CS314 Spring 2023 Exam 2 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. Lack of Zen.)
LE - Logic Error in code.
MCE or GCE - 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.
EFF - Efficiency. Order is worse than expected or unnecessary computations done.

1. Answer as shown or -2 unless question allows partial credit.

First use of quotes in output is wrong, then error carried forward.

No points off for minor differences in spacing, capitalization, commas, and braces.

Text in parenthesis not required. It is simply grading guidance and / or a brief explanation for answer.

A. -5

B. $5N^2 + 6N + 5$ (+/- 1 on each coefficient allowed)

C. O(N) (1 + 2 + 4 + 8 + … + N/4 + N/2) = (N - 1) operations

D. Runtime error (ClassCastException)

E. $O(N^3)$ (or infinite loop due to logic error in inner loop)

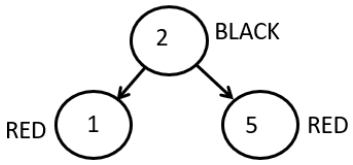
F. 0 6 9

G. $O(N^2)$ (substring is O(N))

H. quick sort

I. -3 0 2 5

J. The dreaded case 2, double rotation. 2 and 5 and then 2 and 1



K. 84 seconds

L. Always

M. APPELLEE

N. adj matrix $O(V^2)$ 1 point hash map of adj lists O(V) 1pt

O. A D C E B

P. 128 seconds

Q. 32 seconds

R. 512 seconds

S. $O(VE^{1/2})$ (OR V square root of E of course)

T. 10

U. O(N)

V. sometimes

W. false

X. 3 6 9 9 5 9

Y. level order traversal

Extra Credit: Any valid assignment (+2 points)

2. Comments:

```
    private void hopHelp(ArrayList<String> result, Vertex currentVertex,
                    int goalEdges, int currentEdges) {
        // Failure base case, already been here, nothing to do.
        if (currentVertex.scratch == 0) {
            // success?
            currentVertex.scratch = 1;
            if (goalEdges == currentEdges) {
                // this way we do NOT go beyond goalEdges
                if (!result.contains(currentVertex.name)) {
                    result.add(currentVertex.name);
                }
            } else { // we know goalEdges < currentEdges
                currentEdges++;
                for (Edge e : currentVertex.adjacent) {
                    hopHelp(result, e.dest, goalEdges, currentEdges);
                }
            }
            currentVertex.scratch = 0; // undo for other paths
        }
    }
```

17 points, Criteria:

- failure base case, current edges > goal edge, do nothing, 2 points
- failure base case, already been to this vertex in current path, 2 points
- recursive case, set scratch to 1, 2 point
- success base case, currentEdges == goalEdges, 2 points
- in success base case check !already present and add, 2 points
- recursive case, loop neighbors of current, 2 points
- recursive case, make recursive call with currentEdges on more, 3 points
- recursive case, undo scratch so we can check other paths through this vertex, 2 points

Other:

- ++ local copy of current edges (logic error, -2
- calling clearAll, logic error, -2
-

3. Comments: A fairly simple tree question

```
public int inRangeSum(int[] valueRange, int[] depthRange) {
    return helper(root, 0, valueRange, depthRange);
}

private int helper(IntNode n, int currentDepth,
                           int[] valueRange, int[] depthRange) {
    if (n == null || currentDepth > depthRange[1]) {
        return 0;
    }
    // We won't go below the max depth, so only need to worry
    // if current value is in range.
    int result = 0;
    if (currentDepth >= depthRange[0] && valueRange[0] <= n.val
            && n.val <= valueRange[1]) {
        result += n.val;
    }
    result += helper(n.left, currentDepth + 1, valueRange, depthRange)
          + helper(n.right, currentDepth + 1, valueRange, depthRange);
    return result;
}
```

16 points, Criteria:

- make a helper that returns an int, 1 point
- base case for current node is null, 2 points
- base case for beyond max depth, 2 points
- return 0 in base case, 1 point
- recursive case, check current depth is within range, 2 points
- recursive case, check current value is in range and if so add to result, 2 points
- correct recursive calls to left and right children, 3 points
- add results of recursive calls, 2 points
- return result, 1 point

Other:

- parameter for result instead of local variable if leads to logic error, -4
- change array, -3
- array for sum, no other classes, -4
- infinite loop, -4
- stop early, -4
- change the tree, -6

4. Comments:

```
    public boolean add(Object o) {
        int oldSize = size;
        // Determine the bucket o should be at.
        int index = o.hashCode();
        index = Math.abs(index);
        index %= con.length;
        if (con[index] == null) {
            // Empty bucket!
            con[index] = new Node(o, null);
            size++;
        } else {
            // Not an empty bucket. I'll use a trailer reference.
            Node lead = con[index];
            Node trail = null;
            boolean found = false;
            while (lead != null && !found) {
                if (lead.data.equals(o)) {
                    // The Object o is already here!!!
                    found = true;
                } else {
                    // Go on to next node.
                    trail = lead;
                    lead = lead.next;
                }
            }
            if (!found) {
            // Did not find an object equal to o in the chain.
            // Add to then end. trail is referring to the old last node.
                trail.next = new Node(o, null);
                size++;
            }
        }
        // Should we resize?
        if (1.0 * size / con.length >= LOAD_LIMIT) {
            resize();
         }
        return size != oldSize;
    }
```

17 points, Criteria:

- determine index to place value, 3 points (hashCode, %, Math.abs 1 point each)
- Special case for first element in bucket, 2 points
- correctly check all nodes if necessary (lost if OBOE), 3 points (while loop and condition correct)
- while checking, stop if find equal element, 2 points
- move reference(s) correctly, 2 points
- add to chain if necessary (could be first or last!), 2 points
- resize if necessary, can be before or after, 1 point (resize method already exists!) (lose if int div)
- update size, 1 point
- return correct value, 1 point

Other: Infinite loop, -4 O(N), checking all elements in array or table, -4 uses recursion -2
destroy other elements in chain, -4 NPE -3

Clever solution that adds new element to first element of chain. Question stated order in the chain did not matter. Eliminates special case for first in chain as well.

```
public boolean add(Object o) {
    int oldSize = size;
    // Determine the bucket o should be at.
    int index = o.hashCode();
    index = Math.abs(index);
    index %= con.length;
    Node temp = con[index];
    boolean found = false;
    while (lead != null && !found) {
        if (lead.data.equals(o)) {
            // The Object o is already here!!!
            found = true;
        }
        temp = temp.next;
    }
    if (!found) {
        // Did not find an object equal to o in the chain.
        // Add new node and make the first in chain.
        con[index=] = new Node(o, con[index]);
        size++;
    }
    }
    // Should we resize?
    if (1.0 * size / con.length >= LOAD_LIMIT) {
        resize();
     }
    return size != oldSize; // did size change?
}
```