CS 1332 Practice Exam
Fall Semester 2025 - September 11, 2025

Name: ____

GT Username: ____________________ **GT ID:** ____________________

Section (CIRCLE): A (2:00pm TR) B (5:00pm TR) C (12:30pm T) D (12:30pm R) O (Online)

By signing here, I understand and accept my responsibility to uphold the Georgia Tech Honor Code. I affirm that I have neither provided nor received any unauthorized aid during this exam. I will not discuss the exam contents with any other students, including those who have already taken it, until the exams are returned.

Signature: ____

- ☕ You must scan your Buzzcard before leaving the exam room or you may get a zero. Please talk to a TA if your Buzzcard is not available.
- ☕ You are not allowed to leave the exam room and return except for emergencies. If you leave the room for any other reason, then you must turn in your exam as complete.
- ☕ Raise your hand if you need clarification on a question or need to use the restroom.
- ☕ Notes, calculators, phones, laptops, smart watches, headphones, etc. are not allowed. Extra paper is not allowed. If you have exhausted all space, talk with a TA. There are blank pages in the exam for extra space. Do not share pencils or other items.
- ☕ If you plan on using ear plugs (foam or silicone, NOT earbuds) during the exam, you must show them to a TA for approval.
- 🦆 If you have a rubber duck, you may silently consult with it at any time.
- ☕ All work entered on this exam, whether code, diagrams or multiple choice, must be implemented as was presented in lecture.
- ☕ All code must be in Java. Pseudocode is not sufficient for credit.
- ☕ Efficiency matters. For example, if you code something that uses O(n) time when there is a way to do it in O(1) time, your solution may lose credit. If your code traverses data 5 times when once would be sufficient, this also is considered poor efficiency even though both are O(n).

Writing after time is called will result in a point deduction.

Table of Contents

1) Efficiency - Multiple Choice [12 points]

Goal: Answer the following questions about time complexity.

Requirements:

- Assume an optimal implementation, as taught in the course.
- **Do not** use amortized analysis unless specified.
- Do not assume internal access to the data structure unless specified.
- Choose the tightest Big-O bound possible for the scenario.

A.) Adding to the middle index of an ArrayList.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

B.) Adding to the back of a CircularSinglyLinkedList.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

C.) Adding to the third-to-last spot in a DoublyLinkedList.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

D.) **Average** case of adding an element that is larger than every other data in the BST.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

E.) Removing the oldest element from a Stack using the pop operation.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

F.) Enqueueing an element to a LinkedList-backed Queue.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

G.) Searching for a value in a BST.

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ O(n^2)

2) Scenarios - Multiple Choice [9 points]

Goal: Answer the following data structure scenario questions.

Requirements:

- Follow the implementations taught in lecture when answering questions.
- Select one answer for each question unless otherwise specified.

A.) Krispy Kreme needs your help tracking their supply of donuts. They are constantly making fresh, hot donuts to add to their stock, which then cool at the same rate. Customers may either take the hottest, freshest donut or choose the coldest, most aged donut instead. However, they may not take donuts that are in the middle. Which data structure best represents the donuts?

○ Queue ○ Deque ○ DoublyLinkedList ○ Heap

B.) Halloween is approaching, and you want a way to track the candy you've collected and model the line of trick-or-treaters at each house. You hate digging for candy in your bucket, so you will only ever take candy off the very top. Additionally, you don't like wasting time, so once you commit to waiting at a house, you will not leave the line until you have received candy. Which combination of data structures best represents your candy bucket and the lines of trick-or-treaters waiting at each house?

○ Stack and Queue ○ Stack and Deque ○ Heap and Queue ○ Heap and Deque

C.) What operations would a DoublyLinkedList with only a tail pointer perform faster than a SinglyLinkedList with both a head pointer and a tail pointer? Select all that apply.

□ addToFront() □ addToBack() □ addThirdFromBack() □ removeFromBack()

D.) The Hamby Corporation wants to create a record system for their massive amount of sales. Lookups are rare and only happen in the case of a discrepancy, but hundreds of items are added every minute and even one slow addition could cost thousands of dollars in sales. What structure should they use to optimize adding items fast?

○ ArrayList ○ SinglyLinkedList ○ Deque ○ BST

E.) You have various CPUs numbered 0, 1, 2, …, etc. You want to store and increment the number of operations *each* CPU completes. Because there are so many tasks, the counts go up rapidly. What data structure could store each CPU's count efficiently?

○ ArrayList ○ SinglyLinkedList ○ Queue ○ BST

F.) You are a city planner for MARTA and want to design software that tracks the movement of trains on the circular perimeter of Atlanta. You need to store each train relative to the trains adjacent to it. What data structure can naturally represent a loop of trains?

○ ArrayList ○ LinkedList ○ CircularSinglyLinkedList ○ Binary Tree

G.) Diego is a white-hat hacker that has made a virus to (ethically) infect the computers of 1332 TAs. To avoid suspicion, he programmed the virus to only infect **two** other systems before shutting the original computer down. He wants to make a graphical representation of the virus as it travels through the computers. Which data structure is suitable?

○ ArrayList ○ LinkedList ○ CircularSinglyLinkedList ○ Binary Tree

H.) You are building a delivery dispatch system for Taco Tuesday. Delivery requests are added in the order they are received, and must be delivered in that exact order to ensure fairness. What data structure would best support this?

○ Array ○ Stack ○ Queue ○ Deque

I.) You are designing an undo/redo system for a drawing app. Users can undo their last action or redo a previously undone action. Which data structure would be best for supporting both operations efficiently?

○ BST ○ Stack ○ Queue ○ Deque

3) Linear Data Structures - Diagramming [10 points]

Goal: Given the initial Queue and Stack, both of which are backed by an Array, perform the following list of operations in order.

Requirements:

- Draw the **final state** of the Stack and Queue.
- Assume the Stack and Queue are implemented as taught in lecture
- You must indicate where the front is in the Queue.

Initial Stack:

0	1	2	3	4	5
103	20	45			

Initial Queue:

0	1	2	3	4	5
				50	3
				F	

Operations:

```
1. queue.enqueue(23)
2. queue.enqueue(stack.pop())
3. stack.push(queue.dequeue())
4. queue.enqueue(63)
5. stack.pop()
6. queue.dequeue()
7. queue.enqueue(stack.pop())
8. stack.push(queue.dequeue())
```

Final Stack:

0	1	2	3	4	5

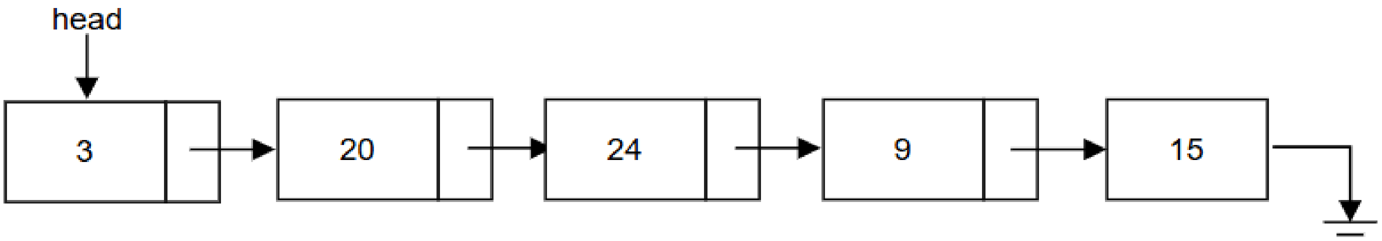
Final Queue:

0	1	2	3	4	5

4) Singly Linked List - Tracing [10 points]

Goal: Trace the execution of the given method when given the head of the SinglyLinkedList below. Write the final state of the LinkedList after the method completes in the box.

```
private class Node {
    public int data;
    public Node next;
    // CONSTRUCTORS OMITTED
}
public void modify() {
     head = helper(head);
}
private Node helper(Node curr) {
    if (curr == null) {
        return head;
    }
    curr.next = helper(curr.next);
    if (curr.data % 2 != 0) {
        return curr.next;
    }
    return curr;
}
```



Final Result after calling modify():

Do not write below this line. It may not be scanned, therefore it will not be graded.

5) Shape Properties - Diagramming [10 points]

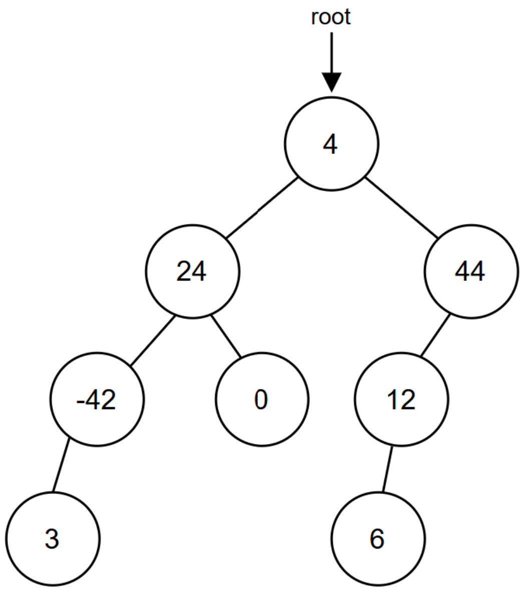
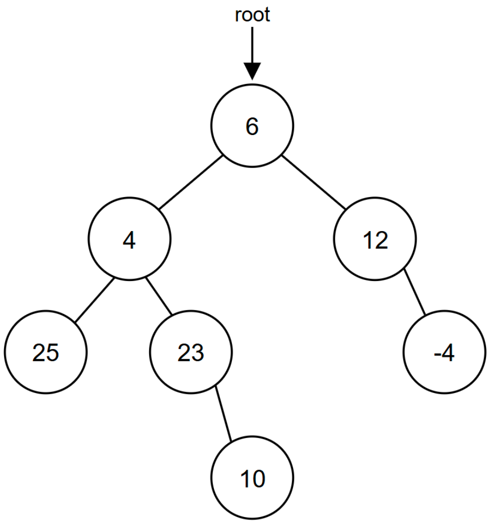
Goal: Create Binary Trees with the given shape properties and **at least four** nodes. It is **not** necessary to put data into the nodes.

Requirements: ● **Full** ● **Not balanced** ● **Not complete**	Tree:
Requirements: ● **Balanced** ● **Complete** ● **Not full**	Tree:

6) Binary Tree Traversal - Multiple Select [10 points]

Goal: Identify which value sequences appear in both listed traversals of the tree.

Note: [X, Y, Z] means we visit X then Y then Z.

Question	**Options** (**select all applicable**)
 Which sequences of values appear in both an **In-order** and **Post-order** traversal of this tree?	□ A [4, 6] □ B [6, 12, 44] □ C [-42, 0, 24] □ D [3, -42]
 Which sequences of values appear in both an **In-order** and **Pre-order** traversal of this tree?	□ A [12, -4] □ B [23, 10] □ C [4, 25, 23, 10] □ D [25, 4, 23]

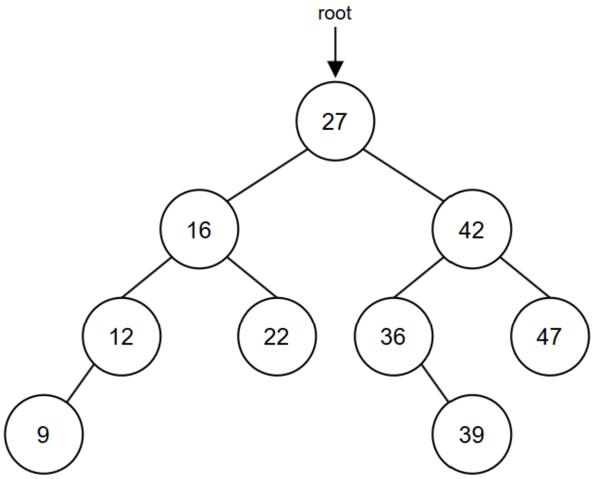
7) Binary Search Trees - Diagramming [12 points]

Goal: Perform the sequence of operations on the provided BST's and draw the resulting trees.

Requirements:

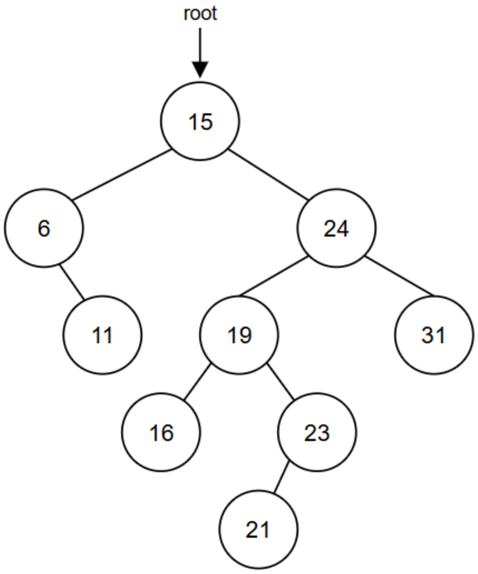
- If needed, use the predecessor node when removing.
- If you draw multiple steps, circle your final solution.

remove(12)
add(10)
remove(39)



Final Tree:

remove(24)
add(22)



Final Tree:

8) ArrayList - Coding [12 points]

Goal: Given the following ArrayList class, implement **`removeFirstAndLast()`** which removes and returns a size-2 array containing the first and last elements of the ArrayList.

Requirements:

- **Your code should be as efficient, in both space and time, as possible.**
- You may **not** assume any other method in the `ArrayList` class is implemented.
- Assume `size` accurately reflects the size of the `ArrayList` prior to execution of the method.

Example:

	0	1	2	3	4	5
Initial Array	4	9	13	3	2	

removeFirstAndLast() ⟶ returns: [4, 2]

	0	1	2	3	4	5
Final Array	9	13	3			

```
public class ArrayList {

    private int[] backingArray;
    private int size;

    /**
     * Removes the first and last elements in the ArrayList and returns those
     * elements in an array.
     *
     * @throws NoSuchElementException if there are less than 2 items in the
     *         ArrayList
     * @returns int[] int array of size 2
     */
    public int[] removeFirstAndLast() {
        // YOUR CODE HERE
```

```
    } // END OF METHOD
} // END OF CLASS
```

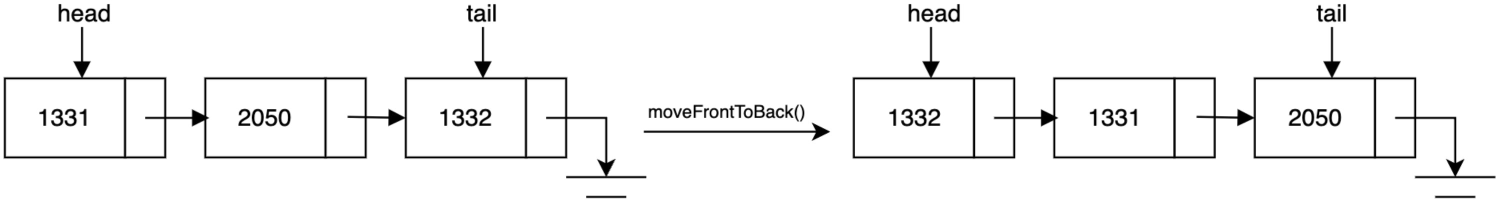
Do not write below this line. It may not be scanned, therefore it will not be graded.

9) Singly Linked List - Coding [15 points]

Goal: Given the following SinglyLinkedList class, implement the **`moveBackToFront()`** method which relocates the last node in the list (the tail) to the front, making it the new head.

Requirements:

- Since Node is an inner class, **you should access its fields directly** (e.g., **`node.data`**).
- **Your code should be as efficient as possible.**
- You may **not** assume any other methods in the SinglyLinkedList class are implemented. You must correctly update both the head and tail pointers.



```
public class SinglyLinkedList {

    private class Node<T> {
        public T data;
        public Node<T> next;
        public Node(T data, Node<T> next) { ... }
    }

    private Node<T> head;
    private Node<T> tail;
    private int size;

    /**
     * Moves the last node (tail) to the front of the list.
     *
     * If the list is empty or has only one node, the list remains unchanged.
     * After completion, the last node becomes the new head.
     */
    public void moveBackToFront() {
        // YOUR CODE GOES HERE, USE NEXT PAGE
```

Do not write below this line. It may not be scanned, therefore it will not be graded.

```
    } // END OF METHOD

} // END OF CLASS
```