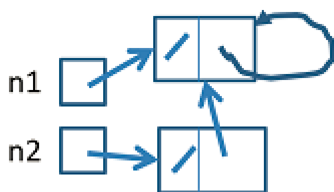
CS314 Fall 2021 Exam 3 Solution and Grading Criteria.

Grading acronyms:
AIOBE - Array Index out of Bounds Exception may occur.
BOD - Benefit Of the Doubt. Not certain code works, but, can't prove otherwise.
Gacky or Gack - Code very hard to understand even though it works. (Solution is not elegant.)
LE - Logic Error in code.
MCE - Major Conceptual Error. Answer is way off base, question not understood.
NAP - No Answer Provided. No answer given on test.
NN - Not Necessary. Code is unneeded. Generally, no points off.
NPE - Null Pointer Exception may occur.
OBOE - Off By One error. Calculation is off by one.
RTQ - Read The question. Violated restrictions or made incorrect assumption.

1. Answer as shown or -2 unless question allows partial credit. (Statements in parenthesis not required, only for explanation of answer to students.)
No points off for minor differences in spacing, capitalization, commas, and braces.

A. Insertion Sort
B. maxHeap
C. 4
D. 36 seconds
E. 2 black nodes in path to node that contains -3 child nulls, 3 black nodes in all other paths
F. {A=[3, 1, 4], C=[7, 10], J=[7, 10], N=[3, 1, 4]}
G. C
H. 12 bits
I. 40 seconds
J. Sparse
K. 2
L. Directed, Cyclic



M.

N. 4
O. 65 bits
P. 8 seconds
Q. 22, (21 - 23 accepted)
R. 80 seconds
S. Mergesort
T. [M, V, D, Z, /, /, A, S, J, /]
U. -2 7
V. h(x) = x
W. 5.5N + 4, +/- 1 on each
X. 12
Y. 16
EC 1: Any
EC 2: Dijktsra and Emerson (both for credit)

2.

```
private int helper(E tgt, Node<E> n) {
    int result = 0;
    // n won't be null due to initial precon and our logic below
    if (n.children != null) {
        // current node isn't a leaf
        for (int i = n.children.size() - 1; i >= 0; i--) {
            Node<E> child = n.children.get(i);
            if (child.children == null) {
                // leaf node
                if (child.data.equals(tgt)) {
                    // child is a leaf to be removed, swap last
                    result++;
                    size--;
                    Node<E> last =
                      n.children.remove(n.children.size() - 1);
                    if (i != n.children.size())
                        n.children.set(i, last); // otherwise last
                }
            } else {
                result += helper(tgt, child);
            }
        }
        // Am I a leaf now?
        if (n.children.size() == 0) {
            n.children = null;
        }
    } // else current node is a leaf, but not to be removed
    return result;
}
```

12 points, Criteria:

- Check current node is not a leaf (check children list isn't null) Can do this at start or inside loop and simply not recurse on leaves or both. 1 point
- loop through children correctly (for each okay, front to back or back to front okay), 1 point
  total -3 if Null Pointer Exception could occur, children could be null
- In loop, check if current child is leaf correctly (its children list is null), 1 point
- In loop, if child is leaf, correctly check if it has the target data, must use .equals, 1 point
- In loop, remove child from children list if criterion met. 1 point
- Removing a node from children list does not result in shifting. Lose if remove by position, by object, or iterator remove. 1 point
- decrement size on removal, 1 point
- correctly make recursive call on children nodes if not removed. (lose if early return), 3 points
- Add result of recursive calls to running total of number removed. (lose if early return), 1 points
- Return result, 1 point (odd ignore return except first of oldSize – size works)

Other deductions:

- [i] instead of get(i) for list, -2
- No attempt to remove from list, total of -3

3. Comments

```
private boolean verifyTree(BitInputStream bis) {
    int numCodes = bis.readBits(8);
    for (int code = 0; code < numCodes; code++) {
        int value = bis.readBits(9);
        int length = bis.readBits(8);
        int bitNum = 0;
        TreeNode n = root;
        while (n != null && bitNum < length) {
            int bit = bis.readBits(1);
            n = (bit == 0) ? n.left : n.right;
            bitNum++;
        }
        if (n == null) {
            return false; // fell off of tree
        }
        if (n.value != value) {
            return false; // wrong value in leaf
        }
        if (n.left != null || n.right != null) {
        // OR (!(n.left == null && n.right == null)
            return false; // not a leaf node
        }
    }
    return true; // no problems
}
```

13 points, Criteria:

- read number of code and loop for that number, 1 point
- for a given code, read bits for value and length of code, 1 point
- start at root of tree when checking code, 1 point
- read single bit from code and move left or right correctly, 3 points
- return false if fell off tree before completing code (null check, lose if NPE), 2 points (many students miss this)
- stop when moved correct number of times based on code, 1 point
- return false if value in leaf not correct, 1 point
- return false if code ends in node that is not a leaf, 2 points (most students miss this)

Other:

- missing return true if all codes present and correct, 1 point
- not checking internal nodes store value of -1, no points off, (Mike, note to self)

4. Comments:

```
public boolean hasHubs(int requiredHubs, double factor) {
    int totalEdges = 0;
    for (String name : vertices.keySet()) {
        Vertex vertex = vertices.get(name);
        totalEdges += vertex.adjacent.size();
    }
    double average = 1.0 * totalEdges / vertices.size();
    double degreeToQualifyAsHub = average * factor;
    int numHubs = 0;
    for (String name : vertices.keySet()) {
        Vertex v = vertices.get(name);
        int degree = v.adjacent.size();
        if (degree >= degreeToQualifyAsHub) {
            numHubs++;
            if (numHubs == requiredHubs) {
                return true;
            }
        }
    }
    return false;
}
```

12 points, Criteria:

- accumulator variable for total number of edges, 1 point
- correctly access all vertices from map when calculating total edges, 2 points
- correctly add size of adjacent list to accumulator, 1 point
- calculate average degree (lose if int div), 1 point (works if accumulator is double)
- var to count number of hubs, 1 point
- access all vertices again to determine number of hubs, 1 point
- correctly determine if a given vertex is a hub, 2 points
- return as soon as we have the required number of hubs (lose if keep going), 2 points
- return false if not enough hubs, 1 point

Other:

- creating new data structures (calls to values(), keyset(), iterator() okay), -2
-

5.

```
private void add(String value) { // Noah Solution
    Node lead = FIRST;
  // store trailer node, because the final node we add will be a "leading" 0
    Node trailer = lead;

    int carry = 0;
    for (int i = value.length() - 1; i >= 0; i--) {
        int digit = value.charAt(i) - '0';
        carry = addVal(lead, carry + digit);
        if (lead.next == null) {
            // deal with leading 0 later
            lead.next = new Node(0);
        }
        trailer = lead;
        lead = lead.next;
    }

// Account for final carry, can potentially overflow remaining digits in list
    while (carry == 1) {
        carry = addVal(lead, carry);
        // carry could still be 1, deal with leading 0 later
        if (lead.next == null) {
            lead.next = new Node(0);
        }
        trailer = lead;
        lead = lead.next;
    }

    // deletes the leading 0 we add on line 14/25
    if (carry == 0) {
        trailer.next = null;
    }

}


// Adds val n.digit and return the carry
private int addVal(Node n, int val) {
    n.digit += val;
    int result = n.digit / 10;
    n.digit %= 10;
    return result;
}
```

```
Single Loop Solution:
public void add(String value) { // Nina Solution
    Node trailer = null;
    Node currNode = FIRST;
    int i = value.length() - 1;
    int carry = 0;

    while(i >= 0 || carry > 0) {
        int add = carry;
        if(i >= 0) {
            add += value.charAt(i) - '0';
        }

        if(currNode == null) {
            currNode = new Node(add);
            trailer.next = currNode;
        } else {
            currNode.digit += add;
        }

        carry = currNode.digit / 10;
        currNode.digit %= 10;

        trailer = currNode;
        currNode = currNode.next;
        i--;
    }
}
```

13 points, Criteria:

- Temp node that starts at FIRST, 1 point
- loop all elements of String (okay if wrong way), 1 point
- get correct digits in String for current Node. (String is Big Endian, List is Little Endian), 1 points
- calculate and update digit for Node, 2 points
- Calculate and apply carry, 1 point
- Add and link new nodes as necessary (lose if NPE), 3 points
- Update (move) reference(s) temp Node through linked structure, 3 points
- Handle possible carry past end of String, 1 point

Other

- Attempt to convert list and string to ints, add those, and rebuild list. -5. (The whole purpose of the LinkedBigInteger is to represent ints that are beyond the bounds of int or long