# CS314 Spring 2019 Exam 1 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 -1 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. $6N^2 + 2N + 5$, +/- 1 on each coefficient (no syntax error)
B. $O(N^2)$ (due to String concatenation)
C. O(NlogN) (base 4 okay)
D. $O(N^3)$
E. 135 seconds
F. 4.4 seconds (22/5 accepted)
G. 10,000 elements
H. [M, I, AS] (quotes -1, minor differences in spacing and brackets okay)
I. -5 2 [-5, 0, 5, 7] (minor differences in spacing and brackets okay)
J. Runtime error (ConcurrrentModification error due to list.add)
K. invalid valid
L. valid invalid
M. y: 3
N. false
O. true false
P. fac64 L:32
Q. 84 L:42
R. 3
S. No, interfaces cannot have instance variables (or words to that effect)
T. Yes (class implements the abstract method pay and inherits the toString method from Object)

2. Comments. A difficult problem due to moving elements around in the array. The obvious O(N^2) solution was accepted, although there is a clever O(N) solution. Common problems were starting from the front skipping over adjacent equal elements. (On first equal element, shift all elements after down one spot, increment loop control variable which skips over the second , adjacent equal element.) Another problem was not decrementing size in the loop while still shifting. This pulls nulls into the array which are then deferenced with the equals method.

```
    public int removeAll(E tgt) {
        int index = 0;
        int numRemoved = 0;
        while (index < size) {
            if (con[index].equals(tgt)) {
                remove(index);
                numRemoved++;
            } else {
                index++;
            }
        }
        return numRemoved;
    }

    private void remove(int index) {
        size--;
        for (int i = index; i < size; i++) {
            con[i] = con[i + 1];
        }
        con[size] = null;
    }
```

O(N) Solution:

```
public int removeAll(E tgt) {
     // this is also the number of elements not equal to tgt
     int indexToAdd = 0;
     for (int i = 0; i < size; i++) {
          if (!con[i].equals(tgt) {
               con[indexToAdd] = con[i];
               indexToAdd++;
          }
     }
     // now null out number from last element not equal to target to size
     for (int i = indexToAdd; i < size; i++) {
          con[i] = null;
     }
     // how many did we remove?
     int result = size - indexToAdd;
     // update size
     size = indexToAdd;
     return result;
}
```

20 points, Criteria:

- loop through elements of list, 5 points (-4 if con.length, -4 if skip elements due to for loop logic error)
- correctly check if current element equal target, 3 points (lose if ==)
- correctly removes element:
    - shifts elements correctly, 5 points (partial credit possible)
    - decrements size, 2 points
    - null out empty spot at end, 3 points
- return number of elements removed correctly, 2 points (variable declared, updated, and returned)

Other deductions:
skip logic error -4 (for loop, starting from front, not decrementing loop control var)
create new array, -5 (not allowed by question)
off by one errors, -2
No attempt to shift elements, -12
infinite loop due to logic error, -5

3. Comments: A relatively simple problem involving two GenericLists. I believe this is the first question on an exam where the code had to return the this reference in some cases. The problem was not clear on what to do in the case of a non-zero tie with the sizes the same so we accepted any approach.

```java
public GenericList<E> maxFrequency(GenericList<E> other, E tgt,
        int start) {
    int countThis = count(start, tgt);
    int countOther = other.count(start, tgt);
    if (countThis > countOther) {
        return this;
    } else if (countThis < countOther) {
        return other;
    } else {
        // tie
        if (countThis == 0) {
           return null;
        } else if (size < other.size) {
           return this;
        } else {
           return other;
        }
    }
}
private int count(int st, E tgt) {
    int result = 0;
    for (int i = st; i < size; i++) {
        if (con[i].equals(tgt)) {
            result++;
        }
    }
    return result;
}
```

20 points, Criteria:

- determine frequency of target from starting index for a list
    - loop from starting index, 2 points
    - stop at size of list, 3 points
    - call equals correctly, 2 points (lose if ==)
    - increment counter if match, 1 point
- correctly determine frequency of target for both lists, 4 points
- return reference to this if countThis larger, 3 points
- return other if countOther larger, 1 point
- return null if both counts equal 0, 2 points
- for non-zero tie correctly return reference to correct list, 2 points

Others:

- off by one errors, -3
- early return -3
- logic error on return -3
- assume con / array is list -3
- size() method without writing it, -2
- worse than O(1) space, -4
- worse than O(N) time, -5
- return an array instead of a list, -5
- null pointer exceptions possible, -4
- call size() method without writing, -2
- infinite loop logic error, -4

4. Comments: More of a Scanner question than a Map question. The question was not clear on what to do if the target word was in a sentence with the same word twice. For example:

should he be here when he is tired

If the target word is should there are two instances of he. It was acceptable to double count the word he or to try and deal with this possibility.

Biggest problem by far was using the contains method on strings. For example, if the target word was "her" then " should he be **her**e when he is tired" returns true even though the word "her" is not in that sentence. Necessary to create a String with spaces on either end.

```
public static Map<String, Integer> getSematicDescriptor(Scanner sc,
        String word) {

    String wordWithSpaces = " " + word + " ";
    HashMap<String, Integer> result = new HashMap<>();
    while(sc.hasNextLine()) {
        String sentence = sc.nextLine();
        if (sentence.contains(wordWithSpaces)) {
            Scanner lineScanner = new Scanner(sentence);
            while (lineScanner.hasNext()) {
                String nextWord = lineScanner.next();
                if (!nextWord.equals(word)) {
                    Integer freq = result.get(nextWord);
                    if (freq == null) {
                        result.put(nextWord, 1);
                    } else {
                        result.put(nextWord, freq + 1);
                    }
                }
            }
        }
    }
    return result;
}
```

20 points, Criteria:

- create result correctly, 3 points (-2 if TreeMap, efficiency)
- outer loop for lines, 2 points
- check line if it contains target word, 3 points (-2 if don't add spaces. for example "there".contains("the") returns true, but "the" may not be in sentence)
- inner loop for tokens, 2 points
- correctly check if current word doesn't equal target word, 3 points (-2 for ==)
- if first occurence, put frequency of 1, 3 points
- if subsequent occurrence, put old freq + 1, 3 points (lose if freq++ or don't call put, Integers immutable)
- return result, 1 point

Other:

- efficiency (triple loops typically, -3)

5. Comments: An interesting implementation of a new data structure. Biggest issues were stopping at size (elements are spread through out the array), not guarding against null which leads to a NPE, not stopping as soon as the number of elements equal to val equals max so we know we can't add, and not resizing if necessary

```
    // Less efficient in that it checks all elements in more cases.
    // We could stop when non null elements checked equals size!
    public boolean addIfFewerThan(E val, int max) {
        int indexNull = -1;
        int numFound = 0;
        for (int i = 0; i < con.length; i++) {
            if (con[i] == null) {
                indexNull = i; // spot to place if we add
            } else if (con[i].equals(val)) {
                numFound++;
                if (numFound == max) {
                    return false;// too many
                }
            }
        }
        // there weren't too many instances of val
        if (con.length == size) {
            resize();
             indexNull = size; // there were no nulls
        }
        con[indexNull] = val;
        size++;
        return true;
    }

    private void resize() {
        E[] temp = (E[]) new Object[con.length * 2];
        for (int i = 0; i < con.length; i++) {
            temp[i] = con[i]; // okay if null
        }
        con = temp;
    }
```

20 points, Criteria:

- skip null elements correctly, 3 points
- check if non null elements equal target correctly, 1 point
- count number of occurrences of target val correctly, 1 point
- return false as soon as number of elements equal to val equals max allowed, 3 points
- resize only if actually adding, 2 points (lose if check to resize at start)
- resize if necessary, 2 points
- resize method written correctly, 2 points (okay if create new E[cap] on exam)
- place val in a null element in con correctly, 3 points
- increment size if necessary, 2 points
- return correct answer, 1 point

Other:

- Worse than O(con.length), -4
- Always resize, even if not needed, -4
- No resize attempt, -5
- Null Pointer Exception due to x.equals(null) check, -2
- Using ArrayList, -5
- early return, -6
- N^2, -5