CS 1332 Practice Exam 1
Fall Semester 2018

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

Signature: ______________________________

GT account username (msmith3, etc): ______________________________

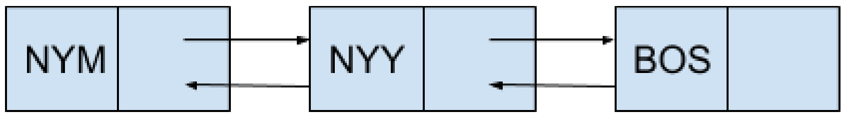
GT account number (903000000, etc): ______________________________

- You must have your BuzzCard or other form of identification on the table in front of you during the exam. When you turn in your exam, you will have to show your ID to the TAs before we will accept your 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 turn in your exam as complete.
- 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.
- Pens/pencils and erasers are allowed. Do not share.
- 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).
- Style standards such as (but not limited to) use of good variable names and proper indentation is always required. (Don't fret too much if your paper gets messy, use arrows or whatever it takes to make your answer clear when necessary.)
- Comments are not required unless a question explicitly asks for them.

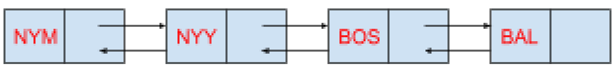
This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.

1) Linked Deque - Diagramming

The following Deque is backed with a Doubly Linked List. This deque has a head pointer and a tail pointer. addFirst() and removeFirst() calls on the Deque operate on the head of the Linked List. Likewise, addLast() and removeLast() operate on the tail of the Linked List. Perform the following *addFirst(), addLast(), removeFirst(), and removeLast()* operations in order on the deque starting from the initial deque. **Do this by filling in each node with the correct data following each function call.** As a default, do what you were asked to do in the homework. Pay careful attention to the order of the operations, they go left to right then down.

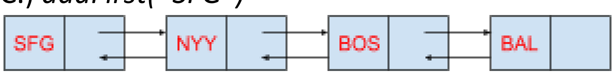


A.) *addLast("BAL")*



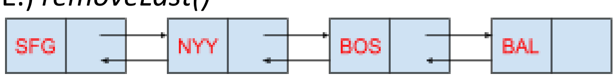
B.) *removeFirst()*

C.) *addFirst("SFG")*

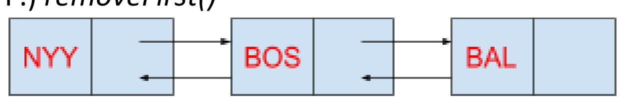


D.) *addLast("LAA")*

E.) *removeLast()*



F.) *removeFirst()*



2) Queue Diagram

The following queue is array backed with initial capacity of 5. This queue has a size variable and a front variable. This queue is implemented by removing from the front and adding to the back. Perform the following *enqueue* and dequeue operations in order on the queue starting from the initial queue (e.g. perform operation A on the initial queue, operation B on the resulting queue from part A, etc.). **Place the result below the operation and mark where the Front and Back pointers are at the end of the operation**. As a default, do what you were asked to do in the homework. Pay careful attention to the order of the operations, they go left to right then down.

Index	0	1	2	3	4
Markers		F		B	
Initial Queue		b	e		

A.) *dequeue()*

Index	0	1	2	3	4
Markers			F	B	
Queue			e		

B.) *dequeue()*

Index	0	1	2	3	4
Markers	F B				
Queue					

C.) *enqueue('d')*

Index	0	1	2	3	4
Markers	F	B			
Queue	d				

D.) *enqueue('a')*

Index	0	1	2	3	4
Markers	F		B		
Queue	d	a			

E.) *dequeue()*

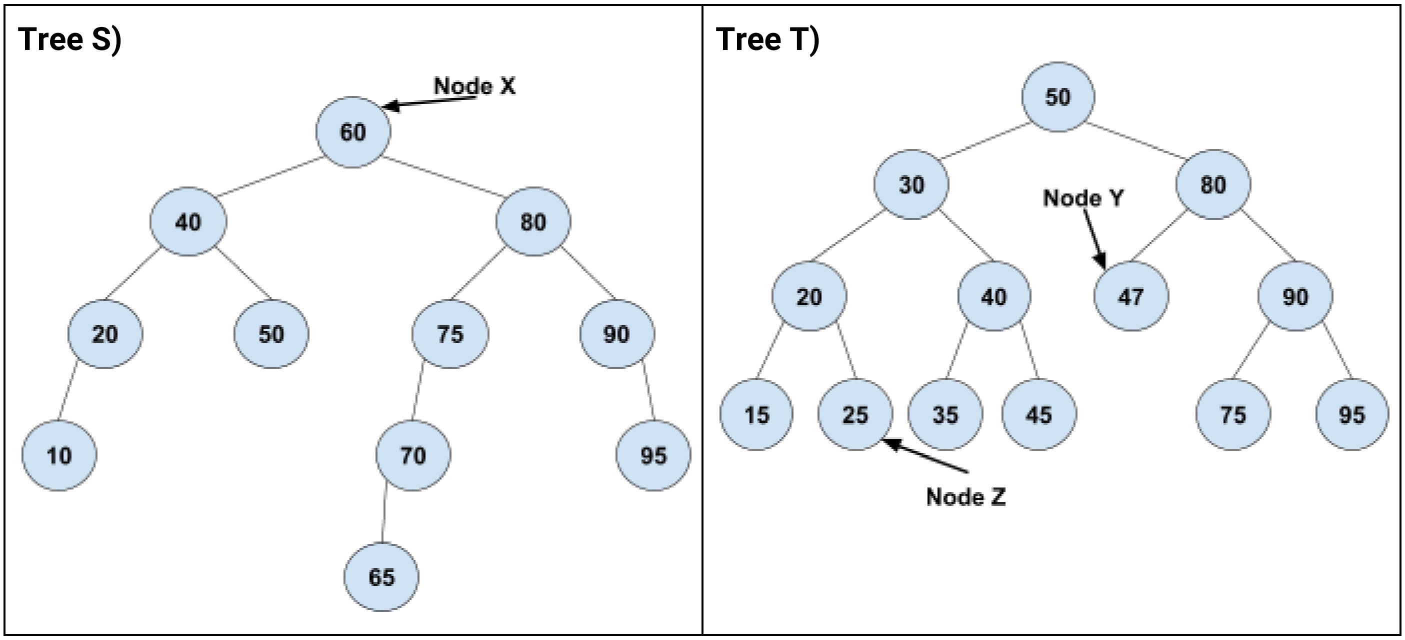
Index	0	1	2	3	4
Markers		F	B		
Queue		a			

F.) *enqueue*('p')

Index	0	1	2	3	4
Markers		F		B	
Queue		a	p		

3) BST - Diagram

Given the following two trees, answer the questions below.



A.) What is the height and depth of: **[3 points each]**

i.) **Node X** - Height: ___4___, Depth: ___0___

ii.) **Node Y** - Height: ___0___, Depth: ___2___

iii.) **Node Z** - Height: ___0___, Depth: ___3____

B.) Fill in the bubble to the left of your response:

i.) Which of the trees is a BST? **[2 points]**

○ Tree S ○ Tree T ○ Neither Tree ○ Both Trees

ii.) Which of the trees is complete? **[3 points]**

○ Tree S ○ Tree T ○ Neither Tree ○ Both Trees

C.) Is the given node a leaf or not a leaf? **[2 points each]**

i.) **Node X**: ______Not a leaf______

ii.) **Node Y**: _____Leaf______

Iii.) **Node Z:** _____Leaf______

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

4) Doubly Linked List

Given the following unordered five steps for a Doubly Linked List. Arrange the steps in a correct order that would correctly add *"CS 1332"* to the **BACK** of a Doubly Linked List without a tail pointer. There may be more than one correct answer. If you don't need all five steps, leave the extra spaces blank. Be careful to notice the differences between the bolded words.

A.) Access tail pointer.
B.) Set *newNode's* next pointer to null.
C.) Iterate to the last node of the Doubly Linked List.
D.) Create a new LinkedListNode with data "CS 1332" called *newNode*
E.) Set *newNode's* previous pointer to the last node in the list.
F.) Set last node's next pointer to *newNode.*
G.) Set tail pointer to *newNode*

Order:

Step 1: ________

Step 2: ________

Step 3: ________

Step 4: ________

Step 5: ________

Step 6: ________

Step 7: ________

There are multiple answers for this question. Two example answers have been provided:

- **C D F E B**
- **C D E F B**

A and G are not possible since there is no tail pointer. B is optional since the next pointer is null by default.

5) Efficiency - Matching

For each of the operations listed below, determine the time complexity of the operation. Select the bubble corresponding to your choice in the space provided. 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.

A.) Iterating over a Linked List using an Iterator.

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

B.) 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)

C.) Adding to the back of an ArrayList **without** a size variable.

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

D.) Adding to the end of a linked-list-backed deque.

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

E.) Accessing the data at index 2 of a Singly-Linked List of size at least 4.

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

Note: It is possible to do C in O(log n) time, but this is outside the scope of the class.

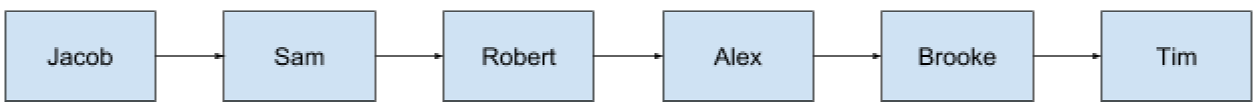
6) Iterator Tracing

Given the following `Iterator` code that iterates over the given linked list `list`, draw what the list `output` would look like following the execution.

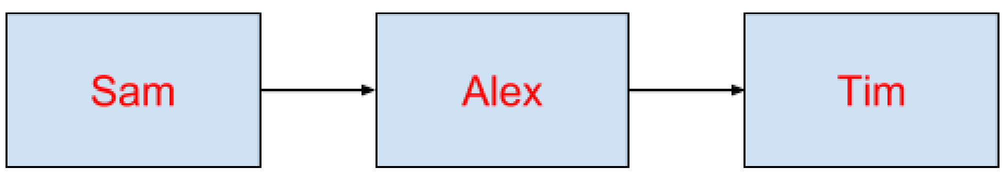
```
LinkedList<String> output = new LinkedList<>();
Iterator<String> it = list.iterator();
while (it.hasNext()) {
    String s = it.next();

    if (s.length() <= 4) {
        output.addLast(s)
    }
}
```

`list =`



`output =`



7) LinkedList - Coding

Given the following starter code, you are responsible for implementing the *replaceData()* method for the *LinkedList* class. You must not violate any of the rules for *LinkedList* that we've outlined in class, e.g. head should always point to the first elements. You must be as efficient as possible.

```
public class LinkedList {
    private Node head;

    private class Node {
        int data;
        Node next;
        public Node(int data, Node next) {
            this.data = data;
            this.next = next;
        }
    }
    /* implementation omitted */

    /**
     * Replace the data at the index with the given data.
     *
     */
    public void replaceData(int data, int index) {
        Node curr = head;
        int idx = 0;
        while (idx < index) {
            curr = curr.next;
            idx++;
        }
        curr.data = data;
    }
}
```

This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.