MSOE Fall 2023-24
CSC-1120
Practice Final Exam

Name:_____

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
TF	Sort	Func	FXML	Recursion	BST	Traversal	HashTable	Implement	Usage
10	15	10	10	5	15	5	5	5	20

1. (10 points) Please answer T/F to the following questions as appropriate:
   a) ____ If you keep a size attribute for your Linked List structure, then your `public E get(int index)` method can be implemented with a runtime of O(1)

   b) ____ For JavaFX, both Button and TextField are the source of action events

   c) ____ If your code includes Path.of("data.txt"), it will crash unless you have a file named "data.txt" in the project folder that contains the "src" and "out" folders.

   d) ____ For most reads at random locations, an ArrayList is typically more efficient than a LinkedList.

   e) ____ An $O(n^2)$ algorithm always takes more time to complete than an O(n) algorithm when run on the same hardware

   f) ____ An $O(n^2)$ algorithm is also O(n), once n gets meaningfully above n0

   g) ____ It's best practice to make some variables public, even when they aren't part of the public contract for a class, to make testing and maintenance easier.

   h) ____ the add(int index, E elt) method for java.util.LinkedList is O(n)

   i) ____ You need a different underlying data structure to hold the data for Stack than you do for a Queue.

   j) ____ The two elements you need to implement a recursive algorithm are a base case and a recursive step.

   k) ____ Binary Search is a fast search that is a feature specific to Binary Search Trees. Other data structures cannot be used to implement this algorithm.

   l) ____ The primary difference between a regular Queue and a Circular Queue is that you never resize a Circular Queue.

   m) ____ When designing a recursive algorithm, the recursive step works on the same data set as the original problem but uses a simpler approach than the original.

n) ____ The contains() method on an unbalanced binary search tree is faster than the contains() method on a LinkedList.

o) ____ The @Ignore annotation can be used with unit tests to indicate that the test should run but not produce an error if the test fails.

2. (15 points) Please implement the following method that sorts a list using InsertionSort
   a. (5 points)
```
public static void sort ( List<Integer> nums) {

}
```
   b. (5 points) What is the worst-case performance, as O(), for your method

   c. (5 points) Given an input list of 7,6,3,8,22,9, what is the order of the list after the second pass through your outer loop

3. (10 points) Without using any loops, implement the following method that returns the squared values of only the numbers divisible by 7

   public static List<Integer> squaresOfMultiplesOfSeven(Stream<Integer> nums) {

   }

4. (10 points) Given the following FXML file: ** swap - use this problem on final **

```
<VBox fx:controller="final.Controller" prefHeight="50"
prefWidth="200"
          xmlns="http://javafx.com/..."
xmlns:fx="http://javafx.com/fxml/1">
  <children>
    <HBox>
      <children>
        <TextField fx:id="input" />
        <Button onAction="#handleButton" text="Capitalize" />
      <children>
    </HBox>
    <Label fx:id="result" />
  </children>
</VBox>
```

   a) 5 points - please sketch what the UI specified would look like

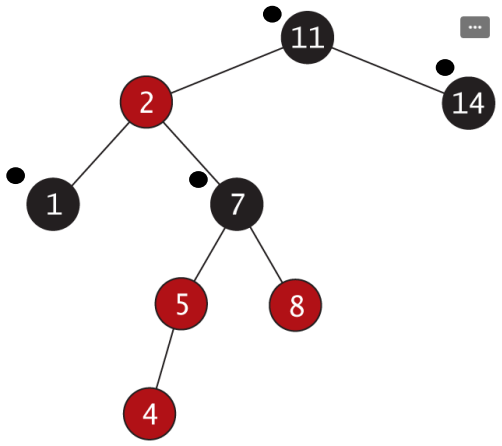
b) 5 points – please Implement the controller class corresponding to the FXML file. Include everything necessary for the Java source file except \keyc{import} statements. When the button is activated, the program should convert what the user entered into all uppercase and display the label in the label.

5. (5 points) Implement the following recursive method without loops, multiplication, or functional programming techniques:

```
/**
 * The Fibonacci sequence is 0 1 1 2 3 …
 * The first element, Fibonacci(0) = 0
 * the second, Fibonacci(1) = 1
 * and all others are the sum of the prior two
 * so Fibonacci(2) = Fibonacci(0) + , Fibonacci(1) = 0 + 1 = 1
 * and so on
 * @return the number in the sequence for the given 0-based index
 **/
public static int Fibonacci(int index) (









}
```

Name:_____

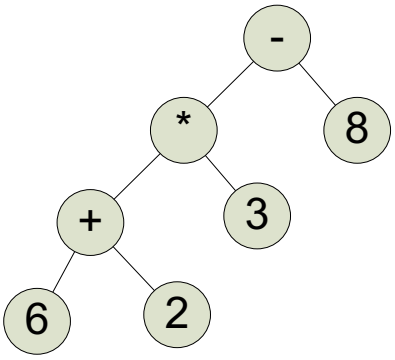
6. (15 points) Given the following Red-Black tree – (black has dot next it)



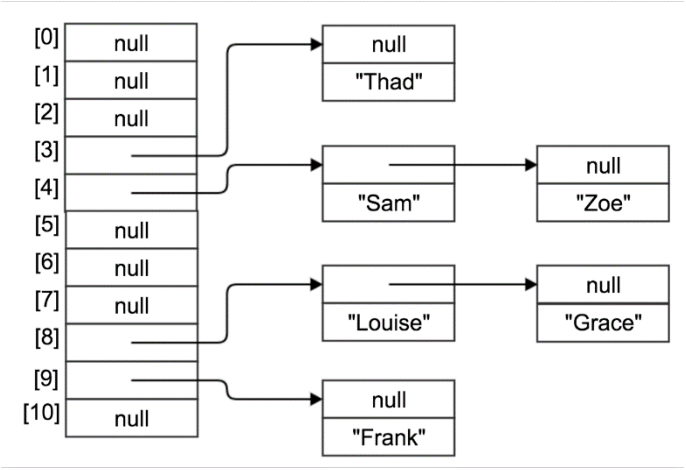
a. (10 points) Please describe how to rebalance the tree after adding the node with value ‘4’ as shown:

b. (5 points) For the AVL algorithm – what information is captured, how is that calculated, and how can you tell whether a node needs to be rotated to balance the tree

7. (5 points) *What* is the result of printing each node value during a post-order traversal of the following tree?



8. (5 points) For the following chained HashTable, `names`:



"Sam"	4
"Louise"	8
"Thad"	3
"Frank"	9
"Grace"	8
"Steve"	8
"Zoe"	4

a. *illustrate* the result of: `names.put("Steve");` Refer to the table to the right to determine the hash code value for each name. *What* is the load factor, `L`, for the hashtable in the figure above after calling `put()` in the previous part. *Show* how you determined your answer.

b. Assuming evenly distributed elements within a hashtable and a reasonably low load factor, what is the *Big-O* for insertion using chaining? *Explain* why.

9. (5 points) *Implement* the incomplete method below. Be sure to throw an `IndexOutOfBoundsException` when appropriate. Do not reduce the capacity of `array`.

```
public class ArrayList<E> implements List<E> {
   private E[] array = (E[])new Object[10]; // set initial capacity
   private int size = 0;            // size may be less than capacity
         // ...
    @Override
    public E remove(int index) {









    }
}
```

10. (20 points) Consider the following application descriptions and *select* the most appropriate 1) ***interface*** and 2) ***data structure*** implementation to use. Unless otherwise noted, assume that the amount of data being processed is large. ***Justify your answers***.

    a. A factory needs to keep track of information about the various machines they build. Each machine, contained in a `Machine` object, has a known ID associated with it. The IDs are assigned sequentially (starting at zero). Information about each is regularly searched and updated, but new machines are rarely added. Circle your choice and justify your selection.

Interface: **`List | Collection | Map | PureStack | PureQueue | Set | Heap`**
Justification:

Data structure: **`LinkedList | ArrayList | Tree | Hashtable | CircularQueue`**
Justification:

    b. A graphical editor needs to save a potentially long sequence of commands/operations a user executes to enable the undo button. Circle your choice and justify your selection.

Interface: **`List | Collection | Map | PureStack | PureQueue | Set | Heap`**
Justification:

Data structure: **`LinkedList | ArrayList | Tree | Hashtable | CircularQueue`**
Justification: