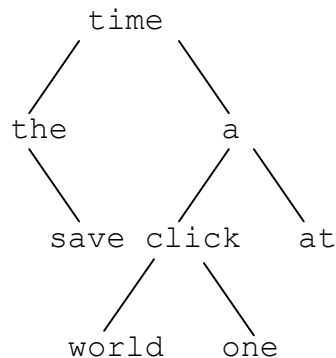


CSE 143 SAMPLE FINAL EXAM SOLUTION

1. (10 points) Draw the tree of strings that would give the following traversals:

pre-order: time the save a click world one at
in-order: the save time world click one a at
post-order: save the world one click at a time



2. (40 points) Recall the definitions of the `ListNode` and `LinkedList` class:

```
public class ListNode {
    ...
}

public class LinkedList {
    ...
}
```

Write a method `removeEveryOther(void)` for the `LinkedList` class that removes every other node in the list. If the list contained `[11,4,34,67,5]`, then a call to `removeEveryOther` would result in `[11,34,5]`. If the list is empty or there is only a single node, the method does nothing and simply returns. You may NOT create any new nodes or overwrite the data variable of any node. You must solve this problem by manipulating the `next` variable in the `ListNode` class. Do NOT use recursion.

```
public void removeEveryOther() {
    ListNode current = front;
    while (current != null && current.next != null) {
        current.next = current.next.next;
        current = current.next;
    }
}
```

3. (40 points) Assume the following definition of binary trees of integers:

```
public class TreeNode {
    public int data;          // data stored at this node
    public TreeNode left;    // reference to left subtree
    public TreeNode right;   // reference to right subtree

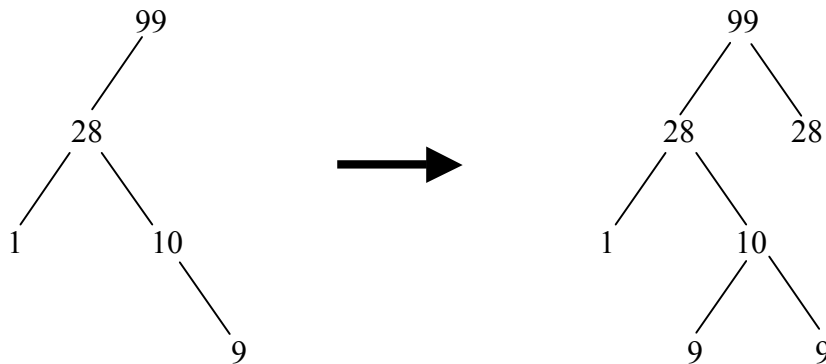
    // post: constructs a leaf node with given data
    public TreeNode(int data) {
        this(data, null, null);
    }

    // post: constructs a node with the given data and links
    public TreeNode(int data, TreeNode left, TreeNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}

public class Tree {
    private TreeNode root;

    <methods>
}
```

Write a method `cloneLonelyChildren(void)` in the `Tree` class that takes every node that has *exactly* one child and attaches a new leaf node to the childless branch with the same data as the existing child. For example, a call to `cloneLonelyChildren` would result in the following mutation:



Notice that every node is now a leaf node or has exactly 2 children.

[The problem statement is on the back of the previous page]

```
public void cloneLonelyChildren() {
    cloneLonelyChildren(root);
}

private void cloneLonelyChildren(TreeNode node) {
    if (node == null) {
        return;
    }

    if ((node.left == null) && (node.right != null)) {
        node.left = new TreeNode(node.right.data);
    }

    if ((node.right == null) && (node.left != null)) {
        node.right = new TreeNode(node.left.data);
    }

    cloneLonelyChildren(node.left);
    cloneLonelyChildren(node.right);
}
```

```
public void cloneLonelyChildren() {
    root = cloneLonelyChildren(root);
}

private TreeNode cloneLonelyChildren(TreeNode node) {
    if (node == null) {
        return null;
    } else if ((node.left == null) &&
               (node.right != null)) {
        node.left = new TreeNode(node.right.data);
        node.right = cloneLonelyChildren(node.right);
    } else if ((node.right == null) &&
               (node.left != null)) {
        node.left = cloneLonelyChildren(node.left);
        node.right = new TreeNode(node.left.data);
    } else { // has two children
        node.left = cloneLonelyChildren(node.left);
        node.right = cloneLonelyChildren(node.right);
    }
    return node;
}
```

4. (12 points) For the following, either answer True/False or fill in the blank and explain your answer in a short sentence or two.

(a) [DELETED]

(b) Search on a binary search tree takes $O(\log n) / O(n)$ time.

Explanation:

We accepted either answer if your explanation described why it should be one or the other. Search could take up to $O(n)$ time if the tree is unbalanced. However, on a balanced tree, it is $O(\log n)$.

(c) `add(int index, int value)` in the vanilla `LinkedList` (shown below) takes $O(n)$ time.

```
public class ListNode {
    int data;
    ListNode next;
}

public class LinkedList {
    private ListNode front;

    <methods>
}
```

Explanation:

Finding the index is linear, since you have to start from the beginning each time `add` is called. The index could potentially be the last spot in the list.

(d) An $O(n^3+5)$ algorithm is also $O(n^3)$. **True**

Explanation:

Constants don't matter.

5. (12 points) List 4 critical (but simple) errors with the following code and/or declarations. Circle the error and write a brief explanation. Code in bold is correct.

```
public interface Person
{
    public void greet() {
        System.out.println("Hello!");
    }
}

public abstract class Student implements Person
{
    public abstract void doHomework();

    public void greet() {
        System.out.println("sup?");
    }
}

public class HumanitiesStudent extends Student {}

public class EngineeringStudent extends Student
{
    public void doHomework() {
        solveProblems();
    }

    public void solveProblems() { }
}

public class CseStudent extends EngineeringStudent {}

Student alice = new Student();
Student bob = new CseStudent();
CseStudent cynthia = new EngineeringStudent();
CseStudent doug = new CseStudent();
Object emily = new EngineeringStudent();
```

Descriptions of errors from top to bottom:

1. Interfaces can't have methods with implementations.
2. HumanitiesStudent should either be abstract or implement doHomework.
3. Student is an abstract class and thus cannot be instantiated.
4. EngineeringStudent is superclass and does not fit in CseStudent role.

6. (40 points) The following class represents a simple bank account:

```
public class Account {
    private int amount;

    public Account(int initialAmount) {
        if (initialAmount < 0) {
            throw new IllegalArgumentException();
        }
        amount = initialAmount;
    }

    // post: deposits money into this account
    public void deposit(int depositAmount) {
        if (depositAmount < 0) {
            throw new IllegalArgumentException();
        }
        amount += depositAmount;
    }

    // post: withdraws money from this account
    public void withdraw(int withdrawAmount) {
        if (withdrawAmount < 0) {
            throw new IllegalArgumentException();
        }
        int total = amount - withdrawAmount;
        if (total < 0) {
            System.out.println("Not enough money!");
        } else {
            amount = total;
        }
    }
}
```

Design a new class called `SecureAccount` that is a secure version of the `Account` class. The constructor takes as parameters a password and an initial amount. Depositing into a secure account is the same as a regular account (getting more money into your account is always ok!). However, to withdraw money, a password must also be given in addition to the amount to be withdrawn via:

```
public void withdraw(int withdrawAmount, String password)
```

If no password is given (*i.e.*, the `withdraw` method with one parameter is called), then the message "Please enter your password." should be printed. If a password is entered and the password is wrong, the message "Sorry, wrong password." is printed. After 3 failed calls to the `withdraw` method with 2 parameters, the account locks down. Future attempts (including those with the correct password) will print the

message “Your account has been locked.” If before 3 failed tries, the user gets the right password, the counter resets back to 0.

Use inheritance to solve this problem. Redundant code will result in lost points.

```
public class SecureAccount extends Account {
    private String password;
    private int attempts = 0;

    public SecureAccount(String password, int amount) {
        super(amount);
        this.password = password;
    }

    public void withdraw(int amount) {
        System.out.println("Please enter your password.");
    }

    public void withdraw(int amount, String password) {
        if (attempts >= 3) {
            System.out.println("Your account has been locked.");
            return;
        }

        if (!password.equals(this.password)) {
            System.out.println("Sorry, wrong password.");
            attempts++;
        } else {
            super.withdraw(amount);
            attempts = 0;
        }
    }
}
```