You are to honor the Emory Honor Code. This is a closed-book and closed-notes exam. You have 150 minutes to complete this exam. Read each problem carefully, and review your answers. Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Below are some potentially helpful formulas.

\[1 + 2 + 3 + \ldots + (n - 1) + n = \frac{n \times (n + 1)}{2}\]

\[1 + x + x^2 + \ldots + x^{n-1} + x^n = \frac{x^{n+1} - 1}{x - 1}\]
Problem 1. Give brief answers to the following questions.

1. What are the general rules for selecting which data structure to use among arrays, linked list, and binary search trees?

2. What is the best case and worst case time complexity in Big O notation for selection sort algorithm?

3. What is a binary search tree?

4. If the items are inserted into a binary search tree in random order, the worst-case height can be $O(\ldots)$ but the expected height is $O(\ldots)$.

5. Which graph representation, adjacency list or adjacency matrix, is preferred for large, sparse graphs?

6. Which data structure, queue or stack, is used for depth-first search (DFS) algorithm for a graph?

7. Write down the cost function for mergesort algorithm in recurrence relations, and solve it (4 points)
Problem 2. Below is a recursive method which uses a queue of integers with typical insert and remove operations.

```java
public static int mystery(int n) {
    if (n == 0)
        return 1;
    theQueue.insert(n);
    return mystery(n-1) * theQueue.remove();
}
```

1. Assuming the queue is initially empty, draw a snapshot of the queue after every insert and remove statement for each recursive call for `mystery(4)`. Label each queue snapshot with the recursive call, and the corresponding insert or remove statement. For example, the following shows the first snapshot of the queue after the insert statement for the call `mystery(4).

   4

   mystery(4) - insert

2. What does `mystery(4)` evaluate to? What does the method `mystery` compute?
Problem 3. A circular linked list, is a linked list whose last node links back to the first node (as opposed to null). Consider a circular linked list which maintains integers (> 0) in sorted order, and always contains exactly one node with the value of 0. The code for the Node class, a main method using it, and a picture of the list after the main is executed are given below.

```java
public class Node {
    int value;
    Node next;
    public Node(int v){
        value = v;
        next = null;
    }
}

public static void main(String[] args) {
    Node head = new Node(0);
    head.next = head;
    insertValue(head, 5);
    insertValue(head, 3);
    insertValue(head, 4);
}
```

Questions on next page.
a) Implement the `insertValue` method (used in the `main` method) to insert a new value into the correct position in the list. This method should use a loop, not recursion.

```java
public static void insertValue(Node n, int x)
```

b) Re-implement the `insertValue` method using recursion, but no loops.

```java
public static void insertValue(Node n, int x)
```
Problem 4. a) Briefly describe how quick sort algorithm works. Given an array of integers: 3, 5, 9, 2, 6, 8, 1, 7, show the key intermediate steps of how the array gets sorted. Indicate the pivot value at each step. b) Analyze its runtime cost using recurrence relations.
Problem 5. A Binary Search Tree is a tree in which:

- Each node contains a value, and up to two children known as left and right.
- All nodes in the left subtree contain values strictly less than this node’s value.
- All nodes in the right subtree contain values strictly greater than this node’s value.

The `Node` class

```java
public class Node {
    int x;
    Node left;
    Node right;
    Node parent;
    public Node(Node p, int v){
        parent = p;
        x = v;
    }
}
```

The `addNode` method

```java
static void addNode(Node n, int v) {
    if (v < n.x) {
        if (n.left == null) {
            n.left = new Node(n, v);
        } else {
            addNode(n.left, v);
        }
    } else {
        if (n.right == null) {
            n.right = new Node(n, v);
        } else {
            addNode(n.right, v);
        }
    }
}
```

Example tree
1. Draw the binary search tree that the *main* below would generate.

```java
public static void main(String[] args) {
    Node root = new Node(null, 8);
    addNode(root, 3);
    addNode(root, 2);
    addNode(root, 1);
    addNode(root, 4);
    addNode(root, 9);
}
```

2. Write a method called *findBLT* which finds the biggest value strictly less than $x$ in a binary search tree. If there is no value less than $x$ the method should return $-1$. The *main* below gives examples of using this method against the example tree drawn on page 2. Note: $x$ is not necessarily in the tree. Also, assume all numbers in the tree are positive. Finally, the method should run $O(\log(N))$ i.e. do not traverse the whole tree.

```java
public static void main(String[] args) {
    Node root = new Node(null, 10);
    //...

    System.out.println(findBLT(root, 20)); //prints 19
    System.out.println(findBLT(root, 6));  //prints 4
    System.out.println(findBLT(root, 4));  //prints -1
    System.out.println(findBLT(root, 16)); //prints 15
}
```
Problem 6.

a) Write down the pseudocode (Java is OK) for the BFS algorithm. You can assume an adjacency list representation of the graph. List the sequence of vertices visited with your algorithm, starting from vertex A.

b) Show the execution of Dijkstra’s shortest path algorithm on the graph above, starting from vertex A. Show each vertex visited and shortest path length to it, and the length of the best paths found so far for each vertex after each update step.

<table>
<thead>
<tr>
<th>Next Vertex</th>
<th>Current Path Length to Vertex</th>
<th>Priority Queue contents (optional, but helpful)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 7. Hashing

a) What is hashing?

b) Describe a concrete drawback of using Hash tables to implement a lookup-up dictionary (e.g., look up actor names), compared to using Binary Search Trees.

c) Draw a representation of a 5-celled hash table $B$ (assume no dynamic re-sizing) and its contents after we use the hash function

$$h(x) = x \mod 5$$

to insert the elements in the set $\{12, 18, 23, 46, 5, 3, 4, 9\}$ into $B$, if we handle collisions with the separate chaining method.

Good luck!