CS1332 Campus Spring 2023 Exam 1 Version A1

Aaryan Vinay Potdar

TOTAL POINTS

99.5 / 100

QUESTION 1

Efficiency - Multiple Choice 14 pts

1.1 Part A 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(log n)$$

✓ **+ 2 pts** *$$O(n)$$*

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

1.2 Part B 2 / 2

✓ **+ 2 pts** *$$O(1)$$*

+ **0 pts** $$O(log n)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

1.3 Part C 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(log n)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(n \log n)$$

✓ **+ 2 pts** *$$O(n^2)$$*

1.4 Part D 2 / 2

✓ **+ 2 pts** *$$O(1)$$*

+ **0 pts** $$O(log n)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

1.5 Part E 2 / 2

✓ **+ 2 pts** *$$O(1)$$*

+ **0 pts** $$O(log n)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

1.6 Part F 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(log n)$$

✓ **+ 2 pts** *$$O(n)$$*

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

1.7 Part G 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(log n)$$

✓ **+ 2 pts** *$$O(n)$$*

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

QUESTION 2

2 Doubly Linked List - Tracing 10 / 10

✓ **+ 10 pts** *Completely Correct*

![dll-tracing-correct-version-a.png](/files/39fd252b-

67c5-405b-a712-4d4232c4e19b)

+ 7 pts Reversed print order

Partial Credit

+ 1.5 pts "Bingo!" is printed

+ 1.5 pts "11" is printed

+ 1.5 pts "6" is printed

+ 1.5 pts "10" is printed

- 1 pts Minor error

QUESTION 3

Scenarios - Multiple Choice 10 pts

3.1 Part A 2 / 2

+ 0 pts ArrayList

✓ **+ 2 pts** *Array-backed Queue*

+ 0 pts SLL-backed Stack

+ 0 pts Array

+ 0 pts incorrect

3.2 Part B 2 / 2

+ 0 pts Binary Ttree

+ 0 pts Array List

+ 0 pts SLL-backed Queue

✓ **+ 2 pts** *Array-backed Stack*

+ 0 pts incorrect

3.3 Part C 2 / 2

+ 0 pts Stack

+ 0 pts Queue

✓ **+ 2 pts** *Tree*

+ 0 pts ArrayList

3.4 Part D 2 / 2

+ 0 pts SLL-backed Stack

✓ **+ 2 pts** *Array*

+ 0 pts SLL w/ tail

+ 2 pts Queue

+ 0 pts incorrect

3.5 Part E 2 / 2

+ 0 pts Stack

+ 0 pts SLL-backed Queue

✓ **+ 2 pts** *Deque*

+ 0 pts Array-backed Queue

QUESTION 4

Data Structure Properties - Multiple Select 10 pts

4.1 Part 1 3 / 3

✓ **+ 1 pts** *A: ArrayList*

✓ **+ 1 pts** *E: Array-backed Stack*

✓ **+ 1 pts** *G: Array-backed Queue*

Extra answers

- 1 pts One extra answer

- 2 pts Multiple extra answers

4.2 Part 2 4 / 4

✓ **+ 2 pts** *E: Array-backed Stack*

✓ **+ 2 pts** *F: Linked List-backed Stack*

Extra answers

- 1 pts One extra answer

- 2 pts Multiple extra answers

4.3 Part 3 3 / 3

✓ **+ 1 pts** *B: Singly Linked List with Tail*

✓ **+ 1 pts** *C: Doubly Linked List with Tail*

✓ **+ 1 pts** *D: Circular Singly Linked List without Tail*

Extra answers

- 1 pts One extra answer

- 2 pts Multiple extra answers

QUESTION 5

5 Stacks - Diagramming 10 / 10

✓ **+ 10 pts** *Completely correct*

![Screen_Shot_2023-02-09_at_5.13.00_PM.png](/files/0a88dc8a-9920-404e-b115-9a1f3866f530)

Partial credit

+ 3 pts Has at least one pop() in the first two blanks

+ 3 pts Has at least one pop() in the second two blanks

+ 4 pts Has two pop()s in the second two blanks

If they have more than 3 pop()s

- 3 pts Has 4 pop()s in total

- 6 pts Has 5 pop()s in total

- 10 pts Has 6 pop()s in total

QUESTION 6

6 Binary Search Trees - Diagramming 10 / 10

`add(53)`

✓ **+ 5 pts** *Completely Correct:*

![add53.png](/files/a79448c5-6c76-43e5-86fa-cfeb3d56ac3c)

+ 1 pts Partial: BST has correct shape property

+ 1 pts Partial: BST has correct order property

+ 1 pts Partial: `53` is added to the final tree & no other data is added or removed

`remove(72)`

✓ **+ 5 pts** *Completely Correct:*

![remove72.png](/files/bf14b8b8-80bf-4255-81ad-db2dcd24f05e)

+ 1 pts Partial: BST has correct shape property

+ 1 pts Partial: BST has correct order property

+ 1 pts Partial: `72` is removed from the final tree, & no other data has been added or removed

+ 0 pts Incorrect/ No answer

QUESTION 7

7 Binary Tree Traversal - Diagramming 10 / 10

✓ **+ 10 pts**

![Traversal_Answer_1.png](/files/e0656d19-b12b-45c9-9bc5-b1c5da908fbc)Correct

Partial Credit

+ 8 pts 8 nodes have correct data

+ 6 pts 6+ nodes have correct data

+ 4 pts 4+ nodes have correct data

+ 2 pts 2+ nodes have correct data

Wrong Traversal (DO NOT COMBINE WITH PARTIAL)

+ 6 pts Completely Correct In order traversal

+ 6 pts Completely Correct Post order traversal

+ 6 pts Completely Correct Level order traversal

+ 0 pts Incorrect/No Answer

QUESTION 8

8 Circular Singly Linked List - Coding 8.5 / 10

Exceptions

✓ **+ 1 pts** *Some exception thrown*

✓ **+ 1 pts** *`IllegalArgumentException` thrown when data is `null`*

New Node

✓ **+ 1 pts** *Creates a new Node in **all the cases the code explores***

+ 0.5 pts Creates a new Node in **some of the cases the code explores**

Edge case `size == 0`

✓ **+ 1 pts** *Attempts to handle the empty list case (has an if statement with a correct conditional)*

+ 0.5 pts Sets head's next to itself **by doing `head.next = head;`**
(**only if** they attempt to handle the empty list case)

✓ **+ 0.5 pts** *Sets head's data to the passed in data (**only if** they attempt to handle the empty list case)*

General case (O(1) implementation **only**)

✓ **+ 1 pts** *Sets node's data to head's data*

✓ **+ 1 pts** *Sets the new node's next to head.next*

✓ **+ 1 pts** *Sets head's next to new node*

✓ **+ 1 pts** *Sets head's data to passed data*

O(n) Implementation

+ 1 pts Correctly iterates to node at **size - 1**

+ 1 pts Correctly adds node to the front of the List **for all cases this code explores **

Size

✓ **+ 1 pts** *Increments size in all of the cases the code explores*

+ 0.5 pts Increments size in some of the cases that the code explores

✓ **- 1 pts** *Minor error*

- 1 pts Syntax error

- 1 pts Throws a `NullPointerException`

- 2 pts Loses data

- 1.5 pts Adds to the back instead of adding to the Front

1

QUESTION 9

9 ArrayList - Coding 16 / 16

✓ **+ 0.5 pts** *Exception thrown when data1 == null or data2 == null*

✓ **+ 0.5 pts** *Correct exception thrown (IllegalArgumentException)*

✓ **+ 0.5 pts** *Attempt to loop through the backingArray*

✓ **+ 1 pts** *No IndexOutOfBoundsException will occur when attempting to traverse the backingArray*

✓ **+ 2 pts** *Correctly loops through the array*

✓ **+ 0.5 pts** *Attempt to shift data*

✓ **+ 3 pts** *Correctly shifts data (no overwriting/loss of data)*

✓ **+ 1 pts** *Attempt to add data1 and data2 to the backingArray*

✓ **+ 1 pts** *Correctly add data1 to index 2*

✓ **+ 1 pts** *Correctly add data2 to index 3*

✓ **+ 2 pts** *Increment size by 2*

✓ **+ 1.5 pts** *Time Efficient (No extra looping)*

✓ **+ 1.5 pts** *Space Efficient (No extra array created since we're ignoring resize case)*

- 1 pts Minor error

- 1 pts Syntax error

+ 0 pts Incorrect/No answer

QUESTION 10

10 Bonus - Animal Drawing **1 / 0**

✓ **+ 1 pts** *There is an animal*

+ 0 pts There is not an animal

QUESTION 11

11 Bonus 2 **0 / 0**

+ 1 pts Correct

✓ **+ 0 pts** *incorrect*

CS 1332 Exam 1 - Version A1
Spring Semester 2023 - February 8, 2023

Name (print clearly including your first and last name): AARYAN POTDAR

Section (8:25 am - A, 9:30 am - B, 2:00 pm D, 3:30 pm - C, Online - O): B

Signature: Aarp

GT account username (msmith3, etc): apotdar31

GT account number (903000000, etc): 903795148

- You must have your BuzzCard or another form of identification on the table in front of you during the exam. It is your responsibility to have your ID prior to beginning the exam.
- You are not allowed to leave the exam room and return. If you leave the room, then you must turn in your exam as complete. If you need to use the restroom, ask a TA to escort you.
- Signing and/or taking this exam signifies you are aware of and in accordance with the Academic Honor Code of Georgia Tech and the Georgia Tech Code of Conduct.
- Notes, books, calculators, phones, laptops, smart watches, headphones, etc. are not allowed. Extra paper is not allowed. If you have exhausted all space on this test, talk with your instructor. There are extra blank pages in the exam for extra space.
- If you plan on using ear plugs (foam or silicone, NOT AirPods) during the exam, you must show them to the instructor for approval.
- Pens/pencils and erasers are allowed. Do not share.
- All work entered on this exam, whether code, diagrams or multiple choice, must be implemented as was presented in lecture and recitation.
- If the second to last word of this sentence is circled, you will receive a bonus point.
- 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 the data 5 times when once would be sufficient, then this also is considered poor efficiency even though both are O(n).
- If you brought a duck with you to the exam, you may silently consult with it at any time.
- All code must be in Java.

Table of Contents

1) Efficiency - Multiple Choice [14 points]

For each of the operations listed below, determine the time complexity of the operation as it pertains to the data structure. Select the bubble corresponding to your choice in the space provided, and completely fill in the bubble. Unless otherwise stated, assume the ***worst-case*** time complexity. However, make sure you choose the tightest Big-O upper bound possible for the operation. **Do not use an amortized analysis for these operations *unless* otherwise specified.**

A) Adding to the front of an ArrayList where the size is strictly less than the capacity of the backing array.

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

B) Removing the most recently added element from an Linked List-backed Stack.

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

C) Constructing a BST by repeatedly calling the add method using elements from an array of *n* unsorted elements.

○ O(1) ○ O(log n) ○ O(n) ● O(n log n) ● $O(n^2)$ ← ANS

↳ ignore

D) Removing the third-to-last node from a Doubly Linked List with a tail pointer and size of at least 3.

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

E) Deleting an element at any index in an array.

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

F) Accessing the largest element in a BST.

○ O(1) ● ~~O(log n)~~ ignore ● O(n) ↳ ANS ○ O(n log n) ○ $O(n^2)$

G) Removing from the back of a Singly Linked List with a tail pointer.

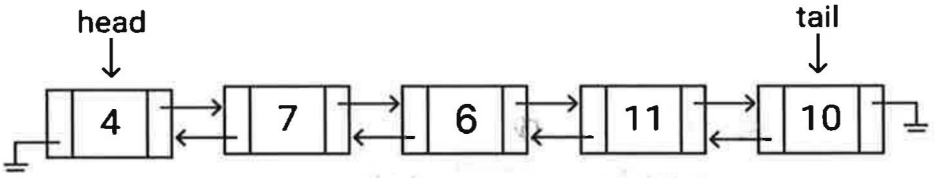
○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ $O(n^2)$

2) Doubly Linked List - Tracing [10 points]

Trace the execution of the mystery method when given the **TAIL of the doubly linked list** diagrammed below. Write the printed output of the method in the indicated output box. Write each print statement's output on a new line.

```
private class Node {
    private int data;
    private Node next;
    private Node prev;
    // constructors & class methods not shown
}

public static void mystery(Node curr) {
    if (curr == null) {
        System.out.println("Bingo!");
    } else {
        if (curr.data % 5 == 0) {
            mystery(curr.prev.prev);
        } else if (curr.data % 2 == 0) {
            mystery(curr.next);
        } else {
            mystery(curr.next.next);
        }
        System.out.println(curr.data);
    }
}
```



Printed output of `mystery(tail)`:

Bingo!
11
6
10

3) Scenarios - Multiple Choice [10 points]

For each of the scenarios listed below, determine the **best** choice for the data structure. Select the bubble corresponding to your choice in the space provided, and completely fill in the bubble.

A) Imagine you are programming a CPU scheduler which receives tasks and decides which task to run next in a FIFO order. Which data structure would you use to store the tasks?

○ ArrayList ● Array-backed Queue ○ SLL-backed Stack ○ Array

B) Imagine you are working on a text editor, and you want to implement the functionality to undo the last change you just made. Which data structure would you use to undo the change in the editor?

○ Binary Tree ○ ArrayList ○ SLL-backed Queue ● Array-backed Stack

C) Imagine you are writing a file system for an operating system. In this file system, you can have nested files and folders. There is no limit to how many files/folders can be nested within each other. Which data structure would you use?

○ Stack ○ Queue ● Tree ○ ArrayList

D) Imagine you are developing a network router. Your router can send 10 messages at a time. While the last group of data segments are being sent, you need to store incoming messages in a fixed capacity of 10. Which data structure should you use?

○ SLL-backed stack ● Array ○ Singly Linked List with tail ✗ Queue

Answer

→ ignore

E) Imagine you are in a long line at Piggly Wiggly where people join at the end and exit at the front. While in the line you start to notice that a lot of people at the very end of the line are beginning to get annoyed and also leave the line. What data structure best represents this situation?

○ Stack ○ SLL-backed Queue ● Deque ○ Array-backed Queue

4) Data Structure Properties - Multiple Select [10 points]

You are given a list of data structures on the right column. For each question, fill the squares corresponding to every data structure with the properties given. More **than one answer** may be correct. **Please make sure you completely fill in the squares.**

1) May need to **resize** the underlying structure.

- [x] A
- [ ] B
- [ ] C
- [ ] D
- [x] E
- [ ] F
- [x] G
- [ ] H
- [ ] I

2) LIFO (Last In First Out) data structure.

- [ ] A
- [ ] B
- [ ] C
- [ ] D
- [x] E
- [x] F
- [ ] G
- [ ] H
- [ ] I

3) Can perform **addToFront()** in **O(1)** (including amortized).
(For this question, only consider **A, B, C, D**)

- [ ] A
- [x] B
- [x] C
- [x] D

A. ArrayList

B. Singly Linked List with Tail

C. Doubly Linked List with Tail

D. Circular Singly Linked List without Tail

E. Array-backed Stack

F. Linked List-backed Stack

G. Array-backed Queue

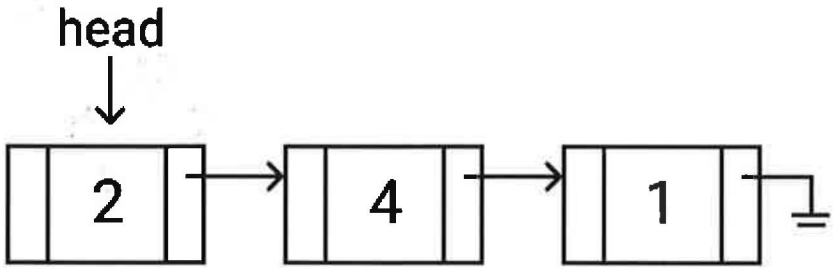
H. Linked List-backed Queue

I. Binary Search Tree

5) Stacks – Diagramming [10 points]

Shown below is the final state of a LinkedStack (implemented as shown in class) and a sequence of push() operations. However, the pop() operations are missing! Write "pop()" on EXACTLY three of the six blanks provided so that the resulting sequence of operations, when performed on an **initially empty** stack, produces the stack shown below. Do not write any new "push()" operations.

Final Stack:



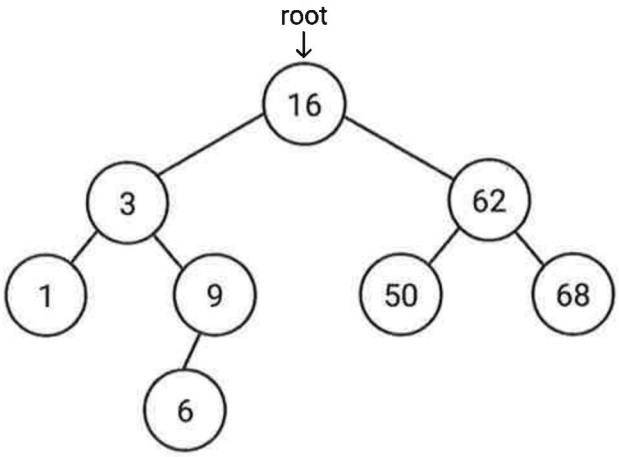
Operations (write "pop()" on three of the six blanks):

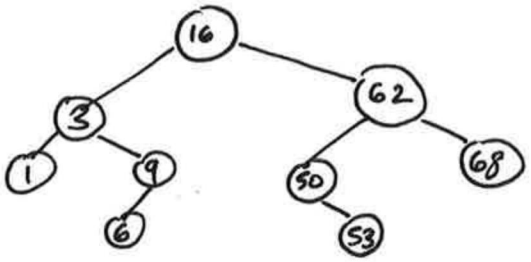
```
push(1);
push(2);
pop();
_____
_____
push(3);
push(4);
pop();
pop();
push(4);
push(2);
_____
_____
```

6) Binary Search Trees – Diagramming [10 points]

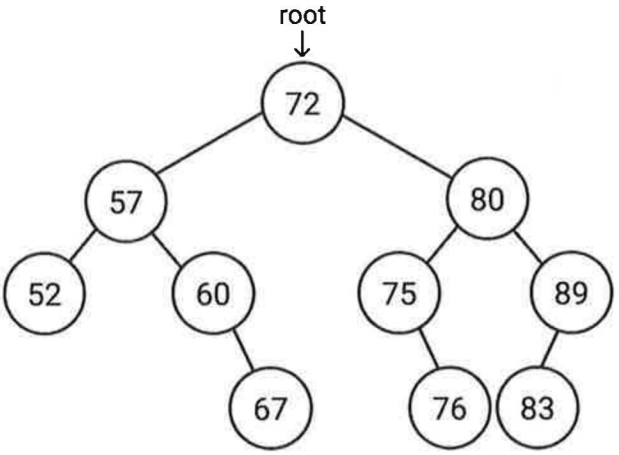
Given the following initial BSTs in the left column below. Perform the stated operation, add or remove, for each tree. Draw the resulting BST in the right column. If you want, you can draw multiple steps (**circle the final step if you do so**). If necessary for any operation, use the **successor node**.

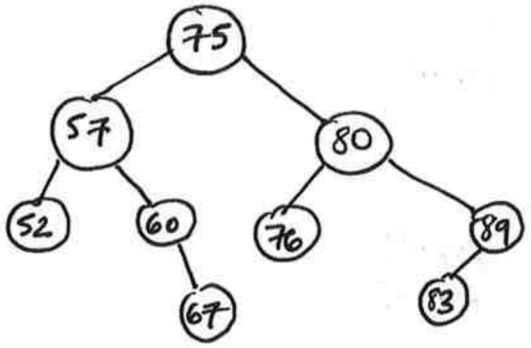
add(53)





***remove*(72)**





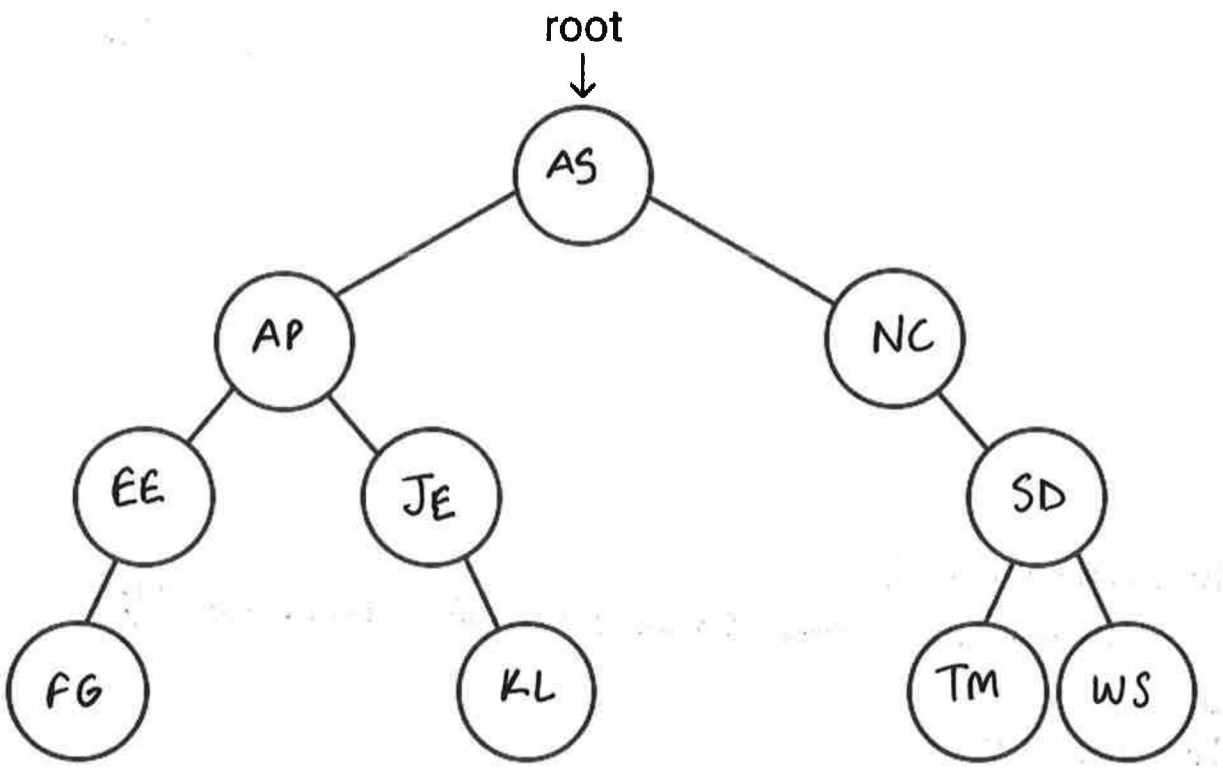
7) Binary Tree Traversal - Diagramming [10 points]

Below you are given the outline of a binary tree and the result of a **pre-order** traversal. Fill in the outline with the appropriate data to recreate the binary tree that produced this traversal.

Pre-order Traversal Output:

[AS, AP, EE, FG, JE, KL, NC, SD, TM, WS]

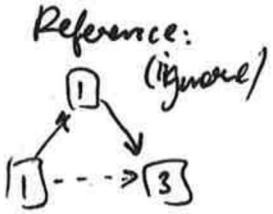
Fill in the tree:



8) Circular Singly Linked List - Coding [10 points]

Goal: Given the following **CircularSinglyLinkedList** class, implement the **addToFront(T data)** method.

Requirements: Since Node is an inner class, **you should access its fields directly** (e.g. `node.data`) instead of using getters/setters. **Your code should be as efficient as possible.** You may **not** assume any other method in the CircularSinglyLinkedList class is implemented. Assume all necessary imports are made.




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

     private Node<T> head;
     int size;

     /* Adds data to the front of the Circular Singly Linked List.
      *
      * @param  data - data to be added
      * @throws java.lang.IllegalArgumentException if data is null
      */
     public void addToFront(T data) {
      // YOUR CODE GOES HERE
           if (data == null) {
                throw new IllegalArgumentException("Data is null");
           }
           if (size == 0) {
                head = new Node<T>(data, head.next);
                size ++;
           } else {
                Node<T> temp = new Node<T>(head.data, head.next);
                head.next = temp;
                head.data = data;
                size ++;
           }
      }
```

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

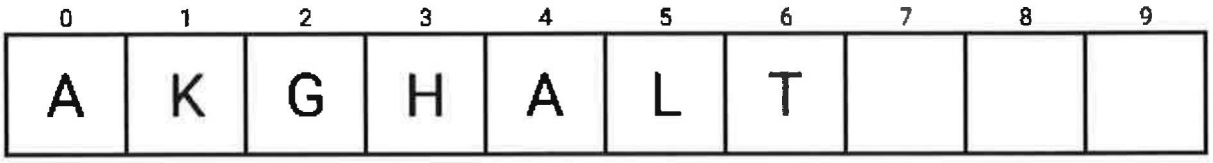
9) ArrayList - Coding [16 points]

Goal: Given a basic class for an **ArrayList**, write an **addAt2And3(T data1, T data2)** method. As the name suggests, this method should add the 2 provided data into the arraylist at index = 2 and index = 3. The Arraylist is guaranteed to **ALWAYS** have enough **space** for 2 more data (i.e. **ignore the resize case**), and there are **ALWAYS** at least 2 data in the Arraylist already.

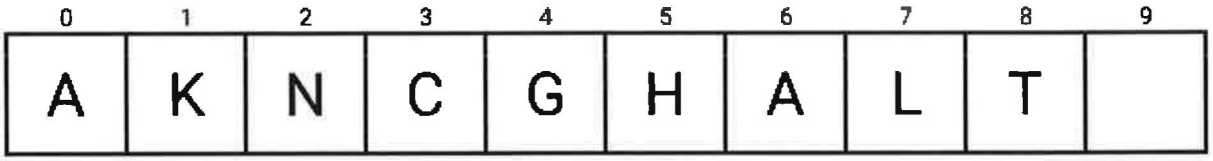
Requirements: This method should run in O(n) time and be as efficient as possible. **Do NOT assume any methods other than the constructor are implemented.** Assume all necessary imports are made.

Example: This is an example with an ArrayList of type String.

Initial ArrayList:



After calling addAt2And3("N", "C"):




```
public class ArrayList<T> {
     T[] backingArray;
     int size;

     public ArrayList() {
          backingArray = (T[]) new Object[INITIAL_CAPACITY];
          size = 0;
     }

     // METHOD ON NEXT PAGE
```

```
    /* Add data into index 2 and index 3
     *
     * @param data1 - data to be added at index 2
     * @param data2 - data to be added at index 3
     * @throws java.lang.IllegalArgumentException if data1 or data2 are null
     */
    public void addAt2and3(T data1, T data2) {
     // YOUR CODE GOES HERE
            if (data1 == null || data2 == null) {
                throw new IllegalArgumentException ("Cannot add
                        null data");
            }
            if (size == 2) {
                backingArray [2] = data1;
                backing_Array [3] = data2;
                size += 2;
            } else {
                for (i= size-1; i >=2; i--;){
                    backingArray [i+1] = backingArray [i];
                    backingArray [i+2] = backingArray [i+1];
                }
                size += 2;
                backingArray [2] = data1;
                backingArray [3] = data2;
            }
    } // END OF METHOD
} // END OF ARRAYLIST CLASS
```

10) Bonus - Animal Drawing [1 point]

For **1** extra point, please draw your favorite animal in the box below. You may also give it a name. Inappropriate content will **not** receive credit.

For inspiration, here are some of our favorites 🐶🦆🦘🐼🐐🐿️🐨!