CSIS-10B PRACTICE FINAL EXAM **Open Notes & Book**
Part I: Written Portion (70 Points) **Closed Computer**
All questions, unless noted, are worth 10 points each.

1) Suppose that nameList is a list that contains the following strings: *Kyle, Cathy, Sam, Austin, Sara.* What output is produced by the following sequence of statements?

```
Iterator<String> nameIterator = nameList.getIterator();
System.out.println(nameIterator.next());
nameIterator.remove();
nameIterator.next();
System.out.println(nameIterator.next());
nameIterator.remove();
nameIterator.next();
System.out.println(nameIterator.next());
System.out.println("the modified list: ");
displayList(nameList);
```

Kyle Sam Sara, the modified list: Cathy Austin Sara

2) Suppose we have an unsorted array of Student objects:

```
Student[]  unsorted = new Student[1000];
```

a) What would be the advantages of putting them into a `SortedArrayDictionary`, using the `Student` ID field as the Key?

In an unsorted array, searching for a student by ID would be O(n), but in a sorted array dictionary, searching for student would use the binary search algorithm which is O(logn)

A sorted array dictionary would allow us to easily modify the collection of student objects, adding or deleting entries.

We would automatically know how many entries we have in the sorted array dictionary, whreas the unsorted array could have null values in it.

b) If the Student class consists of

```
public class Student
{
   private String firstName, lastName;
   private double GPA;
   private int ID;

… and appropriately named set and get methods…
}
```

Write a statement that creates a SortedArrayDictionary of 1000 ID/Student entries

```
SortedArrayDictionary<Student> database = new SortedArrayDictionary<Student>(1000);
```

c) Write statements that add the contents of array `unsorted` to the Dictionary you created in b)

```
for (int k=0; k<1000; k++)
        database.add(unsorted[k].getID(), unsorted[k])
```

3) Suppose an object hashes to cell index 7 in an 11 cell table. List the next 4 cell indices that will be examined (if there are collisions), using

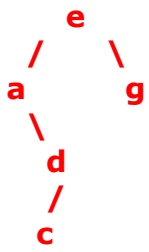
With 11 cells, the available index locations are 0 through 10

a) Linear Probing: **8,9,10,0 (skip to the next one in sequence, then rollover after 10)**

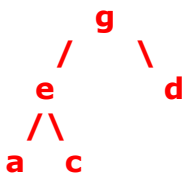
b) Quadratic Probing: **8,0,5,1 (skip by 1,3,5,7 and rollover after 10)**

4) draw the data structure you get when inserting the values "e" "a" "d" "g" "c" into:

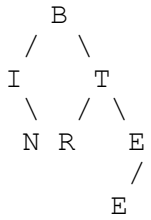
a) a binary search tree



b) a max heap

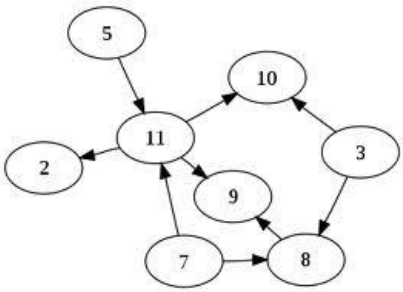


5) Here is a small binary tree:



Write the order of the nodes visited in:
A. An in-order traversal: **I,N,B,R,T,E,E**
B. A pre-order traversal: **B,I,N,T,R,E,E**
C. A post-order traversal: **N,I,R,E,E,T,B**

6) In the graph below, show the Vertices visited while performing a **SKIP THIS**



a. Breadth-first traversal starting at Vertex 7

b. Depth-first traversal starting at Vertex 7

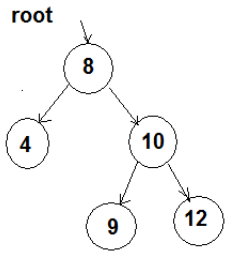
c. Represent the graph with an adjacency matrix

7) Suppose we discover a mysterious pair of methods in the simple BST class from Lab 12:

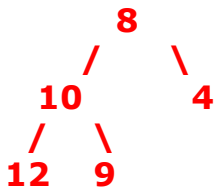
```
public void mystery()
{
    mysteryAux(root);
}

private void mysteryAux(TreeNode subNode)
{
    if (subNode != null)
    {
            TreeNode temp = subNode.left;
            subNode.left = subNode.right;
            subNode.right = temp;
            mysteryAux( subNode.left);
            mysteryAux( subNode.right);
    }
}
```

a) Show what happens to the following tree AFTER we invoke the mystery() method on it:

BEFORE



AFTER



b) Does it still obey the BST property?

No. For a Binary Search Tree (BST), all left children of a node must have smaller data values than the node's data, and all right children must have data values that are larger than the node's data. The modified tree is kind of a "reverse" BST—it obeys the opposite rule.

When you have completed the written portion of the exam turn in your work and pick up Part 2, the Open Computer Problem.

CSIS-10B FINAL EXAM PART 2 (30 Points) Open Computer Book, Notes

When you have completed the written portion of the exam, download and extract the folder www.tomrebold.com/csis10b/SimpleBST.zip to your desktop.

8) (30 points) Open the project in BlueJ or Eclipse and run the code in the BSTmainApp class. You'll notice it generates the familiar tree from lab 12. Now, make the following changes to the project:

A) In class Student define a compareTo method that compares Students by ID

B) create new classes StuBST, and StuTreeNode. Then copy the code from BST and TreeNode into these classes and modify them so that they store Student objects instead of Strings. Then activate the code and add more code to create a tree of Student data.

C) Create an iterator for the `student` tree and use it to print all the Student last names in the tree (print only the last names)

D) Extra Credit (5 points): Convert the BST and TreeNode classes to be competely generic and modify the code for B to use these generic tree classes.

E) Extra Credit (5 points): Read the entire file StudentA.txt into the BST you created above

Add your name to the top of each file in the project.

When you are finished, make a jar of the project folder and upload it to the server. Print out the uploader response page with your code on it, staple and turn in.

```
import java.util.Scanner;
/**
 * Write a description of class Student here.
 * Class student represents a student record in a database.
 * @author (Villarreal)
 * @version 2/9/2011
 */
public class Student implements Comparable<Student>
{
   private String firstName, lastName;
   private double GPA;
   private int ID;

   public int compareTo(Student other)
   {
       return this.ID - other.ID;
   }

   // rest of Student class omitted for clarity
```

```
/**
 * Write a description of class TreeNode here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class StuTreeNode
{
   Student data;              // change String to Student
   StuTreeNode left, right;   // change TreeNode to StuTreeNode

   StuTreeNode( Student data){
      this.data=data;
      left=null;
      right=null;
   }

   public String toString(){  // inorder recursive traversal
      String result="";
      if (left!=null)
         result+= left.toString();
      result+=" "+data+ " ";
      if (right!=null)
         result+= right.toString();
      return result;
   }
}


import java.util.Iterator;
import java.util.Vector;
/**
 * Write a description of class BST here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class StuBST
{
   StuTreeNode root;  // same idea, all the way down

   StuBST(){
      root = null;
   }

   public String toString(){
      if (root!=null)
         return root.toString();
```

```
    else
      return "";
}

public void add(Student element){
   StuTreeNode current=root, parent=root;
   if (root == null)
      root = new StuTreeNode(element);
   else {
     // Locate the parent node
     current = root;
     while (current != null)
       if (element.compareTo(current.data)<0) {
         parent = current;
         current = current.left;
       }
       else if (element.compareTo(current.data)>0) {
         parent = current;
         current = current.right;
       }
       else {
         System.out.println("item already in tree");
         return; // Duplicate node--nothing inserted
       }
     // Create the new node and attach it to the parent node
     if (element.compareTo(parent.data)<0)
       parent.left = new StuTreeNode(element);
     else
       parent.right = new StuTreeNode(element);
   }
}

public boolean contains(Student element){
   if (root == null)
      return false;
   else {
     // traverse through tree looking for element
     StuTreeNode current = root;
     while (current != null)
       if (element.compareTo(current.data)<0) {
         current = current.left;
       }
       else if (element.compareTo(current.data)>0) {
         current = current.right;
       }
       else {
         return true;
       }
```

```
      }
      return false;
}


public Student get(Student element){
    if (root == null)
        return null;   // tree empty
    else {
      // traverse through tree looking for element
      StuTreeNode current = root;
      while (current != null)
        if (element.compareTo(current.data)<0) {
          current = current.left;
        }
        else if (element.compareTo(current.data)>0) {
          current = current.right;
        }
        else {
          return current.data;
        }
      }
      return null;  // not in tree
}

public Iterator<Student> iterator(){
    Vector<Student> vs = new Vector<Student>();
    iterAux(vs,root);
    return vs.iterator();
}

public void iterAux(Vector<Student> v, StuTreeNode subTree){
    if (subTree!=null){
        iterAux(v, subTree.left);
        v.add(subTree.data);
        iterAux(v, subTree.right);
    }
}


public void graph(){
    this.graphAux(0, root);
}

public void graphAux(int indent, StuTreeNode subTree){
    if (subTree != null)
    {
      graphAux(indent + 8, subTree.right);
```

```
        for (int k=0; k<indent; k++)
            System.out.print(" ");
        System.out.println(subTree.data);
        graphAux(indent + 8, subTree.left);
      }
   }

}


import java.util.Iterator;
/**
 * Write a description of class BSTmainApp here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BSTmainApp
{
   public static void main( String [] args){
//        // BST demo, linking nodes directly
//        BST names = new BST();
//
//        // Iterative methods: add, contains, get
//        names = new BST();
//        names.add("George");
//        names.add("Barbara");
//        names.add("Jack");
//        names.add("Harry");
//        names.add("Josephine");
//        names.add("Robert");
//        names.add("Sarah");
//
//        names.graph();

       // A)  define compareTo for the Student class so it compares
       //       Students by ID
       // B)  make new classes StuBST, StuTreeNode, and activate this program to create
       //       a tree of Student data
       //
        StuBST students = new StuBST();
        students.add(new Student("Lilianna", "Jadyn",152275, 2.04));

        //also add these students
       students.add(new Student("Reid", "Madilyn", 167160, 3.5));
       students.add(new Student("Charlie", "Laylah",140164, 3.67));
       students.add(new Student("Arianna", "Kendall",109127, 2.84));
       students.add(new Student("Sydney", "Junior",187831, 3.07));
```

```
        students.graph();
        //
        // C) Create an iterator for the students tree and use it to print
        //       all the Student last names in the tree (only the last names)

        Iterator<Student> sit = students.iterator();

        while(sit.hasNext())
           System.out.println( sit.next());


        // D) Extra Credit: Make the BST and TreeNode classes to be competely generic
        //       and modify the code for B to use these generic tree classes
        //
        // E) Extra Credit: Read the entire file StudentA.txt into the BST you created above

    }
}
```