CS314 Spring 2017 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.

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.

A. 18 seconds

B. O(N^3)

C. O(N^3) (Still)

D. 15 seconds (code is O(N))

E. 16 seconds

F. 84 seconds

G. [C, B, Z, Z, M, 6] (differences in spacing, brackets okay

H. {-5=A, 0=D, 5=A, 12=D} (differences in spacing, brackets okay)

I. 7 [3, 5, 4, 10, -5] (differences in spacing, brackets okay)

J. S: 100 50

K. S: 100 100

L. S: 200

M. 500 1000

N. Runtime Error OR Exception OR ClassCastException

O. Nothing. (The class is already abstract so we are not <u>required</u> to implement the methods of the interface.)

2. Comments. The simple generic list problem. Very similar to the quiz given Monday. In fact, easier.

```
public void removeFirstN(int n) {
    final int NUM_SHIFT = size - n;
    // shift elements down
    for (int i = 0; i < NUM_SHIFT; i++) {
        con[i] = con[i + n];
    }
    // null out empty spots;
    for (int i = NUM_SHIFT; i < size; i++) {
        con[i] = null;
    }
    size -= n;
}
```

15 points , Criteria:

- shift correct number of elements down to correct spots, 6 points
  - too many elements, -4
  - off by on error, - 4
  -
- null out unused spots to prevent memory leak, 6 points
  - too many spots, -4
  - off by one error -4
  -
- adjust size correctly, 3 points

Other deductions:

$O(N^2)$ -5

disallowed methods: size() - 2, remove() -10

-3 create new array with size # of elements, space efficiency and later time efficiency if no extra capacity
-3 null to con.length, time efficiency
-4 wrong type for temp array
-4 for out of bounds exceptions

3. Comments: A question involving multiple Generic Lists. This method is in the GenericList class so we have access to private fields of any GenericLists in scope such as other and result. Solution requires adding an element from calling list at most one time. Many solutions added a given element multiple times. For example given

[A, A, A].inOtherList([A,A, A]) should return [A, A, A] not [A, A, A, A, A, A, A, A, A] which many solutions did. This is incorrect for the method and takes more time than necessary.

```
public GenericList<E> inOtherList(GenericList<E> other) {
    GenericList<E> result = new GenericList<>(size + 10);
    for (int indexThis = 0; indexThis < size; indexThis++) {
        boolean found = false;
        int indexOther = 0;
        while (!found && indexOther < other.size) {
            found =
this.con[indexThis].equals(other.con[indexOther]);
            indexOther++;
        }
        if (found) {
            result.con[result.size] = this.con[indexThis];
            result.size++;
        }
    }
    return result;
}
```

20 points, Criteria:

- create result with enough space (or implement resize method) , 1 point
- loop for elements in this list, 3 points
- for each item checks of in other (or already in result, gack)
    - loop other elements, 3 points
    - stop when found (efficiency), 4 points
    - calls equals correctly, 3 points
- add to correct spot in result, 3 points
- increment resulting size correctly, 3 points

Other deductions:

- using add method without implementing, size - 3, get -3, contains -8, add -8
- O(N^3) -4
- accessing list variable as if it is an array, other[index] instead of other.container[index], -6
- destroys or damages either or both original lists, -8
- Add a single element from this list multiple times to result, -8
- Null Pointer Exception Possible, -4

4. Comments: A lot to keep track of. Many possible solutions. Best solutions added a helper method to determine if the name made a comeback. Requirement was for three 0's in a row.

Common problems:

- Not

```
public ArrayList<String> getComebackNames() {
    ArrayList<String> result = new ArrayList<>();
    for (NameRecord nr : nameRecordList) {
        if (comebackName(nr))
            result.add(nr.getName());
    }
    return result;
}

private boolean comebackName(NameRecord nr) {
    if (nr.getRank(0) == 0)
        return false;
    int limit = nr.numDecades() - 2;
    boolean threeZeros = false;
    int i = 1;
    while (!threeZeros && i < limit) {
        int r1 = nr.getRank(i);
        int r2 = nr.getRank(i + 1);
        int r3 = nr.getRank(i + 2);
        threeZeros = r1 == 0 && r2 == 0 && r3 == 0;
        i++;
    }
    // now must find non zero after third zero
    if (threeZeros) {
        for (int j = i; j < nr.numDecades(); j++) {
            if (nr.getRank(j) != 0)
                return true;
        }
    }
    return false;
}
```

25 points, Criteria:

- create result, 1 point
- loop through name records, 2 points
- determine if given name record is a comeback name
  - stop if first rank 0, efficiency, 3 points
  - determine if 3 zeros in a row correctly, 6 points
  - check after 3 zeros for a non-zero rank, 5 points
  - stop as soon as answer known for success, 2 points
- If comeback name, add to result, 2 points
- return result, 2 points
- correctly deal with ArrayLists and NameRecords, 2 points

Others:

Adding a given name multiple times -3, Adding NameRecord instead of name / String -2

5A. Comments: Students did fairly well on this problem. Biggest issues were not creating a HashMap. No need to use TreeMap as keys do not need to be in any order, so use HashMap for time efficiency. Also many answers created a new array for every element in schools. This is not necessary and creates far too many garbage objects. When only need to create a new array the first time when have seen a school / key.

Suggested Solution:

```
public Map<String, int[]> getStats (String[] schools, boolean[] results) {
    HashMap<String, int[]> m = new HashMap<>();
    for (int i = 0; i < schools.length; i++) {
        String school = schools[i];
        int[] temp = m.get(school);
        if (temp == null) {
           temp = new int[2];
           m.put(school, temp);
        }
       // temp either refers to old value, or new array
        temp[1]++; // number of applicants
        if (results[i]) {
            temp[0]++; // number of admits
        }
    }
    return m;
}
```

13 points, Criteria:

- create result and use HashMap for efficiency, 2 points
- loop through arrays, 1 point
- check if current school a key correctly, 2 points
- create new arrays for first time keys, 4 points
- update number of applications and admits correctly, 3 points
- put new arrays into map, 1 point

Other:

- not returning result, -1
- creating too many arrays, lots of garbage objects, -2

5B. Comments: Fairly simple problem. No need to use random. When we say break tie arbitrarily, we just mean pick any value among the tied schools. It can be the first one we see. Lots of problems with int division instead of floating point division.

Suggested Solution:

```
public String getMostSelectiveSchool(Map<String, int[]> map) {
    double minSelect = 1.1; // not possible to be > 1.0
    int maxApplicants = 0;
    String result = "";
    for (String key : map.keySet()) {
        int[] temp = map.get(key);
        double selectivity = 1.0 * temp[0] / temp[1];
        if (selectivity < minSelect
                || (selectivity == minSelect
                && temp[1] > maxApplicants)) {
            result = key;
            minSelect = selectivity;
            maxApplicants = temp[1];
        }
    }
    return result;
}
```

12 points, Criteria:

- variables to track min selectivity, result, and max applicants, 2 points
- loop through keys correctly, 2 points
- get value correctly, 2 points
- calculate selectivity correctly, 2 points (lose for integer division
- check if new min, 2 points
- handles tie break correctly for number of applicants, 2 points

Other:
Alters map, -4