Data Structures
CSCI H343, Fall 2019 [A]

Midterm

Name: ____________________

This exam has 10 questions, for a total of 100 points.

1. $\boxed{\text{6 points}}$ What is the output of the following Java program?

```
class Node {
    Node next; int data;
    Node(int d) { data = d; }
}
public class Main {
    static void f(Node n, int k) {
        if (k != 0) {
            n.next = new Node(0);
            f(n.next, k - 1);
            n.data = k;
        }
    }
    public static void main(String[] args) {
        Node x = new Node(4);
        x.next = new Node(3);
        f(x, 2);
        System.out.println(x.data);
        System.out.println(x.next.data);
        System.out.println(x.next.next.data);
    }
}
```

Solution: Rubric: 2 points per correct output

```
2
1
0
```

2. $\boxed{\text{12 points}}$ What is the big-O time complexity of the following `sort` method in terms of the length of the array `A`, represented by n? Provide an argument for your answer that analyzes every statement in the method and how their individual time complexities combine into the total time complexity.

```
static public <K> void sort(ArrayList<K> A, BiPredicate<K, K> lessThan) {
    AVLTree<K> T = new AVLTree<K>(lessThan);
    for (int i = 0; i != A.size(); ++i) {
        T.insert(A.get(i));
    }
    A.clear();
    List<K> L = T.keys();
    for (K k : L) {
        A.add(k);
    }
}
```

Name: ____________________

Solution:

1. The constructor for `AVLTree` to create `T` is $O(1)$. **(1 point)**
2. The call `T.insert(...)` is $O(\log(n))$ **(2 points)** and it is inside a `for` loop with n iterations, so the `for` loop is $O(n \log(n))$ **(2 points)**.
3. The call `A.clear()` is $O(1)$ **(1 point)**.
4. The call `T.keys()` is $O(n)$ **(1 point)**.
5. The call `A.add(k)` is amortized $O(1)$ **(1 point)** and it is inside a `for` loop with n iterations, so this `for` loop is $O(n)$ **(2 points)**.
6. The maximum of this sequence of time complexities is the one for the first `for` loop, so the overall time complexity of `sort` is $O(n \log(n))$. **(2 points)**

3. $\boxed{\text{8 points}}$ Consider the following binary tree.

$$\begin{tikzcd}[column sep=small]
 & & 8 \arrow[dl] \arrow[dr] & & \\
 & 2 \arrow[dl] \arrow[d] & & 7 \arrow[dr] & \\
5 & 3 & & & 1
\end{tikzcd}$$

The following questions are with respect to an inorder traversal.

1. Which node comes immediately after node 8?
2. Which node comes immediately after node 3?
3. Which node comes immediately before node 1?
4. Which node comes immediately before node 8?

Solution: 2 points each

1. 7
2. 8
3. 7
4. 3

4. $\boxed{\text{12 points}}$ Fill in the blanks to complete the proof of correctness for the following function named `sum_to`.

```
// precondition: $0 \le n$
// postcondition: sum_to($n$) $= \frac{n(n+1)}{2}$
static int sum_to(int n) {
    int result, s;
    assert ___(a)___;
    if (n == 0) {
        assert n == 0 && 0 <= n;
        result = 0;
        assert result == 0 && n == 0;
        // $\frac{0(0+1)}{2} = \frac{0}{2} = 0$
        assert ___(b)___;
    } else {
        assert ___(c)___ && 0 <= n;
        // From the above two facts and the definition of $\le$.
        assert 1 <= n;
        // apply rules of algebra
        assert ___(d)___;
        s = sum_to(n - 1);
        assert ___(e)___;
1         // apply rules of algebra
```

```
        assert s == ((n - 1) * n) / 2;
        result = n + s;
        assert ___(f)___ && s == ((n - 1) * n) / 2;
        // substitute s
        assert result == n + ((n - 1) * n) / 2;
        // apply rules of algebra.
        assert result == (n * (n + 1)) / 2;
    }
    assert result == (n * (n + 1)) / 2;
    return result;
}
```

Solution: Rubric: 2 points each

```
a) 0 <= n
b) result == (n * (n + 1)) / 2
c) n != 0
d) 0 <= n - 1
e) s == ((n - 1) * ((n - 1) + 1)) / 2
f) result == n + s
```

5. $\boxed{\text{9 points}}$ For the following node class in a binary tree, implement the `next` method that returns the node that comes after the `this` node with respect to an inorder traversal, if there is one, and `null` if there is none. You are not given any helper methods, but may implement any that you wish to use.

```
class Node {
    T data;
    Node left, right, parent;
    ...
}
```

Solution: Rubric:

- Code for dispatching to the two cases depending on whether there is a right child or not. (3 points)
- Code for finding the first node in a subtree. (3 points)
- Code for finding the first ancestor that comes after with respect to inorder traversal. (3 points)

```
Node next() {
    if (right == null) {
        return nextAncestor();
    } else {
        return right.first();
    }
}
Node first() {
    if (left == null) {
        return this;
    } else {
        return left.first();
    }
}
Node nextAncestor() {
    Node n = this, p = parent;
    while (p != null && p.right == n) {
        n = p;
        p = p.parent;
    }
    return p;
}
Node nextAncestor() { // alternative recursive version
    if (parent == null) {
        return null;
    } else if (this == parent.left) {
        return parent;
    } else {
        return parent.nextAncestor();
    }
}
```

6. $\boxed{\text{10 points}}$ What is the big-O time complexity of the following `flood` method in terms of the length of the total number of tiles, represented by n? Provide an argument for your answer that analyzes every statement in the method and how their individual time complexities combine into the total time complexity.

```
static void flood(WaterColor color, LinkedList<Coord> flooded_list,
                  Tile[][] tiles, Integer board_size)
{
    for (int i = 0; i != flooded_list.size(); ++i) {
        Coord c = flooded_list.get(i);
        for (Coord n : c.neighbors(board_size))
            if (!flooded_list.contains(n)
                && tiles[n.getY()][n.getX()].getColor() == color)
                flooded_list.add(n);
    }
}
```

Solution: The outer `for` loop has at most n iterations because the `flooded_list` may contain up to n coordinates. **(1 point)**

- The call `flooded_list.get(i)` is $O(n)$. **(2 points)**
- The inner `for` loop may iterate up to 4 times, so it doesn't matter.
- The call to `flooded_list.contains(n)` is $O(n)$. **(2 points)**
- The `n.getX()`, `n.getY()`, `getColor()`, and `flooded_list.add(n)` methods are all $O(1)$ and the access to the `tiles` array is $O(1)$ **(1 point)**.

The maximum time complexity of the operations inside the outer `for` is $O(n)$ **(2 points)**, so the time complexity of the `flood` method is $O(n^2)$ **(2 points)**.

7. $\boxed{\text{10 points}}$ Fill in the blanks to complete the following implementation of the `Sequence` interface using a Java array.

```
class Array<T> implements Sequence<T> {
    T[] data;
    public Array(T[] a) { data = a; }
    public class ArrayIter implements Iter<T> {
        int pos;
        ArrayIter(int p) { pos = p; }
        public T get() { return ___(a)___; }
        public void advance() { ___(b)___; }
        public Iter<T> clone() { return new ArrayIter(___(c)___); }
        public boolean equals(Iter<T> other) {
            return pos == ___(d)___.pos;
        }
    }
    public Iter<T> begin() { return new ArrayIter(0); }
```

```
        public Iter<T> end() { return new ArrayIter(___(e)___); }
    }
```

Solution: Rubric: 2 points each

```
a) data[pos]
b) ++pos
c) pos
d) (ArrayIter)other
e) data.length
```

8. $\boxed{\text{10 points}}$ Prove that $5\log_2 n \in O(\log_4 n)$. Recall that logarithm is inverse to exponentiation:

$$\log_b(x) = y \quad \text{if and only if} \quad b^y = x$$

and the rule for changing the base of a logarithm:

$$\log_b x = \frac{\log_k x}{\log_k b}$$

Solution:

We need to show that $\exists kc.\forall n \geq k. 5\log_2 n \leq c\log_4 n$. By change of base, we have $\log_4 n = \frac{\log_2 n}{\log_2 4} = \frac{1}{2}\log_2 n$.

To counter the $\frac{1}{2}$ in the bound $\frac{1}{2}\log_2 n$ and make up for the 5 in the function $5\log_2 n$, we choose $c = 10$ **(3 points)**. Here's the table for a few values of n:

n	$\log_2 n$	$5\log_2 n$	$\log_4 n$	$10\log_4 n$
1	0	0	0	0
4	2	10	1	10
16	4	20	2	20
64	6	30	3	30

It looks like we can choose $k = 0$ **(3 points)**.

Given our choice of k and c, we need to show $\forall n \geq 0.\ 5\log_2 n \leq 10\log_4 n$ **(2 points)**. Let n be an arbitrary non-negative integer. By change of base, we have

$$10\log_4 n = 10\frac{\log_2 n}{\log_2 4} = 10\frac{\log_2 n}{2} = 5\log_2 n$$

So indeed, $5\log_2 n \leq 10\log_4 n$. **(2 points)**

Name: ______________________

9. $\boxed{\text{10 points}}$ You are given a hashtable with table size 4 that uses the multiply-add-and-divide (MAD) method for the hash function: $h(k) = ((ak + b) \mod p) \mod m$ (with $p = 11$, $a = 5, b = 7$) and uses separate chaining for collisions. Write down the equation for the MAD method hash function, specialized for this hashtable. Insert the keys $1, 2$, 3, and 4 into the hashtable. Show the hash value for each key. Draw the resulting hashtable. How many collisions occurred?

Solution: The equation for the multiply-add-and-divide method for this hashtable is: **(1 point)**

$$h(k) = ((5k + 7) \mod 11) \mod 4$$

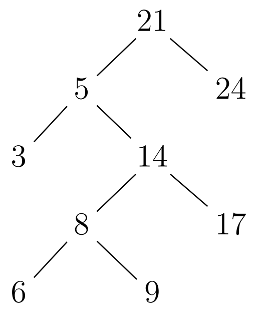
The results from hashing are:

- $h(1) = 1$ **(1 point)**
- $h(2) = 2$ **(1 point)**
- $h(3) = 0$ **(1 point)**
- $h(4) = 1$ **(1 point)**

The table is: $\begin{array}{c|ll} 0 & \rightarrow 3 & \\ 1 & \rightarrow 1 & \rightarrow 4 \\ 2 & \rightarrow 2 & \\ 3 & \rightarrow & \end{array}$ **(4 points)**

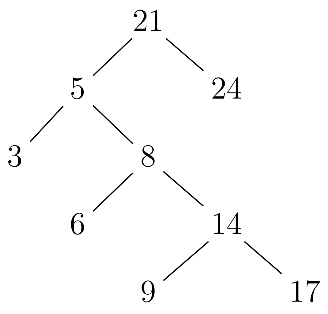
There was one collision: 4 collided with 1. **(1 point)**

10. $\boxed{\text{13 points}}$ Balance the following binary search tree so that it satisfies the AVL property, fixing nodes that are lower in the tree before fixing nodes that are closer to the root. Only use tree rotations to change the tree. Draw the tree after each rotation.

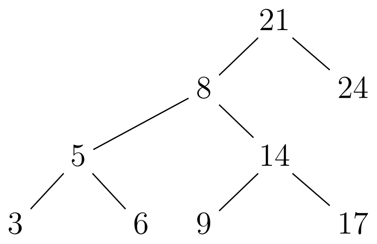


Solution:

Node 5 does not satisfy the AVL property **(2 point)**. It's right node is left heavy, so we first rotate right on 14 **(3 points)**.



Then we rotate left on 5 **(3 points)**.



Now 21 is the lowest node that does not satisfy the AVL property **(2 points)**. It's left child is balanced, so we rotate right on 21 **(3 points)**.