give a summation (tree method) or equation (unrolling method) to express the total work in terms the height of the tree *k*. Include all constants and non-dominant terms.

Unrolling: $3^kT(n/3^k)+6kn$

Tree: (incl base case)

$$3 \cdot 3^i + \sum_{k=0}^{i-1} 3^k 6\frac{n}{3^k} = 3^{i+1} + 6n \sum_{k=0}^{i-1} 1$$

c) Indicate the level of the tree or unrolling in which the base case(s) occur, in terms of *n*.

$i = \log_3 n$

d) Give the closed form of the recurrence, simplifying your answer from part b using your tree height from part c. This should include relevant *exact* constants and bases of logarithms (eg do NOT use "c_1, c_2" in your answer; your answer should have integer coefficients/constants). It is NOT in big-O notation; it also must NOT have any summation symbols or recursion.

$3n + 6\, n \log_3 n$

e) Finally, give a simplified **Θ** bound on the recurrence you found in the previous step.

O(n log n)

Q7: AVL Trees & Heaps (12 pts)

We would like to store and perform certain operations on a collection of integers. Help us analyze the time complexity tradeoffs between using a **binary min heap** and an **AVL tree** by answering the series of short-answer questions below. First, **check the circle** corresponding to the data structure you think is more efficient for each operation. Then, give the big-O time complexity of the operation using each data structure to justify your choice, with a brief description of what you are basing the complexity on. **Please limit your explanation to 1 sentence per answer (we mean it - just time complexity plus a few words).**

a) (3 pts) Starting from an empty collection, we insert a series of `n` strictly increasing integers.

● **Heap** ○ AVL Tree

Justify your answer by analyzing the time complexity of each data structure:

Heap	**O(n) - strictly increasing means we don't have to percolate, just add to the end of the array**
AVL Tree	**O(n log n) - must traverse the tree to the bottom (log n) on each insertion (n times), then rotate as needed**

b) (3 pts) We want to determine the smallest integer in an existing collection.

● **Heap** ○ AVL Tree

Justify your answer by analyzing the time complexity of each data structure:

Heap	**O(1) - first element of a min heap is the smallest**
AVL Tree	**O(log n) - traverse down the left children**

Q7: (continued)

c) (3pts) We want to find the maximum integer in an existing collection.

◯ Heap ● **AVL Tree**

Justify your answer by analyzing the time complexity of each data structure:

Heap	**O(n) - largest could be anywhere in the heap, must search - there are about n/2 leaves**
AVL Tree	**O(log n) - traverse down the right children**

d) (3pts) We want to print out all elements in an existing collection in sorted descending order.

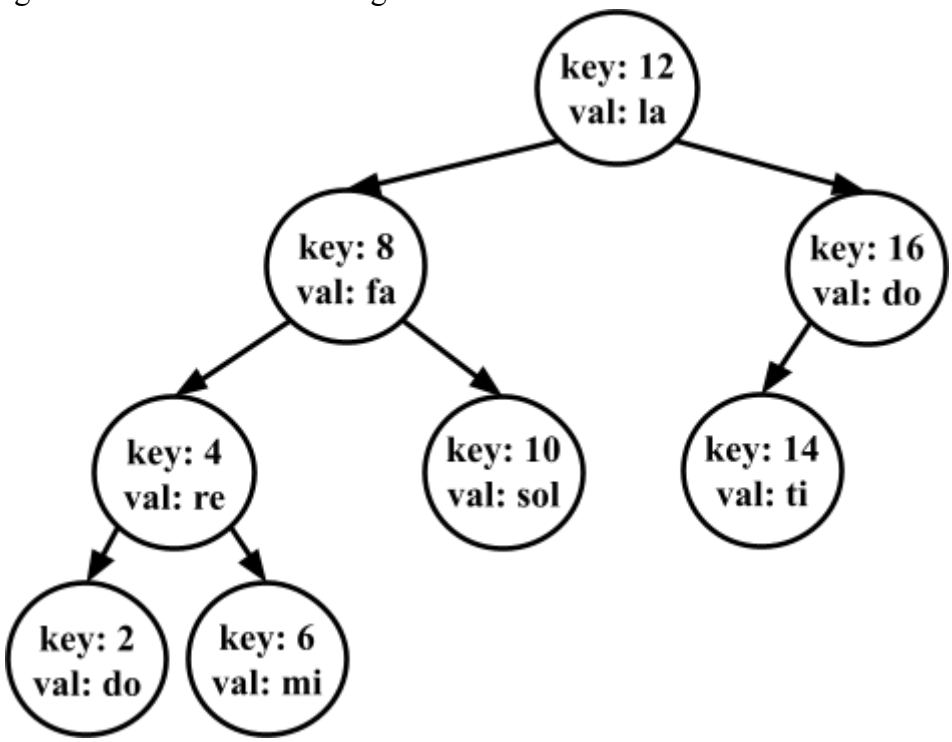
◯ Heap ● **AVL Tree**

Justify your answer by analyzing the time complexity of each data structure:

Heap	**$O(n^2)$ - to print in DESCENDING order, we can't use deleteMin() - we essentially have to do part c (above) n times. Any O(n log n) solution would involve another data structure or destroying the heap.**
AVL Tree	**O(n) - in-order traversal must visit all nodes once (although this is the inverse of a standard in-order traversal, visiting right children first)**

Q8: AVL Trees (9 pts)

Answer the questions about the following AVL tree, which represents a dictionary where the keys are integers and the values are strings.



Assume we call `insert(k,"new")` for some value of `k`, which may or may not be a key already present in the tree. Consider only `0 < k ≤ 17`.

a) (3 pts) What values of `k` would **NOT** cause the AVL tree to rebalance via rotation?

Existing keys (this is dictionary so we replace their values): 2, 4, 6, 8, 10, 12, 14, 16
Also 9, 11, and 17

b) (3 pts) What values of `k` would cause the AVL tree to rebalance via a **SINGLE** rotation?

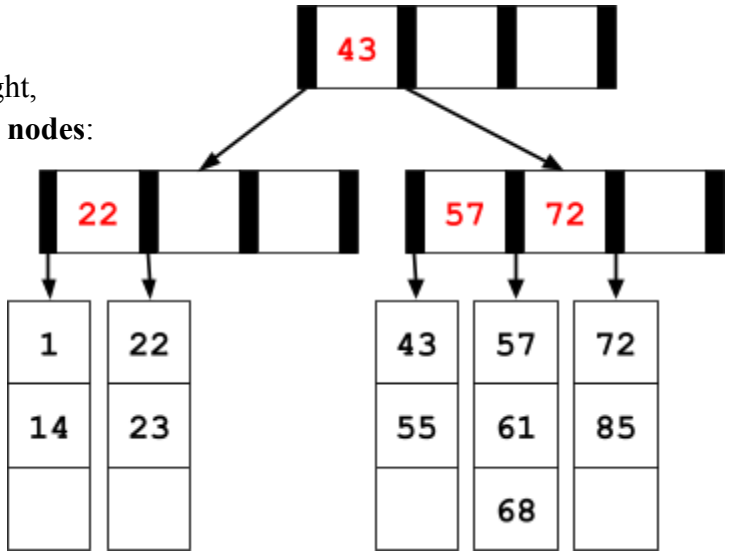
1, 3, 13

c) (3 pts) What values of `k` would cause the AVL tree to rebalance via a **DOUBLE** rotation?

5, 7, 15

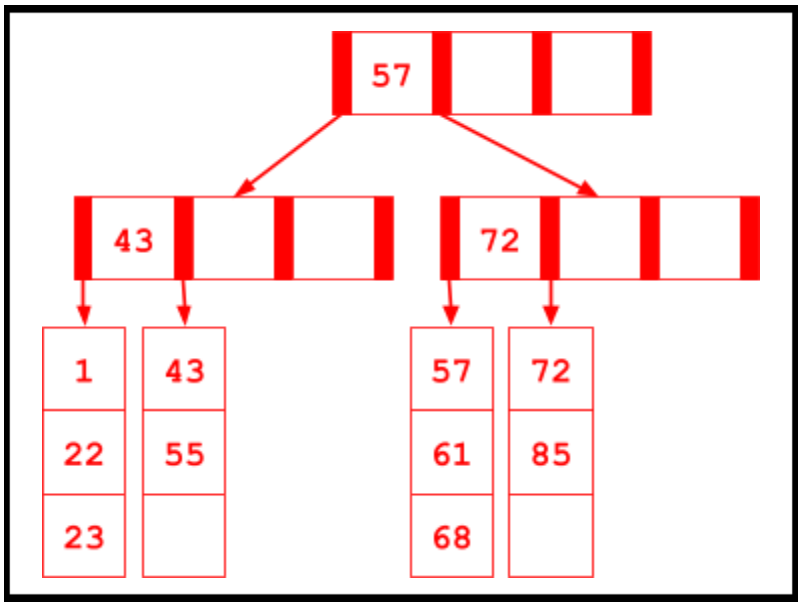
Q9: B Trees (9 pts)

a) (2 pts) In the B-tree shown to the right, **write in the values for the interior nodes**:



b) (3 pts) Starting with the tree above, draw the tree resulting after deleting 14 from the B-tree. **Please draw it in the box.**

Remove 14
Then merge with 22/23
Then parent adopts from its sibling (43/55)
Then internal node values are updated



c) (2 pts) What are *M* and *L* in the tree above?

M= 4 L= 3

d) (2 pt) If we insert 69 into the **original** tree in part (a), how many disk blocks would be accessed in the operation, assuming that the insert operation works as described in lecture and the B Tree is laid out on disk as described in lecture?

# of disk blocks: 4 **root, right child, middle leaf and one sibling**

Extra Credit (1pt)

In the space below, draw a picture of your favorite dinosaur.