CS314 Spring 2024 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. N + 5, (range: +/- 1.0 on each coefficient)
B. $3N^3 + 3N^2 + 6N + 5$, (range: +/- 1.0 on each coefficient)
C. $O(N^2)$
D. $O(N^3)$
E. 27 seconds
F. 10,100 seconds
G. 0.42 seconds (must be 0.42 exactly)
H. $O(N^2)$ (Removing the first half of the list.)
I. [1, 7, 5, 3]
J. 1 3
K. 10 seconds (Code is O(N) with given resize.)
L. ofalse
M. 6 (only)
N. 3
O. 5
P. compile error
compiles
Q. compiles
compile error
R. compile error (declared type of vr is Vehicle. No recharged method in the Vehicle or Object classes.)
S. 150
T. zoom: 5
U. 90
V. Yes (getClass() method inherited from the Object class.)
W. false false
X. Compile error (No 0 arg constructor in Vehicle class. Lose default constructor if any constructors declared.)
Y. Compile error. (Method in Vehicle class cannot access private instance variables of a Car object.)

2. Comments:

```
public int removeLast(E target) {
    // Start from the back of the list to find the last occurrence of
    // target in this list.
    for (int i = size - 1; i >= 0; i--) {
        if (con[i].equals(target)) {
            // Found it! Remove from list.
            remove(i);
            return i;
        }
    }
    // Never found target in this list.
    return -1;
}

public  remove(int pos) {
    // Remove the element at the given position from this list.
    E old = con[pos]; // Not necessary for exam 1 solution.
    size--;
    // shift element at given position forward one spot.
    for (int i = pos; i < size; i++) {
        con[i] = con[i + 1];
    }
    // Null out last element to help prevent memory leak.
    con[size] = null;
    return old; // Not necessary for exam 1 solution.
}
```

14 points, Criteria:

- loop that searches correct part of array, 1 point
- start from back for efficiency, 1 point
- correctly finds last occurrence of target, 2 points
- use equals to check if current element is equivalent to target, 2 points (lose if ==)
- if found, attempt to shift elements, 2 points
- if found, correctly shift elements, 2 points
- if found update size, 2 points
- if found, null out old last element, 1 point
- return correct value, 1 point

Other deductions:

- using disallowed methods unless implemented, varies: size() -1, get(index) -1, remove(index) -7
- adding inappropriate public methods that violate encapsulation, -3 (size, get not necessary, but okay)
- Worse than O(N). O(N^2) or worse. -4
- NPE not covered by other criteria, -2
- AIBOE not covered by other criteria, -2
- calling get on an array, -3
- create new array, -3
- removes more than 1 occurrence, -3

3. Comments:

```
public ArrayList<String> formerlyPopular(int minRank, int numUnranked) {
    ArrayList<String> result = new ArrayList<>();
    for (NameRecord nr : names) {
        if (meetsCriteria(nr, minRank, numUnranked) {
            result.add(nr.getName());
    }
    return result;
}

private boolean meetsCriteria(NameRecord nr, int minRank,
        int numUnranked) {
    final int NUM_DECADES = nr.numDecades();
    final int CUTOFF_DECADE = NUM_DECADES - numUnranked;

    // Check last numUnranked decades are all 0's.
    for (int = CUTOFF_DECADE; i < NUM_DECADES; i++)
        if (nr.getRank(i) != 0)
            return false;
    // Check at least one decade meets the requirement.
    for (int i = 0; i < CUTOFF_DECADE; i++) {
        int rank = nr.getRank(i);
        if (rank != 0 && rank <= minRank) {
            return true; // found one
        }
    }
    return false;
}
```

18 points, Criteria:

- Create resulting ArrayList, 1 point
- loop through names ArrayList, (for-each loop okay), 2 points
- attempt to verify the last numUnrankedDecades are 0 (unranked), 2 points
- correctly verifies the last numUnrankedDecades are 0 (unranked), 3 points
- stop when find a rank != 0 in numUnranked, 1 point
- attempt to check if name is ranked at or better than cutoff in allowed decades, 2 points
- correctly check if name is ranked at or better than cutoff in allowed decades, 3 points (lose if don't handle 0's correctly)
- stop when know name is ranked at or better than cutoff, 1 point
- 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 (min required) or required number of decades not ranked instead of parameters, -6
- treating names instance variables like an array, [] instead of get, -2
- Add same name multiple times, -3

4. Comments:

```
public boolean isSubset(MultiSet<E> other) {
    // Need to check if each element in other is in this.
    for (int i = 0; i < other.numDistinct; i++) {
        E currentElement = other.con[i].element;
        int indexThis = find(currentElement);
        if (indexThis == -1
            || con[indexThis].frequency < other.con[i].frequency) {

            // either this does not have element or not enough
            return false;
        }
    }
    return true;
}

// Find the index in this of tgt. Return -1 if not present.
private int find(E tgt) {
    for (int i = 0; i < numDistinct; i++)
       if (con[i].element.equals(tgt))
           return i;
    return-1;
}
```

18 points, Criteria:

- outer loop for elements of other, 2 points, must stop at numDistinct
- correctly access other.con, 2 points
- correctly access con[i].element in each MultiSet, 2 points
- attempt inner loop to see if current element from other in this, 1 point
- correct inner loop to find match in this, 1 point
- calls equals on objects correctly, 2 points (lose if ==)
- inner loops stops when found, 1 point
- if never found match for current element return false right away, 2 points
- if found match correctly check frequency in other <= frequency in this (or other frequency is >), 3 points
- if frequency > in other, return false right away, 1 point
- return true if subset, 1 point

Other:

- alter either multiset -4

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

```
public class Vehicle {
    private int topSpeed;

    public Vehicle(int s) { topSpeed = s; }

    public void enhance() { topSpeed += 5; }

    public int getTopSpeed() { return topSpeed; }
}


public class Motorcycle extends Vehicle {

    private String color;

    private Motorcycle(String c) {
        super(150);
        color = c;
    }

    public String getString() { return color + getTopSpeed(); }
}

public class Car extends Vehicle {

    private int doors;

    public Car() {
        super(90);
        doors = 4;
    }

    public void enhance() { doors++; }

    public String toString() { return "zoom: " + doors; }
}


public class ElectricCar extends Car {

    private int batteryLife;

    public void recharge() { batteryLife = 4; }

    public String toString() { return "days: " + batteryLife; }
}
```