You are to honor the Emory Honor Code and the Math/CS SPCA. This is a closed-book and closed-notes exam. You have 75 minutes to complete this exam.

Short answers

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>Earned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Long answers

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>10</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Earned</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Code completion

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>12</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>15</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>Earned</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total points
Short Answers

1. Describe situations when a lazy priority queue, an eager priority queue, and a binary heap is preferred over the others. (3 points)

2. Explain why heap-sort is considered a selection-based sorting algorithm. Describe how it achieves the $O(n \log n)$ complexity. (3 points)

3. Either the Knuth or the Hibbard sequence can be used in shell-sort. Describe a difference between the two sequences and explain why one sequence is preferred over the other as the input gets larger. (3 points)

4. Explain why the worst-case complexity of quick-sort is $O(n^2)$. Describe a way of improving quick-sort to avoid the $O(n^2)$ complexity. (3 points)

5. What is the worst-case complexity of merge-sort? Explain why merge-sort often performs slower than quick-sort for randomly ordered input. Describe a situation where merge-sort has significant advantage over quick-sort. (3 points)

6. What is the worst-case complexity of bucket-sort (in terms of comparisons) when the input contains only integer values? What about when the input contains double values? (3 points)

7. Describe how an AVL tree keeps it balanced after a node is added. (3 points)

8. Dynamic tables can be represented as arrays or maps. Describe two different ways of populating dynamic tables. Give an example of each way. (3 points)

Long Answers

9. Given the list of [12, 3.4, 20.1, 34.4, 1.3, 13.2], explain how it can be sorted by radix-sort using the least significant digit. Describe each step (which key goes into which bucket using which digit). (10 points)

10. Explain how the following tree can be balanced using left and right rotations. Describe each step (which node becomes a child or a parent of which node in which rotation). (10 points)
11. Complete the “balance” method in pseudo-code that takes a node whose parent and uncle are red and balances it to be a Red-Black tree. (15 points)

```java
interface Node
{
    /** @return the grand-parent if exists; otherwise, null. */
    Node getGrandParent();
    /** @return the parent if exists; otherwise, null. */
    Node getParent();
    /** @return true if the specific node is the left child. */
    boolean isLeftChild(Node node);
    /** @return true if the specific node is the right child. */
    boolean isRightChild(Node node);
    /** Sets the color of this node to black. */
    void setToBlack();
    /** Sets the color of this node to red. */
    void setToRed();
    /** Rotates this node to the left. */
    void rotateLeft();
    /** Rotates this node to the right. */
    void rotateRight();
}

void balance(Node node)
{
    // To be filled.
}
```
12. Complete the “remove” method in pseudo-code that takes a key and removes the nodes related to the key from the Trie. (15 points)

```java
interface TrieNode {
    /** @return the parent if exists; otherwise, null. */
    TrieNode getParent();
    /** @return the key. */
    char getKey();
    /** @return true if this node is an end state. */
    boolean isEndState();
    /** @return true if this node has children. */
    boolean hasChildren();
    /** Sets the end state to the specific boolean. */
    void setEndState(boolean b);
    /** Removes the child with the specific key from this node. */
    void removeChild(char key);
}

/** @return the node with the key if exists; otherwise, null. */
abstract TrieNode find(String key);

public void remove(String key) {
    TrieNode node = find(key);
    // To be filled.
}
```