CS314 Fall 2021 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.)
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. 13
B. 19
C. 40
D. -1
E. INAA234
F. O(N^2)
G. O(N)
H. O(N) (sum of 1 + 2 + 4 + 8 + 16 + ... + N/4 + N/2 + N) = 2N - 1 -> O(N)
I. add(E val) only -1 for each other circled
J. 12 4 9 3
K. None are true (blank counted as wrong)
L. 5 4 3 (size cross over)
M. O(N^2) (near best case insertion sort)
N. O(N^2logN) (base 2 okay, mergesort always NlogN)
O. {0=5, 5=4, 7=3}
P. O(N^2)
Q. Mergesort
R. 4
S. T V D C J K X
T. {2=4, 3=2, 5=3}
U. compile error (cannot add String when E is String[])
V. O(NlogN) (base 2 okay)
W. A and B only. -1 per missing, -1 per extra
X. 40 seconds
Y. runtime error (Stack overflow due to not hitting base case any time soon)

2.
```java
public static int updateAuthors(Map<String, List<String>> map,
            String[] authors) {
        if (authors.length == 1) {
            return 0;
        }
        for (String author : authors) {
            List<String> collabs = map.get(author);
            if (collabs == null) {
                collabs = new LinkedList<>();
                map.put(author, collabs);
            }
            for (String otherAuthor : authors) {
                if (!author.equals(otherAuthor)) {
                    collabs.add(0, otherAuthor);
                }
            } // collabs is a reference so no need to put back
        }
        // now do the second part
        int result = 0;
        String first = authors[0];
        String last = authors[authors.length - 1];
        for (String name : map.get(first)) {
            if (name.equals(last)) {
                result++;
            }
        }
        return result;
    }
```
18 points, Criteria:

- return 0 right away if single author paper, 2 points
- outer loop for authors of paper (for each loop okay), 2 points
- create LinkedList **only if current author not already in map**, and put in map with current author as key ,2 points (avoid creating unnecessary objects)
- if author was present, correctly access list of co-authors with get method, 2 points
- Only access list for authors present one time, not once for each co-author, 1 point
- add co-authors to this authors list of co-authors, adding at front, 2 point
- avoid adding current author as co-author for their selves, 1 point

- determine first and last author, 1 point
- access list of co-authors from first or last author from map, 2 points
- loop through list of co-authors (for each okay), 1 point
- correct check and logic for incrementing number of times first and last author of current paper have collaborated, must use .equals method on Strings, 1 point
- track and return result, includes creating and initializing cumulative sum variable, 1 point

Other deductions:
alter elements of authors, array of Strings, -5
remove elements during count, -4
creating data structures besides lists to add to map for first time authors, -4
disallowed methods, (-1 to -5 depending on severity)

3. Comments

Simplest?

```
public LinkedList314<E> interleave(LinkedList314<E> other) {
    LinkedList314<E> result = new LinkedList314<>();
    // add the first two elements. Precon, neither list empty
    result.first = new Node<>(first.data);
    result.first.next = new Node<>(other.first.data);
    Node<E> thisTemp = first.next;
    Node<E> otherTemp = other.first.next;
    Node<E> resultTemp = result.first.next;
    // Now add until both used up.
    while (thisTemp != null || otherTemp != null) {
        if (thisTemp != null) {
            resultTemp.next = new Node<>(thisTemp.data);
            resultTemp = resultTemp.next;
            thisTemp = thisTemp.next;
        }
        if (otherTemp != null) {
            resultTemp.next = new Node<>(otherTemp.data);
            resultTemp = resultTemp.next;
            otherTemp = otherTemp.next;
        }

    }
    return result;
}
```

18 points, Criteria:

- create resulting linked list, 1 point
- create first node in result and add first element or elements to result, 3 points
- while loop until one null (or combine with if inside until both null, don't need other loops.) Lose if use size, 3 points (miss last node, OBOE - 2)
- add nodes with data to result, 3 points
- advance references in lists, 4 points
- add left over data if one list is longer, 3 points (can be handled in single loop)
- return result, 1 point

Other:

- New data structures or arrays -4
- > O(N), -4 (typically due to a O(N) implemented as part of solution)
- NPE not covered by other criteria, -4
- not creating new nodes, -8 (If the lists share nodes, then future logic errors VERY likely)
- assuming add method available, - 9 (defeats the whole purpose of the question, creating and linking nodes)
- destroy or alter either implicit or explicit parameter, -5
- assuming Iterator available, -7
- assuming size instance variable, -4
- public methods that expose nodes, -3
- assuming header node, -3 (first would not be null when empty list if list had a header node)

LL Alternative (More complicated?)

```
    public LinkedList314<E> interleaveAlt(LinkedList314<E> other) {
        LinkedList314<E> result = new LinkedList314<>();
        Node<E> thisTemp = first;
        Node<E> otherTemp = other.first;
        // add the first two elements. Precon, neither list empty
        result.first = new Node<>(thisTemp.data);
        thisTemp = thisTemp.next;
        result.first.next = new Node<>(otherTemp.data);
        otherTemp = otherTemp.next;
        Node<E> resultTemp = result.first.next;
        // Now add pairs
        while (thisTemp != null && otherTemp != null) {
                resultTemp.next = new Node<>(thisTemp.data);
                resultTemp = resultTemp.next;
                thisTemp = thisTemp.next;
                resultTemp.next = new Node<>(otherTemp.data);
                resultTemp = resultTemp.next;
                otherTemp = otherTemp.next;
        }
        // one of thisTemp or otherTemp must be null
        addRest(resultTemp, thisTemp);
        addRest(resultTemp, otherTemp);
        return result;
    }

    private void addRest(Node<E> to, Node<E> from) {
        while (from != null) {
            to.next = new Node<>(from.data);
            to = to.next;
            from = from.next;
        }
    }
```

4. Comments:

```
public static void findWords(String word, Set<String> d,
             List<String> result) {

        if (word.length() > 0) {
            if (d.contains(word) && !result.contains(word)) {
                result.add(word);
            }
            for (int i = 0; i < word.length(); i++) {
                String reduced = word.substring(0, i) +
                        word.substring(i + 1);
                findWords(reduced, d, result);
            }
        }
    }
// Method is O(N!)
```

14 points, Criteria:

- Base case, empty String, do nothing. Can be in middle of loop. (Lose if check dictionary and / or resulting list for empty String) 1 point
- recursive case, check if current word in dictionary, 1 point
- recursive case, if word IS in dictionary, check that current word not already in result, 1 point
- recursive case, add new words to result (at end), 1 point
- recursive case, loop for length of String. (Can handle what appear as special cases elsewhere.), 2 points
- recursive case, create some reductions of current word (this is the partial credit part), 1 point
- recursive case, **correctly** create all reductions of current word, 2 points
- recursive case, correct recursive call (lose if infinite recursion, for example by sending in word unchanged or multiple, unnecessary calls. Lose if early return), 4 points
- Statement of order, must be O(N!), 1 point

Other:

- creating any Objects besides Strings, -3
- use of charAt (disallowed), -2
- checking present in result list with anything other than contains method, -3
- attempting to return anything at end of method (void method), -2
- output to System.out, -2
- adding a helper, -2