0CS314 Spring 2022 Exam 3 Solution and Grading Criteria.

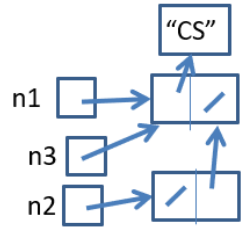
Grading acronyms:
AIOBE - Array Index out of Bounds Exception may occur.
BOD - Benefit Of the Doubt. Not certain code works, but, can't prove otherwise.
Gacky or Gack - Code very hard to understand even though it works. (Solution is not elegant.)
LE - Logic Error in code.
MCE - Major Conceptual Error. 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.
No points off for minor differences in spacing, capitalization, commas, and brace unless noted.

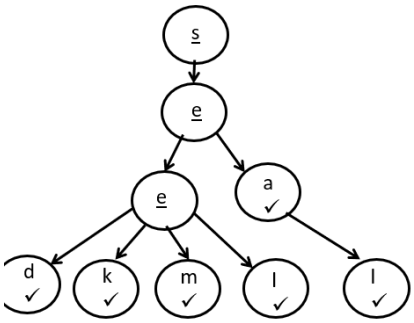
A. 36 seconds
B. 80 seconds
C. 4 seconds
D. 4
E. 12
F. D
G. A
H. 9
I. E
J. $3N^2 + 5N + 4$, +/- 1 each coefficient
K.

```
          5
        /   \
       8     5
      / \   /
     12 12  5
```

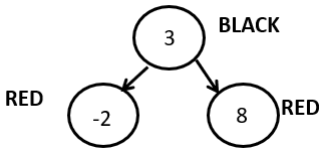
L. O(N)
M. Selection, Quicksort (1 point each, -1 per other to 0 points)
N. 65



O.
P. false true
Q. 3 0 -5 12 17 7



R.
S. O(N)
T. $O(N^2)$
U. $O(N^3)$



V.
W. B
X. false
Y. 20

2. Comments

```
private boolean helper(String currentName) {
    Vertex currentVertex = vertices.get(currentName);
    int currentColor = currentVertex.scratch;
    int adjacentcolor = (currentColor == 1) ? 2 : 1;
    for (Edge e : currentVertex.adjacent) {
        Vertex neighbor = e.dest;
        if (neighbor.scratch == currentColor) {
            return false; // same color adjacent
        } else if (neighbor.scratch == 0) {
            // Neighbor is uncolored, so color it and recurse
            neighbor.scratch = adjacentcolor;
            if (!helper(neighbor.name)) {
                return false;
            }
        }
        // else neighbor is other color, so just keep going.
    }
    // Never got back false. Success!
    return true;
}
```

18 points, Criteria:

- clear explanation, 3 points (partial credit possible)
-
- get current Vertex from map, 1 point
- loop through all edges, 2 points
- any edge colored same as this, return false 3 points
- return false right away when adjacent colors the same, 1 point
- if neighbor is uncolored, recurse on it, 2 points
- if get back false from recursive call, return false, 3 points
- color Vertex correctly using scratch and correct color (opposite of neighbors), 2 points
- return true after loop if coloring correct, 1 point

Other deductions:

- flip true / false, -4
- early return anywhere, -5
- call help but ignore return value, -5
- all scratch set to same, -4

3. Comments:

```
public int depthSum() {
        return help(root, 0);
    }

    private int help(IntNode n, int depth) {
        if (n == null) {
            return 0;
        } else {
            return depth * n.val
                    + help(n.left, depth + 1)
                    + help(n.right, depth + 1);
        }
    }
```

14 points, Criteria:

- create private helper method, 1 point
- helper parameters, only current node and current depth, 2 points (lose if pass in cumulative sum variable)
- correctly call method with root and 0 (zero), 1 point (okay if treat empty as special case and / or just call on children of root)
- base case present (null or leaf node acceptable), 2 points
- account for current node value and depth added and / or returned as part of sum, 2 points
- recursive calls to left and right children, 2 points
- new depth in recursive call, 2 points
- add result of recursive calls and return that. 2 points

Others:

- altering tree, -5
- Creates new array, even length 1, -4
- infinite loop due to while n != null instead of if, -4
- Return anything other than 0 for empty tree (not a special case), -2

4. Comments:

```
public HuffmanTree(String[][] codes) {
 // have to make the root
 root = new TreeNode(-1, -1);
 numLeaves = codes.length;
 for (int row = 0; row < codes.length; row++) {
      int value = Integer.parseInt(codes[row][0]);
      String code = codes[row][1];
      TreeNode n = root;
      for (int i = 0; i < code.length(); i++) {
           char bit = code.charAt(i);
           if (bit == '0') {
                if (n.left == null)
                     n.left = new TreeNode(-1, -1);
                n = n.left;
           } else {
                // current bit must be a '1'
                if (n.right == null)
                     n.right = new TreeNode(-1, -1);
                n = n.right;
           }
      }
      // n must be referring to the leaf node for the value.
      n.value = value;
 }
```

18 points, Criteria:

- create root node, 1 point
- loop through rows (codes) of parameter. for-each loop okay, 2 points
- for given code loop through characters (bits), 2 points
- temp node that starts at root of tree to move to new leaf node, 1 point
- check if moving left (a '0') or right (a '1') correctly, 2 point
- correctly create new nodes when necessary and only when necessary, 3 points
- all nodes frequency set to -1, internal nodes value set to -1, 1 point
- move temp node reference to next node (lower in tree), 3 points
- set value of leaf nodes, call Integer.parseInt on correct element from codes 2d array (1 point each)
- set numLeaves instance variable. Can be at start or for each leaf added, 1 point

Other:

- triple (or quad) loop, -2
- recursion, -4
- substring, -2
- create a HuffmanTree object or return (this is the constructor), -1