CS323: Data Structures and Algorithms

Midterm Exam
October 21, 2014

Name: ___________________________________________

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>9</td>
<td>16</td>
<td>6</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>Earned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>6</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>30</td>
<td>15</td>
<td>45</td>
</tr>
<tr>
<td>Earned</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Earned</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total points
Short Answers

1. (a) What are the worst-case and best-case complexities of the following sorting algorithms in terms of the number of comparisons? (0.5 point each, 3 points total)

<table>
<thead>
<tr>
<th></th>
<th>Insertion</th>
<th>Heap</th>
<th>Quick</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst-case</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best-case</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) What is the worst-case complexity of shell-sort using the Knuth-sequence? Give an example where shell-sort outperforms insertion-sort and explain why. (3 points)

(c) All the sorting algorithms in #1 are comparison-based. Give two sorting algorithms that are distribution-based. What is the key difference between comparison-based and distribution-based sorting algorithms? (3 points)

2. (a) What are the worst-case complexities of the following operations for unbalanced and balanced binary search trees? (0.5 point each, 3 points total)

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbalanced</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balanced</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Given k-number of intersecting intervals and n-number of nodes in a binary search tree, what is the worst-case complexity of finding all intervals? Explain why. (3 points)

(c) Explain how the AVL tree keeps itself in balance when 3 keys, {1, 3, 2}, are inserted in order. Your explanation must involve balance factors of AVL nodes. (5 points)

(d) Explain how the Red-Black tree keeps itself in balance when 3 keys, {3, 1, 2}, are inserted in order. Your explanation must involve colors of Red-Black nodes. (5 points)

3. (a) It is possible to traverse a Trie using depth-first or breadth-first search. Which search strategy outperforms for finding the k'th shortest word given a prefix? Explain why. (3 points)

(b) Each Trie node consists of multiple children representing followup character sequences. What is a problem of using arrays to store these children? What is an alternative way of storing them? What are the advantages of using this alternative way over arrays? (3 points)

4. (a) What are the worst-case complexities of solving the “Towers of Hanoi” problem using recursion and dynamic programming? (2 points)

(b) Which occasion should we use dynamic programming over recursion? How do we transform recursion into dynamic programming in general? (2 points)
Java Code Completion

5. Complete the “heapsort” and “sink” methods to perform heap-sort. 
   (15 points each, 30 points total)

```java
/** Sorts the array in ascending order using heap-sort. */
void heapsort(int[] array)
{
    int endIndex = array.length;
    // To be completed.
}

/**
 * Sinks the k'th item in the array.
 * @param endIndex the ending index of the array (exclusive).
 */
void sink(int[] array, int k, int endIndex)
{
    // To be completed.
}

/** Swaps array[i] and array[j]. */
void swap(int[] array, int i, int j)
{
    int t = array[i];
    array[i] = array[j];
    array[j] = t;
}

/** @return the parent index of k. */
int getParentIndex(int k)
{
    return k/2 - 1;
}

/** @return the left child index of k. */
int getLeftChildIndex(int k)
{
    return 2*k + 1;
}
```
6. Complete the “knapsack” method to solve the knapsack problem using recursion. (15 points)

```java
interface KnapsackItem
{
    /** @return the weight of this item. */
    int getWeight();
    /** @return the value of this item. */
    int getValue();
}

/** @return the list of items maximizing the total value. */
List<KnapsackItem> knapsack(KnapsackItem[] items, int maxWeight)
{
    Arrays.sort(items);
    return knapsack(items, maxWeight, items.length-1);
}

/**
 * @return the list of items maximizing the total value.
 * @param items     items to be entered into the knapsack.
 * @param maxWeight the maximum weight that the knapsack can hold.
 * @param index     the index of the item to be considered.
 */
List<KnapsackItem> knapsack(KnapsackItem[] items, int maxWeight, int index)
{
    if (index < 0 || maxWeight == 0) return new ArrayList<>();
    KnapsackItem item = items[index];

    if (item.getWeight() > maxWeight)
        return knapsack(items, maxWeight, index-1);
    else
    {
        // To be completed.
    }
}

/** @return the total value of the items. */
int getTotalValue(Collection<KnapsackItem> items)
{
    int total = 0;

    for (KnapsackItem item : items)
        total += item.getValue();

    return total;
}
```
Pseudo code completion

7. Complete the “traverse” method that traverses the trie using breadth-first search and returns the list of the top-20 shortest words with the specific prefix, sorted first by length then by alphabet in ascending order. (20 points)

```java
interface TrieNode {
    /** @return the map in which (keys and values) are (characters and children nodes), respectively. */
    Map<Character, TrieNode> getChildrenMap();
    /** @return true if this node is the end state of any word; otherwise, false. */
    boolean isEndState();
    /** @return the full character sequence from the root to this node. */
    String getForm();
}

interface Trie {
    /** @return the Trie node matching the prefix if exists; otherwise, null. */
    TrieNode findNode(String prefix);
}

/** @return the list of the top-20 shortest words with the prefix, sorted first by length then by alphabet in ascending order. */
List<String> traverse(Trie trie, String prefix) {
    TrieNode node = trie.findNode(prefix);
    List<String> list = new ArrayList<>();
    if (node == null) return list;
    // To be completed.
    return list;
}
```