CS314 Spring 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.
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. $4N^2 + 4N + 4$, +/- 1 on each coefficient

B. 7N + 6, +/- 1 on each coefficient, no credit if an N^2 term included.

C. 132 seconds

D. 8 seconds (method is O(N^2))

E. 4 seconds (method is O(N), inner loop is O(1) if Strings have a length of 5 or less.)

F. 12 seconds. (method is average case O(N^2) as half the time we add at the front of the list which is O(N) due to shifting all elements)

G. 40 seconds (method is O(N^2) due to the size of the String being created is proportional in length to the number of elements in the array.)

H. [12, 8, 4, 2]

I. A (uses least space, like our IntList but for chars. StringBuilder actually uses a byte[] as its internal con.s)

J. Runtime error (Exception acceptable. A NoSuchElementException occurs due to a logic error.)

K. [4, 2, 0, 7]

L. false true

M. {0=5, 3=0, 5=-2}

N. $O(N^2)$ (The iterator remove still requires elements to be shifted in the underlying array.)

O. $O(N^2)$

P. 0

Q. pink 16

R. len: 100

S. 25

T. 20

U. Compile or Syntax error (Declared type is Book, Book does not have a material method.)

V. false

W. 10

X. 30 16

Y. Runtime error (class cast exception)

2. Comments: Biggest issues were not realizing this was a constructor, not summing up counts to know how large an array we need (unless resize implemented as part of answer), and not adding any extra capacity to internal array. +10 elements was fine.

```
public GenericList(GenericList<E> org, int[] counts) {
    // First, how much space do we need?
    int newSize = 0;
    for (int count : counts) {
        newSize += count;
    }
    // Add some extra capacity
    con = (E[]) new Object[newSize + 10];
    // Could add check if size > 0 to avoid extra
    // work, but not required.
    // Could use org.size instead of counts.length
    for (int i = 0; i < counts.length; i++) {
        for (int j = 0; j < counts[i]; j++) {
            con[size] = org.con[i];
            size++;
        }
    }
    // size should now equal newSize
}
```

20 points, Criteria:

- determine how many elements shall be in the new GenericList, 4 points
- correctly create new array, array type must be Object and must cast to E[], 1point
- new array has some (at least one) extra capacity, 2 points
- loop through elements of original GenericList, 2 points counts.length okay! (lose if org.con.length)
- add the correct number of copies to the new GenericList, 3 points
- correctly access elements from org, org.con[index], NOT org[con], 2 points
- elements are added at the correct spot in the new GenericList (the end typically), 3 points
- set size of new GenericList correctly, (multiple ways to do this), 3 points

Other deductions:

- Create a new GenericList, -3 (this is a constructor)
- using disallowed methods, varies: size() -1, get(index) -2, add(val) -5, set(index, val) -2
- adding inappropriate public methods that violate encapsulation, -3 (size, get not necessary, but okay) -1 for a method that sums up counts (that would confuse a client)
- Worse than O(N) where N is org.size
- NPE not covered by other criteria, -4
- AIBOE not covered by other criteria, -4
- calling get on an array, -2
- return an int -1
- infinite loop possible, -3

3. Comments: This question had the largest number of different approaches. No need to guard against checking if a name is out of sync with itself as minDiff shall be >= 1. (First checked decade would show a name is not out sync with itself as the difference in ranks would be 0.)

```
public ArrayList<String> outOfSync(String name, int minDiff) {
    ArrayList<String> result = new ArrayList<>();
    NameRecord target = getRecord(name);
    for (NameRecord nr : names) {
        if (outOfSync(target, nr, minDiff)) {
            result.add(nr.getName());
        }
    }
    return result;
}

private boolean outOfSync(NameRecord target, NameRecord other,
        int minDiff) {
    for (int i = 0; i < target.numDecades(); i++) {
        int targetRank = target.getRank(i);
        targetRank = targetRank == 0 ? 1001 : targetRank;
        int otherRank = other.getRank(i);
        otherRank = otherRank == 0 ? 1001 : otherRank;
        int diff = targetRank - otherRank;
        if (Math.abs(diff) < minDiff) {
            return false;
        }
    }
    return true;
}
```

15 points, Criteria:

- Create resulting ArrayList, 1 point
- get NameRecord for target name, 1 point
- loop through Names.names, 2 points
- for a given NameRecord loop through decades, 1 point
- access ranks of NameRecord correctly, 1 point
- convert 0, unranked, to 1001 for both NameRecords, 2 points
- calculate difference between ranks, 1 point
- check difference is at least minDiff (< or <= okay) 1 point
- stop when answer known, first tim difference is < (or equal) minDiff, 2 points
- if current NameRecord meets criteria add name (String) to result, 2 points
- return result, 1 point

Other:

- O(N^2) or finding all names out of sync, -4
- Not getting NameRecord for given name, -3
- treating names like an array, [] instead of get, -2
- trying to increment ArrayList size, -2
- Logic error where name added multiple time, -3
- Hard coded difference instead of parameter, -3
- No outer loop or no inner loop, -6
- Adding too soon, before checking all diffs, -3
- Calling getRecord more than once or calling getRecord on names.get(i), -3

4. Comments: As the Entry objects are scattered no need to shift or swap if we find the key. Just null out that element in the array after retrieving the value in the Entry.

```
public V remove(K key) {
    for (int i = 0; i < con.length; i++) {
        if (con[i] != null && con[i].key.equals(key)) {
            V value = con[i].value;
            con[i] = null;
            size--;
            return value;
        }
    }
    return null;
}
```

15 points, Criteria:

- loop through all elements of con, 2 points (okay if track number of non-null and stop when it equals size)
- check element not null first, 2 points
- if not null, check if element equals key, 2 points
- when correct key found:
    - get value from Entry correctly, 2 points
    - null out old element, 2 points
    - decrement size, 2 points
    - return value as soon as key found, 1 point
- return null if never found key, 2 points

Other:

- infinite loop possible, -3
- shift elements (efficiency, -1)

Alternate solution that ensures we don't check unnecessary elements:

```
public V remove(K key) {
    int numNonNull = 0;
    while (numNonNull < size) {
        if (con[i] != null) {
            numNumNull++:
            if (con[i].key.equals(key)) {
                V value = con[i].value;
                con[i] = null;
                size--;
                return value;
            }
        }
    }
    return null;
}
```

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

```
public abstract class Book implements Comparable<Book> {
    private int pages;

    public Book() {}

    public Book(int p) { pages = p; }

    public String toString() { return "len: " + length(); }

    public int length() { return pages; }
}

public class PhysicalBook extends Book {
    public String color;

    public PhysicalBook() { color = "orange"; }

    public PhysicalBook(int pages, String c) {
        super(pages);
        color = c;
    }

    public int compareTo(Book b) { return length() - b.length(); }

    public String getColor() {return color; }
}

public class HardCover extends PhysicalBook {

    public HardCover(int pages, String color) { super(pages * 2, color); }

    public int getFont() { return 16; }
}

public class SoftCover extends PhysicalBook {

    private int saved;

    public SoftCover(int pages) {
        super(pages, "black");
        saved = pages / 2;
    }

    public int length() { return saved; }

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

    public String material() { return "paper"; }
}
```