CS323: Data Structures and Algorithms
Final Exam (May 4, 2015)

Name: ________________________________

You are to honor the Emory Honor Code and the Math/CS SPCA. This is a closed-book and closed-notes exam. You have 150 minutes to complete this exam.

Concept questions

<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>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pseudo-code completion

<table>
<thead>
<tr>
<th></th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total points

________________________
Concept Questions

1. Describe how the “add” and the “remove” operations work in binary heaps. Give the worst-case complexities of these two operations. Explain how they achieve such complexities. (6 points)

2. Give the worst-case complexities of the bucket sort and the LSD radix sort when the keys are double values. Explain how they achieve such complexities. (4 points)

3. Explain when you should use AVL trees over Red-Black trees, and vice versa. (2 points)

4. Given the input of [5, 3, 4, 7, 6], describe how an AVL tree balances when each key is added in order. Describe how balance factors are used and rotations are performed. (6 points)

5. Give the worst-case complexities of solving the “Towers of Hanoi” problem using recursion and dynamic programming. (2 points)

6. Give one situation when you should use Prim’s algorithm over Kruskal’s algorithm. Explain your answer. (2 points)

7. Prove the correctness of Kruskal’s algorithm by induction. (6 points)

8. Describe how the Chi-Liu-Edmonds algorithm updates edge weights after detecting a cycle. Explain how this update leads to find the minimum spanning tree. (4 points)

9. Explain why the Ford-Fulkerson algorithm does not find an optimum solution when there is no backward pushing. Describe your answer with an example. (5 points)

10. Prove the max-flow min-cut theorem. (7 points)

11. Explain how to formulate the objective function and the constraints for linear programming to solve the min-cut problem. Explain the function of dual variables ($y$ variables). (6 points)

12. Prove that when a match $M$ in a graph $G$ is maximum, there is no augmenting path $P$ in $G$. (6 points)

13. Describe the complexity classes of P, NP, NP-complete, and NP-hard when P $\neq$ NP. (4 points)
**Pseudo-code Completion**

*Use only methods in provided interfaces, classes, and Java built-in APIs for all questions. Extra codes that either hurt or do not contribute to the assigned tasks will be penalized.*

14. Complete the “getShortestPath” method that returns the Integer array representing the shortest path between the source and the target in the graph using the Dijkstra’s algorithm. (10 points)

```java
class VertexDistancePair implements Comparable<VertexDistancePair> {
    public int vertex;
    public double distance;

    public VertexDistancePair(int vertex, double distance) {
        this.vertex = vertex;
        this.distance = distance;
    }

    @Override
    public int compareTo(VertexDistancePair p) {
        return (int)Math.signum(distance - p.distance);
    }
}

/** Initializes the distances and previous vertices. */
abstract void init(double[] distances, Integer[] previous, int target);

public Integer[] getShortestPath(Graph graph, int source, int target) {
    PriorityQueue<VertexDistancePair> queue = new PriorityQueue<>();
    Integer[] previous = new Integer[graph.size()];
    double[] distances = new double[graph.size()];
    Set<Integer> visited = new HashSet<>();

    init(distances, previous, target);
    queue.add(new VertexDistancePair(target, 0));

    while (!queue.isEmpty())
    {
        // To be filled.
    }

    return previous;
}
```
15. Complete the “solve” method that recursively stores all steps to the list for solving the “Towers of Hanoi” problem using dynamic programming. (15 points)

```java
/** @return the step representing the current state. */
abstract String getStep(int n, char source, char destination);

/** Populates the list with the items between fromIndex to toIndex. */
abstract void addAll(List<String> list, int fromIndex, int toIndex);

/** *
* @param list the list containing all steps.
* @param n the number of plates.
* @param source the source tower.
* @param intermediate the intermediate tower.
* @param destination the destination tower.
* @param map the dynamic table.
*/
void solve(List<String> list, int n, char source, char intermediate,
            char destination, Map<String, int[]> map)
{
    // To be filled.
}
```

16. Write a pseudo-code for finding an augmenting path (an alternating path that begins and ends at free vertices) using the Blossom algorithm. Your code must take a graph $G$ and a match $M$, and returns an alternating path $P$. (15 points)