CS314 Spring 2014 Final 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.)
GCE - Gross Conceptual Error. Did not answer the question asked or showed fundamental misunderstanding
LE - Logic error in code.
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.

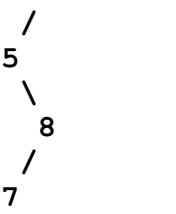
1. Answer as shown or -1 unless question allows partial credit.
No points off for minor differences in spacing, capitalization, commas, and braces OR missing O( )
If """ included in output -1 first occurrence then error carried forward.

A. **34**

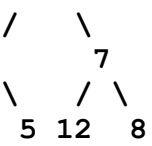
B. **10531!!2225**
(extra spaces okay)

C. **13**

D. **O(2^N)**
(any base from 1.6 to 2 okay)

E.


F. **-5 7 8 5 12**

G.


H. **O(N)**

I. **O(NlogN) base 2 okay**

J. **O(N)**

K. **1. 5 appears twice in tree**
2. one path with 3 black nodes and all others have 2 black nodes

L. **ArrayLists do not implement the Comparable interface**

M. **-2 AA 5 C 9 AA**

N. **W X D X K C**
(spaces or not okay)

O. **15**

P. **E**

Q. **O(N^3)**

R. **5 bits (just 5 okay)**

S. **equals method not overridden in Property**

T. **4**

2. Comments. Simple linked list question.

Suggested Solution:

```
public void insertAfterNth(E newValue, E target, int n) {
     int count = 0;
     Node<E> temp = first;
     while(temp != null && count < n) {
          if(temp.getData().equals(target))
               count++;
          if(count < n)
               temp = temp.getNext();
     }
     if(temp != null)
          temp.setNext(new Node<E>(newValue, temp.getNext()));
}
```

15 points , Criteria:

- temp node set to first, 1 point
- counter, 1 point
- loop that goes until correct number found or no more nodes, 3 (okay to return from middle of loop)
- check current item equal target, 2 (lose this if ==)
- increment count, 1
- move temp correctly and only if appropriate, 4 points
- add new node correctly just once, 3 points (lose this if add multiple times)

Other:

disallowed classes or methods: -3
Worse than O(N): -3
off by one error due to moving one past correct node, -2
doesn't stop after inserting, -2 EFFECIENCY
recursive solution, O(N) space, -3
using members instead of methods, -1
calling non existent methods, -2

3. Comments:

Common problems:

- using && instead of || on logic
- not handling empty case
- not handling case when target == 0 correctly (0 length path not allowed)
- assuming path must go all the way to a lead
- stopping too early. With negative or positive numbers possible in lower levels, it is a logic error to stop if temporary sum is greater than target or target is less than 0.

Suggested Solutions:

```
public boolean pathFromRootExists(int target) {
    return helper(root, target);
}

private boolean helper(BinaryNode n, int target) {
    if(n == null)
        return false;
    else {
        target -= n.getData();
        if(target == 0)
            return true;
        else
            return helper(n.getLeft(), target)
                || helper(n.getRight(), target);
    }
 }
```

15 points , Criteria:

- create helper, 1
- base case, node is null, return false, 3
- if node not null, update target or sum, 3
- base case, target = 0 or sum equal to target, return true, 3
- recursive call with left and right, 3
- correct logic on recursive calls, OR not AND, 2

Other:
DISALLOWED METHODS -3 per
early return when target <= 0 or sum exceeds target -3
using an array, even of size 1, -1
assuming path must reach leaf, -3

4. Comments:

Common problems:

- Looping through the codes instead of reading from the file
- reading in more than 1 bit at a time
- writing the PEOF to the output file
- writing past the PEOF
- not calling containsKey or get correctly

Suggested Solution:

```
public static void uncompress(Map<String, Integer> codes,
                BitInputStream in, BitOutputStream out) {

    String temp = "";
    boolean keepGoing = true;
    while(keepGoing) {
        temp += in.readBits(1); // String concat works with anything
        if(codes.containsKey(temp)) {
            int value = codes.get(temp);
            if(value != PSEUDO_EOF) {
                temp = "";
                out.writeBits(BITS_PER_WORD, value);
            }
            else
                keepGoing = false; // just read in PEOF
        }
    }
    out.close();
    in.close();
}
```

15 points, Criteria:

- keep reading bits until PEOF, 3
- read 1 bit at a time, 4
- concat 0 or 1 as appropriate, 1
- check if code / key present, 2
- output value correctly, 2
- reset code, 2
- close, 1

Other deductions:

- write out PEOF to file, -2
- skips bits, -3
- write past the PEOF, -3

5. Comments: Classic recursive backtracking problem given the constraints

Common problems:

- returning early
- trying an N^2 iterative solution (only going out two links)

Suggested Solution:

```
private boolean connectedHelper(Vertex currentVertex, int[] numVisited) {
    if(numVisited[0] == vertices.size())
        return true;
    else if(currentVertex.scratch != 0)
        // been here before
        return false;
    else {
        // visit this vertex
        currentVertex.scratch = 1;
        numVisited[0]++;
        for(Edge e : currentVertex.adjacent) {
            if(connectedHelper(e.dest, numVisited))
                return true;
        }
        return false;
    }
}
```

15 points, Criteria:

- base case all visited, 3 points (efficiency?)
- base case, visited vertex before?, 3 points
- recursive case:
- mark vertex visited, 1 point
- increment num visited, 1 point
- loop through edges, 2 points
- recursive call and return if true, 3 points
- return false if no solution, 2 points

Common Deductions:

Early return -5

6. Comments: Not stated but based on assignment we know the min is in the left most node

Common problems:

- not handling empty case
- not altering root node when appropriate (when it is the min / left most node)
- not removing node with min from tree
- moving past node with min

Suggested Solution:

```
public E deqeue() {
     E result = null;
     if(root != null) {
          if(root.left == null) {
               result = root.data;
               root = root.right;
          }
          else {
               BSTNode<E> temp = root;
               while(temp.left.left != null) {
                    temp = temp.left;
               }
               result = temp.left.data;
               temp.left = temp.left.right;
          }
     }
     return result;
}
```

15 points, Criteria:

- empty case, 2 points
- root case handled correctly, 3 points
- general case, temp at root, 1 point
- general case, move to left only, 2 points
- general case, move until just before left most node or use trailer, 3 points
- general case, remove node with min from tree correctly, 3 points
- return correct answer, 1 point

Common Deductions: