CS314 Spring 2016 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. 5N + 6, ± 1 on each coefficient and the constant

B. O(N)

C. $O(N^2)$

D. $O(N^3)$

E. O(N)

F. $O(N^2 \log N)$ // base 3 okay

G. $O(N^2)$

H. 20 seconds

I. 6000 items

J. 92 seconds

K. [J, C, K, X, K] // quotes = -1, differences in brackets, commas ok

L. [BA, 12, 1.5, []] // differences in brackets, commas ok

M. {A=3, G=9, M=-1, X=5}

N. faster

O. -5 0 [-5, 3, 4, 0] // differences in brackets, commas ok

P. runtime error or exception // just error is -1

Q. 1. valid, 2. invalid

R. 1. valid, 2. invalid

S. buzz 100

T. compile error or syntax error // just error is -1

U. tone

V. Phone: 25 OR compile error or syntax error. (Turns out correct answer was compile error as original class reference had LandLine and code referred to Landline.) either accepted

W. 100 2

X. runtime error or exception // just error is -1

Y. default default buzz chime

2. Comments. A fairly straight-forward problem. The only algorithmic difficulty was keeping the three different indices in this.container, other.container, and result.container separate and correct. A lot of students used method that were not allowed. The question said no other methods from GenericList could be used unless you implemented them yourself as a part of your answer.

Common problems:

- use methods not allowed by method such as add
- confusing other and other.container. For example other[index] instead of other.container[index]
- not updating size of result
- altering one of the two calling objects
- confusing indices in this, other and/or result

Suggested Solution:

```
public GenericList<E> getDualSublist(GenericList<E> other,
                                                int start, int stop) {
    final int NEW_SIZE = this.size + other.size();
    GenericList<E> result = new GenericList<E>(NEW_SIZE + 10);
    final int NUM_ELEMENTS = stop - start;
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        result.container[i] = this.container[start + i];
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        result.container[i + NUM_ELEMENTS] = other.container[start + i];
    }
    result.size = NUM_ELEMENTS * 2;
    return result;
}
```

20 points , Criteria:

- create resulting GenericList with adequate capacity. 3 points (okay if no **extra** capacity)
- add elements from calling list (this to result)
  - loop with correct bounds, 1 point
  - access correct elements from this, 3 points
  - place in correct indices in result, 2 points
- add elements from other list (other to result)
  - loop with correct bounds, 1 point
  - access correct elements from other, 2 points
  - place in correct indices in result, 3 points
- correctly set size of resulting list, 3 points
- return correct GenericList, 2 points

Usage errors:

any list[index] instead of list.container[index] -5

disallowed methods (unless implemented): get -3, add -5, size -3 , resize - 5

alter parameters: -5

3. Comments: Just a 2d array problem. Not a lot of encapsulation going on evening though we are writing an instance method for the MathMatrix class. The key was to realize there had to be exactly one non-zero value per row and one non-zero value per column to be possible strictly diagonal.

Common problems:

- just checking one non zero per column
- not checking exactly one non zero per row and column. A LOT of solutions would return true with a MathMatrix of all zeros because that simply checked the count was less than 0.
- not stopping when answer known

```
public boolean possibleDiagonal () {
      // square matrix so we can use same nested loop to traverse row and column.
      // Must have exactly one non zero value per row and per column to be
      // possible strictly diagonal.
      for (int i = 0; i < coeffs.length; i++) {
            int rowCount = 0;
            int colCount = 0;
            for (int j = 0; j < coeffs.length; j++) {
                  if (coeffs[i][j] != 0)
                        rowCount++;
                  if (coeffs[j][i] != 0)
                        colCount++;
            }
            if (rowCount != 1 || colCount != 1)
                  return false;
      }
      return true;
}
```

20 points, Criteria:

- access instance variables correctly: 1 point
- determine exactly one non-zero value per row: 7 points
- determine exactly non-zero value per row: 7 points
- stop early if answer known to be false, 4 points
- return correct answer, 1 point

Other deductions:

- Only checks less than 1 0 per row / column -5
- Only check columns -7
- $O(N^3)$ solution: -5

4. Comments: A good problem. Lots of abstraction and encapsulation going on.
Common problems:

- not resizing the array if necessary. By far this was the biggest mistake
- Not stopping as soon as correct pair found in container
- checking ALL elements of container (container.length) instead of the number of distinct items in the Bag

Suggested Solution:

```
public void add(Object value) {
    sizeOfBag++;
    boolean found = false;
    int index = 0;
    while (!found && index < distinctItemsInBag) {
        Pair currentPair = container[index];
        if (value.equals(currentPair.getObject())) {
            // found match
            int newFreq = 1 + currentPair.getFrequency();
            currentPair.setFrequency(newFreq);
            found = true;
        }
        index++;
    }

    if (!found) {
        // first occurrence of value in this Bag
        if (distinctItemsInBag == container.length) {
            resize();
        }
        container[distinctItemsInBag] = new Pair(value, 1);
         distinctItemsInBag++;
    }
}

private void resize() {
    Pair[] temp = new Pair[container.length * 2 + 1];
    for (int i = 0; i < container.length; i++) {
        temp[i] = container[i];
    }
    container = temp;
}
```

25 points, Criteria:

- increment size of bag, 2 points
- search for item already present
    - loop that only checks valid items, 4 points
    - check item present correctly, equals not ==, 3 points
    - stop when found, 3 points
    - increment frequency correctly if found, 2 points
- if not already present
    - check capacity and resize if necessary, 5 points (resize must be correct)
    - add new Pair with frequency of 1 at correct spot in array, 3 points
    - increment distinct items in bag, 3 points

Other: using methods not present: -5 per, efficiency T(N): -2,

For questions Q - Y, consider the following classes and interface:

```
public class Phone {
    private int cost;

    public Phone() {cost = 100;}

    public Phone(int c) {cost = c;}

    public int getCost() {return cost;}

    public String toString() {return "Phone: " + getCost();}

    public String sound() {return "default";}
}

public class Landline extends Phone {
                public String sound() {return "ring";}

                   public int getCost() {return 25;}
}

public interface AppStore {
                        public int numApps();
}

public class Smart extends Phone implements AppStore {
                public Smart(int cost) {super(cost);}

                 public int numApps() {return 1000;}
}


public class Android extends Smart {
                    public Android() {super(200);}

                public String sound() {return "tone";}
}


public class Apple extends Smart {
                     public Apple() {super(500);}

                public String sound() {return "chime";}

                   public int numApps() {return 500;}

                   public int getCost() {return 500;}
}


public class Feature extends Phone {
                public String sound() {return "buzz";}
}
```