Problem 1:

(a) `obj2 = obj1;`

   ILLEGAL because type of reference must always be a supertype of type of object

(b) `obj3 = obj1;`

   ILLEGAL because type of reference must always be a supertype of type of object

(c) `obj3 = obj2;`

   ILLEGAL because type of reference must always be a supertype of type of object

(d) `I1 b = obj3;`

   LEGAL because C3 is a subclass of C1 which implements I1

(e) `I2 c = obj1;`

   ILLEGAL because type of reference must always be a supertype of type of object

Grading:

(max points off each: -2)

wrong conclusion: -2

right conclusion, wrong reason: -1
Problem 2(a):

Base case: $n = 2$

L.H.S. $1 - \frac{1}{2^2} = \frac{3}{4}$

R.H.S. $\frac{2 + 1}{2 \times 2} = \frac{3}{4}$

Therefore, base case proved.

Inductive Hypothesis: For $n = k$ let

\[
\left(1 - \frac{1}{4}\right)\left(1 - \frac{1}{9}\right)\ldots\left(1 - \frac{1}{k^2}\right) = \frac{k + 1}{2k}
\]

We now need to show that for $n = k + 1$

\[
\left(1 - \frac{1}{4}\right)\left(1 - \frac{1}{9}\right)\ldots\left(1 - \frac{1}{k^2}\right)\left(1 - \frac{1}{(k+1)^2}\right) = \frac{(k+1) + 1}{2(k+1)} = \frac{k + 2}{2k + 2}
\]

Using our inductive hypothesis, we can rewrite:

\[
\left(1 - \frac{1}{4}\right)\left(1 - \frac{1}{9}\right)\ldots\left(1 - \frac{1}{k^2}\right)\left(1 - \frac{1}{(k+1)^2}\right) = \frac{k + 1}{2k}\left(1 - \frac{1}{(k+1)^2}\right)
\]

\[
= \frac{k + 1}{2k}\left(\frac{(k+1)^2 - 1}{(k+1)^2}\right)
\]

\[
= \frac{1}{2k}\left(\frac{k^2 + 2k}{k + 1}\right)
\]

\[
= \frac{k(k + 2)}{2k(k + 1)}
\]

\[= \frac{k + 2}{2k + 2} \quad \text{proved.}\]
Grading:

(max points off: -10)
wrong base case: -1
base case stated but not proved: -2
inductive hypothesis wrong: -2
inductive step wrong: -4
conclusion wrong: -2
bad algebra: -2
Problem 2(b):

Base cases: \( n = 1, n = 2 \)

\[ a_1 = 2^1 + 1 = 3 \quad \text{and} \quad a_2 = 2^2 + 1 = 5 \]

Therefore, base case proved.

Inductive Hypothesis: For \( n = 1, 2, 3, \ldots, k \) let \( a_k = 2^k + 1 \)

We need to show that for \( n = k + 1 \), \( a_{k+1} = 2^{k+1} + 1 \)

Now it's given that

\[ a_{k+1} = 3a_k - 2a_{k-1} \]

Using our inductive hypothesis,

\[
\begin{align*}
&= 3 \cdot (2^k + 1) - 2 \cdot (2^{k-1} + 1) \\
&= 3 \cdot 2^k + 3 - 2^k - 2 \\
&= 2 \cdot 2^k + 1 \\
&= 2^{k+1} + 1
\end{align*}
\]

Grading:

(max points off: -10)

- no base cases or 2 base cases but no proof: -2
- only 1 base case with proof: -1
- inductive hypothesis wrong: -2
- inductive step wrong: -4
- conclusion wrong: -2
- bad algebra: -2
Problem 3:

(a) public static int A(int i, int j) {
    if (i==1 && j>=1)
        return (int) Math.pow(2,j);
    else if (i>=1 && j==1)
        return A(i-1,2);
    else
        return A(i-1,A(i,j-1));
}

(b) invoke A(2,2)
    invoke A(2,1)
    invoke A(1,2)
    return 4 from invocation A(1,2)
    return 4 from invocation A(2,1)
    invoke A(1,4)
    return 16 from A(1,4)
    return 16 from invocation A(2,2)

Grading:

part a. (max points off for part a: -7)

   function return type wrong: -1

   function parameter types or number wrong: -2

   no typecast for Math.pow(): -1

   for every wrong branch (3 branches total): -2

   no recursion: -5

part b. (max points off for part b: -3)

   minor error in sequence: -1

   major error: -3
Problem 4:

// iterative solution
public static ListCell reverse(ListCell f) {
    if (f==null)
        return f;
    
    ListCell curr = f;
    ListCell next = f.getNext();
    curr.setNext(null);

    while(next != null) {
        ListCell temp = next.getNext();
        next.setNext(curr);
        curr = next;
        next = temp;
    }

    return curr;
}

// recursive solution
public static ListCell reverse(ListCell f) {
    if (f==null)
        return null;
    else if (f.getNext()==null)
        return f;
    else {
        ListCell head = reverse(f.getNext());
        f.getNext().setNext(f);
        f.setNext(null);
        return head;
    }
}
Grading:

(max points off: -30)

fails if f==null: -5

fails if length of f is 1: -5

last node’s "next" in the reversed list does not point to null: -5

creates new ListCell: -15

uses List class or creates an entirely new list: -25

inefficient (e.g. iterates list more than once): -10

returned wrong node in iterative traversal: -3

loses pointer to all or part of list: -10
Problem 5:

```java
public static int sumTree(GTreeCell root) {
    if (root == null)
        return 0;
    else
        return ((Integer) root.getDatum()).intValue() +
                sumTree(root.getLeft()) +
                sumTree(root.getSibling());
}
```

Grading:

(max points off: -15)

return type not int: -3
parameter type not GTreeCell: -3
does not use recursion: -10
base case wrong: -5
Integer typecast not used: -2
intValue() not used correctly: -1
does not add sum of left tree: -2
does not add sum of siblings: -4
incorrect or no use of get functions: -2
method not named sumTree: -2
no base case if root == null: -2
Problem 6:

(a) not unique. one possibility:

(b) not unique. one possibility:

Grading:

part a. (max points for part a: -10)

preorder traversal fails: -5

postorder traversal fails: -5

part b. (max points for part b: -5)

answer is "unique": -5

answer is "not unique" but wrong tree: -3