CS314 Fall 2023 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. 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. 4.5N + 4, (range: +/- 0.5 on N, +/- 1 on the constant value)
B. $4N^3 + 4N + 5$, +/- 1 on each coefficient, no credit if an N^2 term included.
C. 80 hours
D. 9 seconds
E. 0.21 seconds (Code is O(NlogN))
F. [5, 3, 2, 5, 2]
G. [U, V, J, S]
H. 20 seconds (code is $O(N^2)$)
I. O(N) (removing third from the end. Not shifting N elements, shifting 2 elements which is O(1))
J. 20 seconds (with the given resize, average case of add is now O(N). We are resizing every 5^{th} element.)
K. Compile error (Cannot instantiate an object of type Comparable, an interface type.)
L. The Book class does not implement a compareTo method.
M. --
N. 4
O. 3 (Cannot add a StringBuilder object to an ArrayList<String>.)
P. compile error (no 0 arg con.) compiles
Q. compile error compiles
R. 4 fly
S. 3 run
T. 8
U. Unknown until code is run. (Or words to that affect. Uses the inherited Object.toString() the output of which cannot be known until code is actually run.)
V. Flora
W. false slith
X. Compile error
Y. 5 100

2. Comments: Turned out to be the hardest of the 3 coding questions which surprised me. Many, many students did not notice or handle the possibility of elements in the lists being null.) Also many issues with placing elements in the resulting lists internal array at the correct spot.

```
public GenericList getMatchingElements(GenericList other) {
    int minSize = size < other.size ? size : other.size;
    GenericList result = new GenericList(minSize + 10);
    int rs = 0; // result size
    for (int i = 0; i < minSize; i++) {
        if (con[i] == null && other.con[i] == null) {
            result.con[rs] = con[i];
            rs++;
        } else if(con[i] != null && con[i].equals(other.con[i])) {
            result.con[rs] = con[i];
            rs++;
        }
    }
    result.size = rs;
    return result;
}
```

18 points, Criteria:

- Create array of Objects to store result. Local array or new GenericList, 2 points
- array has some extra capacity, 1 point
- loop through small of two lists correctly, 2 points (lose if loop through size of LARGER list)
- handle case when both elements are null, 2 points
- handle case when one or both elements are not null. 2 points (lost if NPE possible)
- Correctly calls equals on a non-null object, 2 points (lost if NPE possible)
- if match, add element at correct spot in resulting array, 2 points
- update result size correctly (Can be as we add or at the end) 2 points
- access elements from other list correctly. other.con[i] or a get method (without implementing) Lose if use other[i] at any point, 2 points
- return resulting GenericList 1 point

Other deductions:

- using disallowed methods unless implemented, varies: size() -2, get(index) -2, add(val) -6, resize() -4 Math.min() -1
- adding inappropriate public methods that violate encapsulation, -3 (size, get not necessary, but okay)
- Worse than O(N). O(N^2) or worse.
- NPE not covered by other criteria, -4
- AIBOE not covered by other criteria, -4
- calling get on an array, -3
- infinite loop possible, -3
- assumes result not to contain duplicates, -3
- add on native array, -2

3. Comments: Fairly straight-forward and students did well. Dealing with ArrayList of objects.

```
public ArrayList<String> getRarish(int max, int minRanks) {
    ArrayList<String> result = new ArrayList<>();
    for (NameRecord r : names) {
        if (isRarish(r, max, minRanks)) {
            result.add(r.getName());
        }
    }
    return result;
}

private boolean isRarish(NameRecord r, int cutoff, int minRanks) {
    int ranked = 0;
    for (int i = 0; i < r.numDecades(); i++) {
        int rank = r.getRank(i);
        if (rank != 0) {
            if (rank < cutoff) {
                return false;
            } else {
                ranked++;
            }
        }
    }
    return ranked >= minRanks;
}
```

18 points, Criteria:

- Create resulting ArrayList, 1 point
- loop through names ArrayList, (for each loop okay), 2 points
- for a given NameRecord loop through decades, 2 points
- access ranks of NameRecord correctly, 2 points
- Check ranks not < max (cutoff), 2 points
- if current rank less than cutoff stop immediately (break okay, boolean in while loop, helper method), 2 points
- count the number of decades the NameRecord is ranked (or not ranked), 2 points
- check criteria met (no rank below cutoff and ranked min number of decades, 2 points
- if current NameRecord meets criteria add name (String) to result, 2 points (lose if NameRecord)
- return result, 1 point

Other:

- Hard coded numbers for cutoff (max) or required number of decades ranked instead of parameters, -6
- treating names instance variables like an array, [] instead of get, -2
- Add same name multiple times, -3

4. Comments: A very interesting problem and most students did quite well. Interesting problem of dealing with the position which is 0 based and the run lengths which are counting from 1. Most solutions took this into account. Some confusion of size of list being represented and the number of ElementRun objects being stored. The question stated the array of ElementRun objects could have extra capacity.

```
    public void remove(int pos) {
        // First find the run the contains the element.
        int index = 0;
        int elementCount = con[0].runLength;
        // We know pos is inbounds so we won't hit a null element
        // or get an index out of bounds.
        while (pos > elementCount  - 1) {
            index++;
            elementCount += con[index].runLength;
        }
        // Decrement the count for this run and size
        // to remove one of the elements in the run.
        con[index].runLength--;
        size--;
        // Was the last element in the run? If so need to shift.
        if (con[index].runLength == 0) {
            numRuns--;
            for (int i = index; i < numPairs; i++) {
                con[i] = con[i + 1];
            }
            // Null out the last pair.
            con[numPairs] = null;
        }
    }
}
```

14 points, Criteria:

- some way to check number of elements seen or find the run that contains the element at position pos in the list, 2 points
- loop through the array of ElementRun objects, 3 points (correctly track current index in array with for loop control variable or separate variable) (Lose if loop trhough current one count 1 at a time instead of simply adding -> efficiency)
- correctly find the run that contains the element at the given position, 2 points
- when find the run with the element at the given position
  - decrement size of list, 1 point
  - decrement runLength of ElementRun object, 1 point
  - check if ElementRun must be removed (runLength now 0), 1 points
  - if so, decrement the numRuns instance variable, 1 point
  - if necessary to remove an ElementRun, shift elements down in array, 2 point
  - null out last ElementRun, 1 point

Other:

- Destroy list, -4 not stopping when found / remove more than 1, -3
- public methods added that violate encapsulation, -3
- Creating any new data structures including arrays -3 (question prohibited)

For questions P through Y, refer to the following classes. You may detach this page from the exam.

```
public class Animal {
    private int age;

    public Animal(int a) { age = a; }

    public void bday() { age++; }

    public int getAge() { return age; }

    public int bodyTemp() { return 100; }

    public String move() { return "oof"; }
}


public class Bird extends Animal {
    public Bird(int age) { super(age); }

    public String move() { return "fly"; }
}


public class Eagle extends Bird {
    public Eagle(int age) { super(age); }

    public int bodyTemp() { return 106; }

    public String toString() { return "Flora"; }
}


public class Emu extends Bird {
    public Emu() { super(3); }

    public String move() { return "run"; }
}


public class Snake extends Animal {
    public Snake() { super(5); }

    public int bodyTemp() { return 80; }

    public int bodyTemp(int x) { return bodyTemp() + x; }

    public String move() { return "slith"; }
}
```