CS314 Fall 2021 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.

A. $5N^2 + 4N + 4$, +/- 1 on each coefficient
B. 2N + 4, +/- 0.5 on the N term, +/- 1 on the constant
C. O(N)
D. $O(N^2)$
E. $O(N^2)$
F. O(N)
G. $O(N^3)$
H. [A, C, E, B, D] (quotes -2, minor differences in spacing and brackets okay)
I. 32 seconds
J. runtime error (out of bounds error)
K. 2 seconds
L. 44 seconds
M. compile error OR syntax error (cannot instantiate iterator)
N. JS C R (quotes -2, minor differences in spacing and brackets okay)
O. Runtime error (calls remove twice in a row)
P. 2 and 3 (1 point if just 2, 1 point if just 3)
Q. compile error OR syntax error (cannot add int or Integer to list of Strings)
R. $O(N^2)$
S. 1 and 2 (1 point each)
T. 2 only (1 point if 1 and 2)
U. V10
V. compile error OR syntax error (declared type Vacation, no days method in Vacation class or Object class)
W. OS
X. 2 only (-1 if 1 and 2)
Y. false false

2. Comments. This turned out to be the hardest question on the test. We saw lots and lots of different approaches. Given the default constructor the method either had to create an array of E's for the result with some extra capacity OR define a constructor that took in an initial capacity. **Recall, inside of a class your code has access to the private instance variables of any objects of that class. So, as this method is in the GenericList class we have access to the private instance variables of the parameter named other (a GenericList). Realize compareTo was meant to be used on the elements of the two lists to determine which one was larger. compareTo cannot be called on primitive ints or GenericLists as that class does not implement the Comparable interface.**

```
int newSize = size > other.size ? size : other.size;
E[] resultArray = (E[]) (new Object[newSize + 10]);
int minSize = size < other.size ? size : other.size;
for (int i = 0; i < minSize; i++) {
      if (con[i].compareTo(other.con[i]) > 0) {
            resultArray[i] = con[i];
      } else {
            resultArray[i] = other.con[i];
      }
}

E[] rest = size > other.size ? con : other.con;
for (int i = minSize; i < newSize; i++) {
      resultArray[i] = rest[i];
}
GenericList<E> result = new GenericList<>();
result.con = resultArray;
result.size = newSize;
return result;
```

18 points, Criteria:

- create resulting array once with some extra capacity. Lose if call constructor with capacity unless constructor part of answer. 2 points
- loop min size (can combine to one loop with if statement in size, going to max size) 3 points
- proper call and logic on copareTo method 2 points
- correct access of con[i] and other.con[i]. Lose if other[i] anywhere. 3 points
- add elements to correct location in resulting array, 2 points
- if one list longer, correctly add elements from longer list. Can combine into one loop with conditional. Must stop at max size. 3 points
- set resulting GenericList size and con array correctly, 1 point each, 2 points total
- return resulting GenericList, 1 point

Other deductions:
destroy or alter either list -4
disallowed methods (especially add), -2 to -6 depending on severity
null pointer exception if not covered by other criteria, -4
Worse than O(N) where N is the size of the largest list, -4

3. Comments: A fairly straightforward question. The biggest issue was potential logic errors due to removing an element and the other elements in the list being shifted forward one spot.

```
int orgSize = data.size();
// start from back to avoid shifting error
for (int i = data.size() - 1; i >= 0; i--) {
    if (shouldRemove(data.get(i), n) {
        data.remove(i);
    }
}
return orgSize - data.size();

private boolean shouldRemove(NameRecord nr, int n) {
    int start = NUM_DECADES - n;
    for (int i = start; i < NUM_DECADES; i++)
        if (nr.getRank(i) != 0)
            return false;
    return true;
}
```

18 points, Criteria:

- track original size or num removed, 1 point
- loop through data correct (okay here if front to back), 2 points
- correctly access elements from ArrayList with get, 1 point
- correctly call size method on ArrayList, 1 point
- inner loop or method to check if current NameRecord not ranked in last n decades, 1 point (attempt)
- inner loop or method to check if currentNameRecord not ranked last n decades is correct, 3 points
- stop computing (checking) current NameRecord when answer know, 2 points
- remove from ArrayList data if criteria met, 2 points
- correctly avoid the shift error when removing. (back to front, adjust loop control var, other approaches possible), 3 points
- return number of records removed, 1 point

Others:

- New data structures or arrays -4

4. Comments: Could I have given any more hints that the other data structure was going to be a question about Maps? The/'s were meant to be nulls. If students placed a "/" instead of null that was accepted. Several answers assumed that since the keys and values couldn't be null then there couldn't be any nulls in the internal storage container. That is not the case. Just like our GenericList, the extra capacity spots do store null. Those extra capacity spots **are not** keys or values. Also, as it was stated the client could not expect the keys to remain in order, then the fastest way to remove a key / value pair was to swap the last element, not shift elements down.

```
for (int i = 0; i < keys.length; i++) {
    Object key = keys[i];
    // look for this key
    int j = 0;
    while (j < size) {
        if (kvPairs[0][j].equals(key)) {
            // remove by swapping with last KV pair and null
            size--;
            kvPairs[0][j] = kvPairs[0][size];
            kvPairs[1][j] = kvPairs[1][size];
            kvPairs[0][size] = null;
            kvPairs[1][size] = null;
            j = size; // so we stop. break okay
        }
        j++; // could be in else
    }
}
```

14 points, Criteria:

- loop through array elements (for each okay) 2 points
- inner loop for keys (doesn't go past size), 2 points
- correctly check current array key equals kvPairs[0][index] key. must use .equals, 2 points
- decrement size if match found, 1 point
- remove the key value pair from kvPairs, 2 points
- array map: O(1) for removing key value pair (swap end). Lose if shift elements, 2 points
- null out old last key and value in kvPairs, 2 points
- stop inner loop if and when key found (keys in map are distinct, no repeats), 1 point

Other:

- creating any new data structures, including native arrays, -4
- disallowed methods, -3
- null pointer exception if not covered by other criteria, -3
- nulls left in middle of array, - 4
- shift / skip error possible if outer loop is for the kvPairs array, -2
- public methods that expose the internal details (especially positions) to the client