CS314 Spring 2024 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. Lack of Zen.)
LE - Logic Error in code.
MCE - Major Conceptual Error. Answer is way off base, question not understood based on answer provided.
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. 94-121

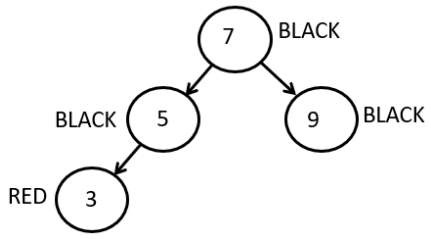
B. $3N\log_2 N + 4\log_2 N + 4$ (+/- 1 on EACH coefficient allowed. base 2 not required. No N term.)

C. H D I N

D. 20 seconds (code is $O(N^2)$)

E. 22 seconds (TreeSet uses Red-Black tree, code is O(NlogN)

F.



G. [1003, 124, 725, 131, 99]

H. A N P V

I. 120 bits (with 10 internal nodes there will be 11 leaf nodes)

J. A

K. O(logN) (base 3 okay)

L. B

M. O(NlogN) (base 2 okay, Java PQ uses a min heap)

N. sorted

O. 3

P. B and C (1 point each) -1 for each missing, -1 for each other than B, C min score 0

Q. [(0, 4], /, (3, 4), /, (4, 10)] (arrows to objects okay)

R. 10

S. 10 seconds (method is O(N))

T. 4 times

U. $O(VE^2)$

V. A

W. Yes

X. No path exists from G to A

Y. [12, 24]

Extra Credit: +2 for any valid answer

2. Comments:

```
public int count(E tgt, int minDepth) {
    return help(root, tgt, 0, minDepth);
}

private int help(BNode<E> n, E tgt, int currentDepth,
                 int minDepth) {
    if (n == null)
        return 0;
    int result = 0;
    if (currentDepth >= minDepth && n.left != null
            n.right != null && n.data.equals(tgt)) {
        result++;
    }
    int newDepth = currentDepth + 1;
    return result + help(n.left, tgt, newDepth, minDepth)
            + help(n.right, tgt, newDepth, minDepth);
}
```

15 points, Criteria:

- create helper method, 1 point
- helper returns int, 1 point
- base case, current node is null, return 0, 3 points
- check if current depth >= required depth, 1 point
- if so check data equals target (must be after check depth, efficiency) , 1 point
- check if node has 2 children, 2 points
- if conditions met, count this node in total, 1 point
- recursive calls, 3 points
- pass new depth correctly, 1 point
- return result of recursive calls and local result, 1 point

Other deductions:

- early return of 1 when condition met, -3
- sum variable as parameter, overcount logic error, -4
- call helper without using results, -4
- new data structures, (even array of length 1), -4
- alter tree, -5
- NPE possible for reason not covered here, -3

3. Comments: Not required to handle case when multiple * in a row. Question did not state what to do in that case, so can be ignored.

```
public String decode(String s) {
    // Check special case that does not end with *.
    if (s.charAt(0) == '* || s.charAt(s.length() - 1) != '*')
        return null;
    String result = "";
    MNode n = root;
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (c == '*') {
            // End of letter.
            // Could add check here if still on root, return null.
            result += n.letter;
            // go back to root
            n = root;
        } else if (c == '.')
            n = n.left;
        } else if (c == '-') {
            n = n.right;
        } else
            // BAD CHAR
            return null;
        // If we walked off tree, bad code.
        if (n == null)
            return null;
    }
    return result;
}
```

19 points, Criteria:

- return null if starts with *, 1 point
- return null if last char not *, 1 point
- Local node variable that starts at root, 1 point
- loop through characters of String, 2 points (can use nested to search for *)
- if current char ., go to left child, 2 points
- if current char _, go to right child, 2 points
- if current char *, add letter in current node to result, 2 points
- when add letter, reset node to root, 3 points
- if current char a bad char, return null, 2 points (lose if The ch != '.' || ch != '-' || ch != '*' logic error)
- if walked off tree, bad code, return null, 2 points
- return correct result after loop, 1 point

Other:

- recursion -3
- substring and / or contains -3
- equals on chars, -2
- infinite loops, -4
- create new data structure besides Strings, -4

4. Comments: A repeat question from 2013 AND essentially the same question as exam 2's recursive backtracking question. Just a different base case and using an explicit graph.

```
private boolean helper(String currentVertexName, int verticesInPath) {
    Vertex current = vertices.get(currentVertexName);
    // been here before?
    if (current.scratch == 1) {
        return false;

    // now we are here
    current.scratch = 1;
    verticesInPath++; // We are in the path.
    // Is this Vertex the last Vertex in the path?
    if (verticesInPath == vertices.size()) {
        return true;

    // Recursive case, check my edges. I am part of the path now.
    for (Edge e : current.adjacent) {
        if (helper(e.dest.name, verticesInPath)) {
            return true;
        }
    }
    // undo being here in case different route can work
    current.scratch = 0;
    return false;
}
```

16 points, Criteria:

- access Vertex object from map with name correctly, 1 point
- check failure base case, been to this Vertex before, 2 points (can be in loop before making recursive call)
- count current Vertex in the path, 1 point
- check success base case, number of vertices visited equals size of map, 2 points
- Mark scratch for current Vertex, 2 points
- loop through current Vertex's edges, 1 point
- make recursive call, 2 points
- return true if recursive call succeeds, 2 points
- undo scratch after tried all edges, 2 points
- return false in case when at a dead end after trying all edges, 1 point

Other:

- early return, -5
- infinite recursion, -3
- Creating new data structures, -4