CS314 Spring 2022 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.
Text in parenthesis not required. It is simply grading guidance and / or a brief explanation for answer.

A. $3N^2 + 6N + 4$, +/- 1 on each coefficient
B. $O(N^2)$ (adding at front)
C. $O(N^2)$ (length of String dependent on number of elements)
D. 80 seconds
E. 20 seconds
F. [A, D, F, E, B] (quotes -2, minor differences in spacing and brackets okay)
G. 27 seconds
H. compile or syntax error (Interfaces do not have constructors.)
I. 3.2 seconds
J. [6, 7, 5, 8] (minor differences in spacing and brackets okay) Due to shallow copy
K. compile or syntax error (no calling object for static methods, this undefined)
L. $8N^2 + 4N + 5$, +/- 1 on each coefficient
M. unknown until run or words to that effect (we were very strict on this question)
N. [3, 1, 4] (minor differences in spacing and brackets okay)
O. Object does not have a size method or words to that affect.
P. $O(N^2)$
Q. [[], 2.72, []] (on this question we only took answer that matched the actual output exactly)
R. 21 seconds
S. compile or syntax error (cannot access private field outside the class)
T. Runtime error (due to NPE)
U. 1 6
V. 2 7
W. compile or syntax error (PieceOfFurniture does not have an addDrawer method)
X. 2
Y. No for both, 1 point each

2. Comments. Array based list: Interesting question having to track the correct indices. Saw a lot more difficulties than we expected.

```
public GenericList<E> getEqualFronts(GenericList<E> other) {
      // first pass to determine needed space
      final int MIN_SIZE  = (size < other.size) ? size : other.size;
      int i = 0;
      while (i < MIN_SIZE && con[i].equals(other.con[i])) {
          i++;
      }
      GenericList<E> result = new GenericList<>();
      // i is number of matches
      result.con = (E[]) (new Object[i * 2 + 10]);

      // we know the first i elements match.
      for (int j = 0; j < i; j++) {
           result.con[result.size] = con[j];
           result.con[result.size + 1] = other.con[j];
           result.size += 2;
      }
      return result;
}
```

17 points, Criteria:

- Create result GenericList<E>. (Given constructor or by writing their own.) 1 point
- Create array for result with some extra capacity. Okay if 2 * min size + 10 (or other constant). Must create array of Objects and cast to E[] correctly. Can be done in constructor added in answer. 2 points
- Determine min list size between this and other. -1 if use Math.min. (can simply be part of loop check) 1 point
- loop correct, stop on first mismatch or reach limit of min size. 2 points
- access elements from this and other correctly. Plan in result correctly. Lose if this[i], other[i], or result[i] at any point. 2 points
- check elements equal correctly b calling equals method. Lose if ==. 2 points
- add 2 copies of equal elements to result array (or implement their own add method and resize if necessary) 2 points
- add at correct indices in resulting array. (can handle by implementing add method correctly.) 2 points
- Update size of result. 2 points
- return result. 1 point

Other deductions:

Worse than O(N) - 3. infinite loop possible, -4

Altering this or other -4. add more than elements at front that are equal -6

disallowed methods (especially add), -2 to -6 depending on severity (add method -6)

null pointer exception if not covered by other criteria, -4 public add with no check to resize, -2

3. Comments: Straight forward.

```
public ArrayList<String> getOneHitWonders(int requiredRank) {
    ArrayList<String> result = new ArrayList<>();
    for (int i = 0; i < records.size(); i++) {
        if (oneHitName(records.get(i), requiredRank)) {
            result.add(records.get(i).getName());
        }
    }
    return result;
}

private boolean oneHitName(NameRecord nameRecord, int requiredRank) {
    final int LIMIT = NUM_DECADES - 1;
    for (int i = 1; i < LIMIT; i++) {
        if (nameRecord.getRank(i - 1) == 0
                && nameRecord.getRank(i + 1) == 0) {
            int current = nameRecord.getRank(i);
            if (current != 0 && current < requiredRank) {
                return true;
            }
        }
    }
    return false;
}
```

17 points, Criteria:

- create resulting ArrayList correctly (Must be <String>) 1 point
- loop through records ArrayList correctly (for each loop okay) 2 points
- loop through NameRecord ranks with correct bounds 2 points
- access ranks from NameRecord correctly 1 point (lose if use [])
- correctly check CURRENT rank is better than required AND previous and next ranks are 0 (partial credit possible). In other words, correctly check if current decade meets the requirement for a one hit wonder name, 5 points (partial credit possible)
  -2 if don't guard against current rank also being 0 (3 points earned)
  -3 for OBOE (<= required instead of < on required rank)
- Stop checking current NamRecord once criteria for one hit wonder is met. 2 points
- correctly add name (String) that meets criteria to result. 1 point
  Adding same name multiple times -2.
- Always use ArrayList get and size methods correctly. Lose if names[i] or names.length. 2 points
- return resulting ArrayList 1 point

Others:

- infinite loop possible -4.
- Adding methods to NameRecord -4.
- calling contains method, -4
- adding same record multiple times, -4
- treat list as an array, -3
- equals method on ints, -2 (does not work, syntax error)

4. Comments: New data structures: The hardest question on the test. Always a challenge to understand a new data structure, especially if it has not appeared on any previous exams. The trick was to simply swap an element being removed with the last valid element. Had to be careful this didn't cause us to never examine that element. Another clever approach, was to just move the items to keep towards the front and then null out the elements from the new size to the old size. Also saw a lot of outer loops going through this list, which can lead to a logic error where elements are skipped due to shifting or swapping unless outer loop starts at back or adjusts loop control variable (GACK!).

```
public int removeAll(ArraySet<E> other) {
     int sizeAtStart = size;
      // for each element in other, check it exists in this
     for (int i = 0; i < other.size; i++) {
         E currentOther = other.con[i];
         boolean foundInThis = false;
         int indexThis = 0;
          while (indexThis < size && !foundInThis)) {
              if (con[indexThis].equals(currentOther) {
                  // found match
                  foundInThis = true; // so we stop
                  size--
                  // tricky part, we can just swap the last element with
                  // this one to be more time efficient
                    con[indexThis] = con[size];
                  // prevent memory leak in case the old last element is
                  // removed as well
                  con[size] = null;
              }
               indexThis++;
           }
    }
    return sizeAtStart - size;
}
```

16 points, Criteria:

- some way of tracking number of elements removed, 2 points
- loop through all of other elements correctly, (lose if other.con.length) 2 points
- Nested loop to look through all of calling object's elements. (lose if con.length) 2 points (lose if outer loop and doesn't guard against shift / skip error)
- call equals method on elements, (lose if ==), 2 points
- decrement size when match found (or update size correctly later) 2 points
- swap last element with element to be removed. Lose if shift. Order does not matter in the set, be efficient. Alternatively, move the items to keep towards the front and then null out the elements from the newsize to the old size. 1 point
- null unused elements of array, 2 points
- stop looking when match found for current element. It is a set, no duplicates. A given value from other can occur only once in this. 2 points
- return number of elements removed. 1 point

Other:

- Create new array -4
- Leave nulls scattered throughout array -4.
- get method on arrays, -2

Incorrect access of other.con[i] - 4.
Altering other in any way -4.
remove method assumed, -6