## Solutions

### Question | Points | Score
--- | --- | ---
Short Answer | 35 | 
Execute a Sort | 10 | 
A Function | 15 | 
Induction (choose 1 of 2) | 15 | 
Balancing a BST | 15 | 
A Graph Algorithm | 10 | 
Total: | 100 |
1. (a) (5 points) Is the following quantification true or false? Explain your answer.

\[ \exists x \in \mathbb{R} \quad \forall y \in \mathbb{Z} \quad (x < y) \]

\[ \mathbb{R} \text{- real numbers} \]
\[ \mathbb{Z} \text{- integers} \]

**Solution:** FALSE. No matter what value of \( x \) that we choose, we can always find a \( y \) which is \( \leq x \).

(b) (5 points) In a Red-Black tree, we use widgets to simulate 2-3-4 nodes. Draw all of the possible widgets here. You don’t have to give any keys for the nodes, but be very clear which nodes are black, and which are red.

**Solution:**

```
black   black   black   black
/ \   / \   / \   / \\
red   red   red   red
```

(c) (5 points) What is the (normal, though not guaranteed) performance of an ‘insert’ operation in a hash table?

**Solution:** \( O(1) \)

(d) (5 points) Give a **recursive structural** definition for a non-empty singly-linked list (as if you were about to prove something about it inductively). Don’t forget the base case!

I’ll help you by starting it:

“A non-empty singly-linked list is...

**Solution:** either a single node, or a single node, attached to a non-empty singly linked list.
(e) (5 points) Name two of the rules (out of four) of a red-black tree.

**Solution:** Choose any 2 of the 4:

- Root is always black
- Virtual leaves: leaves are black, but store nothing. (Alternate explanation: a null node is black.)
- All leaves are at the same “black height”
- No adjacent reds

I don’t care about the specific wording of these rules, but I care a lot about **clarity**.

(f) (5 points) What is the best performance that is possible for any comparison sort (don’t explain why this is, simply give the performance).

**Solution:** $O(n \lg n)$

What is the best performance that is possible for **any** type of sort? Explain your answer.

**Solution:** $O(n)$
Any sort must examine every value in the array - thus, all sorts at least perform one operation per value.

(g) (5 points) Name an example of a stable sort:

**Solution:** Answers will vary. Some options: Merge Sort, Insertion Sort, Bubble Sort

and also an unstable sort:

**Solution:** Answers will vary. Some options: Quicksort, Selection Sort, Heap Sort

2. (10 points) Use radix sort to sort the following set of words. **Show your work after each step!**

```
bat
rad
bar
foo
fee
```
Solution:

<table>
<thead>
<tr>
<th>rad</th>
<th>rad</th>
<th>bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>fee</td>
<td>bar</td>
<td>bat</td>
</tr>
<tr>
<td>foo</td>
<td>bat</td>
<td>fee</td>
</tr>
<tr>
<td>bar</td>
<td>fee</td>
<td>foo</td>
</tr>
<tr>
<td>bat</td>
<td>foo</td>
<td>rad</td>
</tr>
</tbody>
</table>

**Instructor’s Reminder:** The first column was sorted by the last letter; the second column was sorted by the middle letter; the third column was sorted by the last letter.
3. (15 points) Suppose that we have declared a red-black node using the following class:

```java
class RBNode {
    int key;
    RBNode left, right;
    boolean isRed;
}
```

Write a function which takes the root of a red-black tree as its parameter. It must scan the entire tree, and return the number of widgets which have exactly 3 keys in them. You may find the provided helper function useful. (You’re not required to use it, but you may.)

```java
boolean isRed(RBNode node)
{
    return (node != null) && node.isRed;
}

int numFull(RBNode node)
{
    if (node == null)
        return 0;

    // VERSION A: longer but clearer
    if (isRed(node))
        return numFull(node.left) + numFull(node.right);
    if (isRed(node.left) && isRed(node.right))
        return 1 + numFull(node.left) + numFull(node.right);
    return numFull(node.left) + numFull(node.right);

    // VERSION B: Less writing, more efficient, but trickier
    int tmp = numFull(node.left) + numFull(node.right);
    if (isRed(node.left) && isRed(node.right))
        return tmp+1;
    else
        return tmp;
}
```
Choose only one of the induction problems to do. Do not do both - if you do, we will only grade one of them!

4. (15 points) Prove the following conjecture with structural induction: “A nonempty ternary tree of height $h$ has at most $\frac{3^{h+1} - 1}{2}$ nodes.”

(“ternary” means 3. So this is just like a binary tree, except that each node can have up to 3 children.)

Solution: Base:
The base case is a single node ($h = 0$).

$$\frac{3^{0+1} - 1}{2} = \frac{3 - 1}{2} = 1$$

Thus, the base case holds.

Inductive:
Assume that the conjecture holds for trees of height $h$. We will try to prove that the conjecture holds for trees of height $h + 1$.

A ternary tree of height $h + 1$ is a single node, plus (up to) 3 child trees, each of which has height (up to) $h$. The version with the most nodes, of course, is the one which actually has 3 child trees, and all are their largest versions.

By the I.H., we know how many nodes those child trees might have; thus, the total number of nodes in the tree is

$$1 + 3 \cdot \frac{3^{h+1} - 1}{2} + 3 \cdot \frac{3^{(h+1)+1} - 3}{2} + \frac{3^{(h+1)+1} - 3 + 2}{2}$$

Thus, the inductive step holds.
Choose only one of the induction problems to do. Do not do both - if you do, we will only grade one of them!

Prove the following conjecture using induction:

\[
\sum_{i=0}^{n} 7^i = \frac{7^{n+1} - 1}{6}
\]

for all non-negative \( n \).

**Solution:**

**Base:** \( n = 0 \)

LHS: \( \sum_{i=0}^{0} 7^i = 7^0 = 1 \)

RHS: \( \frac{7^{0+1} - 1}{6} = \frac{7 - 1}{6} = 1 \)

Thus, the base case holds.

**Inductive:**

Assume that the conjecture holds for some \( k \). We will attempt to prove that it holds for \( k + 1 \).

\[
\sum_{i=0}^{k+1} 7^i = 7^{k+1} + \sum_{i=0}^{k} 7^i
\]

By the I.H.:

\[
7^{k+1} + \frac{7^{k+1} - 1}{6}
\]

\[
6 \cdot 7^{k+1} + 7^{k+1} - 1
\]

\[
\frac{6 \cdot 7^{k+1} + 7^{k+1} - 1}{6}
\]

Thus, the inductive step holds.
5. (15 points) Each of the AVL trees below is imbalanced somewhere. Circle the node where the imbalance is detected, and then redraw the tree, to the right, to show how it would look after the required rotation(s).

Solution: The imbalance is at 20.

Solution: The imbalance is at foo.
Solution: The imbalance is at 1500.
6. (10 points) Run Prim’s Algorithm on the graph below, starting at node D. Show the spanning tree that you’ve selected by coloring the edges to make them much darker.

You are not required to show the contents of the priority queue as it changes - although you are welcome to do so if you want.