CS1332 Campus Fall 2022 Exam 1 Version A

Thomas Nam Minh Le

TOTAL POINTS

91.5 / 100

QUESTION 1

Efficiency 14 pts

1.1 A 2 / 2

✓ **+ 2 pts Correct: $$O(1)$$**

+ 0 pts Incorrect: $$O(n)$$

1.2 B 0 / 2

+ 2 pts Correct: $$O(n)$$

✓ **+ 0 pts Incorrect: $$O(1)$$**

+ 0 pts Incorrect: $$O(\log{n})$$

+ 0 pts Blank

1.3 C 2 / 2

✓ **+ 2 pts Correct: $$O(n)$$**

+ 0 pts Incorrect: $$O(\log{n})$$

+ 0 pts Incorrect: $$O(1)$$

+ 0 pts Incorrect: $$O(n \log{n})$$

+ 0 pts Incorrect: $$O(n^2)$$

+ 0 pts Blank

1.4 D 2 / 2

✓ **+ 2 pts Correct: $$O(n)$$**

+ 0 pts Incorrect: $$O(1)$$

+ 0 pts Incorrect: $$O(n^2)$$

+ 0 pts Incorrect: $$O(n \log{n})$$

+ 0 pts Incorrect: $$O(\log{n})$$

+ 0 pts Blank

1.5 E 2 / 2

✓ **+ 2 pts Correct: $$O(n)$$**

+ 0 pts Incorrect: $$O(\log{n})$$

+ 0 pts Incorrect: $$O(1)$$

+ 0 pts Incorrect: $$O(n \log{n})$$

+ 0 pts Incorrect: $$O(n^2)$$

+ 0 pts Blank

1.6 F 2 / 2

✓ **+ 2 pts Correct: $$O(1)$$**

+ 0 pts Incorrect: $$O(\log{n})$$

+ 0 pts Incorrect: $$O(n)$$

1.7 G 0 / 2

+ 2 pts Correct: $$O(n)$$

✓ **+ 0 pts Incorrect: $$O(1)$$**

+ 0 pts Incorrect: $$O(n^2)$$

+ 0 pts Incorrect: $$O(\log{n})$$

+ 0 pts Incorrect: $$O(n\log{n})$$

+ 0 pts Blank

QUESTION 2

Tree Shape Properties MC 10 pts

2.1 1 2 / 2

First tree

- 1 pts Complete tree box checked

✓ **+ 2 pts Full tree box checked**

- 1 pts Degenerate tree box checked

+ 0 pts blank

2.2 2 6 / 6

Second tree

✓ **+ 3 pts Complete tree box checked**

✓ **+ 3 pts Full tree box checked**

- 2 pts Degenerate tree box checked

+ 0 pts blank

2.3 3 2 / 2

Third tree

- 1 pts Complete tree box checked

- 1 pts Full tree box checked

✓ **+ 2 pts Degenerate tree box checked**

QUESTION 3

Binary Tree Traversal MC 10 pts

3.1 Pre 5 / 5

✓ **+ 5 pts** **pre-order B**

+ 0 pts incorrect

3.2 Post 5 / 5

✓ **+ 5 pts** **Correct C**

+ 0 pts Incorrect

QUESTION 4

Stacks and Queues Inference MC 12 pts

4.1 A 4 / 4

A)

✓ **+ 4 pts** **Correct: D might be a Stack**

+ 0 pts Incorrect

4.2 B 4 / 4

B)

✓ **+ 4 pts** **Correct: D is not a Stack or a Queue**

+ 0 pts Incorrect

4.3 C 4 / 4

C)

✓ **+ 4 pts** **Correct: D might be a Queue**

+ 0 pts Incorrect

QUESTION 5

5 Deques Diagramming 10 / 10

✓ **+ 10 pts** **Fully Correct:**

`[-,-,-,-,-,TS]`

F

Partially Correct

+ 2 pts Front is at index 5

+ 2 pts Final Deque contains `TS`

+ 2 pts Final Deque doesn't contain `HW`

+ 1.5 pts Final Deque doesn't contain `KJ`

+ 1 pts Final Deque doesn't contain `NK`

- 3 pts Backing Array is not drawn

- 1 pts Front Pointer is not drawn

- 1 pts Backing Array has incorrect length

+ 0 pts Incorrect/No Answer

QUESTION 6

6 BST Diagramming 8 / 10

add(12)

✓ **+ 5 pts** **Completely correct**

![Screenshot_2022-09-22_at_4.58.17_PM.png](/files/6b3bfac0-d37a-471f-8804-8c996b161a73)

+ 1 pts Answer is a valid BST (Shape and order property satisfied)

+ 1 pts `12` is added to the final tree

+ 1 pts `12` is a leaf

+ 1 pts Other than `12`, no data is added or removed

remove(34)

+ 5 pts Completely correct

![Screenshot_2022-09-22_at_4.59.28_PM.png](/files/09fcd115-5ab3-4b6f-b43c-7065fa8af45a)

✓ **+ 1 pts** **Answer is a valid BST (shape and order property satisfied)**

✓ **+ 1 pts** **`34` is removed from the tree**

+ 1 pts `34` is is replaced with its successor (`36`)

✓ **+ 1 pts** **Other than `34`, no data is added or removed from the tree**

+ 0 pts Incorrect/No Answer

- 1 pts ERROR: Is not a BST

QUESTION 7

7 CSLL Recursion Tracing 9 / 9

✓ **+ 9 pts** **Completely Correct**

![Screen_Shot_2022-09-

21_at_6.21.14_PM.png](/files/1e2b9aac-8d09-4c29-b88f-5a255b7bdef5)

+ **1 pts** Attempted recursion

ROW 2

+ **3 pts** Row Correct: **[3, 3, 8, 7]** (only choose this if totally correct)
+ **0.5 pts** Current Index: **3**
+ **0.5 pts** Index to Modify: **3**
+ **1.5 pts** Current Temp: **8**
+ **0.5 pts** Returned Temp: **7**

ROW 3

+ **2 pts** row correct: **[4, 4, 9, 8]** (only choose this if totally correct)
+ **0.5 pts** Current Index: **4**
+ **0.25 pts** Index to Modify: **4**
+ **1 pts** Current Temp: **9**
+ **0.25 pts** Returned Value: **8**

ROW 4

+ **2 pts** Row Correct: **[0, X, 10, 10]** (only choose this if totally correct)
+ **0.5 pts** Current Index: **0**
+ **0.5 pts** Index to Modify: **X** or **blank**
+ **0.75 pts** Current Temp: **10**
+ **0.25 pts** Returned Value: **10**

Final Linked List State:

+ **1 pts** Completely Correct: **[2, 8, 7, 8, 10]**
+ **0.75 pts** Linked List is correct based on incorrect **_return values:_**
- only award if `list[0] = 2` **and** `list[1] = 8`

How to check:
- index 0 = 2
- index 1 = 8
- index 2 = row 2 returned value
- index 3 = row 3 returned value
- index 4 = row 4 returned value

+ **0.75 pts** Linked list is correct based on incorrect **_recursive calls:_**
- all **_index to modify_** indicies are now equal to the returned value from the row below

+ **0.75 pts** Linked list is correct if:

```
for (row x: table) {
int data = x.returned_value;
linked_list[x.index_to_modify] == data;
}
```

+ **0 pts** Did not do the problem

QUESTION 8

8 ArrayLists Coding 12 / 15

Partial

✓ **+ 2 pts Throws IOBE if `index < 0 || index >= size`**
✓ **+ 1 pts Attempts to loop over the backingArray**
✓ **+ 2 pts Accesses elements after and including `backingArray[index]` in the loop**
✓ **+ 1 pts Attempts to shift elements**
✓ **+ 2 pts Correctly shifts elements (does not overwrite data when shifting)**
✓ **+ 2 pts returns the **correct** data from the list**
✓ **+ 2 pts Decrements size**

+ **3 pts** Final `backingArray` is correct

- **3 pts** Efficiency space or time
- **2 pts** Code throws IndexOutOfBoundsException while looping (access of the `backingArray[size]`)
- **1 pts** Minor error
- **1 pts** Syntax
+ **0 pts** Incorrect/No answer

QUESTION 9

9 DLL Coding 9.5 / 10

✓ **+ 1 pts Throws IAE if `data == null`**
✓ **+ 2 pts Creates a new node with the parameterized data**

Empty List Case

✓ **+ 2 pts Sets head to new node**
✓ **+ 1 pts Sets tail to new node**

General Case

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

O(n) Implementation

+ **0.5 pts** Attempts to traverse through list with a loop

+ **0.5 pts** Attempts to add a new node with `data` into the list

+ **1 pts** Correct O(n) implementation

✓ **+ 1 pts** **Increments size**

+ **0 pts** Incorrect / No answer

✓ **- 0.5 pts** **Syntax Error**

- 0.5 pts Minor Error

1 Remember generics!

QUESTION 10

10 Bonus 1 / 0

✓ **+ 1 pts** **Correct**

+ **0 pts** Blank/No answer

CS 1332 Exam 1 - Version A
Fall Semester 2022 - September 21, 2022

Name (including first and last name): Thomas Le

Signature: Thomas Le

GT account username (msmith3, etc): thomas30

GT account number (903000000, etc): 903 696 568

- You must have your BuzzCard or other 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 for any reason, then you must submit your exam as complete. (If you need to use the restroom for an emergency, then a TA will 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, earbuds, etc. are not allowed. Extra paper is not allowed. If you have exhausted all space on this exam, 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.
- If you brought a duck with you to the exam, 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 / module videos and recitation.
- All code must be in Java.
- Efficiency matters. For example, if you code something that uses O(n) time or worse when there is an obvious 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).
- Comments are not required unless a question explicitly asks for them.

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

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 zero.

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

B) Removing the first element that was added to an array-backed Deque.

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

C) Computing the height of a BST.

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

D) Removing the third smallest element from a Doubly Linked List with a tail pointer.

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

E) Accessing the largest element in a binary tree. Degenerate Tree Imbalanced Worst Case

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

F) Removing an element from a linked list-backed Stack. Removing final element?

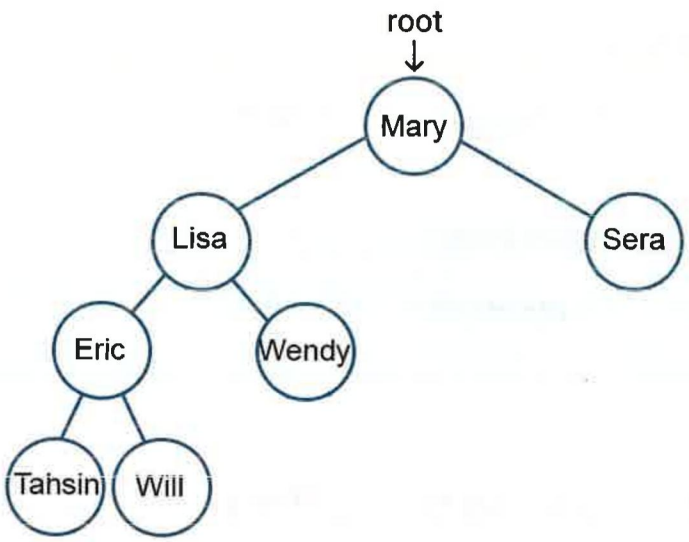
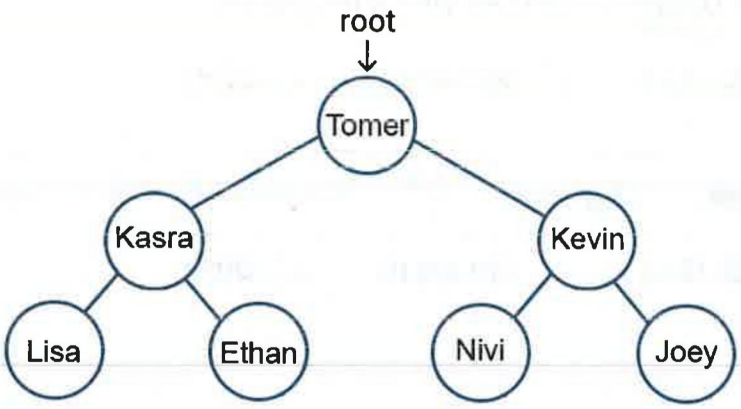
● O(1) ○ O(log n) ○ O(n) ○ O(n log n) ○ $O(n^2)$

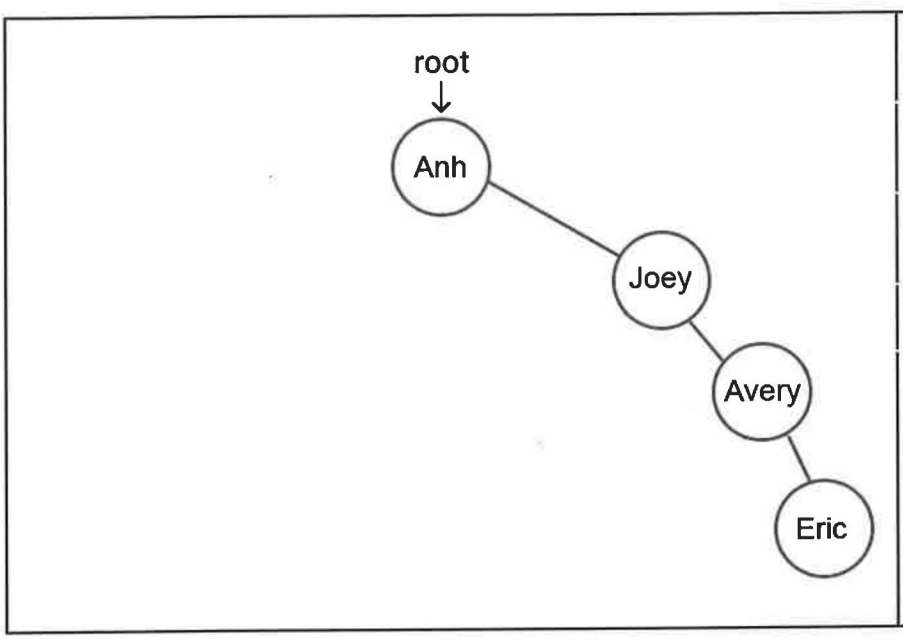
G) Removing an element from a linked list-backed Queue and adding it to an array-backed Stack.

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

2) Tree Shape Properties – Multiple Choice [10 points]

For each of the given trees, **select all** shape properties that apply to the tree. There are 3 trees to identify.

Given Tree	Tree Shape (Select all that apply)
	☐ Complete Tree ☑ Full Tree ☐ Degenerate Tree
	☑ Complete Tree ☑ Full Tree ☐ Degenerate Tree



☐ Complete Tree

☐ Full Tree

☑ Degenerate Tree

3) Binary Tree Traversal - Multiple Choice [10 points]

Given the following results of a pre-order and post-order traversals, **select a binary tree** which yields the resulting list for each corresponding traversal.

Pre-order traversal result:

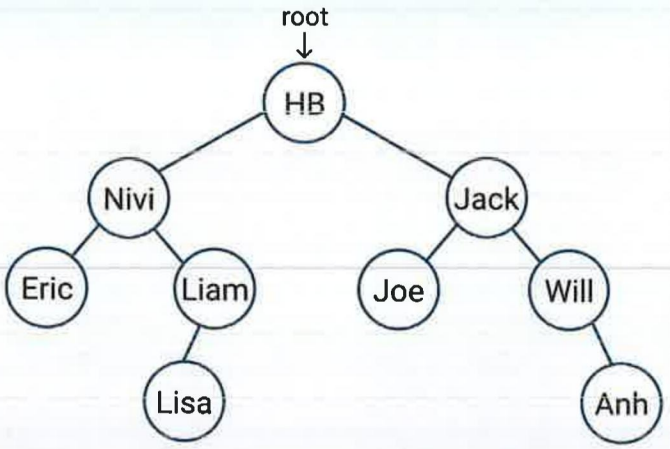
[HB, Jack, Nivi, Will, Lisa, Anh, Liam, Eric, Joe]

Post-order traversal result:

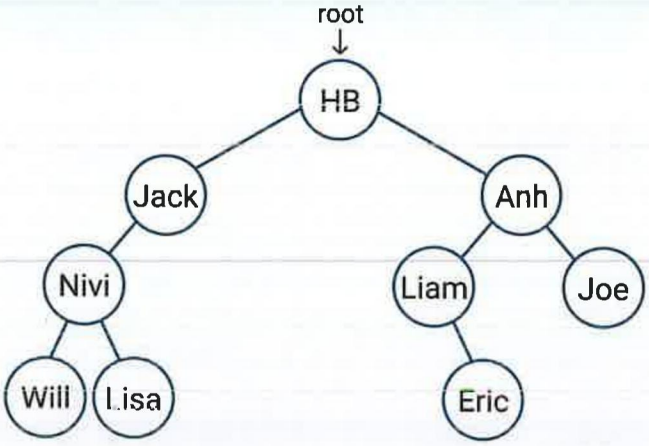
[Liam, Lisa, Eric, Anh, Nivi, Joe, Jack, Will, HB]

Corresponding Tree: B	**Corresponding Tree:** C

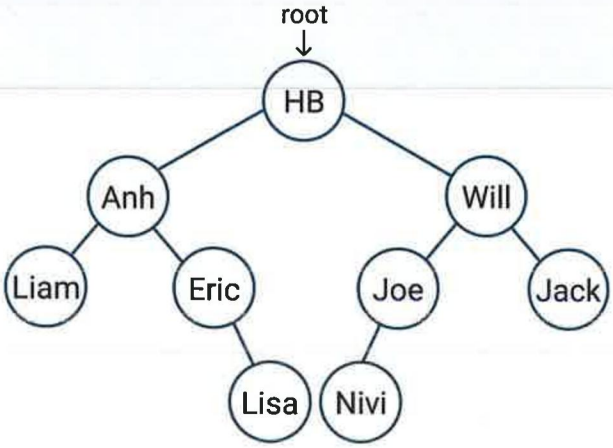
Tree A:



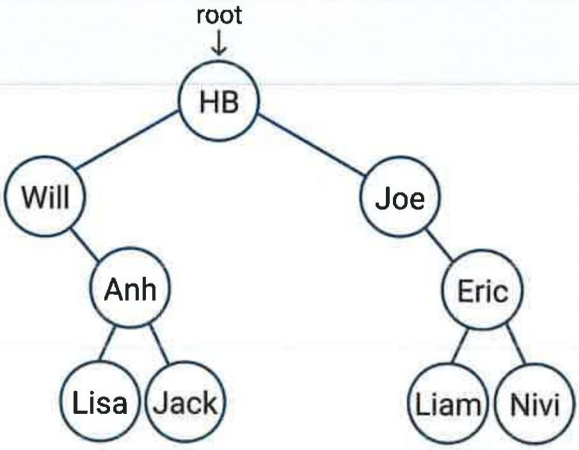
Tree B:



Tree C:



Tree D:



4) Inference - Multiple Choice [12 points]

You are given an empty data structure **D**, but you do not know whether it is a stack, a queue, or some other structure. You perform the sequence of operations listed below in order. Given the data remaining in **D** after these operations, deduce the type of **D**.

Operations:

```
D.add(1);
D.add(4);
D.add(7);
D.remove();
D.add(3);
D.add(5);
D.remove();
```

1 4 7 3 5

X 4 7 3 5

1 4 7

A) Suppose that the following data remains in **D** (not necessarily in this order): **1 4 3**. Which statement is true?

(●) **D** might be a Stack ( ) **D** might be a Queue ( ) **D** is not a Stack or a Queue

B) Suppose that the following data remains in **D** (not necessarily in this order): **4 7 5**. Which statement is true?

( ) **D** might be a Stack ( ) **D** might be a Queue (●) **D** is not a Stack or a Queue

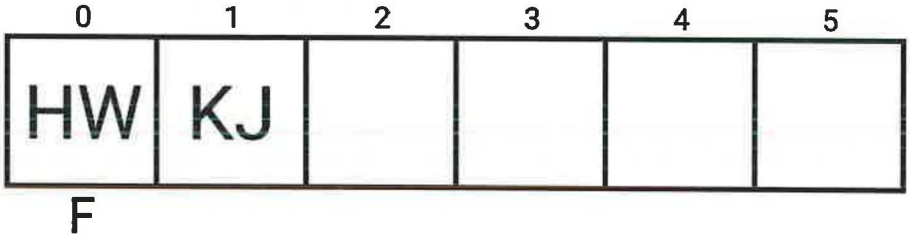
C) Suppose that the following data remains in **D** (not necessarily in this order): **7 3 5**. Which statement is true?

( ) **D** might be a Stack (●) **D** might be a Queue ( ) **D** is not a Stack or a Queue

5) Deques – Diagramming [10 points]

Given the following partially-filled Deque, **backed by an array**, perform the following operations. Draw the ***final state*** of the Deque, including an "F" for where the front of the Deque would be.

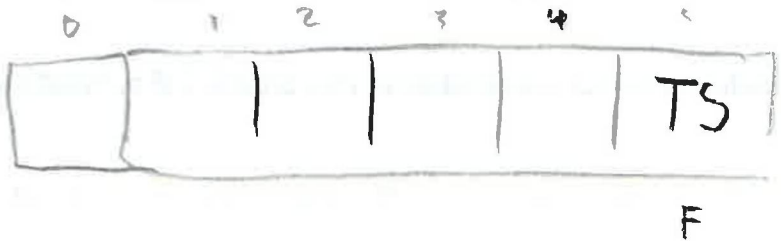
Initial Queue:



Operations:

```
addFirst("TS");
addFirst("NK");
removeLast();
removeLast();
removeFirst();
```

Final Deque:

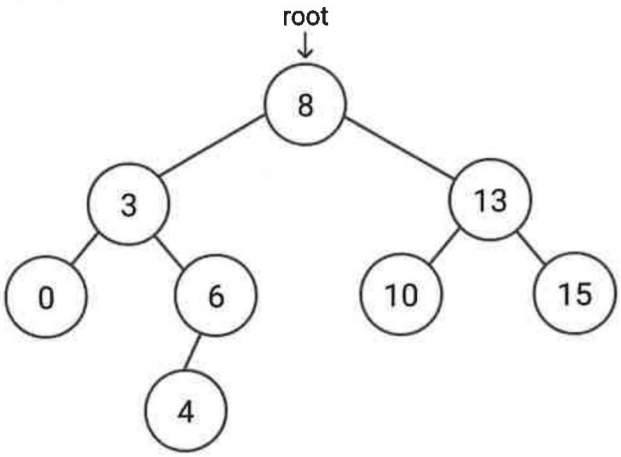


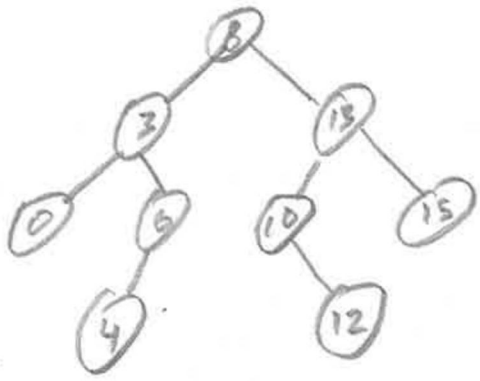
Do not write below this line. It may not be scanned, therefore it will not be graded.

6) Binary Search Trees – Diagramming [10 points]

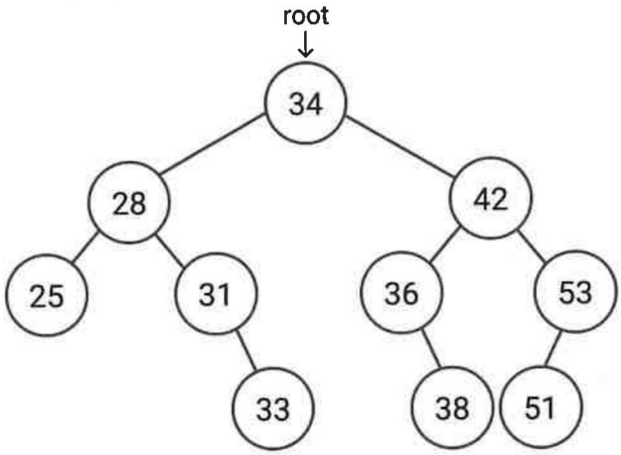
Given the following initial BSTs in the left column below. Perform the stated operation, add or remove, for each tree. Draw the resulting BST in the right column. If you want, you can draw multiple steps (**circle the final step if you do so**). If necessary for any operation, use the **successor node**.

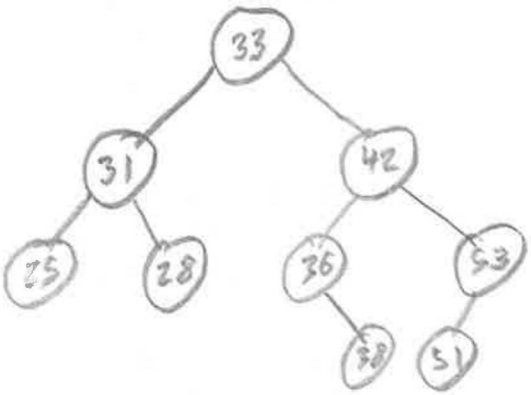
add(12)





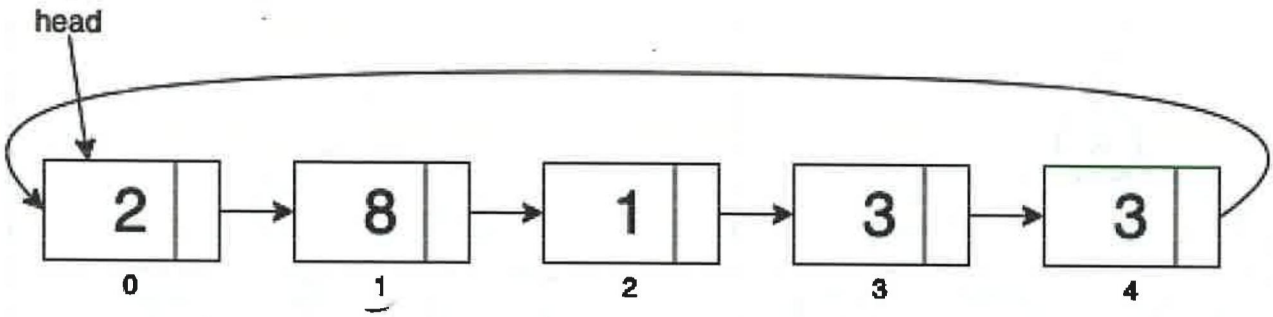
***remove*(34)**





7) Circularly Singly Linked List - Tracing [9 points]

Given the sample class (implementation omitted) trace the recursive **circularly singly linked list** method **mystery,** and fill in the table accordingly. After filling in the table, fill in the linked list at the bottom with the final state of the CSLL. The value returned to x is not important, do not write it. **For clarification, the first line of the table has been filled in for you.**




```
public class doMystery {
      private class Node {
            //implementation omitted
      }
      private Node head;
      public int mystery(Node curr, int temp) {
            if (curr.equals(head)) {
                  return temp;
            }

            if (curr.data % 2 == 0) {
                  curr.next.data = mystery(curr.next.next, temp + 1);
                  return temp + 1;
            }
            curr.data = mystery(curr.next, temp + 1);
            return temp - 1;
      }
}

int x = mystery(head.next, 7);

// TABLE AND BLANK LINKED LIST ON FOLLOWING PAGE
```

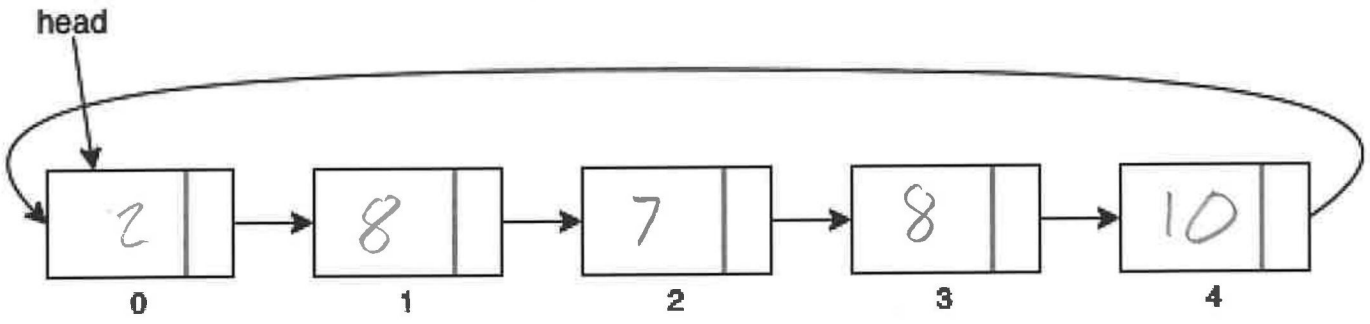
Fill in the table according to the labeled columns. For clarification, the first line has been filled in for you. If no index is modified, or any other column does not need a value, leave the box blank, or write an **X**. The **Current Temp** column should be filled with the value of the second input parameter named **temp** of the **mystery** method.

Notes:

- If the executed line is `curr.data = ...` , the index to modify will **match** the current index.
- If the executed line is `curr.next.data = ...` , the index to modify will be **current index + 1**

Current Index	Index to Modify	Current Temp	Returned Value
1	2	7	8
3	3	8	7
4	4	9	8
0	X	10	10

Fill in the final state of the Linked List below:



8) ArrayLists - Coding [15 points]

Goal: Given the following **ArrayList** class, implement the **removeAtIndex(int index)** method.

Requirements: **Your code should be as efficient as possible.** You may **not** assume any other method in the ArrayList class is implemented. You may assume that everything from java.util is imported.

```
public class ArrayList<T> {

    private T[] backingArray;
    int size;

   /**
    * Removes and returns the element at the specified index.
    *
    * @param index the index of the element to remove
    * @return the data formerly located at the specified index
    * @throws java.lang.IndexOutOfBoundsException if index < 0 or index >= size
    */
    public T removeAtIndex(int index) {
        // YOUR CODE HERE, USE THE NEXT PAGE IF NEEDED
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException("Index must be: [0, size)");
        } else {
            T removedData = backingArray[index];
            if (size == 1) {
                size = 0;
            } else {
                for (int i = index; i < size - 1; i++) {
                    backingArray[i] = backArray[i + 1];
                }
                size = size - 1;
            }
            return removedData;
        }
```

```
    } // END OF METHOD

} // END OF CLASS
```

9) Doubly Linked List - Coding [10 points]

Goal: Given the following **DoublyLinkedList** class, implement the **addToBack(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 DoublyLinkedList class is implemented.

```
public class DoublyLinkedList<T> {

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

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

   /**
    * Adds the data to the back of the list.
    *
    * @param data the data to add to the back of the list
    * @throws java.lang.IllegalArgumentException if data is null
    */
    public void addToBack(T data) {
        // YOUR CODE HERE, USE THE NEXT PAGE IF NEEDED
```

Has tail

```
if (data == null) {
    throw new IllegalArgumentException("Data can't be null");
} else {
    Node newNode;
    if (size == 0) {
        newNode = new Node(data, null, null);
        head = newNode;
        tail = newNode;
        size = 1;
    } else {
        newNode = new Node(data, tail, null);
```

// Continued on next page

```
        tail.next = newNode;
        tail = newNode;
        size = size + 1;
    }
```

```
    } // END OF METHOD

} // END OF CLASS
```

10) Bonus - Fill in the Blank [1 point]

For 1 extra point, please describe your favorite dessert (inappropriate content will not receive credit):

Cake tastes sweet!)

This page is blank beyond your Wildest Dreams.

(If you use this page for your work, please reference this page number on the question you are answering so we can find your response)