CS314 Fall 2018 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.
No points off for minor differences in spacing, capitalization, commas, and braces.

A. O(N^2)

B. O(N^2)

C. [J, V, J4, V]

D. 5 seconds

E. O(logN) (base 2 okay)

F. 16 seconds

G. O(N^2)

H. {A=12, C=5, G=10}

I. 9 seconds

J. 0

K. 40

L. 150 false

M. Runtime error OR Exception OR ClassCastException

N. 150false

O. Compile Error or Syntax Error (Can't instantiate instance of abstract class)

2. Comments. Meant to be an easy question involving the array base list we developed in class and covered on quizzes. Common problems were going to com.length instead of size and using == instead of the equals method.

```
public E mode() {
     int maxFrequency = 0;
     E result = null;
     for (int i = 0; i < size; i++) {
          E current = con[i];
          int currentFrequency = 1;
          for (int j = i + 1; j < size; j++) {
               if(current.equals(con[j]) {
                    currentFrequency++:
               }
          }
          if (currentFrequency > maxFrequency) {
               maxFrequency = currentFrequency;
               result = current;
          }
     }
     return result;
}
```

17 points, Criteria:

- variable to track max frequency, 1 point
- variable to track resulting value, 1 point
- outer loop that goes through elements of list, 4 points (-3 if use con.length, size - 1 okay)
- variable to track current frequency, 1 point
- loop from current position to end of active portion of array, 5 points
    - (-3 if start at 0, -2 if use con,length)
- check for equality between current val and inner loop val, 3 points (lose if == instead of equals)
- increment current frequency correctly, 1 point
- after determining current check if new mode found and update variables correctly, 3 points
- return result, 1 point (handles ties correctly)

Other deductions:

Not O(1) space, -4
inner loops starts at 0 instead of i + 1, efficiency, 1 point
modifies list, -5
all distinct, returns null logic error, -3

3. Comments: A question involving 2 GenericLists. Necessary to be clear on code working with calling object and code working with result. Common problem was not updating the size variable of the resulting list.

```
public GenericList<E> getRevCopyWithoutValue(E val) {
     // create result with extra capacity
     GenericList<E> result = new GenericList<>(size + 10);
     // start from back of this list so result in reverse order
     for (int i = size - 1; i >= 0; i--) {
          // check to ensure current doesn't equal val
          if (!con[i].equals(val) {
               // add it to result
               result.con[result.size] = con[i];
               result.size++;
          }
     }
     return result;
}
```

17 points, Criteria:

- create result correctly, 1 point
- some guaranteed extra capacity in result, 1 point
- loop from back of current elements
    - start at size - 1, not con.length - 1, 3 points
    - loop from back of array with correct bounds, 2 points
- check if current val, not equal to target
    - attempt, 1 point
    - uses equals and inverts with ! as opposed to !=, 2 points
- add new value to correct location in result, 3 points
- update size of result correctly, 3 points
- return correct result, 1 point

Other deductions:

- Not O(N), -5
- result[] instead of result.con[] -5
- modify calling object / list, -5
- modify calling list size variable, -3
- use add method without defining, -6
- make public add and not handling resize, -3

4. Comments: Jacob and Chris had zero fun grading this question. More than half of the attempted solutions were different than any other attempts. Most of the solutions that tried to count zeros and non-zeros did not work. The position of the zeros and non-zeros was necessary to know if a condition was violated so simply counting was not sufficient.

Common problems were not returning as soon as the answer was known, not checking that calling objects was square, not checking calling object was bigger than 1 x 1.

```
public boolean isUpperBidiagonal() {
     if (myCells.length == 1 || myCells.length != myCells[0].length)
          return false; // not square or possibly 1 by 1

     for (int r = 0; r < myCells.length; r++) {
          for (int c = 0; c < myCells[0].length; c++) {
               if (r == c || c == r + 1))
                    // cell is on main diagonal or upper diagonal
                    if (myCells[r][c] == 0)
                         return false;
               else if (myCells[r][c] != 0)
                    // non zero in a bad spot
                    return false;
          }
     }
     return true;
}
```

17 points, Criteria:

- return false if 1 by 1, 1 point
- return false if not square, 2 points
- nested loops to check all cells, with correct bounds, 3 points (possible to take off 1 or 2)
- main diagonal non-zero check, 2 points
- upper diagonal non-zero check 3 points
- other cells zero check, 2 points
- return false as soon as problem found, 3 point
- return true at end if no problems, 1 point

Others:

- worse than O(1) space, -4
- only checking cells on main diagonal and the above that, -6
- treating not square and not 1 x 1 as a precondition, -1

5. Comments: A lot of abstract going on. Had to deal with lots of different data structures. With the miswording on being in sync I took either solution, all differences greater than maxDiff or all ranks less than maxDiff. A name is not in sync with itself.

Common problems included calling getRecord on the ArrayList names instead of this, not converting 0's to 1001, adding multiple copies of a name to the result, not stopping when we know we exceed the given max allowed difference.

Suggested Solution:

```
public ArrayList<String> getNamesInSync(String name, maxDiff) {
        ArrayList<String> result = new ArrayList<>();
        NameRecord tgt = getRecord(name);
        final int NUM_DECS = tgt.numDecades();
        for (NameRecord other : names) {
            if (!other.getName().equals(name)) {
                boolean inSync = true;
                int dec = 0;
                while(inSync && dec < NUM_DECS) {
                    int tgtRank = tgt.getRank(dec);
                    if (tgtRank == 0)
                           tgtRank = 1001;
                    int otherRank = other.getRank(dec);
                    if (otherRank == 0)
                           otherRank = 1001;
                    int diff = Math.abs(tgtRank - otherRank);
                    inSync = diff <= maxDiff;
                    dec++;
                }
                if (inSync) {
                    result.add(other.getName());
                }
            }
        }
        return result;
    }
```

17 points, Criteria:

- get target NameRecord from this Names, 2 points (must do this just once)
- loop through all NameRecords, 1 points
- guard against adding self, 3 points
- loop through ranks correctly, 1 points
- stop looping through ranks as soon as outside bounds, 3 points
- correctly get ranks and check difference between them within bounds, 2 points
- change 0 ranks to 1001, 2 points
- correctly add other to result only if in snyc, 2 points
- return result, 1 point

Other:

- altering list of NameRecods -5
- treating the ArrayList names as an array, -4
- adding more than one instance of a given name, -5
- creating extra arrays, -2

6. Comments: Same sparse list as on past exams, but a different question. Lots of interesting ways to solve the problem even given the restrictions.

Suggested Solution:
Short question, points: By adding multiple references to the same default object we could introduce a logic error if that object is mutable and one reference is used to alter the object. (Or words to that affect)

```
public ArrayList<E> getExplicitLit() {
     ArrayList<E> result = new ArrayList<E>();
     int indexInValues = 0;
     for (int i = 0; i < sizeOfList; i++) {
          // do we store an explicit element?
          if (indexInValues < elementsStored
               && values[indexInValues].getPosition() == i) {

               result.add(values[indexInValues].getData();
               indexInValues++;
          } else {
               // element at index i is a default value
               result.add(defaultValue);
          }
     return result;
}
```

14 points, Criteria:

- create result correctly, 1 point
- loop through all correct positions of values, 4 points
- add default values correctly, 4 points
- add non default values correctly, 4 points
- return result, 1 point

Other:

- Worse than O(N), -5 (Any shifting of elements) (using the add at a given position method on non-default values after adding the correct number of default values leads to unnecessary shifting of elements in the resulting ArrayList.)
- AIOBE, -4
- missing default values at the end, -2
- not calling methods on ListElem objects correctly, 2 points
- trying to update size variables on ArrayList (it's private, -4