CS 1332 Exam 1A

Jordan Shartar

TOTAL POINTS

75.5 / 100

QUESTION 1

1 Recursion Tracing 11 / 15

✓ + 2 pts **Andrew printed**

✓ + 2 pts **THWg printed**

+ 2 pts THWg printed

✓ + 2 pts **Go Jackets printed**

✓ + 2 pts **GT printed**

+ 2 pts Nothing is printed outside of Andrew (x1), THWg (x2), Go Jackets (x1), GT (x1).

✓ + 2 pts **Correct order of correct lines**

✓ + 1 pts **The lines are in between the before and after lines.**

+ 0 pts Incorrect/No Answer

- 1 pts Did not print on separate lines

QUESTION 2

2 Queue Diagram 15 / 15

✓ + 2 pts **A - removes the b at index 2 and no other changes**

✓ + 1 pts **A - moves the front to index 3**

✓ + 2 pts **B - q is added correctly to the queue from part A and no other changes**

✓ + 2 pts **C - n is added correctly to the queue from B and no other changes**

✓ + 2 pts **D - the item where the front pointer is in part C is removed and no other changes**

✓ + 1 pts **D - the front pointer is moved forward by one**

✓ + 2 pts **E - x is added correctly to the queue from D and no other changes**

✓ + 3 pts **F - z is added correctly to the queue from E and no other changes**

- 5 pts Did operations in wrong order

+ 0 pts Incorrect/No Answer

QUESTION 3

BST Diagram 20 pts

3.1 Height and Depth 7.5 / 9

✓ + 1.5 pts **X Height: 3**

✓ + 1.5 pts **X Depth: 0**

+ 1.5 pts Y Height: 0

✓ + 1.5 pts **Y Depth: 2**

✓ + 1.5 pts **Z Height: 0**

✓ + 1.5 pts **Z Depth: 3**

+ 0 pts Incorrect/No Answer

3.2 BST Tree Classification 2 / 2

✓ + 2 pts **Tree S**

+ 0 pts Tree T

+ 0 pts Neither Tree

+ 0 pts Both Trees

3.3 Complete Tree Classification 0 / 3

+ 3 pts Neither Tree

+ 0 pts Tree S

+ 0 pts Tree T

✓ + 0 pts **Both Trees**

3.4 Internal or External 6 / 6

✓ + 2 pts **X: Internal**

✓ + 2 pts **Y: External**

✓ + 2 pts **Z: External**

+ 0 pts Incorrect/No Answer

QUESTION 4

4 Circular Linked List 11 / 15

✓ + 4 pts **Step 1: C**

✓ + 4 pts **D occurs before E**

✓ + 3 pts **A occurs after C and before B**

+ 4 pts Step 5: B

+ 0 pts Incorrect/No Answer

QUESTION 5

Efficiency Matching 10 pts

5.1 pop 0 / 2

✓ + 0 pts **O(1)**

+ 0 pts O(log n)

+ 2 pts O(n)

5.2 remove 2 / 2

+ 0 pts O(1)

✓ + 2 pts **O(n)**

5.3 add 0 / 2

✓ + 0 pts **O(1)**

+ 2 pts O(n)

5.4 search 2 / 2

+ 0 pts O(1)

+ 0 pts O(log n)

✓ + 2 pts **O(n)**

+ 0 pts O(n log n)

5.5 access 0 / 2

+ 2 pts O(1)

+ 0 pts O(log n)

✓ + 0 pts **O(n)**

QUESTION 6

Big O Analysis 5 pts

6.1 Part A 2 / 2

✓ + 1 pts **Overall: O(np)**

✓ + 1 pts **Inner Loop: O(p)**

+ 0 pts Incorrect/No Answer

6.2 Part B 3 / 3

✓ + 1 pts **Overall: O(n + p)**

✓ + 1 pts **First Inner Loop: O(n)**

✓ + 1 pts **Second Inner Loop: O(p)**

+ 0 pts Incorrect/No Answer

QUESTION 7

7 Array List Coding 14 / 20

✓ + 2 pts **Iterates all the way through the array.**

✓ + 2 pts **Checks to see if data is equal while iterating.**

✓ + 3 pts **Correctly finds the first occurrence and attempts to remove it.**

✓ + 2 pts **Attempts to shift the data over to make the data contiguous.**

✓ + 2 pts **Other data is not modified in the structure (no unnecessary duplication or unnecessary removals).**

+ 2 pts Data is still contiguous and begins at index 0 after the method finishes in all cases.

+ 2 pts The first occurrence is no longer in the array after the method finishes.

✓ + 1 pts **Decrements size only when there is a first occurrence.**

+ 2 pts Returns true when there is an occurrence.

✓ + 2 pts **Returns false when there is no occurrence.**

+ 0 pts Incorrect/No Answer

- 1 pts Syntax error (not for missing brackets, semicolons, etc, but for Java issues like primitive .equals())

- 3 pts Inefficient

- 2 pts Infinite Loop

- 2 pts Throws IndexOutOfBoundsException in some cases

- 1 pts Throws inappropriate exception

💬 return statement inside the for loop will exit the method itself, so the second for loop will never run

CS 1332 Exam 1A

Spring Semester 2018 - February 7, 2018

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

Signature: 

GT account username (msmith3, etc): Jshartar6

GT account number (903000000, etc): 903131050

- 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) Recursion - Tracing [15 points]

Given the following code snippet of a recursive method, trace through the method and write the output as it would appear in the console in the area designated below. The beginning and end are provided for you.

```
public static void main(String[] args) {
     ArrayList<String> tanames = new ArrayList<String>();
     tanames.add("Raymond");
     tanames.add("Chad");
     tanames.add("Andrew");
     tanames.add("Tim");
     System.out.println("Before Recursion:");
     wreck(tanames.size()-1, tanames);
     System.out.println("After Recursion.");

}

public static void wreck(int x, ArrayList<String> ta) {
    if (x == 0) {
        System.out.println("Go Jackets");
        return;
    }
    if (ta[x].length()%2 == 0) {
        System.out.println("THWg");
        wreck(x - 1, ta);
        return;
    }else{
        System.out.println(ta[x-1]);
    }
    wreck(x - 1, ta);
    System.out.println("GT");
}
```

{Raymond, Chad, Andrew, Tim}
0 1 2 3

3
Andrew → 2
THWg → 1
Raymond → 0
Go Jackets
GT
GT
GT
GT

Console :

```
Before Recursion:
Andrew
THWg
Raymond
Go Jackets
GT
GT

After Recursion.
```

2) Queue Diagram [15 points]

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 pointer is 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
Front			F		
Initial Queue			b	b	

A.) *dequeue()*

Index	0	1	2	3	4
Front				F	
Queue				b	

B.) *enqueue('q')*

Index	0	1	2	3	4
Front				F	
Queue				b	q

C.) *enqueue('n')*

Index	0	1	2	3	4
Front				F	
Queue	n			b	q

D.) *dequeue()*

Index	0	1	2	3	4
Front					F
Queue	n				q

E.) *enqueue('x')*

Index	0	1	2	3	4
Front					F
Queue	n	x			q

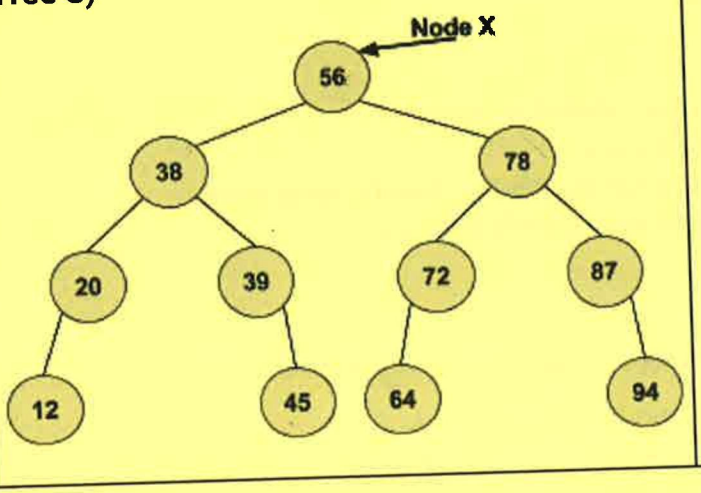
F.) *enqueue('z')*

Index	0	1	2	3	4
Front					F
Queue	n	x	z		q

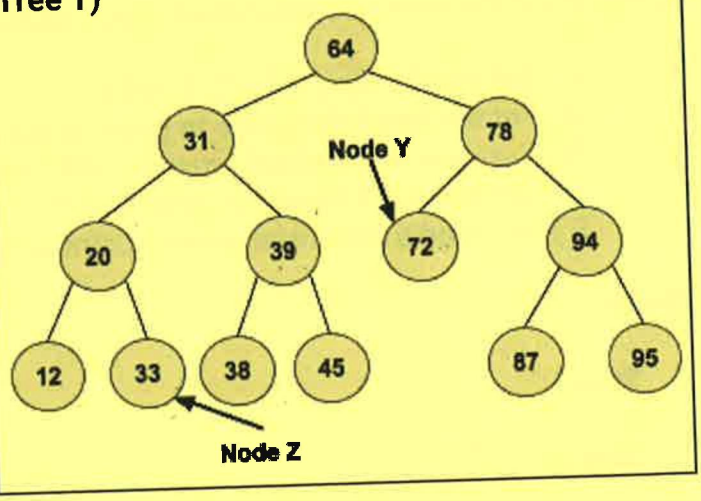
3) BST - Diagram [20 points]

Given the following two trees, answer the questions below.

Tree S)



Tree T)



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

i.) **Node X** - Height: 3, Depth: 0

ii.) **Node Y** - Height: 1, 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 internal or external? **[2 points each]**

i.) **Node X**: internal

ii.) **Node Y**: external

iii.) **Node Z**: external

4) Circular Linked List [15 points]

Given the following unordered five steps for a Circular Linked List. Arrange the steps in a correct order that would correctly add *"CS 1332"* to the **BACK** of a Circular Linked List. 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.) Set head's data to be "CS 1332".
B.) Set the **head reference** to point at the new LinkedListNode.
C.) Create a new LinkedListNode with head's data.
D.) Set the new LinkedListNode's next reference to point at **head's next node**.
E.) Set **head's next reference** to point at the new LinkedListNode.

Order:

Step 1: C

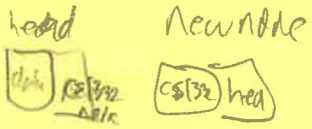
Step 2: A

Step 3: D

Step 4: E

Step 5: _____

set head in temp node
Seach until end
set to new node
set new nodes next to head



5) Efficiency - Matching [10 points]

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.) Popping from an array-backed stack **without** a size variable.

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

B.) Removing from the back of a Circular-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 **with** a size variable.

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

D.) Searching for an arbitrary element in an array-backed queue.

○ 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) ○ $O(n^2)$

6) Big-O Analysis [5 points]

For each of the code snippets below, fill in the blanks with the required Big-O. Remember, **Big-O is an upper bound on the performance of code based on the number of operations and inputs.** That being said, give the **tightest possible upper bound** for each code snippet. Assume that all basic operations (arithmetic, memory access, memory assignments, condition checking) are all O(1). You must use proper Big-O notation (for example, if it's $O(n + p^2)$, write $O(n + p^2)$, not just $n + p^2$), **discarding constant factors and lower order terms**. Failure to do so will **result in loss of points** for that part of the question.

Inputs: Non-negative integers **n** and **p**, with no particular relationship between the two inputs.

A.)
i.) Overall Big-O: O(n·p)
ii.) Big-O of Inner Loop: O(p)

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = p; j > 0; j--) {
        count++;
    }
}
```

B.)
i.) Overall Big-O: O(n+p)
ii.) Big-O of First Inner Loop: O(n)
iii.) Big-O of Second Inner Loop: O(p)

```
int count = 0;
for (int i = 0; i < 100000; i++) {
    for (int j = 0; j < 5 * n; j++) {
        count++;
    }
    for (int j = 0; j < 11 * p; j++) {
        count++;
    }
}
```

7) ArrayList - Coding [20 points]

Given the following starter code, you are responsible for implementing the *removeFirstOccurrence()* method for the *ArrayList* class. You must not violate any of the rules for *ArrayList* that we've outlined in class; the first item should always be at index 0, and there should be no data in between elements that isn't a part of the list. Since ints can't be null, if you were to null out an index, just leave the data as it is. You may **NOT** import any classes or create any helper methods or instance variables (local variables are fine). You must be as efficient as possible.

```
public class ArrayList {
    private int[] arr;
    private int size;

    /* implementation omitted */

    /**
     * This method should remove the first occurrence of the
     * given data. Remember to maintain all List properties and
     * be as efficient as possible. Ints can never be null.
     *
     * @param data The target data to be removed
     * @return true if something was removed, false otherwise
     */
    public boolean removeFirstOccurrence(int data) {
        // YOUR CODE HERE
        boolean did = false;
        for (int i=0; i < size; i++) {
            if (data == arr[i]) {
                int index = i;
                did = true;
                size--;
                return;
            }
        }
        for (int j=index; j < size; j++) {
            arr[j] = arr[j+1];
        }

        return did;

    }
}
```

r(3)

1 2 3 4 5

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.